Improving the Practicality of Model-Based Reinforcement Learning

An Investigation into Scaling up Model-Based Methods in Online Settings

Ronnie James Stafford

Supervisor: Prof. John Shawe-Taylor

Department of Computer Science
University College London

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements.

Ronnie James Stafford
January 2020

Abstract

This thesis is a response to the current scarcity of practical model-based control algorithms in the reinforcement learning (RL) framework. As of yet there is no consensus on how best to integrate *imperfect* transition models into RL whilst mitigating policy improvement instabilities in online settings. Current state-of-the-art policy learning algorithms that surpass human performance often rely on *model-free* approaches that enjoy unmitigated sampling of transition data. Model-based RL (MBRL) instead attempts to distil experience into transition models that allow agents to *plan* new policies without needing to return to the environment and sample more data.

The initial focus of this investigation is on kernel *conditional mean embeddings* (CMEs) (Song et al., 2009) deployed in an approximate policy iteration (API) algorithm (Grünewälder et al., 2012a). This existing MBRL algorithm boasts theoretically stable policy updates in continuous state and discrete action spaces. The Bellman operator's value function and (transition) conditional expectation are modelled and embedded respectively as functions in a reproducing kernel Hilbert space (RKHS). The resulting *finite-induced* approximate *pseudo*-MDP (Yao et al., 2014a) can be solved *exactly* in a *dynamic programming* algorithm with policy improvement suboptimality guarantees. However model construction and policy planning scale cubically and quadratically respectively with the training set size, rendering the CME impractical for sample-abundant tasks in online settings.

Three variants of CME API are investigated to strike a balance between stable policy updates and reduced computational complexity. The first variant models the value function and state-action representation explicitly in a parametric CME (PCME) algorithm with favourable computational complexity. However a soft *conservative* policy update technique is developed to mitigate policy learning *oscillations* in the planning process. The second variant returns to the non-parametric embedding and contributes (along with external work) to the compressed CME (CCME); a sparse and computationally more favourable CME. The final variant is a fully end-to-end differentiable embedding trained with stochastic gradient updates. The value function remains modelled in an RKHS such that backprop is driven by a non-parametric RKHS loss function. Actively compressed CME (ACCME) satisfies the pseudo-MDP contraction constraint using a sparse softmax activation function. The size of the

pseudo-MDP (i.e. the size of the embedding's last layer) is controlled by sparsifying the last layer weight matrix by extending the truncated gradient method (Langford et al., 2009) with group lasso updates in a novel 'use it or lose it' neuron pruning mechanism. Surprisingly this technique does not require extensive fine-tuning between control tasks.

Impact Statement

This research contributes to the ongoing academic effort to explore, understand and improve goal-based sequential decision-making optimisation algorithms. Humanity's quest to understand *intelligence* requires, to an extent, efforts both from the computer science and biological disciplines to converge. In particular this investigation improves the practicality of certain model-based reinforcement learning methods which belong to a set of algorithms thought integral to behavioural planning in the animal kingdom. Several avenues of future work are stated which will build upon this investigation and contribute to the overall research effort to understand intelligence.

Any major improvement to reinforcement learning and artificial intelligence in the research community has the potential to impact society in both positive and negative ways. Deploying robust behavioural-learning algorithms will free humanity from repetitive jobs in the labour force and provide efficient new services such as automated assistants and self-driving cars. But in the same breath, making large portions of the population unemployed will negatively impact society. The automation industrial revolution is beginning to materialise. Although the work in this thesis will not on its own lead to major disruptions in society, the entire body of artificial intelligence research going forward will. Therefore as a species we must carefully manage the introduction of this potentially disruptive technology to benefit society as a whole.

Acknowledgements

I would like to acknowledge my supervisor for being a deep source of knowledge, essential guide and collaborator throughout this process; my second supervisor Simon Julier for his support; Guy Lever who advised and collaborated with me on work related to the first three research chapters of this thesis; Csaba Szepesvári for contributions as described in Appendix C.1 and additional discussion. I would also like to acknowledge how writing a thesis can instigate a sparsification effect on close relationships.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

$\mathcal{X}$	Vector space
$\mathbf{x}$	Element $\mathbf{x} = [x_1, .., x_{\dim(\mathcal{X})}]^\top \in \mathcal{X}$
x_j	Member of $\mathbf{x}$
$\dim(\mathcal{X})$	Vector space dimensionality
$\lvert\mathcal{X}\rvert$	Vector space cardinality (element count)
X	Random variable whose instance is an $\mathbf{x}$
$\mathbf{X}$	Matrix of n vectors, $[\mathbf{x}_1, ..., \mathbf{x}_n]^\top \in \mathbb{R}^{n \times \dim(\mathcal{X})}$ where $\mathbf{x}_i \in \mathcal{X}$
$\mathcal{S}$, $\mathbf{s}$, $\mathbf{S}$, S	State nomenclature
$\mathbf{s}'$, S'	Successor state to an $\mathbf{s}$ or S, also $\mathbf{s}_{t+1}$ is a successor to $\mathbf{s}_t$
$\mathcal{A}$, $\mathbf{a}$, $\mathbf{A}$, A	Action nomenclature
$\mathcal{Z}$, $\mathbf{z}$, $\mathbf{Z}$, Z	State-action nomenclature where $\mathcal{Z} = \mathcal{S} \times \mathcal{A}$
r	Reward function $r : \mathcal{S} \times \mathcal{A} \to [0, r_{\max}]$, $r_{\max} \in \mathbb{R}^+$
P	Markov transition distribution $S' \sim P(\cdot \lvert S = \mathbf{s}, A = \mathbf{a})$
P_1	Start state distribution $S_{t=1} \sim P_1(\cdot)$
γ	Returns discount factor $\in [0, 1)$
$\mathcal{M}$	MDP tuple $\langle \mathcal{S}, \mathcal{A}, P, P_1, r, \gamma \rangle$
π	Stochastic policy $\mathbf{a} \sim \pi(\cdot \lvert \mathbf{s})$ or deterministic $\mathbf{a} = \pi(\mathbf{s})$
$v^\pi(\mathbf{s})$	Value function at $\mathbf{s} \in \mathcal{S}$, $v^\pi \in \mathbb{R}^{\mathcal{S}}$
$\mathbf{v}^\pi$	Vector of state-value function evaluations $[v^\pi(s_1), ..., v^\pi(s_{\lvert\mathcal{S}\rvert})]^\top$
$q^\pi(\mathbf{s}, \mathbf{a})$	Action-value function at $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$, $q^\pi \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$
$\mathcal{D}$	Dataset whose i^{th} element is $(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}')_i$
$\boldsymbol{\psi}$, $\boldsymbol{\Psi}$	State-action features $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\psi}}$, $\boldsymbol{\Psi} = [\boldsymbol{\psi}((\mathbf{s}, \mathbf{a})_1), ..., \boldsymbol{\psi}((\mathbf{s}, \mathbf{a})_n)]^\top$
$\boldsymbol{\phi}$, $\boldsymbol{\Phi}$	State features $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}_{\boldsymbol{\phi}}$, $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{s}_1), ..., \boldsymbol{\phi}(\mathbf{s}_n)]^\top$
$\mathcal{H}$	Hypothesis space or a Hilbert space
K	Kernel function $K : (\mathcal{S} \times \mathcal{A}) \times (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$

$\mathbf{K}$	Kernel matrix with element $K_{ij} = K((\mathbf{s}, \mathbf{a})_i, (\mathbf{s}, \mathbf{a})_j)$
$\mathcal{H}_K$	RKHS of scalar-valued functions $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, with kernel K
$K((\mathbf{s}, \mathbf{a}), \cdot)$	Implicit state-action feature map $\boldsymbol{\varphi}(\mathbf{s}, \mathbf{a}) \in \mathcal{H}_K$
L	Kernel function $L : \mathcal{S}\times\mathcal{S} \to \mathbb{R}$
$\mathbf{L}$	Kernel matrix with element $L_{ij} = L(\mathbf{s}_i, \mathbf{s}_j)$
$\mathcal{H}_L$	RKHS of scalar-valued functions, $\mathbb{R}^{\mathcal{S}}$ with kernel L
$L(\mathbf{s}, \cdot)$	Implicit state feature map $\boldsymbol{\phi}(\mathbf{s}) \in \mathcal{H}_L$

Acronyms / Abbreviations

MDP	Markov decision process
MBRL	Model-based reinforcement learning
RL	Reinforcement learning
DP	Dynamic programming
PI	Policy iteration
RKHS	Reproducing kernel Hilbert space
CME	Conditional mean embedding
PCME	Parametric conditional mean embedding
CCME	Compressed conditional mean embedding
ACCME	Actively compressed conditional mean embedding
LAM	Linear action model
FLAM	Factored linear action model
ERM	Empirical risk minimisation
PERM	Penalised empirical risk minimisation
OLS	Ordinary least squares

Chapter 1

Introduction

1.1 Foreward

Artificial intelligence (AI) is an ubiquitous term that now pervades the layman's vernacular and is usually accompanied with misunderstanding and hype about the imminent existential threat that the research community may unleash on humanity. In computer science it is an umbrella term used to describe a range of data-driven *machine learning* algorithms such as supervised, unsupervised and sequential decision-based learning. This investigation focusses on reinforcement learning (RL) (Sutton and Barto, 1998); a set of reward-based *value-prediction* and sequential decision optimisation *control* algorithms.

It is still uncertain if in the future we will be able to build super-intelligent machines. But as biological human computer scientists in the present, we are developing algorithms that can learn increasingly sophisticated behaviour that may eventually equal or surpass human abilities in domains that are as *general* as possible. There exists several niche domains where state-of-the-art sequential decision-making algorithms have surpassed the performance of their biological counterparts. These algorithms are either entirely based around a principled RL approach or have a subset of components that are RL-related. Thus they offer clues about which algorithmic components have been successful in unlocking super-human decision-making, but more importantly they offer clues about what components are *lacking* in the current offerings.

In the realm of competitive games, AlphaGo's 2015 victory (Silver et al., 2016) over European champion Fan Hui at the game of Go, shortly followed by world champion Lee Sedol's defeat in 2016 demonstrate the progress AI research has made. Such watershed moments in competitive play come after a long line of human-beating algorithms that stretch back to Tesauro (1994) in the game of Backgammon. Given also the publicised development of the deep Q-network (DQN) (Mnih et al., 2015) and its subsequent variants, achieving human-level performance playing a subset of Atari games in the

arcade learning environment (ALE) (Bellemare et al., 2015), the general public can be forgiven for thinking that the *rise of superintelligence* (Bostrom, 2014) is imminent. But contrary to popular belief, these algorithms remain confined to specialised domains. Although AI algorithm development does pose a risk that will instigate significant disruption to society as the automation revolution approaches, the 'AI apocalypse' remains firmly in the distant future. Both algorithms assume unfettered access to their environments such that transition data is cheap and abundant. DQN assumes unbridled access to the ALE and has no dynamics model. Alpha-Go is programmed with the deterministic rules of Go and therefore has both a simple and perfect model of its environment to optimise its policies with. This thesis focusses on a more real-world practical goal; learning imperfect transition models in unknown environments with non-linear stochastic dynamics and deploying them in policy optimisation. Intimately tied to this is the question of learning rich feature representations and this thesis develops three algorithms to explore how the choice of function approximation can affect the stability of policy learning.

1.1.1 Model-based vs model-free RL

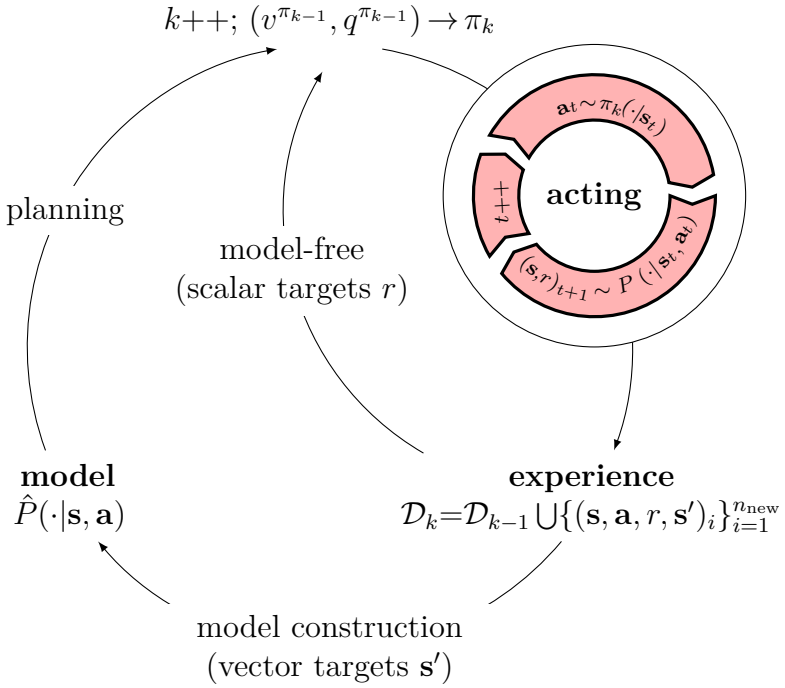
A formal introduction to RL is deferred until Chapter 2, however a high-level summary of value-based RL behavioural learning is described below. Referring to fig. 1.1, an *environment* is defined to be a Markov decision process (MDP) (Bellman, 1957) with states $\mathbf{s} \in \mathcal{S}$ and actions $\mathbf{a} \in \mathcal{A}$. At the k^{th} learning iteration an *agent's* behaviour is defined by its stationary *policy* π_k, such that at time step t with the agent in state $\mathbf{s}_t$, an action is drawn $\mathbf{a}_t \sim \pi_k(\cdot|\mathbf{s}_t)$ and the environment transitions by emitting a successor state $\mathbf{s}_{t+1} \in \mathcal{S}$ and immediate scalar reward $r_{t+1} \in \mathbb{R}$ from the MDP's hidden stationary distribution $(\mathbf{s}_{t+1}, r_{t+1}) \sim P(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. By repeating this process over many time steps, each transition $\{\mathbf{s}, \mathbf{a}, r, \mathbf{s}'\}$[1] is collected into a repository $\mathcal{D}_k$.

At each k, value-based RL aims to estimate either state-value functions $v^{\pi_k} : \mathcal{S} \to \mathbb{R}$ or action-value functions $q^{\pi_k} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, both of which estimate the utility of following π_k from initial states or state-actions respectively. Utility is defined as how well π_k can cumulatively collect *reward* $r \in \mathbb{R}$ under the hidden stationary distribution. How well an agent's policy can collect reward determines how well it is performing in the specified task that the MDP defines. Policy improvement or 'control' uses the updated value function to extract an improved policy for use in the next $(k{+}1)^{\text{th}}$ learning iteration. The aim is to find an *optimal policy* which no other policy can outperform in collecting reward.

Given data $\mathcal{D}_k$, model-free control updates value functions using scalar targets in data pairs $\{(\mathbf{s}, \mathbf{a}), r\}$ whereas model-based control updates value functions by first

[1] Shorthand notation used to represent $\{\mathbf{s}_t, \mathbf{a}_t, r_{t+1}, \mathbf{s}_{t+1}\}$ when dropping the time index.

Figure 1.1 Basic value-based RL Architecture (adapted from Sutton and Barto (1998)). At the k^{th} iteration, a policy is used to gather more experience such that it can be improved by either direct model-free updates or by planning using a transition model. New experience of n_{new} transitions is added to past experience with a total of n_k transitions available for training.



constructing a model $\hat{P}(\cdot|\mathbf{s}, \mathbf{a})$ using data pairs $\{(\mathbf{s}, \mathbf{a}), (r, \mathbf{s}')\}$. Atkeson and Santamaria (1997); Deisenroth and Rasmussen (2011) provide strong evidence that maintaining a model can reduce task sample complexity. Henaff et al. (2017) identify the reason for this sample efficiency as due to rich high-dimensional targets (e.g. in a supervised loss) used in model construction which extract more information from the transition data.

Sutton (1990) defines RL *planning* as the process by which an agent takes a *model* of the environment and outputs an improved policy π_{k+1}, a process which is known as model-based RL (MBRL). There are many approaches that define how the model is created and used in planning. As implied by the diagram, the joint distribution itself may be explicitly estimated such that samples can be drawn to estimate utility, but other approaches may only estimate this distribution implicitly and further examples will be summarised below.

Other RL flavours exist such as *actor-critic* algorithms (Peters and Schaal, 2007), (Szepesvári, 2010, sec. 4.4) or the related *direct policy search* (Peters and Schaal, 2008; Sutton et al., 1999a; Williams, 1992) where a policy is represented as a function in its own right whose range usually resides in a continuous action space. However this investigation focusses on value-based policy optimisation.

1.1.2 Neuropsychological motivation for MBRL

Informally from human experience we can validate for ourselves that a certain amount of our behavioural-planning occurs in our heads; thinking about certain tasks (such as navigating a metropolis, refining a tennis backhand or making a chess move) before executing a set of actions is commonplace. It is said that we possess models of the environment in our heads to plan future behaviour. Our internal models are also not precise reproductions of the complex dynamics of our real world, instead they contain enough information to plan adequate behaviour.

Pavlov (1927) first investigated canine behaviour made malleable by a *reinforcement* stimulus in temporal proximity to another *neutral* stimulus. He famously showed that dogs could be conditioned to salivate (a manifestation of behaviour) prompted by the ringing of a bell (as the neutral stimulus). If the bell was rung during 'training' in the temporal vicinity of food being presented, then bell-induced salivation was successfully conditioned to occur in 'post-training' regardless if food was presented or not. No behavioural conditioning was observed if during training the bell was rung outside the temporal vicinity of food presentation. Evidence for this *Pavlovian* learning has provided partial justification for computational *value-based* RL which conditions agent behaviour with reward signals. Related to this is the *temporal credit assignment* problem i.e. what actions taken in a sequence of decisions should be attributed to the reward collected. *Temporal difference* (TD) (Sutton, 1988) methods (a class of which DQN is a member) solve this problem by nudging value functions by an error - with a stronger nudge if a reward is unexpected. Schultz et al. (1997) claim tentative evidence that such model-free updates possess similarities to the observed outputs of dopaminergic neurons in the mammalian brain.

Pavlov's dogs did not actively *plan* to salivate upon the ringing of a bell. Rather they were passive participants to conditioning that became *instinctive* behaviour. Model-free RL updates are characterised by updating the the long-term estimate of expected reward with passive value function updates. There is no active planing to modify agent behaviour. It is widely accepted (Doll et al., 2012) that a more active model-based animal learning mechanism exists. Dayan and Berridge (2014) argue that in laboratory investigations, the passive model-free mechanism is not enough to explain all observed animal decision making phenomena. The detail of how model-based and model-free mechanisms coexist is still under debate. Dayan and Niv (2008) show that model-based mechanisms take a more active role in updating estimates of long term expected reward. Maintaining a forward model enables an agent to re-plan its behaviour at will, rendering it more adaptive and responsive to its environment, liberating it from being a passive laboratory learner.

1.1.3 Existing MBRL Approaches

There is as yet no consensus about how best a transition model should feature in MBRL algorithms. The Dyna architecture (Sutton, 1991) builds models from experience that can predict reward and successor states $\{r, \mathbf{s}'\}$ conditioned on current states and actions $\{\mathbf{s}, \mathbf{a}\}$. Either one-step or entire trajectories are imagined to form successive simulated scalar TD targets, such that value function updates consist of a mixture of real and simulated targets. Recent variants include DQN integration (Gu et al., 2016) and a model that explicitly separates transient and permanent memories (Silver et al., 2008). Yet this approach has not yet been inducted into the accepted DQN benchmark suite (Hessel et al., 2018) and value function updates still rely on model-free updates. Kaiser et al. (2019) demonstrate sample efficiency but ultimately final policies are not as good as model-free counterparts. A question that is not currently answered is that if a model exists that can be efficiently updated online and model-based planning is also computationally efficient, then do we need model-free updates at all?

Classical optimal control (OC) (Boyd and Vandenberghe, 2009, chapters 4,10) is another approach to model-based sequential decision-making. Both OC and RL operate in an environment seeking to optimise a cost function by finding an optimal policy. Characteristically however, in the convex optimisation literature classic OC assumes various structural assumptions where the transition model is usually provided a-priori. Classical optimal control techniques (e.g. Linear-Quadratic Regulation (LQR)) assume explicit access to exact transition models, ensuring strong policy optimisation guarantees (Kober et al., 2013, sec 1.2). However these assumptions break down when approximate models are used. This is in contrast with MBRL that estimates statistical models directly from noisy experience. However there have been attempts to fuse sample-based models with OC. Embed to control (Watter et al., 2015) learns to generate image trajectories from a latent space in which the dynamics are constrained to be locally linear, with an OC algorithm deployed for policy optimisation. Applying OC to learnt lower dimensional state representations is common e.g. Wahlström et al. (2015) learn a space within which model predictive control can be applied.

Kroemer and Peters (2011) present a non-parametric dynamic programming algorithm where kernel density estimates are used to model the system dynamics. In a related approach (Ormoneit and Sen, 2002) present a non-parametric method that represents continuous state MDPs on a discrete set of states over which the evaluation of the value function is maintained. Other non-parametric approaches have been proposed such as maintaining a GP for the transition dynamics such as Gaussian process dynamic programming (Deisenroth et al., 2009) and PILCO for direct policy search (Deisenroth and Rasmussen, 2011). However under certain non-Gaussian assumptions

GP methods require expensive numerical integration for inference (Rasmussen and Ghahramani, 2002).

Related to extracting lower dimensional state representations is the use of autoencoders as an unsupervised learning technique in the deep neural network literature (Gregor et al., 2014; Vincent et al., 2008), examples of which have been used to optimise end-to-end 'visuomotor' policies (Levine et al., 2016). State-temporal representation models (Ha and Schmidhuber, 2018) and imagination augmented policy learning (Weber et al., 2017) all require separate model, policy and memory modules. Common to these methods is the learning of a separate transition model and a policy module, used to fit good policies by trajectory optimisation using model-imagined roll-outs. This approach is popular in the deep-learning community and best summarised by Henaff et al. (2017).

1.2 Rationale

Policy performance guarantees are rarely forthcoming for model-based approaches that insist on separating transition model, policies and value functions into independent components. Therefore this investigation begins with the recently developed conditional mean embedding (CME) (Grünewälder et al., 2012a) approach whose transition model is an integral part of the value function. Consequently CME policy improvement guarantees are a function of *model accuracy*. The following briefly describes the focus on environment models that are *integral* components of value function estimates themselves as described by the Bellman operator (Bellman, 1957). This will be elaborated on in Chapter 2 but a brief introduction is required to fully describe the goals of this investigation.

1.2.1 Approximate Value Prediction

If $\mathcal{B}(\mathcal{S})$ is the set of real-valued functions $v : \mathcal{S} \to \mathbb{R}$, then the Bellman operator is a map $T^\pi : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ defined as

$$\begin{aligned}(T^\pi v)(\mathbf{s}) &= \mathbb{E}_{A\sim\pi(\cdot|\mathbf{s})}\Big[r(\mathbf{s}, A) + \gamma \mathbb{E}_{S'\sim P(\cdot|\mathbf{s},A)}\Big[v(S')\Big]\Big], \quad \forall \mathbf{s} \in \mathcal{S}, \\ &= r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \mathbb{E}_{S'\sim P(\cdot|\mathbf{s},\pi(\mathbf{s}))}\Big[v(S')\Big], \end{aligned} \tag{1.1}$$

where the successor state's conditional expectation is known as the *transition dynamics*. Assumptions throughout this thesis include a deterministic policy (second line) and known immediate average reward function, although a demonstration of learning the reward function will be presented. If the reward function and transition dynamics are known, both of which represent the *environment*, then T^π can be shown to be

a *contraction* (i.e. non-expansive) w.r.t the sup-norm (over $\mathcal{S}$) with a fixed point $v_* = v^\pi$. Exact *dynamic programming* (DP) exploits this property by iterating the application of T^π from an initial guess $v = v_0$ for guaranteed convergence to v^π. This value prediction method is used in exact policy iteration (Howard, 1960) for policy optimisation and is related to value iteration (Bellman, 1957).

If $\mathcal{S}$ is large or continuous, then maintaining value estimates over the entire state space quickly becomes intractable. As an example, although a Go board looks disarmingly discrete, the number of legal game states for a 19×19 board is $\sim 10^{170}$ (Tromp and Farnebäck, 2006), far greater than the number of atoms in the universe which is $\sim 10^{80}$ (usually estimated from density measurements of the universe such as in Ade et al. (2016)). Even a 'simple' MDP whose state space is defined on an interval in $\mathbb{R}$ has an infinite number of states, therefore maintaining value estimates for each state is impossible. The solution is to make a linear value function approximation

$$v^\pi(\mathbf{s}) \approx \hat{v}^\pi(\mathbf{s}) := \langle v^\pi, \boldsymbol{\phi}(\mathbf{s}) \rangle_{\mathcal{F}}\,, \tag{1.2}$$

where $\mathbf{s} \in \mathcal{S}$, $v^\pi, \boldsymbol{\phi}(\mathbf{s}) \in \mathcal{F}$ with state-feature representation $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}$ and the Hilbert space $\mathcal{F}$ is considered a *feature space*. Taking the conditional expectation of this linear form leads to

$$\mathbb{E}_{S' \sim P(\cdot|\mathbf{s},\mathbf{a})}\left[\hat{v}^\pi(S')\right] = \langle v^\pi, \mu_{S'|\mathbf{s},\mathbf{a}} \rangle_{\mathcal{F}}\,, \tag{1.3}$$

where the actual embedding $\mu_{S'|\mathbf{s},\mathbf{a}} := \mathbb{E}_{S' \sim P(\cdot|\mathbf{s},\mathbf{a})}\left[\boldsymbol{\phi}(S')\right]$ is the *expected next feature map*. If a parametric feature space is chosen $\mathcal{F} = \mathcal{F}_{\boldsymbol{\phi}}$, substituting the approximation scheme (1.2) and (1.3) into equation (1.1) leads to the following *approximate* relationship which is no longer a contraction for any given explicit $\boldsymbol{\phi}$ (Bertsekas, 2012) (see also section 2.5.2),

$$\langle \mathbf{w}^\pi, \boldsymbol{\phi}(\mathbf{s}) \rangle_{\mathcal{F}_{\boldsymbol{\phi}}} \approx r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \langle \mathbf{w}^\pi, \mu_{S'|\mathbf{s},\pi(\mathbf{s})} \rangle_{\mathcal{F}_{\boldsymbol{\phi}}} =: T^\pi \hat{v}^\pi, \quad \mathbf{s} \in \mathcal{S}. \tag{1.4}$$

Methods such as least squares temporal difference learning (LSTD) and Bellman residual minimisation (BRM) are required (see also section 2.5.2) to fit the value function i.e. find an appropriate approximate solution $\hat{\mathbf{w}}^\pi \in \mathcal{F}_{\boldsymbol{\phi}}$ to equation (1.4) using sampled transition data. Value function approximation coupled with policy improvement is known as approximate policy iteration (API).

Most API literature is based on model-free algorithms where $\mu_{S'|\mathbf{s},\mathbf{a}}$ is not explicitly estimated and instead sample successor states are used to fit $\hat{\mathbf{w}}^\pi$. API is notorious for being difficult and prone to policy instability, even in *linear* value function approximation schemes as described by definition 1.

Definition 1 (Deadly Triad (Sutton and Barto, 2018, Section 11.3))**.** *Three desirable algorithmic characteristics that when used together, conspire to induce instability in value-based RL;*

1. *Bootstrapping - Reusing existing approximate value function estimates for value function updates is efficient. However bootstrapped approximate value function update targets are both constantly changing and biased.*

2. *Function approximation - Required to generalise to large or continuous state spaces. However Bellman operators acting on fitted value functions may no longer be contractions.*

3. *Off-policy learning - Using all collected data to make value-function updates minimises data wastage. Collecting data from a behaviour policy in order to learn a different target policy aids exploration. However this may lead to instabilities due to the data distribution induced by the behaviour policy being different to the data distribution of the intended target policy.*

1.2.2 CMEs and Pseudo MDPs

The CME approach offers a way to avoid these instabilities and, counter-intuitively, does so by the introduction of an approximation to $\mu_{S'|\mathbf{s},\mathbf{a}}$. It is sufficient to learn the function $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$ by minimising the ideal loss

$$\hat{\mu} = \arg\min_{\mu \in \mathcal{H}} \Big[\mathbb{E}_{(S,A) \sim D, S' \sim P(\cdot|S,A)} ||\mu(S,A) - \boldsymbol{\phi}(S')||^2_{\mathcal{F}} \Big], \tag{1.5}$$

where D is a distribution over $\mathcal{S} \times \mathcal{A}$ and $\mathcal{H}$ is the hypothesis space in which to search for the embedding function. Importantly the CME chooses $\mathcal{F} = \mathcal{H}_L$ as a reproducing kernel Hilbert space (RKHS) (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004, p 63) (see section B.4.2 for a summary) of scalar-valued functions $v : \mathcal{S} \to \mathbb{R}$. State features $\boldsymbol{\phi}$ are implicit and are associated with kernel $L : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. The batch solution to the penalised version of equation (1.5) (see also section B.4.3) using data set $\mathcal{D}$ is the approximate embedding $\hat{\mu}_{S'|\mathbf{s},\mathbf{a}}$. This defines the approximate conditional expectation,

$$\begin{aligned} \hat{\mathcal{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] &:= \langle v, \hat{\mu}_{S'|\mathbf{s},\mathbf{a}} \rangle_{\mathcal{H}_L} = \langle v, \sum_{j=1}^{|\mathcal{S}'|} \alpha_j(\mathbf{s},\mathbf{a}) \boldsymbol{\phi}(\mathbf{s}'_j) \rangle_{\mathcal{H}_L}, \quad \mathbf{s}'_j \in \mathcal{S}' \\ &= \sum_{j=1}^{|\mathcal{S}'|} \alpha_j(\mathbf{s},\mathbf{a}) v(\mathbf{s}'_j), \end{aligned} \tag{1.6}$$

where $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n$ are the successor states in data set $\mathcal{D}$ and the conditional weights $\alpha_j(\mathbf{s},\mathbf{a}) \in \mathbb{R}$ are specified by the minimiser of equation (1.5). Most importantly equation

(1.6) is due to the *reproducing property* (Aronszajn, 1950) which means that the embedding approximation can be constructed as a *finite weighted sum* of value function estimates over training data $\mathcal{S}'$ without explicitly representing $\boldsymbol{\phi}$. By forming an *approximate* optimal Bellman operator and only considering value estimates over $\mathcal{S}'$,

$$(\hat{T}^* v)(\mathbf{s}') := \arg\sup_{\mathbf{a}\in\mathcal{A}} \Big[r(\mathbf{s}', \mathbf{a}) + \gamma \hat{\mathcal{E}}^{\mu}_{(\mathbf{s}',\mathbf{a})}[v] \Big], \quad \forall \mathbf{s}' \in \mathcal{S}', \tag{1.7}$$

then if $\hat{T}^*$ is a contraction, it has a fixed point $v_* = v^*_\mu$ which is the optimal value function for the embedding. Once v^*_μ is found then an action-value function is formed such that a greedy policy can be defined for policy optimisation.

The CME belongs to a larger class of MDP abstractions known as *finite-induced pseudo-MDPs* (Yao et al., 2014a). The premise is that the real MDP defined by the actual hidden expectation and reward function, which may be defined over large discrete (either finite or countably infinite) or even continuous state spaces $\mathcal{S}$, can be approximated by an approximate pseudo-MDP. The approximate MDP can be solved *exactly* using a DP algorithm so long as the constraint

$$||\boldsymbol{\alpha}(\mathbf{s}', \mathbf{a})||_1 \leq 1, \qquad \forall (\mathbf{s}', \mathbf{a}) \in \mathcal{S}' \times \mathcal{A},$$

is satisfied where $\boldsymbol{\alpha}(\cdot) := [\alpha_1(\cdot), ..., \alpha_{|\mathcal{S}'|}(\cdot)]^\top \in \mathbb{R}^{|\mathcal{S}'|}$. Algorithm computational complexities are decoupled from the size of the state space $|\mathcal{S}|$ and instead depend on training set size $|\mathcal{D}|$.

Unlike model-based control efforts popular in the deep learning community (Ha and Schmidhuber, 2018; Henaff et al., 2017), CME API is equipped with policy improvement sub-optimality guarantees *as a function of the underlying model accuracy* (Grünewälder et al., 2012a, theorem 3.2). The CME enjoys the strong theoretical framework of RKHS theory (Christmann and Steinwart, 2007), including consistency results (Grünewälder et al., 2012a, lemma 2.2) of the embedding such that in the limit of infinite data $|\mathcal{D}| \to \infty$, the trained pseudo-MDP converges to the real MDP.

CME API avoids the pitfalls of the 'deadly triad' of algorithm instability (definition 1). Constructing an approximate MDP and solving it exactly means that convergence to its value function is guaranteed. Biased sample value function updates (such as in TD algorithms) are avoided as full backups are available in the approximate MDP. Off-policy data is also used to build an approximate MDP and therefore doesn't affect value function convergence. It remains to be seen however if CME-API is stable if $\mathcal{D}$ is not given a priori and this question is the subject of the first part of this investigation.

CME policy iteration presented in Grünewälder et al. (2012a) has several shortcomings. Model construction is a *batch* regression problem with non-parametric function approximation. Computational cost for model construction scales cubically and planning scales quadratically with the training set size $|\mathcal{D}|$, rendering the algorithm useless

for large MDPs where data is abundant. In addition, the current CME assumes sufficient data a priori for learning good policies. It is not applied to online contexts as described in fig. 1.1 where the agent starts with nothing and frequently returns to the environment to gather more data.

Other kernel-based methods include probabilistic GP-regression for value-based model-free RL (Engel et al., 2005, 2003), model-based direct policy search (Deisenroth and Rasmussen, 2011) and model-based approximate RL (Deisenroth et al., 2009). The advantage with the CME method is that it has strong policy improvement guarantees, makes no distribution assumptions and requires no sampling during value prediction and policy improvement - samples are only used to estimate the CME transition model.

1.2.3 Investigation Summary and Contributions

Focus is made on improving the practicality of the non-parametric batch CME algorithm by developing techniques that reduce computational complexity of model construction and policy planning. To solve this problem requires an investigation in parametric and non-parametric linear value function approximation; how does the choice of $\mathcal{F}$ and $\mathcal{H}$ affect policy learning stability? Is it possible to introduce deep non-linear function approximators in order to learn rich data-driven representations without compromising policy learning stability? Is it possible to learn a transition model in online settings (fig. 1.1) without compromising policy learning stability?

Chapter 2 (Literature Review): Basic RL, known finite MDPs with dynamic programming and model-free methods in unknown finite MDPs are summarised. Function approximation (where penalised risk minimisation, regularisation, RKHS theory and neural networks are reviewed in appendix B.4) fused with conventional value-based RL is also reviewed. Focus is on the instability of approximate policy iteration (API) when function approximation, off-policy learning and bootstrapping are used, motivating the use of different approaches. Finally, existing benchmark model-based pseudo MDPs including CMEs are summarised as components of a policy iteration algorithm with emphasis on strong theoretical guarantees that mitigate instability. This motivates further development of CMEs and pseudo-MDP architectures.

Chapter 3 (Benchmark Algorithms and Initial Improvements) Are pseudo-MDP algorithms really that stable? Benchmark Pseudo-MDPs are implemented and compared with a focus on i) task performance and ii) timings for model construction and planning. CME improvements are made to i) improve model construction complexity with a fast matrix inverse and ii) use of a post hoc fast L1-projection (Duchi et al., 2008) of the embedding weights to satisfy the pseudo-MDP contraction constraint.

In addition, data is no longer assumed given a-priori such that an *explorative* policy iteration (explorative-PI) is developed in order to instigate exploration. Experimental evidence is provided for algorithm comparisons which also suggests that explorative-PI better explores the MDP. All Pseudo-MDP algorithms are shown to be stable but they suffer high computational costs in both planning and model construction, motivating us to seek modifications that make the algorithms more practical.

Chapter 4 (Parametric CME and Conservative Policy Updates) Is a parametric CME (PCME) policy iteration algorithm really that unstable? Both $\mathcal{H}$ and $\mathcal{F}$ are chosen as parametric in order to develop a PCME with the hope that computational complexity is reduced in model construction and planning. Both explicit state-action $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{F}_{\boldsymbol{\psi}}$ and state $\boldsymbol{\phi} : \mathcal{S} \rightarrow \mathcal{F}_{\boldsymbol{\phi}}$ feature representations are learnt using a modified kernel matching pursuit (Vincent and Bengio, 2002) algorithm that maintains compact representations in a data driven process. Unlike the non-parametric CME, the PCME algorithm requires least squares temporal difference (LSTD) (Bradtke and Barto, 1996) or Bellman residual minimisation (BRM) (Baird, 1995) to fit the value function. Both fitting methods are adapted to the model-based embedding setting and are compared empirically. Policy learning instability is demonstrated to exist as discussed in the literature review. Stability is restored using conservative greedy policy updates inspired by Kakade and Langford (2002). A deep PCME is proposed as future work.

Chapter 5 (Non-Parametric Sparsified CME) Given the instabilities induced by a parametric value function, we return to the non-parametric CME with the aim to sparsify the embedding for more computationally efficient model-construction and planning. Rewriting the embedding's penalised RKHS empirical loss function allows the derivation of a more efficient model construction process when combined with a sparse state-action representation and sparse successor state set $\mathcal{C} \subseteq \mathcal{S}'$. Group lasso was used to try and maintain $\mathcal{C}$ by sparsifying the embedding's weight matrix, however this ultimately failed. The final practical solution to this problem is provided externally from this thesis and is a lasso algorithm summarised in the appendix section C.1. The external work also includes a lasso component to maintain the post hoc pseudo-MDP contraction constraint. The final algorithm known as the compressed CME (CCME) with all sparsified components is implemented in explorative-PI and compared to its predecessors.

Chapter 6 (Differentiable CME) Dissatisfied with the computational complexity of batch model training, an actively compressed CME (ACCME) is developed as a fully end-to-end differentiable CME. Embedding weights $\boldsymbol{\alpha}(\cdot)$ are implemented as a neural network where each output neuron $\alpha_j(\cdot)$ has a one-to-one relationship with a member

of $\mathcal{C}$. Unlike the post hoc contraction constraint of the CME and CCME, ACCME maintains the constraint on $\boldsymbol{\alpha}(\cdot)$ *during* both training *and* inference by a sparse softmax activation function (Shazeer et al., 2017). The value function remains non-parametric in order to maintain a pseudo-MDP with contraction guarantees; model training is therefore driven by a non-parametric RKHS loss function. It was found that applying the CCME method for controlling $\mathcal{C}$ was insufficient for the hardest MDP. Instead groups of last layer weights $\mathbf{w}_j$ each leading to each j^{th} output neuron were degraded using a group lasso version of the truncated gradient update (Langford et al., 2009) during SGD. Zeroed groups of weights (and therefore corresponding output neurons) are removed along with members of $\mathcal{C}$. The novelty is that group lasso truncated gradient updates are only applied to *inactive* weight groups (induced by the sparse softmax function) for each backprop pass. *Frequently* inactive output neurons 'wither away' in a 'use-it-or-lose-it'-inspired neuron pruning process. Surprisingly ACCME's architecture and pruning mechanism works with little adjustment between control tasks.

Contributions

The work contained in this thesis is spread between two conference and two workshop publications.

- Lever, G., Stafford, R., and Shawe-Taylor, J. (2014). Learning Transition Dynamics in MDPs with Online Regression and Greedy Feature Selection. In *Autonomously Learning Robots Workshop (ALR NIPS Workshop)*, Chapters 4 and 5).
- Lever, G. and Stafford, R. (2015). Modelling Policies in MDPs in Reproducing Kernel Hilbert Space. In *AIStats*, pages 590–598, Chapters 4 and 5).
- Lever, G., Shawe-Taylor, J., Stafford, R., and Szepesvári, C. (2016). Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1779–1787, Phoenix, Arizona, Chapters 3 and 5).
- Stafford, R. and Shawe-Taylor, J. (2018). ACCME : Actively Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning. In *European Workshop on Reinforcement Learning (EWRL)*, Lille, France, Chapter 6).

Chapter 2

Literature Review

2.1 Reinforcement Learning

2.1.1 Overview

Bellman (1957) framed the discrete-time stochastic learning problem as solving a Markov Decision Process (MDP). RL algorithms rely on the assumption that an agent's environment is modelled as an MDP within which sequential decision optimisation is formalised.

Definition 2 (MDP)**.** *Given sets $\mathcal{S}$ and $\mathcal{A}$, known as the state and action spaces, stationary transition kernel P, start-state distribution P_1, immediate reward function $r : \mathcal{S} \times \mathcal{A} \to [0, r_{max}]$ and discount factor $\gamma \in [0, 1)$ then an MDP is defined as the 6-tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P, P_1, r, \gamma \rangle$.*

The *stationary* transition kernel P assigns a probability distribution $P(\cdot|\mathbf{s}, \mathbf{a})$ over $\mathcal{S}$ for each $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. If the state space is *discrete* (either finite or countably infinite) then the probability distribution assigns a transition probability $P(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \in [0, 1]$ for the transition $(\mathbf{s}, \mathbf{a}) \mapsto \mathbf{s}'$ i.e. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. If the state space is *continuous* (or uncountable e.g. when states are in $\mathbb{R}$) then $\mathcal{S}$ is assumed to be a measurable space (Ω, Σ) where for example $\Omega = \mathbb{R}$ and Σ is a collection of all open subsets (intervals) in $\mathbb{R}$ (Grimmett and Stirzaker, 2001, p 91). The probability distribution (or measure - see appendix B.3.1) assigns probabilities to the elements in Σ i.e. $P : \mathcal{S} \times \mathcal{A} \times \Sigma \to [0, 1]$ (Ferns et al., 2012). Solving a continuous MDP using classic policy iteration techniques that concern this thesis is intractable. The majority[1] of this thesis is concerned with approximate MDPs whose state and action sets are discrete. A formal description of continuous MDPs can also be found in Yao et al. (2014a).

[1] Continuous MDPs are briefly revisited during the review of pseudo-MDPs. Note that all simulated MDPs expressed in floating point arithmetic are in fact very large but finite MDPs.

An *agent* interacts with an MDP using a *stochastic stationary* control policy which assigns to each state $\mathbf{s} \in \mathcal{S}$ a probability distribution $\pi(\cdot|\mathbf{s})$ over $\mathcal{A}$. If the action space is finite then the probability of drawing action $\mathbf{a}$ in state $\mathbf{s}$ is $\pi(\mathbf{a}|\mathbf{s})$. Beginning with an initial state drawn from an initial state distribution $S_{t=1} \sim P_1(\cdot)$, an agent interacts with an MDP such that at time $t \in \mathbb{N}$ i) an action is drawn $A_t \sim \pi(\cdot|S_t)$, ii) the environment transitions to a successor state $S_{t+1} \sim P(\cdot|S_t, A_t)$ and iii) the MDP returns to the agent an immediate scalar reward signal $R_{t+1} = r(S_t, A_t)$. A sampled transition at time t is defined as the 4-tuple $\{\mathbf{s}, \mathbf{a}, r, \mathbf{s}'\}$ of the random variables $\{S_t, A_t, R_{t+1}, S_{t+1}\}$. An agent is said to act *continuously* until it reaches an absorbing state or *episodically* if it generates many trajectories each with H transitions, the latter being typical for the experimental settings explored in this investigation.

An MDP's stationary transition dynamics depend only on the current state and action i.e. $S_{t+1} \sim P(\cdot|S_i, A_i \ \forall i < t) : P(\cdot|S_t, A_t)$. This 'memoryless' characteristic is known as a *Markov property* which assumes all information used to define a transition is contained in the conditioning state-action pair. Similarly a stationary policy is defined independently of past states, extracting all required information from the current state when drawing new actions. Real world environments are frequently non-Markovian, however the cost of the Markov assumption is outweighed by the simplifications made to RL theory.

An agent's 'trajectory' (or 'rollout') is the realised values of the collection of random variables $\Xi_{t:H} := (S_t, A_t, S_{t+1}, A_{t+1}, ..., S_{t+H}, A_{t+H})$, where t is the start index (usually $t = 1$) and trajectory length $H \in \mathbb{N}$. Notation Ξ_t is used to define a trajectory of length $H = \infty$. Trajectories are drawn from a joint distribution[2] denoted by $\Xi_{t:H} \sim p(\cdot) := p(S_t, A_t, S_{t+1}, A_{t+1}, ..., S_{t+H}, A_{t+H})$. By fixing the policy π, an agent's behaviour is ranked by how much discounted cumulative reward it can collect in expectation under the joint trajectory distribution.

Definition 3 (Return)**.** *By following a stationary policy π, the random process of discounted return is $R^\gamma(\Xi_{t:H}) := \sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau} = \sum_{\tau=1}^{H} \gamma^{\tau-1} r(S_{t+\tau}, A_{t+\tau})$ then an agent's expected discounted cumulative reward is*

$$\begin{aligned} J(\pi) &:= \mathbb{E}_{\Xi_{t:H} \sim p(\cdot)} \Big[R^\gamma(\Xi_{t:H}) \Big], \\ &= \lim_{H \to \infty} \mathbb{E}_{\Xi_{t:H} \sim p(\cdot)} \left[\sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau} \right]. \end{aligned}$$

[2]Distribution notation is abused here only to distinguish what random variable belongs to each component distribution in the upcoming factorisation.

By factorising the joint distribution using the chain rule for conditional probabilities and removing conditional dependencies made redundant by a Markov property, then

$$\begin{aligned} p(\Xi_{t:H}) &= p(S_t, A_t, S_{t+1}, ..., S_{t+H}, A_{t+H}), \\ &= P_1(S_t)\pi(A_{t+H}|S_{t+H}) \prod_{\tau=1}^{H} \pi(A_{t+\tau-1}|S_{t+\tau-1})P(S_{t+\tau}|S_{t+\tau-1}, A_{t+\tau-1}), \\ \implies p(\Xi_{t:H}|S_t) &= \pi(A_{t+H}|S_{t+H}) \prod_{\tau=1}^{H} \pi(A_{t+\tau-1}|S_{t+\tau-1})P(S_{t+\tau}|S_{t+\tau-1}, A_{t+\tau-1}), \\ &= \pi(A_t|S_t)P(S_{t+1}|S_t, A_t)p(\Xi_{t+1:H}|S_{t+1}), \end{aligned} \tag{2.1}$$

where P_1 is the start-state distribution.

Definition 4 (State-value function (Sutton and Barto, 1998))**.** *By fixing π, a state-value function $v^\pi : \mathcal{S} \to \mathbb{R}$ is the expected cumulative discounted reward collected by an agent with respect to the trajectory distribution $p(\cdot|S_t)$ whose conditioning variable is fixed at $S_t = \mathbf{s}$ (i.e. start-state is deterministic),*

$$\begin{aligned} v^\pi(\mathbf{s}) &= \mathbb{E}_{\Xi_{t:H} \sim p(\cdot|S_t=\mathbf{s})}\Big[R^\gamma(\Xi_{t:H})\Big], \quad \mathbf{s} \in \mathcal{S}, \\ &= \lim_{H\to\infty} \mathbb{E}_{\Xi_{t:H} \sim p(\cdot|S_t=\mathbf{s})}\left[\sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau}\right]. \end{aligned} \tag{2.2}$$

If $\xi_{t:H} := \{S_t, S_{t+1}, ..., S_{t+H}\}$, then by fixing the stationary policy π and marginalising out $A_{t+\tau}$ (for all τ), then

$$\begin{aligned} p^\pi(\xi_{t:H}|S_t) &:= \sum_{\mathcal{A}} \pi(A_t|S_t)P(S_{t+1}|S_t, A_t)p^\pi(\xi_{t+1:H}|S_{t+1}), \\ &= P^\pi(S_{t+1}|S_t)p^\pi(\xi_{t+1:H}|S_{t+1}), \end{aligned}$$

effectively turning an MDP into a Markov reward process (MRP) $\langle \mathcal{S}, P^\pi, P_1, r, \gamma \rangle$ where P^π is the state-transition kernel. This thesis assumes *deterministic* stationary policies such that $A_t = \pi(S_t)$ and consequently an MDP's state-transition dynamics $\sum_{\mathcal{A}} \pi(A_t|S_t)P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, \pi(S_t)) = P^\pi(S_{t+1}|S_t)$.

Definition 5 (Optimal state-value function and optimal policy (Sutton and Barto, 1998))**.** *An optimal state-value function is the state-value function induced by the optimal policy, of which no other policy in the space of stationary policies Π can improve upon, such that*

$$v^*(\mathbf{s}) := \sup_{\pi \in \Pi} \Big[v^\pi(\mathbf{s})\Big], \quad \forall \mathbf{s} \in \mathcal{S}. \tag{2.3}$$

In simple domains whose MDPs have small finite state spaces, it is possible to visit each state such that this definition holds. However when $|\mathcal{S}|$ is large, infinitely countable or continuous then function approximation techniques are required to generalise over states. All recent RL algorithms deploy function approximators in varying degrees

within their algorithmic components such as in state-value function approximation, policy functions and transition models. A more detailed discussion on function approximation and its implications for RL algorithmic convergence and stability is deferred until later.

The *action-value function* is related to the state-value function and is critical in the control domain because policies can be extracted from it which will be described in detail later. Conditioning the trajectory distribution on the first state and action gives,

$$\begin{aligned} p(\Xi_{t:H}|S_t, A_t) &= P(S_{t+1}|S_t, A_t)\pi(A_{t+H}|S_{t+H}) \prod_{\tau=2}^{H} \pi(A_{t+\tau-1}|S_{t+\tau-1})P(S_{t+\tau}|S_{t+\tau-1}, A_{t+\tau-1}), \\ &= P(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1})p(\Xi_{t+1:H}|S_{t+1}, A_{t+1}), \end{aligned} \tag{2.4}$$

Definition 6 (Action-value function (Sutton and Barto, 1998))**.** *By fixing π, an action-value function $q^\pi : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ is the expected cumulative discounted reward, collected by an agent with respect to the trajectory distribution $p(\cdot|S_t, A_t)$ fixing conditioning variables on $S_t = \mathbf{s}$ and $A_t = \mathbf{a}$,*

$$\begin{aligned} q^\pi(\mathbf{s}, \mathbf{a}) &= \mathbb{E}_{\Xi_{t:H}\sim p(\cdot|S_t=\mathbf{s}, A_t=\mathbf{a})}\Big[R_t^\gamma(\Xi_{t:H})\Big], \quad (\mathbf{s}, \mathbf{a}) \in \mathcal{S}\times\mathcal{A}, \\ &= \lim_{H\to\infty} \mathbb{E}_{\Xi_{t:H}\sim p(\cdot|S_t=\mathbf{s}, A_t=\mathbf{a})}\left[\sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau}\right]. \end{aligned} \tag{2.5}$$

Definition 7 (Stationary Distribution (Sutton et al., 1999a))**.** *An ergodic Markov process (where states are visited an infinite number of times without any systematic state-visitation periodicity) has a unique limiting stationary distribution. By fixing a policy π, then an Ergodic MDP with state-transition distribution P^π has a unique stationary distribution D^π such that*

$$D^\pi(\mathbf{s}') = \sum_{\mathbf{s}\in\mathcal{S}} D^\pi(\mathbf{s}) P^\pi(\mathbf{s}', \mathbf{s}).$$

The stationary distribution $D^\pi(\mathbf{s})$ can be interpreted as the fraction of time spent in $\mathbf{s}' \in \mathcal{S}$ under the policy π (Sutton et al., 1999a). The joint distribution for future state-actions is $D^\pi(S, A) := \pi(A|S)D^\pi(S)$.

2.1.2 Bellman Equations

The *Bellman equations* (Bellman, 1957) describe the recursive nature of value functions and are the progenitors of all value-based RL. By unravelling the summation of the cumulative discounted reward (which is a random variable) by one step, $R^\gamma(\Xi_{t:H}) = R_{t+1} + \gamma R^\gamma(\Xi_{t+1:H})$, then the value function expectation decomposed[3] by

[3]Notation is simplified such that the conditional distributions under which expectations are made should be obvious from context.

equation (2.1) is

$$\begin{aligned} v^{\pi}(\mathbf{s}) &= \lim_{H\to\infty} \mathbb{E}_{\Xi_{t:H}|S_t=\mathbf{s}}\Big[R_{t+1} + \gamma \sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau+1}\Big], && \mathbf{s}\in\mathcal{S}, \\ &= \mathbb{E}_{A_t,S_{t+1}|S_t=\mathbf{s}}\Big[R_{t+1} + \gamma \lim_{H\to\infty} \mathbb{E}_{\Xi_{t+1:H}|S_{t+1}}[R^{\gamma}(\Xi_{t+1:H})]\Big], \\ &= \mathbb{E}_{A_t|S_t=\mathbf{s}}\Big[r(\mathbf{s},A_t) + \gamma \mathbb{E}_{S_{t+1}|S_t=\mathbf{s},A_t}[v^{\pi}(S_{t+1})]\Big], \\ &= r(\mathbf{s},\pi(\mathbf{s})) + \gamma \mathbb{E}_{S_{t+1}|\mathbf{s},\pi(\mathbf{s})}[v^{\pi}(S_{t+1})], \end{aligned} \tag{2.6}$$

where in the last line $\pi : \mathcal{S} \to \mathcal{A}$ is a deterministic function. A similar unravelling of the action-value function using equation (2.4) gives

$$\begin{aligned} q^{\pi}(\mathbf{s},\mathbf{a}) &= \lim_{H\to\infty} \mathbb{E}_{\Xi_{t:H}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[R_{t+1} + \gamma \sum_{\tau=1}^{H} \gamma^{\tau-1} R_{t+\tau+1}\Big], && (\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}, \\ &= \mathbb{E}_{S_{t+1},A_{t+1}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[R_{t+1} + \gamma \lim_{H\to\infty} \mathbb{E}_{\Xi_{t+1:H}|S_{t+1},A_{t+1}}[R^{\gamma}(\Xi_{t+1:H})]\Big], \\ &= \mathbb{E}_{S_{t+1},A_{t+1}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[R_{t+1} + \gamma q^{\pi}(S_{t+1},A_{t+1})\Big], \\ &= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{A_{t+1}|S_t=\mathbf{s}}\Big[\mathbb{E}_{S_{t+1}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[q^{\pi}(S_{t+1},A_{t+1})\Big]\Big], \\ &= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{S_{t+1}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[q^{\pi}(S_{t+1},\pi(S_{t+1}))\Big], \\ &= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{S_{t+1}|S_t=\mathbf{s},A_t=\mathbf{a}}\Big[v^{\pi}(S_{t+1})\Big]. \end{aligned} \tag{2.7}$$

where in the fifth line the policy is assumed deterministic. An MDP can be re-defined for a stochastic reward signal such that it is drawn from the distribution $(S_{t+1}, R^{\text{stoch}}_{t+1}) \sim P(\cdot|S_t, A_t)$ which features in the trajectory distribution $p(\Xi_{t:H})$. The deterministic reward function that is used throughout this thesis is therefore equivalent to the average immediate stochastic reward $r(S_t, A_t) := \mathbb{E}_{R^{\text{stoch}}_{t+1}|S_t,A_t}[R^{\text{stoch}}_{t+1}]$ (Szepesvári, 2010, p 8).

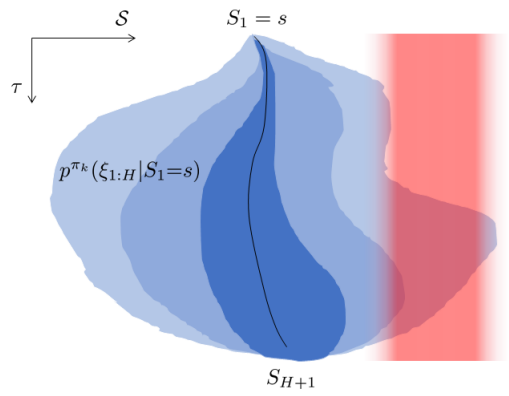
There are four sources of stochasticity for episodic learning in *fully observable* MDPs; initial start state, policy, reward and transition dynamics distributions. We initially restrict stochasticity to the start state distribution and transition dynamics by assuming a deterministic immediate reward function and policy function. Learning the immediate reward function is deferred until the final stages of this investigation. The discount factor γ controls how important future returns are and typically $\gamma = 0.99$. Extensions to MDPs such as *partially observable Markov decision processes* (POMDPs) (Monahan, 1982) where the agent only has a partial view of the environment's state vector for any one state sample are not considered. POMDPs along with non-stationary dynamics/policies are viewed as possible future extensions.

2.1.3 The Control Problem

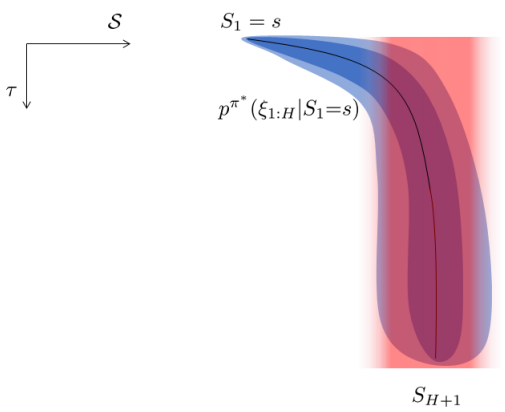
Value-prediction refers to estimating the state-value function for a particular policy. RL *control* is an umbrella term for algorithms that find optimal policies, usually incorporating value-prediction as a subcomponent. One way of visualising the control problem is in fig. 2.1.

Figure 2.1 Visual representation of distribution of states (blue) generated by p^π for a 1-D state space. Both a suboptimal and optimal policy for collecting immediate reward (red) are shown.

(a) Sub-optimal policy from some k^{th} iteration of a policy learning algorithm



(b) Optimal policy



2.2 Theory of Dynamic Programming in Known MDPs

The following summarises the theory of Dynamic programming (DP) algorithms. Basic value prediction and control algorithms in an MDP are summarised below on the assumption that immediate reward function $r(\mathbf{s}, \mathbf{a})$ and the transition dynamics $P(\cdot|\mathbf{s}, \mathbf{a})$ are known. DP for value prediction relies on iterating the Bellman operator T^π (based on the Bellman equation) that creates a converging sequence of value functions. Similarly DP for control requires iterating what is known as the *optimal* Bellman operator T^* until convergence to the optimal value function. In both cases contraction properties of these operators must be adhered to and therefore it is important to leverage some functional analysis machinery to fully describe both DP algorithms.

2.2.1 Space of Value Functions

Basic functional analysis machinery is used to show *contraction* guarantees w.r.t the sup-norm for i) the *Bellman operator* whose fixed point is the value function and ii) the *Bellman optimality operator* whose fixed point is the optimal value function.

Definition 8 (Space of uniformly bounded functions)**.** *A Banach space (see definition B.8) of uniformly bounded functions is defined as* $\mathcal{B}(\mathcal{S}) := \{v : \mathcal{S} \to \mathbb{R}, ||v||_\infty < +\infty\}$, *where* $||v||_\infty := \sup_{\mathbf{s}\in\mathcal{S}} \left[|v(\mathbf{s})|\right]$.

This definition uses function notation to describe the space of value functions $\mathbb{R}^{\mathcal{S}}$. However a Banach space is a normed vector space $(\mathcal{V}, ||\cdot||)$ and therefore equally valid notation for an element is $\mathbf{v} := [v(\mathbf{s}_1), ..., v(\mathbf{s}_n)]^\top \in \mathcal{B}(\mathcal{S})$. It is therefore easy to see that i) the dimensionality of $\mathcal{B}(\mathcal{S})$ is equal to the number of states $n = |\mathcal{S}|$ and ii) norms act along the length of the vector $\mathbf{v}$. Vector notation will be used to visualise function approximation in the latter sections. The bounded norm is made possible by setting the absolute of the maximum immediate reward to $r_{max} \geq 0$. If $\gamma \in [0, 1)$ then the return objective (definition 3) is bounded $|J(\pi)| \leq \frac{1}{1-\gamma} r_{\max}$ and therefore the value function norm is bounded such that $||v||_\infty = \sup_{\mathbf{s}\in\mathcal{S}} |v^\pi(\mathbf{s})| \leq \frac{1}{1-\gamma} r_{\max}$.

The following results are integral to the dynamic programming algorithms used for value prediction/control.

Definition 9 (Contraction)**.** *Given a Banach space* $\mathcal{B}(\mathcal{S})$ *of uniformly bounded functions then a mapping* $T : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ *is Lipschitz with constant* $L \geq 0$ *if*

$$||Tu - Tv||_\infty \leq L||u - v||_\infty, \quad u, v \in \mathcal{B}(\mathcal{S}). \tag{2.8}$$

In addition if $L<1$ *then* T *is a sup-norm contraction such that the distance between functions* u *and* v *w.r.t* $||\cdot||_\infty$ *is reduced.*

Theorem 1 (Banach fixed point theorem (Kreyszig, 1978, Thm 5.1-2, p.300))**.** *Given a Banach space of uniformly bounded functions* $\mathcal{B}(\mathcal{S})$, *let* $T : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ *be a Lipschitz contraction mapping, then* T *has precisely one fixed point* $v_* \in \mathcal{B}(\mathcal{S})$ *such that* $Tv_* = v_*$.

Proof. Szepesvári (2010, p.77) shows that the Cauchy sequence (c.f. equation (B.8)) $(v_n)_{n\geq 0}$ converges after applying $v_{n+1} = Tv_n$ an infinite number of times. More precisely it is shown that $\lim_{j\to\infty} \sup_{\tau\geq 0} ||v_{j+\tau} - v_j||_\infty = 0$.

To show that this limit is the fixed point v_* of T, iteratively apply T and use definition 9,

$$\begin{aligned}
||v_j - v_*||_\infty &= ||Tv_{j-1} - Tv_*||_\infty, \\
&\leq L||v_{j-1} - v_*||_\infty, \\
&= L||Tv_{j-2} - Tv_*||_\infty \\
&\leq L^2||v_{j-2} - v_*||_\infty, \\
&\cdots, \\
&\leq L^j||v_0 - v_*||_\infty, \\
\implies \lim_{j\to\infty} ||v_{j+1} - v_*||_\infty &= 0 \text{ as } \lim_{j\to\infty} L^j = 0.
\end{aligned}$$

□

2.2.2 Bellman Operators

Convergence of the Bellman operators is summarised below, noting that the stationary policy is deterministic.

Bellman Operator & Policy Evaluation

Definition 10 (State-value Bellman operator)**.** *For an arbitrary state-value function $v \in \mathcal{B}(\mathcal{S})$, the Bellman operator $T^\pi : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ induced by a deterministic policy π is defined as the application of the one step Bellman equation (2.6)*

$$(T^\pi v)(\cdot) = r(\cdot, \pi(\cdot)) + \gamma \mathbb{E}_{S_{t+1}|\cdot,\pi(\cdot)}\Big[v(S_{t+1})\Big]. \tag{2.9}$$

The notation $(T^\pi v)(\cdot)$ designates precedence of operator T^π over a value function's point evaluation. If $\gamma \in [0, 1)$ then T^π is a sup-norm contraction mapping (as proven in lemma B.1; appendix B.1) and therefore it has a fixed point v_*, whose rate of convergence is independent of $\mathbf{s}$ (Szepesvári, 2010, p.78). By construction the value function is $v^\pi = T^\pi v^\pi$ (see Bellman equation (2.6)) and therefore the fixed point must be the value function $v_* = v^\pi$. This forms the basis of a value-prediction algorithm known as *policy evaluation* and whose practical implementation is summarised in the next section.

The action-value Bellman operator, although not directly utilised in this thesis, does serve the broad discussion on the derivation of model-free RL algorithms and is included in below.

Definition 11 (Action-value Bellman operator)**.** *For an arbitrary action-value function $q \in \mathcal{B}(\mathcal{S}\times\mathcal{A})$, the action-value Bellman operator $T^\pi : \mathcal{B}(\mathcal{S}\times\mathcal{A}) \to \mathcal{B}(\mathcal{S}\times\mathcal{A})$ is defined as*

$$(T^\pi q)(\cdot,\cdot) = r(\cdot,\cdot) + \gamma \mathbb{E}_{S_{t+1}|\cdot,\cdot}\Big[q(S_{t+1}, \pi(S_{t+1}))\Big], \tag{2.10}$$

and by the same contraction arguments this operator has a fixed point $q_* = q^\pi$.

Bellman Optimality Operator & Policy Improvement

Recall equation (2.3), then an optimal policy π^* is one that generates a state-value function that cannot be exceeded anywhere in $\mathcal{S}$ by any other policy.

Definition 12 (State-value Bellman optimality operator)**.** *For an arbitrary* $v \in \mathcal{B}(\mathcal{S})$ *the Bellman optimality operator* $T^* : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ *is defined as*

$$\begin{aligned}(T^*v)(\cdot) :=, \sup_{\mathbf{a}\in\mathcal{A}} \Big[r(\cdot,\mathbf{a}) + \gamma \mathbb{E}_{S'|\cdot,\mathbf{a}}[v(S')]\Big],\\ = \sup_{\mathbf{a}\in\mathcal{A}} \Big[q(\cdot,\mathbf{a})\Big],\end{aligned} \tag{2.11}$$

where in the second line $q(\cdot,\mathbf{a}) = r(\cdot,\mathbf{a}) + \gamma \mathbb{E}_{S'|\cdot,\mathbf{a}}[v(S')]$. T^* is indeed a sup-norm contraction as proven in lemma B.3; appendix B.1 and therefore it has a unique fixed point v_*. But what is the fixed point? The answer requires the *policy improvement theorem* and the definition of a *greedy* policy.

Theorem 2 (Policy improvement theorem, adapted from Sutton and Barto (1998, Ch4, sec. 4.2))**.** *For two arbitrary stationary and deterministic policies* $\hat{\pi}$ *and* π'*, their two Bellman operators* $T^{\hat{\pi}}$ *and* $T^{\pi'}$ *whose fixed points are* $v^{\hat{\pi}}, v^{\pi'} \in \mathcal{B}(\mathcal{S})$ *respectively, then; if for each* $\mathbf{s} \in \mathcal{S}$ $v^{\hat{\pi}}(\mathbf{s}) \leq q^{\hat{\pi}}(\mathbf{s}, \pi'(\mathbf{s}))$ *it follows that* $v^{\hat{\pi}}(\mathbf{s}) \leq v^{\pi'}(\mathbf{s})$*.*

Proof. Beginning with the Bellman equation then for all $\mathbf{s} \in \mathcal{S}$

$$\begin{aligned} v^{\hat{\pi}}(\mathbf{s}) &= (T^{\hat{\pi}} v^{\hat{\pi}})(\mathbf{s}),\\ &= q^{\hat{\pi}}(\mathbf{s}, \hat{\pi}(\mathbf{s})),\\ &\leq q^{\hat{\pi}}(\mathbf{s}, \pi'(\mathbf{s})),\\ &= (T^{\pi'} v^{\hat{\pi}})(\mathbf{s}),\\ &\leq \lim_{\tau\to\infty} ((T^{\pi'})^{\tau} v^{\hat{\pi}})(\mathbf{s}),\\ &= v^{\pi'}(\mathbf{s}).\end{aligned} \tag{2.12}$$

□

Corollary 2.1. $v^{\hat{\pi}}(\mathbf{s}) \leq v^{\pi'}(\mathbf{s})$ *defines the ranking* $\hat{\pi} \leq \pi'$*. The strict inequality* $v^{\hat{\pi}}(\mathbf{s}) < v^{\pi'}(\mathbf{s})$ *holds if* π' *is identical to* $\hat{\pi}$ *over all* $\mathbf{s} \in \mathcal{S}$ *apart from choosing a better action (which receives a higher immediate reward) for at least one state. This is the criteria for a strictly improving policy and assumes* $\mathcal{S}$ *is discrete. If* $v^{\hat{\pi}}(\mathbf{s}) = v^{\pi'}(\mathbf{s})$ *over all states, then the policies are identical and no further improvement can be made. Further elaboration is discussed in Szepesvári (2010, Thm A.11)*

Definition 13 (Greedy policy distribution)**.** *Given an arbitrary policy* $\hat{\pi}$ *and related action-value function* $q^{\hat{\pi}}$*, a discrete greedy policy distribution is defined as*

$$\pi_{\text{greedy}}(\mathbf{a}|\mathbf{s}) = \begin{cases} 1, & \mathbf{a} == \arg\sup_{\mathbf{a}'\in\mathcal{A}} [q^{\hat{\pi}}(\mathbf{s}, \mathbf{a}')] \\ 0, & \textit{otherwise}, \end{cases} \tag{2.13}$$

Definition 14 (Greedy policy)**.** *For discrete action spaces $\mathcal{A}$ and an arbitrary $\hat{\pi}$, then a deterministic greedy policy relative to $\hat{\pi}$ is defined as*

$$\begin{aligned}\pi_{\text{greedy}}(\mathbf{s}) &:= \arg\sup_{\mathbf{a}\in\mathcal{A}} \Big[q^{\hat{\pi}}(\mathbf{s},\mathbf{a})\Big], && \forall \mathbf{s}\in\mathcal{S},\\ &= \arg\sup_{\mathbf{a}\in\mathcal{A}} \Big[r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{S_{t+1}|\mathbf{s},\mathbf{a}}[v^{\hat{\pi}}(S_{t+1})]\Big]\\ &=: \pi_{v^{\hat{\pi}}}(\mathbf{s}) = \pi_{q^{\hat{\pi}}}(\mathbf{s}),\end{aligned}$$

where notation in the last line makes explicit that the greedy policy is improving a different policy $\hat{\pi}$ associated with an existing state-action $q^{\hat{\pi}}$ or state value $v^{\hat{\pi}}$ function. A greedy policy meets the requirements of a strictly improving policy (corollary 2.1). It is assumed that there always exists a greedy policy in discrete action spaces but additional policy smoothness assumptions need to be made (Szepesvári, 2010) for continuous action spaces.

Lemma 1 (Equivalence of T^* and acting greedily)**.** *Given an arbitrary value function $v\in\mathcal{B}(\mathcal{S})$, $q(\mathbf{s},\mathbf{a}) := r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{S_{t+1}|\mathbf{s},\mathbf{a}}[v(S_{t+1})]$, $\pi_{greedy}(\mathbf{s}) := \arg\sup_{\mathbf{a}\in\mathcal{A}}\Big[q(\mathbf{s},\mathbf{a})\Big]$, then T^*v is equivalent to extracting π_{greedy} and applying $T^{\pi_{\text{greedy}}}v$.*

Proof.

$$\begin{aligned}T^{\pi_{\text{greedy}}}v &= q(\mathbf{s},\pi_{\text{greedy}}(\mathbf{s})),\\ &= r(\mathbf{s},\pi_{\text{greedy}}(\mathbf{s})) + \gamma\mathbb{E}_{S_{t+1}|\mathbf{s},\pi_{\text{greedy}}(\mathbf{s})}[v(S_{t+1})],\\ &= \sup_{\mathbf{a}\in\mathcal{A}}\Big[r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{S_{t+1}|\mathbf{s},\mathbf{a}}[v(S_{t+1})]\Big],\\ &= \sup_{\mathbf{a}\in\mathcal{A}}\Big[q(\mathbf{s},\mathbf{a})\Big],\\ &= T^*v.\end{aligned}$$

□

Combining the idea of policy improvement through acting greedily, then the identity of the fixed point of T^* can now be verified.

Theorem 3 (The fixed point of T^* is v^*, adapted from Szepesvári (2010, Thm A.10))**.** *If v_* is a fixed point of T^*, and applying T^* to a state-value function is equivalent to acting greedily $T^{\pi_{\text{greedy}}}v_* = T^*v_*$, then $v_* = v^*$ is the optimal value function and π_{v^*} is an optimal policy.*

Proof. Applying T^* once to an arbitrary $v\in\mathcal{B}(\mathcal{S})$ is equivalent to acting greedily over a one-step lookahead (see equation (2.11)). Then by the policy improvement theorem, the fixed point $\lim_{k\to\infty}(T^*)^k v_0 = v_*$ gives a new value function such that $v\leq v_*$. If $T^*v_* = v_*$, then this is equivalent to acting greedily with no improvement to the value function. When a value function cannot be improved as stated in equation (2.3),

then the fixed point $v_* = v^*$ is the optimal value function. Acting greedily against the optimal value function, $T^* v^* = v^*$ is equivalent to acting as the optimal policy π^*. □

For completeness, an equivalent Bellman optimality operator can also be derived from the action-value Bellman equation (2.10).

Definition 15 (Action-value Bellman optimality operator)**.** *The Action-value Bellman optimality operator acting on an arbitrary* $q \in \mathcal{B}(\mathcal{S}\times\mathcal{A})$*,* $T^* : \mathcal{B}(\mathcal{S}\times\mathcal{A}) \to \mathcal{B}(\mathcal{S}\times\mathcal{A})$ *is defined as*

$$(T^* q)(\cdot,\cdot) = r(\cdot,\cdot) + \gamma \mathbb{E}_{S_{t+1}|\cdot,\cdot}\Big[\sup_{\mathbf{a}_{t+1}\in\mathcal{A}} [q(S_{t+1}, \mathbf{a}_{t+1})]\Big],$$

where the fixed point is the optimal action-value function q^**.*

2.3 Implementation of DP Algorithms

The following summarises implementation of value prediction and control DP algorithms using theory developed in the previous section. Assumptions are that i) policies are deterministic and ii) the MDP is *known* (i.e. the reward function and transition kernel are known) and iii) *discrete* (i.e. both $\mathcal{S}$ and $\mathcal{A}$).

2.3.1 Value Prediction: Policy Evaluation

Given a finite MDP then the expectations in the Bellman operator (definition 10) are replaced with summations,

$$T^\pi v(\mathbf{s}) := r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s})) v(\mathbf{s}') \quad \mathbf{s}\in\mathcal{S}, \tag{2.14}$$

lending itself to vector form,

$$T^\pi \mathbf{v} := \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}, \tag{2.15}$$

where

$$\begin{aligned} \mathbf{v} &:= [v(\mathbf{s}_1), ..., v(\mathbf{s}_n)]^\top, \\ \mathbf{r}^\pi &:= [r(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., r(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top, \\ (\mathbf{P}^\pi)_{ij} &:= P(\mathbf{s}'_j|\mathbf{s}_i, \pi(\mathbf{s}_i)),\ \mathbf{P}^\pi \in \mathbb{R}^{n\times n}. \end{aligned}$$

As T^π is a sup-norm contraction with unique fixed point $\mathbf{v}^\pi$ then the sequence,

$$\mathbf{v}_0 \to (\mathbf{v}_1 = T^\pi \mathbf{v}_0) \to (\mathbf{v}_2 = T^\pi \mathbf{v}_1) \to \cdots \to \mathbf{v}^\pi,$$

converges geometrically to $\mathbf{v}^\pi$ at a rate of γ. This algorithm is known as *policy evaluation* (Sutton and Barto, 1998, Ch. 4, sec. 4.1) and is tractable for small discrete $\mathcal{S}$ and $\mathcal{A}$. Full knowledge of $\mathbf{P}^\pi$ and $\mathbf{r}^\pi$ are assumptions and the algorithm can either be carried out iteratively (see algorithm 1) at a computational cost $\mathcal{O}(n^2)$ or by inverting the vector equation (see algorithm 2) at a cost $\mathcal{O}(n^3)$. If fast matrix inversion algorithms are available, it might be quicker to invert equation (2.15) (after rearranging for $\mathbf{v}^\pi$) when $|\mathcal{S}|$ is low, rather than waiting for contractions to get small enough. However the iterative method fairs better as $|\mathcal{S}|$ gets large where convergence is γ-geometric which is fast.

Algorithm 1 Exact Policy Evaluation (Iterative T^π)

procedure PolicyEvaluation(π, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, $\mathbf{v}$, δ) ▷ $\mathcal{O}(|\mathcal{S}|^2)$
  Input: T^π components π, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, arbitrary value function $\mathbf{v}$, tolerance δ.
  Initialise: $(\mathbf{r}^\pi)_i := r(\mathbf{s}_i, \pi(\mathbf{s}_i))$, $(\mathbf{P}^\pi)_{ij} := P(\mathbf{s}_j|\mathbf{s}_i, \pi(\mathbf{s}_i))$.
  Output: $\mathbf{v} \approx \mathbf{v}^\pi$.
  do
    $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{v}$
    $\mathbf{v} \leftarrow \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}$ ▷ $\mathcal{O}(|\mathcal{S}|^2)$
  while $||\mathbf{v} - \mathbf{v}_{\text{prev}}||_\infty > \delta$ ▷ Stop contractions if oscillations are small
  return $\mathbf{v}$
end procedure

Algorithm 2 Exact Policy Evaluation (Invert T^π)

procedure PolicyEvaluationInversion(π, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ) ▷ $\mathcal{O}(|\mathcal{S}|^3)$
  Input: T^π components π, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ.
  Initialise: $(\mathbf{r}^\pi)_i := r(\mathbf{s}_i, \pi(\mathbf{s}_i))$, $(\mathbf{P}^\pi)_{ij} := P(\mathbf{s}_j|\mathbf{s}_i, \pi(\mathbf{s}_i))$.
  Output: $\mathbf{v}^\pi$
  $\mathbf{v}^\pi \leftarrow \left(\boldsymbol{I}_{|\mathcal{S}|} - \gamma \mathbf{P}^\pi\right)^{-1} \mathbf{r}^\pi$ ▷ $\mathcal{O}(|\mathcal{S}|^3)$
  return $\mathbf{v}^\pi$
end procedure

2.3.2 Control: Value Iteration

For the same discrete MDP the Bellman optimality operator (definition 12) is

$$T^* v(\mathbf{s}) := \sup_{\mathbf{a}\in\mathcal{A}} \left[r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) v(\mathbf{s}') \right] \quad \mathbf{s}\in\mathcal{S}. \tag{2.16}$$

As T^* a sup-norm contraction with unique fixed point $\mathbf{v}^*$, then the sequence,

$$\mathbf{v}_0 \to (\mathbf{v}_1 = T^*\mathbf{v}_0) \to (\mathbf{v}_2 = T^*\mathbf{v}_1) \to \cdots \to \mathbf{v}^*,$$

converges geometrically to $\mathbf{v}^*$ at a rate of γ. This is known as value iteration (VI), but there is no inverse solution to the Bellman optimality equations due to the non-linear sup operator and it must be performed iteratively (see algorithm 3).

Algorithm 3 Exact Value Iteration (Iterative T^*)

procedure ValueIteration($\mathcal{S}$, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, $\mathbf{v}$, δ) ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$
  Input: A set of states $\mathcal{S}$, T^* components $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, arbitrary value function $\mathbf{v}$, tolerance δ.
  Output: $\mathbf{v} \approx \mathbf{v}^*$.
  do
    $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{v}$
    for each $\mathbf{s} \in \mathcal{S}$ **do** ▷ $\mathcal{O}(|\mathcal{S}|)$
      $\mathbf{v}(\mathbf{s}) \leftarrow \sup_{\mathbf{a}\in\mathcal{A}} \left[r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \mathbf{v}(\mathbf{s}') \right]$ ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|)$
    end for
  while $||\mathbf{v} - \mathbf{v}_{\text{prev}}||_\infty > \delta$ ▷ Stop contractions if oscillations are small
  return $\mathbf{v}$
end procedure

2.3.3 Control: Policy Iteration

Policy iteration (PI) (Howard, 1960); (Sutton and Barto, 1998, Ch. 4, sec. 4.3), is the generalisation of value iteration. Recall that by lemma 1 then a single application of $T^*\mathbf{v}$ is the same as i) extracting a deterministic greedy policy $\pi_{\mathbf{v}}$ and then ii) making a single $J_{\text{eval}} = 1$ application of the state-value Bellman operator $T^{\pi_{\mathbf{v}}}$. Thus value iteration creates a sequence of greedy policy extraction and evaluation steps, which has already been proved to converge to v^*,

$$\mathbf{v}_0 \underset{\text{extract}}{\rightarrow} \pi_{\mathbf{v}_0} \underset{(T^{\pi_{\mathbf{v}_0}})\mathbf{v}_0}{\rightarrow} \mathbf{v}_1 \underset{\text{extract}}{\rightarrow} \ \dots \ \underset{\text{extract}}{\rightarrow} \pi_{\mathbf{v}_i} \underset{(T^{\pi_{\mathbf{v}_i}})\mathbf{v}_i}{\rightarrow} \mathbf{v}_{i+1} \underset{\text{extract}}{\rightarrow} \ \dots \ \mathbf{v}^*.$$

For policy iteration, the policy evaluation operator is applied more than once over $J_{\text{eval}} > 1$ times (or until convergence within a tolerance) of the value function at each i^{th} greedy policy extraction,

$$\mathbf{v}_0 \underset{\text{extract}}{\rightarrow} \pi_{\mathbf{v}_0} \underset{(T^{\pi_{\mathbf{v}_0}})^{J_{\text{eval}}}\mathbf{v}_0}{\rightarrow} \mathbf{v}_1 := \mathbf{v}^{\pi_{\mathbf{v}_0}} \underset{\text{extract}}{\rightarrow} \ \dots \ \underset{\text{extract}}{\rightarrow} \pi_{\mathbf{v}_i} \underset{(T^{\pi_{\mathbf{v}_m}})^{J_{\text{eval}}}\mathbf{v}_i}{\rightarrow} \mathbf{v}_{i+1} := \mathbf{v}^{\pi_{\mathbf{v}_i}} \underset{\text{extract}}{\rightarrow} \ \dots \ \mathbf{v}^*.$$

The same contraction arguments apply for the policy improvement step and therefore policy iteration converges to the optimal value function, see Szepesvári (2010, Thm A.11) for a formal proof. In the finite MDP setting there are a finite number of policies and states where convergence is guaranteed. If $i \leq J_{\text{imp}}$ is fixed, then value iteration incurs a total cost $\mathcal{O}(J_{\text{imp}}|\mathcal{A}||\mathcal{S}|^2)$ whereas if J_{eval} is also fixed then policy iteration incurs a total cost $\mathcal{O}(J_{\text{imp}}(J_{\text{eval}}|\mathcal{S}|^2 + |\mathcal{A}||\mathcal{S}|^2))$. Superficially it would seem

that value iteration has lower complexity. However as each policy improvement is made on a converged value function, PI converges to the optimal policy quicker than VI (Geramifard et al., 2013; Sutton and Barto, 1998) i.e. $J_{\text{imp}}^{(\text{PI})} << J_{\text{imp}}^{(\text{VI})}$, which has been confirmed experimentally during this investigation.

Algorithm 4 Exact Greedy Policy Improvement

procedure PolicyImprovement($\mathcal{S}$, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, $\mathbf{v}$) ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$
  Input: A set of states $\mathcal{S}$, T^* components $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, value function $\mathbf{v}$
  Output: $\pi_q(\cdot) := \pi_{\text{greedy}}(\cdot)$
  for each $\mathbf{s} \in \mathcal{S}$ **do** ▷ $\mathcal{O}(|\mathcal{S}|)$
    $\pi_{\mathbf{v}}(\mathbf{s}) \leftarrow \arg\sup_{\mathbf{a}\in\mathcal{A}}[r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'} P(\mathbf{s}'|\mathbf{s},\mathbf{a})\mathbf{v}(\mathbf{s}')]$ ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|)$
  end for
  return $\pi_{\mathbf{v}}(\cdot)$
end procedure

Algorithm 5 Exact Policy Iteration

procedure PolicyIteration($\mathcal{S}$, $\mathcal{A}$, $r(\cdot,\cdot)$,$P(\cdot|\cdot,\cdot)$, γ, δ) ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$
  Input: $\mathcal{S}$, $\mathcal{A}$, $r(\cdot,\cdot)$, $P(\cdot|\cdot,\cdot)$, γ, δ tolerance.
  Output: $\pi^*(\cdot)$
  Initialise: $\mathbf{v}$, $\pi(\cdot) \leftarrow$ PolicyImprovement$(r(\cdot,\cdot), P_(\cdot|\cdot,\cdot), \gamma, \mathbf{v}, \delta)$
  do
    $\pi_{\text{prev}}(\cdot) \leftarrow \pi(\cdot)$
    $\mathbf{v} \leftarrow$ PolicyEvaluation$(\pi, r(\cdot,\cdot), P(\cdot|\cdot,\cdot), \gamma, \mathbf{v}, \delta)$ ▷ $\mathcal{O}(|\mathcal{S}|^2)$
    $\pi(\cdot) \leftarrow$ PolicyImprovement$(r(\cdot,\cdot), P(\cdot|\cdot,\cdot), \gamma, \mathbf{v}, \delta)$ ▷ $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$
  while $||\pi - \pi_{\text{prev}}||_\infty > 0$ ▷ Stop if policy stable
  return $\pi^*(\mathcal{S})$
end procedure

2.3.4 Summary

The policy improvement theorem ensures there are *no* local optima for learning optimal policies if the following conditions are met; i) $\mathcal{S}$ is fully observable and discrete, ii) $\mathcal{A}$ is discrete, iii) the average reward function and transition dynamics are known and iv) Bellman operators use *full backups*, meaning that all states and actions are searched/updated over. DP control is therefore guaranteed to converge to the optimal policy. Loosely, there are a discrete number of actions to take at each discrete spatial point such that there exists a discrete number of policies $\pi \in \Pi$. As greedy policies are better or equal to the previous policy at collecting reward, then generalised policy iteration will converge in a finite number of steps.

Puterman and Shin (1978) develop *modified policy iteration* (MPI), describing the spectrum between value iteration and policy iteration with a full description of convergence guarantees. Common to all DP methods described here is *bootstrapping*,

meaning that in any iteration, value functions over states are calculated using the current estimate of the value function over successor states. DP *sequentially* executes two interacting, full-backup processes of i) making the value function consistent with the current policy and ii) acting greedily against the current value function. *Generalised policy iteration* (GPI) (Sutton and Barto, 1998, Ch. 4, sec. 4.8) relaxes the synchronous, full-backup update requirement e.g. Bertsekas and Yu (2010) develop an asynchronous GPI. GPI describes the general description of updating value estimates and improving policies in value-based RL.

In real world settings it would seem that DP algorithms are useless as the reward function and transition dynamics must be known. In their exact form, DP method computational complexity scales as $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$ which is clearly not sustainable for large MDPs and intractable for continuous state spaces. DP algorithms are usually abandoned in favour of model-free algorithms in unknown MDPs.

2.4 Model-Free RL in Unknown Discrete MDPs

Model-free policy learning methods are an important part of the discussion because their stability has been extensively studied w.r.t. bootstrapping, function approximation and off-policy learning which are all relevant issue for model-based methods. It is also vital that the shortcomings of model-free methods are reviewed which motivates the development of the model-based algorithms in this investigation.

2.4.1 Value Prediction: TD Learning

Assuming a discrete MDP, the policy evaluation DP method (section 2.3.1) adjusts the current value function estimate by

$$v(\mathbf{s}) \mapsto (T^\pi v)(\mathbf{s}) := \mathbb{E}_{R_{t+1},S_{t+1}|\mathbf{s},\pi(\mathbf{s})}\Big[R_{t+1} + \gamma v(S_{t+1})\Big],\ \forall \mathbf{s} \in \mathcal{S}. \tag{2.17}$$

This is known as a *full backup* which is enabled by having full access to the MDP (i.e. reward function and transition dynamics are known), allowing T^π to be iterated until value function convergence. Value-based *model-free* algorithms do not have access to the MDP and instead typically update value functions using sequential *sample* state-value Bellman updates,

$$v(\mathbf{s}_t) \mapsto (\hat{T}^\pi v)(\mathbf{s}_t) := r_{t+1} + \gamma v(\mathbf{s}_{t+1}), \quad \mathbf{a}_t = \pi(\mathbf{s}_t),\ (r_{t+1}, \mathbf{s}_{t+1}) \sim P(\cdot|\mathbf{s}_t, \mathbf{a}_t). \tag{2.18}$$

A value function v^π specifies an average of the cumulative discounted return when following a policy π. Lemma B.5 shows that an empirical average quantity can be maintained by an online average as each new sample is experienced. The revered TD($\lambda = 0$) algorithm (Sutton, 1988) incorporates a single $n = 1$ step lookahead target

(equation (2.18)) into an online average such that

$$\delta(\mathbf{s}_t) \leftarrow (\hat{T}^\pi v)(\mathbf{s}_t) - v(\mathbf{s}_t),$$
$$v(\mathbf{s}_t) \leftarrow v(\mathbf{s}_t) + \alpha_t \delta(\mathbf{s}_t),$$

where $\delta(S_t)$ is the TD error and α_t is a learning rate. TD(λ) (Sutton, 1988) targets generalise over a mixture of n-step returns which is controlled by picking $\lambda \in [0, 1]$.

Definition 16 (TD-λ Bellman operator)**.** *Given the multi-step Bellman operator induced by π,*

$$(T^\pi_{[n]} v)(\mathbf{s}_t) = \mathbb{E}_{\Xi_{t:n} \sim p(\cdot|S_t=\mathbf{s}_t),\ S_{t+n} \sim P(\cdot|S_{t+n-1}, \pi(S_{t+n-1}))} \Big[R^\gamma(\Xi_{t:n}) + \gamma^n v(S_{t+n}) \Big], \quad \mathbf{s}_t \in \mathcal{S},$$

then the TD-λ Bellman operator is

$$\begin{aligned}(T^\lambda v)(\mathbf{s}_t) &= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} T^\pi_{[n]} v(\mathbf{s}_t), \quad \mathbf{s}_t \in \mathcal{S}, \\ &= v(\mathbf{s}_t) + (I - \gamma\lambda P^\pi)^{-1} \Big((T^\pi v)(\mathbf{s}_t) - v(\mathbf{s}_t) \Big),\end{aligned} \tag{2.19}$$

where by inspection v^π is a fixed point (see lemma B.4 for derivation).

This leads to the forward-view TD(λ) sample targets defined as

$$(\hat{T}^\pi_{[n]} v)(\mathbf{s}_t) := \sum_{\tau=1}^{n} \gamma^{\tau-1} r_{t+\tau} + \gamma^n v(\mathbf{s}_{t+n}),$$

where $\lambda \in [0, 1]$, which defines the original Bellman operator as $\hat{T}^\pi := \hat{T}^{\lambda=0}$. If a terminal state is reached at $t=H$, then the series is truncated to

$$(\hat{T}^\lambda v)(\mathbf{s}_t) := (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} (\hat{T}^\pi_{[n]} v)(\mathbf{s}_t) + \lambda^{H-1} r^{\text{MC}}_{t+H},$$

where r^{MC}_{t+H} is the Monte Carlo return. TD(λ) unifies the n-step return, from single step targets (λ=0), all the way to Monte Carlo (λ=1) targets, with a mixture in between. *Forward* TD(λ) is the theoretical variant that sets the TD target as $(\hat{T}^\lambda v)(\mathbf{s}_t)$. When $\lambda > 0$ then *backwards* TD(λ) is a practical implementation of the update that uses *eligibility traces* that don't require the sampling and storing of entire n-step trajectory returns. Eligibility traces assign credit to recently frequented states and are mathematically equivalent to using the forward $(\hat{T}^\lambda v)(\mathbf{s}_t)$ target. The following $\lambda > 0$ backwards update is

$$\delta(\mathbf{s}_t) \leftarrow (\hat{T}^\pi v)(\mathbf{s}_t) - v(\mathbf{s}_t),$$
$$e(\mathbf{s}_t) \leftarrow 1 + \gamma\lambda e(\mathbf{s}_t),$$
$$v(\mathbf{s}_t) \leftarrow v(\mathbf{s}_t) + \alpha_t \delta(\mathbf{s}_t) e(\mathbf{s}_t),$$

where $e(\mathbf{s}_t)$ is the eligibility trace at $\mathbf{s}_t$. For episodic tasks, online forward and backwards value function updates (after each sample) are slightly different to a single batch update (occurring at the end of an episode). Seijen and Sutton (2014) eliminate this difference by modifying the eligibility trace mechanism.

Algorithm convergence is dependent upon α_t satisfying the Robbins-Monro (RM) conditions (Szepesvári, 2010, p. 13) (see also equation (B.71)). For a given MDP and stationary policy, tabular TD algorithms asymptotically converge to the value function, but do so at different rates depending on λ. The reason being is that the value of λ specifies a particular trade-off between bias and variance of the target. For low λ bias is high and variance is low. To illustrate, each TD(0) (equation (2.18)) target has low variance because they only need one sample of immediate reward and therefore their exposure to the MDP's stochasticity is limited. However the target also bootstraps the current estimate of the value function. The constant movement of the bootstrapped target introduces bias into the update. On the other hand TD(1) (or Monte Carlo) estimate has high variance because it consists entirely of many reward samples from the environment, but low bias because no estimate of the value function is required (Sutton and Barto, 1998, Ch 4.8). For more detailed discussions see Szepesvári (2010, sec 2.1.2-3), Sutton and Barto (1998, sec 6.2-3) and references therein. Sutton and Barto (1998, figure 7.9) and Sutton (1996) include empirical analysis of this λ-specified trade-off on a random walk and puddle world respectively.

2.4.2 Control: SARSA & Q-Learning

Assuming a discrete MDP is known, then the *model-based* value iteration DP method (section 2.3.2) adjusts the current value function estimate towards the optimal value function,

$$v(\mathbf{s}) \mapsto (T^* v)(\mathbf{s}) = \sup_{\mathbf{a} \in \mathcal{A}} \Big[\mathbb{E}_{R_{t+1}, S_{t+1} | \mathbf{s}, \mathbf{a}} [R_{t+1} + \gamma v(S_{t+1})] \Big], \ \forall \mathbf{s} \in \mathcal{S}, \tag{2.20}$$

which is also a *full backup*. Value-based *model-free* control algorithms use *sample* action-value Bellman operator (equation (2.10)) backups,

$$q(\mathbf{s}_t, \mathbf{a}_t) \mapsto (\hat{T}^\pi q)(\mathbf{s}_t, \mathbf{a}_t) := r_{t+1} + \gamma q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \quad (r_{t+1}, \mathbf{s}_{t+1}) \sim P(\cdot | \mathbf{s}_t, \mathbf{a}_t), \tag{2.21}$$

where $\mathbf{a}_{t+1} = \pi(\mathbf{s}_{t+1})$ is known as the *target policy* because it is used to decide what bootstrapped estimate is used to form $\hat{T}^\pi$. The TD(0) online update for the action-value function is

$$q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow q(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t \Big((\hat{T}^\pi q)(\mathbf{s}_t, \mathbf{a}_t) - q(\mathbf{s}_t, \mathbf{a}_t) \Big). \tag{2.22}$$

An agent draws actions from a *behaviour* policy $A_t \sim \nu(S_t)$ which conditions any transition. Similarly the TD-target is formed from actions sampled from the *target*

policy $A_{t+1} \sim \pi(S_{t+1})$. Such an arrangement allows the learning of the target policy while following the behaviour policy. The SARSA (Rummery and Niranjan, 1994) control algorithm specifies an ϵ-greedy policy for both its target and behaviour policies, which means that it is an *on-policy* control algorithm because $\pi = \nu$. By decaying ϵ to zero over an infinite number of updates (e.g. $\epsilon = \frac{1}{t}$), also known as *greedy in the limit of infinite exploration* (GLIE), then along with the RM conditions this ensures SARSA converges to the optimal policy.

Definition 17 (ϵ-greedy policy distribution (Sutton and Barto, 1998, Ch 5.4))**.** *An ϵ-greedy policy is defined by taking the greedy action w.p. $1-\epsilon$ and any action w.p. ϵ.*

$$\pi(\mathbf{a}|\mathbf{s}) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}|, & \mathbf{a} == \arg\sup_{\mathbf{a}} [q(\mathbf{s}, \mathbf{a})] \\ \epsilon/|\mathcal{A}|, & \textit{otherwise.} \end{cases} \tag{2.23}$$

Q-learning (Watkins, 1989; Watkins and Dayan, 1992) specifies the target policy as the greedy policy $\pi = \pi_{\text{greedy}}$. The TD target is therefore

$$(\hat{T}^{\pi_{\text{greedy}}} q)(\mathbf{s}_t) := (\hat{T}^{*} q)(\mathbf{s}_t) = r_{t+1} + \gamma \sup_{\mathbf{a}_{t+1} \in \mathcal{A}} [q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})],$$

which is equivalent to making a sample backup of the action-value Bellman optimality operator (equation (15)). This is an *off-policy* algorithm because it draws behavioural actions (and therefore generates a data distribution) from an annealed ϵ-greedy policy (w.r.t to the current q estimate) while learning the greedy policy (specified in its targets) such that $\nu \neq \pi$. The advantage of an off-policy algorithm is that historical data or data gathered from an additional exploration policy can be used to learn the optimal policy. Recent work has shown that convergence is still possible with a relaxation of GLIE (Harutyunyan et al., 2016) for the off-policy case. Equivalent SARSA(λ) (Sutton and Barto, 1998, sec. 7.5) and Q(λ) (Peng and Williams, 1996; Watkins, 1989) exist but are out of the scope of this investigation.

2.5 Model-Free Approximate Policy Iteration

Now that tabular model-free methods have been introduced, it is now possible to discuss the notoriety of *approximate* RL and eventually how the algorithms studied in this investigation (based on DP) mitigate policy learning instabilities. Even for small discrete state spaces, Bellman's curse of dimensionality tells us that the number of states grows exponentially in the dimensionality of the state space. Tabular DP planning scales quadratically with the size of the state space and therefore rapidly becomes impractical. Tabular model-free updates rapidly become ineffective because the probability of returning to a state more than once in order to assign temporal

credit assignment diminishes to zero. It is infeasible to update tabular value functions over continuous state spaces.

Function approximation is a solution that enables approximate value functions $v^\pi \approx \hat{v}^\pi \in \mathcal{F}$ to generalises over states where function class $\mathcal{F}$ is to be picked. Linear parametric, non-parametric and intrinsically non-linear function classes are all candidates. However function approximation in RL, both in the model-based and mode-free contexts is *notoriously* difficult. The purpose of this review is to navigate these issues before the development of the algorithms in this thesis. Approximate policy iteration (API) describes control algorithms whose structure consists of value prediction and policy improvement as described by fig. 1.1, however state-value functions and/or action-value functions are estimated with function approximators. This review focusses on linear function approximation and compares supervised learning with iterative approximate TD methods.

2.5.1 Value Function Approximation (Supervised Learning)

An extensive review of function approximation, empirical risk minimisation (ERM), penalised ERM (PERM) and training different function classes can be found in section B.4. In order to obtain the machinery for contraction mappings and DP algorithms, value functions were assumed to reside in a Banach space (see definition 8). Extending this perspective to accommodate function approximation, value functions are assumed to reside in a Hilbert space $\mathcal{H}(\mathcal{S})$ whose elements are vectors with dimensionality equal to the number of states $m = |\mathcal{S}|$ in the MDP.

Definition 18 (Value function space)**.** *Value functions are assumed to exist in a Hilbert space* $\mathcal{H}(\mathcal{S}) := \{v : \mathcal{S} \to \mathbb{R} \mid ||v||_D = \sqrt{\sum_{\mathbf{s}\in\mathcal{S}} D(\mathbf{s})v^2(\mathbf{s})} < +\infty\}$*, where* $D(\cdot)$ *is some density over states.*

Clearly if the state space is continuous then the norm would be defined by integration over a measurable space, however for ease of exposition we will assume $\mathcal{S}$ is discrete. It is more intuitive to adopt vector notation such that the value function space is $\mathcal{H}(\mathcal{S}) := \{\mathbf{v} \in \mathbb{R}^m \mid ||\mathbf{v}||_D = \sqrt{\mathbf{v}^\top \mathbf{D}\mathbf{v}} < +\infty\}$ where $||\cdot||_D$ is the D-weighted norm whose matrix $\mathbf{D}$ is positive definite and weights the i^{th} state by $D(\mathbf{s}_i)$ along its diagonal ($\mathbf{D} = \mathbf{I}$ if states are equally weighted). A natural choice for the distribution over states is $D = D^\pi$, the (time-invariant) stationary distribution (definition 7) of the MDP. For any stationary policy π, a natural performance objective of a function approximator v

is the mean squared error (MSE),

$$\begin{aligned}\mathcal{L}_{\text{MSE}}(\mathbf{v}) &:= ||\mathbf{v} - \tilde{\mathbf{v}}^\pi||_D^2 \\ &= \Big(\mathbf{v} - \tilde{\mathbf{v}}^\pi\Big)^\top \mathbf{D}\Big(\mathbf{v} - \tilde{\mathbf{v}}^\pi\Big), \\ &= \mathbb{E}_{S\sim D(\cdot)}\Big[\Big(v(S) - \tilde{v}^\pi(S)\Big)^2\Big], \\ &= \sum_{\mathbf{s}\in\mathcal{S}} D(\mathbf{s})\Big(v(\mathbf{s}) - \tilde{v}^\pi(\mathbf{s})\Big)^2,\end{aligned} \tag{2.24}$$

where $\mathbf{v} := [v(\mathbf{s}_1), ..., v(\mathbf{s}_m)]$ and $\tilde{v}^\pi$ is the target function value that v is being measured against. If available the targets would be set as the real value functions $\tilde{v}^\pi = v^\pi$.

Linear Approximation Scheme A linear function approximation scheme is defined as $v(\mathbf{s}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{s})\rangle_{\mathcal{F}}$. Choosing $\mathcal{F} = \mathcal{F}_\phi$ specifies a class of linear parametric function approximators that require an *explicit* state representation function which maps states to a feature space $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}_\phi$. The ideal loss and solution (for actual value function targets) is therefore

$$\begin{aligned}&\mathcal{L}_{\text{MSE}}(\mathbf{w}) := ||\boldsymbol{\Phi}\mathbf{w} - \mathbf{v}^\pi||_D^2, \\ \implies\ &\mathbf{w}^\pi_{\text{MSE}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d}\Big[\mathcal{L}_{\text{MSE}}(\mathbf{w})\Big].\end{aligned} \tag{2.25}$$

where $\mathbf{v} = \boldsymbol{\Phi}\mathbf{w}$, $\mathbf{w} := [w_1, ..., w_d]^\top \in \mathbb{R}^d$, $d = \dim(\mathcal{F}_\phi)$ and $\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{s}_1), .., \boldsymbol{\phi}(\mathbf{s}_m)]^\top \in \mathbb{R}^{m\times d}$ is the design matrix. If $\boldsymbol{\Phi}$ is full rank where $n \geq d$ and all columns are linearly independent then a minimiser exists and is the MSE solution $\mathbf{w}^\pi_{\text{MSE}} = (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{v}^\pi$ (see section B.4.1). If m is very large and D is unknown, empirical risk minimisation (ERM) samples data $\mathcal{D}_n := \{(\mathbf{s}, v^\pi(\mathbf{s}))_i\}_{i=1}^n$ to form empirical loss $\hat{\mathcal{L}}$ whose sample estimate of the minimiser is $\hat{\mathbf{w}}^\pi_{\text{MSE}} = (\hat{\boldsymbol{\Phi}}^\top\mathbf{D}\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\Phi}}^\top\mathbf{D}\mathbf{v}^\pi$ where $\hat{\boldsymbol{\Phi}} := [\boldsymbol{\phi}(\mathbf{s}_1), .., \boldsymbol{\phi}(\mathbf{s}_n)]^\top \in \mathbb{R}^{n\times d}$ and each sample is equally weighted $\mathbf{D} := \mathbf{I}/n$.

The geometry of the least squares solution is the key to understanding the effect of applying the Bellman operator and therefore algorithm convergence. Consider the MSE solution $\mathbf{v}^\pi_{\text{MSE}}$ as a linear combination of the basis elements (defined by the

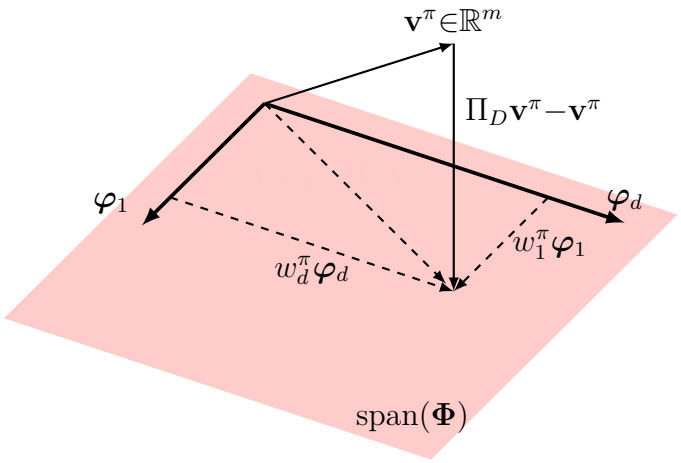
columns in the design matrix),

$$\begin{aligned}\mathbf{v}^\pi_{\text{MSE}} &= \mathbf{\Phi}\mathbf{w}^\pi_{\text{MSE}}, \\ &= \begin{pmatrix} — & \boldsymbol{\phi}^\top(\mathbf{s}_1) & — \\ & \vdots & \\ — & \boldsymbol{\phi}^\top(\mathbf{s}_m) & — \end{pmatrix} \begin{pmatrix} w^\pi_1 \\ \vdots \\ w^\pi_d \end{pmatrix}, \\ &= \begin{pmatrix} | & & | \\ \boldsymbol{\varphi}_1 & \cdots & \boldsymbol{\varphi}_d \\ | & & | \end{pmatrix} \begin{pmatrix} w^\pi_1 \\ \vdots \\ w^\pi_d \end{pmatrix}, \\ &= \sum_{i=1}^{d} w^\pi_i \boldsymbol{\varphi}_i \,. \end{aligned} \tag{2.26}$$

Substituting the least squares solution $\mathbf{w}^\pi_{\text{MSE}}$ into equation (2.26),

$$\begin{aligned}\mathbf{v}^\pi_{\text{MSE}} &= \mathbf{\Phi}\mathbf{w}^\pi_{\text{MSE}} \\ &= \mathbf{\Phi}(\mathbf{\Phi}^\top\mathbf{D}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{D}\mathbf{v}^\pi \\ &= \Pi_D\mathbf{v}^\pi, \end{aligned} \tag{2.27}$$

Figure 2.2 Directly fitting $\mathbf{v}^\pi \in \mathbb{R}^m$ by orthogonal projection onto $\text{span}(\mathbf{\Phi}) \subset \mathbb{R}^m$.



where $\mathbf{v}^\pi = (\mathbf{I} - \mathbf{P}^\pi)^{-1}\mathbf{r}^\pi$ if the MDP is known. The operator Π_D projects (w.r.t. the weighted norm $||\cdot||_D$) the target vector $\mathbf{v}^\pi \in \mathcal{H}(\mathcal{S})$ onto a lower dimensional subspace, whose basis is the column space of $\mathbf{\Phi}$ i.e. $\text{span}(\mathbf{\Phi}) := \{\mathbf{\Phi}\mathbf{w} \,|\, \mathbf{w} \in \mathbb{R}^d\} \subset \mathcal{H}(\mathcal{S})$ (fig. 2.2).

An explicit feature map $\boldsymbol{\phi}(\mathbf{s}) := [1(\mathbf{s} = \mathbf{s}_1), ..., 1(\mathbf{s} = \mathbf{s}_d)]^\top$ will recover the tabular representation such that for the i^{th} state, $w^\pi_i = v^\pi(\mathbf{s}_i)$. However for large state spaces such a tabular representation becomes an unusable infinite dimensional matrix. Instead a feature map $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}_\phi$ defining a lower-dimensional rich state-representation is sought for $d << m$ which can either be specified a priori or learnt using a feature selection scheme. The choice of $\boldsymbol{\phi}$ is a compromise between *expressiveness* (or how

rich the representation is in order to generalise over states) and *compactness* (where $d<<m$ is purposefully chosen to minimise computational costs). Tsitsiklis and Van Roy (1996) summarise ways of integrating feature selection into linear RL, Sutton and Barto (1998, sec. 8.3); Bertsekas (2012, Section 6.1.1) list feature representation types for linear methods such as coarse coding, tile coding, radial basis functions and high degree polynomial representations (Yao and Szepesvári, 2012).

2.5.2 Approximate Value Prediction

Actual value function target vectors $\tilde{\mathbf{v}}^\pi = \mathbf{v}^\pi$ are not available from an oracle in the RL problem setting, thus rendering v^π estimation more difficult than regular supervised learning techniques. Instead value prediction must proceed with careful analysis of the Bellman operator on linear value function approximators.

It quickly becomes apparent that contraction guarantees are not forthcoming for linear function approximation schemes. Instead the solution to approximate value prediction is to either minimise the mean squared Bellman error/residual (MSBE) or projected Bellman error/residual (MSPBE). Both approaches can be considered as minimising the original supervised MSE whose targets are proxies formed by applying the Bellman operator to the function approximator; $\tilde{\mathbf{v}}^\pi = T^\pi \mathbf{v}$ and $\tilde{\mathbf{v}}^\pi = \Pi_D T^\pi \mathbf{v}$ respectively which have closed form least squares solutions utilised in this investigation.

Dynamic Programming with Linear Value Function Approximation

Considering the geometry of how the application of $T^\pi v$ changes v in feature space is the key to understanding value prediction convergence and why certain approximate RL objective functions are used. Consider a linear parametric approximator then the Bellman equation applied is

$$\begin{aligned} \langle \mathbf{w}^\pi, \boldsymbol{\phi}(\mathbf{s}) \rangle_{\mathcal{F}_\phi} &\approx r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \mathbb{E}_{S' \sim P(\cdot|\mathbf{s}, \pi(\mathbf{s}))} \Big[\langle \mathbf{w}^\pi, \boldsymbol{\phi}(S') \rangle_{\mathcal{F}_\phi} \Big], \\ &= r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \Big\langle \mathbf{w}^\pi, \mathbb{E}_{S' \sim P(\cdot|\mathbf{s}, \pi(\mathbf{s}))} [\boldsymbol{\phi}(S')] \Big\rangle_{\mathcal{F}_\phi}. \end{aligned} \tag{2.28}$$

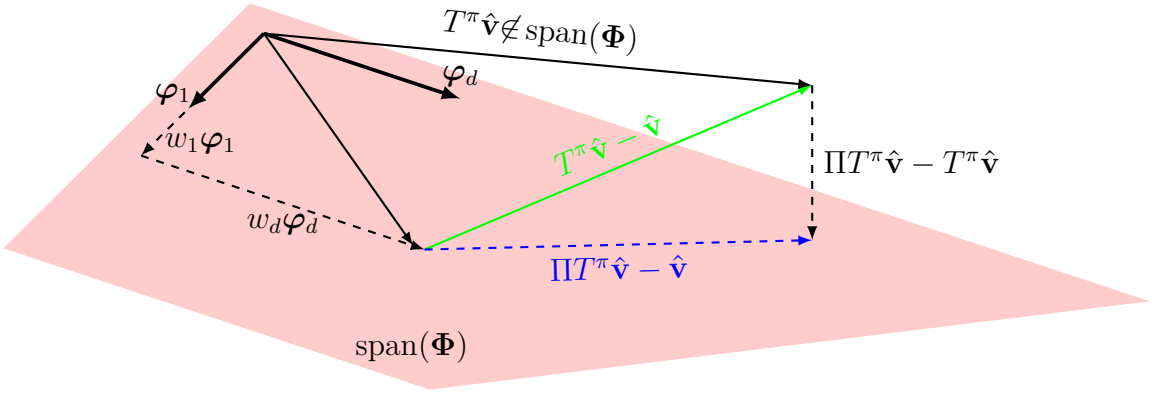
Unfortunately the relationship (2.28) is approximate because no fixed point solution $\mathbf{w}^\pi$ can be found to preserve equality when *any explicit* $\boldsymbol{\phi}$ is provided a priori (Bertsekas, 2012). The reason is better illustrated using vector notation as follows,

$$\begin{aligned} T^\pi(\boldsymbol{\Phi}\mathbf{w}) &:= \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \boldsymbol{\Phi} \mathbf{w}, \\ &= \mathbf{r}^\pi + \gamma \boldsymbol{\Phi}' \mathbf{w}, \end{aligned} \tag{2.29}$$

where $\boldsymbol{\Phi}' := [\mathbb{E}_{S' \sim P(\cdot|\mathbf{s}_1, \pi(\mathbf{s}_1))}[\boldsymbol{\phi}(S')], ..., \mathbb{E}_{S' \sim P(\cdot|\mathbf{s}_n, \pi(\mathbf{s}_n))}[\boldsymbol{\phi}(S')]]^\top$. The application of T^π takes the value function approximation into a new basis (or column space of) $\boldsymbol{\Phi}'$ induced by $\mathbf{P}^\pi$, as illustrated in fig. 2.3. Therefore the Bellman operator, which is

non-expansive under the norm $||\cdot||_\infty$, is expansive in the norm $||\cdot||_D$ if (as reviewed below) the distribution of states D is not carefully selected. Two major objective functions are used to mitigate this divergence effect. Vector or function notation is used where appropriate.

Figure 2.3 Approximate value prediction by applying T^π to value functions in a linear function approximation scheme. Mitigating this is achieved either by i) minimising the magnitude of the Bellman residual vector known as Bellman residual minimisation (BRM), or ii) minimising the magnitude of the **projected** Bellman residual vector such as in TD fixed point methods.



Bellman Residual (MSBE Objective)

The Bellman residual (BR) is formed (see fig. 2.3) by choosing the target in equation (2.24) as $\tilde{\mathbf{v}}^\pi = T^\pi\mathbf{v}^\pi$. However as $\mathbf{v}^\pi$ is not available, the proxy target $T^\pi\mathbf{v}$ is chosen instead to find the fixed point $\mathbf{v}_{\text{BRM}}$ of $\mathbf{v} = T^\pi\mathbf{v}$. The mean squared Bellman error (MSBE) objective (Baird, 1995) is

$$\begin{aligned}
\mathcal{L}_{\text{MSBE}}(\mathbf{v}) &= ||\mathbf{v} - T^\pi\mathbf{v}||_D^2, \\
&= \mathbb{E}_{S\sim D}\Big[\Big(v(S) - T^\pi v\Big)^2\Big], \\
&= \mathbb{E}_{S\sim D}\Big[\Big(v(S) - \mathbb{E}_{R,S'|S,\pi(S)}\Big[R + \gamma v(S')\Big]\Big)^2\Big], \qquad (2.30)\\
&= \mathbb{E}_{S\sim D}\Big[\Big(v(S) - \bar{Y}\Big)^2\Big], \qquad (2.31)\\
&= \mathbb{E}_{S\sim D}\Big[\Big(\mathbb{E}_{R,S'|S,\pi(S)}\Big[\delta(S,R,S')\Big]\Big)^2\Big], \qquad (2.32)
\end{aligned}$$

where $Y(R,S') := R + \gamma v(S')$, $\bar{Y} := \mathbb{E}_{R,S'|S,\pi(S)}\Big[Y(R,S')\Big]$ and the TD(0) error for function approximator $\hat{v}$ is $\delta(S,R,S') := Y(R,S') - v(S)$. BRM objective functions suffer the problem of having the successor state expectation *inside* the squared loss term (2.32) whose consequences are described as follows. Further manipulation under

the assumption of linear value function approximation gives

$$\mathcal{L}_{\text{MSBE}}(\mathbf{w}) = ||\mathbf{\Phi}\mathbf{w} - T^{\pi}\mathbf{\Phi}\mathbf{w}||^2_D, \tag{2.33}$$

$$= \mathbb{E}_{S\sim D}\Big[\Big(\big(\boldsymbol{\phi}^{\top}(S) - \mathbb{E}_{S'|S,\pi(S)}\big[\gamma\boldsymbol{\phi}^{\top}(S')\big]\big)\mathbf{w} - \mathbb{E}_{R|S,\pi(S)}\big[R\big]\Big)^2\Big],$$

$$= \mathbb{E}_{S\sim D}\Big[\Big(\mathbf{C}^{\pi\top}(S)\mathbf{w} - R^{\pi}(S)\Big)^2\Big]. \tag{2.34}$$

Equation (2.34) looks like a regular supervised loss problem that learns linear function weights using inputs $\mathbf{C}^{\pi\top}(S)$ and target outputs $R^{\pi}(S)$, naively estimated from sample pairs $(\boldsymbol{\phi}(\mathbf{s}) - \gamma\boldsymbol{\phi}(\mathbf{s}'), r)$ taken from trajectory data (Dann et al., 2014). However empirical risk minimisation assumes only noise in the *target* variables (see section B.4.1) and does not consider noise in the input samples generated from the MDP's stationary transition distribution. This adds bias to the estimator (Bradtke and Barto, 1996, p. 40) which is known as the *error-in-variables* problem and can be explored by considering the following.

Instead of the objective equation (2.31), it is preferable to specify the ERM problem with the transition expectation outside the squared loss, labelling this new objective the mean squared TD error (MSTDE) i.e.

$$\mathcal{L}_{\text{MSTDE}}(v) := \mathbb{E}_{S\sim D}\mathbb{E}_{R,S'|S,\pi(S)}\Big[\delta^2(S,R,S')\Big], \tag{2.35}$$

$$= \mathbb{E}_{S\sim D}\mathbb{E}_{R,S'|S,\pi(S)}\Big[\Big(v(S) - Y(R,S')\Big)^2\Big],$$

where trajectory data $\mathcal{D} := \{(\mathbf{s},\mathbf{a},r,\mathbf{s}')_i\}_{i=1}^n$ is used to form the empirical loss

$$\hat{\mathcal{L}}_{\text{MSTDE}}(v) = \frac{1}{2n}\sum_{i=1}^{n}\Big(v(\mathbf{s}_i) - r_i - \gamma v(\mathbf{s}'_i)\Big)^2. \tag{2.36}$$

However $\lim_{n\to\infty}[\hat{\mathcal{L}}_{\text{MSTDE}}(v)] \neq \mathcal{L}_{\text{MSBE}}(v)$ because of the aforementioned bias (Antos et al., 2008; Farahmand et al., 2008) and is demonstrated by applying a bias-variance analysis (section B.4.1) to the inner expectation of the MSTDE (2.35);

$$\mathbb{E}_{R,S'|S,\pi(S)}\Big[\Big(v(S) - Y(R,S')\Big)^2\Big]$$

$$= \mathbb{E}_{R,S'|S,\pi(S)}\Big[\Big(v(S) - \bar{Y} + \bar{Y} - Y(R,S')\Big)^2\Big],$$

$$= \mathbb{E}_{R,S'|S,\pi(S)}\Big[\Big(v(S) - \bar{Y}\Big)^2 + \Big(\bar{Y} - Y(R,S')\Big)^2\Big],$$

$$= \Big(v(S) - \bar{Y}\Big)^2 + \mathbb{E}_{R,S'|S,\pi(S)}\Big[\Big(\bar{Y} - Y(R,S')\Big)^2\Big],$$

$$= \Big(v(S) - T^{\pi}v\Big)^2 + \text{var}_{R,S'|S,\pi(S)}\Big[Y(R,S')\Big].$$

Minimising objective (2.36) minimises both (2.32) and variance term induced by the MDP's stochastic transition dynamics. To remedy this, Baird (1995); Sutton and Barto (1998) *double-sample* the pair (R,S') iid to create an unbiased loss. The empirical loss

is therefore

$$\hat{\mathcal{L}}_{BR}(v) := \frac{1}{2n}\sum_{i=1}^{n}\Big(v(\mathbf{s}_i) - r_i - \gamma v(\mathbf{s}_i')\Big)\Big(v(\mathbf{s}_i) - r_i - \gamma v(\mathbf{s}_i'')\Big),$$

where for simplicity the reward is assumed independent of the successor state, confirming $\lim_{n\to\infty}[\hat{\mathcal{L}}_{\text{BR}}(v)] = \mathcal{L}_{\text{BR}}(v)$. In conclusion the single successor state sampling BRM solution is $\hat{v}_{\text{TDE}}$ and $\hat{v}_{\text{BR}}$ with double sampling. However double sampling is not practical when only trajectory data is available as only *one* transition is experienced from each state, rendering this technique impossible unless access to a generative model of the MDP is available (Sutton and Barto, 1998, p 220). Antos et al. (2008) suggest a solution by introducing an auxiliary function to directly cancel the variance term and (as described below) TD learning methods Mnih et al. (2015) fix target function approximator weights such that they are not involved in the optimisation.

If double sampling is available via a model, Baird (1995) solves the BRM problem using stochastic gradient descent by differentiating equation (2.30),

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L}_{\text{BR}} &= \mathbb{E}_{S\sim D}\Bigg[\Big(v(S) - \mathbb{E}_{R,S'|S,\pi(S)}\Big[R + \gamma v(S')\Big]\Big)\Big(\gamma\mathbb{E}_{S''|S,\pi(S)}\Big[\nabla_{\mathbf{w}}v(S) - \nabla_{\mathbf{w}}v(S'')\Big]\Big)\Bigg],\\
&\approx \Big(\Big(\boldsymbol{\phi}^\top(\mathbf{s}) - \gamma\boldsymbol{\phi}^\top(\mathbf{s}')\Big)\mathbf{w} - r\Big)\Big(\boldsymbol{\phi}(\mathbf{s}) - \gamma\boldsymbol{\phi}(\mathbf{s}'')\Big),\\
&= \delta(\mathbf{s}, r, \mathbf{s}')\Big(\gamma\boldsymbol{\phi}(\mathbf{s}'') - \boldsymbol{\phi}(\mathbf{s})\Big).\end{aligned}$$

This gives the full gradient of the Bellman residual and in the last line an unbiased sample gradient is made using data $(\mathbf{s}, \pi(\mathbf{s}), r, \mathbf{s}', \mathbf{s}'')$. By performing stochastic gradient descent with sample weight updates formed by drawing samples from an experience replay memory $\mathcal{D} := \{(\mathbf{s}, \pi(\mathbf{s}), r, \mathbf{s}', \mathbf{s}'')_i\}_{i=1}^n$ until convergence is one approach to solve for the value function. Equivalently the least squares solution of equation (2.34) is calculated by

$$\begin{aligned}\mathbf{0} &= \nabla_{\mathbf{w}}\Big(\frac{1}{2n}\Big(\mathbf{C}^{\pi'}\mathbf{w} - \mathbf{r}^\pi\Big)^\top\mathbf{D}\Big(\mathbf{C}^{\pi''}\mathbf{w} - \mathbf{r}^\pi\Big)\Big),\\
\Rightarrow \hat{\mathbf{w}}^\pi_{\text{BRM}} &= \frac{1}{2}\Big(\hat{\mathbf{C}}^{\pi'\top}\mathbf{D}\hat{\mathbf{C}}^{\pi''}\Big)^{-1}\Big(\hat{\mathbf{C}}^{\pi'} + \hat{\mathbf{C}}^{\pi''}\Big)^\top\mathbf{D}\hat{\mathbf{r}}^\pi,\end{aligned} \tag{2.37}$$

where sample matrices formed from $\mathcal{D}_n$ can be defined as $\hat{\mathbf{C}}^{\pi'} = \hat{\boldsymbol{\Phi}} - \gamma\hat{\boldsymbol{\Phi}}'$, $\hat{\mathbf{C}}^{\pi''} = \hat{\boldsymbol{\Phi}} - \gamma\hat{\boldsymbol{\Phi}}''$, $\hat{\boldsymbol{\Phi}} = [\boldsymbol{\phi}(\mathbf{s}_1), ..., \boldsymbol{\phi}(\mathbf{s}_n)]^\top$, sample successor design matrices (because the transition model is not known) as $\hat{\boldsymbol{\Phi}}' = [\boldsymbol{\phi}(\mathbf{s}_1'), ..., \boldsymbol{\phi}(\mathbf{s}_n')]^\top$, $\hat{\boldsymbol{\Phi}}'' = [\boldsymbol{\phi}(\mathbf{s}_1''), ..., \boldsymbol{\phi}(\mathbf{s}_n'')]^\top$ and the inverse is assumed to exist. Convergence results and error bounds exist for the sup-norm (Williams and Baird, 1993, Prop 3.1), quadratically weighted norms (Guestrin et al., 2001) and general user-defined state distributions (Geist et al., 2017) such that data can be collected off-policy.

Projected Bellman Residual (MSPBE Objective)

It is unfortunate that BRM is impractical as it represents the 'correct' error that must be minimised. An alternative is to minimise the *projected* Bellman residual (see fig. 2.3) which is equivalent to finding the fixed point $\hat{\mathbf{v}}_{\lambda=0}$ of the composition $\Pi_D T^\pi$ in the projected Bellman equation $\mathbf{v} = \Pi_D T^\pi \mathbf{v}$. The MSPBE objective is derived by taking the Bellman residual (2.33), and fixing the target weights so that they are not involved in the optimisation (Kolter and Ng, 2009),

$$\begin{aligned}\mathbf{u}^* &= \arg\min_{\mathbf{u}\in\mathbb{R}^d} \Big[||\boldsymbol{\Phi}\mathbf{u} - T^\pi \boldsymbol{\Phi}\mathbf{w}||_D^2 \Big], \\ &= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D} T^\pi \boldsymbol{\Phi}\mathbf{w}, \\ &= f(\mathbf{w}).\end{aligned} \tag{2.38}$$

Finally we seek the solution to $\boldsymbol{\Phi}\mathbf{w} = \boldsymbol{\Phi} f(\mathbf{w})$ which is the fixed point and is equivalent to finding the minimiser of the mean squared projected Bellman error (MSPBE),

$$\begin{aligned}\mathcal{L}_{\text{MSPBE}}(\mathbf{w}) &:= ||\Pi_D T^\pi \boldsymbol{\Phi}\mathbf{w} - \boldsymbol{\Phi}\mathbf{w}||_D^2, \\ &= ||\Pi_D (T^\pi \mathbf{v} - \mathbf{v})||_D^2,\end{aligned}$$

where the second line exploits the idempotent projection $\Pi_D \boldsymbol{\Phi}\mathbf{w} = \boldsymbol{\Phi}\mathbf{w}$. The MSPBE only considers the error as represented by features $\boldsymbol{\phi}$ and does not consider the orthogonal component measuring distance to the actual value function targets. However this is the key advantage to minimising the MSPBE because the projected distance $||\Pi_D \Big(T^\pi \mathbf{v} - \mathbf{v} \Big)||_D$ can be made zero (if features are linearly independent). This leads to the following manipulation (see e.g. Geramifard et al. (2013)) which is known as the least squares temporal difference or the LSTD(0) (Bradtke and Barto, 1996) solution,

$$\begin{aligned}\mathbf{0} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{r}^\pi + \gamma \boldsymbol{\Phi}'\mathbf{w}) - \boldsymbol{\Phi}\mathbf{w}, \\ &= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{r}^\pi + \gamma \boldsymbol{\Phi}'\mathbf{w}) - \mathbf{w}, \\ &= \boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{r}^\pi + \gamma \boldsymbol{\Phi}'\mathbf{w}) - \boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi}\mathbf{w}, \\ &= \boldsymbol{\Phi}^\top \mathbf{D}(\gamma \boldsymbol{\Phi}' - \boldsymbol{\Phi})\mathbf{w} + \boldsymbol{\Phi}^\top \mathbf{D}\mathbf{r}^\pi, \\ \implies \mathbf{w}^\pi_{\lambda=0} &= \Big(- \boldsymbol{\Phi}^\top \mathbf{D}(\gamma \boldsymbol{\Phi}' - \boldsymbol{\Phi}) \Big)^{-1} \boldsymbol{\Phi}^\top \mathbf{D}\mathbf{r}^\pi, \qquad (2.39) \\ &= (-\mathbf{A})^{-1}\mathbf{b}, \qquad (2.40) \\ &= \Big(- \mathbb{E}_{S\sim D}\mathbb{E}_{S'|S,\pi(S)} \Big[\boldsymbol{\phi}(S) \Big(\gamma \boldsymbol{\phi}(S') - \boldsymbol{\phi}(S) \Big)^\top \Big] \Big)^{-1} \mathbb{E}_{S\sim D}\mathbb{E}_{R|S,\pi(S)} \Big[\phi(S) R \Big]\end{aligned}$$

where sample estimates are $\hat{\mathbf{A}} = \hat{\boldsymbol{\Phi}}^\top \mathbf{D}(\gamma \hat{\boldsymbol{\Phi}}' - \hat{\boldsymbol{\Phi}})$ and $\hat{\mathbf{b}} = \hat{\boldsymbol{\Phi}}^\top \mathbf{D}\hat{\mathbf{r}}^\pi$.

The model-free sample estimates of $\mathbf{A}$ and $\mathbf{b}$ are formed from trajectory data $\mathcal{D} := \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')_i\}_{i=1}^n$ whose behavioural policy determines the state distribution. LSTD minimisation of the MSPBE using trajectory data avoids the double-sampling

problem of BRM. Incremental approximations to $\mathbf{A}$ and $\mathbf{b}$ can also be updated online,

$$\begin{aligned}\hat{\mathbf{A}}_{1:n} &= \sum_{t=1}^{n} D(\mathbf{s}_t)\boldsymbol{\phi}(\mathbf{s}_t)\Big(\gamma\boldsymbol{\phi}(\mathbf{s}'_t) - \boldsymbol{\phi}(\mathbf{s}_t)\Big)^\top, \\ &= \hat{\mathbf{A}}_{1:n-1} + D(\mathbf{s}_n)\boldsymbol{\phi}(\mathbf{s}_n)\Big(\gamma\boldsymbol{\phi}(\mathbf{s}'_n) - \boldsymbol{\phi}(\mathbf{s}_n)\Big)^\top, \\ \hat{\mathbf{b}}_{1:n} &= \sum_{t=1}^{n} D(\mathbf{s}_t)\boldsymbol{\phi}(\mathbf{s}_t)r_t, \\ &= \hat{\mathbf{b}}_{1:n-1} + D(\mathbf{s}_n)\boldsymbol{\phi}(\mathbf{s}_n)r_n,\end{aligned}$$

which converge to $\mathbf{A}$ and $\mathbf{b}$ as $n \to \infty$ (Nedic and Bertsekas, 2003, section 5). Each update incurs $\mathcal{O}(d^3)$ complexity but this can be reduced to $\mathcal{O}(d^2)$ using the Sherman-Morrison formula to efficiently make incremental updates to the inverse calculation in equation (2.40). Such least squares solutions do not require the existence and tuning of learning rates as featured in stochastic gradient methods. LSTD also has the advantage in that $D(\cdot)$ does *not* have to be the on-policy distribution D^π (Bradtke and Barto, 1996, section 9), therefore trajectory data can be gathered off-policy.

LSTD Link to Linear TD

Sutton et al. (2009) and Dann et al. (2014, p 818) show that minimising the MSPBE is still equivalent to minimising the MSBE, but under a norm weighted by the metric $\mathbf{U} = \mathbf{D}\boldsymbol{\Phi}\Big(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi}\Big)^{-1}\boldsymbol{\Phi}^\top\mathbf{D}$ (which is positive definite under the assumption of independent features),

$$\begin{aligned}\mathcal{L}_{\text{MSPBE}}(\mathbf{v}) &= ||T^\pi\mathbf{v} - \mathbf{v}||^2_{\mathbf{U}}, \\ &= ||\boldsymbol{\Phi}^\top\mathbf{D}\Big(T^\pi\mathbf{v} - \mathbf{v}\Big)||^2_{(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}}.\end{aligned}$$

Under the same assumption that the columns of $\boldsymbol{\Phi}$ are linearly independent, then the solution to the MSPBE (with fixed point $\hat{\mathbf{v}}_{\lambda=0}$) is reached iff (Dann et al., 2014, p 819)

$$\mathbf{0} = \boldsymbol{\Phi}^\top\mathbf{D}\Big(T^\pi\mathbf{v} - \mathbf{v}\Big) = \mathbf{A}\mathbf{w} + \mathbf{b}, \tag{2.41}$$

$$= \mathbb{E}_{S\sim D}\mathbb{E}_{R,S'|S,\pi(S)}\Big[\boldsymbol{\phi}(S)\delta(S,R,S')\Big]. \tag{2.42}$$

Further analysis on the convergence of LSTD($\lambda = 0$) and comparisons to BRM can be found in Schoknecht (2003) and Scherrer (2010).

In addition to least squares methods, several fixed-point stochastic gradient descent approaches exist that attempt to minimise the MSPBE. The celebrated linear TD(λ=0) algorithm (Sutton, 1988)[4] makes stochastic gradient updates to the fixed weight Bellman residual (2.38). At each t^{th} update, the target weights are set to the current function approximator weights $\mathbf{w}_t$ i.e $\tilde{v}(\mathbf{s}) = T^\pi\boldsymbol{\phi}^\top(\mathbf{s})\mathbf{w}_t$. The gradient therefore ignores

[4]Also known as TD *learning* (TDL) to distinguish from other TD gradient methods.

$\mathbf{w}_t$ associated with the transition dynamics, characterising the approach as a *semi-gradient* method, derived from the mean squared temporal difference learning (TDL) error,

$$\begin{aligned}
\mathcal{L}_{\text{TDL}}(\mathbf{w}) &:= ||T^\pi \boldsymbol{\Phi}\mathbf{w}_t - \boldsymbol{\Phi}\mathbf{w}||_D^2, \\
&= \mathbb{E}_{S\sim D}\Big[\Big(\mathbb{E}_{R,S'|S,\pi(S)}\Big[\gamma\boldsymbol{\phi}^\top(S')\mathbf{w}_t - \boldsymbol{\phi}^\top(S)\mathbf{w} + R\Big]\Big)^2\Big], \\
\implies \nabla_{\mathbf{w}}\mathcal{L}_{\text{TDL}}(\mathbf{w}) &= -\mathbb{E}_{S\sim D}\mathbb{E}_{R,S'|S,\pi(S)}\Big[\boldsymbol{\phi}(S)\Big(\gamma\boldsymbol{\phi}^\top(S')\mathbf{w}_t - \boldsymbol{\phi}^\top(S)\mathbf{w} + R\Big)\Big], \\
&= -\mathbb{E}_{S\sim D}\mathbb{E}_{R,S'|S,\pi(S)}\Big[\boldsymbol{\phi}(S)\delta(S,R,S')\Big]
\end{aligned}$$

The t^{th} SGD update uses the sample gradient formed at the t^{th} transition $(\mathbf{s},\mathbf{a},r,\mathbf{s}')_t$,

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t\nabla_{\mathbf{w}}\hat{\mathcal{L}}_{\text{TDL}}(\mathbf{w})|_{\mathbf{w}_t,(\mathbf{s},\mathbf{a},r,\mathbf{s}')_t}, \\
&= \mathbf{w}_t + \eta_t\boldsymbol{\phi}(\mathbf{s}_t)\delta_t(\mathbf{s}_t,r_t,\mathbf{s}'_t), \\
&= \Big(\mathbf{I}_d + \eta_t\hat{\mathbf{A}}_t\Big)\mathbf{w}_t + \eta_t\hat{\mathbf{b}}_t,
\end{aligned} \tag{2.43}$$

with TD error $\delta_t(\mathbf{s}_t,r_t,\mathbf{s}'_t) := \Big(\gamma\boldsymbol{\phi}(\mathbf{s}'_t) - \boldsymbol{\phi}(\mathbf{s}_t)\Big)^\top\mathbf{w}_t + r_t$, $\hat{\mathbf{A}}_t := \boldsymbol{\phi}(\mathbf{s}_t)\Big(\gamma\boldsymbol{\phi}(\mathbf{s}'_t) - \boldsymbol{\phi}(\mathbf{s}_t)\Big)^\top$ and $\hat{\mathbf{b}}_t := \boldsymbol{\phi}(\mathbf{s}_t)r_t$. More efficient use of data sees all samples experienced being accumulated into an experience replay (Lin, 1992) memory $\mathcal{D} := \{(\mathbf{s},\mathbf{a},r,\mathbf{s}')_i\}_{i=1}^n$ such that at time t the sample gradient is formed from a minibatch $\mathcal{B}\sim\mathcal{D}_n$. This is demonstrated in the DQN algorithm (albeit for the optimal action-value function) where gradient updates are continuously made by drawing mini-batches uniformly from replay memory until function approximator improvements stop.

The stochastic update (2.43) re-written with actual $\mathbf{A}$ and $\mathbf{b}$ (see equation (2.40)) becomes the deterministic update,

$$\mathbf{w}_{t+1} = \Big(\mathbf{I}_d + \eta_t\mathbf{A}\Big)\mathbf{w}_t + \eta_t\mathbf{b}, \tag{2.44}$$

whose convergence determines whether the stochastic TDL converges. Schoknecht (2003) shows that if η_t obeys the Robbins and Monro (1951) decay conditions (see section B.71), then equation (2.44) converges if all eigenvalues of matrix $\mathbf{A}$ have a negative real part and diverges for positive eigenvalues. $\mathbf{A}$ is negative definite (hence convergence is guaranteed) if the state distribution is chosen as the stationary distribution $D = D^\pi$ (definition 7) of the Markov chain (Bertsekas and Tsitsiklis, 1996, Chapter 6). Exploitation of the stationary distribution (also known as *on-policy* learning) is at the heart of TDL convergence proofs (Bertsekas, 2012, Chapter 6). At convergence $\mathbf{w}_{t+1} = \mathbf{w}_t = \mathbf{w}_{\lambda=0}$ is the fixed point such that equation (2.44) becomes

$$\mathbf{0} = \mathbf{A}\mathbf{w}_{\lambda=0} + \mathbf{b},$$

thus solidifying TDL's link to the LSTD solution (2.40) and in finding the fixed point that minimises the MSPBE.

Semi-gradient methods solve an objective whose targets are constantly changing for each update because the targets are formed from snapshots of the function approximator which is known as *bootstrapping*. This occurs because the TD($\lambda = 0$) targets use only the one step Bellman operator T^π. By considering the exponentially λ-weighted Bellman operator T^λ (see equation (2.19)) then the *generalised* projected Bellman equation is $\hat{\mathbf{v}}_\lambda = \Pi_D T^\lambda \hat{\mathbf{v}}_\lambda$ and the MSPBE is

$$\mathcal{L}_{\text{MSPBE}}(\mathbf{w}) := ||\Pi_D T^\lambda \boldsymbol{\Phi}\mathbf{w} - \boldsymbol{\Phi}\mathbf{w}||_D^2,$$

where $\hat{\mathbf{v}}_\lambda = \boldsymbol{\Phi}\mathbf{w}_\lambda$ is the TD(λ) fixed point of the composite operator $\Pi_D T^\lambda$. Van Roy (1998, section 4.7); Tsitsiklis and Van Roy (1997) derive the celebrated convergence results for the TD(λ) algorithm again by relying heavily on setting $D = D^\pi$ as the *on-policy* stationary state distribution. They show that the *composition* $\Pi_{D^\pi} T^\lambda$ is a contraction w.r.t. the weighted norm $||\cdot||_{D^\pi}$. The final result (Tsitsiklis and Van Roy, 1997) is the asymptotic error between the fixed point $\hat{\mathbf{v}}_\lambda$ and the true value function $\mathbf{v}^\pi$ is upper-bounded (as a function of λ) by

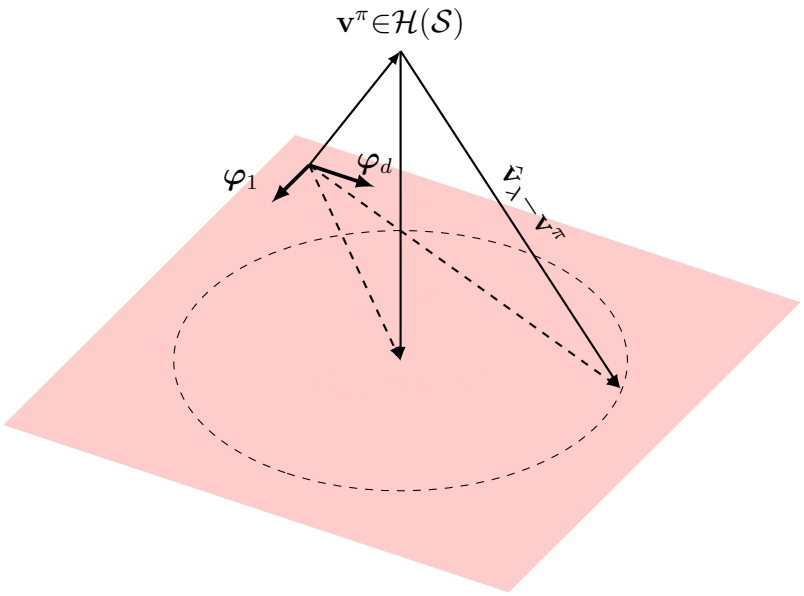
$$\begin{aligned} ||\hat{\mathbf{v}}_\lambda - \mathbf{v}^\pi||_{D^\pi} &\leq \frac{1}{\sqrt{1-\gamma_\lambda}} ||\Pi_{D^\pi}\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi}, \\ \Rightarrow ||\hat{\mathbf{v}}_\lambda - \mathbf{v}^\pi||_{D^\pi}^2 &\leq \frac{1-\lambda\gamma}{1-\gamma} ||\mathbf{v}^\pi_{\text{MSE}} - \mathbf{v}^\pi||_{D^\pi}^2, \end{aligned} \tag{2.45}$$

where $\gamma_\lambda = \gamma(1-\lambda)/(1-\lambda\gamma)$ is the effective contraction rate, such that $\gamma_\lambda = \gamma$ for TD(0) and $\gamma_\lambda = 0$ for TD(1). This bound is illustrated in fig. 2.4 and is a function of the projection distance $||\Pi_{D^\pi}\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi}$ where $\Pi_{D^\pi}\mathbf{v}^\pi$ is the solution to the MSE (see section 2.5.1). As $\lambda \to 0$, the TD target (and therefore function updates) acquires more bias, thus the upper bound increases to a maximum of $1/(1-\gamma)$ multiple of the projection distance. If $\lambda = 1$ then the targets are equivalent to Monte Carlo updates with no bias thus the error is upper-bounded solely by the projection distance. In the same way that TD(0) and LSTD(0) are related to the fixed point solution and convergence results, the TD(λ) is related to the generalised LSTD(λ) (Boyan, 1998) algorithm. Practical implementation of $\lambda > 0$ algorithms require the use of eligibility traces but this is out of the scope of this investigation. Other TD algorithms exist that perform SGD on alternative objectives such as gradient temporal difference (GTD) (Sutton et al., 2008b), linear (Sutton et al., 2009) and non-linear (Bhatnagar et al., 2009) GTD2/gradient correction (TDC), but are also outside the scope of this investigation.

Value Prediction Stability and Motivation for the CME Approach

It is possible to motivate the development of CME-based algorithms by highlighting the difficulties model-free approaches incur w.r.t. the 'deadly triad' characteristics

Figure 2.4 Illustrating the upper bound of $||\hat{\mathbf{v}}_\lambda - \mathbf{v}^\pi||_{D^\pi}$ varying with bootstrapping.



(definition 1). All three characteristics are desirable; function approximation for large state spaces, off-policy learning for efficient data reuse and bootstrapping for efficient updates. But as just reviewed, linear function approximation convergence relies on on-policy learning and bootstrapping clearly varies the convergence of TD(λ) algorithms.

Early examples supporting the 'deadly triad' hypothesis include Baird (1995, table 1) which demonstrate value-prediction instability on an embarrassingly small MDP. Divergence occurs when off-policy data is used and convergence is restored once an on-policy distribution is used (Sutton and Barto, 1998, p 217). Tsitsiklis and Van Roy (1996) also demonstrate divergence for value-prediction with bootstrapped least-squares targets. Tsitsiklis and Van Roy (1997) show that function approximation w.r.t. a quadratic norm may be expansive in the sup-norm of the Bellman operator (Gordon, 1995), where contractions can be restored if the function is fit with the quadratic norm weighted by the on-policy distribution. Bertsekas and Tsitsiklis (1996) discuss the even greater challenge of dealing with instabilities when using non-linear function approximators.

It will become obvious that a CME-based value prediction does not suffer from the deadly triad as approximate MDPs are built from off-policy data, offer full backup bootstrapping and by their very definition define value function approximation. Part of the reason for this is that a CME does make a non-parametric value function approximation rather than parametric. Alternatively model-free non-parametric value prediction algorithms have been developed with kernel and GP function approximation, rendering explicit feature generation obsolete. GP temporal difference (GPTD) (Engel

et al., 2003) model a distribution of functions to extract a posterior value function distribution conditioned on observations. Xu et al. (2005) introduce Kernel LSTD (KLSTD) and Taylor and Parr (2009) unify the approach by showing equivalence between GPTD and KLSTD.

Approximate Action-Value Prediction

All approximate state-value prediction and convergence results in section 2.5.2 are directly applicable to approximate action-value prediction. A state action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ has a linear approximation which takes the form $q(\mathbf{s}, \mathbf{a}) = \langle \boldsymbol{\psi}(\mathbf{s}, \mathbf{a}), \mathbf{w} \rangle_{\mathcal{F}_\psi}$, where explicit feature representation over states and actions $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_\psi$ is required, either given to the agent a priori or learnt. By applying the Bellman operator to q then MSBE/MSPBE objectives are readily formed and can be solved in the same way as for state-value functions. For the projected Bellman equation approach, linear SARSA(λ) is directly comparable to TD(λ) and linear LSTDQ(λ) is an extension to linear LSTD(λ). When API uses LSTDQ in its q-estimation stage the algorithm is known as LSPI (Lagoudakis and Parr, 2003). As with linear TD(λ), linear SARSA(λ) is on-policy and inherits similar convergence conditions and sensitivity to the deadly triad. Kernel-based methods have also been developed for SARSA (Engel et al., 2005) and SARSA(λ) (Robards et al., 2011).

2.5.3 Approximate Policy Improvement

Recalling section 2.3.3, *exact* policy iteration is guaranteed to find optimal policies in discrete known MDPs. At the k^{th} policy iteration; i) exact action-value function q^{π_k} evaluation is carried out for the k^{th} policy (known as policy evaluation) and ii) a new improved policy π_{k+1} is found by acting greedily against q^{π_k} (known as policy improvement). *Approximate* policy iteration (API) (Bertsekas and Tsitsiklis, 1996, Chapter 6) replaces the first step of exact policy evaluation with approximate value-prediction to estimate an action-value function (or as an intermediate step, the state-value function first). The policy improvement stage therefore acts greedily on the approximate action-value function,

$$\pi_{k+1}(\mathbf{s}) = \arg\sup_{a \in \mathcal{A}} \left[\hat{q}^{\pi_k}(\mathbf{s}, \mathbf{a}) \right], \quad \mathbf{s} \in \mathcal{S}, \tag{2.46}$$

producing a policy whose performance has suboptimal improvement bounds as stated in lemma 2.

Lemma 2 (Greedy policy suboptimality is bounded (Singh and Yee, 1994))**.** *For any approximate action-value function $\hat{q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, actual optimal action-value function q^*, then if $||q^* - \hat{q}||_\infty \leq \epsilon$, the suboptimality of acting greedily $\hat{\pi} = \arg\sup_{\mathbf{a} \in \mathcal{A}} \left[\hat{q}(\mathbf{s}, \mathbf{a}) \right]$ is*

bounded,

$$||v^* - v^{\pi_q}||_\infty \leq \frac{2}{1-\gamma}||q^* - \hat{q}||_\infty, \tag{2.47}$$

where $v^*(\mathbf{s}) = q^*(\mathbf{s}, \pi^*(\mathbf{s}))$.

This result limits how suboptimal a greedy policy can be when acting against an existing approximate value function. The closer $\hat{q}$ is to the optimal q^*, the less suboptimal the greedy policy will likely be. A similar bound in terms of just state-value functions is found in (Bertsekas, 2012, Section 6.1.1). Although the sup-norm is seldom useful for practical settings, other bounds expressed with more suitable weighted quadratic norms have been developed and will be summarised below.

The convergence result for TD(λ) (2.45) is also directly applicable to SARSA(λ) q-function approximation. Intuition suggests that it is preferable to not bootstrap by selecting $\lambda = 1$, however Sutton and Barto (1998, section 8.6) provide experimental results that compare the empirical performance of linear SARSA control policies vs. λ suggest otherwise; if a *moderate* amount of bootstrapping is used, then not only is the learnt policy's performance superior, but also convergence to fixed points $\hat{q}_\lambda$ is quicker. This suggests that for approximate control, an MSE-type upper bound as in equation (2.45) is not sufficient to gauge algorithm performance and more work is required to understand the deeper mechanism at work.

Policy Improvement Stability and Chatter

Unfortunately lemma 2 specifies greedy policy suboptimality in terms of sup-norms which are not practical in MDPs with large or continuous state space. Quadratic norms weighted by a distribution over states which suits function approximation schemes are more desirable. Munos (2003, 2005) provides such suboptimality bounds. During this analysis, Munos (2003, Section 1) identifies a phenomenon called *policy chatter* or *policy oscillation* (Wagner, 2011) which can induce a prolonged sequence of very suboptimal improvements, eventually undoing the policy learning process. Given an approximation $\hat{v}_k$ of v^{π_k} during the k^{th} iteration of API, the algorithm proceeds rapidly at first because the approximation error $||\hat{v}_k - v^{\pi_k}||$ is small relative to the error from the optimal policy $||v^{\pi_k} - v^*||$. During this phase of learning, if $\hat{v}_k$ is good, then the greedy improvement π_{k+1} is good i.e. $\pi_{k+1} \geq \pi_k$. However this improvement degrades as the distance to optimality reduces and the policy optimisation process enters a stationary phase. This is because the state-value function approximation error now dominates, preventing efficiency in the policy improvement step. This inhibits any guaranteed convergence for API and the performance of successive improved policies can oscillate wildly. Such oscillations have also been attributed to overshooting

(Wagner, 2014). A deeper understanding has yet to materialise and conservative policy updates is one way of mitigating oscillations in order to smooth out policy learning.

Conservative Policy Iteration Scherrer (2014) introduces the following notation to generalise greedy policy improvements in API algorithms. Given a state distribution D, approximate value function $\hat{v}_k$ and a small error ϵ_k then the greedy operator $\mathcal{G}_\epsilon$ returns a policy π_{k+1} that is (ϵ,D)-approximately greedy w.r.t. $\hat{v}_k$ which obeys

$$||T^*\hat{v}_k - T^{\pi_{k+1}}\hat{v}_k||_D \leq \epsilon_{k+1}.$$

The vanilla greedy update in equation (2.46) is therefore represented by

$$\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(D, \hat{v}_k),$$

where the state distribution is equally weighted $\mathbf{D} = \mathbf{I}_{|\mathcal{S}|}$ and $\epsilon_{k+1} = 0$. Kakade and Langford (2002) introduce conservative policy iteration (CPI) which uses a smoother greedy update,

$$\pi_{k+1} \leftarrow (1 - \alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}_{\epsilon_{k+1}}(d_{\pi,\nu}, \hat{v}_k),$$

where $d_{\pi,\nu}$ is the discounted cumulative occupancy measure for an initial state distribution ν. The motivation is that gradual updates may mitigate policy oscillation. A related approach is a 'softer' greedy policy improvement (Wagner, 2013, equation 4) that makes similar α_{k+1}-weighted updates on the policy parameters.

Policy Improvement Stability and Motivation for the CME Approach

In addition to value prediction stability, the CME approach theoretically enjoys policy improvement sub-optimality bounds. These bounds are a function of the distance between the finite approximate MDP and the actual MDP. Previously, the stability of such an algorithm has not been explored experimentally. Does the CME approach suffer from the same policy oscillations as conventional API when the policy approximation error becomes large with respect to the error from the optimal policy? It is predicted that the CME approach does not suffer from this ailment because i) CME approximate MDPs are guaranteed to converge to the real MDP as more data is added to the training set and therefore ii) policy suboptimality as a function of the error between the approximate and real MDP will always reduced as more data is collected.

2.6 Model-based Approximate Policy Iteration

This investigation is concerned with avoiding the pitfalls of API w.r.t. value prediction convergence, the deadly triad and policy improvement oscillations. Simply pursuing

a Dyna architecture (Sutton, 1991) would betray this goal because the underlying approximate value prediction and policy improvement algorithms remain unchanged. Using an already precarious API algorithm with data generated from inaccurate models risks instigating a *deadly approximate cauldron* which may make policy learning stability unreachable. The consumption of simulated data from imperfect models is poorly understood but some theoretical work exists for linear TD with linear model-based planning (Sutton et al., 2008a).

The weakness of conventional API algorithms lies in searching for value functions as *approximate* solutions to the real MDP (as defined by the Bellman optimality equation T^* with unknown transition expectation). An alternative approach is to sample data from the real MDP, build an approximate finite MDP (with approximate dynamics[5] $\hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[v] \approx \mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[v(S')]$ which defines the approximate Bellman optimality operator $\hat{T}^*$) and solve it *exactly*. In this way the transition model is an integral component to the value function estimate itself. Common approaches to estimating conditional expectations require intermediate steps of density estimation and numerical integration. However the following methods are based on estimating the operator $v \mapsto \mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[v(S')]$ as a regression function (section B.4.1).

2.6.1 Hilbert Space Embeddings of Conditional Expectations

Recall section 2.5.1 where value function approximations are assumed to reside in a Hilbert space $v \in \mathcal{F}$ whose evaluations are modelled as $v(\mathbf{s}) = \langle v, \boldsymbol{\phi}(\mathbf{s}) \rangle_{\mathcal{F}}$ and where $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}$ is a feature mapping. Recall also section 2.5.2 where the Bellman operator was applied to this linear function approximation scheme (2.28) such that the transition dynamics conditional expectation is written as,

$$\begin{aligned}
\mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[v(S')] &= \langle \mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[\phi(S')], v \rangle_{\mathcal{F}} \\
&= \langle \mu(\mathbf{s},\mathbf{a}), v \rangle_{\mathcal{F}} \\
&\approx \langle \hat{\mu}(\mathbf{s},\mathbf{a}), v \rangle_{\mathcal{F}} \\
&=: \hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[v],
\end{aligned}$$

where $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$ is the expected next feature map. Choice of $\mathcal{F}$ determines whether the approximate MDP is a finite MDP.

Conditional Mean Embeddings

The following is a summary of the non-parametric conditional mean embeddings (CME) (Song et al., 2009) framework which embeds conditional distributions into an

[5] For brevity the immediate reward function is assumed to be known $r(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\cdot|\mathbf{s},\mathbf{a}}[R]$, although methods presented here and those developed later in this thesis do support learning the $\hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[R]$.

Table 2.1 Notation inherited from Song et al. (2010)

Domain	$\mathcal{X}$	$\mathcal{Y}$
Random Variable	X	Y
Observation	$\mathbf{x}$	$\mathbf{y}$
RKHS Function	$f \in \mathcal{H}_K$	$g \in \mathcal{H}_L$
Feature Map	$\boldsymbol{\varphi}(\mathbf{x}) = K(\mathbf{x}, \cdot) \in \mathcal{H}_K$	$\boldsymbol{\phi}(\mathbf{y}) = L(\mathbf{y}, \cdot) \in \mathcal{H}_L$
Feature Matrix	$\boldsymbol{\Upsilon} := [\boldsymbol{\varphi}(\mathbf{x}_1), ..., \boldsymbol{\varphi}(\mathbf{x}_n)]^\top$	$\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{y}_1), ..., \boldsymbol{\phi}(\mathbf{y}_n)]^\top$
Kernel	$K(\mathbf{x}, \mathbf{x}')$	$L(\mathbf{y}, \mathbf{y}')$
Kernel Matrix	$(\mathbf{K} := \boldsymbol{\Upsilon}\boldsymbol{\Upsilon}^\top)_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$	$(\mathbf{L} := \boldsymbol{\Phi}\boldsymbol{\Phi}^\top)_{ij} = L(\mathbf{y}_i, \mathbf{y}_j)$

RKHS, providing a way to train conditional expectation models $\hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[v]$. Grünewälder et al. (2012a) formalise the training of a CME into a principled regularised regression problem using vector-valued RKHS theory (Grünewälder et al., 2012b). The resulting approximate Bellman operator $\hat{T}$ can be solved exactly in a policy iteration algorithm (section 2.3.3), formalised by the pseudo-MDP framework (Yao et al., 2014a) which gives deeper insights into the conditions for contraction mappings (definition 9).

Reproducing Kernel Hilbert Spaces Given a non-empty set $\mathcal{X}$, then a scalar-valued function $f : \mathcal{X} \to \mathbb{R}$ belongs to a reproducing kernel Hilbert space (RKHS) $\mathcal{H}_K$ (Shawe-Taylor and Cristianini, 2004) if its evaluation takes the form

$$f(\mathbf{x}) = \langle f, \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}_K}, \qquad \mathbf{x} \in \mathcal{X},\ \boldsymbol{\varphi}(\mathbf{x}) \in \mathcal{H}_K,$$

where $\boldsymbol{\varphi}(\mathbf{x})$ is a feature map (also written as $K(\mathbf{x}, \cdot)$) and the reproducing kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is defined as $K(\mathbf{x}, \mathbf{x}') := \langle \boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{x}') \rangle_{\mathcal{H}_K}$; see section B.4.2 for a formal review. The defining quality of an RKHS function is that its evaluation is bounded by its norm $||\cdot||_{\mathcal{H}_K}$. Intuitively this means that if two functions $f, g \in \mathcal{H}_K$ are close as measured by $||f - g||_{\mathcal{H}_K}$, then the distance between their point-wise evaluations $|f(\mathbf{x}) - g(\mathbf{x})|$ is *also* close. This is not the case for other Hilbert spaces of functions. The RKHS norm encodes a sense of *smoothness* such that smoother functions have lower norm (c.f. L2 regularisation for parametric function approximators). Typically $f = \sum_{j=1}^{n} \alpha_j \boldsymbol{\varphi}(\mathbf{x}_j, \cdot)$ in a regression setting (where $\alpha_j \in \mathbb{R}$) with training data $\mathcal{D}_n$ such that $f(\mathbf{x}) = \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \mathbf{x})$. The overwhelming advantage of an RKHS function is that the feature map $\boldsymbol{\varphi}(\mathbf{x}) \in \mathcal{H}_K$ is *implicit* and only a positive semi-definite kernel needs to be defined.

Embedding Conditional Distributions The following summarises Song et al. (2009) and uses notation as specified in table 2.1. The conditional distribution $P(Y|X = \mathbf{x})$ embedding is

$$\mu_{Y|X=\mathbf{x}} := \mathbb{E}_{Y|X=\mathbf{x}}[\boldsymbol{\phi}(Y)] = \int_{\mathcal{Y}} \boldsymbol{\phi}(\mathbf{y}) P(\mathrm{d}\mathbf{y}|\mathbf{x}) \quad \in \mathcal{H}_L, \tag{2.48}$$

such that

$$\mathbb{E}_{Y|X=\mathbf{x}}[g(Y)] = \langle g, \mu_{Y|X=\mathbf{x}} \rangle_{\mathcal{H}_L}, \qquad \forall g \in \mathcal{H}_L.$$

The embedding sweeps over a family of points in $\mathcal{H}_L$, each indexed by the value of the conditioning variable. Thus by fixing the embedding to a specific value $X = \mathbf{x}$, it becomes a single element in $\mathcal{H}_L$. We therefore seek an operator that maps between the two RKHSs $\mathcal{C}_{Y|X} : \mathcal{H}_K \to \mathcal{H}_L$, such that the conditional mean embedding is

$$\begin{aligned} \mu_{Y|X=\mathbf{x}} &= \mathcal{C}_{Y|X} \boldsymbol{\varphi}(\mathbf{x}), \\ &= \mathcal{C}_{YX} \mathcal{C}_{XX}^{-1} \boldsymbol{\varphi}(\mathbf{x}). \end{aligned}$$

The operators $\mathcal{C}_{XX} := \mathbb{E}_X[\boldsymbol{\varphi}(X) \otimes \boldsymbol{\varphi}(X)]$, $\mathcal{C}_{YX} := \mathbb{E}_{YX}[\boldsymbol{\varphi}(Y) \otimes \boldsymbol{\phi}(X)]$ are variance and cross covariances $\mathcal{C}_{XX} : \mathcal{H}_K \to \mathcal{H}_K$ and $\mathcal{C}_{YX} : \mathcal{H}_K \to \mathcal{H}_L$ respectively (drawing similarities with the primal regression setting in equation (B.14)). Given a dataset $\mathcal{D}_n = \{(\mathbf{x}, \mathbf{y})_i\}_{i=1}^n$ drawn i.i.d from $P(X, Y)$, then the sample estimates are used to compute the empirical conditional embedding (see notation in table 2.1),

$$\begin{aligned} \hat{\mu}_{Y|X=\mathbf{x}} &= \hat{\mathcal{C}}_{YX} \hat{\mathcal{C}}_{XX}^{-1} \boldsymbol{\varphi}(\mathbf{x}), \\ &= \boldsymbol{\Phi}^\top \boldsymbol{\Upsilon} (\boldsymbol{\Upsilon}^\top \boldsymbol{\Upsilon})^{-1} \boldsymbol{\varphi}(\mathbf{x}), \\ &= \boldsymbol{\Phi}^\top (\boldsymbol{\Upsilon} \boldsymbol{\Upsilon}^\top)^{-1} \boldsymbol{\Upsilon} \boldsymbol{\varphi}(\mathbf{x}), \\ &= \boldsymbol{\Phi}^\top (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \boldsymbol{\Upsilon} \boldsymbol{\varphi}(\mathbf{x}), \\ &= \boldsymbol{\Phi}^\top (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \boldsymbol{\psi}(\mathbf{x}), \qquad (2.49) \\ &= \boldsymbol{\Phi}^\top \mathbf{W} \boldsymbol{\psi}(\mathbf{x}), \qquad (2.50) \\ &= \boldsymbol{\Phi}^\top \boldsymbol{\alpha}(\mathbf{x}), \\ &= \sum_{j=1}^n \alpha_j(\mathbf{x}) \boldsymbol{\phi}(\mathbf{y}_j), \qquad (2.51) \end{aligned}$$

where the ridge term is added for numerical stability, $\boldsymbol{\psi}(\mathbf{x}) := [K(\mathbf{x}_1, \mathbf{x}), ..., K(\mathbf{x}_n, \mathbf{x})]^\top$ and $\boldsymbol{\alpha}(\mathbf{x}) := [\alpha_1(\mathbf{x}), ..., \alpha_n(\mathbf{x})]^\top = \mathbf{W} \boldsymbol{\psi}(\mathbf{x}) \in \mathbb{R}^n$. Finally the approximate expectation operator is estimated by

$$\begin{aligned} \hat{\mathscr{E}}_{(\mathbf{x})}[g] &= \langle g, \hat{\mu}_{Y|X=x} \rangle_{\mathcal{H}_L}, \\ &= \langle g, \sum_{j=1}^n \alpha_j(\mathbf{x}) \boldsymbol{\phi}(\mathbf{y}_j) \rangle_{\mathcal{H}_L}, \\ &= \sum_{j=1}^n \alpha_j(\mathbf{x}) g(\mathbf{y}_j). \qquad (2.52) \end{aligned}$$

Embeddings as Regressors The following summarises that learning a CME can be posed as a principled penalised empirical risk minimisation problem (Grünewälder et al., 2012b); see section B.4.6 for a summary on vector-value RKHS (vvRKHS)

theory. Given $Y \sim P(\cdot|X)$, then the aim is to describe a suitable ideal loss for $\mu \in \mathcal{H}_L$, the true embedding of $P(\cdot|X)$. Supervised targets $\mathbb{E}_{Y|X}[g]$ cannot be sampled and are not given from an oracle, therefore Grünewälder et al. (2012a, eqn 18) propose the following loss,

$$\begin{aligned}\mathcal{L}(\mu) &:= \mathbb{E}_{X\sim D}\Big[\Big(\mathbb{E}_{Y|X}[g] - \langle \mu(X), g\rangle_{\mathcal{H}_L}\Big)^2\Big], \\ &\leq \mathbb{E}_{X\sim D, \cdot Y\sim P(\cdot|\mathcal{X})}\Big[||\boldsymbol{\phi}(Y) - \mu(X)||^2_{\mathcal{H}_L}\Big], \end{aligned} \tag{2.53}$$

where $X \sim D$ for some distribution D. Equation (2.53) is a surrogate loss that is more appropriate for supervised learning. The embedding can now be considered as a vector-valued function $\mu : \mathcal{X} \to \mathcal{H}_L$ and will be modelled in a vvRKHS $\mu \in \mathcal{H}_{\boldsymbol{\Gamma}}$ associated with kernel $\boldsymbol{\Gamma}$. An estimate of μ can then be made by collecting training data $\mathcal{D} := \{(\mathbf{x}, \boldsymbol{\phi}(\mathbf{y}))_i\}_{i=1}^n$ and forming a vvRKHS penalised risk minimisation problem (c.f. equation (B.65)),

$$\hat{\mu} = \arg\min_{\mu\in\mathcal{H}_{\boldsymbol{\Gamma}}}\left[\frac{1}{2n}\sum_{i=1}^n ||\boldsymbol{\phi}(\mathbf{y}_i) - \mu(\mathbf{x}_i)||^2_{\mathcal{H}_L} + \lambda||\mu||^2_{\boldsymbol{\Gamma}}\right]. \tag{2.54}$$

By specifying the special case $\boldsymbol{\Gamma}(\mathbf{x}, \mathbf{x}') := K(\mathbf{x}, \mathbf{x}')\mathbf{I}$, where $\mathbf{I} : \mathcal{H}_L \to \mathcal{H}_L$ then the solution is the mean embedding (2.51) (see section B.4.6). This approach formally confirms that we can use the machinery of penalised risk minimisation and cross-validation in order to find the regularised solution $\hat{\mu}$.

Solving Approximate MDPs as Embeddings Grünewälder et al. (2012a) show that by assuming the value function resides in $\mathcal{F} = \mathcal{H}_L$, then the approximation $\mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[v(S')] \approx \hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[v]$ is modelled by embedding the conditional distribution in an RKHS. Recall embedding (2.52), then given $\mathcal{H}_K$ over $\mathcal{X} = \mathcal{S}\times\mathcal{A}$ and $\mathcal{H}_L$ over $\mathcal{Y} = \mathcal{S}$, the transition dynamics approximate expectation is

$$\begin{aligned}\hat{\mathscr{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] &= \langle v, \hat{\mu}_{S|\mathbf{s},\mathbf{a}}\rangle_{\mathcal{H}_L}, \\ &= \sum_{j=1}^n \alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a})v(\mathbf{s}'_j).\end{aligned}$$

The embedding $\mu : \mathcal{S}\times\mathcal{A} \to \mathcal{H}_L$ is estimated by solving (2.54) in a batch penalised vvRKHS regression scheme using transition data $\mathcal{D} := \{(\mathbf{s}, \mathbf{a}, \boldsymbol{\phi}(\mathbf{s}'))_i\}_{i=1}^n$, whose solution is $\boldsymbol{\alpha}^{\mathrm{CME}}(\mathbf{s},\mathbf{a}) = \mathbf{W}\boldsymbol{\psi}(\mathbf{s},\mathbf{a})$ and $\mathbf{W} = (\mathbf{K} + \lambda\mathbf{I}_n)^{-1}$. A cross validation scheme on kernel bandwidth and λ costs $\mathcal{O}(n^3)$ due to the kernel matrix inverse but the embedding evaluation is only linear in n.

Grünewälder et al. (2012a, algorithm 1) assume transition data $\mathcal{D}_n$ is provided *a priori* such that the learnt approximate expectation is used to form an approximate

optimal Bellman operator $\hat{T}^*_\mu : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$,

$$(\hat{T}^*_\mu v)(\mathbf{s}) := \sup_{\mathbf{a} \in \mathcal{A}} \Big[r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{j=1}^{n} \alpha_j^{\text{CME}}(\mathbf{s}, \mathbf{a}) v(\mathbf{s}'_j) \Big], \ \mathbf{s} \in \mathcal{S}, \ \mathbf{s}' \in \mathcal{S}'_n, \tag{2.55}$$

where $\mathcal{S}'_n := \{\mathbf{s}'_i\}_{i=1}^n$ are the successor states in training data. Applying the sup-norm convergence proof in lemma B.3 to $\hat{T}^*_\mu$ proves that a contraction exists if $\hat{\mathcal{E}}_{(\mathbf{s},\mathbf{a})}[v]$ is w.r.t a valid probability distribution[6] for all $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. The CME embedding is defined over the discrete set of successor states in the training data, therefore the value function only needs to be maintained over $\mathcal{S}'_n$. Adopting vector notation $\hat{\mathbf{v}} := [\hat{v}(\mathbf{s}'_1), ..., \hat{v}(\mathbf{s}'_n)]^\top$, then Grünewälder et al. (2012a, algorithm 3) execute exact value iteration (section 2.3.2) by iterating $\hat{\mathbf{v}}_{k+1} = \hat{T}^*_\mu \hat{\mathbf{v}}_k$. Convergence $\hat{\mathbf{v}}_k \to \hat{\mathbf{v}}^*_\mu$ is guaranteed if T^*_μ is a contraction mapping where $\hat{\mathbf{v}}^*_\mu$ is the optimal value function of the approximate MDP as defined by the approximate embedding.

Finally, a policy is extracted at iteration $k = \kappa$ by forming a state action-value function

$$\hat{q}_\kappa(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{j=1}^{n} \alpha_j^{\text{CME}}(\mathbf{s}, \mathbf{a}) \hat{v}_\kappa(\mathbf{s}'_j),$$

$$\hat{\pi}_\kappa(\mathbf{s}) := \arg\sup_{\mathbf{a} \in \mathcal{A}} [\hat{q}_\kappa(\mathbf{a}, \mathbf{a})].$$

Unlike kernel density estimates, the rate of embedding convergence as described by $\epsilon(\mathcal{D}_n)$ is independent to the dimensionality of the underlying space $\mathcal{X} = \mathcal{S} \times \mathcal{A}$ (Grünewälder et al., 2012a; Song et al., 2013).

Theoretical Claims for Stability The quality of the value function v^{π_κ} in the approximate MDP (defined by $\hat{\mu}$) w.r.t. the actual optimal value function is bounded as follows.

Theorem 4 (CME Policy Suboptimality Bound (Grünewälder et al., 2012a, Theorem 3.2))**.** *Given κ applications of an approximate optimal Bellman operator $\hat{T}^*_\mu$ which consists of an embedding $\hat{\mu}$ trained on $\mathcal{D}_n$, then the quality of $v^{\hat{\pi}_\kappa}$ is upper-bounded by,*

$$||v^{\hat{\pi}_\kappa} - v^*||_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \Big(\epsilon(\kappa) + 2||v^* - \tilde{v}^*||_\infty + \epsilon(\hat{\mu}) ||\tilde{v}^*||_{\mathcal{H}_L} \Big)$$

where $\epsilon(\kappa) := \gamma^\kappa ||\hat{v}_1 - \hat{v}_0||_\infty$, $\epsilon(\hat{\mu}) := \sup_{s,a \in \mathcal{S} \times \mathcal{A}} ||\mu(\mathbf{s}, \mathbf{a}) - \hat{\mu}(\mathbf{s}, \mathbf{a})||_{\mathcal{H}_L}$, $\hat{v}_0$ is an initial guess and $\tilde{v}^* \in \mathcal{H}_L$ is an arbitrary optimal value function approximation.

The term $\epsilon(\kappa)$ derives from the error $||\hat{v}_\kappa - \hat{v}^*||_\infty$ where $\hat{v}^*$ is the actual optimal value function in the approximate MDP, which can be sent to zero by increasing κ. The second term describes the distance between the actual optimal value function and

[6] Meaning that $\sum_j \alpha_j^{\text{CME}}(\mathbf{s}, \mathbf{a}) = 1$, however this condition is relaxed in the Pseudo-MDP architecture and will be discussed in the next section.

an arbitrary approximation $\tilde{v}^* \in \mathcal{H}_L$. This term captures how well the actual value function v^* can be modelled by a function in $\mathcal{H}_L$, therefore increasing the richness of $\mathcal{H}_L$ will drive this error towards zero. Finally the term $\epsilon(\hat{\mu})$ describes the sup-norm quality of the approximate MDP $\hat{\mu}(\mathbf{s}, \mathbf{a}) \in \mathcal{H}_L$ w.r.t. to the population embedding $\mu \in \mathcal{H}_L$, measured in $\mathcal{H}_L$ and weighted by the size of an arbitrary value function approximation $\tilde{v} \in \mathcal{H}_L$.

Smooth RKHS functions have smaller norms and therefore if $\tilde{v}$ is non-smooth, this final error term is reduced by increasing the quality of the embedding $\hat{\mu}$ by increasing the number of training samples $\mathcal{D}_n$. Grünewälder et al. (2012a, lemma 2.2) show that under certain conditions then in the limit of large data, $\epsilon(\kappa) \to 0$ and $\epsilon(\hat{\mu}) \to 0$. Theoretically this suggests that value function approximation can be improved by the quality of the embedding (by training on more data). CMEs therefore offer a mechanism to mitigate policy improvement oscillations by improving the value estimates when in proximity to the optimal value function

Therefore CME policy improvements may exhibit more stability when value functions are close to the optimal value function by simply training the embedding on more data, rendering it immune to the policy oscillations seen in parametric API as reviewed in section 2.5.3. By improving the quality of the embedding, then relatively large value estimate errors will be thwarted and will be empirically tested in the upcoming investigation.

Pseudo MDPs

As discussed with reference to definition 2, the actual optimal Bellman operator (definition 12) is defined in part by the probability kernel $P : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \to [0, 1]$ that assigns the probability measure (i.e. distribution) $P(\cdot|\mathbf{s}, \mathbf{a}) : \mathcal{S} \to [0, 1]$ to $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$, which in turn assigns probability $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ for the transition to $\mathbf{s}' \in \mathcal{S}$. For continuous state spaces, $\mathcal{S}$ is a measurable space with well-defined expectations over measurable subsets of states (see Yao et al. (2014a) for details). The approximate optimal Bellman operator (2.55) is only valid if the sup-norm contraction is conserved. Given that the value function only needs to be represented over the successor states $\mathcal{S}'_n$ in the training data, then a contraction is clearly maintained if $\alpha_j(\mathbf{s}', \mathbf{a}) \geq 0$ and $\sum_{j=1}^{n} \alpha_j(\mathbf{s}', \mathbf{a}) = 1$ for all j and every $(\mathbf{s}', \mathbf{a}) \in \mathcal{S}'_n \times \mathcal{A}$. $\hat{\mathcal{E}}^{\hat{\mu}}_{(\mathbf{s},\mathbf{a})}[v]$ is therefore an expectation w.r.t. a valid probability distribution $P(\cdot|\mathbf{s}, \mathbf{a})$,

$$P(\mathbf{s}'_j|\mathbf{s}, \mathbf{a}) = \begin{cases} \alpha_j(\mathbf{s}, \mathbf{a}), & \mathbf{s}'_j \in \mathcal{S}'_n, \\ 0, & \text{otherwise.} \end{cases} \tag{2.56}$$

Yao et al. (2014a) develop a *pseudo*-MDP abstraction framework which unifies the description approximate MDPs like the CME. Most importantly a pseudo-MDP

relaxes the contraction constraint such that $\hat{T}^*_\mu$ does not have to be made w.r.t. a probability measure $P(\cdot|\mathbf{s},\mathbf{a})$. The following result is stated in their work, but the simple contraction proof is provided here.

Lemma 3. *The approximate optimal Bellman operator $\hat{T}^*_\mu : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ formed by the pseudo-MDP is a contraction mapping if* $\sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} ||\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})||_1 \leq 1$.
Proof: The contraction argument lemma B.3 is applied to approximate dynamics,

$$\begin{aligned}
||T^*u - T^*v||_\infty &\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} \Big[\Big| \sum_{j=1}^{n} \alpha_j(\mathbf{s},\mathbf{a})(u(\mathbf{s}'_j) - v(\mathbf{s}'_j)) \Big| \Big] \\
&\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} \Big[\sum_{j=1}^{n} |\alpha_j(\mathbf{s},\mathbf{a})| \, |u(\mathbf{s}'_j) - v(\mathbf{s}'_j)| \Big] \\
&\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} \Big[\sum_{j=1}^{n} |\alpha_j(\mathbf{s},\mathbf{a})| \, ||u - v||_\infty \Big] \\
&= \gamma ||u - v||_\infty \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} \Big[||\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})||_1 \Big],
\end{aligned}$$

then given $\gamma \in [0,1)$, to preserve the contraction requires $\sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}} \Big[||\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})||_1 \Big] \leq 1$. □

Clearly α_j values no longer assign transition probabilities and instead they assign a weight which can be negative. A pseudo-MDP is defined as $\mathcal{N} := \{\mathcal{S}, \mathcal{A}, \hat{P}, P_1, r, \gamma\}$ where $\hat{P}(\cdot|\mathbf{s},\mathbf{a})$ does not necessarily have to be a probability distribution, elaborated as follows. Loosely following Yao et al. (2014a, Part III) then if $\mathbf{s}'_j \in \mathcal{S}'_n$, the pseudo-MDP produced by the CME method takes the form

$$\begin{aligned}
\hat{P}(\mathrm{d}\mathbf{s}'|\mathbf{s},\mathbf{a}) &:= \boldsymbol{\xi}^\top(\mathrm{d}\mathbf{s}')\boldsymbol{\psi}(\mathbf{s},\mathbf{a}), \\
&= \Big[m(\mathrm{d}\mathbf{s}') f(\mathbf{s}') \Big] \boldsymbol{\psi}(\mathbf{s},\mathbf{a}), \\
&= \Big[\Big(\sum_{j=1}^{n} \delta_{\mathbf{s}'_j}(\mathrm{d}\mathbf{s}') \Big) (\mathbf{1}_{\mathbf{s}'=\mathbf{s}'_j} \mathbf{w}_{j:}) \Big] \boldsymbol{\psi}(\mathbf{s},\mathbf{a}), \\
&= \sum_{j=1}^{n} \delta_{\mathbf{s}'_j}(\mathrm{d}\mathbf{s}') \alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a}), \\
\implies \int_{\mathcal{S}'} \hat{P}(\mathrm{d}\mathbf{s}'|\mathbf{s},\mathbf{a}) &= \sum_{j:\mathbf{s}'_j\in\mathcal{S}'_n}^{n} \alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a}),
\end{aligned} \tag{2.57}$$

where $m(\cdot)$ is a measure (see definition B.2) chosen to be a delta function concentrated at $\mathbf{s}'_j \in \mathcal{S}'_n$ is $\delta_{\mathbf{s}'_j}(\cdot)$ (where $\int_{\mathcal{S}} \delta_{\mathbf{s}'_j}(\mathrm{d}\mathbf{s}')\boldsymbol{\phi}(\mathbf{s}') = \boldsymbol{\phi}(\mathbf{s}'_j)$), $\mathbf{w}_{j:} \in \mathbb{R}^{1\times n}$ is a row of $\mathbf{W}$ and $\alpha_j(\mathbf{s},\mathbf{a}) = \mathbf{w}_{j:}\boldsymbol{\psi}(\mathbf{s},\mathbf{a})$. In a pseudo-MDP $\alpha_j(\mathbf{s},\mathbf{a})$ assigns a weight to each $\mathbf{s}'_j \in \mathcal{S}'_n$ whose sum (2.57) does not necessarily have to be 1, so long as contraction (lemma 3) is maintained. A CME satisfying this constraint is known as a *proper* CME. The form of the original CME can then be recovered by using the pseudo-MDP's probability

kernel,

$$\begin{aligned}\hat{\mathcal{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] &= \Big\langle v, \int_{\mathcal{S}} \hat{P}(\mathrm{d}\mathbf{s}'|\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}')\Big\rangle_{\mathcal{F}}, \\ &= \Big\langle v, \int_{\mathcal{S}} \sum_{j=1}^{n} \delta_{\mathbf{s}'_j}(\mathrm{d}\mathbf{s}')\alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}')\Big\rangle_{\mathcal{F}}, \\ &= \Big\langle v, \sum_{j=1}^{n} \alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}'_j)\Big\rangle_{\mathcal{F}}, \\ &= \sum_{j=1}^{n} \alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a})v(\mathbf{s}'_j),\end{aligned}$$

where the last line is due to the reproducing property (definition B.12). Grünewälder et al. (2012a) maintain the contraction constraint for the CME by normalising the α weights,

$$\alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a}) \leftarrow \frac{\alpha_j^{\mathrm{CME}}(\mathbf{s},\mathbf{a})}{||\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})||_1}, \qquad (\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}, \tag{2.58}$$

which is enforced directly after every evaluation of the embedding. Unfortunately this constraint is not part of the vvRKHS optimisation.

2.6.2 Additional Pseudo-MDPs

Two other non-parametric finite induced models are described by the pseudo-MDP abstraction framework.

Non-Parametric Finite-Induced MDPs

KBRL Ormoneit and Sen (2002) approximate the conditional expectation with a *weighting kernel* $\alpha_j(\mathbf{s},\mathbf{a})$ centred at any $\mathbf{s}\in\mathcal{S}$ in continuous space and formed from transition data $\mathcal{D}_n := \{(\mathbf{s},\mathbf{a},\mathbf{s}')_i\}_{i=1}^n$,

$$\begin{aligned}\hat{\mathcal{E}}_{(\mathbf{s},\mathbf{a})}[v] &= \sum_{j=1}^{|\mathcal{S}'_{\mathbf{a}}|} \alpha_j^{\mathrm{KBRL}}(\mathbf{s},\mathbf{a})v(\mathbf{s}'_j), \quad \mathbf{s}'\in\mathcal{S}'_{\mathbf{a}}\subset\mathcal{S}'_n \\ &= \sum_{\mathbf{s}'\in\mathcal{S}'_{\mathbf{a}}} \frac{K(\mathbf{s},\mathbf{s}')}{\sum_{\mathbf{s}''\in\mathcal{S}'_{\mathbf{a}}} K(\mathbf{s},\mathbf{s}'')}v(\mathbf{s}').\end{aligned}$$

$\mathcal{S}'_{\mathbf{a}}$ is the subset of sampled successor states which were only reached by applying action $\mathbf{a}\in\mathcal{A}$ and $K:\mathcal{S}\times\mathcal{S}\to\mathbb{R}$ is a Gaussian kernel (see section B.4.2) with a bandwidth parameter σ that controls the extent of local averaging.

Kernel Smoothing The weighting kernel approach can be generalised to kernel smoothing (KS) as follows. Given transition data $\mathcal{D}_n$, then the Nadaraya-Watson

(Nadaraya, 1963; Watson, 1964) estimator can be implemented as

$$\begin{aligned}\hat{\mathscr{E}}^{\hat{\rho}^{\mu}}_{(\mathbf{s},\mathbf{a})}[v] &= \sum_{j=1}^{n} \alpha_j^{\mathrm{KS}}(\mathbf{s},\mathbf{a}) v(\mathbf{s}'_j), \quad \mathbf{s}' \in \mathcal{S}'_n \\ &= \sum_{j=1}^{n} \frac{K((\mathbf{s},\mathbf{a})_j, (\mathbf{s},\mathbf{a}))}{\sum_{i=1}^{n} K((\mathbf{s},\mathbf{a})_i, (\mathbf{s},\mathbf{a}))} v(\mathbf{s}'_j),\end{aligned}$$

where $K\colon (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A})$ is a Gaussian kernel and $\mathcal{S}'_n$ is the set of successor states in training data $\mathcal{D}_n$. A product kernel is also a kernel (Shawe-Taylor and Cristianini, 2004, proposition 3.22 iii)), therefore we decompose K into an action and state kernel,

$$K((\mathbf{s},\mathbf{a}),(\mathbf{s}',\mathbf{a}')) := K_{\sigma_{\mathbf{s}}}(\mathbf{s},\mathbf{s}') K_{\sigma_{\mathbf{a}}}(\mathbf{a},\mathbf{a}'), \tag{2.59}$$

where $\sigma_{\mathbf{s}}$ and $\sigma_{\mathbf{a}}$ are the state and action bandwidths respectively. As $\sigma_{\mathbf{a}} \to 0$ then the KRLS weighting kernel is recovered, such that the action Gaussian kernel behaves like a delta function. This is under the assumption $\mathcal{S}'_n := (\bigcup_{\mathbf{a}\in\mathcal{A}} \mathcal{S}'_{\mathbf{a}})$ in MDPs whose $\mathcal{S}$ is likely continuous and transition dynamics are stochastic. Increasing $\sigma_{\mathbf{a}}$ increases the smoothing across actions if required. By choosing K as the Gaussian kernel, the pseudo-MDP constraint over $\boldsymbol{\alpha}^{\mathrm{KS}}$ is naturally maintained as the KS weights form a probability distribution. Ormoneit and Sen (2002, Part 5) provide consistency results for KBRL such that in the limit of infinite data, the pseudo-MDP converges to the actual MDP.

Factored Linear Action Models Yao et al. (2014a, Part IV A.) propose estimating the transition conditional expectation by solving the following ideal loss or Factored Linear Action Model (FLAM),

$$\mathcal{L}(\mathbf{b}) := \mathbb{E}_{(S,A)\sim D,\, S'\sim P(\cdot|S,A)}\Big[\Big(v(S') - \mathbf{b}^{\top}\boldsymbol{\psi}(S,A)\Big)^2\Big],$$

where feature representation $\boldsymbol{\psi}\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}^d$. Arranging data $\mathcal{D}_n := \{(\mathbf{s},\mathbf{a},\mathbf{s}')_i\}_{i=1}^n$ into matrices $\boldsymbol{\Psi} := [\boldsymbol{\psi}(\mathbf{s}_1,\mathbf{a}_1),...,\boldsymbol{\psi}(\mathbf{s}_n,\mathbf{a}_n)]^{\top} \in \mathbb{R}^{n\times d}$ and $\bar{\mathbf{v}} := [v(\mathbf{s}'_1),...,v(\mathbf{s}'_n)]^{\top} \in \mathbb{R}^n$, then the solution is given by,

$$\begin{aligned}\hat{\mathbf{b}} &= \operatorname*{arg\,min}_{\mathbf{b}\in\mathbb{R}^d} \left[\sum_{i=1}^{n} \Big(\mathbf{b}^{\top}\boldsymbol{\psi}(\mathbf{s}_i,\mathbf{a}_i) - v(\mathbf{s}'_i)\Big)^2\right], \\ &= (\boldsymbol{\Psi}^{\top}\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^{\top}\bar{\mathbf{v}}, \\ &= \mathbf{F}^{\top}\bar{\mathbf{v}},\end{aligned}$$

where for simplicity the inverse is assumed to exist. The approximate expectation operator can then be expressed in the same form as the CME in the pseudo-MDP

framework,

$$\begin{aligned}\hat{\mathscr{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] &= \mathbf{b}^{\top}\boldsymbol{\psi}(\mathbf{s},\mathbf{a}),\\ &= \bar{\mathbf{v}}^{\top}\mathbf{F}\boldsymbol{\psi}(\mathbf{s},\mathbf{a}),\\ &= \int_{\mathcal{S}}\sum_{j=1}^{n}\delta_{\mathbf{s}'_j}(\mathrm{d}\mathbf{s}')\mathbf{f}_{j:}\boldsymbol{\psi}(\mathbf{s},\mathbf{a})v(\mathbf{s}'), \quad \mathbf{s}'_j\in\mathcal{S}'_n,\\ &= \sum_{j=1}^{n}\alpha_j^{\mathrm{FLAM}}(\mathbf{s},\mathbf{a})v(\mathbf{s}'_j),\end{aligned}$$

where $\mathbf{f}_{j:}\in\mathbb{R}^{1\times d}$ is the j^{th} row of $\mathbf{F}\in\mathbb{R}^{n\times d}$ and $\alpha_j^{\mathrm{FLAM}}(\mathbf{s},\mathbf{a}) := \mathbf{f}_{j:}\boldsymbol{\psi}(\mathbf{s},\mathbf{a})$. Consequently the following empirical loss for $\mathbf{F}$ should be small,

$$\begin{aligned}\hat{\mathcal{L}}(\mathbf{F}) &:= \frac{1}{2n}\sum_{i=1}^{n}\left(\bar{\mathbf{v}}^{\top}\mathbf{F}\boldsymbol{\psi}(\mathbf{s}_i,\mathbf{a}_i) - v(\mathbf{s}')\right)^2,\\ &= \frac{1}{2n}||\bar{\mathbf{v}}^{\top}(\mathbf{F}\Psi^{\top}-\mathbf{I}_n)||_2^2.\end{aligned}$$

Under mild value function assumptions, Yao et al. (2014a, Part IV B.) then choose the following objective for learning a FLAM pseudo-MDP,

$$\begin{aligned}\underset{\mathbf{F}}{\text{minimise}} \quad & \frac{1}{2n}||\mathbf{F}\Psi^{\top}-\mathbf{I}_n||_{\mathrm{Fr}}^2,\\ \text{subject to} \quad & \sum_{j=1}^{n}|\mathbf{f}_{j:}\boldsymbol{\psi}(\mathbf{s}'_i,\mathbf{a})|\leq 1, \quad \mathbf{a}\in\mathcal{A},\ 1\leq i\leq n.\end{aligned}$$

which is used to form a Lagrangian that is solved iteratively with steps based on the alternative direction method of multipliers (ADMM) optimisation procedure (Boyd et al., 2011). Each iteration consists of i) an L1-projection (Duchi et al., 2008), ii) $\mathbf{F}$ weight update and finally iii) a Lagrangian multiplier update. Both $\boldsymbol{\psi}$ and $\mathcal{D}_n$ are assumed to be provided at the beginning of the algorithm before model construction and subsequent value iteration on the approximate MDP. This approach differs from the CME approach in that the contraction constraint is maintained during the optimisation procedure instead of as a post hoc procedure when the embedding is evaluated.

Parametric Linear Action Models

Recall the linear value function approximation scheme where $v(\mathbf{s})\approx\langle\mathbf{w},\boldsymbol{\phi}(\mathbf{s})\rangle_{\mathcal{F}}$. By assuming $\mathcal{F}$ is non-parametric then conditional expectations are modelled by finite induced approximate MDPs such that the value prediction problem can be solved exactly. In contrast if $\mathcal{F}$ is a Hilbert space $\mathbb{R}^{d'}$ then this gives rise to parametric conditional expectation models which do not suffer unfavourable memory and computational complexities with sample size n. However parametric MDP models cannot be solved exactly and approximate value prediction methods (see section 2.5.2) are required.

Parr et al. (2008) discuss learning *policy-specific* parametric linear dynamics models with least squares, although their exposition does not explicitly state what targets are to be used in the supervised learning procedure (c.f. Grünewälder et al. (2012a)'s policy invariant models (2.53) which *are* learnt with a suitable supervised learning scheme). Yao and Szepesvári (2012) develop *policy-invariant* parametric linear action models (LAMs) and use them as components to an approximate policy iteration (LAM-API) scheme as summarised below.

A LAM is learnt by defining explicit feature representation $\boldsymbol{\phi}: \mathcal{S} \to \mathbb{R}^{d'}$ (not to be confused with the implicit feature $\boldsymbol{\phi} \in \mathcal{H}_L$ in the RKHS setting) and forming the following loss for each $\mathbf{a} \in \mathcal{A}$,

$$\mathcal{L}(\mathbf{F}^{\mathbf{a}}) := \mathbb{E}_{S \sim D,\, S' \sim P(\cdot|S,\mathbf{a})}\Big[||\boldsymbol{\phi}(S') - \mathbf{F}^{\mathbf{a}\top}\boldsymbol{\phi}(S)||_2^2\Big].$$

Separating transitions $\mathcal{D} := \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}_{i=1}^n$ for each action, $\mathcal{D}_{\mathbf{a}} := \{(\mathbf{s}, \mathbf{s}')_i\}_{i=1}^{n_{\mathbf{a}}}$, then $\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{s}_1), ..., \boldsymbol{\phi}(\mathbf{s}_{n_{\mathbf{a}}})]^\top \in \mathbb{R}^{n_{\mathbf{a}} \times d'}$, $\boldsymbol{\Phi}' := [\boldsymbol{\phi}(\mathbf{s}'_1), ..., \boldsymbol{\phi}(\mathbf{s}'_{n_{\mathbf{a}}})]^\top \in \mathbb{R}^{n_{\mathbf{a}} \times d'}$ and the transition model for $\mathbf{a} \in \mathcal{A}$ is (see section B.4.4)

$$\begin{aligned}
\hat{\mathbf{F}}^{\mathbf{a}} &= \underset{\mathbf{F}^{\mathbf{a}} \in \mathbb{R}^{d \times d}}{\arg\min}\left[\frac{1}{2n}\sum_{i=1}^{n}||\boldsymbol{\phi}(\mathbf{s}') - \mathbf{F}^{\mathbf{a}\top}\boldsymbol{\phi}(\mathbf{s})||_2^2 + \lambda||\mathbf{F}^{\mathbf{a}}||_{\mathrm{Fr}}^2\right], \\
&= \underset{\mathbf{F}^{\mathbf{a}} \in \mathbb{R}^{d \times d}}{\arg\min}\left[\frac{1}{2n}||\boldsymbol{\Phi}' - \boldsymbol{\Phi}\mathbf{F}^{\mathbf{a}}||_{\mathrm{Fr}}^2 + \lambda||\mathbf{F}^{\mathbf{a}}||_{\mathrm{Fr}}^2\right], \\
&= (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\mathbf{I}_{d'})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}', \\
&= \mathbf{W}^{\mathbf{a}\top}\boldsymbol{\Phi}',
\end{aligned} \tag{2.61}$$

where $\mathbf{W}_{\mathbf{a}} \in \mathbb{R}^{n_{\mathbf{a}} \times d}$. The embedding

$$\begin{aligned}
\hat{\mu}_{S|\mathbf{s},\mathbf{a}} &= \hat{\mathbf{F}}^{\mathbf{a}\top}\boldsymbol{\phi}(\mathbf{s}), \\
&= \boldsymbol{\Phi}'^\top\mathbf{W}^{\mathbf{a}}\boldsymbol{\phi}(\mathbf{s}), \\
&= \boldsymbol{\Phi}'^\top\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a}), \\
&= \sum_{j=1}^{|\mathcal{S}_{n_{\mathbf{a}}}|}\boldsymbol{\alpha}_j^{\mathrm{LAM}}(\mathbf{s}, \mathbf{a})\boldsymbol{\phi}(\mathbf{s}'_j).
\end{aligned}$$

forms the parametric approximate expectation operator,

$$\begin{aligned}
\hat{\mathcal{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] &= \langle \mathbf{w}, \hat{\mu}_{S|\mathbf{s},\mathbf{a}} \rangle_2, \\
&= \mathbf{w}^\top\mathbf{F}^{\mathbf{a}\top}\boldsymbol{\phi}(\mathbf{s}),
\end{aligned}$$

noting that as the embedding is *not* a function of the successor training samples, then the Bellman operator is inexact. As a consequence the LAM-API algorithm requires LSTD or Bellman residual minimisation in order to fit (or find $\mathbf{w} \in \mathbb{R}^{d'}$ of) a value function.

2.6.3 Research Preamble

Both non-parametric (KS, CME, FLAM) and parametric (LAM) MBRL policy iteration are stated in algorithm 6 and algorithm 7 respectively. The immediate reward function is assumed to be known in order to focus on the transition dynamics.

Algorithm 6 Existing Non-Parametric MBRL Policy Iteration (KS, CME, FLAM)

Input: Kernel $L : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, implicit state representation $\boldsymbol{\phi}(\mathbf{s}') := L(\mathbf{s}', \cdot)$, data $\mathcal{D} := \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}_{i=1}^n$, $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n \in \mathcal{D}$ extracted from an unknown MDP $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, r, \gamma, P\}$ with unknown probability kernel $P : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \to [0, 1]$ and known average reward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$.
Output: $\pi_K(\mathbf{s}) \approx \pi^*(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}$.
Model: $\hat{\mu}_{S'|\mathbf{s},\mathbf{a}} := \sum_{j=1}^n \alpha_j(\mathbf{s}, \mathbf{a}) \boldsymbol{\phi}(\mathbf{s}'_j) \leftarrow \textsc{BatchTrain}\Big(L, \mathcal{D}\Big)$, $\mathbf{v} := [v(\mathbf{s}'_1), ., v(\mathbf{s}'_n)]^\top$, $\hat{\mathscr{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] := \sum_{j=1}^n \alpha_j(\mathbf{s}, \mathbf{a}) v(\mathbf{s}'_j)$, $(\hat{T}^{\pi_k}_{\mu} v)(\cdot) := r(\cdot, \pi_k(\cdot)) + \gamma \boldsymbol{\alpha}^\top(\cdot, \pi_k(\cdot))\mathbf{v}$.
Initialise: $\hat{q}_0(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a})$, $\pi_1(\mathbf{s}) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}_0(\mathbf{s}, \mathbf{a})]$,
Planning:
for $k = 1, 2, ...K-1$ **do** ▷ k^{th} policy iteration master index
  $\mathbf{v} \leftarrow \mathbf{0}$
  for $j = 1$ **to** J_{eval} **do** ▷ exact policy evaluation
    $\mathbf{v} \leftarrow \hat{T}^{\pi_k}_{\mu} \mathbf{v}$
  end for
  $\pi_{k+1}(\cdot) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[r(\cdot, \mathbf{a}) + \gamma \boldsymbol{\alpha}^\top(\cdot, \mathbf{a})\mathbf{v}]$ ▷ policy improvement
end for
return $\pi_K(\cdot)$

Algorithm 7 Existing Parametric MBRL Policy Iteration (LAM)

Input: Explicit state representation $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^{d'}$, data $\mathcal{D} := \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}_{i=1}^n$, $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n \in \mathcal{D}$ extracted from an unknown MDP $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, r, \gamma, P\}$ with unknown probability kernel $P : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \to [0, 1]$ and known average reward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$.
Output: $\pi_K(\mathbf{s}) \approx \pi^*(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}$.
Model: $\{\hat{\mu}_{S'|\mathbf{s},\mathbf{a}} := \mathbf{F}^{\mathbf{a}\top} \boldsymbol{\phi}(\mathbf{s})\}_{\mathbf{a} \in \mathcal{A}} \leftarrow \textsc{BatchTrain}\Big(\boldsymbol{\phi}, \mathcal{D}_n\Big)$, $\hat{\mathscr{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}[v] := \mathbf{w}^\top_{\pi_k} \mathbf{F}^{\mathbf{a}\top} \boldsymbol{\phi}(\mathbf{s})$, $(\hat{T}^{\pi_k}_{\mu} v)(\cdot) := r(\cdot, \pi_k(\cdot)) + \gamma \mathbf{w}^\top_{\pi_k} \mathbf{F}^{\mathbf{a}\top} \boldsymbol{\phi}(\cdot)$, for some $\mathbf{w}_{\pi_k} \in \mathbb{R}^{d'}$ to be found.
Initialise: $\hat{q}_0(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a})$, $\pi_1(\mathbf{s}) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}_0(\mathbf{s}, \mathbf{a})]$.
Planning:
for $k = 1, 2, ...K-1$ **do** ▷ k^{th} policy iteration master index
  $\mathbf{w}_{\pi_k} \leftarrow \textsc{FitValueFunction}\Big(\{\mathbf{F}^{\mathbf{a}\top} \boldsymbol{\phi}(\mathbf{s})\}_{\mathbf{a} \in \mathcal{A}}, r, \pi_k, \mathcal{D}_n\Big)$ ▷ LSTD or BRM
  $\pi_{k+1}(\mathbf{s}) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbf{w}^\top_{\pi_k} \mathbf{F}^{\mathbf{a}\top} \boldsymbol{\phi}(\mathbf{s})]$ ▷ policy improvement
end for
return $\pi_K(\cdot)$

Recall that KS, CME and FLAM build approximate expectations that are finite weighted sums of value function evaluations over the training data $\mathcal{S}_n$. Classical DP algorithms such as policy iteration can then be deployed to solve an approximate

MDP by iterating $\hat{T}^*_\mu$ exactly. A CME embedding is finite because it models the value function in a rich non-parametric function class such as an RKHS $\mathcal{F} = \mathcal{H}_L$. Policy iteration computational complexity is therefore decoupled from the size of the original MDP's state space (which could be infinite). The advantage to this approach is that function approximation procedures (such as supervised learning) are relegated to fitting a model only (in function BatchTrain) and do not contaminate the 'actual RL' value prediction/policy evaluation process. The pseudo-MDP framework provides policy improvement guarantees based on the accuracy of the model and methods such as KS or CME boast consistency guarantees in the limit of infinite data. Explicit feature representation is also not needed for KS or CME, instead kernels are only needed whose parameters can be found with cross-validation during training. However the disadvantage with such methods is that planning and model construction computational complexities scale unfavourably with the training set size n.

Table 2.2 Approximate policy iteration schemes; $|\mathcal{D}| = |\mathcal{S}'| = n$. Cubic complexities are due to matrix inversion and therefore ripe for more efficient inversion techniques. †LSTD or BRM complexities. ‡ Pseudo MDPs have policy improvement suboptimality guarantees, however KS and CME also have explicit *consistency* results such that their approximate MDPs converge to the actual MDP in the limit of infinite data; LAM parametric lacks this quality because its feature representation is not augmented with new data. ⋆KS cross-validation scheme is quadratic, however if hyperparameters are known then this is linear.

Algorithm	‡Guarantees	Planning		Model $\hat{\mathscr{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}$	
		π evaln	π impvt	build	eval
DP	✓	$\lvert\mathcal{S}\rvert^2$	$\lvert\mathcal{A}\rvert\lvert\mathcal{S}\rvert^2$	-	$\lvert\mathcal{S}\rvert$
KS	✓	n^2	$\lvert\mathcal{A}\rvert n^2$	$^\star n^2$	n
CME	✓	n^2	$\lvert\mathcal{A}\rvert n^2$	n^3	n
FLAM	✓	n^2	$\lvert\mathcal{A}\rvert n^2$	$\lvert\mathcal{A}\rvert n \dim(\boldsymbol{\psi})^2$	$n \dim(\boldsymbol{\psi})$
LAM	✓	$^\dagger\dim(\boldsymbol{\phi})^3$	$\lvert\mathcal{A}\rvert\dim(\boldsymbol{\phi})^2$	$\dim(\boldsymbol{\phi})^3$	$\dim(\boldsymbol{\phi})^2$

A direct contrast with this approach is LAM which approximates v in a *parametric* function class $\mathcal{F} = \mathcal{F}_\phi$ such that the approximate transition expectation is *not* a finite sum of value function evaluations. Consequently the approximate Bellman operator $\hat{T}$ is inexact (as discussed in section 2.5.2) and value prediction constitutes fitting the value function by searching for $\mathbf{w} \in \mathbb{R}^{d'}$ (in function FitValueFunction). This approach inherits the same characteristics of model-free value-based RL under parametric value function approximation and therefore may exhibit policy oscillations under the 'deadly triad'. This approach lacks consistency guarantees because the expressiveness of ϕ is not improved by data. However LAM is a pseudo-MDP and therefore it does have policy improvement guarantees. The primary advantage of the

parametric setting is that it scales favourably with n, but the primary disadvantage is that it requires explicit feature representation.

This investigation treads a line between this apparent dichotomy of parametric and non-parametric $\mathcal{F}$, policy iteration stability and computational complexity. The objective is to produce an algorithm that has favourable complexity with n, rich data-driven feature representations (as opposed to them being given a priori) and policy improvement stability. All existing algorithms assume adequate training data a priori and therefore policy learning stability during exploration is not considered. Model learning is also a batch process and it is desirable to find online solutions. An empirical study that compares the parametric and non-parametric approaches has not yet been done and it is desirable to compare policy learning stability for both paradigms. The existing characteristics for both sets of algorithms are listed in table 2.2.

Chapter 3

Benchmark Algorithms and Initial Improvements

The following work compares existing pseudo-MDP algorithms of which the most successful are used in the rest of the investigation as benchmarks. This is achieved by running CME, KS and FLAM pseudo-MDP embeddings in a policy iteration algorithm as described in section 2.3.3. The assumption that training data is given a priori is removed and instead the algorithm forces the agent to repeat the process of gathering data, building a new model and planning for an improved policy as described in fig. 1.1. In anticipation of policy oscillations an exploration procedure is developed such that the entire algorithm is known as *explorative pseudo-PI*. Although each task is a continuing task, the learning algorithm developed here takes an episodic perspective such that training data is collected in trajectories of finite length H, beginning from states drawn from an initial state distribution P_1. Empirical cumulative reward and timings of algorithm components are compared at each k^{th} policy iteration.

In the context of model-based approximate policy iteration, one goal of this thesis is to empirically investigate policy learning stability with function approximation. Therefore establishing the stability of the benchmark algorithms is important. Another goals is to develop an *online* MBRL algorithm characterised by approximate MDP embeddings that can be solved exactly. The CME model construction algorithm is therefore modified by efficient block inversion of the kernel matrix which improves model construction complexity from cubic in the training data to quadratic. In addition a pseudo-MDP contraction constraint is implemented for the CME as a fast L1-projection.

The parametric LAM algorithm is not compared as it was identified as an existing pseudo-MDP algorithm only after experiments were carried out. It is however stated in Yao et al. (2014a) that due to LAM embeddings keeping separate action models rather that generalising over $\mathcal{S}\times\mathcal{A}$, its performance may not be competitive. This issue

will become relevant in chapter 4. An empirical investigation of LAM in explorative pseudo-PI is left for future work.

3.1 Explorative PI

Throughout this analysis the average immediate reward function is assumed known in order to concentrate on the task of learning good transition models.

3.1.1 Online Data Acquisition

The explorative pseudo-PI control algorithm is described by algorithm 8 and is an umbrella learning algorithm for KS, CME and FLAM variants. Data acquisition is made at the beginning of each k^{th} policy iteration such that the existing data set $\mathcal{D}_{n_{k-1}}$ is augmented by new data $\mathcal{D}_{\text{new}}$ gathered from interactions with the real MDP. At the beginning of each iteration a new approximate MDP is built in a batch learning process over the entire augmented data set.

It is useful to reflect on how model-free control methods incorporate exploration in section 2.4.2. On-policy or off-policy sample targets of the action-value Bellman operator form SARSA or Q-learning respectively. When combined with TD learning and function approximation then off-policy learning can elicit algorithm divergence as discussed empirically in Dann et al. (2014, section 2.4.2)). Exploration of an MDP usually proceeds by using an ϵ-greedy behaviour policy that injects stochasticity into action selection in order to make the probability of exploring all actions in each state non-zero.

In contrast for dynamic programming, each (full backup) policy iteration update is towards the optimal value function (equation (2.20)). For explorative pseudo-PI, then the value function point-evaluations on the training set successor states $\mathcal{S}_n$ are nudged towards the optimal policy of the current pseudo-MDP defined by the current embedding μ,

$$v(\mathbf{s}') \mapsto (\hat{T}^*_\mu v)(\mathbf{s}'), \qquad \forall\, \mathbf{s}' \in \mathcal{S}'_n,$$

At each $k = \kappa$ policy iteration, the transition data $\mathcal{D}_{n_\kappa}$ used to train $\hat{\mu}$ was gathered by all previous $k < \kappa$ embedding optimal policies. Explorative pseudo-PI is therefore inherently off-policy.

All behavioural policies generated by an embedding at the k^{th} iteration is a function of the embedding model. If an agent strays to regions of $\mathcal{S} \times \mathcal{A}$ that are far from the data distribution that the model is trained on, then control signals drawn from the policy may quickly become inaccurate. Ross and Bagnell (2012) mitigate this problem by developing an *agnostic system identification* method in online settings known as

Algorithm 8 Explorative Pseudo-PI (KS, CME, FLAM)

Input: Kernel $L : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, implicit state representation $\boldsymbol{\phi}(\mathbf{s}') := L(\mathbf{s}', \cdot)$, access to unknown MDP $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, P, P_1, r, \gamma\}$ with continuous $\mathcal{S}$, discrete $\mathcal{A}$, but known average reward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$. $H = 100$ transitions per trajectory, $J_{\text{imp}} := 10$, $J_{\text{eval}} := 4000$, start-state distribution P_1.
Output: $\pi_K(\cdot) \approx \pi^*(\cdot)$.
Initialise: $\hat{q}_0(\cdot, \mathbf{a}) \leftarrow r(\cdot, \mathbf{a})$, $\pi_1(\cdot) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}(\cdot, \mathbf{a})]$, exploration policy $\nu^{\pi_1}(\cdot) \leftarrow \epsilon\text{-greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}(\cdot, \mathbf{a})]$, $\epsilon \leftarrow 0.3$, $n_0 \leftarrow 0$, $n_{\text{new}} \leftarrow 2H$, $\mathcal{D}_0 = \emptyset$.
for $k = 1, 2, ..., K-1$ **do** ▷ k^{th} policy iteration master index
  Data acquisition: Collect $\mathcal{D}_{\text{new}} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_{k\text{-}1}+n_{\text{new}}}$; one trajectory from each π_k and ν^{π_k}, beginning from start state $\mathbf{s} \sim P_1$,
  $\mathcal{D}_k \leftarrow \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}}$, $n_k \leftarrow n_{k\text{-}1} + n_{\text{new}}$. ▷ aggregate data
  Model: $\sum_{j=1}^{n_k} \alpha_j(\cdot, \cdot)\boldsymbol{\phi}(\mathbf{s}'_j) \leftarrow \textsc{BatchTrain}\Big(L, \mathcal{D}_k\Big)$, $\mathbf{v} := [v(\mathbf{s}'_1), ., v(\mathbf{s}'_{n_k})]^\top$, $(\hat{T}^\pi_\mu v)(\cdot) := r(\cdot, \pi(\cdot)) + \gamma \boldsymbol{\alpha}^\top(\cdot, \pi(\cdot))\mathbf{v}$.
  Planning:
  for $i = 1$ **to** J_{imp} **do** ▷ policy improvement index
    $\mathbf{v} \leftarrow \mathbf{0}$, $\pi \leftarrow \pi_k$
    for $j = 1$ **to** J_{eval} **do** ▷ exact policy evaluation
      $\mathbf{v} \leftarrow \hat{T}^\pi_\mu \mathbf{v}$
    end for
    $\hat{\mathbf{v}}^\pi \leftarrow \mathbf{v}$
    $\hat{q}^\pi(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \gamma \boldsymbol{\alpha}^\top(\mathbf{s}, \mathbf{a})\hat{\mathbf{v}}^\pi, \ (\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$
    $\pi(\cdot) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}^\pi(\cdot, \mathbf{a})]$ ▷ greedy policy improvement
  end for
  $\pi_{k+1} \leftarrow \pi$
end for
return $\pi_K(\cdot)$

DAgger. This approach specifies that if *new* model training data $\mathcal{D}_{\text{new}}$ comprises of i) on-policy data and ii) data taken from a suitable exploration distribution, then optimal policy convergence guarantees exist. Their premise is that if a model's training data is drawn from an on-policy distribution bathed in an envelope of data drawn from an exploration distribution, then controller performance is robust at test-time to trajectories perturbed away from those generated by the current policy. Therefore at the k^{th} policy iteration, explorative pseudo-PI collects data $\mathcal{D}_{\text{new}}$ drawn from both i) the current policy π_k and ii) an ϵ-greedy exploration policy ν^{π_k} based on π_k with a constant $\epsilon = 0.3$ i.e. 15% of all actions are random during new data acquisition. Both policies combined can be considered as the behaviour policy, however in practice training data consists of a single trajectory from both the current and exploration policies. The envelope of exploratory data protects controller performance when operating at test time and provides enough exploration for approximate MDP convergence to the real MDP.

3.1.2 CME Improvements

At the k^{th} policy iteration the training data comprises of $\mathcal{D}_k = \mathcal{D}_{k-1} \bigcup \mathcal{D}_{\text{new}}$ of size n_k. Training the CME embedding over $\mathcal{D}_k$ requires the computation $(\mathbf{K}_{n_k n_k} + \lambda \mathbf{I}_{n_k})^{-1}$ (see equation (2.49)) and naively costs $\sim \mathcal{O}(n_k^3)$. However by only using this regularised inverse with cross-validation on iterations $k = \{1, 2, 5\}$ for a *full* model relearn, then for all other iterations an efficient block inverse can be used instead,

$$\mathbf{K}_{n_k n_k} + \lambda \mathbf{I}_{n_k} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \begin{pmatrix} \mathbf{K}_{n_{k-1} n_{k-1}} + \lambda \mathbf{I}_{n_{k-1}} & \mathbf{K}_{n_{k-1} n_{\text{new}}} \\ \mathbf{K}_{n_{\text{new}} n_{k-1}} & \mathbf{K}_{n_{\text{new}} n_{\text{new}}} + \lambda \mathbf{I}_{n_{\text{new}}} \end{pmatrix},$$

$$(\mathbf{K}_{n_k n_k} + \lambda \mathbf{I}_{n_k})^{-1} = \begin{pmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{W}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{W}/\mathbf{A})^{-1} \\ (\mathbf{W}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{W}/\mathbf{A})^{-1} \end{pmatrix},$$

where the Schur compliment is $(\mathbf{W}/\mathbf{A})^{-1} := (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$, $\mathbf{K}_{n_k n_k} \in \mathbb{R}^{n_k \times n_k}$ and $\left(\mathbf{K}_{n_k n_k}\right)_{ij} = K((\mathbf{s}, \mathbf{a})_i, (\mathbf{s}, \mathbf{a})_j)$. The block inverse is therefore reduced to matrix multiplication which costs $\mathcal{O}(n_k^2)$ whose efficiency is realised by the reuse of the previous iteration's inverted kernel matrix $\mathbf{A}^{-1} = (\mathbf{K}_{n_{k-1} n_{k-1}} + \lambda \mathbf{I}_{n_{k-1}})^{-1}$. For these quick *partial* re-learns, kernel/cross-validation hyper-parameters are not tuned and instead the previous iteration's parameters are used, therefore the cost of cross validation is avoided.

The final modification is to apply Duchi et al. (2008)'s efficient $\sim \mathcal{O}(n_k)$ L1-projection of $\boldsymbol{\alpha}^{\text{CME}}(\mathbf{s}, \mathbf{a})$ whenever the embedding is evaluated for any $(\mathbf{s}, \mathbf{a})$ during planning or the evaluation of a greedy policy. The vector is normalised if it already exists in the interior of the L1-ball. The L1-projection satisfies the pseudo-MDP contraction constraint as specified in Yao et al. (2014a). Sparsity is induced when projecting onto the L1-ball's geometry as described in section B.4.5.

3.1.3 Experimental Method

Each variant; kernel smoothing (KS), conditional mean embedding (CME) and factored linear action model (FLAM-ADMM), are implemented in explorative pseudo-PI over three MDPs. Where possible the original model construction method is true to the existing algorithm.

General Settings

At the k^{th} policy iteration the dataset $\mathcal{D}_k$ is decomposed into initial state-actions $\mathcal{Z}_k := \{(\mathbf{s}, \mathbf{a})\}_{i=1}^{n_k}$ and successor states $\mathcal{S}'_k := \{\mathbf{s}'_i\}_{i=1}^{n_k}$. CME and KS algorithms are batch trained with 5-fold cross validation for 20 regularisation hyperparameters $\lambda \in 2^{\{\log_2(10^{-8}):\log_2(1000)\}}$ and 10 state-action kernel $K : (\mathcal{S} \times \mathcal{A}) \times (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ bandwidth hyperparameters $\sigma_{\mathcal{S}\times\mathcal{A}} \in 2^{\{\log_2(0.01):\log_2(5)\}}$. K is a product kernel as described by equa-

tion (2.59). It was found that adding an additional a priori scaling (unique to each MDP) $\mathbf{M}_{\mathcal{S}\times\mathcal{A}}$ of each state-action's elements garnered better results such that

$$K((\mathbf{s},\mathbf{a}),(\mathbf{s}',\mathbf{a}')) = \exp\Big(-\frac{1}{2\sigma^2_{\mathcal{S}\times\mathcal{A}}}([\mathbf{s}^\top,\mathbf{a}^\top]-[\mathbf{s}'^\top,\mathbf{a}'^\top])\mathbf{M}_{\mathcal{S}\times\mathcal{A}}([\mathbf{s}^\top,\mathbf{a}^\top]-[\mathbf{s}'^\top,\mathbf{a}'^\top])^\top\Big).$$

The sample mean $\hat{r}_\gamma$ of the cumulative discounted reward was calculated at each iteration k by averaging empirical cumulative reward over 20 trajectories (each with a horizon of $H=100$ starting from $\mathbf{s}\sim P_1$) by acting on-policy with π_k. Error bars were estimated with 95% confidence bounds by repeating pseudo-PI 20 times for each MDP. Care was made to randomize the random number generator seed at the start of each experiment.

CME Settings

The state kernel bandwidth $\sigma_{\mathcal{S}}$ was provided a priori for each MDP after an informal search but could in the future be cross-cross validated such as in van Hoof et al. (2015). Each element of a state was also scaled a priori (unique to each MDP) such that

$$L(\mathbf{s},\mathbf{s}') = \exp\Big(-\frac{1}{2\sigma^2_{\mathcal{S}}}(\mathbf{s}-\mathbf{s}')^\top\mathbf{M}_{\mathcal{S}}(\mathbf{s}-\mathbf{s}')\Big).$$

FLAM-ADMM Settings

The FLAM-ADMM implementation was kept as close to that in Yao et al. (2014a) as possible, but it proved problematic and several assumptions were required for it to demonstrate good performance.

Feature representation $\boldsymbol{\psi}\in\mathbb{R}^{n_k}$ was implemented at the k^{th} policy iteration as $\boldsymbol{\psi}(\mathbf{s},\mathbf{a}) := [K(\mathbf{b}_1,(\mathbf{s},\mathbf{a})),...,K(\mathbf{b}_{n_k},(\mathbf{s},\mathbf{a}))]^\top$ where $\mathbf{b}_i\in\mathcal{Z}_k$. This was based on data collected from each trajectory and different to the approach in the original paper that required discretizing the state-space a priori. $\boldsymbol{\psi}$ therefore grows with n_k and is directly comparable to the equivalent embedding component for the CME. However it was not possible to cross validate any ADMM hyper-parameter. The loss $\frac{1}{2n}||\mathbf{F}\Psi^\top-\mathbf{I}_n||^2_{\text{Fr}}$ was not able to pick out a validation signal that would reliably select the product kernel's bandwidth $\sigma_{\mathcal{S}\times\mathcal{A}}$. As an alternative, the CME loss was used in order to try and detect a validation signal but to no avail. Instead the bandwidth was selected a priori with the value that the CME discovers through its own cross validation procedure.

Referring to Yao et al. (2014a, part IV, B) then the following modifications were made to get the algorithm to learn; i) the optimisation constant was set to $\mu=0.5$ that controls the strength of the L1 contraction constraint and ii) an additional small ridge term $\lambda=1\times10^{-8}$ was added to the inverse term in ADMM step 3, without which the optimisation was unsuccessful.

3.1.4 Experiments

Both mountain car and cart-pole MDPs were used to implement explorative pseudo-PI and calibrate the existing algorithms. Larger quadrocopter MDPs with high dimensional state space and large action spaces are used in the later chapters.

Mountain Car

The agent controls a car whose goal is to drive out of a steep valley. However the car is underpowered such that it must climb a smaller hill in the opposite direction to the goal before accelerating back towards the goal and out of the valley (Singh and Sutton, 1996, appendix B).

States are defined as $\mathbf{s} = [x, v]^\top \in \mathbb{R}^2$ where x is displacement and v is velocity. State space is $\mathcal{S} = (-1.2, -0.7) \times (-0.07, 0.07)$ and the action space is discrete one dimensional $\mathcal{A} = \{-1, 0, 1\}$. Dynamics are $x_{t+1} = x_t + v_t + \epsilon_1$, $v_{t+1} = 0.001a - 0.0025\cos(3x_t) + \epsilon_2/10$ where ϵ_1, ϵ_2 are Gaussian random variables with standard deviation of 0.02. If $x_{t+1} > 0.6$ then the state is reset to $\mathbf{s}_{t+1} \leftarrow [0.6, 0]^\top$. The immediate reward function is defined as $r(\mathbf{s}, \mathbf{a}) = \exp(-8(x - 0.6)^2)$ which concentrates reward at the top of the main hill. Discount factor is $\gamma = 0.99$, trajectory horizon $H = 100$ and the car begins at the bottom of the valley whose mean is $\mathbf{s}_1 = [-0.5, 0]^\top$. The Gaussian kernels are scaled with $\mathbf{M}_\mathcal{S} = \text{diag}(1, 100)$ (whose bandwidth is given as $\sigma_\mathcal{S} = 0.5$ which was chosen from an informal search) and $\mathbf{M}_{\mathcal{S}\times\mathcal{A}} = \text{diag}(1, 100, 1/25)$.

Near-optimal policies collect discounted cumulative reward of 40 or over which is equivalent to the car climbing the hill in the opposite direction to the goal and then accelerating towards the goal to climb out of the valley. Slightly less collected reward indicates that the car intermittently drops back down into the valley.

Cart-Pole

This problem simulates a pole attached at a pivot to a cart, and by applying force to the cart the pole must be swung to the vertical position and balanced. The problem is under-actuated in the sense that insufficient power is available to drive the pole directly to the vertical position, hence the problem captures the notion of trading-off immediate reward for long term gain. The same simulator as Lagoudakis and Parr (2003) was used in this investigation, except here a continuous reward signal is used.

The state space is two dimensional, $\mathbf{s} = [\theta, \dot{\theta}]^\top \in \mathbb{R}^2$ representing the angle ($\theta = 0$ when the pole is pointing vertically upwards) and angular velocity of the pole. The action set is $\mathcal{A} = \{-50, 0, 50\}$ representing the horizontal force in Newtons applied to the cart. Uniform noise in $[-10, 10]$ is added to each action. The system dynamics are

$\theta_{t+1} = \theta_t + \Delta_t \dot{\theta}_t$, $\dot{\theta}_{t+1} = \dot{\theta}_t + \Delta_t \ddot{\theta}_t$ where

$$\ddot{\theta} = \frac{g \sin(\theta) - \alpha m \ell (\dot{\theta})^2 \sin(2\theta)/2 - \alpha \cos\theta u}{4\ell/3 - \alpha m \ell \cos^2(\theta)},$$

$g = 9.8\text{ms}^{-2}$ is the acceleration due to gravity, $m = 2\text{kg}$ is the mass of the pole, $M = 8\text{kg}$ is the mass of the cart, $\ell = 0.5\text{m}$ is the length of the pole, $\alpha = 1/(m+M)$ and $\Delta_t = 0.1\text{s}$. The immediate reward function is $r(\mathbf{s}, \mathbf{a}) = (1 + \cos(\theta))/2$ which concentrates reward at the inverted position. Discount factor is $\gamma = 0.99$, trajectory horizon $H = 100$ and the pole begins in the *downward* rest position whose mean is $\mathbf{s}_1 = [\pi, 0]^\top$ (c.f. Yao et al. (2014a, part V, A) where it seems the pendulum start distribution is centred on the *inverted* balancing state). The Gaussian kernels are scaled with $\mathbf{M}_{\mathcal{S}} = \text{diag}(1, 1/4)$ (whose bandwidth is given as $\sigma_{\mathcal{S}} = 0.5$ which was chosen from an informal search) and $\mathbf{M}_{\mathcal{S}\times\mathcal{A}} = \text{diag}(1, 1/4, 1/10000)$.

Near-optimal policies collect discounted cumulative reward of 50 or over which is equivalent to the pole being swung backwards slightly and then quickly swung up to the inverted position such that it is balanced for an entire episode. Slightly less collected reward indicates that the controller intermittently drops the pole or it isn't balanced in the right position.

Quadrocopter MDPs

A quadrocopter simulator with aerodynamic turbulence models known as QRSim (Denardi, 2012) was used to create more complex MDPs beyond the usual benchmarks. Control problems specified in this simulator are sufficient to stretch the capabilities of all algorithms developed in this thesis and is a good differentiator for deciding on what methods are successful in scaling up to even more complex MDPs. Typically an agent sends control signals $\mathbf{a} \in \mathcal{A}$ to a simulated PID controller (such that maximum power signals and sudden changes to control signals being issued between any two time steps is regulated) which in turn sends the raw control signals to the simulated quadrocopter. In future experiments it would be interesting to experiment with RL agents issuing commands *directly* to the quadrocopter, bypassing any intermediate PID control system. But as it stands, the PID controller is absorbed into the system dynamics, of which the agent has no prior knowledge.

The state kernel scaling is $\mathbf{M}_{\mathcal{S}} = \text{diag}(\mathbf{1}/5, \mathbf{1}/10^5, \mathbf{1}, \mathbf{1}/10^5, 0)$ where $\mathbf{1} = (1, 1, 1)$ and the state-action scaling is $\mathbf{M}_{\mathcal{S}\times\mathcal{A}} = \text{diag}(\mathbf{1}/5, \mathbf{1}/10^5, \mathbf{1}, \mathbf{1}/10^5, 0, \mathbf{1}/100)$. Both scaling matrices impose the assumption that the variation in each dimension is independent. The scaling is also used to artificially focus the relevant information of the state space for each MDP. Future work would be to relax this focussing mechanism (or add a focussing mechanism that can be *learnt*) and it would be interesting to see the performance of such models.

The implicit output feature map $\phi(\mathbf{s}) = L(\mathbf{s}, \cdot)$ for the CME has a bandwidth set to $\sigma_{\mathcal{S}} = 1$ (chosen by informal search). Two MDPs are set up for this investigation; one is a navigation task where the reward function is based on the proximity of the agent to a location in space, the second task is a holding pattern (introduced in a later chapter) such that reward encourages an agent to assume an orbital holding pattern around a fixed point. Both have challenging state spaces $\mathcal{S} \in \mathbb{R}^{13}$ where trivial discretisation is not practical due to the curse of dimensionality. A state is typically described by $\mathbf{s} := [x, y, z, \dot{x}, \dot{y}, \dot{z}, \theta, \phi, \omega, \dot{\theta}, \dot{\phi}, \dot{\omega}, F_{\text{power}}]^\top$ containing the quadrocopter's position (x, y, z), its rates of change $(\dot{x}, \dot{y}, \dot{z})$, angular orientation (θ, ϕ, ω), its rates of change $(\dot{\theta}, \dot{\phi}, \dot{\omega})$ and finally the power issued to the controls, F_{power}.

Navigation Task MDP The action set is of size $|\mathcal{A}| = 81$ where $\mathcal{A} \in \mathbb{R}^3$ which are uniformly spaced directions in 3D space (corresponding to the orientation of the quadrocopter) which can be interpreted as desired direction. The aim of the task is to fly from an initial location drawn from a small initial state distribution centred at the origin and hover as close to a target location 5m away. As usual, the agent has no knowledge of the transition dynamics. The immediate reward function is defined as a three dimensional Gaussian 'blob' centred at the target location $\mathbf{s}_{\text{targ}}$, $r(\mathbf{s}, \mathbf{a}) = \exp(-\frac{1}{\sigma_r^2}||\mathbf{s} - \mathbf{s}_{\text{targ}}||_2^2)$ where $\sigma_r = 5$ is the reward bandwidth. Discounted cumulative reward of 40 or over is collected for good policies, 45 for near optimal policies.

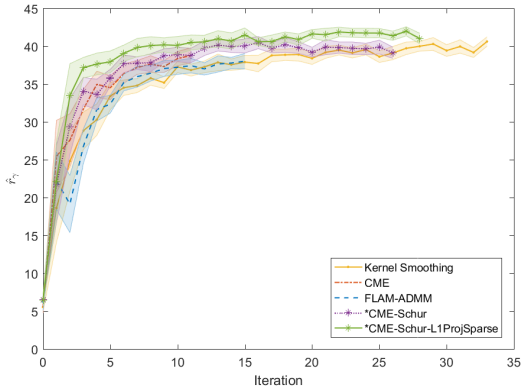
Code Implementation All experiments were written in MATLAB due to it being ideal for algorithm prototyping and due to its optimised linear algebra routines. Code was written from scratch with the exception of a minimal set of speed-optimised libraries such as pairwise distance calculation (Lin, 2007), lasso algorithms (Qian et al., 2013; Schmidt, 2005) and array sorting (Li, 2013). All benchmark algorithms were coded from scratch and a custom neural network implementation was created for the final chapter. MDP implementations were also coded from scratch apart from the core quadrocopter dynamics engine QRSim (Denardi, 2012). No effort was made to parallelise any algorithm as it was thought that comparing sequential implementations was more fair. Future work however will move away from MATLAB's prototyping environment in order to exploit state-of-the-art large-scale libraries.

3.1.5 Results

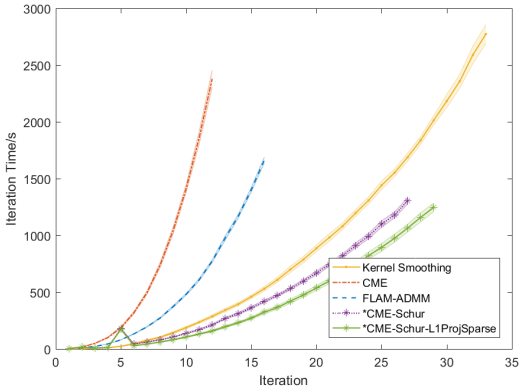
Empirical cumulative discounted reward $\hat{r}_\gamma$ and timings for model construction, planning and iteration duration are presented in fig. 3.1 and fig. 3.2 for both mountain car and cart-pole tasks respectively. Full iteration durations include data acquisition,

Figure 3.1 Mountain car: Benchmarks algorithms and *improvements.

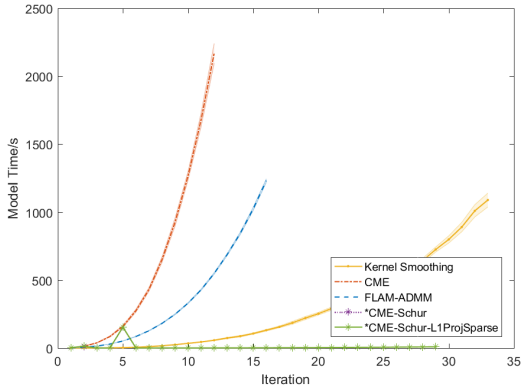
(a) Empirical discounted return



(b) Iteration time



(c) Model construction time



(d) Planning time

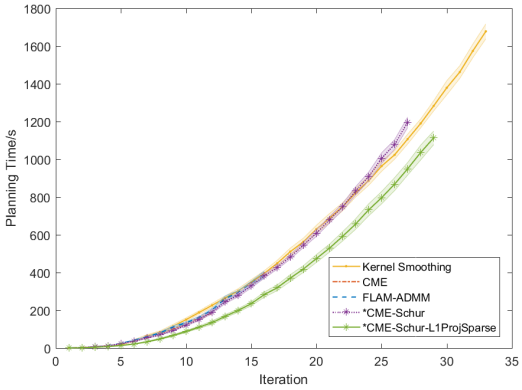
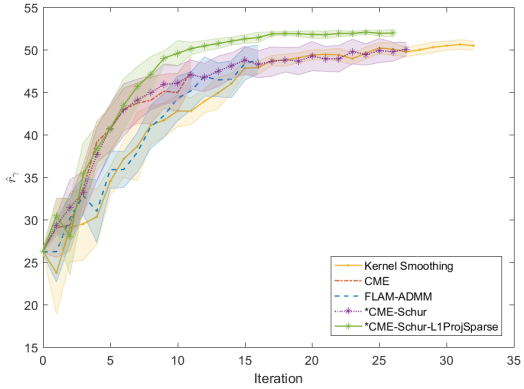
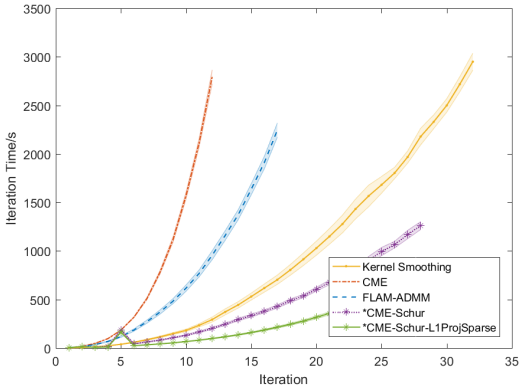


Figure 3.2 Cart Pole: Benchmarks algorithms and *improvements.

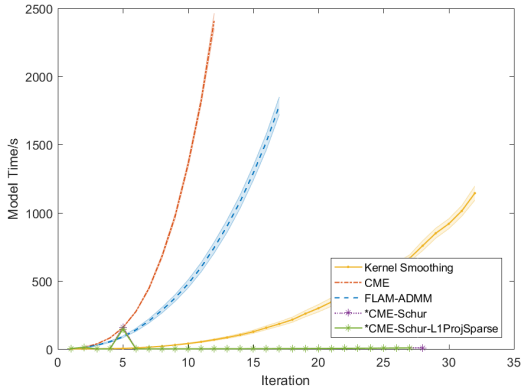
(a) Empirical discounted return



(b) Iteration time



(c) Model construction time



(d) Planning time

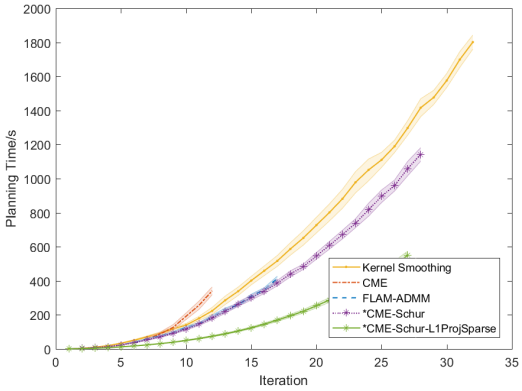
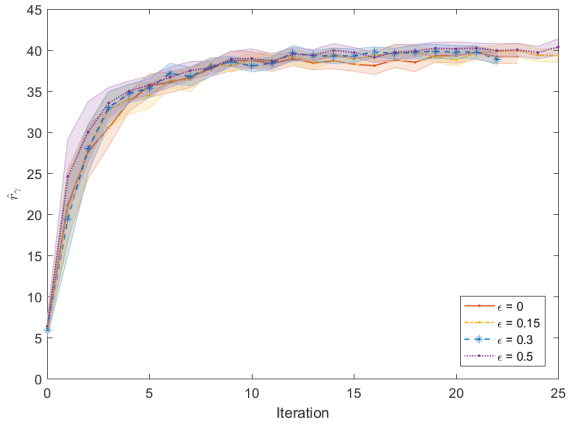
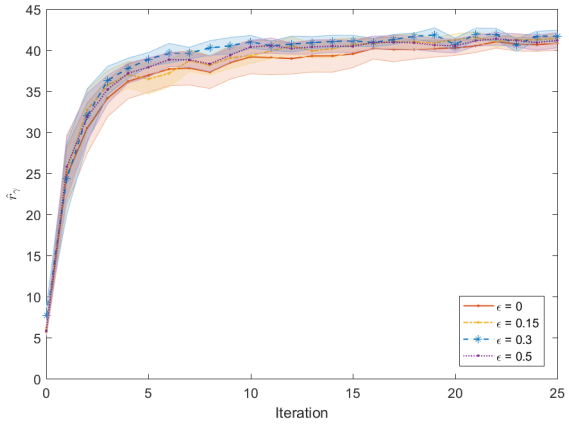


Figure 3.3 Empirical discounted return with varied exploration ϵ for data acquisition. Explorative data acquisition (blue) consistently outperforms non-explorative data acquisition (red).

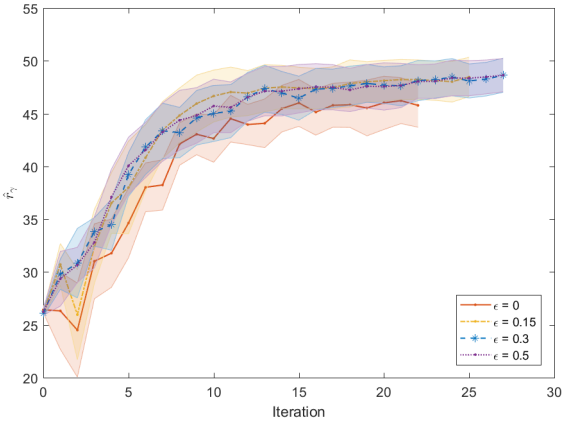
(a) CME: Mountain car,



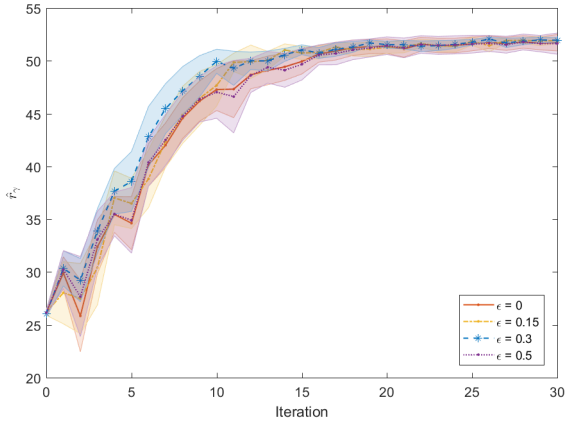
(b) CME-L1ProjSparse: Mountain car



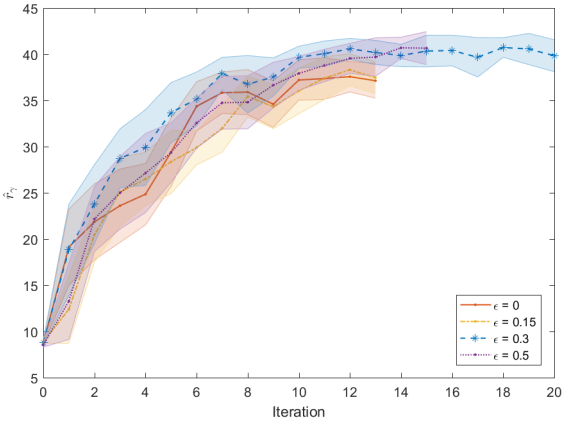
(c) CME: Cart-pole



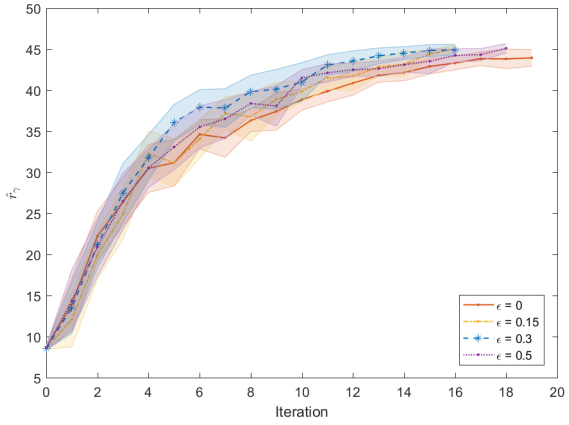
(d) CME-L1ProjSparse: Cart-pole



(e) CME: Quad navigation



(f) CME-L1ProjSparse: Quad navigation



model construction and planning. The complexities of the existing algorithms as detailed in table 2.2 can be modified for explorative pseudo-PI at each k^{th} policy iteration by making the substitution $n = n_k$. For the ADMM implementation then $\dim(\boldsymbol{\psi}) = n_k$. Algorithm component complexities of the modified CME are shown in table 7.1.

A second set of experiments are presented in fig. 3.3 to investigate if the exploration data acquisition approach developed for explorative-API helps build robust transition models. Experiments are carried out for both mountain car, cart-pole and quadrocopter navigation. The exploration parameter ϵ of the behavioural policy ν^π is adjusted for different experiments in the range $\epsilon \in \{0, 0.15, 0.3, 0.5\}$. If $\epsilon = 0$ then no random actions are taken, if $\epsilon = 1$ then all actions are chosen randomly. Recall that data acquisition is performed using a trajectory from the current policy π and a trajectory from ν^π.

3.1.6 Discussion

Surprisingly KS, CME and FLAM-ADMM all readily demonstrate smooth policy improvement in explorative Pseudo-PI. Given that $n_{new} = 200$ for each policy iteration k, all methods learn near-optimal policies with a total of 2000 and 4000 samples for mountain car and cart-pole MDPs respectively. The CME-Schur modification drastically reduces model construction times and adding the L1-projection constraint for CME-Schur-L1ProjSparse demonstrates better policy learning than the original CME algorithm. $\boldsymbol{\alpha}^{\text{CME}}$ sparsity induced by the ℓ_1-Projection gives slightly improved planning times. All other existing algorithms possess poor model construction durations that scale unfavourably with n_k. Note that directly comparing the FLAM-ADMM algorithm to the others is unfair because some of its hyperparameters are not searched for in a cross-validation scheme. Investigation of the FLAM-ADMM algorithm is therefore not pursued for the rest of this investigation due to the difficulty in choosing its a priori parameters that the other methods do not suffer from. The planning components of all pseudo-PI variants presented here scale unfavourably with n_k.

The exploration experiment results in fig. 3.3 support the hypothesis that including additional noisy exploratory data acquisition aids robust system identification. An optimal value of $\epsilon = 0.3$ consistently improves policy learning throughout the entire set of experiments when compared to $\epsilon = 0$. However the improvement is quite small which suggests that there is enough stochasticity in the transition dynamics to explore the MDP. It would be interesting test explorative pseudo-PI with larger MDPs where exploration is far more important.

3.2 Conclusion

Successful integration and modification of existing pseudo-MDP methods into a policy iteration scheme has been achieved where data is not assumed available at the start. Evidence is provided which demonstrates smooth policy improvement without catastrophic policy instabilities. Several algorithms in the literature claim that data can be collected once at the beginning of a learning process with a random policy. However, attempting to explore an entire MDP with a random policy at the beginning of the algorithm will quickly become intractable for large action spaces. Exploring an MDP is preferred with the DAgger-inspired ϵ-greedy data acquisition approach for system identification and is supported with empirical evidence. The optimal exploration parameter for data acquisition is approximately $\epsilon = 0.3$ which implies taking 15% of all actions as random. Performance variance is reduced and learning is shown to occur quicker. It is predicted that for larger MDPs this component will become more important and is open for further investigation.

Explorative pseudo-PI model learning is still effectively a batch model-learning algorithm where batch regression/cross-validation is used over the entire dataset at each policy iteration k to create a transition model. This batch approach may be one of the reasons why policy learning is so stable. However one goal of this investigation is to maintain models with online updates without compromising policy learning stability and will be addressed in later chapters. In an attempt to reduce n_k computational complexities in both model learning and planning, parametric methods are investigated in the next chapter.

Grünewälder et al. (2012a)'s original study assumed batch data was available at the beginning of the experiment to build a one-time embedding. To compare to explorative-PI, this investigation did attempt to create batch data sets at the start only, gathered using random policies in order to build the embedding. This however proved fruitless for the suite of MDPs investigated here as the state space could not be explored adequately before the size of the training set made the experiments too computationally expensive. Discretizing the state space and sampling a transition at each point for the batch data set is unrealistic and quickly becomes intractable for high dimensional and large state-action spaces. Initial experiments showed that the approximate MDPs were not able to catch enough of the dynamics in 'interesting' areas of the state-action space to learn good policies.

Going forward, the exploration parameter is set at $\epsilon = 0.3$. Due to its hyperparameter selection difficulties, the FLAM-ADMM algorithm will not be pursued here and instead is left for future work. The main benchmarks for the remainder of the investigation will therefore be KS and CME-Schur-L1ProjSparse.

Chapter 4

Parametric CME Policy Iteration

Chapter 3 began the investigation by making exploration improvements to existing non-parametric pseudo-MDP algorithms, ensuring robust system identification that mitigates policy improvement instabilities and controller suboptimality at test time. Additionally the CME algorithm was modified to improve model construction and was shown to learn better policies with the addition of the L1-projection contraction constraint. Although the non-parametric value function approach gives rise to exact value prediction during planning, it is tethered to unfavourable complexities to the size of the training set both in model construction and planning.

As a response to the non-parametric embeddings scaling poorly with training data, a parametric CME (PCME) variant is developed such that $v^\pi(\mathbf{s}) \approx \langle w^\pi, \boldsymbol{\phi}(\mathbf{s})\rangle_{\mathcal{F}}$ where $\mathcal{F} = \mathcal{F}_\phi$ defines a parametric state-feature representation. However as discussed in the literature review, parametric value function approximation no longer preserves a contraction in the Bellman operator, even if the transition dynamics are known. Approximate value prediction therefore requires solving the projected Bellman equation (section 2.5.2) using LSTD or BRM. Similarly approximate policy improvement has been claimed to induce instabilities (see section 2.5.3). The PCME deploys explorative parametric-PI to investigate whether policy learning is unstable with linear parametric value function approximation.

Empirical evidence is provided for poor policy stability and a solution inspired by conservative policy iteration (Kakade and Langford, 2002) is proposed that mitigates this instability. A major issue with the parametric approach is that explicit feature representations $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_\psi$ and $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}_\phi$ must be learnt, in comparison to the CME where both representations are implicitly defined by kernels. This chapter develops a vector-valued matching pursuit algorithm that is used to maintain compact parametric feature representations that approximate their kernel-based counterparts.

It is concluded that the investigation goes no further with the pure parametric approach due to the difficulties in learning explicit features and mitigating policy instabilities. However future work is proposed where a deep architecture may be

a more suitable parametric function approximation regime for conservative policy updates. This investigation returns to modifying the non-parametric CME approach in Chapter 5 and Chapter 6.

4.1 Function Approximation with Vector-Valued Matching Pursuit

To facilitate the development of the parametric CME, data-driven representation learning is required. Below is a vector-valued version of the matching pursuit algorithm (Mallat and Zhang, 1993), developed as a method to incrementally build loss-minimising vector-valued function approximators in online environments. In the dual setting, this method can be used to maintain sparse kernel-based function approximators thus mitigating the computational costs of batch kernel regression. In the primal setting it can be used to build regressors whose feature representations are maintained in a data driven way. The matching pursuit algorithm can also be viewed as a method to sparsify existing function approximators to maintain compact representations.

An obvious disadvantage to explicit representations in RL control is that feature engineering may turn out to be task-specific (Peters et al., 2010; Sherstov and Stone, 2005), although sophisticated feature selection techniques have been developed (Painter-Wakefield and Parr, 2012; Parr et al., 2008). A review of the types of explicit representation such as coarse coding, tile coding and radial basis functions can be found in Sutton and Barto (1998, p 202) and Geramifard et al. (2013, chapter 3). The motivation for developing vector-valued matching pursuit is that feature learning is data-driven by minimising a loss function, rendering it task-agnostic.

4.1.1 Algorithm Details

The regression setting (see section B.4.1) is assumed where data $\mathcal{D}_n = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ is collected and modelled as $Y = \mathbf{f}(X) + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, with a vector-valued target function $\mathbf{f} : \mathcal{X} \to \mathcal{Y}$. A vector-valued estimator takes the form,

$$\begin{aligned} \hat{\mathbf{f}}(\cdot) &= \mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot), \\ &= \sum_{\ell=1}^{|\mathcal{B}|} \mathbf{w}_\ell g_\ell(\cdot), \end{aligned} \tag{4.1}$$

where $\mathbf{W} := [\mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{B}|}]$, $\mathbf{w}_\ell \in \mathcal{Y}$ and $\boldsymbol{\psi}_{\mathcal{B}}(\cdot) := [g_1(\cdot), ..., g_{|\mathcal{B}|}(\cdot)]^\top$ is a feature mapping $\boldsymbol{\psi}_{\mathcal{B}} : \mathcal{X} \to \mathcal{F}_{\psi_{\mathcal{B}}}$ which is a vector of $|\mathcal{B}|$ scalar-valued basis functions $g := \mathcal{X} \to \mathbb{R}$. The "basis" of the feature mapping is the collection $\mathcal{B} := \{g_\ell(\cdot)\}_{\ell=1}^{|\mathcal{B}|}$.

In the vvRKHS regression setting (equation (B.68)), the function approximator takes the form of equation (4.1) where $g_\ell(\cdot) := \boldsymbol{\varphi}_\ell(\cdot) = K(\mathbf{x}_\ell, \cdot)$ and $K(\mathbf{x}_\ell, \cdot) : \mathcal{X} \to \mathbb{R}$. The computational cost of batch vvRKHS regression scales $\mathcal{O}(n^3)$ in order to fit $\mathbf{W}$ and is not suitable for online learning environments. In fact $|\mathcal{B}| = n$ such that the $\boldsymbol{\psi}_\mathcal{B}$ representation size scales linearly with the training data size. Vector-valued matching pursuit can instead incrementally build $\mathbf{W}$ and $\mathcal{B}$ rather than using batch regression such that $\mathcal{B} < n$, decoupling the basis from the size of the training set. More importantly matching pursuit can also take the approximator in equation (4.1) as input and sparsify it by finding a compact basis of size $d < |\mathcal{B}|$. Matching pursuit can therefore be understood as a tool for online regression, feature learning and/or function sparsification.

Lemma 4 (Vector-Valued Matching Pursuit for Regression)**.** *Assuming $\mathcal{Y}$ is a Hilbert space, suppose we are given a dictionary $\mathcal{G} = \{\tilde{g}_1, \tilde{g}_2, ...\}$ of candidate real-valued functions $\tilde{g} : \mathcal{X} \to \mathbb{R}$ (which can be any class of scalar-valued functions or function approximators), then matching pursuit aims to find estimators of the form*

$$\hat{\mathbf{f}}^d := \sum_{\ell=1}^{d} \mathbf{w}_\ell g_\ell, \qquad \mathbf{w}_\ell \in \mathcal{Y},$$

where $\mathcal{B} := \{g_\ell\}_{\ell=1}^d \subseteq \mathcal{G}$ is called the basis and $\{\mathbf{w}_\ell\}_{\ell=1}^d$ are the basis weights. If matching pursuit greedily adds a new basis element g_{d+1} to optimize the supervised loss objective evaluated over n data samples,

$$g_{d+1} = \arg\min_{g \in \mathcal{G}} \min_{\mathbf{w} \in \mathcal{Y}} \sum_{i=1}^{n} ||\mathbf{y}_i - (\hat{\mathbf{f}}^d + \mathbf{w}g)(\mathbf{x}_i)||_\mathcal{Y}^2,$$

then the following closed form results hold:
i) If the residue for the d^{th} estimator is defined as

$$\mathbf{r}_i^d = \mathbf{y}_i - \hat{\mathbf{f}}^d(\mathbf{x}_i) \in \mathcal{Y}, \tag{4.2}$$

then the weight of the new $d{+}1$ basis element g_{d+1} is

$$\mathbf{w}_{d+1} = \left(\sum_{i=1}^{n} g_{d+1}(\mathbf{x}_i)\mathbf{r}_i^d\right) / \left(\sum_{i=1}^{n} g_{d+1}(\mathbf{x}_i)^2\right) \in \mathcal{Y}, \tag{4.3}$$

and ii) the new basis element is identified as the solution to

$$g_{d+1} = \arg\sup_{g \in \mathcal{G}} \left[\frac{||\sum_{i=1}^n g(\mathbf{x}_i)\mathbf{r}_i^d||_\mathcal{Y}^2}{\sum_{i=1}^n g(\mathbf{x}_i)^2}\right]. \tag{4.4}$$

Proof. i) Beginning with the optimisation problem,

$$\begin{aligned} g_{d+1} &= \arg\min_{g\in\mathcal{G}} \min_{\mathbf{w}\in\mathcal{Y}} \sum_{i=1}^{n} ||\mathbf{y}_i - (\hat{\mathbf{f}}^d + \mathbf{w}g)(\mathbf{x}_i)||_{\mathcal{Y}}^2 , \\ &= \arg\min_{g\in\mathcal{G}} \min_{\mathbf{w}\in\mathcal{Y}} \sum_{i=1}^{n} ||\mathbf{r}_i^d - \mathbf{w}g(\mathbf{x}_i)||_{\mathcal{Y}}^2 , \end{aligned} \tag{4.5}$$

then since $\nabla_{\mathbf{w}} \sum_{i=1}^{n} ||\mathbf{r}_i^d - \mathbf{w}g(\mathbf{x}_i)||_{\mathcal{Y}}^2 = \mathbf{0}$ at the minimum we have,

$$\begin{aligned} \mathbf{0} &= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left(\langle g(\mathbf{x}_i)\mathbf{w}, g(\mathbf{x}_i)\mathbf{w}\rangle_{\mathcal{Y}} - 2\langle g(\mathbf{x}_i)\mathbf{w}, \mathbf{r}_i^d\rangle_{\mathcal{Y}} \right) \\ &= \sum_{i=1}^{n} 2\mathbf{w}g(\mathbf{x}_i)^2 - 2g(\mathbf{x}_i)\mathbf{r}_i^d \\ \Rightarrow \mathbf{w}_{d+1} &= \left(\sum_{i=1}^{n} g(\mathbf{x}_i)\mathbf{r}_i^d \right) / \left(\sum_{i=1}^{n} g(\mathbf{x}_i)^2 \right), \\ &= \frac{\mathbf{R}^{d\top}\mathbf{g}}{\mathbf{g}^\top\mathbf{g}} \in \mathcal{Y}, \end{aligned} \tag{4.6}$$

where $\mathbf{g} := [g(\mathbf{x}_1), ..., g(\mathbf{x}_n)]^\top$ for any one candidate $g(\cdot) \in \mathcal{G}$, $\mathbf{R}^d := [\mathbf{r}_1^d, ..., \mathbf{r}_n^d]^\top \in \mathbb{R}^{n\times \dim(\mathcal{Y})}$ and the last line costs $\sim\mathcal{O}(n\dim(\mathcal{Y}))$. When $\mathcal{Y}$ is an RKHS will be discussed in the proceeding chapter.

ii) Substituting the value of the new weight into the objective,

$$\begin{aligned} &\sum_{i=1}^{n} ||\mathbf{r}_i^d - \mathbf{w}_{d+1}g(\mathbf{x}_i)||_{\mathcal{Y}}^2 \\ &= \sum_{i=1}^{n} ||\mathbf{r}_i^d||_{\mathcal{Y}}^2 - 2\sum_{i=1}^{n} g(\mathbf{x}_i)\langle \mathbf{r}_i^d, \mathbf{w}^{\min}\rangle_{\mathcal{Y}} + ||\mathbf{w}^{\min}||_{\mathcal{Y}}^2 \sum_{i=1}^{n} g(\mathbf{x}_i)^2 \\ &= \sum_{i=1}^{n} ||\mathbf{r}_i^d||_{\mathcal{Y}}^2 - \frac{2\sum_{i=1}^{n} g(\mathbf{x}_i)\langle \mathbf{r}_i^d, \sum_{k=1}^{n} g(\mathbf{x}_k)\mathbf{r}_k^d\rangle_{\mathcal{Y}}}{\sum_{k=1}^{n} g(\mathbf{x}_k)^2} \\ &\quad + \frac{||\sum_{k=1}^{n} g(\mathbf{x}_k)\mathbf{r}_k^d||_{\mathcal{Y}}^2}{\sum_{i=1}^{n} g(\mathbf{x}_i)^2} \\ &= \sum_{i=1}^{n} ||\mathbf{r}_i^d||_{\mathcal{Y}}^2 - \frac{||\sum_{i=1}^{n} g(\mathbf{x}_i)\mathbf{r}_i^d||_{\mathcal{Y}}^2}{\sum_{i=1}^{n} g(\mathbf{x}_i)^2}. \end{aligned}$$

The next basis that maximally minimises the loss is

$$\begin{aligned} \Rightarrow g_{d+1} &= \arg\sup_{g\in\mathcal{G}} \left[\frac{||\sum_{i=1}^{n} g(\mathbf{x}_i)\mathbf{r}_i^d||_{\mathcal{Y}}^2}{\sum_{i=1}^{n} g(\mathbf{x}_i)^2} \right], \\ &= \arg\sup_{g\in\mathcal{G}} \left[\frac{\langle \sum_{i=1}^{n} g(\mathbf{x}_i)\mathbf{r}_i^d, \sum_{k=1}^{n} g(\mathbf{x}_k)\mathbf{r}_k^d\rangle_{\mathcal{Y}}}{\sum_{i=1}^{n} g(\mathbf{x}_i)^2} \right], \\ &= \arg\sup_{g\in\mathcal{G}} \left[\frac{\sum_{i=1}^{n}\sum_{k=1}^{n} g(\mathbf{x}_i)g(\mathbf{x}_k)\langle \mathbf{r}_i^d, \mathbf{r}_k^d\rangle_{\mathcal{Y}}}{\sum_{i=1}^{n} g(\mathbf{x}_i)^2} \right], \\ &= \arg\sup_{\mathbf{g}\in\mathbb{R}^n} \left[\frac{\mathbf{g}^\top\mathbf{R}^d\mathbf{R}^{d\top}\mathbf{g}}{\mathbf{g}^\top\mathbf{g}} \right]. \end{aligned} \tag{4.7}$$

where the last line costs $\sim\mathcal{O}(n\dim(\mathcal{Y}))$. □

In its simplest form at each iteration we must evaluate equation (4.7) for a selection of $|\mathcal{G}|$ dictionary elements (but not necessarily all) such that the cost of building d-sized basis $\mathcal{B}$ is $\sim\mathcal{O}(\dim(\mathcal{Y})\,|\mathcal{G}|nd)$. The case when $\mathcal{Y}$ is an RKHS will be discussed in the following chapter.

4.1.2 Backfitting

In order to maintain the accuracy of the minimiser of the regression problem, it is good practice to periodically 'backfit' all the weights $\{\mathbf{w}_\ell\}_{\ell=1}^d$ after adding the final d^{th} base by replacing them with the least squares solution as follows. Given a function approximator (4.1) after d matching pursuit iterations then backfit optimisation is specified as a regularised vector-valued regression problem (section B.4.4) with training data $\mathcal{D}_n = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ such that in the primal, $\mathbf{W}$ has closed form solution

$$\mathbf{W}^\top = (\boldsymbol{\Psi}_\mathcal{B}^\top\boldsymbol{\Psi}_\mathcal{B} + \lambda\mathbf{I}_d)^{-1}\boldsymbol{\Psi}_\mathcal{B}^\top\mathbf{R}^{d=0}, \tag{4.8}$$

where $\boldsymbol{\Psi}_\mathcal{B} := [\boldsymbol{\psi}_\mathcal{B}(\mathbf{x}_1), ..., \boldsymbol{\psi}_\mathcal{B}(\mathbf{x}_n)]^\top$ and $\mathbf{R}^{d=0} = \mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_n]^\top$. Backfitting can be applied during the matching pursuit process or at the end. Backfitting in this way relegates the original matching pursuit process (as outlined in lemma 4) to that of a basis $\mathcal{B}$ learner only. The weights are then updated using equation (4.8) after the basis has been learnt. The greedy matching pursuit process does not include a regulariser and therefore it may be vulnerable to overfitting. Empirical evidence presented in figures 4.2 and 4.1 demonstrates backfitting reduces out-of-sample error which suggests matching pursuit without backfitting is prone to overfitting. Matching pursuit can also find a compact representation by adaptively setting a tolerance δ such that the algorithm terminates when it fails to reduce the residue by more than δ. Thus the method will only add features if they significantly enrich the representation.

4.1.3 Function Sparsification

So far vector-valued matching pursuit had been used to greedily find a regressor $\mathbf{f}:\mathcal{X}\to\mathcal{Y}$, but it can also be deployed to sparsify a target function. Sparsification can be made in the same way as in the regression setting; i) the first residue is set as the target function, ii) a dictionary of candidate basis functions is created from the target function's existing basis, iii) an evaluation set $\mathcal{D}_{\text{eval}}$ is drawn from the unseen joint distribution and finally iv) both equation (4.3) and equation (4.4) can be used to greedily find the next weight and basis function. This is still posed as a regression problem.

However vector-value matching pursuit is more powerful if the target function resides in a vector-valued RKHS, $\mathbf{f} \in \mathcal{H}_{\mathbf{\Gamma}}$. Any two RKHS functions which are close in $||\cdot||_{\mathbf{\Gamma}}$ have evaluations which are also close in $||\cdot||_{\mathcal{Y}}$ (see section B.4.2 and section B.4.6 for more details). By exploiting the RKHS norm in this way, it is possible to execute function sparsification without needing $\mathcal{D}_{\text{eval}}$. In Lever and Stafford (2015) we use matching pursuit to maintain a compact representation of a vector-valued RKHS function in this way. The modified matching pursuit derivation is provided in Appendix A.1.4.

4.1.4 Algorithm Implementation

The following implementation demonstrates the vector-valued matching pursuit components. Algorithm 9 (vvMultipleMatchingPursuit) takes as an argument multiple dictionaries of basis elements and cross validates over them to find the best suited dictionary to the data. The difference between each dictionary may be the kernel bandwidth. Its dependencies are algorithm 10 (vvMatchingPursuit) which is the regular matching pursuit algorithm and the backfitting component is algorithm 25 (vvRegression-Primal) found in Appendix B.4.4.

4.1.5 Experiments

The performance of the algorithm is demonstrated in the supervised-learning setting before being applied to learning the PCME in the next section. Results for both scalar and vector-valued target functions are shown in fig. 4.2 and fig. 4.1 respectively. An average out of sample performance (over 20 experiments) is compared for max basis counts of $\{10, 50, 100, 200\}$ with full batch kernel regression (equivalent to a basis of 1000 samples). The higher the sparsification, the higher the out-of-sample error. Backfitting the weights after the basis has been learnt provides significantly lower out-of-sample error and therefore improved generalisation. It is hypothesised that in some cases it could be possible that because greedy updates in lemma 4 do not include regularisation, then matching pursuit function approximators may be prone to overfitting. However cross validating over a regularisation parameter may make the matching pursuit algorithm too costly for online settings and instead backfitting is more efficient. Future work can explore this hypothesis, but going forward, matching pursuit is used to learn a basis and backfitting the weights afterwards maintains good out of sample performance.

Algorithm 9 vvMultipleMatchingPursuit($\mathcal{D}$, $\{\mathcal{G}\}$, δ_{tol}, $d_{\max}$, ℓ_{backfit})

Input: Data $\mathcal{D} := \{\mathbf{X}, \mathbf{Y}\}$, $\mathbf{X} := [\mathbf{x}_1, ..., \mathbf{x}_n]^\top$, $\mathbf{Y} := [\mathbf{y}_1, ..., \mathbf{y}_n]^\top$, $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$, a collection of child basis dictionaries $\{\mathcal{G}\} := \{\mathcal{G}^1, \mathcal{G}^2, ...\}$, mp loss tolerance δ_{tol}, mp maximum basis count $d_{\max}$, backfit weights after every ℓ_{backfit} bases have been added.

Output: Regularised regressor $\mathbf{W}$, new feature basis $\mathcal{B} \subset \mathcal{G}^*$ from the best child dictionary $\mathcal{G}^* \in \{\mathcal{G}\}$ selected from all child dictionaries $\{\mathcal{G}\}$.

Initialise: $\epsilon_{\text{bestTest}} \leftarrow \infty$, $n_{\text{test}} \leftarrow n/n_{\text{folds}}$, $n_{\text{folds}} \leftarrow 5$, $\{\mathcal{D}_1, ..., \mathcal{D}_{n_{\text{folds}}}\} \leftarrow \mathcal{D}$.

for each $\mathcal{G} \in \{\mathcal{G}\}$ **do**

  $\epsilon_{\text{sumTest}} \leftarrow 0$.

  for $j = 1$ **to** n_{folds} **do**

    $\mathcal{D}_{\text{train}} \leftarrow (\dot{\bigcup}_{i=1}^{n_{\text{folds}}} \mathcal{D}_i)_{i \neq j}$ ▷ $\mathcal{D}_{\text{train}} := \{\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}\}$

    $\mathcal{D}_{\text{test}} \leftarrow \mathcal{D}_j$ ▷ $\mathcal{D}_{\text{test}} := \{\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}\}$

    $\mathcal{B}_{\text{train}}, \mathbf{W}_{\text{train}} \leftarrow$ vvMatchingPursuit($\mathcal{D}_{\text{train}}$, δ_{tol}, $d_{\max}$, $\mathcal{G}$, ℓ_{backfit}) ▷ algorithm 10

    $\epsilon_{\text{sumTest}} \leftarrow \epsilon_{\text{sumTest}} + \frac{1}{2n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} || \sum_{\ell=1}^{|\mathcal{B}_{\text{train}}|} \mathbf{w}_\ell g_\ell(\mathbf{x}_i) - \mathbf{y}_i ||_{\text{Fr}}^2$

  end for

  $\epsilon_{\text{avTest}} \leftarrow \epsilon_{\text{sumTest}}/n_{\text{folds}}$

  if $\epsilon_{\text{avTest}} < \epsilon_{\text{bestTest}}$ **then** ▷ compare estimate of out-of-sample error

    $\mathcal{G}^* \leftarrow \mathcal{G}$, $\epsilon_{\text{bestTest}} \leftarrow \epsilon_{\text{avTest}}$.

  end if

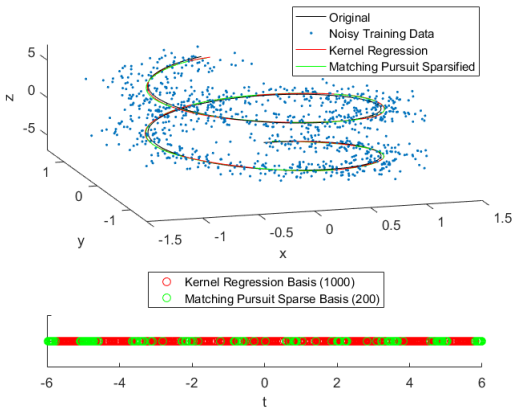
end for

$\mathcal{B}, \mathbf{W} \leftarrow$ vvMatchingPursuit($\{\mathbf{X}, \mathbf{Y}\}$, δ_{tol}, $d_{\max}$, $\mathcal{G}^*$, ℓ_{backfit}) ▷ algorithm 10

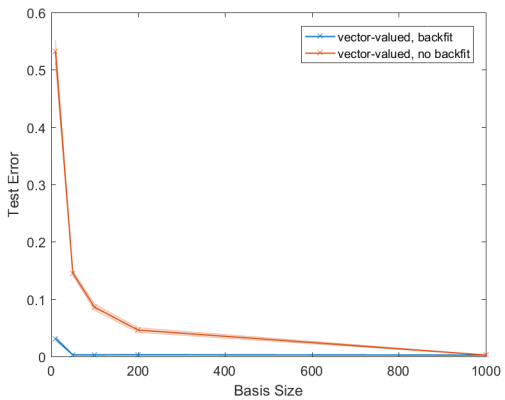
return $\mathcal{B}$, $\mathbf{W}$

Figure 4.1 Kernel regression and matching pursuit regression for a vector-valued function. Test errors are also shown for limiting the basis size to 10, 50, 100, 200 and 1000 (full batch regression on all training points). Centres for of the kernel regression full basis (of size 1000 and coloured red) and matching pursuit sparse basis (of size 200 and coloured green) are also plotted against t.

(a) $[x, y, z] = [\sin(t), \cos(t), t]$



(b) vector-valued test errors



Algorithm 10 vvMatchingPursuit($\mathcal{D}$, δ_{tol}, $d_{\max}$, $\mathcal{G}$, ℓ_{backfit})

Input: Data $\mathcal{D} := \{\mathbf{X}, \mathbf{Y}\}$, $\mathbf{X} := [\mathbf{x}_1, ..., \mathbf{x}_n]^\top$, $\mathbf{Y} := [\mathbf{y}_1, ..., \mathbf{y}_n]^\top$, $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$, loss tolerance δ_{tol}, maximum basis count $d_{\max}$, dictionary $\mathcal{G} := \{g_1, ..., g_m\}$, backfit weights after every ℓ_{backfit} bases have been added.

Output: Basis $\mathcal{B} := \{g_\ell\}_{\ell=1}^{d} \subset \mathcal{G}$, basis weights $\mathbf{W} := [\mathbf{w}_1, ..., \mathbf{w}_d]$, $d \leq d_{\max}$.

Initialise: $\mathcal{B} \leftarrow \emptyset$, $\mathbf{W} \leftarrow \emptyset$, $\ell \leftarrow 0$, $\delta \leftarrow \infty$, $\mathbf{R} \leftarrow \mathbf{Y}$, $\mathbf{\Psi}_{\mathcal{B}} \leftarrow \emptyset$, $\mathbf{G} \leftarrow \{\mathbf{g}_i\}_{i=1}^{m}$ where $\mathbf{g}_i := [g_i(\mathbf{x}_1), ..., g_i(\mathbf{x}_n)]^\top \in \mathbb{R}^n$.

while $\ell < d_{\max}$ **or** $\delta > \delta_{\text{tol}}$ **do**
  $\mathbf{M} \leftarrow \mathbf{R}\mathbf{R}^\top$, $\epsilon_{\min} \leftarrow \infty$, $g_{\text{next}} \leftarrow \emptyset$, $\mathbf{g}_{\text{next}} \leftarrow \emptyset$. ▷ $\mathbf{R} := [\mathbf{r}_1, ..., \mathbf{r}_n]^\top \in \mathbb{R}^{n \times \dim(\mathcal{Y})}$
  for each $\mathbf{g} \in \mathbf{G}$ **do**
    $\epsilon \leftarrow \frac{\mathbf{g}^\top \mathbf{M} \mathbf{g}}{\mathbf{g}^\top \mathbf{g}}$ ▷ equation (4.4)
    if $\epsilon < \epsilon_{\min}$ **then**
      $g_{\text{next}} \leftarrow g$ ▷ basis element g is associated with vector $\mathbf{g}$
      $\mathbf{g}_{\text{next}} \leftarrow \mathbf{g}$, $\epsilon_{\min} \leftarrow \epsilon$.
    end if
  end for
  $\mathcal{B} \leftarrow \mathcal{B} \cup g_{\text{next}}$, $\mathbf{G} \leftarrow \mathbf{G} \backslash \mathbf{g}_{\text{next}}$, $\mathbf{\Psi}_{\mathcal{B}} \leftarrow \mathbf{\Psi}_{\mathcal{B}} \cup \mathbf{g}_{\text{next}}$. ▷ $\mathbf{\Psi}_{\mathcal{B}} := [\mathbf{g}_1, \mathbf{g}_2, ...] \in \mathbb{R}^{n \times |\mathcal{B}|}$
  $\mathbf{w}^{\text{next}} \leftarrow \frac{\mathbf{R}^\top \mathbf{g}_{\text{next}}}{\mathbf{g}_{\text{next}}^\top \mathbf{g}_{\text{next}}}$, $\mathbf{W} \leftarrow \mathbf{W} \cup \mathbf{w}^{\text{next}}$ ▷ equation (4.3)
  $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{g}_{\text{next}} \mathbf{w}_{\text{next}}^\top$ ▷ update residues
  $\delta \leftarrow \frac{1}{2n} \sum_{i=1}^{n} ||\mathbf{r}_i||_{\mathcal{Y}}^2$ ▷ update error i.e. $\frac{1}{2n} ||\mathbf{R}||_{\text{Fr}}^2$
  $\ell \leftarrow \ell + 1$
  if $\text{mod}(\ell, \ell_{\text{backfit}}) = 0$ **then**
    $\mathcal{D} := \{\mathbf{\Psi}_{\mathcal{B}}, \mathbf{Y}\}$
    $\mathbf{W}^\top \leftarrow$ vvRegression-Primal($\mathcal{D}$) ▷ backfit weights
  end if
end while

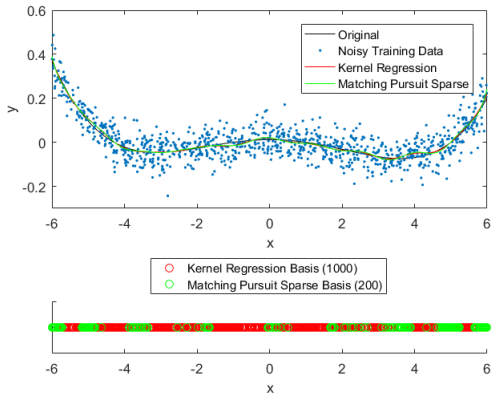
return $\mathcal{B}$, $\mathbf{W}$

4.2 PCMEs: Parametric Embeddings with Greedy Feature Selection

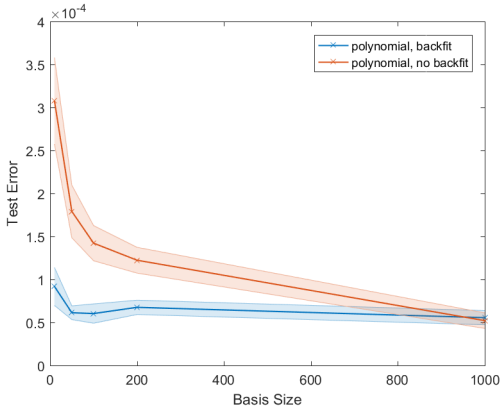
A parametric conditional mean embedding (PCME) is developed below and is considered a primal variant of the CME non-parametric embedding. PCME maintains explicit feature representations in order to avoid scaling cubically with the training set while simultaneously attempting to recover the RKHS embedding performance. The PCME is then deployed in an explorative parametric-PI algorithm which can be directly compared to explorative pseudo-PI. This approach can also be considered as a variant of the LAM algorithm as detailed in section 2.6.2. However LAMs are based only on state feature representations and LAM models are maintained separately for each $\mathbf{a} \in \mathcal{A}$ which may scale unfavourably for large $\mathcal{A}$. In addition they were only shown to work when training data is made available at the beginning of the algorithm

Figure 4.2 Kernel regression and Matching Pursuit regression for various scalar-valued functions. Test errors are also shown for limiting the basis size to 10, 50, 100, 200 and 1000 (full batch regression on all training points). Centres for of the kernel regression full basis (of size 1000 and coloured red) and matching pursuit sparse basis (of size 200 and coloured green) are also plotted along the x-axis.

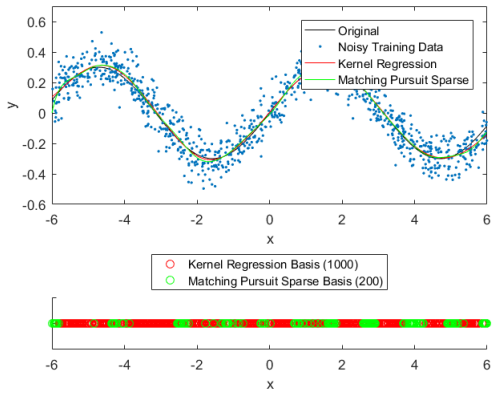
(a) $y = 0.001(x^2 - 1)(0.7x + 3)(0.8x - 4)$



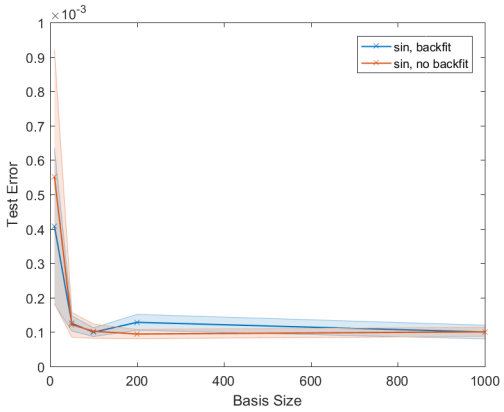
(b) Polynomial test errors



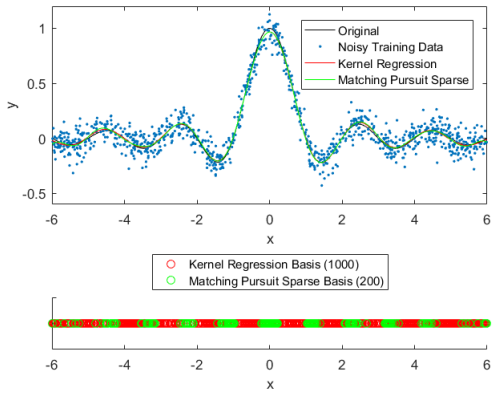
(c) $y = \sin(x)$



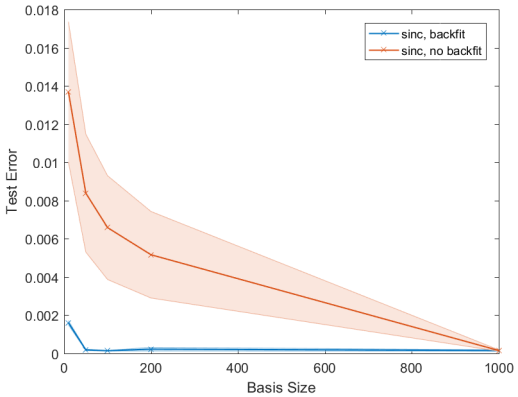
(d) sin test errors



(e) $y = \text{sinc}(x)$



(f) sinc test errors



and feature learning was not explored. Instead a PCME attempts to learn both state and state-action features for the model-fitting procedure. A direct comparison with LAM and PCME is left for future investigation.

Parametric value functions require solving the projected Bellman equation using LSTD or BRM (see section 2.5.2) as opposed to the CME. The explicit state feature representation $\boldsymbol{\phi}: \mathcal{S} \to \mathcal{F}_{\boldsymbol{\phi}}$ is first given a priori such that transition model construction can be developed. Particular attention is made to learning the state-action representation $\boldsymbol{\psi}: \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\psi}}$ using vector-valued matching pursuit developed in the previous section. It is hypothesised that sharing a feature representation over $\mathcal{S} \times \mathcal{A}$ exploits more information from the training set than if separate action models are maintained such as for LAMs. Empirical evidence is provided for i) policy instabilities hypothesised due to parametric value function approximation and ii) their mitigation with conservative policy updates. In the final section of this chapter, an attempt is made to apply matching pursuit to learn the state features of the parametric value function with limited success.

4.2.1 Algorithm Details

Described below are all the components of PCME as described in algorithm 11. Access to the reward function is assumed in order to concentrate developing the transition model learner.

Data Acquisition

Data acquisition is identical to pseudo-PI using a mixture of on-policy data and exploration data (which is a noisy version of the current policy). Trajectory data $\mathcal{D}_{\text{new}}$ from both policies consist of multiple state-action-successor state triples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. At each k^{th} iteration all data is aggregated to form $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}}$.

Model Learning

PCME model learning[1] is a direct response to the cubic complexity of the original CME's model construction method. Recall that the conditional expectation

$$\mathbb{E}_{S'|\mathbf{s},\mathbf{a}}[v(S')] = \langle v, \mu_{S'|\mathbf{s},\mathbf{a}} \rangle_{\mathcal{F}},$$

is estimated by a non-parametric CME if $\mathcal{F}$ is chosen as an RKHS $\mathcal{H}_L$ of value functions $v: \mathcal{S} \to \mathbb{R}$ and the embedding $\mu: \mathcal{S} \times \mathcal{A} \to \mathcal{H}_L$ is modelled in a vvRKHS $\mu \in \mathcal{H}_{\boldsymbol{\Gamma}}$. PCME's embedding $\mu: \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\phi}}$ is a composition $\mu(\cdot) = \mathbf{W}\boldsymbol{\psi}(\cdot)$ where $\boldsymbol{\psi}: \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\psi}}$ and linear map $\mathbf{W}: \mathcal{F}_{\boldsymbol{\psi}} \to \mathcal{F}_{\boldsymbol{\phi}}$. Explicit feature representations $\mathcal{F}_{\boldsymbol{\phi}} = \mathbb{R}^m$ and $\mathcal{F}_{\boldsymbol{\psi}} = \mathbb{R}^d$ are chosen such that $\mathbf{W} \in = \mathbb{R}^{m \times d}$. The model construction procedure assumes both feature representations are known, such that the loss (c.f. CME loss (2.53) and

[1] Note that the feature representation component is executed before model learning at each k^{th} policy iteration.

regression between feature spaces in Grünewälder et al. (2012a, equation 19)) is

$$\begin{aligned}\mathcal{L}(\mu) &:= \mathbb{E}_{(S,A)\sim D,\, S'\sim P(\cdot|S,A)}\Big[||\boldsymbol{\phi}(S') - \mu(S,A)||^2_{\mathcal{F}_{\boldsymbol{\phi}}}\Big],\\ &= \mathbb{E}_{(S,A)\sim D,\, S'\sim P(\cdot|S,A)}\Big[||\boldsymbol{\phi}(S') - \mathbf{W}\boldsymbol{\psi}(S,A)||^2_{\mathcal{F}_{\boldsymbol{\phi}}}\Big],\end{aligned} \tag{4.9}$$

where $\mathbf{W}$ is to be found and D is some state-action data distribution. Given $\boldsymbol{\psi}^k$ and $\boldsymbol{\phi}^k$ at the k^{th} policy iteration[2], the empirical penalised risk minimisation problem is therefore

$$\begin{aligned}\hat{\mathbf{W}}_k &= \operatorname*{arg\,min}_{\mathbf{W}\in\mathbb{R}^{m\times d}}\left[\frac{1}{2n_k}\sum_{i=1}^{n_k}||\boldsymbol{\phi}^k(\mathbf{s}'_i) - \mathbf{W}\boldsymbol{\psi}^k(\mathbf{s}_i,\mathbf{a}_i)||_2^2 + \frac{\lambda}{2}||\mathbf{W}||^2_{\mathrm{Fr}}\right],\\ &= \operatorname*{arg\,min}_{\mathbf{W}\in\mathbb{R}^{m\times d}}\left[\frac{1}{2n_k}||\boldsymbol{\Phi}^k - \boldsymbol{\Psi}^k\mathbf{W}^\top||^2_{\mathrm{Fr}} + \frac{\lambda}{2}||\mathbf{W}||^2_{\mathrm{Fr}}\right],\\ &= {\boldsymbol{\Phi}^k}^\top\boldsymbol{\Psi}^k({\boldsymbol{\Psi}^k}^\top\boldsymbol{\Psi}^k + \lambda\mathbf{I}_d)^{-1},\end{aligned}$$

whose batch solution in the last line is for the vector-valued regression problem (see section B.4.4, equation (B.54)). Feature learning is considered a separate component to model learning therefore the training data is viewed as a function of the features evaluated over samples $\mathcal{D}_k := \{\boldsymbol{\psi}^k(\mathbf{s}_i,\mathbf{a}_i), \boldsymbol{\phi}^k(\mathbf{s}'_i)\}_{i=1}^{n_k}$. The training data is arranged into a state-action feature matrix $\boldsymbol{\Psi}^k := [\boldsymbol{\psi}^k(\mathbf{s}_1,\mathbf{a}_1), ..., \boldsymbol{\psi}^k(\mathbf{s}_{n_k},\mathbf{a}_{n_k})]^\top \in \mathbb{R}^{n_k\times d}$ and state feature matrix $\boldsymbol{\Phi}^k := [\boldsymbol{\phi}^k(\mathbf{s'}_1), ..., \boldsymbol{\phi}^k(\mathbf{s'}_{n_k})]^\top \in \mathbb{R}^{n_k\times m}$. Regularisation constant $\lambda \in 2^{\{\log_2(10^{-8}):\log_2(1000)\}}$ was cross-validated using 20 log-linearly spaced values.

If $\boldsymbol{\psi}^k$ is made from a set of basis functions $\mathcal{B}$ then the computational cost of model construction is $\sim\mathcal{O}(d^3)$. An alternative SGD approach on the empirical loss $\hat{\mathcal{L}}$ was investigated. The derivative of the loss evaluated over minibatch $\hat{\mathcal{D}} := \{\boldsymbol{\psi}^k(\mathbf{s}_i,\mathbf{a}_i), \boldsymbol{\phi}^k(\mathbf{s}'_i)\}_{i=1}^{|\hat{\mathcal{D}}|}$ is

$$\nabla_{\mathbf{W}}\hat{\mathcal{L}}\Big|_{\hat{\mathcal{D}}} = \frac{1}{|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|}\Big(\mathbf{W}_k\boldsymbol{\psi}^k(\mathbf{s}_i,\mathbf{a}_i) - \boldsymbol{\phi}^k(\mathbf{s}'_i)\Big){\boldsymbol{\psi}^k}^\top(\mathbf{s}_i,\mathbf{a}_i) + \lambda\mathbf{W}_k,$$

which can be used in an SGD minimisation scheme (see section B.70),

$$\mathbf{W} \leftarrow \mathbf{W} - \eta\nabla_{\mathbf{W}}\hat{\mathcal{L}}\Big|_{\hat{\mathcal{D}}},$$

where the learning rate η obeys the Robbins-Monro conditions (B.71) and minibatches are drawn i.i.d. from the entire dataset $\hat{\mathcal{D}} \sim \mathcal{D}_k$. The complexity of each online sweep over $|\hat{\mathcal{D}}|$ samples scales $\sim\mathcal{O}(md|\hat{\mathcal{D}}|)$.

A second model type is explored that learns the 'delta' dynamics (Deisenroth and Rasmussen, 2011) such that

$$\mu_{S'|\mathbf{s},\mathbf{a}} = \boldsymbol{\phi}(\mathbf{s}) + \mu^*(\mathbf{s},\mathbf{a}),$$

[2]The dimensionality of each feature representation $m = \dim(\boldsymbol{\phi}^k)$ and $d = \dim(\boldsymbol{\psi}^k)$ are subject to change throughout exploration of an MDP.

where $\mu^*(\mathbf{s},\mathbf{a}) := \mathbb{E}_{S'|\mathbf{s},\mathbf{a}}\Big[\boldsymbol{\delta}(\mathbf{s},S')\Big]$ and $\boldsymbol{\delta}(\mathbf{s},S') := \boldsymbol{\phi}(S') - \boldsymbol{\phi}(\mathbf{s})$. As before by assuming $(S,A)\sim D$ and $S'\sim P(\cdot|S,A)$ then the ideal vector-valued loss is

$$\begin{aligned}\mathcal{L}(\mu^*) &:= \mathbb{E}_{(S,A)\sim D,\, S'\sim P(\cdot|S,A)}\Big[||\boldsymbol{\delta}(S,S') - \mu^*(S,A)||_2^2\Big],\\ &= \mathbb{E}_{(S,A)\sim D,\, S'\sim P(\cdot|S,A)}\Big[||\boldsymbol{\delta}(S,S') - \mathbf{W}\boldsymbol{\psi}(S,A)||_2^2\Big],\end{aligned}$$

A vector-valued regression approach with dataset $\mathcal{D}_k := \{\boldsymbol{\psi}^k(\mathbf{s}_i,\mathbf{a}_i), \boldsymbol{\delta}^k(\mathbf{s}_i,\mathbf{s}_i')\}_{i=1}^{n_k}$ is

$$\begin{aligned}\hat{\mathbf{W}}_k &= \underset{\mathbf{W}\in\mathbb{R}^{m\times d}}{\arg\min}\left[\frac{1}{2n_k}||\boldsymbol{\Delta}^k - \boldsymbol{\Psi}^k\mathbf{W}^\top||_{\text{Fr}}^2 + \frac{\lambda}{2}||\mathbf{W}||_{\text{Fr}}^2\right],\\ &= {\boldsymbol{\Delta}^k}^\top\boldsymbol{\Psi}^k({\boldsymbol{\Psi}^k}^\top\boldsymbol{\Psi}^k + \lambda\mathbf{I}_{D'})^{-1},\end{aligned}$$

where $\boldsymbol{\Delta}^k := [\boldsymbol{\delta}^k(\mathbf{s}_1,\mathbf{s}_1'),...,\boldsymbol{\delta}^k(\mathbf{s}_{n_k},\mathbf{s}_{n_k}')]^\top$. The final expectation estimate at the k^{th} policy iteration is

$$\hat{\mu}_{S'|\mathbf{s},\mathbf{a}} = \boldsymbol{\phi}^k(\mathbf{s}) + \hat{\mathbf{W}}_k\boldsymbol{\psi}^k(\mathbf{s},\mathbf{a}).$$

Greedy Feature Learning

The feature learning component uses the vector-valued matching pursuit algorithm developed in section 4.1 to not only build but also maintain features in a data-driven way throughout exploration of an MDP. The following explanations do not assume a delta transition model, however it is trivial to adapt them for this case if required. Both lemma 4 and its parent algorithm 9 form the core feature learning process.

$\boldsymbol{\psi}$ Features The following is a description of algorithm 16 which seeks to find $\boldsymbol{\psi}^k$ in the k^{th} policy iteration. The existing feature mapping $\boldsymbol{\psi}^{k-1}(\cdot)$ is made from a basis of kernels $\mathcal{B}_{k-1} = \{K_{\sigma_{k-1}}(\tilde{\mathbf{b}}_\ell,\cdot)\}_{\ell=1}^{|\mathcal{B}_{k-1}|}$ where $\tilde{\mathbf{b}}\in\mathcal{S}\times\mathcal{A}$. In the general case each kernel basis function $K_\sigma : (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A})\to\mathbb{R}$ could have different hyperparameters σ, but in this case all hyperparameters are restricted to be identical throughout each basis for any policy iteration. Hyperparameters may change between each policy iteration. Clearly $\mathcal{B}_0$ is an empty set at the beginning of the algorithm, otherwise the ℓ^{th} basis element is defined as $\psi_\ell^{k-1}(\cdot) := K_{\sigma_{k-1}}(\tilde{\mathbf{b}}_\ell,\cdot)$ such that a feature mapping over state-actions is defined as $\boldsymbol{\psi}^{k-1}(\cdot) = [\psi_1^{k-1}(\cdot),...\psi_{|\mathcal{B}_{k-1}|}^{k-1}(\cdot)]^\top$.

Feature learning proceeds by augmenting the previous state-action basis with newly acquired data $\{\mathbf{b}_i\}_{i=1}^{n_{\text{new}}} := \{(\mathbf{s},\mathbf{a})_i\}_{i=1}^{n_{\text{new}}}$ such that $\mathcal{G}\leftarrow\mathcal{B}_{k-1}\bigcup\{K_{\sigma_{k-1}}(\mathbf{b}_i,\cdot)\}_{i=1}^{n_{\text{new}}}$. A new dictionary of candidate features $\mathcal{G}^\sigma \leftarrow \{K_\sigma(\hat{\mathbf{b}}_i,\cdot)\}_{i=1}^{|\mathcal{G}|}$ are created for each $\sigma\in\{0.01,..,5\}$ and collected in $\{\mathcal{G}\} := \{\mathcal{G}^{\sigma_1},\mathcal{G}^{\sigma_2},...\}$ that will be cross-validated over. For quick updates (specified for certain policy iterations a priori), only one dictionary with the previous bandwidth is used $\{\mathcal{G}\} := \{\mathcal{G}^{\sigma_{\text{prev}}}\}$ such that cross-validation is not needed.

For any one dictionary $\mathcal{G}^\sigma \in \{\mathcal{G}\}$, then the usual matching pursuit procedure is carried out as described by lemma 4. After the d^{th} basis function $\psi_\ell(\cdot) = K(\mathbf{b}_\ell, \cdot)$ has been added, the vector-valued *model residue* (c.f. equation (4.2)) evaluated at a transition sample $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is

$$\mathbf{r}^d(\mathbf{s}, \mathbf{a}, \mathbf{s}') := \boldsymbol{\phi}^{k-1}(\mathbf{s}') - \sum_{\ell=1}^{d} \mathbf{w}_\ell \psi_\ell(\mathbf{s}, \mathbf{a}) \in \mathcal{F}_{\boldsymbol{\phi}^{k-1}}.$$

By drawing a subset of sample transitions $\mathcal{D}_{\text{eval}} \sim \mathcal{D}_k$ then the empirical loss (c.f. equation (4.5)) for enriching the feature representation by one more feature basis function, taken without replacement from the dictionary $\psi_{d+1}(\cdot) \in \mathcal{G}^\sigma$ is

$$\begin{aligned} \mathcal{L}(\mathbf{r}^{d+1}) &:= \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||\mathbf{r}_i^{d+1}||^2_{\mathcal{F}_{\boldsymbol{\phi}^{k-1}}}, \\ &= \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||\mathbf{r}_i^{d} - \mathbf{w}_{d+1}\psi_{d+1}(\mathbf{s}_i, \mathbf{a}_i)||^2_{\mathcal{F}_{\boldsymbol{\phi}^{k-1}}}, \quad \mathbf{w}_{d+1} \in \mathcal{F}_{\boldsymbol{\phi}^{k-1}}, \end{aligned} \tag{4.10}$$

where $\mathbf{r}_i^d := \mathbf{r}^d(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$. A closed form solution exists for $\mathbf{w}_{d+1}$ (equation (4.3)) and equation (4.4) is the condition for greedily selecting ψ_{d+1} from $\mathcal{G}^\sigma$. The loss is reduced at each step $\mathcal{L}(\mathbf{r}^{d+1}) \leq \mathcal{L}(\mathbf{r}^d)$ by sequentially enriching the $\boldsymbol{\psi}$-representation, repeating this process $\ell_{\text{new}} := d_{\max} - d$ more times (where $d_{\max}$ is the upper limit on the total number of basis functions allowed for the compact representation). The new basis $\mathcal{B}_k := \{\psi_\ell(\cdot) = K(\mathbf{x}_\ell, \cdot)\}_{\ell=1}^{d_{\max}} \subseteq \mathcal{G}^\sigma$ and weights $\{\mathbf{w}_\ell \in \mathcal{F}_{\boldsymbol{\phi}^{k-1}}\}_{\ell=1}^{d_{\max}}$ selected by matching pursuit defines the PCME

$$\hat{\mu}^k_{S'|\cdot} = \mathbf{W}_k \boldsymbol{\psi}^k(\cdot) := \sum_{\ell=1}^{d_{\max}} \mathbf{w}_\ell \psi_\ell(\cdot),$$

where $\mathbf{W} := [\mathbf{w}_1, .., \mathbf{w}_{d_{\max}}]$, $\boldsymbol{\psi}^k = [\psi_1(\cdot), .., \psi_{d_{\max}}(\cdot)]^\top$, $\mathbf{w}_i \in \mathcal{F}_{\boldsymbol{\phi}^{k-1}}$, $\dim(\mathcal{F}_{\boldsymbol{\psi}^k}) = d_{\max}$. In this way $d_{\max}$ added features costs $\sim \mathcal{O}(d_{\max}\,|\mathcal{G}^\sigma|\,m\,|\mathcal{D}_{\text{eval}}|)$, where $m = \dim(\mathcal{F}_{\boldsymbol{\phi}^{k-1}})$.

It is entirely possible to over-fit the regressor if the number of features begin to exceed the number of evaluation samples. Back-fitting with regularised vector-valued regression (algorithm 25) is used to recalculate the matrix $\mathbf{W}_k$ after $\boldsymbol{\psi}_k$ has been learnt. In effect the weights learnt during matching pursuit are discarded in favour of the final regularised weights. If over-fitting is a concern *during* matching pursuit, then the frequency of back-fitting is increased, up to a maximum of backfitting immediately after every new basis function has been added. In practice this is not needed and only one backfitting procedure is required after $d_{\max}$ basis functions have been added.

Experiments were carried out by populating only one child dictionary with a range of randomly allocated bandwidths, motivated by the idea that a representation could be learnt at various scales throughout the state-action space. However it was found that cross-validating to choose only one kernel bandwidth for the whole regressor was

entirely adequate and is consistent with the theoretical underpinnings if the regressor is viewed as a sparsified single-kernel RKHS function as in equation (B.67). Future work could compare the out-of-sample performance of the regressor when features are learnt with dictionaries of one bandwidth or many. One avenue for future investigation could be to test whether multiple-bandwidth dictionaries induce overfitting and constraining them to one bandwidth induces better generalisation?

The approach is general and the dictionary could include arbitrary real-valued functions. This method is therefore adaptive in two senses: firstly, by selecting new features only if the loss (4.10) is reduced so that the complexity of the representation adapts to the problem; secondly the feature representation is in terms of kernel functions defined at state-actions that have been discovered. This can be an advantage in RL since a good feature representation can be difficult to choose a priori.

$\boldsymbol{\phi}$ Features The following is a description of algorithm 17 which seeks to find the state representation $\boldsymbol{\phi}^k$ in the k^{th} policy iteration. Perhaps one of the defining challenges of parametric approximate policy iteration is learning a good state feature representation. Note that state feature representation is implicit in non-parametric CMEs and is enriched by additional training data. In comparison, the PCME approach must maintain a compact representation. Many parametric value prediction and control algorithms assume $\boldsymbol{\phi}$ is given a priori, e.g. for LAM policy iteration algorithm (Yao, 2011). On the contrary, this investigation is interested in maintaining the state representation *online* as is done for $\boldsymbol{\psi}$.

The existing feature mapping $\boldsymbol{\phi}^{k-1}(\cdot)$ is formed from a basis $\mathcal{C}_{k-1} = \{L_{\sigma_{k-1}}(\tilde{\mathbf{c}}_j, \cdot)\}_{j=1}^{\mathcal{C}_{k-1}}$ where $\tilde{\mathbf{c}} \in \mathcal{S}$. In the general case each kernel basis function $L_\sigma : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ could have different hyperparameters σ, but in this case all hyperparameters are restricted to be identical throughout each basis for any policy iteration. Hyperparameters may change between each policy iteration. Clearly $\mathcal{C}_0$ is an empty set at the beginning of the algorithm, otherwise the j^{th} basis element is defined as $\phi^{k-1}(\mathbf{c}_j) := L_{\sigma_{k-1}}(\mathbf{c}_j, \cdot)$ such that a feature mapping over states is defined as $\boldsymbol{\phi}^{k-1}(\cdot) = [\phi^{k-1}(\mathbf{c}_1), ...\phi^{k-1}(\mathbf{c}_{|\mathcal{C}_{k-1}|})]^\top$.

Feature learning proceeds by the augmenting the previous state-action basis with newly acquired data $\{\mathbf{c}_i\}_{i=1}^{n_{\text{new}}} := \{\mathbf{s}'_i\}_{i=1}^{n_{\text{new}}}$ such that $\mathcal{G} \leftarrow \mathcal{C}_{k-1} \bigcup \{L_{\sigma_{k-1}}(\mathbf{c}_i, \cdot)\}_{i=1}^{n_{\text{new}}}$. A new dictionary of candidate features $\mathcal{G}^\sigma \leftarrow \{L_\sigma(\hat{\mathbf{c}}_i, \cdot)\}_{i=1}^{|\mathcal{G}|}$ is created for each $\sigma \in \{0.01, .., 5\}$ and collected in $\{\mathcal{G}\} := \{\mathcal{G}^{\sigma_1}, \mathcal{G}^{\sigma_2}, ...\}$ that will be cross-validated over. For a quick update, only one dictionary with the previous bandwidth is used $\{\mathcal{G}\} := \{\mathcal{G}^{\sigma_{\text{prev}}}\}$ such that cross-validation is not needed.

Experimental results presented in the following section provide evidence that constant tampering of the successor representation can cause policy improvement instabilities. Therefore in the first set of experiments, the basis $\mathcal{C}_k$ is made constant throughout all iterations and given a priori in order to focus on embedding construction.

This is practical for small MDPs such as the cart-pole and mountain car where $\mathcal{S} \subset \mathbb{R}^2$. In these cases the static basis is $\mathcal{C}_k := \{L(\mathbf{c}_j, \cdot)\}_{j=1}^{225}$, created by a grid of 15×15 linearly spaced points in $\mathbb{R}^2$. For mountain car the grid is generated in the range $[-1.2, 0.7] \times [-0.07, 0.07]$ and for cart-pole the discretisation is in $[-\pi, \pi] \times [-4\pi, 4\pi]$.

However the *curse of dimensionality* inhibits the use of the same discretisation technique in large MDPs. In order to cover the state space of large MDPs, the basis size is exponential in the dimensionality of the state space and other approaches must be developed. The second set of experiments therefore use a matching pursuit procedure to maintain a data-driven compact state representation. This is the same approach as for the state-action representation but with a objective function that defines a scalar residue. The quadrocopter navigation task MDP is introduced where $\mathcal{S} \subset \mathbb{R}^{13}$ which is clearly impractical for a discretised grid and requires the matching pursuit procedure.

At the k^{th} policy iteration, the residue after the m^{th} base has been added is the (scalar-valued) model-based *Bellman residual*, evaluated at a transition sample $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$,

$$R^m(\mathbf{s}, \pi_k(\mathbf{s})) := r(\mathbf{s}, \pi_k(\mathbf{s})) + \gamma \Big(\mathbf{W}_{k-1} \boldsymbol{\psi}^k(\mathbf{s}, \pi_k(\mathbf{s})) \Big)^\top \mathbf{w}^{\pi_{k-1}} - \sum_{j=1}^{m} \phi_j(\mathbf{s}) w_j \ \in \mathbb{R},$$

where $R^d(\mathbf{s}, \pi_k(\mathbf{s})) \in \mathbb{R}$ and $\mathbf{w}^{\pi_{k-1}} \in \mathcal{F}_{\boldsymbol{\phi}^{k-1}}$ are the fitted value function weights from the previous iteration (see below for value function fitting). By drawing a data subset $\mathcal{D}_{\text{eval}} \sim \mathcal{D}_k$ then the empirical loss for enriching the feature representation by one more feature basis function, taken without replacement from the dictionary $\boldsymbol{\phi}_{m+1} \in \mathcal{G}^\sigma$ is

$$\begin{aligned} \mathcal{L}(R^{d+1}) &:= \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||R_i^{d+1}||_2^2, \\ &= \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||R_i^d - \phi_{d+1}(\mathbf{s}_i) w_{d+1}||_2^2, \quad w_{d+1} \in \mathbb{R}, \end{aligned} \tag{4.11}$$

where $R_i^d := R^d(\mathbf{s}_i, \pi_k(\mathbf{s}_i))$. A closed form solution exists for w_{d+1} (equation (4.3)) and equation (4.4) is the condition for greedily selecting $\phi_{d+1}^k \in \mathcal{G}^\sigma$ without replacement. The loss is reduced at each step $\mathcal{L}(R^{d+1}) \leq \mathcal{L}(R^d)$ by sequentially enriching the $\boldsymbol{\phi}$-representation, repeating this process $j_{\text{new}} := m_{\max} - m$ more times (where $m_{\max}$ is the upper limit on the total number of bases). The set of features $\{\phi_j(\cdot)\}_{j=1}^{m_{\max}} \subseteq \mathcal{G}^\sigma$ and weights $\{w_j \in \mathbb{R}\}_{j=1}^{m_{\max}}$ selected by matching pursuit augments the value function which takes a final form

$$\sum_{j=1}^{m_{\max}} w_j \phi_j(\cdot) = \boldsymbol{\phi}^k(\cdot)^\top \mathbf{w}_k,$$

where $\mathbf{w}_k \in \mathcal{F}_{\phi^k}$. In this way all new features are added in time $\sim \mathcal{O}(m_{\max} |\mathcal{G}_k| |\mathcal{D}_{\text{eval}}|)$. In practice $\mathbf{w}_k$ is discarded such that matching pursuit is only used as a feature learner. Instead regularised approximate policy evaluation (LSTD or BRM) is used to backfit $\mathbf{w}^{\pi_k}$ as described below.

Approximate Policy Evaluation

By assuming parametric value function approximation, the PCME model is not a weighted sum of value function evaluations over states as enjoyed by the CME. Therefore policy evaluation is not exact and the value weights must be fitted by solving the Bellman residual or projected Bellman residual as summarised in section 2.5.2. During a policy iteration, once features and transition model have been estimated then the model-based Bellman operator is $(\hat{T}^\pi v)(\cdot) := r(\cdot, \pi(\cdot)) + \gamma(\mathbf{W}_k \boldsymbol{\psi}^k(\cdot, \pi(\cdot)))^\top \mathbf{w}^\pi$ which is used to form the model-based counterparts of the Bellman residual (c.f. BRM equation (2.37))

$$\hat{\mathbf{w}}^\pi = \left(\mathbf{C}^{\pi\top}\mathbf{C}^\pi + \lambda \mathbf{I}_d\right)^{-1} \mathbf{C}^{\pi\top}\mathbf{r}^\pi, \tag{4.12}$$

where $\mathbf{C}^\pi := (\boldsymbol{\Phi} - \gamma_{\text{vfit}} \boldsymbol{\Psi}^\pi \mathbf{W}^\top)$ and projected Bellman residual (c.f. LSTD equation (2.40))

$$\hat{\mathbf{w}}^\pi = \left(\mathbf{A}^\pi + \lambda \mathbf{I}_d\right)^{-1} \boldsymbol{\Phi}^\top \mathbf{r}^\pi, \tag{4.13}$$

where $\mathbf{A}^\pi := \boldsymbol{\Phi}^\top(\boldsymbol{\Phi} - \gamma_{\text{vfit}} \boldsymbol{\Psi}^\pi \mathbf{W}_k^\top)$. Given n samples at policy iteration k, then $\boldsymbol{\Phi} := [\boldsymbol{\phi}^k(\mathbf{s}_1), ..., \boldsymbol{\phi}^k(\mathbf{s}_n)]^\top$, $\boldsymbol{\Psi}^\pi := [\boldsymbol{\psi}^k(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., \boldsymbol{\psi}^k(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top$ and reward vector $\mathbf{r}^\pi := [r(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., r(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top$. The advantage of using the model-based Bellman residual is that it does not suffer from the double successor state sampling problem suffered by the model-free residual (section 2.5.2).

If $n < \dim(\boldsymbol{\phi}^k)$, then $\mathbf{C}^{\pi\top}\mathbf{C}^\pi$ and $\mathbf{A}^\pi$ are not full rank (see section B.4.1 for a discussion conditions for solutions to regression problems) e.g. this may occur early in policy iteration when there is little data available. A regulariser was therefore introduced to mitigate this problem both for LSTD (Kolter and Ng, 2009) and BRM. The regularisation parameter λ was selected using n_{fold} cross-validation algorithm 12. The discount factor was specifically chosen a priori from values $\gamma_{\text{vfit}} \in \{0.96, 0.98, 0.99\}$ for each experiment, which may not be the same as the discount factor defined in the MDP. This was a key factor in getting the algorithm to work and is elaborated upon in the results discussion below.

Conservative Policy Improvement

Policy improvement for non-parametric pseudo-MDP algorithms (such as for the CME) is shown to be very stable in Chapter 3 and needs no other consideration

other than taking a greedy policy. However it was found that this was not the case for PCME and instead with reference to the discussion in section 2.5.3, a softer or conservative greedy policy improvement is shown to improve algorithm performance. PCME develops a version of conservative policy iteration (Kakade and Langford, 2002) and is fundamental for algorithm stability as discussed in the results below. The following description refers to the planning section in algorithm 11.

For the i^{th} policy improvement during policy iteration k, a pure greedy update for the PCME is

$$\begin{aligned}\hat{q}^{\pi_{i,k}}(\mathbf{s},\mathbf{a}) &:= r(\mathbf{s},\mathbf{a}) + \gamma(\mathbf{W}_k\boldsymbol{\psi}^k(\mathbf{s},\mathbf{a}))^\top \hat{\mathbf{w}}^{\pi_{i,k}}, \quad (\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A},\\ \pi_{i+1,k}(\mathbf{s}) &:= \arg\sup_{\mathbf{a}\in\mathcal{A}}[\hat{q}^{\pi_{i,k}}(\mathbf{s},\mathbf{a})].\end{aligned}$$

A smoother *conservative* update redefines the action-value function estimate as a weighted sum of the previous and current action-value functions,

$$\begin{aligned}\underline{\hat{q}}(\mathbf{s},\mathbf{a}) &:= \hat{q}^{\pi_{i-1,k}}(\mathbf{s},\mathbf{a}) + \omega\Big(\hat{q}^{\pi_{i,k}}(\mathbf{s},\mathbf{a}) - \hat{q}^{\pi_{i-1,k}}(\mathbf{s},\mathbf{a})\Big),\\ &= r(\mathbf{s},\mathbf{a}) + \gamma(\mathbf{W}_k\boldsymbol{\psi}^k(\mathbf{s},\mathbf{a}))^\top\Big(\hat{\mathbf{w}}^{\pi_{i-1,k}} + \omega(\hat{\mathbf{w}}^{\pi_{i,k}} - \hat{\mathbf{w}}^{\pi_{i-1,k}})\Big),\\ \pi_{i+1,k}(\mathbf{s}) &:= \arg\sup_{\mathbf{a}\in\mathcal{A}}[\underline{\hat{q}}(\mathbf{s},\mathbf{a})], \quad \mathbf{s}\in\mathcal{S},\end{aligned}$$

where $\omega \in [0,1]$ controls how smooth the greedy update should be, recovering a vanilla greedy policy when $\omega = 1$.

It is important to understand that the *first* $i = 1$ conservative policy improvement in a k^{th} policy iteration depends not only on the previous iteration's embedding $\mathbf{W}_{k-1}\boldsymbol{\psi}^{k-1}(\cdot)$ but also its fitted value weights $\mathbf{w}^{\pi_{k-1}}$ and state features $\boldsymbol{\phi}^{k-1}(\cdot)$. This is problematic when state features are maintained with matching pursuit because the vector $\boldsymbol{\phi}^{k-1}(\cdot)$ will almost certainly be different from $\boldsymbol{\phi}^k(\cdot)$ both in the order of basis functions and feature dimensionality. Conservative updates on the value weights between policy iterations is therefore nonsensical because their elements refer to different feature basis functions. Note that this is *not* an issue for policy improvements $i > 1$. The solution is to limit $\boldsymbol{\phi}$-learning 'full updates' to $k \in \{1,3,5\}$ and impose non-conservative policy improvements ($\omega = 1$) when $i = 1$ and $k \in \{1,3,5\}$ (see from line 15 in algorithm 11). Further comments on this procedure are made in the results discussion. When $\boldsymbol{\phi}(\cdot)$ is provided a priori, this problem is not an issue and therefore conservative policy improvement can be used in every iteration regardless.

Table 4.1 MDP-specific parameters for PCME

	Mountain car	Cart-pole	Quadrocopter Navigation
$\dim(\boldsymbol{\psi}(\cdot)) \leq d_{\max}$	200	500	200
$\dim(\boldsymbol{\phi}(\cdot)) \leq m_{\max}$	225	225	225
state $\sigma_{\mathcal{S}}$ (if a priori)	0.5	0.5	1.25

Algorithm 11 Explorative Parametric-PI with Conservative π Updates

Input: Access to unknown MDP $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, P, P_1, r, \gamma\}$ with continuous $\mathcal{S}$, discrete $\mathcal{A}$, known average reward function $r : \mathcal{S}\times\mathcal{A} \to [0,1]$, $H = 100$ transitions per trajectory, $J_{\text{imp}} = 10$, start-state distribution P_1, state-action Gaussian kernel $K_\sigma : (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A}) \to \mathbb{R}$, state Gaussian kernel $L_\sigma : \mathcal{S}\times\mathcal{S} \to \mathbb{R}$ (both with unspecified bandwidths σ). For value function fitting choose discount factor $\gamma_{\text{vfit}} \in \{0.96, 0.98, 0.99\}$ and conservative update $\omega \in \{0.1, 0.4, 1\}$.

Output: Deterministic and discrete policy $\pi_\kappa(\cdot) \approx \pi^*(\cdot)$.

Initialise: $\hat{q}_0(\cdot, \mathbf{a}) \leftarrow r(\cdot, \mathbf{a})$, $\pi_1(\cdot) \leftarrow \text{greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}_0(\cdot, \mathbf{a})]$, exploration policy $\tilde{\pi}_1(\cdot) \leftarrow \epsilon\text{-greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}_0(\cdot, \mathbf{a})]$, $\epsilon \leftarrow 0.3$, $n_0 \leftarrow 0$, $n_{\text{new}} \leftarrow 2H$, $\mathcal{D}_0 \leftarrow \emptyset$, model $\mathbf{W}_0 \leftarrow \emptyset$, $\boldsymbol{\psi}^0(\cdot) \leftarrow \emptyset$, $\boldsymbol{\phi}^0(\cdot) \leftarrow \emptyset$, $fullUpdates := \{1, 2, 5\}$, $\hat{\mathbf{w}}^{\pi_0} \leftarrow \emptyset$.

for $k = 1, 2, ..., \kappa{-}1$ **do** ▷ k^{th} policy iteration master index

  $isFullUpdate \leftarrow$ **false** ▷ partial updates possible for some components

  if $k \in fullUpdates$ **then**

    $isFullUpdate \leftarrow$ **true**

  end if

  Data acquisition: $n_k \leftarrow n_{k\text{-}1} + n_{\text{new}}$, collect $\mathcal{D}_{\text{new}} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_k}$; one trajectory from each π_k and ν^{π_k}, beginning from start state $\mathbf{s} \sim P_1$, $\mathcal{D}_k \leftarrow \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}}$, $\mathcal{S}'_{\text{new}} \leftarrow \{\mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_k}$, $\mathcal{Z}_{\text{new}} \leftarrow \{(\mathbf{s}, \mathbf{a})_i\}_{i=n_{k\text{-}1}+1}^{n_k}$. ▷ aggregate data

  ψ**-Features**:
  $(\boldsymbol{\psi}^k, \mathbf{W}_{k-1})$ ▷ algorithm 16
  $\leftarrow \textsc{LearnPsi}\Big(\boldsymbol{\phi}^{k-1}, \boldsymbol{\psi}^{k-1}, \mathbf{W}_{k-1}, r, \mathcal{D}_k, \mathcal{Z}_{\text{new}} K, isFullUpdate\Big)$

  $\boldsymbol{\phi}$**-Features**:
  $(\boldsymbol{\phi}^k, \mathbf{W}_{k-1})$ ▷ algorithm 17
  $\leftarrow \textsc{LearnPhi}\Big(\boldsymbol{\phi}^{k-1}, \boldsymbol{\psi}^k, \mathbf{W}_{k-1}, r, \mathcal{D}_k, \mathcal{S}'_{\text{new}}, L, isFullUpdate, \pi_k, \hat{\mathbf{w}}^{\pi_{k-1}}\Big)$

  Model: $\mathbf{W}_k \leftarrow \textsc{BatchTrain}\Big(\boldsymbol{\phi}^k, \boldsymbol{\psi}^k, \mathcal{D}_k\Big)$. ▷ algorithm 15

  Planning:

  $\pi \leftarrow \pi_k$, $\hat{\mathbf{w}}^\pi_{i=0} \leftarrow \hat{\mathbf{w}}^{\pi_{k-1}}$

  for $i = 1$ **to** J_{imp} **do** ▷ policy improvement index

    $\hat{\mathbf{w}}^\pi_i \leftarrow \textsc{FitValueFunc}\Big(\gamma_{\text{vfit}}, r, \pi, \boldsymbol{\phi}^k, \boldsymbol{\psi}^k, \mathbf{W}_k, \mathcal{D}_k\Big)$ ▷ algorithm 13 or 14

    $\underline{\hat{\mathbf{w}}}^\pi_i \leftarrow \hat{\mathbf{w}}^\pi_{i-1} + \omega(\hat{\mathbf{w}}^\pi_i - \hat{\mathbf{w}}^\pi_{i-1})$ ▷ conservative update

    $\underline{\hat{q}}^\pi(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \gamma\Big(\mathbf{W}_k \boldsymbol{\psi}^k(\mathbf{s}, \mathbf{a})\Big)^\top \underline{\hat{\mathbf{w}}}^\pi_i, \ \forall (\mathbf{s}, \mathbf{a}) \in \mathcal{S}\times\mathcal{A}$

    $\pi(\cdot) \leftarrow \text{greedy}_{\mathbf{a}\in\mathcal{A}}[\underline{\hat{q}}^\pi(\cdot, \mathbf{a})]$ ▷ conservative policy improvement

  end for

  $\pi_{k+1} \leftarrow \pi$, $\hat{\mathbf{w}}^{\pi_k} \leftarrow \underline{\hat{\mathbf{w}}}^\pi_{i=J_{\text{imp}}}$

end for

return $\pi_\kappa(\cdot)$

Algorithm 12 FitValueFunc($\gamma_{\text{vfit}}, r, \pi, \boldsymbol{\phi}, \boldsymbol{\psi}, \mathbf{W}, \mathcal{D}$)

Input: Previous policy π, state representation $\boldsymbol{\phi}(\cdot)$, state-action representation $\boldsymbol{\psi}(\cdot)$, regressor $\mathbf{W}$, data $\mathcal{D}$.
Initialise: Regularisation parameters $\{\lambda\} \leftarrow \{1\times10^{-7}, ..., 1\}$, $n_{\text{folds}} \leftarrow 5$, $\{\mathcal{D}_1, ..., \mathcal{D}_{n_{\text{folds}}}\} \leftarrow \mathcal{D}$, $\epsilon_{\text{bestTest}} \leftarrow \infty$.
Cross-validate regulariser:
for each $\lambda \in \{\lambda\}$ **do**
    $\epsilon_{\text{sumTest}} \leftarrow 0$
    for $k = 1$ **to** n_{folds} **do**
        $\mathcal{D}_{\text{train}} \leftarrow (\dot{\bigcup}_{i=1}^{n_{\text{folds}}} \mathcal{D}_i)_{i \neq k}$
        $\mathcal{D}_{\text{test}} \leftarrow \mathcal{D}_k$
        $\hat{\mathbf{w}}^\pi \leftarrow \text{BRM}\Big(\boldsymbol{\Phi}_{\text{train}}, \boldsymbol{\Psi}^\pi_{\text{train}}, \mathbf{W}, \mathbf{r}^\pi_{\text{train}}, \lambda, \gamma_{\text{vfit}}\Big)$. ▷ algorithm 13 or 14
        $\epsilon_{\text{sumTest}} \leftarrow \epsilon_{\text{sumTest}} + \frac{1}{2n_{\text{test}}} ||\mathbf{r}^\pi_{\text{test}} + \gamma_{\text{vfit}} \boldsymbol{\Psi}^\pi_{\text{test}} \mathbf{W}^\top \hat{\mathbf{w}}^\pi - \boldsymbol{\Phi}_{\text{test}} \hat{\mathbf{w}}^\pi||_2^2$
    end for
    $\epsilon_{\text{avTest}} \leftarrow \epsilon_{\text{sumTest}} / n_{\text{folds}}$
    if $\epsilon_{\text{avTest}} < \epsilon_{\text{bestTest}}$ **then** ▷ compare estimate of out-of-sample error
        $\lambda^* \leftarrow \lambda$, $\epsilon_{\text{bestTest}} \leftarrow \epsilon_{\text{avTest}}$.
    end if
end for
$\hat{\mathbf{w}}^\pi \leftarrow \text{BRM}\Big(\boldsymbol{\Phi}, \boldsymbol{\Psi}^\pi, \mathbf{W}, \mathbf{r}^\pi, \lambda, \gamma_{\text{vfit}}\Big)$. ▷ algorithm 13 or 14
return $\pi_\kappa(\cdot)$

Algorithm 13 BRM($\boldsymbol{\Phi}$, $\boldsymbol{\Psi}^\pi$, $\mathbf{W}$, $\mathbf{r}^\pi$, λ, γ_{vfit})

Input: $\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{s}_1), ..., \boldsymbol{\phi}(\mathbf{s}_n)]^\top$, $\boldsymbol{\Psi}^\pi := [\boldsymbol{\psi}(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., \boldsymbol{\psi}(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top$, $\mathbf{r}^\pi := [r(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., r(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top$
Initialise: $d \leftarrow \dim(\boldsymbol{\phi})$, $\mathbf{C}^\pi \leftarrow (\boldsymbol{\Phi} - \gamma_{\text{vfit}} \boldsymbol{\Psi}^\pi \mathbf{W}^\top)$.
$\hat{\mathbf{w}}^\pi \leftarrow \Big(\mathbf{C}^{\pi\top} \mathbf{C}^\pi + \lambda \mathbf{I}_d\Big)^{-1} \mathbf{C}^{\pi\top} \mathbf{r}^\pi$ ▷ equation (4.12)
return $\mathbf{w}^\pi$

Algorithm 14 LSTD($\boldsymbol{\Phi}$, $\boldsymbol{\Psi}^\pi$, $\mathbf{W}$, $\mathbf{r}^\pi$, λ γ_{vfit})

Input: $\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{s}_1), ..., \boldsymbol{\phi}(\mathbf{s}_n)]^\top$, $\boldsymbol{\Psi}^\pi := [\boldsymbol{\psi}(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., \boldsymbol{\psi}(\mathbf{s}_n, \pi(\mathbf{s}_n))]^\top$.
Initialise: $d \leftarrow \dim(\boldsymbol{\phi})$.
$\hat{\mathbf{w}}^\pi \leftarrow \Big(\boldsymbol{\Phi}^\top (\boldsymbol{\Phi} - \gamma_{\text{vfit}} \boldsymbol{\Psi}^\pi \mathbf{W}^\top) + \lambda \mathbf{I}_d\Big)^{-1} \boldsymbol{\Phi}^\top \mathbf{r}^\pi$ ▷ equation (4.13)
return $\mathbf{w}^\pi$

Algorithm 15 BatchTrain($\boldsymbol{\phi}$,$\boldsymbol{\psi}$,$\mathcal{D}$)

Input: State feature representation $\boldsymbol{\phi}(\cdot)$, state-action feature representation $\boldsymbol{\psi}(\cdot)$, data $\mathcal{D}$.

Output: Regularised regressor $\mathbf{W}$.

Initialise: $\boldsymbol{\Psi} \leftarrow [\boldsymbol{\psi}(\mathbf{s}_1, \mathbf{a}_1), ..., \boldsymbol{\psi}(\mathbf{s}_n, \mathbf{a}_n)]^\top$, $\boldsymbol{\Phi}' \leftarrow [\boldsymbol{\phi}(\mathbf{s}'_1), ..., \boldsymbol{\phi}(\mathbf{s}'_n)]^\top$, redefine data as $\mathcal{D}_{\boldsymbol{\Psi},\boldsymbol{\Phi}} := \{\boldsymbol{\Psi}, \boldsymbol{\Phi}'\}$. If delta transition model then $\mathcal{D}_{\boldsymbol{\Psi},\boldsymbol{\Phi}} := \{\boldsymbol{\Psi}, \boldsymbol{\Delta}\}$.

$\mathbf{W} \leftarrow$ vvRegression-Primal($\mathcal{D}_{\boldsymbol{\Psi},\boldsymbol{\Phi}}$). ▷ algorithm 25

return $\mathbf{W}$.

Algorithm 16 LearnPsi($\boldsymbol{\phi}, \boldsymbol{\psi}, \mathbf{W}, r, \mathcal{D}, \mathcal{Z}_{\text{new}}, K, isFullUpdate$)

Input: Previous $\boldsymbol{\phi}(\cdot)$ and $\boldsymbol{\psi}(\cdot)$ representations, previous transition model regressor $\mathbf{W}$, average immediate reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, transition data $\mathcal{D}$, newly acquired state-actions $\mathcal{Z}_{\text{new}}$, state-action Gaussian kernel $K : (\mathcal{S} \times \mathcal{A}) \times (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$, a boolean *isFullUpdate* specifying quick or full updates.

Output: New state-action feature representation $\boldsymbol{\psi}(\cdot)$, new regressor $\mathbf{W}$.

Initialise: Bandwidth collection $\{\sigma\} := \{0.01, .., 5\}$, collection of basis dictionaries $\{\mathcal{G}\} \leftarrow \emptyset$, $\delta_{\text{tol}} \leftarrow 0$, $\ell_{\text{backfit}} \leftarrow \infty$, $d_{\max} \leftarrow 500$ (MDP-specific).

Prepare dictionaries:

$\mathcal{B}_{\text{prev}} := \{K_{\sigma_{\text{prev}}}(\tilde{\mathbf{b}}_\ell, \cdot)\}_{\ell=1}^{d_{\text{prev}}} \leftarrow \boldsymbol{\psi}(\cdot)$ ▷ extract basis where $\tilde{\mathbf{b}}_\ell \in \mathcal{S} \times \mathcal{A}$

$\mathcal{G} \leftarrow \mathcal{B}_{\text{prev}} \bigcup \{K_{\sigma_{\text{prev}}}(\mathbf{b}_i, \cdot)\}_{i=1}^{n_{\text{new}}}$ ▷ augment candidate functions where $\mathbf{b}_i \in \mathcal{Z}_{\text{new}}$

if *isFullUpdate* **then** ▷ full updates cross-validate over all bandwidths

  for each $\sigma \in \{\sigma\}$ **do**

    $\mathcal{G}^\sigma \leftarrow \{K_\sigma(\hat{\mathbf{b}}_i, \cdot)\}_{i=1}^{|\mathcal{G}|}$ ▷ $\exists K_{\sigma_{\text{prev}}}(\hat{\mathbf{b}}_i, \cdot) \in \mathcal{G}$

    $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \mathcal{G}^\sigma$

  end for

else

  $\mathcal{G}^{\sigma_{\text{prev}}} \leftarrow \{K_{\sigma_{\text{prev}}}(\hat{\mathbf{b}}_i, \cdot)\}_{i=1}^{|\mathcal{G}|}$ ▷ a single child dictionary with σ_{prev}

  $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \mathcal{G}^{\sigma_{\text{prev}}}$.

end if

Prepare residues:

$\mathcal{D}_{\text{eval}} := \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')_i\}_{i=1}^n \sim \mathcal{D}$, $n = |\mathcal{D}_{\text{eval}}|$ ▷ draw evaluation dataset

if $\mathbf{W}$==$\emptyset$ **then**

  $\mathbf{R} \leftarrow [r(\mathbf{s}_1, \mathbf{a}_1), ..., r(\mathbf{s}_n, \mathbf{a}_n)]^\top$ ▷ a proxy residue for the first policy iteration.

else

  $\boldsymbol{\Phi}' \leftarrow [\boldsymbol{\phi}(\mathbf{s}'_1), ..., \boldsymbol{\phi}(\mathbf{s}'_n)]^\top$,

  $\mathbf{R} \leftarrow \boldsymbol{\Phi}'$ ▷ model residue target

end if

$\mathbf{Z} \leftarrow [\mathbf{z}_1, ..., \mathbf{z}_n]^\top$, $\mathbf{z}_i := [\mathbf{s}_i^\top, \mathbf{a}_i^\top]^\top$, $\mathcal{D}_{\text{res}} := \{\mathbf{Z}, \mathbf{R}\}$ ▷ define the residue dataset.

$(\boldsymbol{\psi}(\cdot), \mathbf{W}) \leftarrow$ vvMultipleMatchingPursuit($\mathcal{D}_{\text{res}}$, $\{\mathcal{G}\}$, δ_{tol}, $d_{\max}$, ℓ_{backfit})

return $\boldsymbol{\psi}(\cdot)$, $\mathbf{W}$.

Algorithm 17 LearnPhi($\boldsymbol{\phi}, \boldsymbol{\psi}, \mathbf{W}, r, \mathcal{D}, \mathcal{S}'_{\text{new}}, L, isFullUpdate, \pi, \mathbf{w}^{\pi}$)

Input: Previous $\boldsymbol{\phi}(\cdot)$ and current $\boldsymbol{\psi}(\cdot)$ feature representations, current transition model regressor $\mathbf{W}$, average immediate reward function $r: \mathcal{S}\times\mathcal{A}\to\mathbb{R}$, transition data $\mathcal{D}$, newly acquired successor states $\mathcal{S}'_{\text{new}}$, state Gaussian kernel $L: \mathcal{S}\times\mathcal{S}\to\mathbb{R}$, a boolean *isFullUpdate* specifying quick or full updates, previous policy π, previous fitted weights $\mathbf{w}^{\pi}$.

Output: New state feature representation $\boldsymbol{\phi}(\cdot)$, new regressor $\mathbf{W}$.

Initialise: Bandwidth collection $\{\sigma\} := \{0.01, .., 5\}$, collection of basis dictionaries $\{\mathcal{G}\} \leftarrow \emptyset$, $\delta_{\text{tol}} \leftarrow 0$, $j_{\text{backfit}} \leftarrow \infty$, $m_{\max} \leftarrow 225$ (MDP-specific).

Prepare dictionaries:

$\mathcal{C}_{\text{prev}} := \{L_{\sigma_{\text{prev}}}(\tilde{\mathbf{c}}_j, \cdot)\}_{j=1}^{m_{\text{prev}}} \leftarrow \boldsymbol{\phi}(\cdot)$ ▷ extract basis where $\tilde{\mathbf{c}} \in \mathcal{S}$

$\mathcal{G} \leftarrow \mathcal{C}_{\text{prev}} \bigcup \{L_{\sigma_{\text{prev}}}(\mathbf{c}_i, \cdot)\}_{i=1}^{n_{\text{new}}}$ ▷ augment candidate functions where $\mathbf{c}_i \in \mathcal{S}'_{\text{new}}$

if *isFullUpdate* **then** ▷ full updates cross-validate over all bandwidths.

  for each $\sigma \in \{\sigma\}$ **do**

    $\mathcal{G}^{\sigma} \leftarrow \{L_{\sigma}(\hat{\mathbf{c}}_i, \cdot)\}_{i=1}^{|\mathcal{G}|}$ ▷ $\exists L_{\sigma_{\text{prev}}}(\hat{\mathbf{c}}_i, \cdot) \in \mathcal{G}$.

    $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \tilde{\mathcal{G}}^{\sigma}$

  end for

else

  $\mathcal{G}^{\sigma_{\text{prev}}} \leftarrow \{L_{\sigma_{\text{prev}}}(\hat{\mathbf{c}}_i, \cdot)\}_{i=1}^{|\mathcal{G}|}$ ▷ a single child dictionary with σ_{prev}.

  $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \mathcal{G}^{\sigma_{\text{prev}}}$

end if

Prepare residue:

$\mathcal{D}_{\text{eval}} := \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')_i\}_{i=1}^{n} \sim \mathcal{D}$, $n = |\mathcal{D}_{\text{eval}}|$ ▷ draw evaluation dataset

$\mathbf{r}^{\pi} \leftarrow [r(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., r(\mathbf{s}_n, \pi(\mathbf{s}_n))]^{\top}$

if $\mathbf{W}$==$\emptyset$ **then**

  $\mathbf{R} \leftarrow \mathbf{r}^{\pi}$ ▷ proxy residue for first policy iteration.

else

  $\boldsymbol{\Psi}^{\pi} \leftarrow [\boldsymbol{\psi}(\mathbf{s}_1, \pi(\mathbf{s}_1)), ..., \boldsymbol{\psi}(\mathbf{s}_n, \pi(\mathbf{s}_n))]^{\top}$,

  $\mathbf{R} \leftarrow \mathbf{r}^{\pi} - \gamma \boldsymbol{\Psi}^{\pi} \mathbf{W}^{\top} \mathbf{w}^{\pi}$ ▷ model-based Bellman residual.

end if

$\mathbf{S} \leftarrow [\mathbf{s}_1, ..., \mathbf{s}_n]^{\top}$, $\mathcal{D}_{\text{res}} := \{\mathbf{S}, \mathbf{R}\}$ ▷ define the residue dataset.

$(\boldsymbol{\phi}(\cdot), \mathbf{W}) \leftarrow$vvMultipleMatchingPursuit($\mathcal{D}_{\text{res}}$, $\{\mathcal{G}\}$, δ_{tol}, $m_{\max}$, j_{backfit})

return $\boldsymbol{\phi}(\cdot)$, $\mathbf{W}$.

4.3 Experiments

The parametric-CME experiments include the mountain car, cart-pole and quadrocopter navigation task MDPs as discussed in the previous chapter . Experiments are split into two sets regarding the nature of $\boldsymbol{\phi}(\cdot)$ where i) $\mathcal{C}$ is given a priori with a picked bandwidth (see table 4.1) and ii) both $\mathcal{C}$ and σ are learnt using matching pursuit with cross validation during policy iterations $k \in \{1, 3, 5\}$. Mountain car (fig. 4.3) and cart-pole (fig. 4.4) are presented for fixed $\boldsymbol{\phi}$. Quadrocopter navigation is not presented as the curse of dimensionality inhibits any state space discretisation. Mountain car (fig. 4.5), Cart-pole (fig. 4.6) and Quadrocopter Navigation (fig. 4.7) are presented for the case when $\boldsymbol{\phi}$ is learnt.

In all experiments the following algorithm characteristics are investigated for their effects on policy learning performance; i) delta/vanilla transition models, ii) the strength of greedy policy improvements (the lower ω the more conservative the update) and iii) the variation of γ_{vfit} during the fitting of value weights.

4.4 Discussion

4.4.1 Comparison with non-parametric pseudo MDPs

See section 7.1 for algorithm component timings with comparison of surviving benchmarks from chapter 3 and other algorithms developed in this thesis. PCME clearly achieves an improvement in computational complexity during model construction and planning without losing performance. Creating a viable parametric alternative with explicit feature representations that do not scale with the training set was one of the goals of this investigation. In fact for both mountain car and cart-pole PCME is at least as good at learning policies as the non-parametric algorithms for the benchmark tasks. However there are caveats that mostly relate to learning a state representation and is discussed below.

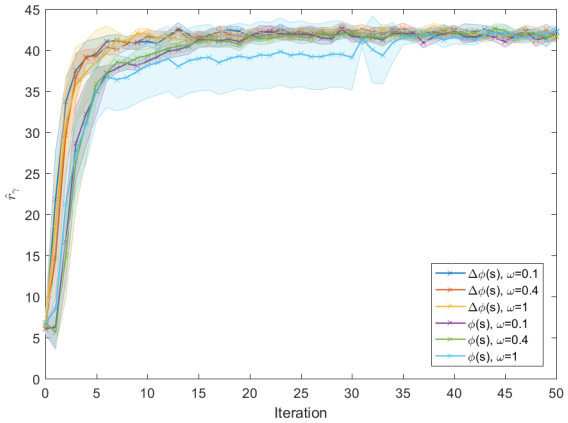
4.4.2 PCME with fixed state representation

One source of instability to PCME is updating the state representation $\boldsymbol{\phi}$. Therefore in the first batch of experiments both σ (table 4.1) and $\mathcal{C}$ (state kernels on a uniform grid over $\mathcal{S}$) are given a priori. The empirical cumulative discounted reward for mountain car and cart-pole are presented in fig. 4.3 and fig. 4.4 respectively. The stability of the PCME algorithm is analysed empirically by altering experiment variables associated with the following choices,

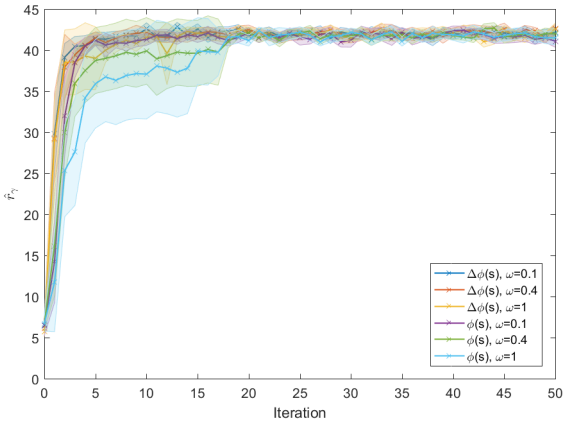
1. LSTD or BRM for fitting the value function.

Figure 4.3 PCME empirical return (mountain car) experiments for **fixed** features $\boldsymbol{\phi}(\mathbf{s})$ or $\boldsymbol{\Delta}\phi(\mathbf{s})$: BRM (left), LSTD (right), γ_{vfit} (top to bottom), conservative update proportion ω.

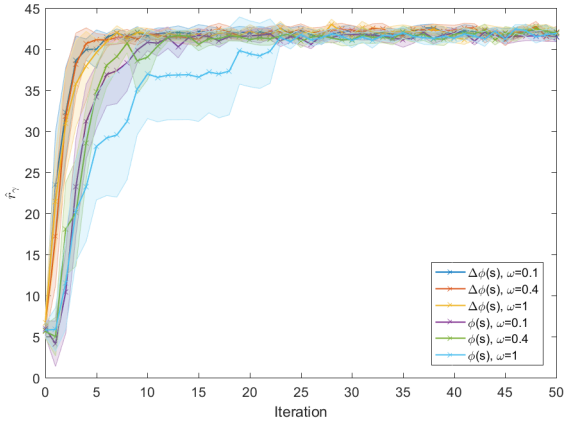
(a) BRM, $\gamma_{\text{vfit}} = 0.96$



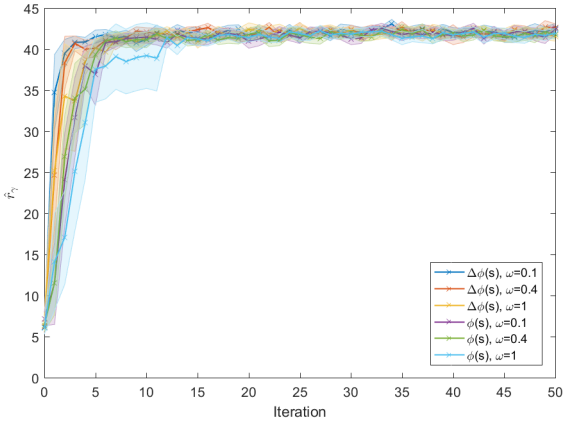
(b) LSTD, $\gamma_{\text{vfit}} = 0.96$



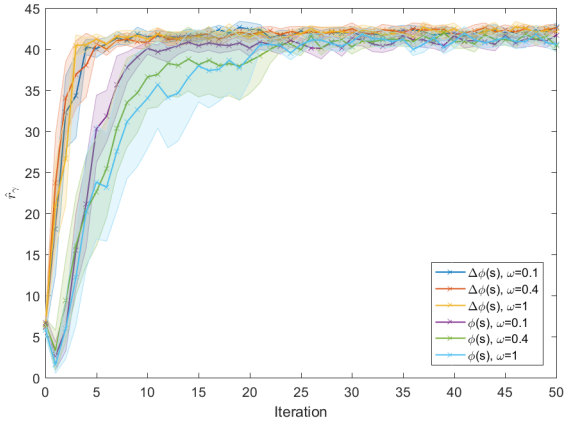
(c) BRM, $\gamma_{\text{vfit}} = 0.98$



(d) LSTD, $\gamma_{\text{vfit}} = 0.98$



(e) BRM, $\gamma_{\text{vfit}} = 0.99$



(f) LSTD, $\gamma_{\text{vfit}} = 0.99$

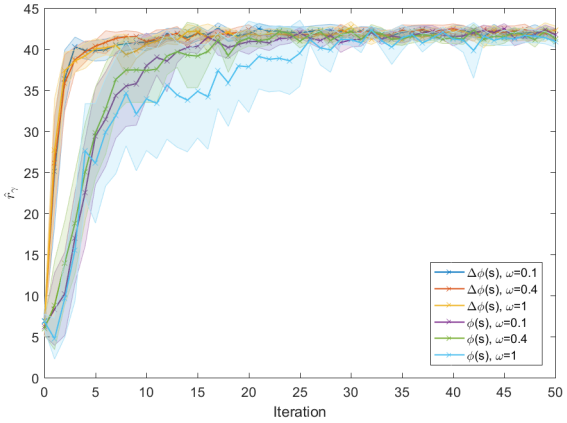
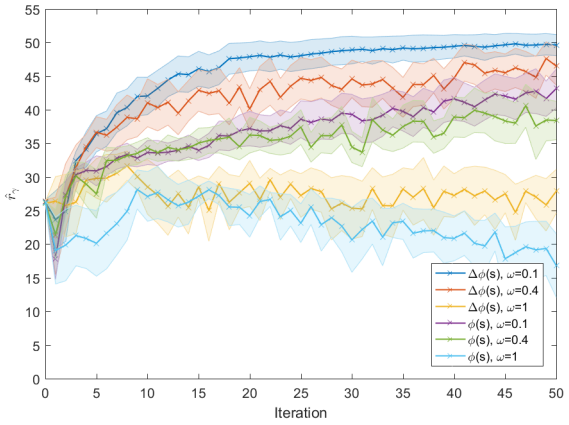
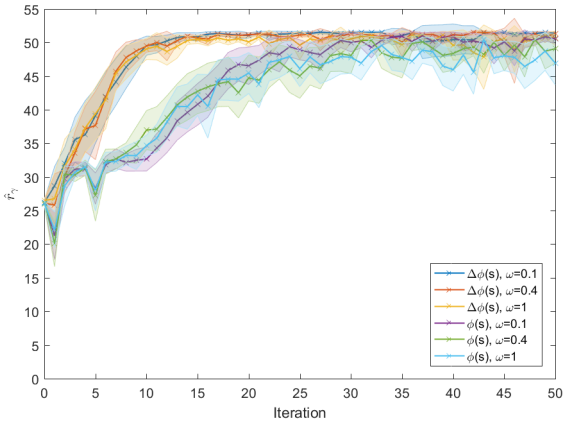


Figure 4.4 PCME empirical return (cart-pole) experiments for **fixed** features $\boldsymbol{\phi}(\mathbf{s})$ or $\boldsymbol{\Delta}\phi(\mathbf{s})$: BRM (left), LSTD (right), γ_{vfit} (top to bottom), conservative update proportion ω.

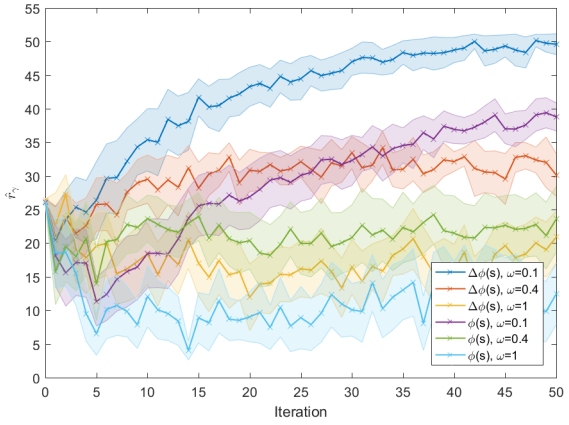
(a) BRM, $\gamma_{\text{vfit}} = 0.96$



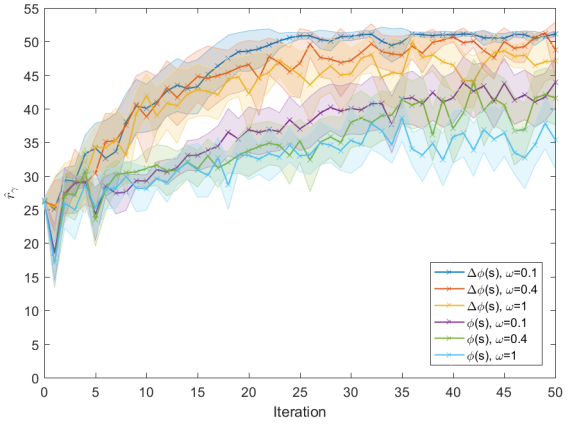
(b) LSTD, $\gamma_{\text{vfit}} = 0.96$



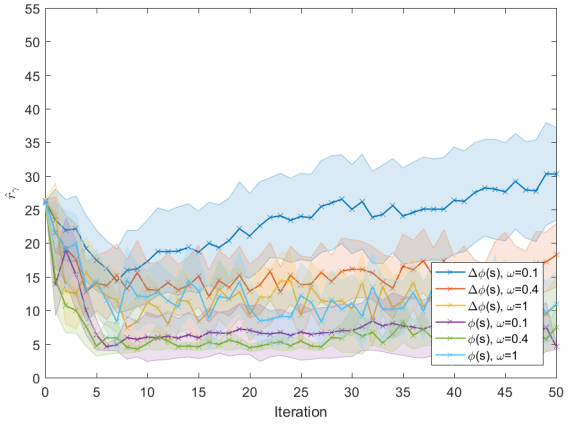
(c) BRM, $\gamma_{\text{vfit}} = 0.98$



(d) LSTD, $\gamma_{\text{vfit}} = 0.98$



(e) BRM, $\gamma_{\text{vfit}} = 0.99$



(f) LSTD, $\gamma_{\text{vfit}} = 0.99$

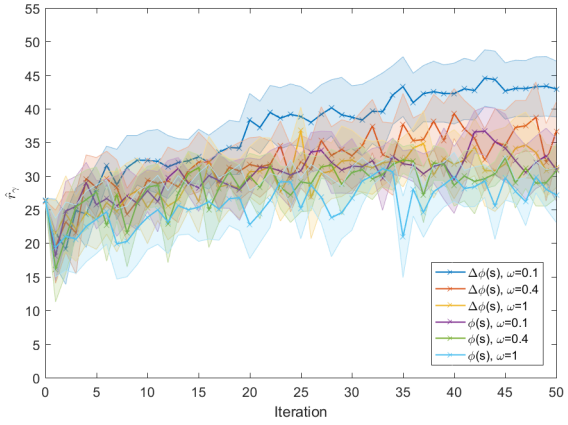
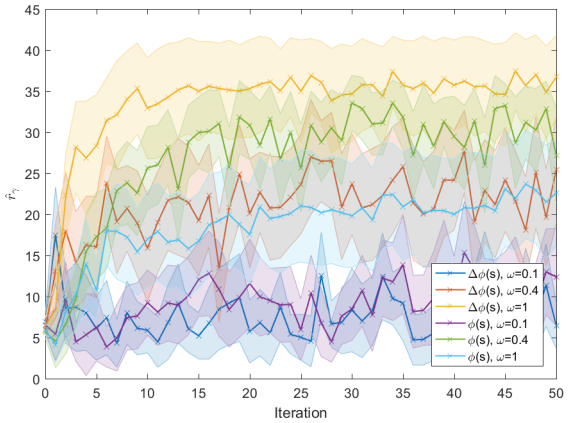
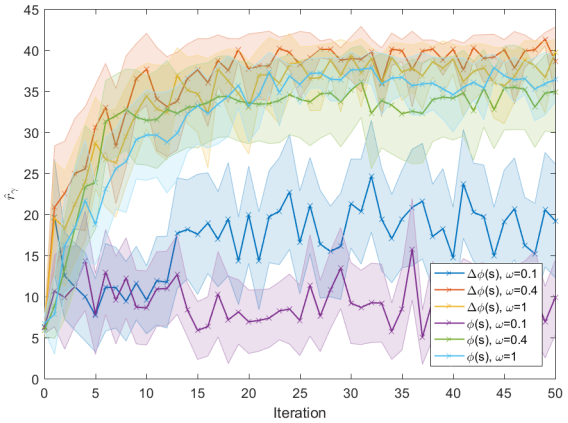


Figure 4.5 PCME empirical return (mountain car) experiments for **learnt** features $\boldsymbol{\phi}(\mathbf{s})$ or $\boldsymbol{\Delta}\phi(\mathbf{s})$: BRM (left), LSTD (right), γ_{vfit} (top to bottom), conservative update proportion ω.

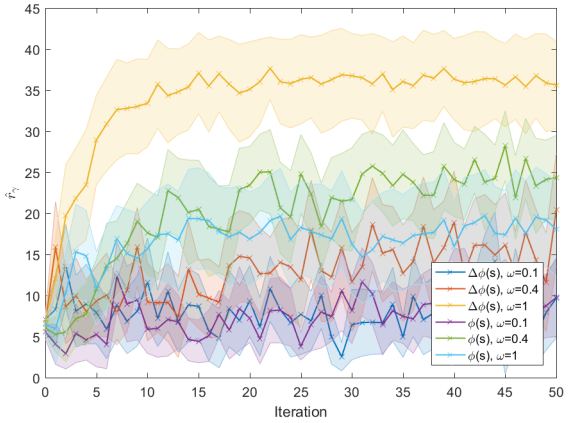
(a) BRM, $\gamma_{\text{vfit}} = 0.96$



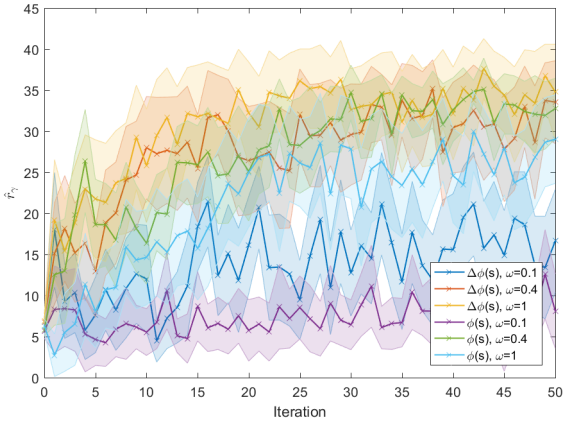
(b) LSTD, $\gamma_{\text{vfit}} = 0.96$



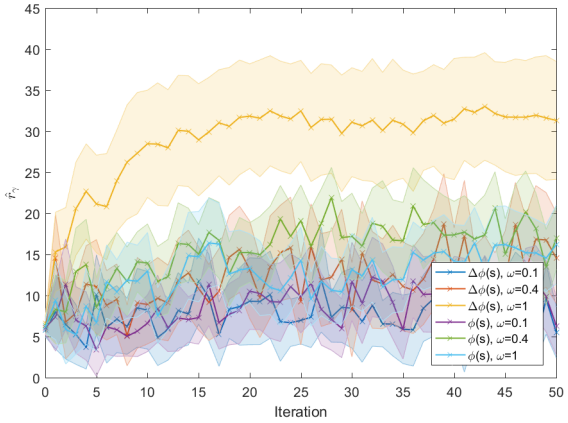
(c) BRM, $\gamma_{\text{vfit}} = 0.98$



(d) LSTD, $\gamma_{\text{vfit}} = 0.98$



(e) BRM, $\gamma_{\text{vfit}} = 0.99$



(f) LSTD, $\gamma_{\text{vfit}} = 0.99$

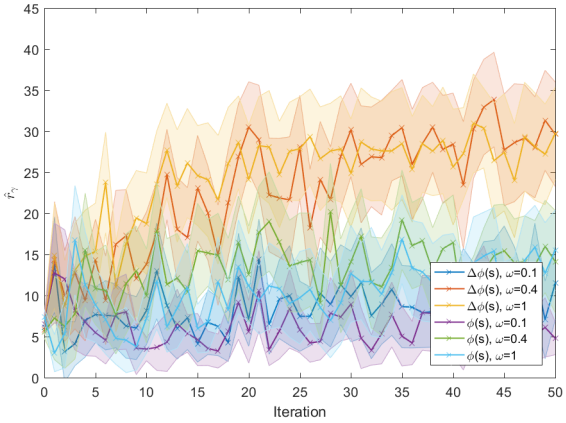
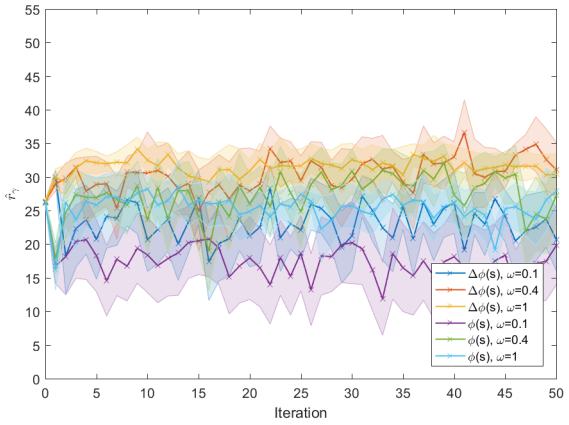
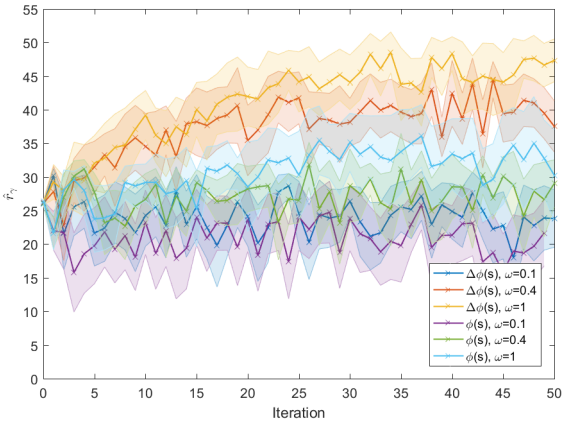


Figure 4.6 PCME empirical return (cart-pole) experiments for **learnt** features $\boldsymbol{\phi}(\mathbf{s})$ or $\boldsymbol{\Delta}\phi(\mathbf{s})$: BRM (left), LSTD (right), γ_{vfit} (top to bottom), conservative update proportion ω.

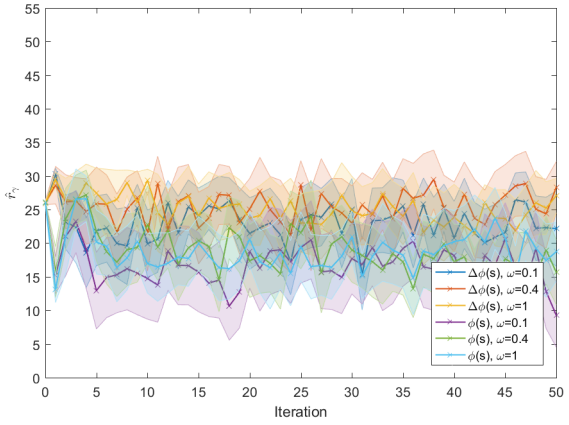
(a) BRM, $\gamma_{\text{vfit}} = 0.96$



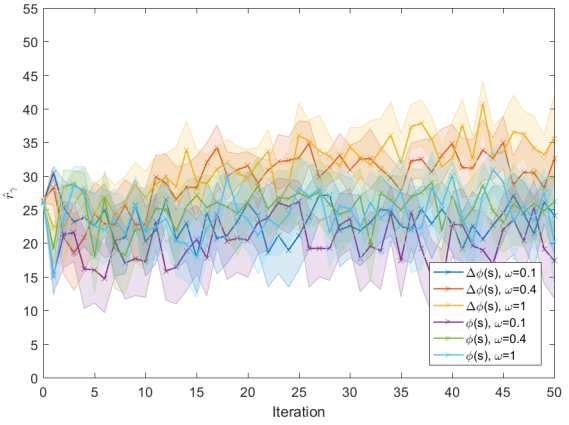
(b) LSTD, $\gamma_{\text{vfit}} = 0.96$



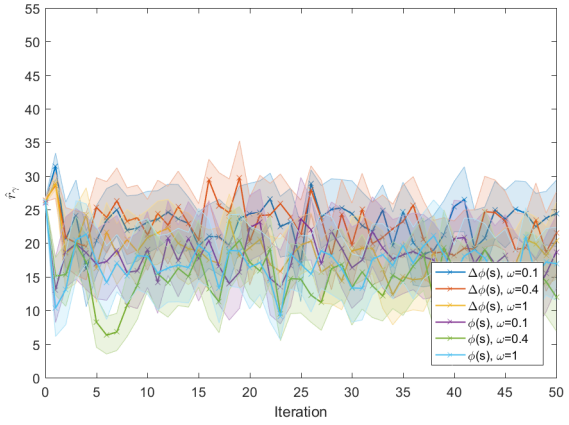
(c) BRM, $\gamma_{\text{vfit}} = 0.98$



(d) LSTD, $\gamma_{\text{vfit}} = 0.98$



(e) BRM, $\gamma_{\text{vfit}} = 0.99$



(f) LSTD, $\gamma_{\text{vfit}} = 0.99$

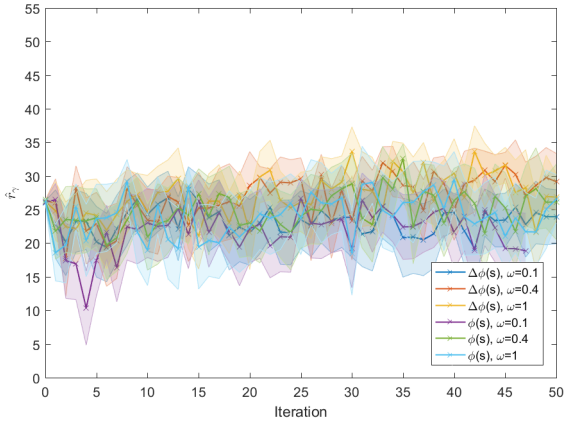
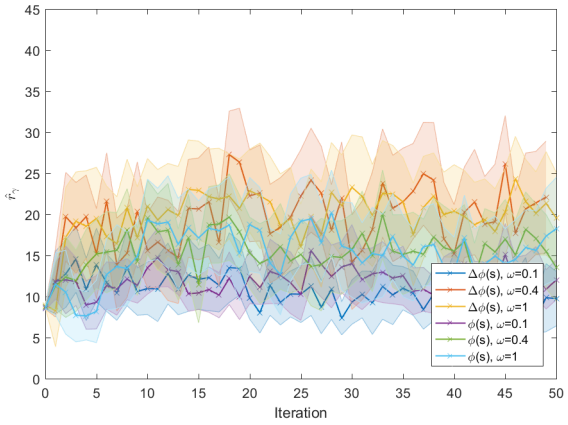
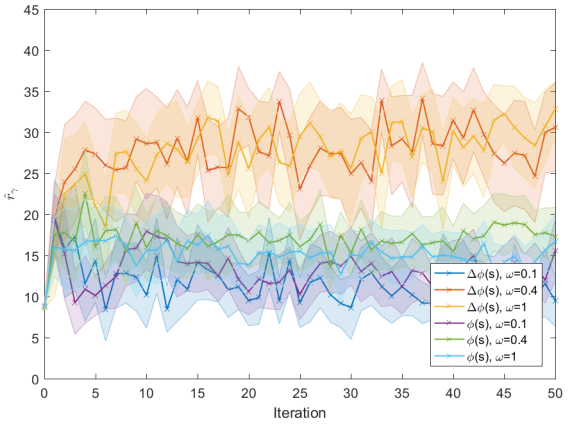


Figure 4.7 PCME empirical return (quadrocopter navigation) experiments for **learnt** features $\boldsymbol{\phi}(\mathbf{s})$ or $\boldsymbol{\Delta}\phi(\mathbf{s})$: BRM (left), LSTD (right), γ_{vfit} (top to bottom), conservative update proportion ω.

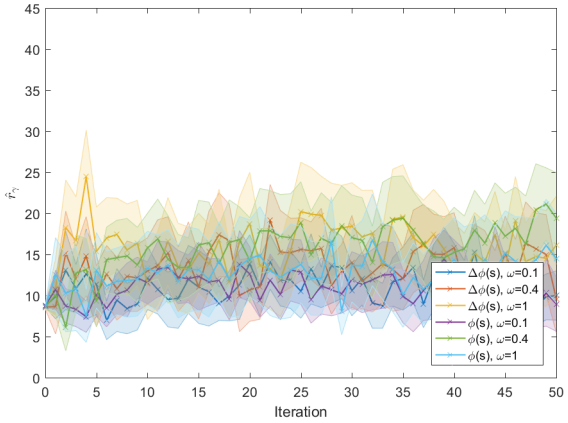
(a) BRM, $\gamma_{\text{vfit}} = 0.96$



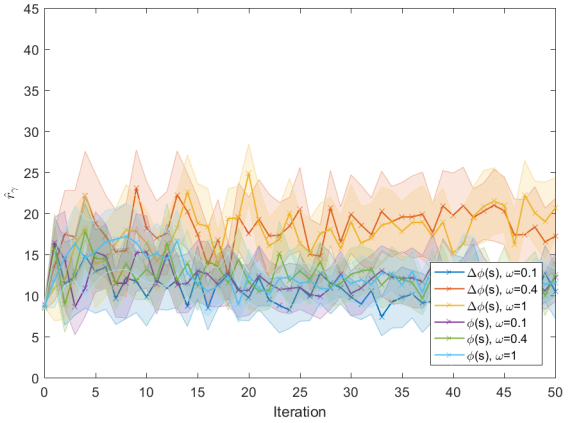
(b) LSTD, $\gamma_{\text{vfit}} = 0.96$



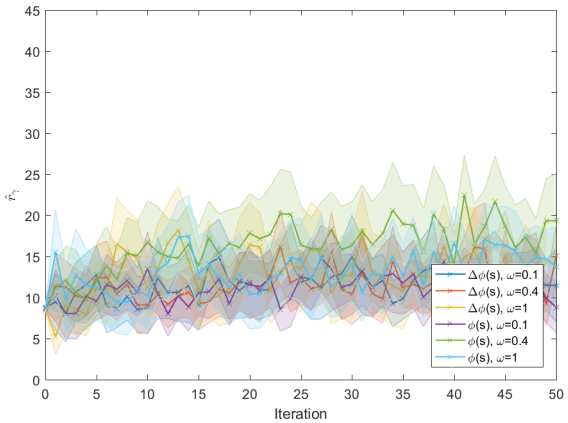
(c) BRM, $\gamma_{\text{vfit}} = 0.98$



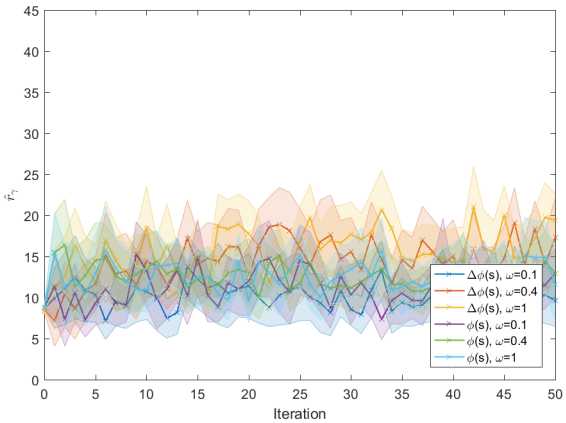
(d) LSTD, $\gamma_{\text{vfit}} = 0.98$



(e) BRM, $\gamma_{\text{vfit}} = 0.99$



(f) LSTD, $\gamma_{\text{vfit}} = 0.99$



2. Delta $\boldsymbol{\Delta\phi}(\mathbf{s})$ or vanilla $\boldsymbol{\phi}(\mathbf{s})$ transition model targets.

3. Conservative policy update proportion ω used in policy improvement.

4. Discount rate γ_{vfit} used in fitting the value function.

The quality of the learning algorithm can be inferred from empirical results by i) the rate at which the return series converges over the course of the experiment, ii) how high this convergent value is and iii) the variance of the returns series. Good learning algorithms generate returns series that converge at high rates to high values with low variance. A returns series with high variance is evidence for policy improvement instability, policy oscillations or in extreme cases catastrophic forgetting.

The most notable observation is that model-based LSTD is more effective than model-based BRM when fitting the value function for all experimental variable combinations. Mountain car is a very simple MDP and therefore differentiating performance between the different combinations of experiment variables is difficult. Cart-pole is a harder MDP, clearly differentiating between good and poor experimental variable settings. LSTD appears to be more agnostic to some experimental settings that can be detrimental to the BRM fitting procedure. The best parameter settings learn good policies in 1000 and 2000 transition samples for the mountain car and cart-pole respectively. LSTD value function fitting not only demonstrates faster learning, but performance variance is also smaller, implying the learning algorithm experiences less policy oscillation.

The delta model consistently out-performs the vanilla model in both MDPs for all combinations of experimental variables. This suggests that models are more effective in generalising over delta transitions (in feature space) rather than over full transitions. Not only is learning faster for the delta models, but performance variance is also lower.

Conservative policy updates are an integral parameter to the success of the PCME approach. In all fixed state representation experiments, conservative updates (by setting $\omega = 0.1$) outperform vanilla greedy updates. The evidence provided here supports what the literature states about unstable policy improvement in approximate policy iteration, and in addition it supports the claim that soft policy improvements mitigate policy oscillations.

The final parameter whose choice is fundamental to the success of the learning algorithm is the discount factor γ_{vfit} used during LSTD or BRM and is the most perplexing. In all experiments the unknown MDP sets $\gamma = 0.99$, however only when $\gamma_{\text{vfit}} = 0.96$ then learning is quick and stable. Nowhere else in the algorithm is the discount factor changed and therefore this phenomena is inextricably linked to fitting the value function. It was investigated to see if γ_{vfit} could be chosen (along with λ) during the cross validation procedure in algorithm 12. However this proved to be

unsuccessful such that the algorithm could not learn good policies. Using $\gamma = 0.99$ in the test error (line 9, algorithm 12) also proved unsuccessful. It was concluded that a validation signal was not detectable in the value fitting procedure in order to select a good γ_{vfit} and instead it was provided a priori. Given that LSTD and BRM are both affected, it would suggest that this parameter requires a deeper explanation that is agnostic to the value function fitting procedure. The approximate Bellman operator used in both cases takes the form

$$(\hat{T}^{\pi} v)(\mathbf{s}) := r(\mathbf{s}, \pi(\mathbf{s})) + \gamma_{\text{vfit}} (\mathbf{W}\boldsymbol{\psi}(\mathbf{s}, \pi(\mathbf{s})))^{\top} \hat{\mathbf{w}}^{\pi},$$

where γ_{vfit} affects the magnitude of the transition dynamics term. Although $\hat{T}^{\pi}$ is not a contraction, a low γ_{vfit} may be enforcing a similar property that enables a better value function fit when a model-based Bellman operator is used. Further investigation is required to understand this phenomenon and it may be linked to reformulating PCME into the pseudo-MDP framework. This would require assuming $\boldsymbol{\phi}$ (either given a priori or learnt) being viewed as the state space as assumed in the LAM algorithm. Then by re-expressing $\boldsymbol{\psi}$ as a function of $\boldsymbol{\phi}$, a contraction constraint could be enforced for the transition dynamics. A naive application of the contraction constraint to the existing PCME was unsuccessful.

4.4.3 PCME with learnt state representation

Learning $\boldsymbol{\phi}$ using matching pursuit means that the state representation will be in constant flux between policy iterations. Therefore as discussed, the conservative update technique developed for the static $\boldsymbol{\phi}$ representation is not always applicable. Executing the correct conservative update when $i = 1$, requires tracking the basis functions of the $\boldsymbol{\phi}$ representation such that ω can be applied to the correct weights. Matching pursuit will greedily select basis functions for $\boldsymbol{\phi}$ and therefore ordering may change, old bases may be discarded and new bases may be added. Basis tracking was not implemented and instead the results presented in figures 4.5, 4.6 and 4.7 use a greedy update whenever a conservative update is nonsensical. Contrary to algorithm 17, $\boldsymbol{\phi}$ was only updated during each $k \in fullUpdates$ and therefore limits the number of non-conservative updates.

Unfortunately using adaptive state representation leads to a considerable performance degradation. However the same observations remain; such as delta models are more effective, in some cases soft conservative updates lead to better performance, LSTD is superior to BRM and $\gamma_{\text{vfit}} = 0.96$ is more effective than using the MDP's $\gamma_{\text{vfit}} = 0.99$.

4.5 Conclusion and future work

For cart-pole and mountain car, PCME successfully matches the performance of the non-parametric CME without the unfavourable computational complexity with training sample size. However the parametric algorithm is less parsimonious and requires additional methods to mitigate instabilities and policy oscillations. The larger quadrocopter MDP exposes PCME's difficulty in learning a good state-feature representation while making conservative weight updates. In order for conservative updates to make sense at the beginning of each policy iteration, basis functions need to be tracked properly such that the correct conservative update can be applied to the correct value weight elements.

4.5.1 Deep PCME

One proposition to resolve this problem is to replace matching pursuit entirely as a feature learning technique. If the dimensionality of the feature representation can be made constant and independent of its 'richness', then conservative policy updates make more sense. Artificial neural networks (see section B.5) are candidate function classes for this property. Beginning with model-based Bellman residual loss for a minibatch drawn from the data $\hat{\mathcal{D}} \sim \mathcal{D}$,

$$\hat{\mathcal{L}}(\hat{\mathscr{E}}, v, \pi)\big|_{\hat{\mathcal{D}}} := \frac{1}{2|\hat{\mathcal{D}}|} \sum_{i=1}^{|\hat{\mathcal{D}}|} \Big(r(\mathbf{s}_i, \pi(\mathbf{s}_i)) + \gamma_{\text{vfit}} \hat{\mathscr{E}}_{(\mathbf{s}_i, \pi(\mathbf{s}_i))}[v] - \hat{v}(\mathbf{s}_i) \Big)^2,$$

then the conditional expectation and value functions can be approximated by neural networks $\mu_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_\phi$ and $\boldsymbol{\phi}_{\boldsymbol{\vartheta}} : \mathcal{S} \to \mathcal{F}_\phi$ (parametrised by $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$ respectively) whose last layers share the same weight vector $\mathbf{w}$,

$$\begin{aligned} \hat{\mathscr{E}}_{(\mathbf{s},\mathbf{a})}[v] &:= \mu_{\boldsymbol{\theta}}^\top(\mathbf{s}_i, \mathbf{a})\mathbf{w}, \\ \hat{v}(\mathbf{s}) &:= \boldsymbol{\phi}_{\boldsymbol{\vartheta}}^\top(\mathbf{s})\mathbf{w}. \end{aligned}$$

Note that the penultimate layer in $\mu_{\boldsymbol{\theta}}$ can be interpreted as the state-action feature representation $\boldsymbol{\psi}$. Given the risk function $V(\mathbf{s}_i, \pi) := \Big(r(\mathbf{s}_i, \pi(\mathbf{s}_i)) + \gamma_{\text{vfit}} \hat{\mathscr{E}}_{(\mathbf{s}_i, \pi(\mathbf{s}_i))}[v] - \hat{v}(\mathbf{s}_i) \Big)^2$ can be evaluated by a forward pass of the function approximators, then by differentiating the loss, the shared last layer can be updated using the gradient signal

$$\begin{aligned} \nabla_{\mathbf{w}} \hat{\mathcal{L}}(\mathbf{w}, \boldsymbol{\vartheta}, \pi)\big|_{\hat{\mathcal{D}}} &= \frac{1}{|\hat{\mathcal{D}}|} \sum_{i=1}^{|\hat{\mathcal{D}}|} V'(\mathbf{s}_i, \pi) \Big(\gamma_{\text{vfit}} \nabla_{\mathbf{w}} \hat{\mathscr{E}}_{(\mathbf{s}_i, \pi(\mathbf{s}_i))}[v] - \nabla_{\mathbf{w}} \hat{v}(\mathbf{s}_i) \Big), \\ &= \frac{1}{|\hat{\mathcal{D}}|} \sum_{i=1}^{|\hat{\mathcal{D}}|} V'(\mathbf{s}_i, \pi) \Big(\gamma_{\text{vfit}} \mu_{\boldsymbol{\theta}}(\mathbf{s}_i, \pi(\mathbf{s}_i)) - \boldsymbol{\phi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i) \Big), \end{aligned} \tag{4.14}$$

where $V'(\mathbf{s}_i, \pi) := r(\mathbf{s}_i, \pi(\mathbf{s}_i)) + \gamma_{\text{vfit}} \hat{\mathscr{E}}_{(\mathbf{s}_i, \pi(\mathbf{s}_i))}[v] - \hat{v}(\mathbf{s}_i)$. Given that μ and $\boldsymbol{\phi}$ are deep networks themselves, the gradient signal can be propagated into their layers, although it is probably prudent to only propagate into $\boldsymbol{\phi}_{\boldsymbol{\vartheta}}$ to update $\vartheta \in \boldsymbol{\vartheta}$ rather than the model $\mu_{\boldsymbol{\theta}}$. After an epoch of minibatches a state-feature representation will have been estimated such that it can be used as targets to drive the training of the parametric transition model. By adapting the existing PCME model loss, then any weight parameter $\theta \in \boldsymbol{\theta}$ can be updated using,

$$\left.\frac{\partial \hat{\mathcal{L}}(\boldsymbol{\theta})}{\partial \theta}\right|_{\hat{\mathcal{D}}} = \frac{1}{|\hat{\mathcal{D}}|} \sum_{i=1}^{|\hat{\mathcal{D}}|} \left(\mu_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i) - \boldsymbol{\phi}_{\boldsymbol{\vartheta}}(\mathbf{s}'_i)\right)^{\top} \frac{\partial \mu_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i)}{\partial \theta}. \tag{4.15}$$

The immediate reward function could also be learnt by assuming a function approximator $\hat{r}$ is a function of $\mu_{\boldsymbol{\theta}}$'s penultimate layer (interpreted as the state-action representation), trained using sample rewards in $\hat{\mathcal{D}}$. By alternating between the model and Bellman residual update, it might be possible to simultaneously learn a model and fit a value function at each k^{th} policy iteration. The dimensionality of $\boldsymbol{\phi}$ will remain constant and its richness is achieved using the successive deep non-linear hidden layers. Conservative policy updates are therefore more sensible as the number of output nodes of the $\boldsymbol{\phi}_{\boldsymbol{\vartheta}}$ network remains constant. A possible implementation of deep PCME (DPCME) is described in algorithm 18 and what differentiates it from existing approximate policy iteration algorithms is i) that there is a model building process intertwined with value prediction, ii) both state-action and state representations are learnt and iii) conservative policy improvements will mitigate policy oscillations. Preliminary results look promising and will be explored further as future work, including incorporating LSTD in the loss function in order to compare to BRM.

The next chapter moves away from the parametric model due to the discussed policy instabilities and instead investigates sparsifying the CME algorithm. Neural networks are extensively used in the last chapter but with non-parametric value function approximation.

4.5.2 Is PCME a pseudo-MDP?

Yao and Szepesvári (2012) develop the parametric LAM as a pseudo-MDP. This is made possible because state representation $\boldsymbol{\phi}$ and transition data are given a priori to form the discrete MDP that can then be solved using parametric methods. PCME is therefore not a pseudo-MDP because the embedding receives raw states in $\mathcal{S}$ as inputs and not $\boldsymbol{\phi}$. It might be possible to re-express PCME's $\boldsymbol{\psi}$ representation (over state-actions) as a function of the state representation $\boldsymbol{\phi}$. Learning a state representation could be achieved using an auto-encoder (as already demonstrated in many RL algorithms) which would integrate well with the DPCME neural embedding.

Algorithm 18 Deep Parametric CME (DPCME)

Input: Access to unknown MDP $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, P, P_1, r, \gamma\}$ with continuous $\mathcal{S}$, discrete $\mathcal{A}$, known average reward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$. $H = 100$ transitions per trajectory, $J_{\text{imp}} := 10$, $J_{\text{eval}} := 4000$, start-state distribution P_1. Neural networks $\mu_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\phi}}$, $\boldsymbol{\phi}_{\boldsymbol{\vartheta}} : \mathcal{S} \to \mathcal{F}_{\boldsymbol{\phi}}$, and shared last layer weight vector $\mathbf{w} \in \mathbb{R}^{\dim(\mathcal{F}_{\boldsymbol{\phi}})}$, all whose parameters are initialised by drawing from $\mathcal{N}(0, 0.01)$.

Output: $\pi_\kappa(\cdot) \approx \pi^*(\cdot)$.

Initialise: $\hat{q}_0(\cdot, \mathbf{a}) \leftarrow r(\cdot, \mathbf{a})$, $\pi_1(\cdot) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}(\cdot, \mathbf{a})]$, exploration policy $\nu^{\pi_1}(\cdot) \leftarrow \epsilon\text{-greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{q}(\cdot, \mathbf{a})]$, $\epsilon \leftarrow 0.3$, $n_0 \leftarrow 0$, $n_{\text{new}} \leftarrow 2H$, $\mathcal{D}_0 = \emptyset$, learning rates η.

for $k = 1, 2, ..., \kappa - 1$ **do** ▷ k^{th} policy iteration master index

  Data acquisition: $n_k \leftarrow n_{k\text{-}1} + n_{\text{new}}$, collect $\mathcal{D}_{\text{new}} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_k}$; one trajectory using π_k and another using $\tilde{\pi}_k$, beginning from start state $\mathbf{s} \sim P_1(\cdot)$. $\mathcal{D}_k \leftarrow \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{n_k}$, . ▷ aggregate data

  Model:

  for $\ell = 1, 2, ..$ **do** ▷ ℓ^{th} minibatch index

    $\hat{\mathcal{D}} \sim \mathcal{D}_k$

    $\theta \leftarrow \theta - \eta \frac{\partial \hat{\mathcal{L}}(\boldsymbol{\theta})}{\partial \theta}\Big|_{\hat{\mathcal{D}}}$ ▷ model update $\forall \theta \in \boldsymbol{\theta}$, eqn (4.15)

  end for

  Planning:

  $\pi \leftarrow \pi_k$, $\mathbf{w}_{i=0} \leftarrow \mathbf{w}$

  for $i = 1$ **to** J_{imp} **do** ▷ policy improvement index

    for $\ell = 1, 2, ..$ **do** ▷ ℓ^{th} minibatch index

      $\hat{\mathcal{D}} \sim \mathcal{D}_k$

      $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \hat{\mathcal{L}}(\mathbf{w}, \boldsymbol{\vartheta}, \pi)\Big|_{\hat{\mathcal{D}}}$ ▷ BRM update for shared $\mathbf{w} \in \mathcal{F}_{\boldsymbol{\phi}}$, eqn (4.14)

      $\vartheta \leftarrow \vartheta - \eta \frac{\partial \hat{\mathcal{L}}(\mathbf{w}, \boldsymbol{\vartheta}, \pi)}{\partial \vartheta}\Big|_{\hat{\mathcal{D}}}$ ▷ BRM update $\forall \vartheta \in \boldsymbol{\vartheta}$

    end for

    $\mathbf{w} \leftarrow \mathbf{w}_{i-1} + \omega(\mathbf{w} - \mathbf{w}_{i-1})$ ▷ conservative update of shared layer

    $\hat{\underline{q}}^{\pi}(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \gamma \mu_{\boldsymbol{\theta}}^{\top}(\mathbf{s}, \mathbf{a}) \mathbf{w}$, $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$

    $\pi(\cdot) \leftarrow \text{greedy}_{\mathbf{a} \in \mathcal{A}}[\hat{\underline{q}}^{\pi}(\cdot, \mathbf{a})]$ ▷ conservative policy improvement

  end for

  $\pi_{k+1} \leftarrow \pi$

end for

return $\pi_\kappa(\cdot)$

This would mean that the DPCME could be treated as a pseudo-MDP such that a contraction constraint (lemma 3) could be applied to the embedding. This may be a solution to the γ_{vfit} phenomenon that was observed in the experiments.

Chapter 5

Sparse Non-Parametric CME Policy Iteration

The investigation returns to the non-parametric approximate value function setting which has already been demonstrated to yield a stable and consistent policy iteration performance. However the CME model remains computationally unsuitable for online environments. This chapter identifies key components of the CME transition model and decouples them from the size of the training set. The focus is on policy stability and computational sustainability both in model updates and planning as new training data is gathered during exploration.

By reformulating the original RKHS penalised loss equation (2.54), it is possible to solve for $\mathbf{W}$ in the primal while retaining the vvRKHS regulariser. Coupled with the experience of using matching pursuit from the PCME, it is possible to maintain a sparse basis $\mathcal{B}$ and therefore compact feature representation over $\mathcal{S}\times\mathcal{A}$. An extensive investigation into decoupling the basis $\mathcal{C}$ (which defines the set of successor states the value function is evaluated on) from the training set was also made such as adding structural sparsity during $\mathbf{W}$ optimisation, but all efforts failed. The solution is instead provided by colleagues in Lever et al. (2016) whose work external to this thesis is summarised in Appendix C.1. Included in this external work is a Lasso (Hastie et al., 2015) projection for maintaining the pseudo-MDP contraction constraint which is shown to instigate superior performance to the previously used L1-Projection. However empirical results show that this contraction constraint creates a computational bottleneck whose solution will be considered in Chapter 6.

It is important to understand where CCME planning places itself w.r.t. existing dynamic programming algorithms. Referring to section 2.3, then the computational complexity of classic DP planning scales quadratically with the number of states $|\mathcal{S}|^2$, clearly becoming intractable for large state spaces. The existing vanilla CME decouples this cost to a quadratic cost on the size of training set $|\mathcal{S}'|^2$, which quickly becomes

impractical for large MDPs. The CCME decouples planning even further to a set of successor states $|\mathcal{C}|^2$ where $\mathcal{C}-\mathcal{S}'$. The goal is to develop an algorithm whose CME components $\mathcal{B}$ and $\mathcal{C}$ do not grow uncontrollably with a training set that grows as an agent explores its environment.

5.1 CCMEs: Non-Parametric Embeddings with Sparsification

Recall the original CME regressor in (2.50) decomposed into

$$\begin{aligned}\hat{\mu}_{\text{CCME}}(\mathbf{s},\mathbf{a}) &= \sum_{j=1}^{m} L(\mathbf{c}_j,\cdot)\sum_{\ell=1}^{d} w_{j\ell}K(\mathbf{b}_\ell,(\mathbf{s},\mathbf{a})) \in \mathcal{H}_L, \quad (\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A},\\ &= \boldsymbol{\Phi}_{\mathcal{C}}^{\top}\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s},\mathbf{a}), \qquad (5.1)\\ &= \begin{pmatrix} | & & | \\ L(\mathbf{c}_1,\cdot) & \cdots & L(\mathbf{c}_m,\cdot) \\ | & & | \end{pmatrix}\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_d \\ | & & | \end{pmatrix}\begin{pmatrix} K(\mathbf{b}_1,(\mathbf{s},\mathbf{a})) \\ \vdots \\ K(\mathbf{b}_d,(\mathbf{s},\mathbf{a})) \end{pmatrix},\end{aligned}$$

where $\mathbf{W}\in\mathbb{R}^{m\times d}$, L and K are kernels associated with RKHSs $\mathcal{H}_L$ and $\mathcal{H}_K$ respectively. If $\mathcal{D}_k := \{(\mathbf{s},\mathbf{a},\mathbf{s}')_i\}_{i=1}^{n_k}$ is the training set at the k^{th} policy iteration, then $\mathcal{S}'_k := \{\mathbf{s}'_i\}_{i=1}^{n_k} \subseteq \mathcal{S}$ and $\mathcal{Z}_k := \{(\mathbf{s},\mathbf{a})_i\}_{i=1}^{n_k} \subseteq \mathcal{S}\times\mathcal{A}$. Recalling PCME notation, then $\mathcal{B} := \{K(\mathbf{b}_\ell,\cdot)\}_{\ell=1}^{d}$ and $\mathcal{C} := \{L(\mathbf{c}_j,\cdot)\}_{j=1}^{m}$. Each $\mathbf{b}$ is a state-action vector and each $\mathbf{c}$ is a successor state. The vanilla (and Schur-modified) CME set $d=m=n_k$ where n_k is the number of transitions in the dataset at policy iteration k, such that $\mathcal{B}$ and $\mathcal{C}$ will grow uncontrollably with $\mathcal{D}_k$. Recalling the algorithm computational complexities listed table 2.2, then CME planning scales quadratically with m, which motivates decoupling $\mathcal{C}$ from $\mathcal{D}_k$. In addition, model construction computational complexity can be controlled by decoupling $\mathcal{B}$ from $\mathcal{D}_k$. Once $\mathcal{C}$ and $\mathcal{B}$ are updated at each k^{th} policy iteration, the non-square matrix $\mathbf{W}$ is fitted using a closed form solution to the RKHS norm-penalised empirical risk minimisation problem.

Existing Work Grünewälder et al. (2012b) first learn a full $\hat{\mu}_{CME}$ on the entire dataset $\mathcal{D}$ provided a priori, then deploy sparsification techniques to the full vanilla embedding. However Grünewälder et al. (2012a) do not present policy learning performance results of the sparsified embedding; only out-of-sample supervised loss errors are presented. Sparsification focusses on driving elements in a CME's weight matrix $\mathbf{W}$ to zero (and thus reducing the size of $\mathcal{C}$ and $\mathcal{B}$) using a structured sparsity inducing norm and FISTA (Beck and Teboulle, 2009) optimisation.

This chapter focusses on developing practical CME sparsification techniques when data isn't provided a priori and analyses how a sparse CME affects policy learning.

This approach seeks to mitigate ever having to calculate the full CME and instead maintains a compact embedding for the entirety of exploring an MDP. As part of the investigation an attempt was made to adapt the sparsification techniques in Grünewälder et al. (2012b, section 5) and will be described below.

5.1.1 Algorithm Details

Solving for W in the Primal

The following analysis assumes $\mathcal{B}$ and $\mathcal{C}$ have already been found. Substituting (5.1) into the regularised loss (2.54) gives

$$\hat{\mathbf{W}} = \underset{\mathbf{W}\in\mathbb{R}^{|\mathcal{C}|\times|\mathcal{B}|}}{\arg\min} \Big[\frac{1}{2n_k}\sum_{i=1}^{n_k} ||\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s}_i,\mathbf{a}_i) - L(\mathbf{s}'_i,\cdot)||_{\mathcal{F}}^2 + \frac{\lambda}{2}||\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot)||_{\boldsymbol{\Gamma}}^2\Big]. \tag{5.2}$$

As before, the non-parametric value function is modelled in an RKHS $\mathcal{F} = \mathcal{H}_L$ and the embedding function resides in a vvRKHS $\mu(\cdot) \in \mathcal{H}_{\boldsymbol{\Gamma}}$ as discussed in section B.4.6. The sets $\mathcal{C}$ and $\mathcal{B}$ are sizes m and d respectively which are not necessarily as large as the number of samples n_k in $\mathcal{D}_k$. The following notation is defined and builds upon that found in table 2.1,

$$\begin{aligned}
\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s},\mathbf{a}) &:= [K(\mathbf{b}_1,(\mathbf{s},\mathbf{a})),...,K(\mathbf{b}_d,(\mathbf{s},\mathbf{a}))]^\top \in \mathbb{R}^d,\\
\boldsymbol{\Upsilon}_{\mathcal{B}} &:= [K(\mathbf{b}_1,\cdot),...,K(\mathbf{b}_d,\cdot)]^\top =: \boldsymbol{\psi}_{\mathcal{B}}(\cdot),\\
\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} &:= [\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s}_1,\mathbf{a}_1),...,\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s}_{n_k},\mathbf{a}_{n_k})]^\top \in \mathbb{R}^{n\times d},\\
\left(\mathbf{K}_{\mathcal{B}\mathcal{B}}\right)_{jk} &:= \left(\boldsymbol{\Upsilon}_{\mathcal{B}}\boldsymbol{\Upsilon}_{\mathcal{B}}^\top\right)_{jk} = \langle K(\mathbf{b}_j,\cdot),K(\mathbf{b}_k,\cdot)\rangle_{\mathcal{H}_K}, \quad \mathbf{K}_{\mathcal{B}\mathcal{B}}\in\mathbb{R}^{d\times d},\\
\boldsymbol{\Phi}_{\mathcal{C}} &:= [L(\mathbf{c}_1,\cdot),...,L(\mathbf{c}_m,\cdot)]^\top,\\
\boldsymbol{\Phi}_{\mathcal{S}'} &:= [L(\mathbf{s}'_1,\cdot),...,L(\mathbf{s}'_{n_k},\cdot)]^\top,\\
\left(\mathbf{L}_{\mathcal{C}\mathcal{C}}\right)_{jk} &:= \left(\boldsymbol{\Phi}_{\mathcal{C}}\boldsymbol{\Phi}_{\mathcal{C}}^\top\right)_{jk} = \langle L(\mathbf{c}_j,\cdot),L(\mathbf{c}_k,\cdot)\rangle_{\mathcal{H}_L}, \quad \mathbf{L}_{\mathcal{C}\mathcal{C}}\in\mathbb{R}^{m\times m}.
\end{aligned}$$

Lemma 5. *Given sparse $\mathcal{B}$ and $\mathcal{C}$, then by choosing $\boldsymbol{\Gamma}((\mathbf{s},\mathbf{a})_i,(\mathbf{s},\mathbf{a})_j) := K((\mathbf{s},\mathbf{a})_i,(\mathbf{s},\mathbf{a})_j)\mathbf{I}$ as the operator-valued kernel in the vvRKHS setting where $\mathbf{I}:\mathcal{H}_L\to\mathcal{H}_L$, then the regularisation term in equation 5.2 is $||\mu(\cdot)||_{\boldsymbol{\Gamma}}^2 = \mathrm{Tr}(\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}})$.*

Proof.

$$\begin{aligned}
||\mu(\cdot)||_{\boldsymbol{\Gamma}}^2 &= \left\langle \boldsymbol{\Phi}_{\mathcal{C}}^\top\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot), \boldsymbol{\Phi}_{\mathcal{C}}^\top\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot)\right\rangle_{\boldsymbol{\Gamma}},\\
&= \Big\langle \sum_{j=1}^m L(\mathbf{c}_j,\cdot)\sum_{\ell=1}^d w_{j\ell}K(\mathbf{b}_\ell,\cdot),\ \sum_{j'=1}^m L(\mathbf{c}_{j'},\cdot)\sum_{\ell'=1}^d w_{j'\ell'}K(\mathbf{b}_{\ell'},\cdot)\Big\rangle_{\boldsymbol{\Gamma}},\\
&= \sum_{\ell=1}^d\sum_{j'=1}^m\sum_{j=1}^m\sum_{\ell'=1}^d \left\langle L(\mathbf{c}_j,\cdot)w_{j\ell}K(\mathbf{b}_\ell,\cdot),\ L(\mathbf{c}_{j'},\cdot)w_{j'\ell'}K(\mathbf{b}_{\ell'},\cdot)\right\rangle_{\boldsymbol{\Gamma}},
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\ell=1}^{d}\sum_{j'=1}^{m}\sum_{j=1}^{m}\sum_{\ell'=1}^{d} w_{j\ell}\,\langle L(\mathbf{c}_j,\cdot), L(\mathbf{c}_{j'},\cdot)\rangle_L\, w_{j'\ell'}\,\langle K(\mathbf{b}_\ell,\cdot), K(\mathbf{b}_{\ell'},\cdot)\rangle_K\,,\\
&= \sum_{\ell=1}^{d}\sum_{j'=1}^{m}\sum_{j=1}^{m}\sum_{\ell'=1}^{d} w_{j\ell}\,L(\mathbf{c}_j,\mathbf{c}_{j'})\,w_{j'\ell'}\,K(\mathbf{b}_\ell,\mathbf{b}_{\ell'}),\\
&= \sum_{\ell=1}^{d}\sum_{j'=1}^{m}\sum_{j=1}^{m} w_{j\ell}\,L(\mathbf{c}_j,\mathbf{c}_{j'})\sum_{\ell'=1}^{d} w_{j'\ell'}\,K(\mathbf{b}_\ell,\mathbf{b}_{\ell'}),\\
&= \sum_{\ell=1}^{d}\sum_{j'=1}^{m}(\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}})_{\ell j'}(\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}}^\top)_{j'\ell},\\
&= \sum_{\ell=1}^{d}(\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}})_{\ell\ell},\\
&= \mathrm{Tr}(\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}}),
\end{aligned}
$$

where the linearity of the inner product is invoked on line three. In line four the inner product associated with the choice of kernel for the vvRKHS $\mathcal{H}_{\mathbf{\Gamma}}$ - see equation (B.69). □

Lemma 6 (Sparse CME Regressor)**.** *While holding both $\mathcal{B}$ and $\mathcal{C}$ constant, the solution in the primal to the optimisation problem 5.2 has the closed form*

$$\mathbf{W} = (\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{L}_{\mathcal{C}\mathcal{S}'}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1}.$$

Proof. Substituting the result of lemma 5 into equation (5.2) then differentiating w.r.t $\mathbf{W}$,

$$
\begin{aligned}
\mathbf{0} = \nabla_{\mathbf{W}}\Big[\frac{1}{2n}\mathrm{Tr}\Big((\mathbf{\Phi}_{\mathcal{C}}^\top\mathbf{W}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top - \mathbf{\Phi}_{\mathcal{S}'}^\top)^\top(\mathbf{\Phi}_{\mathcal{C}}^\top\mathbf{W}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top - \mathbf{\Phi}_{\mathcal{S}'}^\top)\Big)&\\
+ \frac{\lambda}{2}\mathrm{Tr}(\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}})\Big],&\\
= \frac{1}{n}\mathbf{\Phi}_{\mathcal{C}}(\mathbf{\Phi}_{\mathcal{C}}^\top\mathbf{W}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top - \mathbf{\Phi}_{\mathcal{S}'}^\top)\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}} + \lambda\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}}&\\
= \frac{1}{n}(\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}} - \mathbf{L}_{\mathcal{C}\mathcal{S}'}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}) + \lambda\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\mathbf{K}_{\mathcal{B}\mathcal{B}}&\\
= \mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}(\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}}) - \mathbf{L}_{\mathcal{C}\mathcal{S}'}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}&\\
\Rightarrow \mathbf{W} = (\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{L}_{\mathcal{C}\mathcal{S}'}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1}\,.&\quad\square
\end{aligned}
$$

In the second line the derivatives of the trace are given by identities (B.82) and (B.81) are used. The complexity of this operation is $\mathcal{O}(d^3 + n(d^2 + m^2) + m^3)$ which is linear in the training set size n. A constant ridge term $\lambda = 10^{-8}$ was added to the inverse $(\mathbf{L}_{\mathcal{C}\mathcal{C}} + \lambda)^{-1}$ to prevent the possibility of ill-conditioned matrices. □

Corollary 6.1. *The sparse regressor can be specialised to either i) decouple only $\mathcal{B}$ from n such that*

$$\begin{aligned}\mathbf{W} &= (\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{L}_{\mathcal{C}\mathcal{S}'}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1},\\ &= (\mathbf{L}_{\mathcal{S}'\mathcal{S}'})^{-1}\mathbf{L}_{\mathcal{S}'\mathcal{S}'}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1},\\ &= \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1},\end{aligned}$$

or ii) recover the original fully data-coupled CME regressor,

$$\begin{aligned}\mathbf{W} &= \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1},\\ &= \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{Z}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{Z}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{Z}} + n\lambda\mathbf{K}_{\mathcal{Z}\mathcal{Z}})^{-1},\\ &= \mathbf{K}(\mathbf{K}\mathbf{K} + n\lambda\mathbf{K})^{-1},\\ &= (\mathbf{K} + n\lambda\mathbf{I}_n)^{-1},\end{aligned}$$

where in the second line $\mathcal{B} = \mathcal{Z}$ and $\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{Z}} = \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{Z}}^\top = \mathbf{K} \in \mathbb{R}^{n_k \times n_k}$. Both regulariser λ and kernel σ hyper-parameters can be found by an appropriate cross validation scheme.

The task remains to implement methods that maintain both $\mathcal{B}$ and $\mathcal{C}$.

Maintaining a Compact Basis $\mathcal{B}$

Experience of learning compact parametric representations in the PCME (see lemma 4) is exploited here. Vector-valued matching pursuit is considered not as a parametric feature learner, but instead as a method to learn a compact basis $\mathcal{B}$ used to represent the $\boldsymbol{\psi}_{\mathcal{B}}(\cdot)$ component of an RKHS embedding as described in equation (5.2). The difference is that the PCME embedding $\mu : \mathcal{X} \to \mathcal{Y}$ assumes $\mathcal{Y}$ to be a vanilla Hilbert space $\mathcal{F}_\phi$ such that the matching pursuit model residue also resided in $\mathcal{Y}$. In the non-parametric CME setting $\mathcal{Y}$ is an RKHS $\mathcal{H}_L$, therefore an approximation is made for the $\mathcal{H}_L$ components in order to efficiently apply lemma 4 in an approximate matching pursuit algorithm.

Approximate Matching Pursuit Regression By explicitly reordering the summation, then the embedding can be written as equation (A.2),

$${}^{(\bullet)}\mu^d(\mathbf{z}) := \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \mathbf{z}), \quad \tilde{\mathbf{w}}_\ell \in \mathcal{H}_L,\ \mathbf{z} \in \mathcal{S}\times\mathcal{A},$$

where $\mathbf{w}_\ell$ is the ℓ^{th} column of the embedding's weight matrix $\mathbf{W}$. Following lemma 4, then the model residual for the embedding is

$$\begin{aligned}{}^{(\bullet)}\mathbf{r}_i^d &:= L(\mathbf{s}_i', \bullet) - {}^{(\bullet)}\mu^d(\mathbf{z}_i), \qquad (\mathbf{z}, \mathbf{s}')_i \in \mathcal{D}_k,\\ &= L(\mathbf{s}_i', \bullet) - \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \mathbf{z}_i) \in \mathcal{H}_L. \end{aligned} \tag{5.3}$$

Lemma A.1 describes how the $\mathcal{B}$ basis of embedding (A.2) can be maintained by applying lemma 4. As the embedding maps $\mu := \mathcal{S} \times \mathcal{A} \to \mathcal{Y}$ where $\mathcal{Y} = \mathcal{H}_L$, then model residues also lie in $\mathcal{H}_L$ and by corollary A.1.1 such an approach detrimentally increases the size of $\mathcal{C}$. Given that one of our goals will be to control the size of $\mathcal{C}$ an another approach for maintaining $\mathcal{B}$ is required. We are motivated to keep $\mathcal{C}$ fixed while enriching $\mathcal{B}$. One approach is described in lemma A.2 however this was not implemented in the CCME algorithm and remains available for future experiments.

Maintaining model residues that lie in an RKHS are also problematic because additional overhead is required to evaluate and keep track of residue basis functions. Updating residues after each new basis is added also requires additional overhead. Matching pursuit is more efficient when $\mathcal{Y}$ is a regular Hilbert space and therefore an approximate matching pursuit regression algorithm is developed for the embedding. An approximation of the original RKHS model residual equation (5.3) is made, using an incomplete Cholesky decomposition (see section B.4.2) of the kernel matrix associated with $\mathcal{C}$ is made such that

$$\mathbf{P}_{\mathcal{C}}^{\top} \mathbf{P}_{\mathcal{C}} \approx \mathbf{\Phi}_{\mathcal{C}} \mathbf{\Phi}_{\mathcal{C}}^{\top} = \mathbf{L}_{\mathcal{C}\mathcal{C}}, \tag{5.4}$$

where $\mathbf{P}_{\mathcal{C}} := [\mathbf{p}_1, ..., \mathbf{p}_{|\mathcal{C}|}] \in \mathbb{R}^{m_{\text{chol}} \times |\mathcal{C}|}$ and $\mathbf{p}_j \in \mathbb{R}^{m_{\text{chol}}}$. The incomplete Cholesky decomposition effectively builds an implicit orthonormal basis of maximum size $m_{\text{chol}} < |\mathcal{C}|$ that best describes the data points in $\mathcal{C}$. In all experiments a maximum $m_{\text{chol}} = 200$ was set. The RKHS residue equation (5.3) can now be approximated,

$$\mathbf{p}'_i - \sum_{j=1}^{m} \mathbf{p}_j \sum_{\ell=1}^{d} w_{j\ell} K(\mathbf{b}_\ell, \mathbf{z}_i) \in \mathbb{R}^{m_{\text{chol}}} \approx L(\mathbf{s}'_i, \bullet) - \sum_{j=1}^{m} L(\mathbf{c}_j, \bullet) \sum_{\ell=1}^{d} w_{j\ell} K(\mathbf{b}_\ell, \mathbf{z}_i) \in \mathcal{H}_L.$$

Note that $\mathbf{p}'_i$ is derived from the incomplete Cholesky decomposition of the $\mathcal{D}_{\text{eval}}$ data in the following way. An incomplete Cholesky decomposition on the entire training data's successor states $\mathcal{S}' = \{\mathbf{s}'_i\}_{i=1}^{n_k}$ is first made,

$$\mathbf{P}_{\mathcal{S}'}^{\top} \mathbf{P}_{\mathcal{S}'} \approx \mathbf{\Phi}_{\mathcal{S}'}^{\top} \mathbf{\Phi}_{\mathcal{S}'} = \mathbf{L}_{\mathcal{S}'\mathcal{S}'},$$

which costs $\sim \mathcal{O}(n_k\, m_{\text{chol}}^2)$ and of course can be a subset of states when the dataset gets very large. Then an evaluation set $\mathcal{D}_{\text{eval}}$ for the matching pursuit regression problem is drawn $\mathcal{D}_{\text{eval}} \sim \mathcal{D}_{n_k}$ whose indexes in the parent set can be used to extract $\mathbf{P}_{\mathcal{S}'_{\text{eval}}}$ from $\mathbf{P}_{\mathcal{S}'}$ such that $\mathbf{P}_{\mathcal{S}'_{\text{eval}}}^{\top} \mathbf{P}_{\mathcal{S}'_{\text{eval}}} \approx \mathbf{L}_{\mathcal{S}'_{\text{eval}}\mathcal{S}'_{\text{eval}}}$ where $\mathbf{P}_{\mathcal{S}'_{\text{eval}}} := [\mathbf{p}'_1, ..., \mathbf{p}'_{|\mathcal{D}_{\text{eval}}|}] \in \mathbb{R}^{m_{\text{chol}} \times |\mathcal{D}_{\text{eval}}|}$.

The loss of the approximate d^{th} residue over $\mathcal{D}_{\text{eval}}$ is therefore,

$$\begin{aligned}\sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||\mathbf{r}_i^d||^2_{\mathcal{H}_L} &\approx \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||\mathbf{p}'_i - \sum_{j=1}^{m} \mathbf{p}_j \sum_{\ell=1}^{d} w_{j\ell} K(\mathbf{b}_\ell, \mathbf{z}_i)||_2^2, \\ &= \sum_{i=1}^{|\mathcal{D}_{\text{eval}}|} ||\mathbf{p}'_i - \sum_{\ell=1}^{d} \mathbf{q}_\ell K(\mathbf{b}_\ell, \mathbf{z}_i)||_2^2, \\ &= ||\mathbf{P}_{\mathcal{S}'_{\text{eval}}} - \mathbf{Q}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{S}'_{\text{eval}}}||^2_{\text{Fr}}, \\ &= ||\mathbf{R}^d||^2_{\text{Fr}},\end{aligned}$$

where $\mathbf{q}_\ell := \sum_{j=1}^{m} \mathbf{p}_j w_{j\ell}$, $\mathbf{Q} \in \mathbb{R}^{m_{\text{chol}} \times |\mathcal{B}|}$ and $\mathbf{R}^d$ is the residue matrix.

By arranging data matrices $\mathbf{X} = \mathbf{Z}_{\text{eval}}$ and $\mathbf{Y} = \mathbf{P}_{\mathcal{S}'_{\text{eval}}}$, then the matching pursuit regression algorithm described by lemma 4 can be initiated to find a compact basis $\mathcal{B}$. The $d{+}1$ weight vector is calculated and basis function is chosen using equations (4.6) and (4.7) respectively. This approximate procedure is only used to find a good $\mathcal{B}$ such that $\mathbf{Q}$ is discarded and replaced by backfitting with the primal weights calculation (lemma 6) after $\mathcal{B}$ has been found. $\mathbf{Q}$ is only required and maintained during the matching pursuit procedure itself - no incomplete Cholesky decomposition of $\mathbf{L}_{\mathcal{C}\mathcal{C}}$ is executed, only the successor targets $\mathbf{L}_{\mathcal{S}'\mathcal{S}'}$ are required for the matching pursuit procedure. Approximate matching pursuit turns the RKHS-valued problem into one that is of similar operation to the parametric setting in Chapter 4. The residuals are easily updated upon each $d{+}1$ basis being added because they are not RKHS functions themselves. Most importantly $\mathcal{C}$ is not enriched during this procedure. The cost of calculating a new weight vector and its new basis function is $\sim \mathcal{O}(|\mathcal{D}_{\text{eval}}|\, m_{\text{chol}})$.

Implementing a Sparse $\mathcal{C}$

Maintaining the set of successor states $\mathcal{C}$ is considered a harder problem because it defines what states the value function is maintained over, rendering the choice of $\mathcal{C}$ critical to the success of the policy iteration algorithm.

Group Lasso Penalty The first approach investigated was to adapt the structured sparsity technique in Grünewälder et al. (2012b, section 5) to the primal weight optimisation procedure. An additional penalty term is added to the loss equation (5.2) that induces sparsity in the rows of an embedding's weight matrix $\mathbf{W}$. Inducing sparsity at this group level offers a mechanism to remove $L(\mathbf{c}_j, \cdot)$ elements from $\mathcal{C}$, which can easily be seen in

$$\begin{aligned}\mu(\mathbf{z}) &= \boldsymbol{\Phi}_{\mathcal{C}}^\top [\, \mathbf{w}_{1:}^\top, ..., \mathbf{w}_{m:}^\top \,]^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}), \\ &= \sum_{j=1}^{m} L(\mathbf{c}_j, \cdot) \mathbf{w}_{j:} \boldsymbol{\psi}(\mathbf{z})\end{aligned}$$

where $\mathbf{w}_{j:} \in \mathbb{R}^{1\times d}$ is the j^{th} group of weights in $\mathbf{W}$. The following objective was investigated (with the aim of reintroducing the RKHS regulariser in a future algorithm)

$$\begin{aligned}\hat{\mathcal{L}}(\mathbf{W}) &= \frac{1}{2n_k}\sum_{i=1}^{n_k} ||\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{s}_i,\mathbf{a}_i) - L(\mathbf{s}'_i,\cdot)||^2_{\mathcal{H}_L} + \lambda_{21}||\mathbf{W}||_{21}, \\ &= \frac{1}{2n_k}\text{Tr}\Big((\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top - \boldsymbol{\Phi}_{\mathcal{S}'}^\top)^\top(\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top - \boldsymbol{\Phi}_{\mathcal{S}'}^\top)\Big) + \lambda_{21}||\mathbf{W}||_{21}. \qquad (5.5)\end{aligned}$$

This is a variant of the group lasso objective and does not have a closed form solution because the regulariser is non-differentiable. Solving problems like this is described in section B.4.4 and in the spirit of equation (B.57), then the embedding objective equation (5.5) solved by projected gradient descent is

$$\mathbf{W}^{(t+1)} = \text{prox}_{\lambda_{21}||\cdot||_{21}}\Big[\mathbf{W}^{(t)} - \eta\frac{1}{n}(\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}^{(t)}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} - \mathbf{L}_{\mathcal{C}\mathcal{S}'}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})\Big].$$

Given that $\lambda_{21}||\mathbf{W}||_{21} := \lambda_{21}\sum_{j=1}^d ||\mathbf{w}_{j:}||_2$, then the prox operator[1] can be applied independently to each row $\mathbf{w}_{j:}$ in $\mathbf{W}$. The actual algorithm used to carry out the optimisation was FISTA (Beck and Teboulle, 2009).

In the context of explorative-PI, it was intended that the set $\mathcal{C}$ be augmented by new successor states at the beginning of the k^{th} iteration; $\mathcal{C} \leftarrow \mathcal{C}+\{L(\mathbf{s}'_j,\cdot)\}_{j=1}^{n_{\text{new}}}$. Group lasso optimisation would then proceed in place of the original closed form solution in lemma 6 to prune $\mathcal{C}$ by driving group weights to zero. Cross validation was used to select state-action kernel parameter and the group lasso regulariser strength λ_{21}.

However this approach proved unsuccessful. The iterative FISTA algorithm embedded in a cross validation scheme for hyperparameter selection was very slow and unsuitable to make quick updates to the embedding. It wasn't possible to incorporate this approach into a practical policy iteration algorithm to learn good policies because the sparsification component became the primary source of computational cost. This approach was therefore abandoned in order to pursue techniques more suitable to the online setting.

Lossy Compression (External Work) The actual implementation for maintaining $\mathcal{C}$ was provided externally from this thesis as described in section C.1. When operating with this algorithm, the embedding is referred to as a compressed CME (CCME). The advantage to this approach is that computation costs are low (which can be linear in $|\mathcal{C}|$) and the operation retains policy improvement guarantees.

[1] It was also intended to consider the mixed norm $\sum_{j=1}^d ||L(\mathbf{c}_j)\mathbf{w}_{j:}||_{\mathcal{H}_L}$ instead of just rows in the weight matrix $\mathbf{W}$.

Other Approaches to Embedding Sparsification

Matching Pursuit Regression for $\mathcal{C}$ In the same way that matching pursuit regression was used for $\mathcal{B}$, conceivably it could be used for maintaining $\mathcal{C}$ as hypothesised in lemma A.3. This has not been tested experimentally but could form part of a future investigation.

Sparsification in the RKHS Norm By the properties of an RKHS function (see Appendix B.4.2), if any two functions are close in their RKHS norm then their pointwise function evaluations will also be close. Given a target function $\mathbf{h} \in \mathcal{H}_{\mathbf{\Gamma}}$ then lemma A.4 describes a matching pursuit procedure of incrementally building a sparse approximation of a general vector valued function $\hat{\mathbf{f}} \in \mathcal{H}_{\mathbf{\Gamma}}$ by minimising $||\mathbf{h} - \hat{\mathbf{f}}||_{\mathbf{\Gamma}}$. A general implementation is described in algorithm 24. If the residues are in a vanilla Hilbert space then they are easily maintained.

This approach is adapted in an attempt to sparsify an embedding's basis $\mathcal{B}$ or $\mathcal{C}$ as described in lemma A.6. The advantage with this approach is that no regression data is required, only the existing basis functions. However in both cases the residues are in an RKHS, therefore maintaining them during the matching pursuit procedure requires tracking each basis function which is not as simple as when the residues lie in a vanilla Hilbert space. Preliminary tests for maintaining $\mathcal{B}$ in section A.1.5 were carried out using an approximate matching pursuit variant. Future work is required to fully integrate this approach into policy iteration.

Contraction Constraint (External Work)

A post hoc contraction constraint is imposed on the embedding using a sparse projection implemented using lasso as described in section C.1.2 and implemented as LassoSparse. It uses the same incomplete Cholesky kernel approximation method as used in the approximate matching pursuit procedure. The lasso contraction constraint is compared to the fast L1-projection (that was used to modify the CME in section 3.1.2) in the following experiments. The $\boldsymbol{\alpha}$ values were also normalised (see equation (2.58)) after each constraint was enforced.

5.2 Experiments

The final CCME policy iteration (algorithm 19) blends i) learning $\mathbf{W}$ in the primal, ii) maintaining state-action basis $\mathcal{B}$ with approximate matching pursuit regression, iii) maintaining successor state basis $\mathcal{C}$ using the lossy algorithm and iv) the lasso projection with normalisation was used to maintain the contraction constraint. The previous L1-projection contraction constraint is compared.

Algorithm 19 Explorative CCME-PI

Input: Kernel $L:\mathcal{S}\times\mathcal{S}\to\mathbb{R}$, implicit state representation $\boldsymbol{\phi}(\mathbf{s}'):=L(\mathbf{s}',\cdot)$, access to unknown MDP $\mathcal{M}:=\{\mathcal{S},\mathcal{A},P,P_1,r,\gamma\}$ with continuous $\mathcal{S}$, discrete $\mathcal{A}$, but known average reward function $r:\mathcal{S}\times\mathcal{A}\to[0,1]$. $H=100$ transitions per trajectory, $J_{\text{imp}}:=10$, $J_{\text{eval}}:=4000$, start-state distribution P_1, lossy δ.

Output: $\pi_\kappa(\cdot)\approx\pi^*(\cdot)$.

Initialise: $\hat{q}_0(\cdot,\mathbf{a})\leftarrow r(\cdot,\mathbf{a})$, $\pi_1(\cdot)\leftarrow\text{greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}(\cdot,\mathbf{a})]$, exploration policy $\nu^{\pi_1}(\cdot)\leftarrow\epsilon\text{-greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}(\cdot,\mathbf{a})]$, $\epsilon\leftarrow0.3$, $n_0\leftarrow0$, $n_{\text{new}}\leftarrow2H$, $\mathcal{D}_0=\emptyset$, $\mathcal{C}_0\leftarrow\emptyset$, $\mathcal{B}_0\leftarrow\emptyset$, $\mathbf{W}_0\leftarrow\emptyset$.

for $k=1,2,...,\kappa-1$ **do** ▷ k^{th} policy iteration master index

  $isFullUpdate\leftarrow$ **false** ▷ partial updates possible for some components

  if $k\in fullUpdates$ **then**

    $isFullUpdate\leftarrow$ **true**

  end if

  Data acquisition: $n_k\leftarrow n_{k\text{-}1}+n_{\text{new}}$, collect $\mathcal{D}_{\text{new}}=\{\mathbf{s}_i,\mathbf{a}_i,r_i,\mathbf{s}_i'\}_{i=n_{k\text{-}1}+1}^{n_k}$; one trajectory from each π_k and ν^{π_k}, beginning from start state $\mathbf{s}\sim P_1$, $\mathcal{D}_k\leftarrow\mathcal{D}_{k-1}\cup\mathcal{D}_{\text{new}}$, $\mathcal{S}'_{\text{new}}\leftarrow\{\mathbf{s}_i'\}_{i=n_{k\text{-}1}+1}^{n_k}$, $\mathcal{Z}_{\text{new}}\leftarrow\{(\mathbf{s},\mathbf{a})_i\}_{i=n_{k\text{-}1}+1}^{n_k}$. ▷ aggregate data

  Update $\mathcal{C}$:
  $\mathcal{C}_k\leftarrow$AugmentCompressionSet$(L,\mathcal{C}_{k-1},\mathcal{S}'_{\text{new}},\delta)$ ▷ algorithm 26

  Update $\mathcal{B}$ ($\boldsymbol{\psi}$-Features):

  $(\boldsymbol{\psi}^k,\mathbf{W}_k)$ ▷ algorithm 20
  $\leftarrow$LearnPsiCCME$\left(\mathcal{C}_k,L,\boldsymbol{\psi}^k,\mathbf{W}_{k-1},r,\mathcal{D}_k,\mathcal{Z}_{\text{new}},K,isFullUpdate\right)$

  Model:

  $\mathbf{W}_k\leftarrow$PrimalWeights$\left(\boldsymbol{\psi}^k,\mathcal{C}_k,L,K,\mathcal{D}_k\right)$,
  $\mathbf{v}:=[v(\mathbf{s}_1'),.,v(\mathbf{s}'_{|\mathcal{C}_k|})]^\top$, $(\hat{T}_\mu^\pi v)(\cdot):=r(\cdot,\pi(\cdot))+\gamma$LassoSparse$\left(\boldsymbol{\psi}_{\mathcal{B}}^\top(\cdot,\pi(\cdot))\mathbf{W}\right)\mathbf{v}$.
  ▷ see equation (C.5)

  Planning:

  for $i=1$ **to** J_{imp} **do** ▷ policy improvement index

    $\mathbf{v}\leftarrow\mathbf{0}$, $\pi\leftarrow\pi_k$

    for $j=1$ **to** J_{eval} **do** ▷ exact policy evaluation

      $\mathbf{v}\leftarrow\hat{T}_\mu^\pi\mathbf{v}$

    end for

    $\hat{\mathbf{v}}^\pi\leftarrow\mathbf{v}$

    $\hat{q}^\pi(\mathbf{s},\mathbf{a}):=r(\mathbf{s},\mathbf{a})+\gamma$LassoSparse$\left(\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{s},\mathbf{a})\mathbf{W}\right)\hat{\mathbf{v}}^\pi$, $(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}$
    ▷ see equation (C.5)

    $\pi(\cdot)\leftarrow\text{greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}^\pi(\cdot,\mathbf{a})]$ ▷ greedy policy improvement

  end for

  $\pi_{k+1}\leftarrow\pi$

end for

return $\pi_\kappa(\cdot)$

Algorithm 20 LearnPsiCCME($\mathcal{C}, L, \boldsymbol{\psi}, \mathbf{W}, r, \mathcal{D}, \mathcal{Z}_{\text{new}}, K, isFullUpdate$)

Input: Compression set $\mathcal{C}$, state Gaussian kernel $L : \mathcal{S}\times\mathcal{S} \to \mathbb{R}$ and $\boldsymbol{\psi}(\cdot)$ representation, previous transition model weights $\mathbf{W}$, average immediate reward function $r : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, transition data $\mathcal{D}$, newly acquired state-actions $\mathcal{Z}_{\text{new}}$, state-action Gaussian kernel $K : (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A}) \to \mathbb{R}$, a boolean $isFullUpdate$ specifying quick or full updates.

Output: New state-action feature representation $\boldsymbol{\psi}(\cdot)$, new regressor $\mathbf{W}$.

Initialise: Bandwidth collection $\{\sigma\} := \{0.01, .., 5\}$, collection of basis dictionaries $\{\mathcal{G}\} \leftarrow \emptyset$, $\delta_{\text{tol}} \leftarrow 0$, $\ell_{\text{backfit}} \leftarrow \infty$, $d_{\max} \leftarrow 500$, $\xi \leftarrow 10^{-8}$, $m_{\text{chol}} \leftarrow 200$.

Prepare dictionaries:

$\mathcal{B}_{\text{prev}} := \{K_{\sigma_{\text{prev}}}(\hat{\mathbf{b}}_\ell, \cdot)\}_{\ell=1}^{d_{\text{prev}}} \leftarrow \boldsymbol{\psi}(\cdot)$ ▷ extract basis where $\hat{\mathbf{b}}_\ell \in \mathcal{S}\times\mathcal{A}$

$\tilde{\mathcal{B}} \leftarrow \mathcal{B}_{\text{prev}} \bigcup \{K_{\sigma_{\text{prev}}}(\mathbf{b}_i, \cdot)\}_{i=1}^{n_{\text{new}}}$ ▷ augment candidate functions where $\mathbf{b}_i \in \mathcal{Z}_{\text{new}}$

if *isFullUpdate* **then** ▷ full updates cross-validate over all bandwidths

  for each $\sigma \in \{\sigma\}$ **do**

    $\tilde{\mathcal{G}}^{\sigma} \leftarrow \{K_{\sigma}(\tilde{\mathbf{b}}_i, \cdot)\}_{i=1}^{|\tilde{\mathcal{B}}|}$ ▷ $\tilde{\mathbf{b}}_i \in \tilde{\mathcal{B}}$

    $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \tilde{\mathcal{G}}^{\sigma}$

  end for

else

  $\tilde{\mathcal{G}}^{\sigma_{\text{prev}}} \leftarrow \{K_{\sigma_{\text{prev}}}(\tilde{\mathbf{b}}_i, \cdot)\}_{i=1}^{|\tilde{\mathcal{B}}|}$ ▷ a single child dictionary with σ_{prev}

  $\{\mathcal{G}\} \leftarrow \{\mathcal{G}\} \bigcup \tilde{\mathcal{G}}^{\sigma_{\text{prev}}}$.

end if

Prepare Residues:

$\mathbf{P}_{\mathcal{D}} \leftarrow$ IncompleteCholesky$(\mathbf{K}_{\mathcal{D}\mathcal{D}}, m_{\text{chol}}, \xi)$ ▷ $\mathbf{P}_{\mathcal{D}}^{\top}\mathbf{P}_{\mathcal{D}} \approx \mathbf{K}_{\mathcal{D}\mathcal{D}}$ ▷ see section B.4.2

$\mathcal{D}_{\text{eval}} := \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')_i\}_{i=1}^{n} \sim \mathcal{D}$, $n = |\mathcal{D}_{\text{eval}}|$ ▷ draw evaluation dataset

$\mathbf{P}_{\text{eval}} \sim \mathbf{P}_{\mathcal{D}}$ ▷ draw the same evaluation dataset

if $\mathbf{W}$==$\emptyset$ **then**

  $\mathbf{R} \leftarrow [r(\mathbf{s}_1, \mathbf{a}_1), ..., r(\mathbf{s}_n, \mathbf{a}_n)]^{\top}$ ▷ a proxy residual for the first policy iteration.

else

  $\mathbf{R} \leftarrow \mathbf{P}_{\text{eval}}$ ▷ model residual target

end if

$\mathbf{Z} \leftarrow [\mathbf{z}_1, ..., \mathbf{z}_n]^{\top}$, $\mathbf{z}_i := [\mathbf{s}_i^{\top}, \mathbf{a}_i^{\top}]^{\top}$, $\mathcal{D}_{\text{res}} := \{\mathbf{Z}, \mathbf{R}\}$ ▷ define the residual dataset.

$(\boldsymbol{\psi}(\cdot), \mathbf{W}) \leftarrow$ vvMultipleMatchingPursuit($\mathcal{D}_{\text{res}}$, $\{\mathcal{G}\}$, δ_{tol}, $d_{\max}$, ℓ_{backfit})

return $\boldsymbol{\psi}(\cdot)$, $\mathbf{W}$.

Algorithm 21 PRIMALWEIGHTS($\boldsymbol{\psi}_{\mathcal{B}}, \mathcal{C}, L, K, \mathcal{D}$)

Input: State-action features $\boldsymbol{\psi}_{\mathcal{B}}(\cdot) = [K(\mathbf{b}_1, \cdot), ..., K(\mathbf{b}_d, \cdot)]^\top$, successor state basis $\mathcal{C} := \{L(\mathbf{c}_j, \cdot)\}_{j=1}^m$, state-action kernel $K : (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A}) \to \mathbb{R}$, $L : \mathcal{S}\times\mathcal{S} \to \mathbb{R}$

Output: Regularised embedding weights $\mathbf{W} \in \mathbb{R}^{m\times d}$

Initialise: $\boldsymbol{\Phi}_{\mathcal{C}}^\top := [L(\mathbf{c}_1, \cdot), ..., L(\mathbf{c}_m, \cdot)]^\top \leftarrow \mathcal{C}$, $\boldsymbol{\Upsilon}_{\mathcal{B}} := \boldsymbol{\psi}(\cdot)$, $\mathbf{K}_{\mathcal{B}\mathcal{B}} := \boldsymbol{\psi}(\cdot)\boldsymbol{\psi}^\top(\cdot)$, $\mathbf{L}_{\mathcal{C}\mathcal{C}} \leftarrow \boldsymbol{\Phi}_{\mathcal{C}}\boldsymbol{\Phi}_{\mathcal{C}}^\top$, regularisation parameters $\{\lambda\} \leftarrow \{1\times10^{-7}, ..., 1\}$, $n_{\text{folds}} \leftarrow 5$, $\{\mathcal{D}_1, ..., \mathcal{D}_{n_{\text{folds}}}\} \leftarrow \mathcal{D}$, $\epsilon_{\text{bestTest}} \leftarrow \infty$.

Cross-validate regulariser:

for each $\lambda \in \{\lambda\}$ **do**

  $\epsilon_{\text{sumTest}} \leftarrow 0$

  for $k = 1$ **to** n_{folds} **do**

    $\mathcal{D}_{\text{train}} \leftarrow (\dot{\bigcup}_{i=1}^{n_{\text{folds}}} \mathcal{D}_i)_{i\neq k}$

    $\mathcal{D}_{\text{test}} \leftarrow \mathcal{D}_k$

    $\mathbf{W} = (\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{L}_{\mathcal{C}\mathcal{S}'_{\text{train}}}\boldsymbol{\Upsilon}_{\mathcal{Z}_{\text{train}}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}_{\text{train}}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}_{\text{train}}\mathcal{B}} + \lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1}$

    $\epsilon_{\text{sumTest}} \leftarrow \epsilon_{\text{sumTest}} + \frac{1}{2n_{\text{test}}}||\boldsymbol{\Phi}_{\mathcal{S}'_{\text{test}}}^\top - \boldsymbol{\Phi}_{\mathcal{C}}^\top\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}_{\text{test}}}||_{\mathcal{H}_L}^2$

  end for

  $\epsilon_{\text{avTest}} \leftarrow \epsilon_{\text{sumTest}}/n_{\text{folds}}$

  if $\epsilon_{\text{avTest}} < \epsilon_{\text{bestTest}}$ **then** ▷ compare estimate of out-of-sample error

    $\lambda^* \leftarrow \lambda$, $\epsilon_{\text{bestTest}} \leftarrow \epsilon_{\text{avTest}}$.

  end if

end for

$\mathbf{W} = (\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{L}_{\mathcal{C}\mathcal{S}'}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}^\top\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} + n\lambda\mathbf{K}_{\mathcal{B}\mathcal{B}})^{-1}$

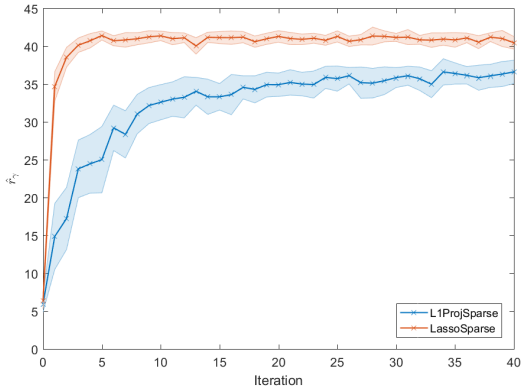
return $\mathbf{W}$.

5.2.1 Holding Pattern Task

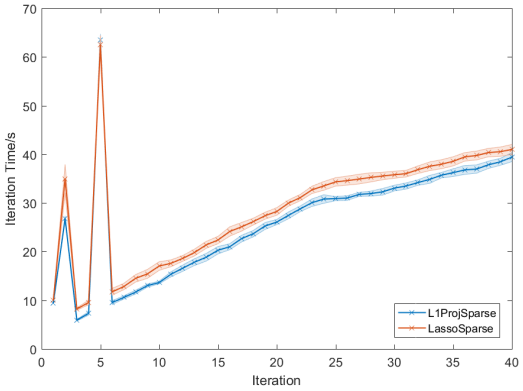
An additional more challenging holding pattern task was created to stress test the learning algorithms. This is similar to the quadrocopter navigation task in the previous chapter with two major differences; the action set is of size $|\mathcal{A}| = 169$ where $\mathcal{A} \in \mathbb{R}^2$ and the aim of the task is to assume a holding (orbital) pattern, or an orbit of radius 5m. The reward function is defined as $r(\mathbf{s}, \mathbf{a}) = \exp(-\frac{1}{\sigma_{\tilde{r}}^2}(\tilde{r}(x, y) - \tilde{r}_{\text{targ}})^2)\exp(-\frac{1}{\sigma_c^2}(c(u, v) - c_{\text{targ}})^2)$, where $\tilde{r}(x, y) = \sqrt{x^2 + y^2}$ is the distance of the quadrocopter from the centre of the holding pattern with a target of $\tilde{r}_{\text{targ}} = 5\text{m}$ and $c(u, v) = \sqrt{u^2 + v^2}$ is the speed in the (x, y) plane with $c_{\text{targ}} = 1.5\text{ms}^{-1}$ is a target speed to assume in the holding pattern. The reward bandwidths were set at $\sigma_c = 1$ and $\sigma_{\tilde{r}} = 2\text{m}$. Discounted cumulative reward of 60 or over is collected for near optimal policies which corresponds to an orbit that is roughly a circle with constant speed.

Figure 5.1 Mountain car: Comparison of contraction constraints

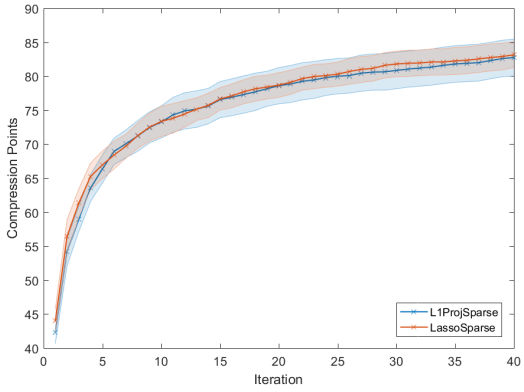
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

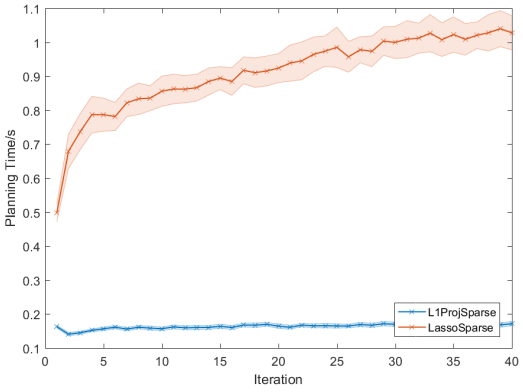
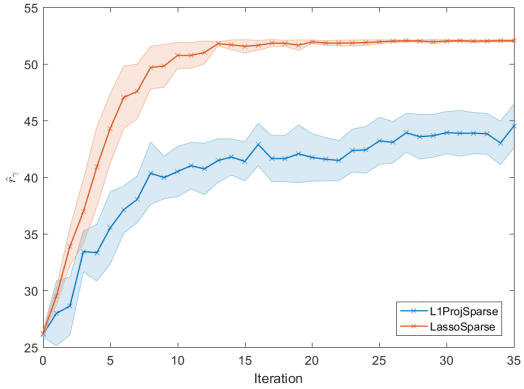
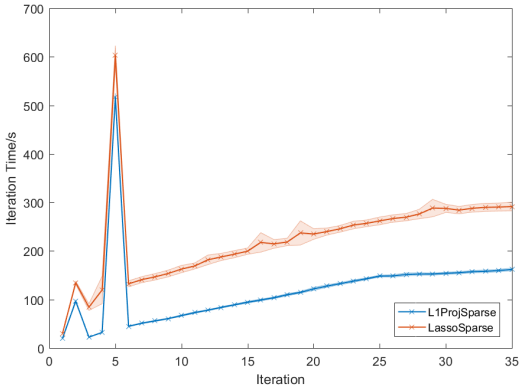


Figure 5.2 Cart-pole: Comparison of contraction constraints

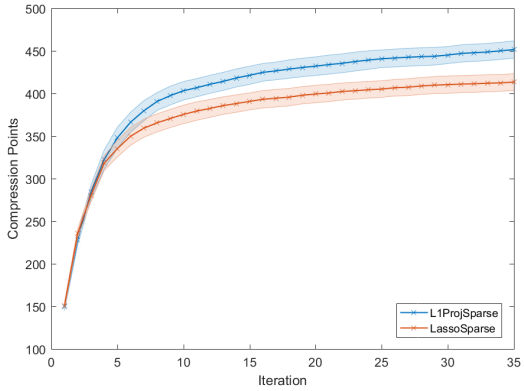
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

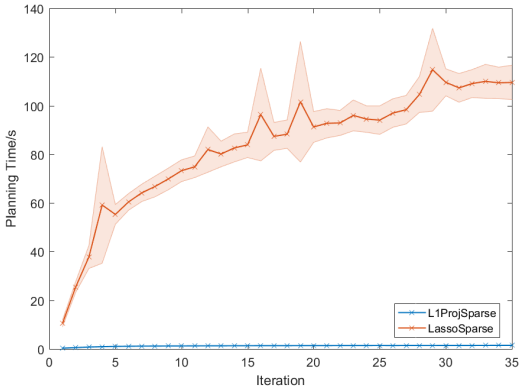
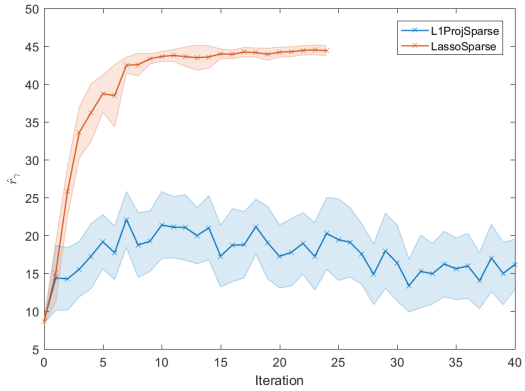
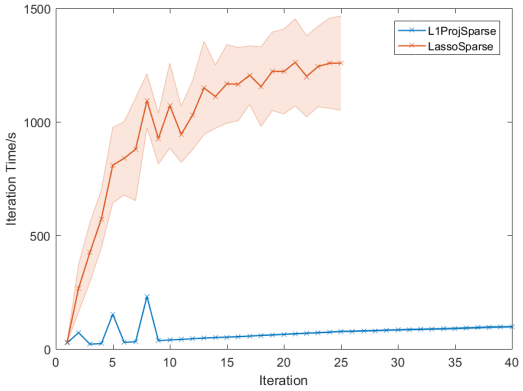


Figure 5.3 Quadrocopter navigation: Comparison of contraction constraints

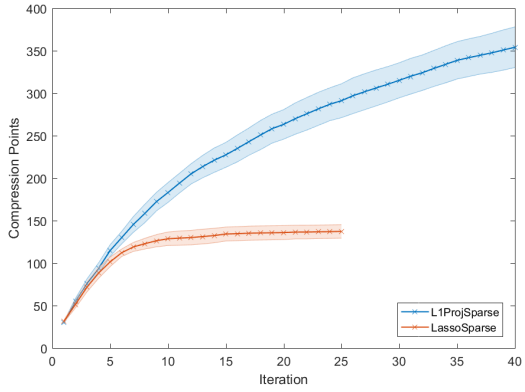
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

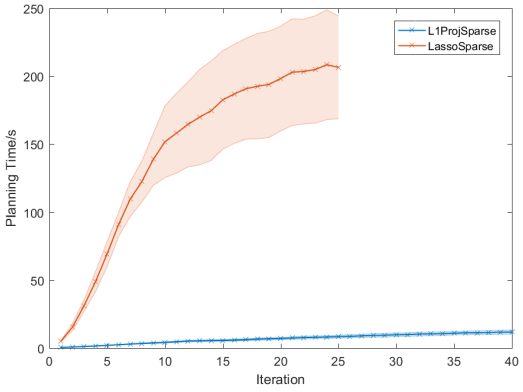
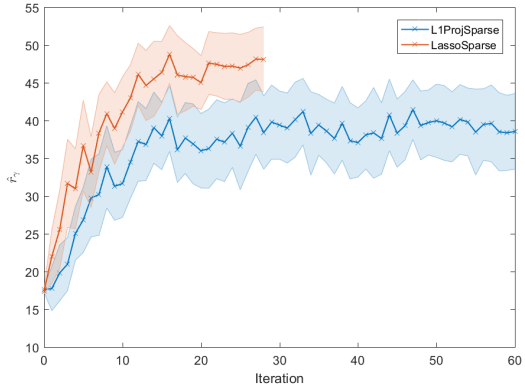
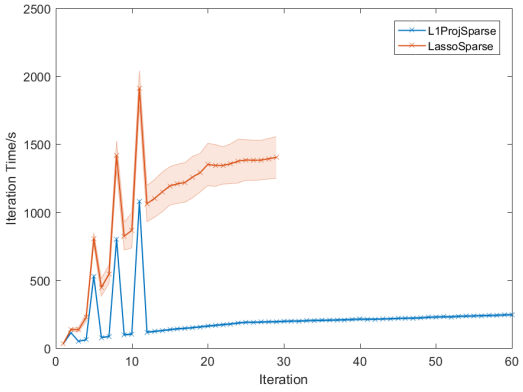


Figure 5.4 Quadrocopter holding pattern: Comparison of contraction constraints

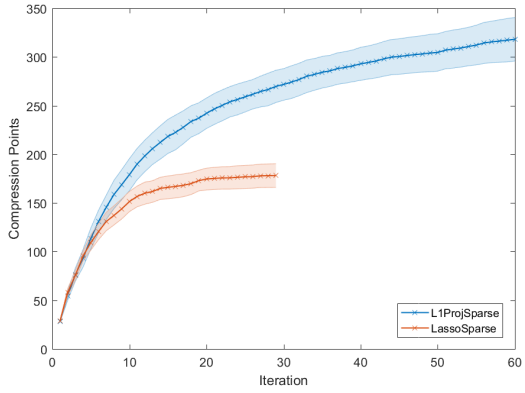
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

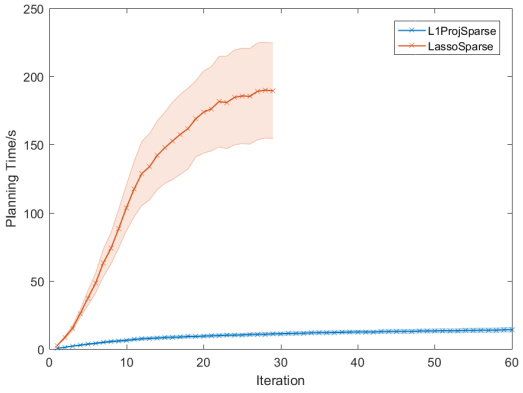
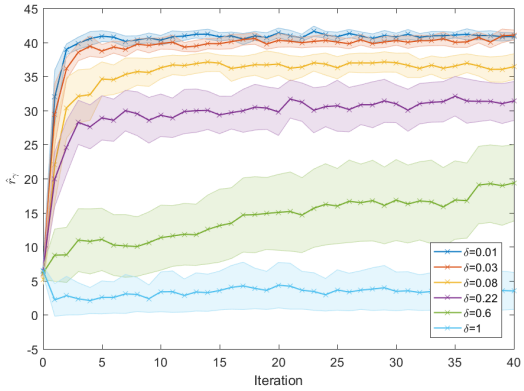
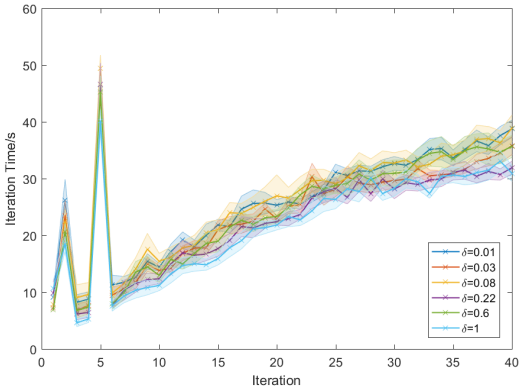


Figure 5.5 Mountain car: varying δ_{lossy}

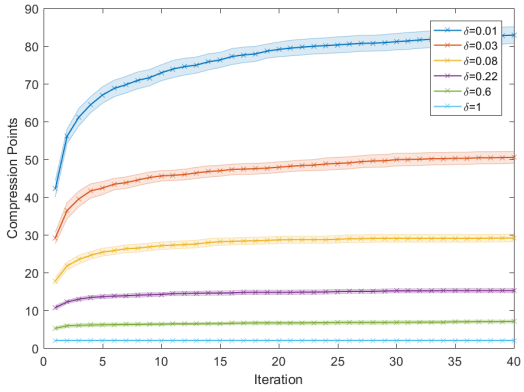
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

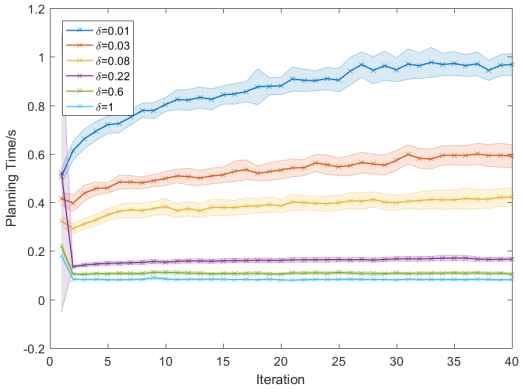
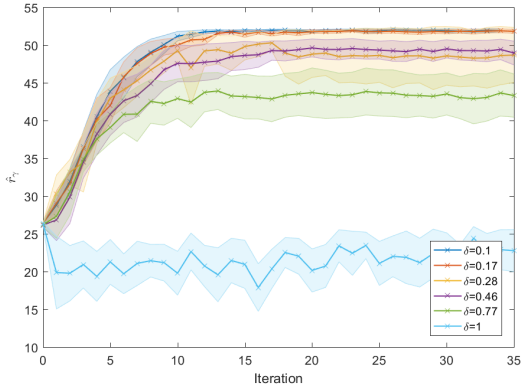
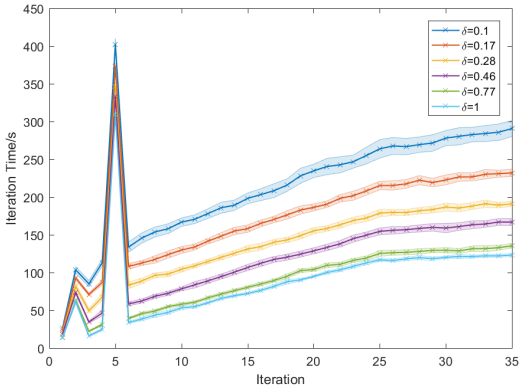


Figure 5.6 Cart-pole: varying δ_{lossy}

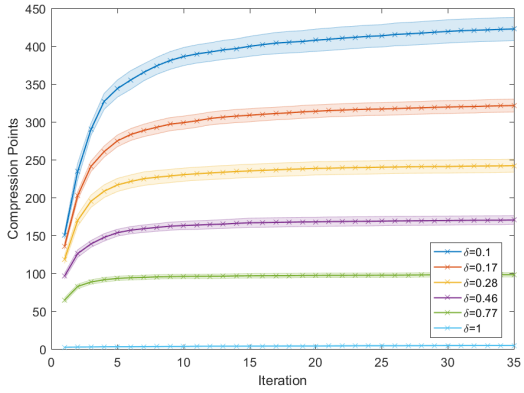
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

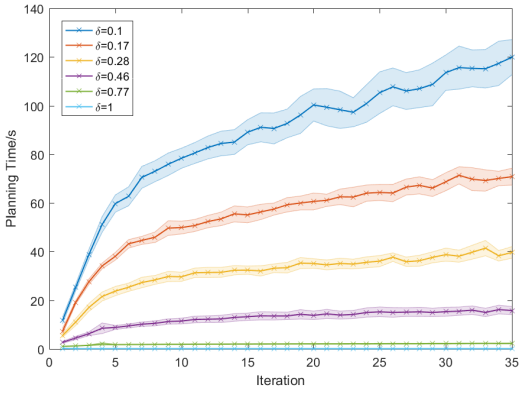
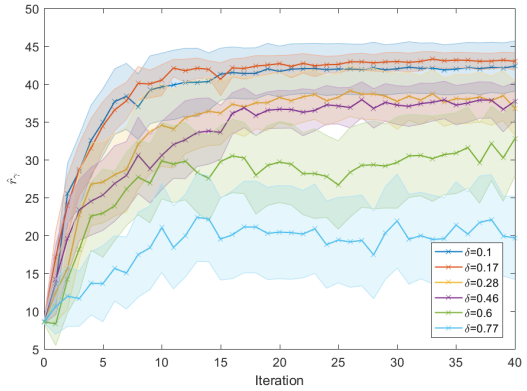
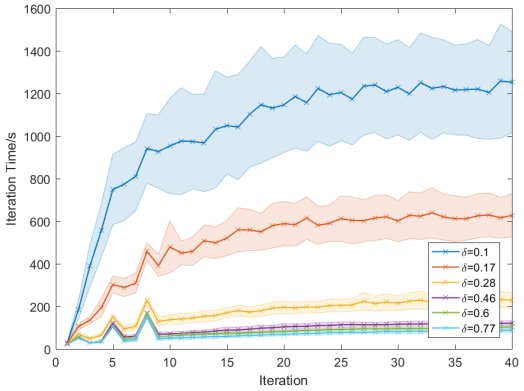


Figure 5.7 Quadrocopter navigation: varying δ_{lossy}

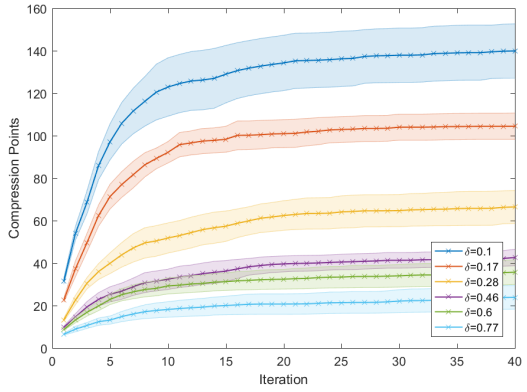
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

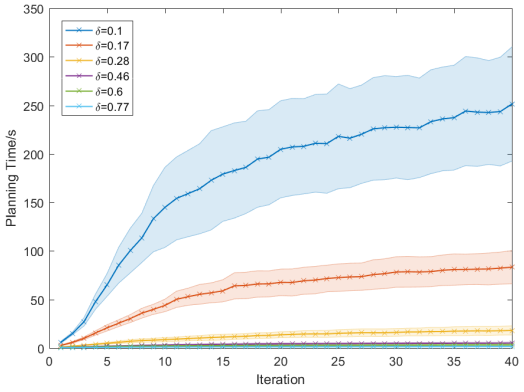
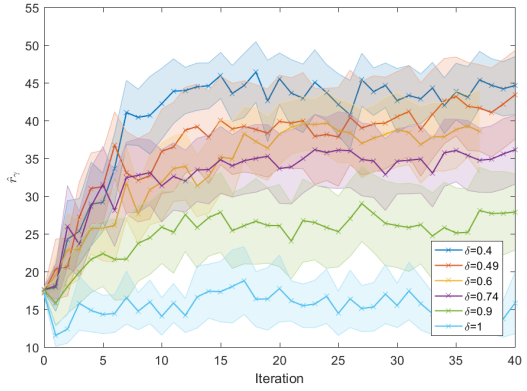
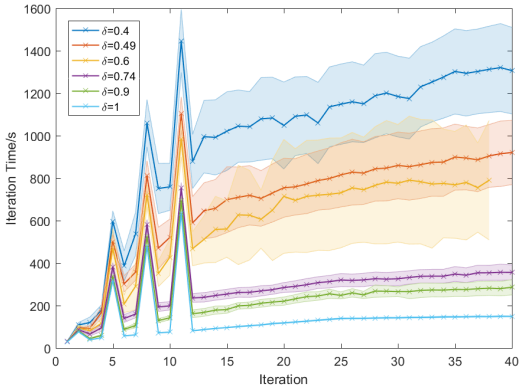


Figure 5.8 Quadrocopter holding pattern: varying δ_{lossy}

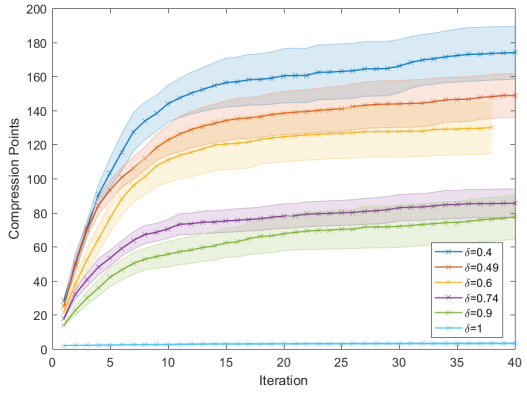
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Compression set $\mathcal{C}$ size



(d) Planning time

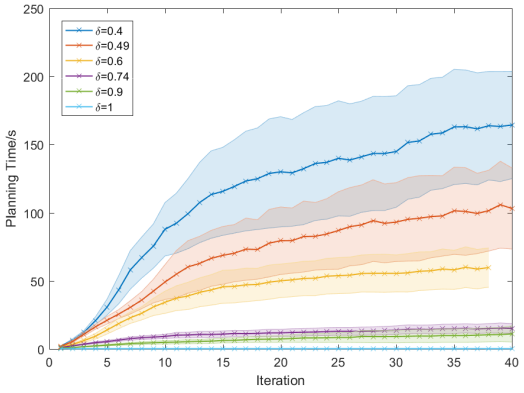


Table 5.1 MDP-specific a priori parameters for CCME

	Mountain car	Cart-pole	Quad nav	Quad holding pat
δ_{lossy}	0.01	0.1	0.1	0.4
L bandwidth $\sigma_{\mathcal{S}}$	0.5	0.5	1.25	1.25
$\lvert\mathcal{B}\rvert \leq d_{\max}$	200	500	200	500

5.3 Discussion

5.3.1 Comparison with Other Algorithms

See section 7.1 for algorithm component timings with comparison to surviving benchmarks from chapter 3 and other algorithms developed in this thesis. CCME clearly achieves an improvement in computational complexity during model construction and planning when compared to the CME algorithm. The sparse embedding is therefore able to reasonably solve the quadrocopter holding pattern MDP without incurring too much computation cost. The CCME outperforms the PCME in learning better policies however it incurs higher planning costs due to the lasso contraction constraint imposed either during planning or whenever the embedding is evaluated, such as when an action is drawn from a policy.

5.3.2 Learning $\mathcal{B}$

As presented in fig. 7.1 to fig. 7.4, it can be seen that the CCME suffers from the same cost spikes as the PCME when learning feature basis $\mathcal{B}$ during model construction. These spikes are due to the full cross validation over multiple dictionaries $\mathcal{G}^{\sigma}$ w.r.t kernel bandwidth σ.

5.3.3 Contraction Constraint

Empirical results in fig. 5.1 to fig. 5.4 consistently demonstrate that the lasso projection contraction constraint on $\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})$ outperforms the previously used fast L1-projection. The difference in performance is significant even as a tolerance parameter of the fast L1-projection is set to zero. The lasso algorithm cost however is significant and scales depending on the solver used i.e. $m_{\text{chol}}|\mathcal{C}_k|^2$ (Friedman et al., 2010) or $m_{\text{chol}}|\mathcal{C}_k|$ (Efron et al., 2004). The GLM method described by Friedman et al. (2010) and implemented by Qian et al. (2013) was the only approach that worked. Specific parameters to the optimiser were tried to see if the computational cost could be reduced, however this was unsuccessful and the results presented are the best achieved using default settings. The cost of the contraction constraint is not only seen in the planning cost, but also in the iteration time. Referring to fig. 7.4b for the quadrocopter holding pattern, the

iteration cost is significantly high because the lasso cost is multiplied by $|\mathcal{A}| = 169$ due to the policy greedily searching over all actions at each state.

Most revealing is that evidence in figures 5.1 to 5.4 suggests that most CCME performance is attributed to the use of the LassoSparse contraction constraint as opposed to the L1-projection. This may be explained because the L1-projection is a 'blind' projection to the L1-ball as opposed to the lasso optimisation that projects the unconstrained embedding by minimising a loss constrained by an L1 penalty. The latter method may preserve the accuracy of the embedding by having a softer constraint. Further investigation should compare out-of-sample error of the embedding under both constraints to see if LassoSparse provides superior performance.

5.3.4 Controlling $\mathcal{C}$

Results presented in fig. 5.5 to fig. 5.8 present performance and computational costs as the size of $\mathcal{C}$ is controlled. The lossy algorithm as described in Appendix C.1.1 works by using lasso to project new candidate compression states onto a sparse basis formed from the existing compression set $\mathcal{C}$. If the error is greater than the δ_{lossy} tolerance then the candidate state is added to $\mathcal{C}$. Therefore the smaller δ_{lossy} tolerance, the larger the compression set $\mathcal{C}$ such that a richer sparse basis can be formed. The empirical results for all MDPs are consistent with this. Referring to the previous discussion about how cost of the contraction constraint, then controlling the size of $\mathcal{C}$ has a direct effect on the cost of planning as demonstrated in the empirical results.

Engel et al. (2004, 2003) develop a similar online sparsification method for kernel and Gaussian process (GP) function approximators that is superficially similar to the compressed embedding approach in theorem C.1.1. Engel et al. (2003) develop a model-free Gaussian process temporal difference (GPTD) algorithm that maintains the value function as a GP over a sparse set of states $\mathcal{C}' = \{\boldsymbol{\phi}(\mathbf{c}_j)\}_{j=1}^{m}$ projected in feature space $\mathcal{F}$, where $\mathbf{c}_j \in \mathcal{S}$. At any t^{th} time step a new state $\mathbf{s}$ is added to $\mathcal{C}'$ if

$$|| \sum_{j=1}^{m} \alpha_j \boldsymbol{\phi}(\mathbf{c}_j) - \boldsymbol{\phi}(\mathbf{s}) ||_{\mathcal{F}}^2 > \delta,$$

where δ is a given threshold. The sparse set of states prevents the computational cost of each GPTD value function update from growing uncontrollably with the number of samples seen. The difference with the CCME approach (see section C.1.1) is that it is a model-based RL algorithm such that $\boldsymbol{\alpha}$ is a function of the state-action conditioning variable for a conditional mean embedding. Furthermore the CCME sparsification not only i) constrains $\boldsymbol{\alpha}$ with the L1-norm due to the contraction constraint, but it also ii) directly relates δ with policy improvement suboptimality guarantees. For future work it would interesting to compare sparsification $\mathcal{C}'$ and compression $\mathcal{C}$ sets of the GPTD and CCME respectively.

5.4 Conclusion

Referring section 7.1, the CCME algorithm generally outperforms the CME (modified with the Schur kernel inverse and fast L1-projection contraction constraint). However the following issues motivate further development of embeddings-based policy iteration.

1. Lasso contraction constraint - Not only is this constraint (and all others used thus far) post hoc (i.e. it is not enforced during model training), but its computational cost is too high for a greedy policy-based algorithm which is linear in $|\mathcal{A}|$.

2. The cost of a full model relearn which includes cross validating the bandwidth for basis $\mathcal{B}$ is too high for online updates.

In the same way that the DPCME (see section 4.5.1) was motivated by the requirement that feature richness is not a function of feature dimensionality, improving the CCME would also benefit from this type of function approximator. The next chapter therefore fuses non-parametric value function approximation with a deep neural representation over state-actions.

Chapter 6

Differentiable Sparse CME Policy Iteration

Both the PCME in Chapter 4 and CCME in Chapter 5 are polar opposites. The PCME approximates the value function parametrically thus the model-based projected Bellman operator must be solved using LSTD or minimising the Bellman residual. Evidence has been provided that demonstrates approximate policy iteration instability and its mitigation with conservative policy improvement. However its computational complexity is suitably decoupled from the size of the training set such that model construction and planning is practical in data abundant settings. The CCME approximates the value function non-parametrically such that an approximate model-based Bellman operator can be solved exactly. The resulting policy iteration algorithm is stable and sparsification techniques are used to decouple the algorithm's computational complexity from the size of the training set. However both state-action feature learning and maintaining a post hoc contraction constraint are computationally expensive.

The work in this chapter can be considered a hybrid of both approaches where i) value functions remain modelled non-parametrically in order to retain exact policy evaluation and ii) the state-action feature representation is replaced with a neural network such that the embedding can be learnt using stochastic gradient descent (SGD) (see Appendix B.5 for a review). This hybrid approach models the embedding weights $\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})$ as a neural network and the contraction constraint is maintained by a softmax-derived activation function on the last layer. This architecture avoids expensive proper CME evaluation costs (therefore planning is also cheap) and model updates require only minibatch SGD. Explorative pseudo-PI with the deep embedding is shown to learn good policies for all test MDPs. Particular focus is on maintaining the compression set $\mathcal{C}$ and more importantly it is found that pruning basis functions is required in order for policy iteration to work. The final algorithm presented is known

as actively compressed conditional mean embeddings (ACCME) and uses a biologically inspired neuron pruning mechanism to maintain $\mathcal{C}$.

6.1 Differentiable CMEs: Deep Embeddings

The deep CME (DCCME) is the first variant that directly modifies the CCME embedding and retains the δ-lossy compression algorithm. The contraction constraint is maintained by architecture construction and therefore it is present during model learning and model evaluation as opposed to a post hoc operation. The actively compressed CME (ACCME) absorbs the final outstanding δ-lossy component into the model's stochastic update itself.

6.1.1 DCCME

Online Dynamics Model

Notation is adopted from the CCME. Given $\mathcal{C}$, the batch CME penalised ERM problem (see equation (2.54)) is modified by replacing the kernel conditional weights with a vector-valued neural network $\boldsymbol{\alpha}_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|\mathcal{C}|}$,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \Big[\frac{1}{2n_k} \sum_{i=1}^{n_k} ||\mu_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i) - \boldsymbol{\phi}(\mathbf{s}'_i)||^2_{\mathcal{F}} + \Omega(\boldsymbol{\theta}) \Big], \tag{6.1}$$

where $\mu_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}) = \boldsymbol{\Phi}_{\mathcal{C}}^{\top} \boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$, $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot) := [\alpha_1(\cdot), .., \alpha_{|\mathcal{C}|}(\cdot)]^{\top}$, $\boldsymbol{\Phi}_{\mathcal{C}} := [\phi(\mathbf{s}'_1), .., \phi(\mathbf{s}'_{|\mathcal{C}|})]^{\top}$, network weights $\boldsymbol{\theta}$, value function space $\mathcal{F} = \mathcal{H}_L$ which is an RKHS of functions over states with kernel $L : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ such that $\phi(\mathbf{s}') = L(\mathbf{s}', \cdot)$, and a real-valued function Ω. Due to the class of the $\boldsymbol{\alpha}$ function approximator, it is no longer possible to select a normed hypothesis class defined by the vector-valued RKHS norm. Instead an appropriate regulariser function such as the L2 norm $\Omega(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||^2_2$ used in penalised ERM (Appendix B.4.1) can be used to prevent both overfitting to noise and network weights growing uncontrollably. This approach is useful when there is a training data budget such that a restriction of the function class is required to inhibit overfitting (see discussion in Appendix B.4.1). However deep learning practitioners usually assume data abundance as a form of regularisation and forgo implementing a regularised loss. For the initial implementation the discussion regarding Ω is deferred until ACCME's description.

As described in section B.5, to train the network requires implementation of the chain rule. Multiplying out the squared loss

$$\frac{1}{2n_k}\sum_{i=1}^{n_k}||\mu_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)-\boldsymbol{\phi}(\mathbf{s}'_i)||^2_{\mathcal{H}_L} = \frac{1}{2n_k}\sum_{i=1}^{n_k}\Big\langle \mu_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)-\boldsymbol{\phi}(\mathbf{s}'_i), \mu_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)-\boldsymbol{\phi}(\mathbf{s}'_i)\Big\rangle_{\mathcal{H}_L}$$
$$= \frac{1}{2n_k}\sum_{i=1}^{n_k}\boldsymbol{\alpha}_{\boldsymbol{\theta}}^\top(\mathbf{s}_i,\mathbf{a}_i)\mathbf{L}_{\mathcal{C}\mathcal{C}}\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i) - 2\boldsymbol{\alpha}_{\boldsymbol{\theta}}^\top\mathbf{L}_{\mathcal{C}\mathbf{s}'_i} + L(\mathbf{s}'_i,\mathbf{s}'_i),$$

where $\mathbf{L}_{\mathcal{C}\mathbf{s}'_i} := [L(\mathbf{c}_1,\mathbf{s}_i),...,L(\mathbf{c}_{|\mathcal{C}|},\mathbf{s}_i)]^\top$ and $(\mathbf{L}_{\mathcal{C}\mathcal{C}})_{ij} = L(\mathbf{c}_i,\mathbf{c}_j)$. In the batch setting the derivative of equation (6.1) is taken wrt a single scalar weight $\theta \in \boldsymbol{\theta}$ somewhere in the network and batch gradient descent is used to minimise the loss by updating all weights in $\boldsymbol{\theta}$. However the common approach is to sample the gradient by drawing a minibatch from the training data $\hat{\mathcal{D}} \sim \mathcal{D}_{n_k}$,

$$\frac{\partial\hat{\mathcal{L}}_{\boldsymbol{\alpha}}}{\partial\theta}\bigg|_{\hat{\mathcal{D}}} = \frac{1}{|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|}\frac{\partial\hat{\mathcal{L}}}{\partial\boldsymbol{\alpha}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{\alpha}_{\boldsymbol{\theta}}}{\partial\theta}\bigg|_{\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)} + \partial_\theta\Omega(\boldsymbol{\theta}), \tag{6.2}$$

$$= \frac{1}{|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|}\Big(\boldsymbol{\alpha}_{\boldsymbol{\theta}}^\top(\mathbf{s}_i,\mathbf{a}_i)\mathbf{L}_{\mathcal{C}\mathcal{C}} - \mathbf{L}_{\mathbf{s}'_i\mathcal{C}}\Big)\frac{\partial\boldsymbol{\alpha}_{\boldsymbol{\theta}}}{\partial\theta}\bigg|_{\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)} + \partial_\theta\Omega(\boldsymbol{\theta}), \tag{6.3}$$

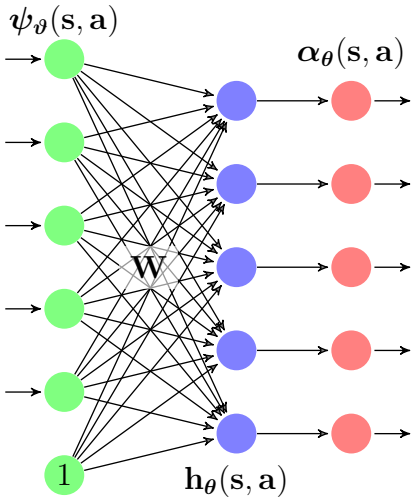
which is then used in one of many SGD minimisation algorithm variants (see Appendix B.5.2) to minimise equation (6.1). The last term is a subgradient which generalises the derivative of functions that are not continuous whose description is deferred until the next section. The derivative $\frac{\partial\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)}{\partial\theta}$ is unravelled by applying the chain rule to the deep network as described by backpropagation in Appendix B.5.2. The novelty here is that a deep parametric function approximator is trained using a non-parametric RKHS loss. The requirement for maintaining a proper CME such that explorative pseudo-PI can be used for policy optimisation forces us to use this non-parametric value function class and therefore the adoption of the hybrid neural-kernel approach.

Architecture

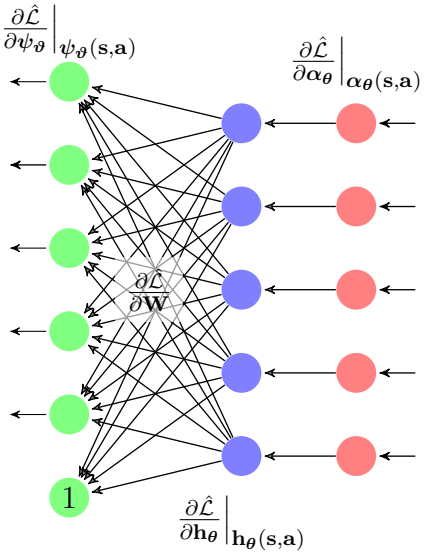
As discussed in Chapter 5, a proper CME decomposes its conditional weights into $\boldsymbol{\alpha}(\cdot) = \text{Proj}_{||\boldsymbol{\alpha}||_1 \leq 1}(\mathbf{W}\boldsymbol{\psi}(\cdot))$ where $\boldsymbol{\psi}(\cdot) = [K(\mathbf{b}_1,\cdot),...,K(\mathbf{b}_{|\mathcal{B}|},\cdot)]^\top$ is non-parametric representation over $\mathcal{S}\times\mathcal{A}$ whose size grows linearly with training data and $\mathbf{W}$ is a weight matrix. The CCME (Chapter 5) maintains a compact basis $\mathcal{B}$ that limits this representation from growing as new training data is gathered during the exploration of an MDP. The projection operation is performed only when the embedding is evaluated and is not involved during the calculation of $\mathbf{W}$ in the primal.

DCCME naturally adopts this decomposition into a neural architecture (see fig. 6.1) such that $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot) = \boldsymbol{\sigma}(\mathbf{W}\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot))$ where $\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot)$ is the penultimate layer (and is the last layer of the remaining part of the network), $\mathbf{W}$ is the final weight layer that maps $\mathbb{R}^{\dim(\psi)} \to \mathbb{R}^{|\mathcal{C}|}$, $\boldsymbol{\sigma}$ is an activation function that enforces the contraction constraint

Figure 6.1 Last layers of the DCCME's vanilla feedforward architecture, $\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) = \mathbf{W}\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s},\mathbf{a})$ and $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) = \boldsymbol{\sigma}(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}))$ with $\boldsymbol{\sigma}$ as a softmax layer.



ii) Forward inference.



ii) Backpropagate error signal.

and $\boldsymbol{\theta} = \boldsymbol{\vartheta} \cup \mathbf{W}$. Enriching the $\boldsymbol{\psi}$ representation no longer requires an increase in its dimensionality (equivalent to the number of nodes). The deep representation's expressiveness instead depends on the network architecture and most importantly the abundance of training data. All hidden layers are ReLUs and weights are trained using the Adam SGD variant (see Appendix B.5.2 for details). Weight initialisation and Adam optimisation parameters are specified in table 6.1. To maintain speed but also an expressive representation (as inspired by Shi et al. (2016)), the hidden layers below $\boldsymbol{\psi}_{\theta}$ were arranged in a narrow but deep configuration as described by fig. 6.5. The $\psi_{\boldsymbol{\vartheta}}$ was purposefully larger than its predecessors to induce a rich feature representation. It was found that good performance was achieved when $\dim(\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot))$ was at least twice the size of $|\mathcal{C}|$. Many different architectures were experimented with and none were as successful as this topology, which remains constant between all control tasks without requiring cross-validation.

Contraction Constraint

In the initial development of DCCME, the post hoc contraction constraints used in the CME-L1ProjSparse and CCME-LassoSparse were experimented with. However post hoc constraints of this class are not differentiable and have to remain as separate operations to model learning. Instead a softmax activation function $\boldsymbol{\sigma}(\mathbf{h}_{\boldsymbol{\theta}}(\cdot)) = \text{softmax}(\mathbf{h}_{\boldsymbol{\theta}}(\cdot))$ was used to fulfil the pseudo-MDP contraction constraint $||\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})||_1 \leq 1 \; \forall(\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}$, whose cost during a single inference pass of the network

Table 6.1 DCCME architecture hyperparameters (all MDPs)

Param	Value
δ_{lossy}	See table 5.1
L bandwidth	See table 5.1
Hidden ReLU layers	$\{100, 200, 200, 200, 200, 200, 1000\}$, see fig. 6.5
$\boldsymbol{\vartheta}$ weights init. (μ, σ^2)	$(0, 0.01)$
$\mathbf{W}$ weights init. (μ, σ^2)	$(0, 1{\times}10^{-4})$
Adam initial rate α	$4{\times}10^{-4}$ (mountain car/cart-pole), $1{\times}10^{-4}$ quads
Adam (β_1, β_2)	$(0.9, 0.99)$
Minibatch size $\lvert\hat{\mathcal{D}}\rvert$	20 samples
Minibatch count	3000 per k^{th} policy iteration

is $\sim\mathcal{O}(|\mathcal{C}_k|)$. The backprop pass is also linear in $|\mathcal{C}_k|$ as described in Martins and Astudillo (2016, equation 13).

Compression Set $\mathcal{C}$

Managing $\mathcal{C}$ was first implemented using the δ-lossy algorithm such that at the k^{th} iteration $\mathcal{C}_k \leftarrow \mathcal{C}_{k-1} \cup \mathcal{C}_{\text{new}}$. Incrementing $\mathcal{C}$ requires an adjustment to the network topology such that $\boldsymbol{\alpha}$ is likewise increased. This corresponds to adding new rows of weights $\mathbf{W} \leftarrow [\mathbf{W}^\top, \mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{C}_{\text{new}}|}]^\top$ where each new weight $\mathbf{w}_j \in \mathbb{R}^{\dim(\boldsymbol{\psi}_{\boldsymbol{\theta}})}$ corresponds to each new basis function in $\mathcal{C}_{\text{new}}$.

Note that the initialisation of these final layer weights $\mathbf{W}$ is treated slightly differently to the rest of the network, notably with a smaller variance than the rest of the network, whose reasoning is described as follows. At the *moment* new weights are added such that $\dim(\boldsymbol{\alpha}_{\boldsymbol{\theta}})$ is increased, it is desirable to minimise their contribution to the the existing function approximator's output such that its accuracy is not diminished. However it is also desirable in an SGD algorithm to non-deterministically initialise weights rather than set them to zero such that any two new neurons will output differently. The compromise is therefore to initialise them stochastically but closer to zero. Note also that the initial learning rate for Adam optimisation is different for the quadrocopter MDPs for maximum performance and is probably due to the difference in state-action space dimensionality.

6.1.2 Pruning $\mathcal{C}$ During SGD

The δ-lossy algorithm only adds new basis functions to $\mathcal{C}$ which may be problematic for large MDPs. Empirical evidence presented below suggests that for the hardest MDP, $\mathcal{C}$ keeps on growing which stifles further policy learning. There is also a quadratic dependency on $\mathcal{C}_k$ during explorative PI (see table 7.1 for more details) because the

softmax activation function is not sparse in $\mathcal{C}_k$ (cf. the CCME's lasso projection and fast L1-projection which are both sparse in $\mathcal{C}_k$).

Contraction Constraint

Setting $\boldsymbol{\sigma}(\cdot) = \textsc{TopNMax}(\cdot) := \textsc{Softmax}\big(\textsc{KeepTopN}(\cdot)\big)$ (Shazeer et al., 2017) as the final layer's activation function, then the pseudo-MDP contraction constraint is still satisfied $\forall(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. The network output $\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a}) \in \mathbb{R}^N$ is fixed N-sparse where N is chosen a priori. The $\textsc{KeepTopN}(\cdot)$ function sorts the raw output of the layer $\mathbf{h}_{\boldsymbol{\theta}}(\cdot)$ which costs $\sim\mathcal{O}(N|\mathcal{C}_k|)$ for every inference pass of the network per state-action pair. The sparse activations are then either used to evaluate the embedding, or they activate only a subset of kernel matrix elements in equation (6.2) during training. During backprop the sparse gradient signal is passed back along the activated elements of $\boldsymbol{\alpha}_{\boldsymbol{\theta}}$ and into the network as described by (Shazeer et al., 2017) and implemented in the embedding as shown in fig. 6.3. By gradient signal is only sparse in the last weight layer and by the time it reaches the layer $\boldsymbol{\psi}_{\boldsymbol{\vartheta}}$, it is dense once more.

Compression Set

An alternative mechanism to maintain $\mathcal{C}$ is described below and is built into the actual model learning process that adaptively removes successor states during SGD-based training and can either be used on its own or in conjunction with the δ-lossy algorithm.

Described in Appendix B.5.3 are several existing weight shrinkage methods such as rounding and soft-thresholding. Described also is the online truncated gradient (Langford et al., 2009) which solves the lasso (Tibshirani, 1996) optimisation problem in the stochastic setting by applying soft-thresholding with strength $\eta\lambda$, only if the weight is within a specified range around the origin i.e. $|w| \leq \omega$. This is a mixture of soft-thresholding and weight rounding and is claimed to provide reliable sparse solutions in the stochastic setting. Although features are assumed linearly independent and the loss function is assumed smooth-convex, in the following work this sparsification technique is adapted to the structural sparsity setting in a neural network, labelled as *truncated group shrinkage*. In the sequel a biologically inspired modification is developed known as *modified truncated group shrinkage* which is found to require less fine-tuning between control tasks.

Truncated Gradient Method We focus on how the last layer weights of a deep embedding are updated. If $\mathbf{W} := [\mathbf{w}_1, ..., \mathbf{w}_m]^\top \in \mathbb{R}^{m \times d}$ then the j^{th} row of this matrix is $\mathbf{w}_{j:} = \mathbf{w}_j^\top \in \mathbb{R}^{1 \times d}$. The sparsification strategy is to develop a mechanism whereby group sparsity is induced in $\mathbf{W}$ where each row is a group. There is a one-to-one mapping between the j^{th} row $\mathbf{w}_{j:}$ and the j^{th} member of $\mathcal{C}$ such that if a row is driven

to zero then its corresponding member $\boldsymbol{\phi}(\mathbf{c}_j)$ can be removed from $\mathcal{C}$. During backprop the vanilla Adam updates are modified by the truncated gradient approach where first a vanilla Adam update is made and then the mapping $\mathcal{T}:\mathbb{R}^d \to \mathbb{R}^d$ is applied (as summarised in Appendix B.5.3),

$$\mathbf{w}_{j:} \leftarrow \mathcal{T}\left(\mathbf{w}_{j:} - \eta \nabla_{\mathbf{w}_{j:}} \hat{\mathcal{L}}\big|_{\hat{\mathcal{D}}}, \boldsymbol{\omega}\right), \qquad \forall \mathbf{w}_{j:} \in \mathbf{W}, \tag{6.4}$$

where $\boldsymbol{\omega}$ is a set of hyperparameters. The existing truncated gradient approach develops a mapping $\mathcal{T} = \mathcal{T}_{\text{trunc}}$ that combines the well known soft-thresholding operation $\mathcal{T}_{\text{soft}}$ (related to lasso) and a rounding operation $\mathcal{T}_{\text{round}}$ as described in Appendix B.5.3. The motivation is that shrinkage is limited to a small range centred at the origin which promotes less aggressive sparsification.

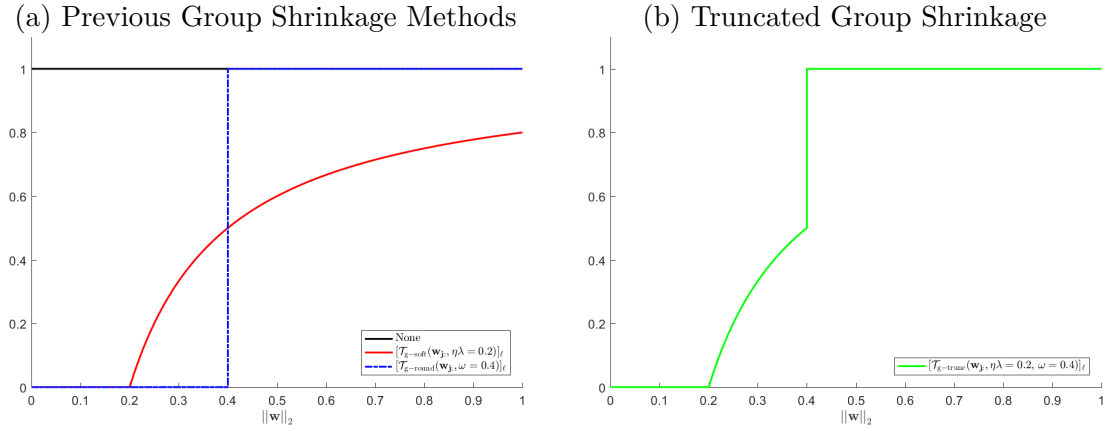
However, when this approach is specialised to the embedding's sparsification requirements, then equation (6.4) must use a mapping $\mathcal{T} = \mathcal{T}_{\text{g-trunc}}$ based on *group*-level shrinkage related to group lasso (Yuan and Lin, 2006). We therefore first develop $\mathcal{T}_{\text{g-trunc}}$. However it is found that the hyperparameters of this method require fine tuning for each control task. *Modified truncated gradient* is developed in response to this where a group shrinkage operation $\mathcal{T}_{\text{mod-g-trunc}}$ are applied not when weight groups $\mathbf{w}_{j:}$ are within $||\mathbf{w}_{j:}||_2 \leq \omega$ but instead for weight groups *not updated* by the sparse backprop signal. 'Inactive' weight groups are therefore diminished at any one backprop pass. If over the course of a training epoch (limited to a set number of minibatches) weight groups have remained at zero then they and their corresponding members in $\mathcal{C}$ are removed. This approach is novel and is inspired by a biological 'use-it-or-lose-it' neuronal strategy, empirically shown to work surprisingly well in controlling the output size of the embedding while facilitating learning good policies.

Truncated Group Shrinkage for Softmax Embeddings The final activation function (contraction constraint) is assumed to be $\boldsymbol{\sigma}(\cdot) = \textsc{Softmax}(\cdot)$. Just as the lasso's soft-thresholding operator (equation (B.59)) was adapted to the truncated shrinkage operator as described by equation (B.78), then the group lasso's soft-thresholding operator (equation (B.58)) is adapted to the truncated shrinkage operator,

$$\left[\mathcal{T}_{\text{g-trunc}}(\mathbf{w}_{j:}, \eta\lambda, \omega)\right]_\ell = \begin{cases} \left[1 - \frac{\eta\gamma}{||\mathbf{w}_{j:}||_2}\right]_+ w_{j\ell} & ||\mathbf{w}_{j:}||_2 \leq \omega, \\ w_{j\ell} & \text{otherwise}, \end{cases} \tag{6.5}$$

where each group of weights $\mathbf{w}_{j:}$ is a row in $\mathbf{W}$ and $w_{j\ell}$ is the ℓ^{th} weight in the $\mathbf{w}_{j:}$ group. The group truncated shrinkage diminishes all weights in a group if the L2-norm of the group is within $[0, \omega]$, otherwise no shrinkage occurs. Note that as $\omega \to \infty$ then $\mathcal{T}_{\text{g-trunc}} \to \mathcal{T}_{\text{g-soft}}$. All $\mathcal{T}$ variants are visualised in fig. 6.2 (cf. fig. B.6).

Figure 6.2 Group Shrinkage weight multiplier: $\eta\lambda = 0.2$ and $\omega = 0.4$



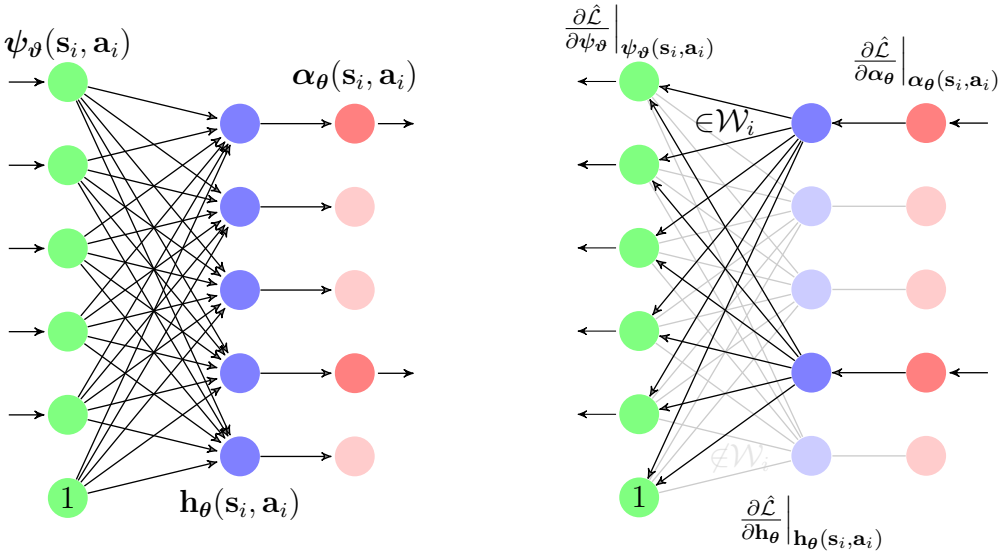
Modified Truncated Group Shrinkage for TopNMaxSparse Embeddings It was found that (ω, λ) pairs needed to be fine-tuned for each task when using group truncated gradient with softmax embeddings. In response to this, a novel modification of the truncated group shrinkage is proposed for embeddings whose last activation is $\boldsymbol{\sigma}(\cdot) = \textsc{TopNMax}(\cdot)$. Referring to fig. 6.3, a sparsity mechanism is pursued that *exploits* the *sparse* backprop gradient signal when updating $\mathbf{W}$. For the i^{th} data sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ in a minibatch $\hat{\mathcal{D}} \sim \mathcal{D}$, then during feed forward inference the embedding's last layer is sparse-activated. Likewise during backprop's backwards pass the gradient signal is only propagated back through these activated neurons (Shazeer et al., 2017). Consequently only a subset $\mathcal{W}_i$ of rows in $\mathbf{W}$ will be updated. Over an entire minibatch of samples then the set $\mathcal{W}_{\hat{\mathcal{D}}} = (\dot{\bigcup}_{i=1}^{|\hat{\mathcal{D}}|} \mathcal{W}_i)$ is the union of all active weight groups. We define $\tilde{\mathcal{W}}_{\hat{\mathcal{D}}}$ identical to $\mathcal{W}_{\hat{\mathcal{D}}}$ but whose elements do *not* include bias weights. Therefore $\tilde{\mathcal{W}}_{\hat{\mathcal{D}}}$ defines all active weight groups without their bias terms. The definition of the modified truncated group shrinkage can now be stated,

$$\Big[\mathcal{T}_{\text{mod-g-trunc}}(\mathbf{w}_{j:}, \eta\lambda_{21}, \tilde{\mathcal{W}}_{\hat{\mathcal{D}}})\Big]_\ell = \begin{cases} \Big[1 - \frac{\eta\lambda_{21}}{||\mathbf{w}_{j:}||_2}\Big]_+ w_{j\ell} & \mathbf{w}_{j:} \notin \tilde{\mathcal{W}}_{\hat{\mathcal{D}}}, \\ w_{j\ell} & \text{otherwise.} \end{cases} \tag{6.6}$$

In the final implementation an L2 penalty is added to any active weight update. This was experimentally found to be *vital* such that extreme weight magnitudes were mitigated. The adding and removing of final layer weights groups proved to be a disruptive activity for the embedding function approximator and therefore the presence of an L2-regulariser helped stabilise this process. Note that as the set $\tilde{\mathcal{W}}_{\hat{\mathcal{D}}}$ becomes dense to include all weight groups in $\mathbf{W}$ then $\mathcal{T}_{\text{mod-g-trunc}} \rightarrow \mathcal{T}_{\text{g-soft}}$. The difference between $\mathcal{T}_{\text{g-trunc}}$ and $\mathcal{T}_{\text{mod-g-trunc}}$ is the definition of the 'truncation'; $\mathcal{T}_{\text{g-trunc}}$ truncates the application of group shrinkage to weight groups whose magnitude is within the range $[0, \omega]$, whereas $\mathcal{T}_{\text{g-trunc}}$ limits its application to inactive weight groups induced by the sparse TopNMax activation function.

Figure 6.3 Adapting $\boldsymbol{\sigma}(\cdot) = \textsc{TopNMax}(\cdot)$ (Shazeer et al., 2017) as the last layer activation function in a deep embedding with vanilla Adam weight updates, where $\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i) = \mathbf{W}\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i, \mathbf{a}_i)$ and $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i) = \boldsymbol{\sigma}(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{a}_i))$: i) Sparse inference only activates output nodes $\{\alpha_j(\mathbf{s}_i, \mathbf{a}_i)\}_{j=\{1,4\}}$. ii) The sparse backprop pass updates only $\mathcal{W}_i := \{\mathbf{w}_{j:}\}_{j=\{1,4\}}$ in the last layer where $\mathbf{w}_{j:}$ is the j^{th} row in $\mathbf{W}$. Note that $\{\mathbf{w}_{j:}\}_{j=\{2,3,5\}} \notin \mathcal{W}_i$.

i) TopNMax forward inference. ii) Backpropagate sparse error signal.



6.1.3 ACCME

A TopNMax embedding is used with the modified truncated group shrinkage operation $\mathcal{T}_{\text{mod-g-trunc}}$ without using the δ-lossy operation for maintaining $\mathcal{C}$. The sparse activation function is central to i) maintaining the pseudo-MDP contraction constraint, ii) providing a cheap and sparse evaluation of the embedding for planning/inference and iii) induces a sparse gradient signal that is exploited by the newly developed modified truncated gradient operation to maintain $\mathcal{C}$.

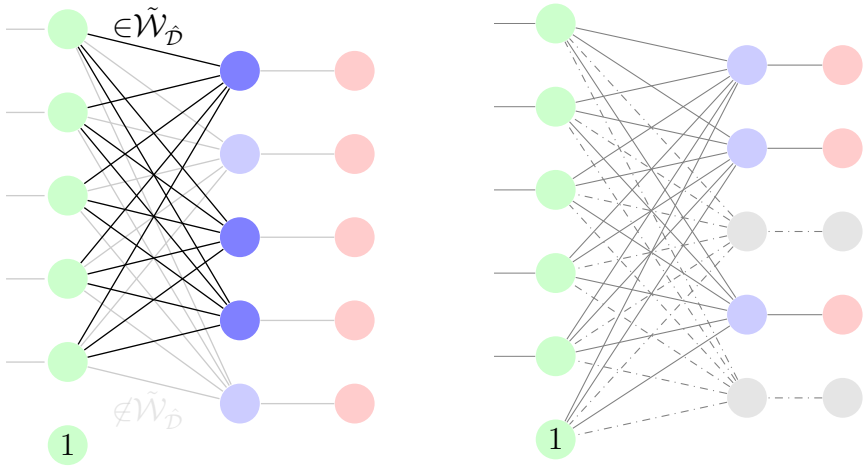
At each k^{th} policy iteration, the embedding is trained using an epoch of $M = 3000$ minibatches drawn $\hat{\mathcal{D}} \sim \mathcal{D}$ from an experience replay memory (Lin, 1992). Each epoch is split into four phases:

1. **Initial phase** (fig. 6.3): All new successor states $\mathcal{S}'_{\text{new}}$ discovered during data acquisition are added as basis functions to $\mathcal{C}$. Corresponding weight groups (rows) are initialised and added to $\mathbf{W}$. Vanilla Adam SGD is executed over $\frac{M}{3}$ minibatches to train the entire network $\boldsymbol{\alpha}_{\boldsymbol{\theta}}$ driven by the RKHS loss. The last layer weights in $\mathbf{W}$ experience sparse updates due to the sparse final activation function. All updated weights in the last layer are penalised with the L2-norm.

Figure 6.4 $\mathcal{T}_{\text{mod-g-trunc}}$ modified truncated gradient: i) $\tilde{\mathcal{W}}_{\hat{\mathcal{D}}}$ defines the set of weight groups (without bias terms) that were updated by Adam in minibatch $\hat{\mathcal{D}}$. Group-shrinkage is applied to all *inactive* weight groups i.e. $\forall \mathbf{w}_{j:} \notin \tilde{\mathcal{W}}_{\hat{\mathcal{D}}}$. ii) At the end of the shrinkage backprop phase, $\forall \mathbf{w}_{j:} \in \mathbf{W}$ whose length (excluding bias terms) $||\mathbf{w}_{j:}||_2 == 0$ (dashed/grey) are removed.

(a) Active/inactive weight groups for a single minibatch $\hat{\mathcal{D}}$.

(b) Remove groups where $||\mathbf{w}_{j:}||_2 == 0$ (grey).



2. **Weight shrinkage phase** (fig. 6.4a): Vanilla Adam SGD is executed over $\frac{M}{3}$ minibatches for the entire network as in the initial stage. All last-layer active weight updates include an L2-regularisation penalty. Inactive last layer weights are shrunk group-wise by applying $\mathcal{T}_{\text{mod-g-trunc}}$ once every minibatch. It was found that setting a constant learning rate $\eta = 1$ for this operation induced adequate sparsification.

3. **Weight and $\mathcal{C}$ pruning phase** (fig. 6.4b): For all groups whose $||\mathbf{w}_{j:}||_2$ that have remained at zero for the last $\frac{3}{4}\frac{M}{3}$ minibatches, are removed from $\mathbf{W}$ along with their corresponding basis functions in $\mathcal{C}_k$.

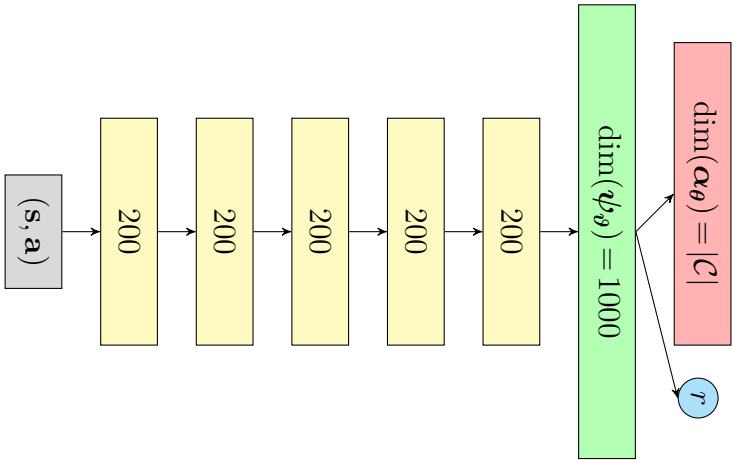
4. **Consolidation phase** (fig. 6.3): Vanilla Adam SGD is executed over $\frac{M}{3}$ minibatches to train the entire network $\boldsymbol{\alpha}_{\boldsymbol{\theta}}$ with L2-regularisation applied to the active weights in the last layer.

The regularisation parameters used are common for each task and are listed in table 6.2. Surprisingly fine tuning was not needed.

Table 6.2 ACCME architecture hyperparameters (in addition to table 6.1)

Param	Value
$\mathbf{W}$ L2 regulariser λ_2 (all MDPs)	1×10^{-4}
$\mathbf{W}$ L21 regulariser λ_{21} (all MDPs)	0.1

Figure 6.5 ACCME architecture: yellow and green are weighted sum layers with ReLU activations, red is weighted sum with TopNMax, blue is scalar-valued weighted sum only. DCCME's red layer is a dense softmax.



6.1.4 ACCME-R: Learning the Reward Function

Throughout this investigation the reward function is assumed known. However ACCME-R demonstrates that the reward function can easily be learnt by exploiting the penultimate layer as the state-action feature representation. The loss is

$$\mathcal{L}_r\big|_{\hat{\mathcal{D}}} := \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|} ||r_{\boldsymbol{\beta}}(\mathbf{s}_i, \mathbf{a}_i) - r_i||_2^2,$$

where reward function approximator is $r_{\boldsymbol{\beta}}(\mathbf{s}, \mathbf{a}) = \boldsymbol{\beta}^\top \boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i, \mathbf{a}_i)$. Reward weights $\boldsymbol{\beta}$ are updated using

$$\nabla_{\boldsymbol{\beta}}\mathcal{L}_r\big|_{\hat{\mathcal{D}}} = \frac{1}{|\hat{\mathcal{D}}|}\sum_{i=1}^{m}\Big(r_{\boldsymbol{\beta}}(\mathbf{s}_i, \mathbf{a}_i) - r_i\Big)\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i, \mathbf{a}_i), \tag{6.7}$$

using backprop. The derivative signal $\frac{\partial \mathcal{L}_r}{\partial \psi_{\boldsymbol{\vartheta}}}\big|_{\hat{\mathcal{D}}}$ is collected with the embedding's $\frac{\partial \mathcal{L}_{\boldsymbol{\alpha}}}{\partial \psi}\big|_{\hat{\mathcal{D}}}$ and backpropagated to the rest of the network during training.

6.2 Experiments

Algorithms 22 and 23 describe the full ACCME-R implementation. DCCME-Softmax /TopNMax use δ-lossy and ACCME variants use the modified group truncated shrinkage operation to maintain $\mathcal{C}$. Experiments were carried out on the same MDPs as previous chapters. The neural network code is a custom MATLAB implementation created specifically for this investigation. No hardware acceleration or asynchronous programming were used such that a fair comparison could be made with previous algorithms. Embeddings were built with an identical vanilla feed-forward architecture for all MDPs. It was found that during a weight shrinkage update, the learning rate

Algorithm 22 ACCME-R Policy Iteration

Input: Kernel $L:\mathcal{S}\times\mathcal{S}\to\mathbb{R}$, implicit state representation $\boldsymbol{\phi}(\mathbf{s}'):=L(\mathbf{s}',\cdot)$, access to unknown MDP $\mathcal{M}:=\{\mathcal{S},\mathcal{A},P,P_1,r,\gamma\}$ with continuous $\mathcal{S}$, discrete $\mathcal{A}$, unknown average reward function $r:\mathcal{S}\times\mathcal{A}\to[0,1]$. $H=100$ transitions per trajectory, $J_{\text{imp}}:=10$, $J_{\text{eval}}:=4000$, start-state distribution P_1, minibatch size $|\hat{\mathcal{D}}|=20$, minibatch epoch count $M=3000$, shared network $\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot,\cdot)$, TopNMax count N.

Parameters: Sample count $n_{\text{new}}{=}2H$, $\lambda_2{=}1{\times}10^{-4}$, $\lambda_{21}{=}1{\times}10^{-1}$, $J_{\text{imp}}{=}10$, $J_{\text{eval}}{=}4000$, compression set $\mathcal{C}$.

Output: $\pi_\kappa(\cdot)\approx\pi^*(\cdot)$.

Initialise: $\boldsymbol{\vartheta}$, $\boldsymbol{\beta}\sim\mathcal{N}(0,\sigma^2{=}0.01)$, $\mathbf{W}^{(0)}{=}\emptyset$, $\mathcal{C}_0{=}\emptyset$, $\hat{q}_0(\cdot,\mathbf{a})\leftarrow r_{\boldsymbol{\beta}}(\cdot,\mathbf{a})$, $\pi_1(\cdot)\leftarrow\text{greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}_0(\cdot,\mathbf{a})]$, exploration policy $\nu^{\pi_1}(\cdot)\leftarrow\epsilon\text{-greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}(\cdot,\mathbf{a})]$, $\epsilon\leftarrow0.3$, $n_0\leftarrow0$, $n_{\text{new}}\leftarrow2H$, $\mathcal{D}_0=\emptyset$, $\mathcal{C}_0\leftarrow\emptyset$.

for $k=1,2,...,\kappa{-}1$ **do** ▷ k^{th} policy iteration master index

  Data acquisition: $n_k{\leftarrow}n_{k\text{-}1}{+}n_{\text{new}}$, collect $\mathcal{D}_{\text{new}}=\{\mathbf{s}_i,\mathbf{a}_i,r_i,\mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_k}$; one trajectory from each π_k and ν^{π_k}, beginning from start state $\mathbf{s}\sim P_1$, $\mathcal{D}_k{\leftarrow}\mathcal{D}_{k-1}{\cup}\mathcal{D}_{\text{new}}$, $\mathcal{S}'_{\text{new}}\leftarrow\{\mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_k}$. ▷ aggregate data

  Augment $\mathcal{C}$: $\mathcal{C}_k{\leftarrow}\mathcal{C}_{k-1}{\cup}\{\boldsymbol{\phi}(\mathbf{s}'_i)\}_{i=n_{k\text{-}1}+1}^{n_k}$

  $\mathbf{W}^{\text{new}}\in\mathbb{R}^{n_{\text{new}}\times\dim(\psi)}{\sim}\mathcal{N}(0,\sigma^2=1{\times}10^{-4})$, $\mathbf{W}^{(k)}{\leftarrow}[\mathbf{W}^{(k-1)};\mathbf{W}^{\text{new}}]$.

  Model:

  $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot,\cdot)$, $\mathcal{C}_k$, $r_{\boldsymbol{\beta}}(\cdot,\cdot){\leftarrow}\textsc{TrainACCME}\big(\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot,\cdot),\mathcal{C}_k,r_{\boldsymbol{\beta}}(\cdot,\cdot),\mathcal{D}_k\big)$ ▷ algorithm 23

  $\mathbf{v}{:=}[v(\mathbf{s}'_1),.,v(\mathbf{s}'_{|\mathcal{C}_k|})]^\top$, $(\hat{T}^\pi_\mu v)(\cdot){:=}r_{\boldsymbol{\beta}}(\cdot,\pi(\cdot)){+}\gamma\boldsymbol{\alpha}^\top_{\boldsymbol{\theta}}(\cdot,\pi(\cdot))\mathbf{v}$

  Planning:

  for $i=1$ **to** J_{imp} **do** ▷ policy improvement index

    $\mathbf{v}{\leftarrow}\mathbf{0}$, $\pi{\leftarrow}\pi_k$

    for $j=1$ **to** J_{eval} **do** ▷ exact policy evaluation

      $\mathbf{v}{\leftarrow}\hat{T}^\pi_\mu\mathbf{v}$

    end for

    $\hat{\mathbf{v}}^\pi\leftarrow\mathbf{v}$

    $\hat{q}^\pi(\mathbf{s},\mathbf{a}):=r(\mathbf{s},\mathbf{a}){+}\gamma\boldsymbol{\alpha}^\top_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})\hat{\mathbf{v}}^\pi$, $(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}$

    $\pi(\cdot)\leftarrow\text{greedy}_{\mathbf{a}\in\mathcal{A}}[\hat{q}^\pi(\cdot,\mathbf{a})]$ ▷ greedy policy improvement

  end for

  $\pi_{k+1}\leftarrow\pi$

end for

return $\pi_\kappa(\cdot)$

on the L-21 norm was set to a constant $\eta=1$. In all other instances (such as regular weight updates), the learning rate was determined by the Adam optimiser.

6.3 Discussion

Figures 6.6, 6.7, 6.8 and 6.9 present results for all MDPs. Empirical cumulative discounted reward and algorithm component timings are provided. Figure 6.11 com-

Algorithm 23 TRAINACCME($\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot,\cdot), \mathcal{C}, r_{\boldsymbol{\beta}}(\cdot,\cdot), \mathcal{D}$)

for $i = 1$ **to** M **do**
  Draw a minibatch $\hat{\mathcal{D}} \sim \mathcal{D}$
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \nabla_{\boldsymbol{\beta}} \hat{\mathcal{L}}_r \big|_{\hat{\mathcal{D}}}$ ▷ update all reward weights
  if $i \leq \frac{1}{3}M$ **or** $i > \frac{2}{3}M$ **then** ▷ initial or consolidation phase
    $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \hat{\mathcal{L}}_{\boldsymbol{\alpha}} \big|_{\hat{\mathcal{D}}} - \lambda_2 \mathbf{W}$ ▷ update **active** weights with L2 regularisation
  else if $\frac{1}{3}M < i \leq \frac{2}{3}M$ **then** ▷ shrinkage phase
    $\mathbf{G} \leftarrow \nabla_{\mathbf{W}} \hat{\mathcal{L}}_{\boldsymbol{\alpha}} \big|_{\hat{\mathcal{D}}}$
    for each $\mathbf{w}_{j:} \in \mathbf{W}$ **do** ▷ update all last layer embedding weights
      $w_{j\ell} \leftarrow \left[\mathcal{T}_{\text{mod-g-trunc}}(\mathbf{w}_{j:}, \eta\lambda_{21}, \tilde{\mathcal{W}}_{\hat{\mathcal{D}}}) \right]_{\ell}$ ▷ diminish **inactive** weights
    end for
    $\mathbf{W} \leftarrow \mathbf{W} - \eta \mathbf{G} - \lambda_2 \mathbf{W}$ ▷ update **active** weights with L2 regularisation
    if $i == 2M/3$ **then** ▷ pruning phase
      for each $\mathbf{w}_{j:} \in \mathbf{W}$ **do**
        if $||\mathbf{w}_{j:}||_2$ remained 0 over the last $\frac{3}{4}\frac{1}{3}M$ minibatches **then**
          $\mathbf{W} \leftarrow \mathbf{W} \backslash \mathbf{w}_{j:}$, $\mathcal{C} \leftarrow \mathcal{C} \backslash \boldsymbol{\phi}(\mathbf{s}'_j)$
        end if
      end for
    end if
  end if
  $\boldsymbol{\vartheta} \leftarrow \boldsymbol{\vartheta} - \eta \left(\left(\frac{\partial \hat{\mathcal{L}}_{\boldsymbol{\alpha}}}{\partial \psi} + \frac{\partial \hat{\mathcal{L}}_r}{\partial \psi} \right) \frac{\partial \psi}{\partial \boldsymbol{\vartheta}} \Big|_{\hat{\mathcal{D}}} \right)^{\top}$ ▷ update the rest of the network
end for
return $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot,\cdot)$, $\mathcal{C}, r_{\boldsymbol{\beta}}(\cdot,\cdot)$.

pares the size of the compression set $\mathcal{C}$ for all deep embeddings and the previous CCME algorithm. A full comparison between all algorithms is presented in Chapter 7 whose results will be included in this discussion. Unfortunately training a neural network is a non-convex optimisation procedure without guarantees of finding a global minimum. Departure from the kernel embedding architecture also means that the CME consistency guarantees (Grünewälder et al., 2012a)[lemma 2.2] are lost. However these theoretical concerns prove unfounded in the practical setting where the goal is to develop an algorithm that can handle data abundance. Until better understanding of training neural networks is achieved, then the theoretical part of the deep embedding training will remain elusive.

Lecun (2007) points out in his celebrated "who is afraid of non-convex loss functions" lecture that it is not a requirement to find the global minimum to produce good generalisable (out of sample) deep function approximators. Instead training only has to select a good local minimum. Choromanska et al. (2015) present empirical evidence that for large neural architectures, good local minima critical to the loss function exist in a band close to the global minimum and is lower-bounded by the global minimum. It is found that loss surfaces generally consist of saddle points and the chance of converging to a good local minimum (corresponding to good out of

sample generalisation) increases as the network size increases. Modern large-scale neural network architectures trained with large data sets have been shown to generalise surprisingly well out of sample without the problem of overfitting. However the mechanism of generalisation and over-fitting for artificial neural networks is still poorly understood and is an active area of research (Nguyen et al., 2015; Szegedy et al., 2013; Zhang et al., 2017).

6.3.1 DCCME

DCCME-Softmax can be considered as the CCME algorithm whose $\boldsymbol{\alpha}(\cdot)$ weights are replaced with a neural network and whose contraction constraint is maintained by construction. The δ-lossy algorithm appears to work well with the deep function approximator in the first three MDPs. Although the deep embedding's performance is not as efficient as the CCME, the objective of speeding up planning has been achieved. The size of $|\mathcal{C}|$ is much larger for the DCCME when compared to the CCME, despite the δ-lossy hyperparameters being identical. This suggests that the DCCME embedding quality is poorer than the CCME such that more states are experienced before a good policy has been learnt. This issue becomes more apparent for the final holding pattern MDP. Referring to fig. 6.9a it is clear that the DCCME embedding has difficulty learning the task. Extensive experimentation including a δ-lossy hyperparameter search was initiated but without success. It was hypothesised that either embedding evaluation or training was being inhibited by an ever increasing compression set size $|\mathcal{C}|$. The larger $\mathcal{C}$, the more spread out $\boldsymbol{\alpha}(\cdot)$ weights will be over the successor states which may stifle accurate embedding accuracy. A compounding cycle of ever-increasing $\mathcal{C}$ size, embedding inaccuracy and therefore inability to learn the control task leads the agent to encounter an excessive number of successor states that are further added to $\mathcal{C}$. Evidence presented for the DCCME-Softmax in fig. 6.11 supports this hypothesis.

Two techniques were proposed to counter this phenomenon. The first was to introduce sparsity into the embedding's evaluation. DCCME-TopNMaxSparse achieves this however no performance increase was experienced over DCCME-Softmax. Inspection of the compression set sizes demonstrates that DCCME-TopNMaxSparse suffers from the same relentless increase of $|\mathcal{C}|$. A sparsemax (Martins and Astudillo, 2016) activation function was also investigated but this did not provide the $\boldsymbol{\alpha}(\cdot)$ sparsity at the sizes of $\mathcal{C}$ encountered in the MDPs. This may change if very large MDPs are used and is left to future work. The second technique to mitigate inferior policy learning in the holding pattern MDP was to introduce a more aggressive pruning strategy of elements in $\mathcal{C}$.

6.3.2 ACCME

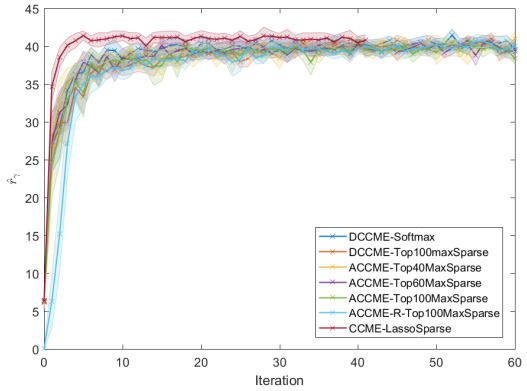
ACCME-TopNMaxSparse successfully applies a pruning strategy that is a novel and biologically-inspired method of maintaining $\mathcal{C}$. Evidence is provided in fig. 6.9 that supports the hypothesis that a more aggressive pruning strategy enables more effective training such that good policies can be learnt. Evidence of sparsification in the embedding's last layer is given in fig. 6.10. ACCME-R-TopNMaxSparse demonstrates that the average reward function can be learnt by exploiting the ψ feature layer, however the speed of policy learning and final performance are slightly diminished. This may be explained by the reward signal being deterministic and future work should include a focus on training ACCME-R-TopNMaxSparse with stochastic rewards.

6.4 Conclusion

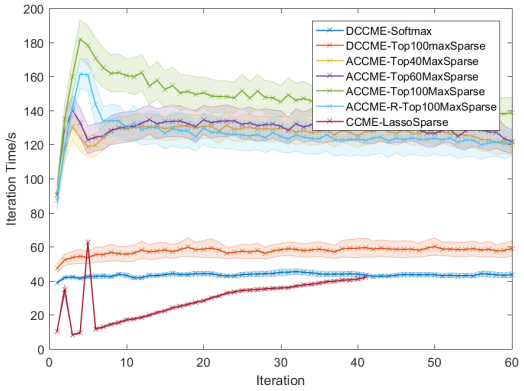
Although a deep embedding no longer enjoys strong theoretical guarantees, it is a practical solution in scaling the CME architecture when data is abundant. The novel fusion of a non-parametric loss that drives backprop has been shown to work. In addition, a novel adaptation of the truncated gradient algorithm enables ACCME to learn good policies in the largest MDP featured in this investigation.

Figure 6.6 Mountain car: DCCME activation functions

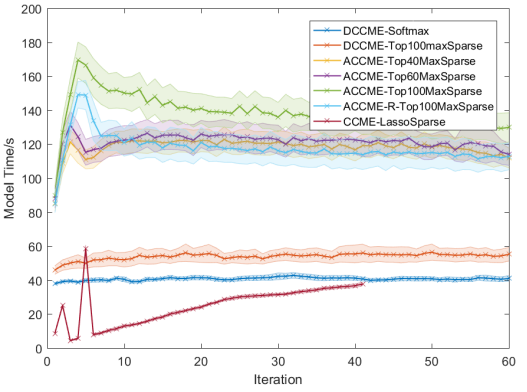
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model construction time



(d) Planning time

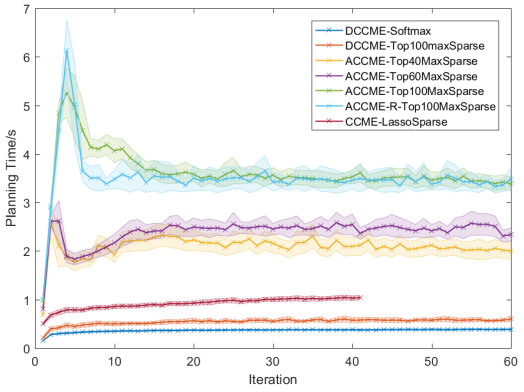
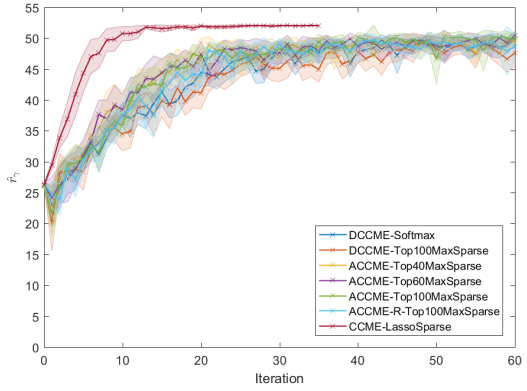
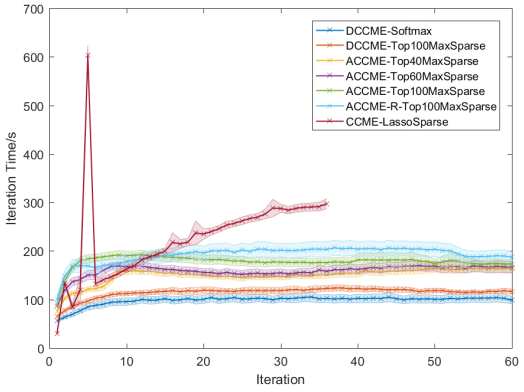


Figure 6.7 Cart pole: DCCME activation functions

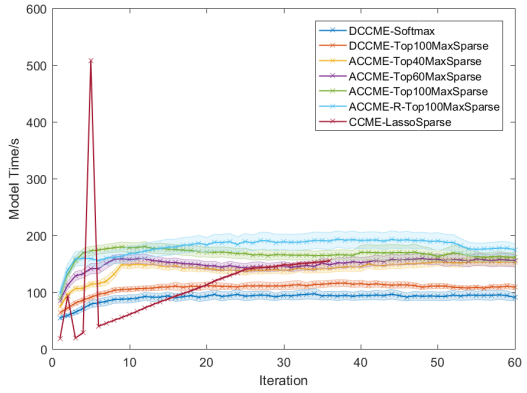
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model construction time



(d) Planning time

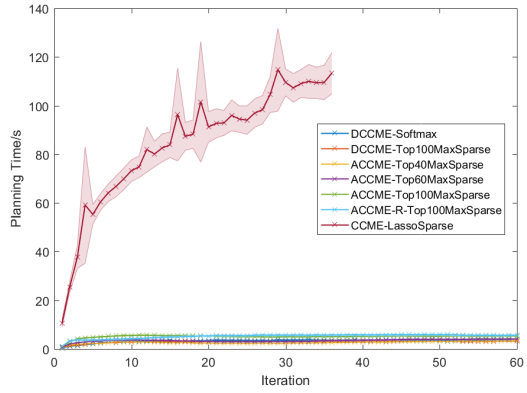
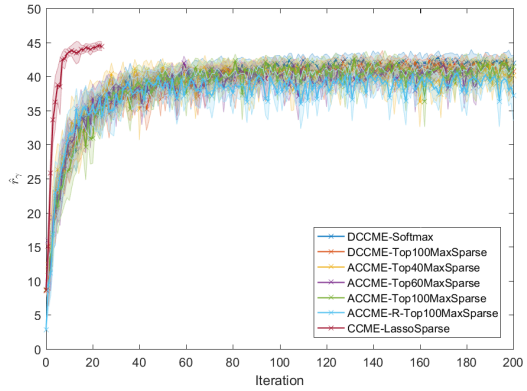
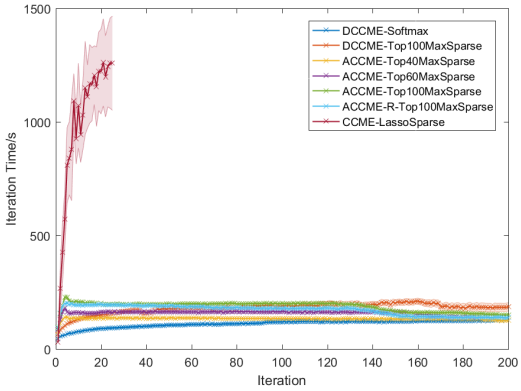


Figure 6.8 Quadrocopter navigation: DCCME activation functions

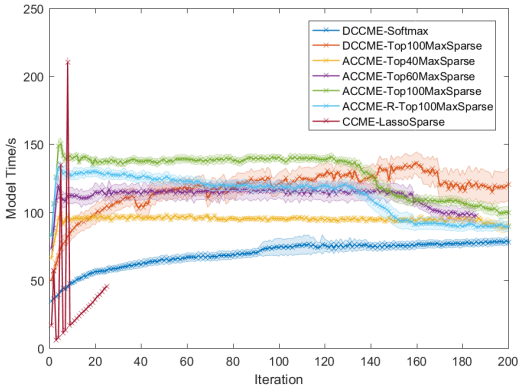
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model construction time



(d) Planning time

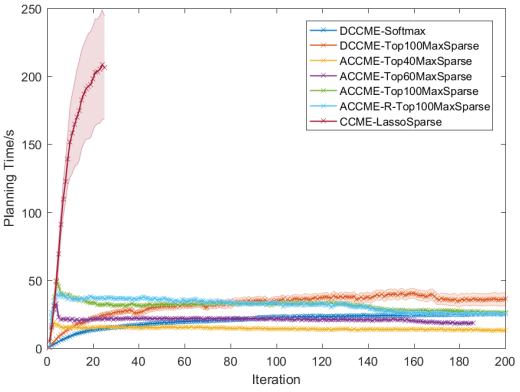
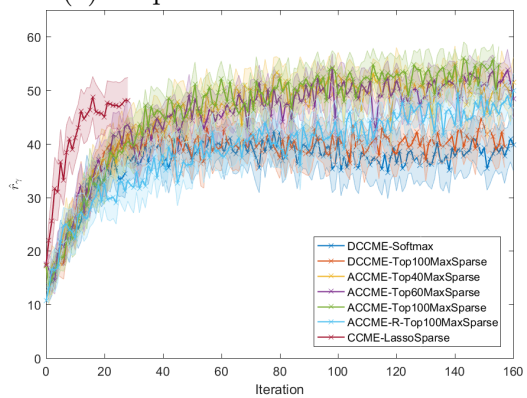
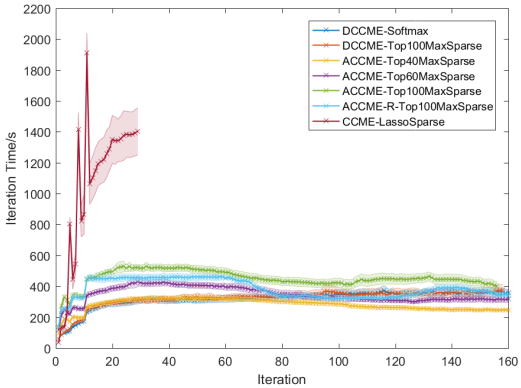


Figure 6.9 Quadrocopter holding pattern: DCCME activation functions

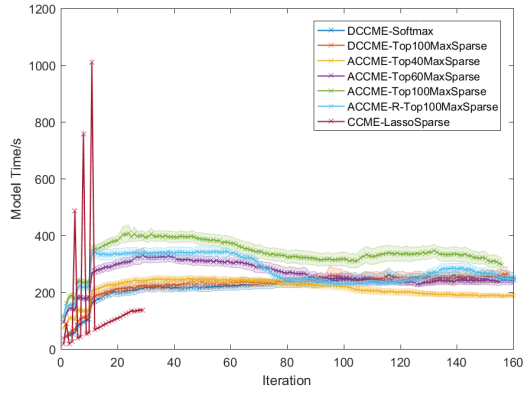
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model construction time



(d) Planning time

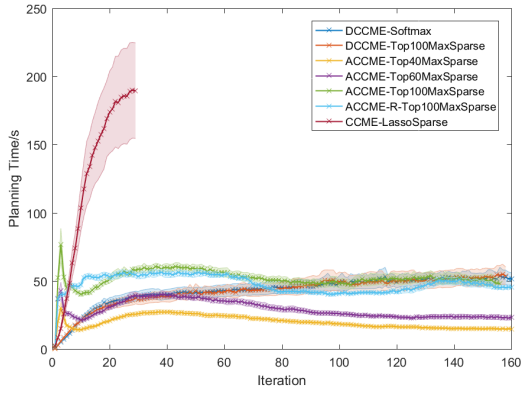
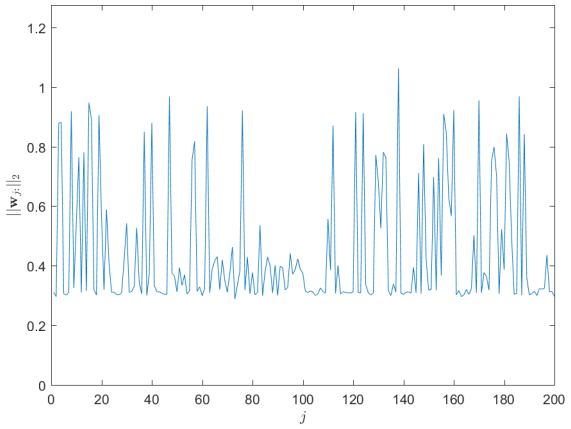
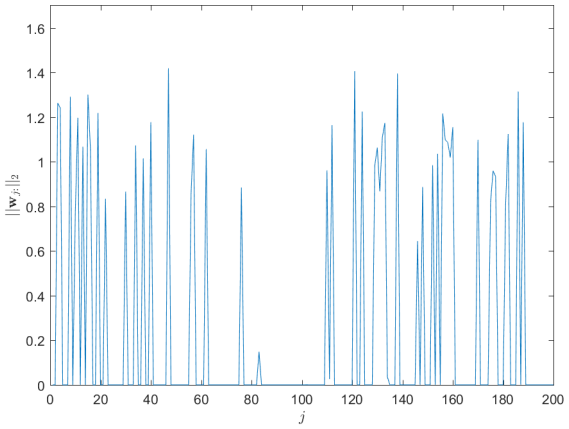


Figure 6.10 ACCME-Top40Max (mountain car) last layer weight group sparsity during model training: for policy iterations $k = \{1, 2, 3\}$ (top to bottom), every plot contains $||\mathbf{w}_{j:}||_2$ for each $\mathbf{w}_j \in \mathbf{W}$. The plots on the left are at the end of the initial phase (fig. 6.3) and those on the right are at the end of the weight shrinkage phase (fig. 6.4a) just before pruning.

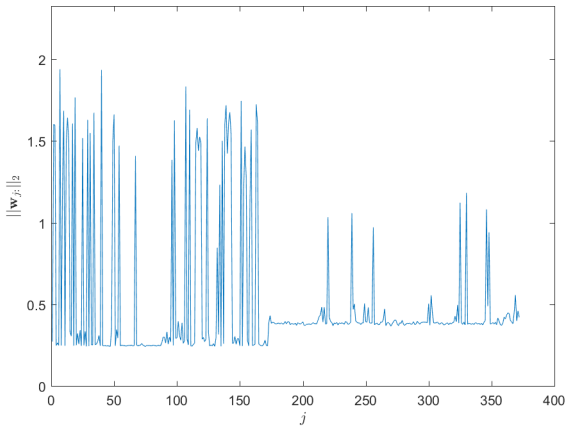
(a) At end of initial phase during $k = 1$



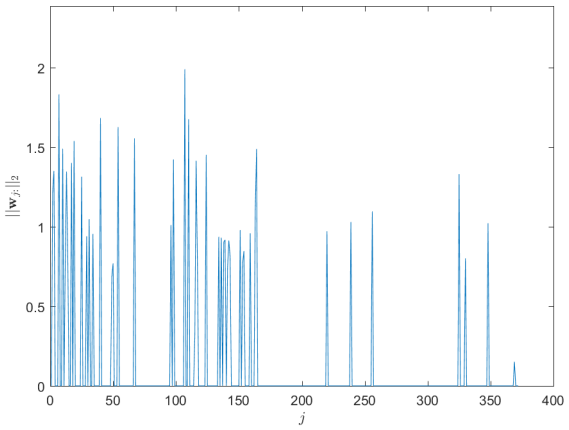
(b) At end of shrinkage phase during $k = 1$



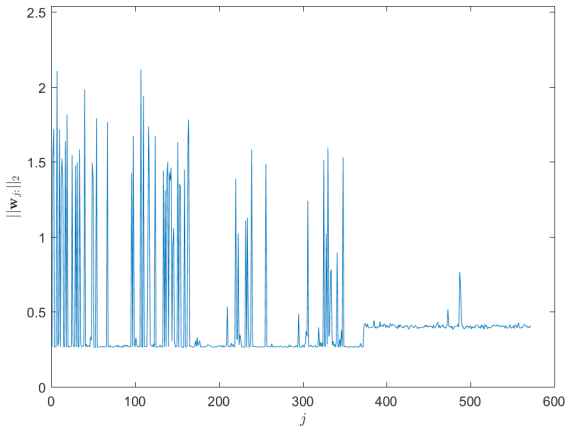
(c) At end of initial phase during $k = 2$



(d) At end of shrikage phase during $k = 2$



(e) At end of initial phase during $k = 3$



(f) At end of shrinkage phase during $k = 3$

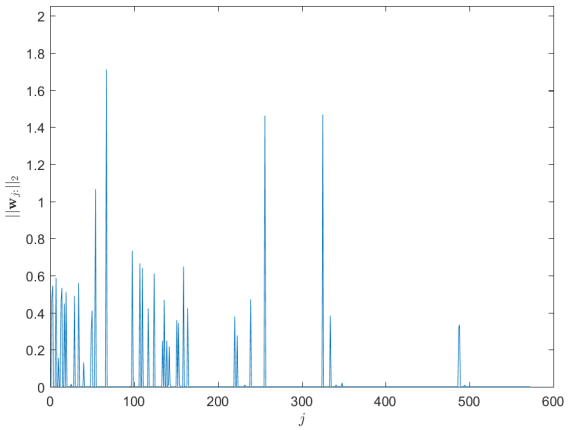
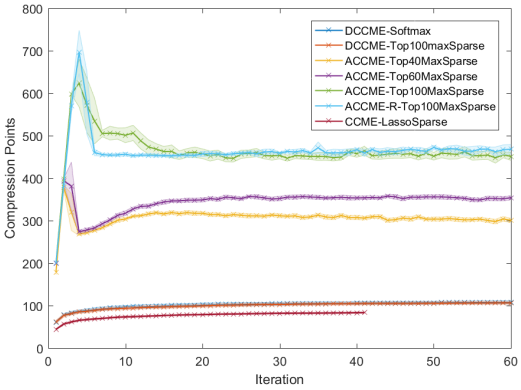
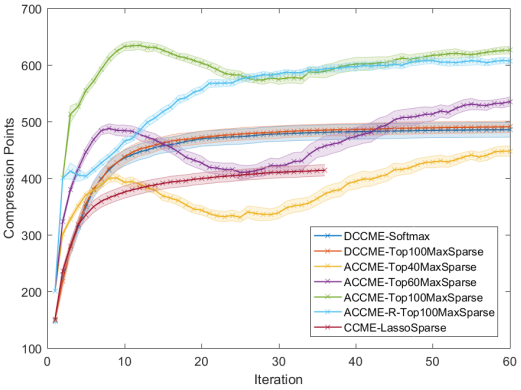


Figure 6.11 DCCME: compression set size $|\mathcal{C}|$

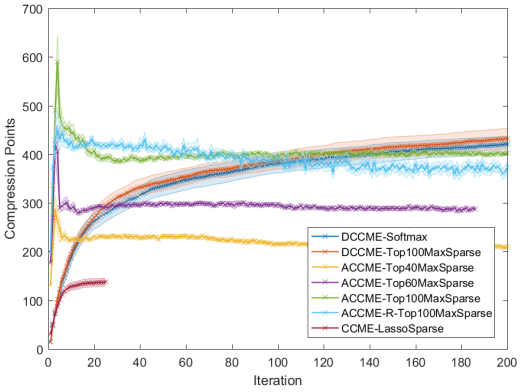
(a) Mountain car



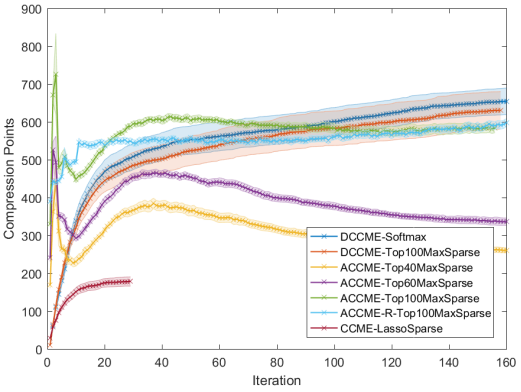
(b) Cart-pole



(c) Quadrocopter navigation



(d) Quadrocopter holding pattern



Chapter 7

Conclusion

7.1 Algorithm Comparison

7.1.1 Overview

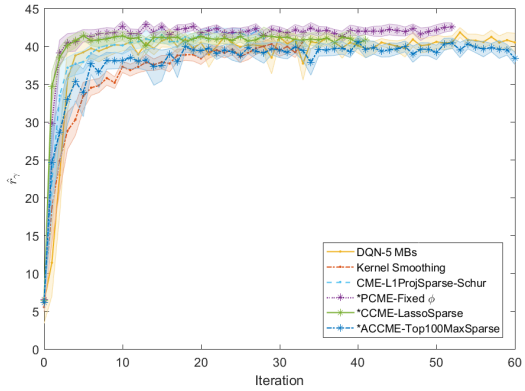
A comparison of the PCME, CCME and ACCME algorithms developed in this thesis are compared to the existing kernel smoothing and modified CME-Schur. In addition a DQN algorithm was deployed to demonstrate the sample efficiency of MBRL compared to an existing model-free algorithm. Policy learning performance and sample efficiency depend not only on the RL method but also function approximator class i.e. kernel methods are very efficient when the training dataset is small. From this perspective it is more fair to compare the DQN algorithm performance to ACCME. Further experimental details for the DQN can be found in section A.2. Computational complexities for algorithm components are specified in table 7.2.

Table 7.1 Computational complexity of algorithms developed in this thesis at the k^{th} policy iteration, which should be compared to table 2.2. †Although ACCME has convergence guarantees to the optimal policy defined by its pseudo-MDP, its neural network embedding does not enjoy the consistency guarantees that a kernel-based embedding has i.e. kernel-based embeddings will tend to the real MDP in the limit of infinite data. ‡PCME batch model construction. N is ACCME's last layer sparse activation count. See table 7.2 for more terminology.

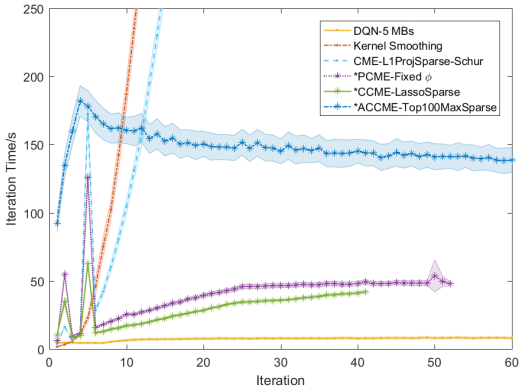
Algorithm	Guarantees	Planning		Model $\hat{\mathcal{E}}^{\mu}_{(\mathbf{s},\mathbf{a})}$	
		π_k evaln	π_k impvt	build	eval
CME-Schur	✓	$n_k^2 + J_{\text{eval}} N^*_{\text{L1-Proj}} n_k$	$\lvert\mathcal{A}\rvert\, n_k^2$	n_k^2	n_k
PCME	✗	$\lvert\mathcal{C}_k\rvert^3$	$\lvert\mathcal{A}\rvert\, \lvert\mathcal{C}_k\rvert\, \lvert\mathcal{B}_k\rvert$	‡$\lvert\mathcal{B}_k\rvert^3$	$\lvert\mathcal{B}_k\rvert$
CCME	✓	$\lvert\mathcal{C}_k\rvert^2 + J_{\text{eval}} N^*_{\text{Lasso}} \lvert\mathcal{C}_k\rvert$	$\lvert\mathcal{A}\rvert(\lvert\mathcal{B}_k\rvert\, \lvert\mathcal{C}_k\rvert + l)$	$\lvert\mathcal{B}_k\rvert^3 + \lvert\mathcal{C}_k\rvert^2$	$\lvert\mathcal{B}_k\rvert + l$
DCCME	†✗	$\lvert\mathcal{C}_k\rvert^2 + J_{\text{eval}} \lvert\mathcal{C}_k\rvert^2$	$\lvert\mathcal{A}\rvert\lvert\mathcal{C}_k\rvert^2$	$\lvert\mathcal{C}_k\rvert^2$	$\lvert\mathcal{C}_k\rvert^2$
ACCME	†✗	$\lvert\mathcal{C}_k\rvert^2 + J_{\text{eval}} N \lvert\mathcal{C}_k\rvert$	$\lvert\mathcal{A}\rvert N \lvert\mathcal{C}_k\rvert$	$N\lvert\mathcal{C}_k\rvert$	$N\lvert\mathcal{C}_k\rvert$

Figure 7.1 Mountain car: Algorithm comparison

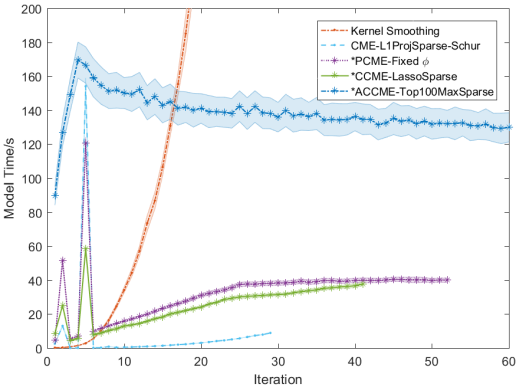
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model (incl. $\mathcal{B}$) construction time



(d) Planning time

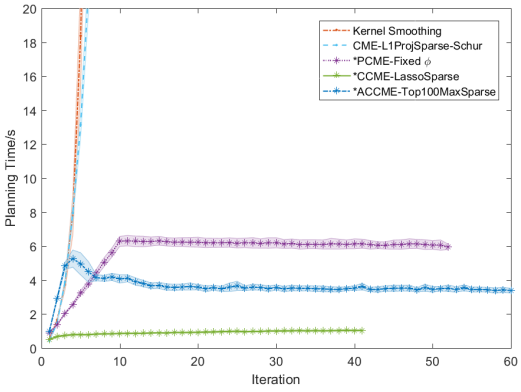
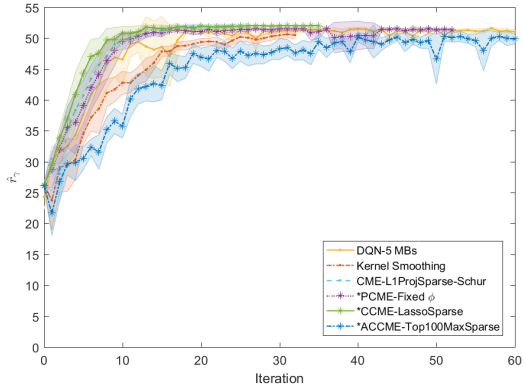
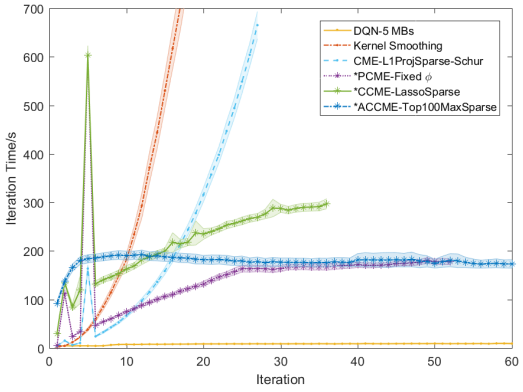


Figure 7.2 Cart-pole: Algorithm comparison

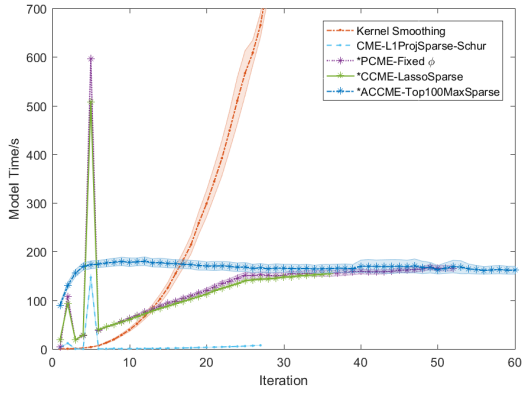
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model (incl. $\mathcal{B}$) construction time



(d) Planning time

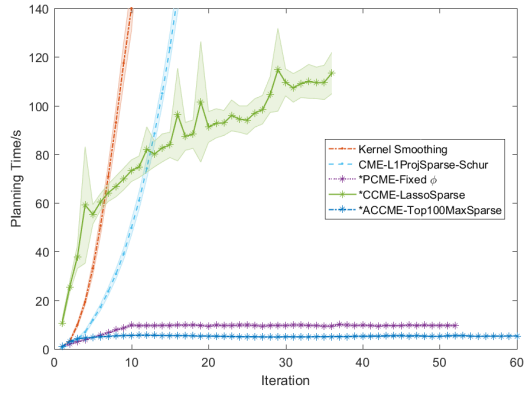
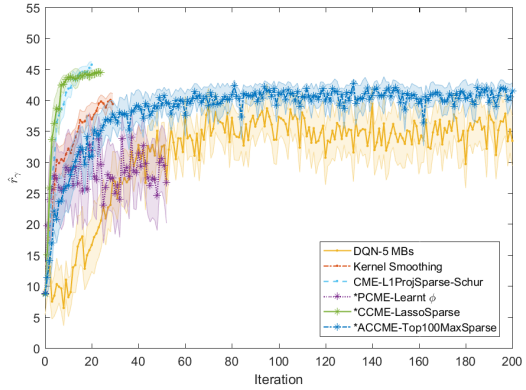
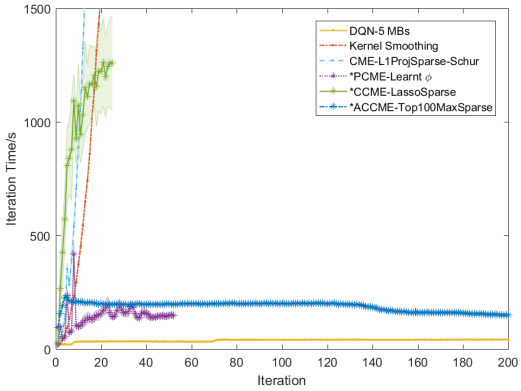


Figure 7.3 Quadrocopter navigation: Algorithm comparison

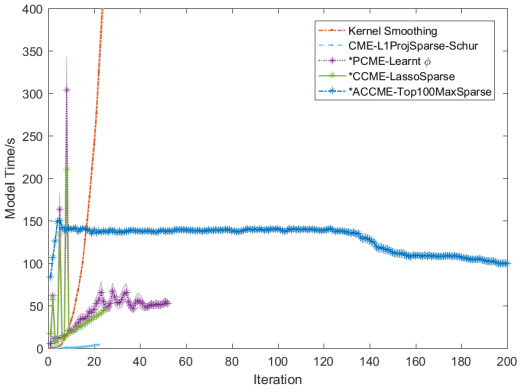
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model (incl. $\mathcal{B}$) construction time



(d) Planning time

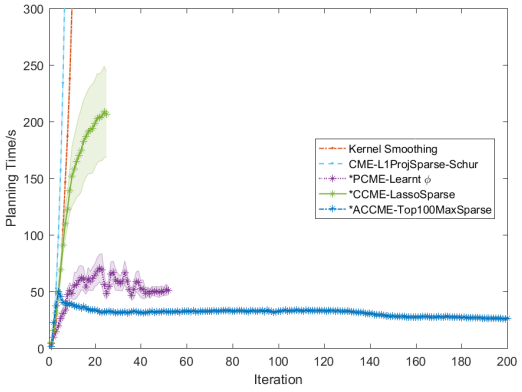
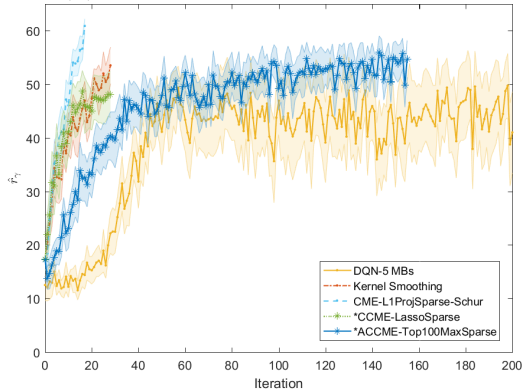
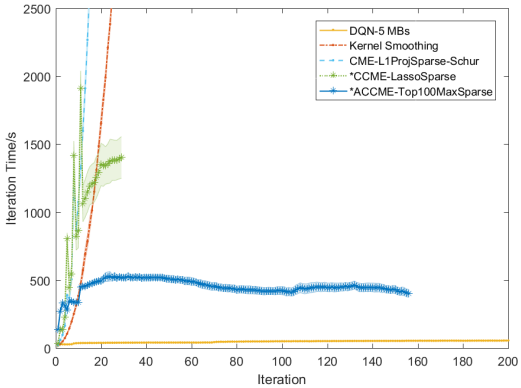


Figure 7.4 Quadrocopter holding pattern: Algorithm comparison

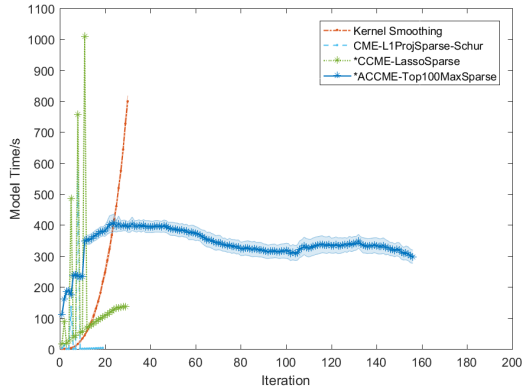
(a) Empirical discounted return



(b) Iteration (incl. data acquisition) time



(c) Model (incl. $\mathcal{B}$) construction time



(d) Planning time

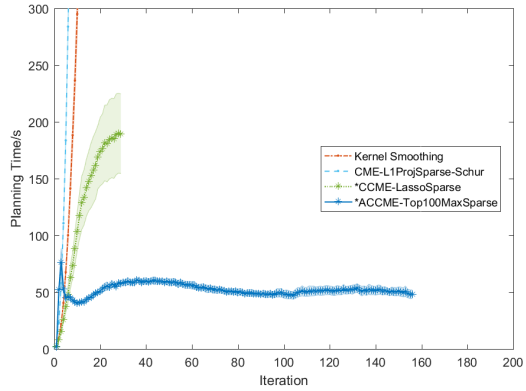


Table 7.2 Computational complexity of algorithm components at the k^{th} policy iteration. CCME Lasso components have complexity $l{=}m_{\text{chol}}^2 + f_{\text{lasso}}(|\mathcal{C}_k|, m_{\text{chol}})$ e.g $f_{\text{lasso}}{=}m_{\text{chol}}|\mathcal{C}_k|^2$ (Friedman et al., 2010) or $f_{\text{lasso}}{=}m_{\text{chol}}|\mathcal{C}_k|$ (Efron et al., 2004). N^*_{lasso} and $N^*_{\text{L1-Proj}}$ are the size of the sparsified $\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})$ which varies for each embedding evaluation.

Algorithm:		**Classic PI**	**CME-Schur**	**PCME**	**CCME**	**ACCME**
Constraint:			L1ProjSparse		LassoSparse	TopNmax
i)	$\mathcal{C}$ update	–	–	$n_k\|\mathcal{C}_k\|$	l	$\|\mathcal{C}_k\|$
ii)	model & $\mathcal{B}$ update	–	n_k^2	$\|\mathcal{B}_k\|^3$	$\|\mathcal{B}_k\|^3{+}\|\mathcal{C}_k\|^2$	$N\|\mathcal{C}_k\|$
iii)	$\|\|\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})\|\|_1{\leq}1$	–	n_k	–	l	$N\|\mathcal{C}_k\|$
iv)	planning	$J_{\text{eval}}\|\mathcal{S}\|^2$	$n_k^2{+}J_{\text{eval}}N^*_{\text{L1-Proj}}n_k$	$\|\mathcal{C}_k\|^3$	$\|\mathcal{C}_k\|^2{+}J_{\text{eval}}N^*_{\text{Lasso}}\|\mathcal{C}_k\|$	$\|\mathcal{C}_k\|^2{+}J_{\text{eval}}N\|\mathcal{C}_k\|$
v)	$\pi_k(\mathbf{s})$	$\|\mathcal{A}\|\|\mathcal{S}\|$	$\|\mathcal{A}\|n_k$	$\|\mathcal{A}\|\,\|\mathcal{B}_k\|$	$\|\mathcal{A}\|(\|\mathcal{B}_k\| + l)$	$\|\mathcal{A}\|N\|\mathcal{C}_k\|$

The original CME decoupled policy iteration from the number of states to the size of the training set. CCME continues this trend by decoupling from a polynomial complexity on the size of the data set to a linear dependency, instead the size of the compression set $|\mathcal{C}_k|$ dominates quadratically (mainly in the lasso implementation for the contraction constraint). ACCME not only breaks the training set dependency by making stochastic updates, but also maintaining the contraction constraint is linear in $|\mathcal{C}_k|$. Both CCME and ACCME incur a one-off $|\mathcal{C}_k|^2$ computational complexity when initiating planning, however the associated expense of this is small and is not a dominant cost in either algorithm. Although ACCME lacks consistency guarantees, its neural-kernel architecture hybrid balances expressive representation, guaranteed policy evaluation convergence and practical computational efficiency. All MBRL algorithms presented here retain sample efficiency when compared to a model-free DQN with experience replay.

7.2 Future Work

7.2.1 Immediate Extensions

Deep Parametric Embeddings (DPCME) The deep parametric CME algorithm with conservative policy updates is described in section 4.5.1. DPCME is hypothesised to integrate better with conservative policy updates because it avoids having a state representation whose basis is constantly changing (such as the matching pursuit-driven feature learning in the PCME algorithm).

More Experiments (ACCME) An immediate extension to ACCME would be to apply the algorithm to the low state dimensional control problems as specified in Todorov et al. (2012).

Sample Efficiency Comparison (ACCME) It would be desirable to not only compare ACCME's performance with all model-free DQN variants (Hessel et al., 2018) but also DYNA architectures (Gu et al., 2016). Additional comparisons that compare learning stability with those found in the deep learning community (Ha and Schmidhuber, 2018; Henaff et al., 2017), including the recent work found in Kaiser et al. (2019).

Model-Based Transfer Learning This investigation sought to develop principled MBRL that is practical in online environments. Apart from general sample efficiency however, this thesis hasn't explored the advantages of maintaining policy-agnostic transition models. The obvious advantage is being able to re-plan for multiple tasks in the same environment i.e. by simply executing planning with a different reward function. Does ACCME (or DPCME) transition models accelerate learning for new tasks? An extension to the successor features for transfer learning may also be feasible using the CCME or ACCME.

General Neural Network Pruning with Modified Group Shrinkage It may be possible to apply the modified group shrinkage algorithm to sparsify parameters deep inside feed forward function approximators with sparse activation functions. ReLU activation functions encourage sparsity and therefore modified group shrinkage may be able to work in conjunction with them. Changing the definition of the groups to be shrunk should also be investigated as well the out-of-sample error wrt. sparsification parameters in order to explore generalisation. Given also that network architecture search is often difficult, could modified group shrinkage with the dynamic adding of new nodes in hidden layers lead to more robust function approximators?

7.2.2 Long Term Extensions

High Dimensional States The ultimate goal would be to see if it is possible to extend ACCME to deal with higher dimensional states. This is a hard problem as it may require learning the state kernel bandwidth hyperparameters, perhaps by a separate value function fitting procedure. The current CCME and CME implementation impose relative weighting a priori to the diagonal of the bandwidth's covariance matrix in the state kernel. It would be desirable to learn these parameters as the agent is exploring its environment. With full hardware acceleration, would it be possible to implement an ACCME variant to solve MDPs as defined by the ALE (Bellemare et al., 2015)?

Options and Hierarchical Learning Now that ACCME provides a practical way of implementing a CME-based architecture, extending to learning SMDPs (Sutton

et al., 1998, 1999b) whose action sets with temporally abstract options (Precup et al., 1998). Recent work for extending model-free approximate value iteration can be found in Mann et al. (2015) and Yao et al. (2014b) with linear function approximation.

Bibliography

Ade, P. A. R., Aghanim, N., Arnaud, M., Ashdown, M., Aumont, J., Baccigalupi, C., Banday, A., Barreiro, R., Bartlett, J., and Al., E. (2016). Planck 2015 results. XIII. Cosmological parameters. *AAP*, 594:A13.

Antos, A., Szepesvári, C., and Munos, R. (2008). Learning Near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path. *Machine Learning*, 71(1):89–129,.

Aronszajn, N. (1950). Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

Atkeson, C. and Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 4:3557–3564.

Baird, L. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 30–37. Morgan Kaufmann.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202.

Bellemare, M., Naddaf, Y., Veness, J., and Bowling, M. (2015). The arcade learning environment: An evaluation platform for general agents. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4148–4152.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, 1 edition.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bertsekas, D. (2012). *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 3rd edition.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Bertsekas, D. and Yu, H. (2010). Distributed asynchronous policy iteration in dynamic programming. *2010 48th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2010*, pages 1368–1375.

Bhatnagar, S., Precup, D., Silver, D., Sutton, R., Maei, H., and Szepesvári, C. (2009). Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 1204–1212. Curran Associates, Inc.

Bishop, C. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bostrom, N. (2014). Superintelligence: Paths, dangers, strategies.

Bottou, L. (1998). On-line Learning and Stochastic Approximations. In *On-line Learning in Neural Networks*, pages 9–42. Cambridge University Press.

Bottou, L. (2012). Stochastic Gradient Descent Tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer Berlin Heidelberg.

Boyan, J. (1998). Least-Squares Temporal Difference Learning. Technical report, CMU.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122.

Boyd, S. and Vandenberghe, L. (2009). *Convex Optimization*. Cambridge University Press, Cambridge, 7th edition.

Bradtke, S. and Barto, A. (1996). Linear Least-Squares Algorithms for Temporal Difference Learning. *Machine Learning*, 22:33–57.

Choromanska, A., Henaff, M., and Mathieu, M. (2015). The Loss Surfaces of Multilayer Networks. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 38, San Diego, CA, USA.

Christmann, A. and Steinwart, I. (2007). Consistency and robustness of kernel-based regression in convex risk minimization. *Bernoulli*, 13(3):799–819.

Dann, C., Neumann, G., and Peters, J. (2014). Policy Evaluation with Temporal Differences: A Survey and Comparison. *Journal of Machine Learning Research*, 15(1):809–883.

Dayan, P. and Berridge, K. (2014). Model-based and model-free Pavlovian reward learning: Revaluation, revision, and revelation. *Cognitive, Affective, & Behavioral Neuroscience*, 14(2):473–492.

Dayan, P. and Niv, Y. (2008). Reinforcement learning: The Good, The Bad and The Ugly. *Current Opinion in Neurobiology*, 18(2):185–196.

Deisenroth, M. and Rasmussen, C. E. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, WA, USA.

Deisenroth, M., Rasmussen, C. E., and Peters, J. (2009). Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7-9):1508–1524.

Denardi, R. (2012). QRSim.

Doll, B., Simon, D., and Daw, N. (2012). The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology*, 22(6):1075–1081.

Dreyfus, S. (1973). The Computational Solution of Optimal Control Problems with Time Lag. *IEEE Transactions on Automatic Control*, 18(4):383–385.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the L1 -ball for learning in high dimensions. *Proceedings of the 25th International conference on Machine learning (ICML)*, pages 272–279.

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., Ishwaran, H., Knight, K., Loubes, J. M., Massart, P., Madigan, D., Ridgeway, G., Rosset, S., Zhu, J. I., Stine, R. A., Turlach, B. A., Weisberg, S., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32(2):407–499.

Engel, Y., Mannor, S., and Meir, R. (2004). The Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with Gaussian processes. In *In Proceedings of the 22nd International Conference on Machine Learning (ICML)*, Bonn, Germany.

Engel, Y., Mannor, S., and Melr RMEIR, R. (2003). Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, Washington DC.

Evgeniou, T., Pontil, M., and Poggio, T. A. (2000). Regularization Networks and Support Vector Machines. *Advanced in Computational Mathematics*, page 53.

Farahmand, A. M., Ghavamzadeh, M., Mannor, S., and Szepesvári, C. (2008). Regularized Policy Iteration. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21 (NIPS)*, pages 441–448. Curran Associates, Inc.

Ferns, N., Panangaden, P., and Precup, D. (2012). Metrics for markov decision processes with infinite state spaces. *CoRR*, abs/1207.1386.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal Of Statistical Software*, 33(1).

Geist, M., Piot, B., and Pietquin, O. (2017). Is the Bellman residual a bad proxy? In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31 (NIPS*, pages 1–13. Curran Associates, Inc., Long Beach, United States.

Geramifard, A., Walsh, T., Tellex, S., Chowdhary, G., Roy, N., and How, J. (2013). A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning. *Foundations and Trends in Machine Learning*, 6(4):375–454.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Pmlr*, 9:249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, Fort Lauderdale, FL, USA.

Gordon, G. (1995). Stable Function Approximation in Dynamic Programming. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*.

Graybill, F. (1961). *An Introduction to Linear Statistical Models*, volume 1. McGraw Hill, New York, NY, USA.

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep AutoRegressive Networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Grimmett, G. and Stirzaker, D. (2001). *Probability and Random Processes*. Oxford University Press, Oxford; New York, 3rd edition.

Grünewälder, S., Lever, G., Baldassarre, L., Gretton, A., and Pontil, M. (2012a). Modelling transition dynamics in MDPs with RKHS embeddings. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 535–542, Edinburgh, Scotland, UK.

Grünewälder, S., Lever, G., Baldassarre, L., Patterson, S., Gretton, A., and Pontil, M. (2012b). Conditional Mean Embeddings as Regressors. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1823–1830, Edinburgh, Scotland, UK.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous Deep Q-Learning with Model-based Acceleration. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*.

Guestrin, C., Koller, D., and Parr, R. (2001). Max-norm Projections for Factored MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, number August, pages 673–680, Seattle, Washington.

Ha, D. and Schmidhuber, J. (2018). World Models. *CoRR*.

Haggarty, R. (1993). *Fundamentals of Mathematical Analysis*. Prentice-Hall, Inc., Harlow, England, 2nd edition.

Harutyunyan, A., Stepleton, T., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 1054–1062.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity The Lasso and Generalizations*. Chapman & Hall/CRC Press.

Henaff, M., Whitney, W. F., and LeCun, Y. (2017). Model-Based Planning in Discrete Action Spaces. *arXiv*.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Bilal, P., Azar, M., and Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. *Association for the Advancement of Artificial Intelligence*.

Hinton, G. E., Srivastava, N., and Swersky, K. (2012). Lecture 6a- overview of mini-batch gradient descent. *COURSERA: Neural Networks for Machine Learning*, page 31.

Hofmann, T., Schölkopf, B., and Smola, A. (2008). Kernel Methods in Machine Learning. *Annals of Statistics*, 36(3):1171–1220.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, USA.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32 nd International Conference on Machine Learning (ICML)*, Lille, France.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. (2019). Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374.

Kakade, S. and Langford, J. (2002). Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, ICML '02, pages 267–274, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kimeldorf, G. and Wahba, G. (1971). Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95.

Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. Technical report, University of Toronto.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Kolter, Z. and Ng, A. (2009). Regularization and feature selection in least-squares temporal difference learning. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 94305:1–8.

Kong, D., Fujimaki, R., Liu, J., Nie, F., and Ding, C. (2014). Exclusive Feature Learning on Arbitrary Structures via L12-norm. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 1655–1663.

Kreyszig, E. (1978). *Introductory Functional Analysis with Applications*. John Wiley & Sons, London.

Krizhevsky, A., Sutskever, I., Hinton, G. E. G. E., Sulskever, I., and Hinton, G. E. G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances In Neural Information Processing Systems 25 (NIPS)*, pages 1097–1105.

Kroemer, O. and Peters, J. (2011). A Non-Parametric Approach to Dynamic Programming. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24 (NIPS)*, pages 1719–1727, Granada, Spain. Curran Associates, Inc.

Lagoudakis, M. and Parr, R. (2003). Least-squares Policy Iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149.

Lang, S. (1987). *Linear Algebra*. Springer New York Inc., New York, NY, USA, 3rd edition.

Langford, J., Li, L., and Zhang, T. (2009). Sparse Online Learning via Truncated Gradient. *Journal of Machine Learning Research (JMLR)*, 10(1):777–801.

Lebedev, V. and Lempitsky, V. (2016). Fast ConvNets Using Group-wise Brain Damage. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lecun, Y. (1998). Efficient BackProp. *Neural Netwoks: tricks of the trade*, 53(9):1689–1699.

Lecun, Y. (2007). Who is afraid of non convex loss functions ? In *Neural Information Processing Systems 21 (NIPS) Workshop Lecture*, Whistler. New York University.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2323.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal Brain Damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann.

Lever, G., Shawe-Taylor, J., Stafford, R., and Szepesvári, C. (2016). Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1779–1787, Phoenix, Arizona.

Lever, G. and Stafford, R. (2015). Modelling Policies in MDPs in Reproducing Kernel Hilbert Space. In *AIStats*, pages 590–598.

Lever, G., Stafford, R., and Shawe-Taylor, J. (2014). Learning Transition Dynamics in MDPs with Online Regression and Greedy Feature Selection. In *Autonomously Learning Robots Workshop (ALR NIPS Workshop)*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, 17:1–40.

Li, P. (2013). nth-element: Array Sorting in Matlab.

Lin, D. (2007). slmetric-pw - Computing Pairwise Distances and Metrics.

Lin, L.-J. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3):293–321.

Maas, A., Hannun, A., and Ng, A. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 28:6.

Mallat, S. G. and Zhang, Z. (1993). Matching Pursuits with Time-frequency Dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415.

Mann, T., Mannor, S., and Precup, D. (2015). Approximate Value Iteration with Temporally Extended Actions. *Journal of Artificial Intelligence Research*, 53(May 2015):375–438.

Martins, A. F. T. and Astudillo, R. F. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proceedings of the 33 rd International Conference on Machine Learning (ICML)*. JMLR.

Mercer, J. (1909). XVI. Functions of positive and negative type, and their connection the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446.

Micchelli, C. and Pontil, M. (2005). On Learning Vector-Valued Functions. *Neural computation*, 17(1):177–204.

Minh, H. Q., Sindhwani, V., Sciences, M., and Heights, Y. (2011). Vector-valued Manifold Regularization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, number 1, pages 57–64, Bellevue, WA, USA.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. a., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Monahan, G. E. (1982). A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16.

Munos, R. (2003). Error Bounds for Approximate Policy Iteration. *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2:560–567.

Munos, R. (2005). Error Bounds for Approximate Value Iteration. *American Association for Artificial Intelligence (AAAI)*, 2:1006–1011.

Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, MA, USA.

Nadaraya, È. (1963). On Estimating Regression. *Theory of Probability & Its Applications*, 9(1).

Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814.

Nedic, A. and Bertsekas, D. (2003). Least Squares Policy Evaluation Algorithms with Linear Function Approximation. *Discrete Event Dynamic Systems*, 13:79–110.

Nesterov, Y. (1983). A Method of Solving A Convex Programming Problem With Convergence rate O(1/kˆ2).

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ormoneit, D. and Sen, Ś. (2002). Kernel-Based Reinforcement Learning. *Machine learning*, 49(2-3):161–178.

Painter-Wakefield, C. and Parr, R. (2012). Greedy Algorithms for Sparse Reinforcement Learning. In *In Proceedings of the 29th International Conference on Machine Learning (ICML)*, Edinburgh, Scotland.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. (2008). An Analysis of Linear Models, Linear Value-function Approximation, and Feature Selection for Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, ICML '08, pages 752–759, New York, NY, USA. ACM.

Pavlov, I. P. (1927). Lectures on conditioned reflexes. *International Journal of Game Theory*, 4:25–55.

Peng, J. and Williams, R. J. (1996). Incremental Multi-Step Q-Learning. *Machine Learning*, 22:283–290.

Peters, J., Mülling, K., and Altün, Y. (2010). Relative Entropy Policy Search. In Fox, M. and Poole, D., editors, *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1607–1612. AAAI Press.

Peters, J. and Schaal, S. (2007). Natural Actor-Critic.

Peters, J. and Schaal, S. (2008). Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 21(4):682–697.

Petersen, K. B. and Pedersen, M. S. (2012). The Matrix Cookbook.

Precup, D., Sutton, R., and Singh, S. (1998). Theoretical Results on Reinforcement Learning with Temporally Abstract Options. In *Proceedings of the 10th European conference on Machine Learning (ECML)*, pages 382–393. Springer Verlag.

Puterman, M. and Shin, M. C. (1978). Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 24(11):1127–1137.

Qian, J., Hastie, T., Friedman, J., Tibshirani, R., and Simon, N. (2013). Glmnet for Matlab.

Rasmussen, C. E. and Ghahramani, Z. (2002). Bayesian Monte Carlo. In *Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS)*, NIPS'02, pages 505–512, Cambridge, MA, USA. MIT Press.

Rasmussen, C. E., Williams, C. K. I., Sutton, R., Barto, A., Spirtes, P., Glymour, C., Scheines, R., Schölkopf, B., and Smola, A. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Robards, M., Sunehag, P., Sanner, S., and Marthi, B. (2011). Sparse Kernel-SARSA (Lambda) with an Eligibility Trace. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML PKDD'11, pages 1–17, Berlin, Heidelberg. Springer-Verlag.

Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *Ann. Math. Statist.*, 22(3):400–407.

Rosenblatt, M. (1956). On the Estimation of Regression Coefficients of a Vector-Valued Time Series with a Stationary Residual. *Annals of Mathematical Statistics*, 27(1):99–121.

Ross, S. and Bagnell, A. (2012). Agnostic System Identification for Model-Based Reinforcement Learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, UK. icml.cc / Omnipress.

Rudin, W. (1976). *Principles of mathematical analysis*. McGraw-Hill Book Co., New York, third edition. International Series in Pure and Applied Mathematics.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, pages 533–536.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical report, Cambridge University, Cambridge, UK.

Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. In *Proceedings of the 27th International Confer- ence on Machine Learning (ICML)*, pages 959–966, Haifa, Israel.

Scherrer, B. (2014). Approximate Policy Iteration Schemes: A Comparison. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, Beijing, China.

Schmidhuber, J. (2015). Deep Learning in neural networks: An overview.

Schmidt, M. (2005). minFunc: unconstrained differentiable multivariate optimization in Matlab.

Schoknecht, R. (2003). Optimality of Reinforcement Learning Algorithms with Linear Function Approximation. *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 1555–1562.

Schölkopf, B., Herbrich, R., and Smola, A. (2001). A Generalized Representer Theorem. In Helmbold, D. and Williamson, B., editors, *Proceedings of the 14th Annual Conference on Computational Learning Theory (COLT) and and 5th European Conference on Computational Learning Theory (EuroCOLT)*, pages 416–426, Amsterdam. Springer-Verlag, Berlin Heidelberg.

Schölkopf, B. and Smola, A. (2002). *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(June 1994):1593–1599.

Seijen, H. and Sutton, R. (2014). True Online TD(lambda). In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 692–700, Bejing, China. PMLR.

Sejdinovic, D. and Gretton, A. (2012). Lecture Notes: What is an RKHS?

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, 1 edition.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *International Conference on Learning Representations (ICLR)*.

Sherstov, A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In Zucker, J. and Saitta, I., editors, *SARA*, pages 194–205. Springer Verlag.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. *Computer Vision and Patter Recognition (CVPR)*, pages 1874–1883.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Sutton, R., and Müller, M. (2008). Sample-Based Learning and Search with Permanent and Transient Memories. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, Helsinki, Finland.

Singh, S. and Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158.

Singh, S. and Yee, R. (1994). An Upper Bound on the Loss from Approximate Optimal-Value Functions. *Machine Learning*, 16(3):227–233.

Song, L., Fukumizu, K., and Gretton, A. (2013). Kernel Embeddings of Conditional Distributions: A Unified Kernel Framework for Nonparametric Inference in Graphical Models.

Song, L., Gretton, A., and Guestrin, C. (2010). Nonparametric Tree Graphical Models via Kernel Embeddings. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Chia Laguna Resort, Sardinia, Italy.

Song, L., Huang, J., Smola, A., and Fukumizu, K. (2009). Hilbert Space Embeddings of Conditional Distributions with Applications to Dynamical Systems. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 961–968.

Stafford, R. and Shawe-Taylor, J. (2018). ACCME : Actively Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning. In *European Workshop on Reinforcement Learning (EWRL)*, Lille, France.

Steinwart, I. and Christmann, A. (2008). *Support Vector Machines*. Springer-Verlag New York, Inc., New York, NY, USA, 1 edition.

Sutton, R. (1988). Learning to Predict by the Methods of Temporal Differences. In *Machine Learning*, volume 3, pages 9–44.

Sutton, R. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the 7th International Conferencec on Machine Learning*, pages 216–224. Morgan Kaufmann.

Sutton, R. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2:160–163.

Sutton, R. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8 (NIPS)*, pages 1038–1044. MIT Press.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. A Bradford Book - MIT Press, Cambridge, MA, USA, 1st edition.

Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book - MIT Press, 2nd edition.

Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 1–8.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (1999a). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12 (NIPS)*, volume 99, pages 1057–1063.

Sutton, R., Precup, D., and Singh, S. (1998). Intra-Option Learning about Temporally Abstract Actions. In *Proceedings of the 15th International Conference on Machine Learning (ICML)*.

Sutton, R., Precup, D., and Singh, S. (1999b). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211.

Sutton, R., Szepesvári, C., Geramifard, A., and Bowling, M. (2008a). Dyna-Style Planning with Linear Function Approximation and Prioritized Sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 528–536.

Sutton, R., Szepesvári, C., and Maei, H. (2008b). A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1609–1616.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv:1312.6199*, pages 1–10.

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*, volume 4 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, San Rafael, CA, USA.

Taylor, G. and Parr, R. (2009). Kernelized Value Function Approximation for Reinforcement Learning. In *Proceedings ofthe 26th International Conference on Machine Learning (ICML)*, Montreal, Canada.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.

Tibshirani, R. (1996). Regression selection and shrinkage via the lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Tromp, J. and Farnebäck, G. (2006). Combinatorics of Go. In H. Jaap van den Herik and Paolo Ciancarini, editors, *Computers and Games*, pages 84–99, Turin. Springer.

Tsitsiklis, J. and Van Roy, B. (1996). Feature-Based Methods For Large Scale Dynamic Programming. *Machine Learning*, 22:59–94.

Tsitsiklis, J. and Van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5).

van Hoof, H., Peters, J., and Neumann, G. (2015). Learning of Non-Parametric Control Policies with High-Dimensional State Features. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Diego, CA, USA.

Van Roy, B. (1998). *Learning and Value Function Approximation in Complex Decision Processes*. Doctor of philosoph y, MIT.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10(5):988–99.

Vincent, P. and Bengio, Y. (2002). Kernel matching pursuit. In *Machine Learning*, pages 165–187. Kluwer Academic Publishers.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*.

Wagner, P. (2011). A Reinterpretation of the Policy Oscillation Phenomenon in Approximate Policy Iteration. In *Advances in Neural Information Processing Systems 24 (NIPS)*, pages 2573–2581.

Wagner, P. (2013). Optimistic Policy Iteration and Natural Actor-Critic: A Unifying View and a Non-Optimality Result. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 1592–1600.

Wagner, P. (2014). Policy Oscillation is Overshooting. *Neural Networks*, 52:43–61.

Wahlström, N., Schön, T., and Deisenroth, M. (2015). From Pixels to Torques: Policy Learning with Deep Dynamical Models. *ArXiv*.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Watkins, C. and Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, 8(3-4):279–292.

Watson, G. (1964). Smooth Regression Analysis. *Sankhya*, 26(4):359–372.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. (2015). Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 2746–2754.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Silver, D., and Wierstra, D. (2017). Imagination-Augmented Agents for Deep Reinforcement Learning. *CoRR*.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning Structured Sparsity in Deep Neural Networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*.

Werbos, P. (1982). Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization in 10th IFIP Conference*, pages 762–770, New York, NY, USA. Springer.

Williams, R. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256.

Williams, R. and Baird, L. (1993). Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions. *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 108–113.

Xu, X., Xie, T., Hu, D., and Lu, X. (2005). Kernel Least-Squares Temporal Difference Learning. *International Journal of Information Technology*, 11(9).

Yao, H. (2011). Off-policy Learning with Linear Action Models: An Efficient One-Collection-For-All Solution. In *Planning and Acting with Uncertain Models Workshop at the 28th ICML*, Bellevue, WA, USA. Citeseer.

Yao, H. and Szepesvári, C. (2012). Approximate Policy Iteration with Linear Action Models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1212–1217.

Yao, H., Szepesvári, C., Pires, B. A., and Zhang, X. (2014a). Pseudo-MDPs and Factored Linear Action Models. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–9.

Yao, H., Szepesvári, C., Sutton, R., Modayil, J., and Bhatnagar, S. (2014b). Universal Option Models. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 990–998.

Yoon, J. and Hwang, S. J. (2017). Combined Group and Exclusive Sparsity for Deep Neural Networks. In *Proceedings of the 34 th International Conference on Machine Learning (ICML)*, Sydney, Australia.

Yu, Y.-l., Cheng, H., Schuurmans, D., and Szepesvári, C. (2013). Characterizing the Representer Theorem. *International Conference on Machine Learning (ICML)*, pages 1–9.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 68(1):49–67.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations (ICLR)*, Toulon. France.

Zhang, F. (2005). *The Schur Complement and its Applications*. Springer, 4th edition.

Appendix A

Supplemental

A.1 Matching Pursuit Variants

A.1.1 Notation

By explicitly reordering the summation then the embedding $\mu = \mathbf{\Phi}_{\mathcal{C}}^{\top}\mathbf{W}\mathbf{\Upsilon}_{\mathcal{B}}$ can be written as

$$
\begin{aligned}
{}^{(\bullet)}\mu^{d}(\cdot) &:= \sum_{\ell=1}^{d}\Big(\sum_{j=1}^{m} L(\mathbf{c}_j,\bullet)w_{j\ell}\Big)K(\mathbf{b}_\ell,\cdot) \in \mathcal{H}_{\mathbf{\Gamma}},\\
&= \sum_{\ell=1}^{d}\Big(\mathbf{\Phi}_{\mathcal{C}}^{\top}\mathbf{w}_\ell\Big)K(\mathbf{b}_\ell,\cdot), \quad \mathbf{w}_\ell \in \mathbb{R}^m, \qquad \text{(A.1)}\\
&= \sum_{\ell=1}^{d}\tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell,\cdot), \quad \tilde{\mathbf{w}}_\ell \in \mathcal{H}_L, \qquad \text{(A.2)}
\end{aligned}
$$

where $\mathbf{w}_\ell$ is the ℓ^{th} column of the embedding's weight matrix $\mathbf{W}$. The embedding summation can also be reordered to define

$$
\begin{aligned}
{}^{(\bullet)}\mu^{m}(\cdot) &:= \sum_{j=1}^{m} L(\mathbf{c}_j,\bullet)\Big(\sum_{\ell=1}^{d} w_{j\ell}K(\mathbf{b}_\ell,\cdot)\Big) \in \mathcal{H}_{\mathbf{\Gamma}},\\
&= \sum_{j=1}^{m} L(\mathbf{c}_j,\bullet)\mathbf{w}_{j:}\mathbf{\Upsilon}_{\mathcal{B}}, \quad \mathbf{w}_{j:}^{\top} \in \mathbb{R}^d, \qquad \text{(A.3)}\\
&= \sum_{j=1}^{m} L(\mathbf{c}_j,\bullet)\hat{\mathbf{w}}_{j:}, \quad \hat{\mathbf{w}}_{j:}^{\top} \in \mathcal{H}_K, \qquad \text{(A.4)}
\end{aligned}
$$

where $\mathbf{w}_{j:}$ is the j^{th} row vector of the embedding's weight matrix $\mathbf{W}$.

A.1.2 Matching Pursuit for RKHS-Valued Regression

Lemma A.1 (RKHS Matching Pursuit Regression for learning $\mathcal{B}$)**.** *Given the embedding $\mu : \mathcal{S}\times\mathcal{A} \to \mathcal{H}_L$ with basis $\mathcal{B}$ which takes the form in equation (A.2),*

$$ {}^{(\bullet)}\mu^d(\mathbf{z}) := \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \mathbf{z}), \quad \tilde{\mathbf{w}}_\ell \in \mathcal{H}_L, \quad \mathbf{z} \in \mathcal{S}\times\mathcal{A}, $$

then the model residue for the current basis of size $d = |\mathcal{B}|$ is

$$ {}^{(\bullet)}\mathbf{r}_i^d := L(\mathbf{s}'_i, \bullet) - {}^{(\bullet)}\mu^d(\mathbf{z}_i), \qquad \mathbf{s}'_i \in \mathcal{S},\ \mathbf{z}_i \in \mathcal{S}\times\mathcal{A}, \tag{A.5} $$

$$ = L(\mathbf{s}'_i, \bullet) - \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \mathbf{z}_i) \in \mathcal{H}_L. \tag{A.6} $$

Given a dictionary $\mathcal{G} := \{K(\hat{\mathbf{b}}_1, \cdot), K(\hat{\mathbf{b}}_2, \cdot), ..\}$ and dataset $\mathcal{D} := \{(\mathbf{z}, \mathbf{s}')_i\}_{i=1}^n$, then if the minimisation problem (4.5) for greedily enriching the state-action feature representation by one more basis function $K(\mathbf{b}, \cdot)$ is

$$ K(\mathbf{b}_{d+1}, \cdot) = \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\min} \min_{\tilde{\mathbf{w}}\in\mathcal{H}_L} \sum_{i=1}^{|\mathcal{D}|} ||{}^{(\bullet)}\mathbf{r}_i^d - \tilde{\mathbf{w}} K(\mathbf{b}, \mathbf{z}_i)||^2_{\mathcal{H}_L}, $$

then the following closed form results follow:
i) The optimal weight of the new basis function takes the form

$$ \tilde{\mathbf{w}}_{d+1} = \begin{bmatrix} \boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top \end{bmatrix} \mathbf{w}_{d+1}, $$

where $\boldsymbol{\Phi}_{\mathcal{S}'} := [L(\mathbf{s}'_1, \cdot), ..., L(\mathbf{s}'_n, \cdot)]^\top$, $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n$, $\mathbf{w}_{d+1} \in \mathbb{R}^{m+n}$.
ii) The next basis function is chosen by which takes the form

$$ K(\mathbf{b}_{d+1}, \cdot) = \underset{K(\mathbf{b},\cdot)\,\in\,\mathcal{G}}{\arg\sup} \left[\frac{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}} \mathbf{R} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}} \right], $$

where $\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}^\top = \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}} := [K(\mathbf{b}, \mathbf{z}_1), ..., K(\mathbf{b}, \mathbf{z}_n)]^\top \in \mathbb{R}^n$, $\mathcal{Z} := \{\mathbf{z}_i\}_{i=1}^n$,
$\mathbf{R} = \left(\mathbf{L}_{\mathcal{S}'\mathcal{S}'} - \mathbf{L}_{\mathcal{S}'\mathcal{C}} \mathbf{W} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} - \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} \mathbf{W}^\top \mathbf{L}_{\mathcal{C}\mathcal{S}'} + \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} \mathbf{W}^\top \mathbf{L}_{\mathcal{C}\mathcal{C}} \mathbf{W} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \right) \in \mathbb{R}^{n\times n}$ and for additional notation see section 5.1.1.

Proof. i) Recalling equation (4.5) then for a new basis function $K(\mathbf{b}, \cdot)$,

$$ \tilde{\mathbf{w}}_{d+1} = \left(\sum_{i=1}^{n} K(\mathbf{b}, \mathbf{z}_i)\, {}^{(\bullet)}\mathbf{r}_i^d \right) / \left(\sum_{i=1}^{n} K(\mathbf{b}, \mathbf{z}_i)^2 \right). $$

By expanding the numerator term,

$$\begin{aligned}\sum_{i=1}^{n} K(\mathbf{b},\mathbf{z}_i)\,{}^{(\bullet)}\mathbf{r}_i^d &= \sum_{i=1}^{n} K(\mathbf{b},\mathbf{z}_i)\left(L(\mathbf{s}_i',\bullet) - {}^{(\bullet)}\mu^d(\mathbf{z}_i)\right),\\ &= \boldsymbol{\Phi}_{\mathcal{S}'}^\top \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}} - \boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}},\\ &= \begin{bmatrix}\boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top\end{bmatrix}\begin{bmatrix}-\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}\\ \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}\end{bmatrix},\end{aligned}$$

then dividing by the denominator gives the weight vector,

$$\begin{aligned}\tilde{\mathbf{w}}_{d+1} &= \frac{1}{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}\begin{bmatrix}\boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top\end{bmatrix}\begin{bmatrix}-\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}\\ \boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}\end{bmatrix},\\ &= \begin{bmatrix}\boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top\end{bmatrix}\begin{bmatrix}\boldsymbol{\zeta}\\ \boldsymbol{\beta}\end{bmatrix},\\ &= \begin{bmatrix}\boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top\end{bmatrix}\mathbf{w}_{d+1},\end{aligned} \tag{A.7}$$

where $\boldsymbol{\zeta}\in\mathbb{R}^m$, $\boldsymbol{\beta}\in\mathbb{R}^n$ and the vector $\mathbf{w}_{d+1}\in\mathbb{R}^{m+n}$ is calculated in $\sim\mathcal{O}(md+dn)$.
ii) By equation (4.4) the next basis function is specified by

$$K(\mathbf{b}_{d+1},\cdot) = \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{\sum_{i=1}^n\sum_{k=1}^n K(\mathbf{b},\mathbf{z}_i)\langle {}^{(\bullet)}\mathbf{r}_i^d, {}^{(\bullet)}\mathbf{r}_k^d\rangle_{\mathcal{H}_L} K(\mathbf{b},\mathbf{z}_k)}{\sum_{i=1}^n K(\mathbf{b},\mathbf{z}_i)^2}\right].$$

The inner product in the numerator is

$$\begin{aligned}\langle {}^{(\bullet)}\mathbf{r}_i^d, {}^{(\bullet)}\mathbf{r}_k^d\rangle_{\mathcal{H}_L} &= \left\langle L(\mathbf{s}_i',\bullet) - {}^{(\bullet)}\mu^d(\mathbf{z}_i),\ L(\mathbf{s}_k',\bullet) - {}^{(\bullet)}\mu^d(\mathbf{z}_k)\right\rangle_{\mathcal{H}_L},\\ &= L(\mathbf{s}_i',\mathbf{s}_k') - \left\langle L(\mathbf{s}_i',\bullet), {}^{(\bullet)}\mu^d(\mathbf{z}_k)\right\rangle_{\mathcal{H}_L} - \left\langle {}^{(\bullet)}\mu^d(\mathbf{z}_i), L(\mathbf{s}_k',\bullet)\right\rangle_{\mathcal{H}_L}\\ &\quad + \left\langle {}^{(\bullet)}\mu^d(\mathbf{z}_i), {}^{(\bullet)}\mu^d(\mathbf{z}_k)\right\rangle_{\mathcal{H}_L},\\ &= (\mathbf{L}_{\mathcal{S}'\mathcal{S}'})_{ik} - (\mathbf{L}_{\mathcal{S}'\mathcal{C}}\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}})_{ik} - (\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{S}'})_{ik}\\ &\quad + (\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}})_{ik},\\ &= \left(\mathbf{L}_{\mathcal{S}'\mathcal{S}'} - \mathbf{L}_{\mathcal{S}'\mathcal{C}}\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} - \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{S}'} + \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathcal{C}}\mathbf{W}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\right)_{ik},\\ &= (\mathbf{R})_{ik},\end{aligned}$$

where $\mathbf{R}$ is computed in time $\sim\mathcal{O}(n^2+m^2)$. Finally

$$\begin{aligned}K(\mathbf{b}_{d+1},\cdot) &= \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{\sum_{i=1}^n\sum_{k=1}^n K(\mathbf{b},\mathbf{z}_i)(\mathbf{R})_{ik}K(\mathbf{b},\mathbf{z}_k)}{\sum_{i=1}^n K(\mathbf{b},\mathbf{z}_i)^2}\right],\\ &= \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}\right],\end{aligned}$$

which is computed in $\sim\mathcal{O}(n^2)$ and is analogous to equation (4.7). □

This is a naive application of the original matching pursuit algorithm which assumes the derivative of the loss wrt. $\tilde{\mathbf{w}}$ exists.

Corollary A.1.1. *In addition, the closed form solution* (A.7) *for* $\tilde{\mathbf{w}}_{d+1} \in \mathcal{H}_L$ *is not only a function of the existing basis* $\mathcal{C}$ *but also a basis* $\mathcal{S}'$ *which consists of the successor states in the dataset* $\mathcal{D}$*. Therefore by incrementing* $\mathcal{B}$ *by one basis function also increments the embedding's* $\mathcal{C}$ *basis by* n *new basis functions as illustrated below (using notation from equation* (A.2)*),*

$$\begin{aligned}
{}^{(\bullet)}\hat{\mu}^{d+1}(\mathbf{z}) &= \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \mathbf{z}) + \tilde{\mathbf{w}}_{d+1} K(\mathbf{b}_{d+1}, \mathbf{z}), \\
&= \boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{W} \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}) + \begin{bmatrix} \boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top \end{bmatrix} \mathbf{w}_{d+1} K(\mathbf{b}_{d+1}, \mathbf{z}), \\
&= \begin{bmatrix} \boldsymbol{\Phi}_{\mathcal{C}}^\top & \boldsymbol{\Phi}_{\mathcal{S}'}^\top \end{bmatrix} \begin{bmatrix} \mathbf{W} & \boldsymbol{\zeta} \\ \mathbf{0} & \boldsymbol{\beta} \end{bmatrix} \begin{bmatrix} \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}) \\ K(\mathbf{b}_{d+1}, \mathbf{z}) \end{bmatrix}, \\
&= \boldsymbol{\Phi}_{\mathcal{C}'}^\top \mathbf{W}' \boldsymbol{\psi}_{\mathcal{B}'}(\mathbf{z}),
\end{aligned}$$

where $\mathcal{C}' := \mathcal{C} \bigcup \mathcal{S}'$, $|\mathcal{C}'| = m+n$, $\mathbf{W}' \in \mathbb{R}^{(m+n)\times(d+1)}$, $\mathcal{B}' := \mathcal{B} \bigcup K(\mathbf{b}_{d+1}, \cdot)$, $|\mathcal{B}'| = d+1$ *and* $\mathbf{0} \in \mathbb{R}^{n \times d}$.

In order to make this approach sustainable would require sparsification of the $\mathcal{C}$ basis continuously which is already a difficult procedure. It might be possible to sparsify $\mathcal{C}$ using the RKHS norm sparsification variant of matching pursuit described in lemma A.6 but this has not been explored. What also makes this approach difficult is the overhead required to manage the calculation of $\mathbf{R}$ between adding each new basis function. It was decided to abandon the RKHS-valued approach and develop one that doesn't add new bases to $\mathcal{C}$.

A.1.3 Other Variants

Lemma A.1 demonstrated that if full RKHS-valued matching pursuit was used to increment $\mathcal{B}$, then as a side effect $\mathcal{C}$ is also enriched by the size of the data set. As a way to mitigate this problem, two modified matching pursuit algorithms are outlined below that enrich any one set of basis functions but keeps the other constant.

Maintaining $\mathcal{B}$

Lemma A.2 (Modified Matching Pursuit for learning $\mathcal{B}$)**.** *Assume the embedding takes the form in equation* (A.2) *then the model residue for the current basis* $\mathcal{B}$ *is*

$${}^{(\bullet)}\mathbf{r}_i^d := L(\mathbf{s}_i', \bullet) - \sum_{\ell=1}^{d} \left(\boldsymbol{\Phi}_{\mathcal{C}}^\top \mathbf{w}_\ell \right) K(\mathbf{b}_\ell, \mathbf{z}_i) \in \mathcal{H}_L, \quad \mathbf{z} \in \mathcal{S} \times \mathcal{A}.$$

Given a dictionary of basis functions $\mathcal{G} := \{K(\hat{\mathbf{b}}_1, \bullet), K(\hat{\mathbf{b}}_2, \bullet), \ldots\}$ *as candidates and dataset* $\mathcal{D} := \{(\mathbf{z}, \mathbf{s}')_i\}_{i=1}^n$ *where* $n = |\mathcal{D}|$*, then by keeping* $\mathcal{C}$ *constant, we want to solve*

the optimisation problem

$$K(\mathbf{b}_{d+1},\bullet) = \underset{K(\mathbf{b},\bullet)\in\mathcal{G}}{\arg\min}\ \min_{\mathbf{w}\in\mathbb{R}^m} \sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^d - \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i)||_{\mathcal{H}_L}^2 ,$$

by greedily selecting bases from $\mathcal{G}$. The following closed form results follow:

i) The optimal weight $\mathbf{w}_{d+1}$ of a new basis function $K(\mathbf{b},\cdot)$ that minimises the loss is

$$\mathbf{w}_{d+1} = (\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}/\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}} \in \mathbb{R}^m,$$

where $\mathbf{R}_{\mathcal{CD}} := \mathbf{L}_{\mathcal{CS}'} - \mathbf{L}_{\mathcal{CC}}\mathbf{W}\mathbf{\Upsilon}_{\mathcal{BZ}} \in \mathbb{R}^{m\times n}$ which costs $\mathcal{O}(m^3 + m^2 n)$.

ii) The next basis function is specified by

$$K(\mathbf{b}_{d+1},\cdot) = \underset{K(\mathbf{b},\cdot)\,\in\,\mathcal{G}}{\arg\sup}\left[\frac{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}_{\mathcal{DC}}(\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}\right]$$

which costs $\sim\mathcal{O}(m^2 n)$ if the inverse is already calculated.

Proof. i) Since $\nabla_{\mathbf{w}} \sum_{i=1}^n ||^{(\bullet)}\mathbf{r}_i^d - \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i)||_{\mathcal{H}_L}^2 = \mathbf{0}$ at the minimum we have,

$$\begin{aligned}
\mathbf{0} &= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left(\left\langle \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i), \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i)\right\rangle_{\mathcal{H}_L} - 2\left\langle \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i), {}^{(\bullet)}\mathbf{r}_i^d \right\rangle_{\mathcal{H}_L} \right), \\
&= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left(K(\mathbf{z}_i,\mathbf{b})\mathbf{w}^\top \mathbf{L}_{\mathcal{CC}}\mathbf{w} K(\mathbf{b},\mathbf{z}_i) - 2K(\mathbf{z}_i,\mathbf{b})\mathbf{w}^\top\ {}^{(\mathcal{C})}\mathbf{r}_i^d \right), \\
&= \sum_{i=1}^{n} 2K(\mathbf{z}_i,\mathbf{b})\mathbf{L}_{\mathcal{CC}}\mathbf{w} K(\mathbf{b},\mathbf{z}_i) - 2K(\mathbf{z}_i,\mathbf{b})\ {}^{(\mathcal{C})}\mathbf{r}_i^d, \\
&= \mathbf{L}_{\mathcal{CC}}\mathbf{w}\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}} - \mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}, \\
\Rightarrow \mathbf{w}_{d+1} &= (\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}/\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}},
\end{aligned}$$

where ${}^{(\mathcal{C})}\mathbf{r}_i^d := \mathbf{L}_{\mathcal{CS}'_i} - \mathbf{L}_{\mathcal{CC}}\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) \in \mathbb{R}^m$, $\mathbf{R}_{\mathcal{CD}} := \mathbf{L}_{\mathcal{CS}'} - \mathbf{L}_{\mathcal{CC}}\mathbf{W}\mathbf{\Upsilon}_{\mathcal{BZ}} \in \mathbb{R}^{m\times n}$ and equation (B.79) is used in line 3.

ii) Substituting the value of the new weight into the objective,

$$\begin{aligned}
&\sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^d - \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{w} K(\mathbf{b},\mathbf{z}_i)||_{\mathcal{H}_L}^2, \\
&= \sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^m||_{\mathcal{H}_L}^2 + K(\mathbf{z}_i,\mathbf{b})\mathbf{w}^{\min\top}\mathbf{L}_{\mathcal{CC}}\mathbf{w}^{\min} K(\mathbf{b},\mathbf{z}_i) - 2K(\mathbf{z}_i,\mathbf{b})\mathbf{w}^{\min\top}\ {}^{(\mathcal{C})}\mathbf{r}_i^d, \\
&= \sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^m||_{\mathcal{H}_L}^2 + \mathbf{w}^{\min\top}\mathbf{L}_{\mathcal{CC}}\mathbf{w}^{\min}\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}} - 2\mathbf{w}^{\min\top}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathcal{B}}, \\
&= \sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^m||_{\mathcal{H}_L}^2 + \frac{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}_{\mathcal{DC}}(\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{(\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}})^2} - 2\frac{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}_{\mathcal{DC}}(\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}, \\
&= \sum_{i=1}^{n} ||^{(\bullet)}\mathbf{r}_i^m||_{\mathcal{H}_L}^2 - \frac{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}_{\mathcal{DC}}(\mathbf{L}_{\mathcal{CC}})^{-1}\mathbf{R}_{\mathcal{CD}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\mathbf{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{\Upsilon}_{\mathcal{Z}\mathbf{b}}},
\end{aligned}$$

then next basis that maximally minimises the loss is

$$K(\mathbf{b}_{d+1},\cdot) = \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}\mathbf{R}_{\mathcal{D}\mathcal{C}}(\mathbf{L}_{\mathcal{C}\mathcal{C}})^{-1}\mathbf{R}_{\mathcal{C}\mathcal{D}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}{\boldsymbol{\Upsilon}_{\mathbf{b}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathbf{b}}}\right].$$

□

Corollary A.1.2. *If the original embedding is* ${}^{(\bullet)}\mu^d(\cdot) = \boldsymbol{\Phi}_{\mathcal{C}}^\top\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot)$ *then the embedding with* $\mathcal{B}'$ *(which is* $\mathcal{B}$ *augmented by one new basis function using matching pursuit regression) is*

$${}^{(\bullet)}\mu^{d+1}(\cdot) \;=\; \boldsymbol{\Phi}_{\mathcal{C}}^\top\begin{bmatrix}\mathbf{W} & \mathbf{w}_{d+1}\end{bmatrix}\begin{bmatrix}\boldsymbol{\psi}_{\mathcal{B}}(\cdot)\\ K(\mathbf{b}_{d+1},\cdot)\end{bmatrix}.$$

Maintaining $\mathcal{C}$

Lemma A.3 (Modified Matching Pursuit for learning $\mathcal{C}$)**.** *Assume the embedding takes the form in equation (A.4) then the model residue for the current basis* $\mathcal{C}$ *is*

$${}^{(\bullet)}\mathbf{r}_i^m := L(\mathbf{s}_i',\bullet) - \sum_{j=1}^m L(\mathbf{c}_j,\bullet)\mathbf{w}_{j:}\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) \in \mathcal{H}_L.$$

Given a dictionary of basis functions $\mathcal{G} := \{L(\hat{\mathbf{c}}_1,\bullet), L(\hat{\mathbf{c}}_2,\bullet), ...\}$ *as candidates and dataset* $\mathcal{D} := \{(\mathbf{z},\mathbf{s}')_i\}_{i=1}^n$ *where* $n = |\mathcal{D}|$ *then by keeping* $\mathcal{B}$ *constant, we want to solve the optimisation problem*

$$L(\mathbf{c}_{m+1},\bullet) = \underset{L(\mathbf{c},\bullet)\in\mathcal{G}}{\arg\min}\min_{\mathbf{w}\in\mathbb{R}^d}\sum_{i=1}^n ||{}^{(\bullet)}\mathbf{r}_i^m - L(\mathbf{c},\bullet)\mathbf{w}^\top\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i)||_{\mathcal{H}_L}^2,$$

by greedily selecting bases from $\mathcal{G}$*. The following closed form results follow:*

i) The optimal weight $\mathbf{w}_{m+1}$ *of a new basis function* $L(\mathbf{c},\cdot)$ *that minimises the loss is*

$$\mathbf{w}_{m+1} = (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\mathbf{R}_{\mathcal{D}\mathbf{c}}/L(\mathbf{c},\mathbf{c}) \in \mathbb{R}^d,$$

where $\mathbf{w}_{m+1}^\top = \mathbf{w}_{m+1:}$, $\mathbf{R}_{\mathcal{D}\mathbf{c}} := \mathbf{L}_{\mathcal{S}'\mathbf{c}} - \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\mathbf{W}^\top\mathbf{L}_{\mathcal{C}\mathbf{c}} \in \mathbb{R}^n$ *and* $\mathbf{L}_{\mathcal{C}\mathbf{c}} := [L(\mathbf{c},\mathbf{c}_1),...,L(\mathbf{c},\mathbf{c}_m)]^\top$*, which costs* $\sim\mathcal{O}(d^3 + d^2n)$*.*

ii) The next basis function is specified by

$$L(\mathbf{c}_{m+1},\bullet) = \underset{L(\mathbf{c},\bullet)\in\mathcal{G}}{\arg\sup}\left[\mathbf{R}_{\mathbf{c}\mathcal{D}}\,\boldsymbol{\Pi}\,\mathbf{R}_{\mathcal{D}\mathbf{c}}/L(\mathbf{c},\mathbf{c})\right],$$

where $\boldsymbol{\Pi} = \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\,(\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}}\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1}\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \in \mathbb{R}^{n\times n}$ *which costs* $\sim\mathcal{O}(d^3 + nd^2 + n^2)$ *in total.*

Proof. i) Since $\nabla_{\mathbf{w}} \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^m - L(\mathbf{c},\bullet)\mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i)||^2_{\mathcal{H}_L} = \mathbf{0}$ at the minimum we have,

$$
\begin{aligned}
\mathbf{0} &= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \Big(\Big\langle L(\mathbf{c},\bullet)\mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i), L(\mathbf{c},\bullet)\mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) \Big\rangle_{\mathcal{H}_L} \\
&\qquad - 2 \Big\langle L(\mathbf{c},\bullet)\mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i), {}^{(\bullet)}\mathbf{r}_i^m \Big\rangle_{\mathcal{H}_L} \Big) \\
&= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \Big(\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i)\, \mathbf{w} \Big\langle L(\mathbf{c},\bullet), L(\mathbf{c},\bullet) \Big\rangle_{H_L} \mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) \\
&\qquad - 2\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i)\, \mathbf{w} \Big\langle L(\mathbf{c},\bullet), {}^{(\bullet)}\mathbf{r}_i^m \Big\rangle_{\mathcal{H}_L} \Big), \\
&= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left(\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i)\, \mathbf{w}\, L(\mathbf{c},\mathbf{c}) \mathbf{w}^\top \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) - 2\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i) \mathbf{w}\ {}^{(\mathbf{c})}\mathbf{r}_i^m \right), \\
&= \sum_{i=1}^{n} 2L(\mathbf{c},\mathbf{c}) \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) \boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i) \mathbf{w} - 2\boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i)\ {}^{(\mathbf{c})}\mathbf{r}_i^m, \\
&= L(\mathbf{c},\mathbf{c}) \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} \mathbf{w} - \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}}, \\
\Rightarrow \mathbf{w}_{m+1} &= (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} / L(\mathbf{c},\mathbf{c}),
\end{aligned}
$$

where $\mathbf{w}_{m+1} \in \mathbb{R}^d$ and ${}^{(\mathbf{c})}\mathbf{r}_i^m := L(\mathbf{s}_i', \mathbf{c}) - \boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i) \mathbf{W}^\top \mathbf{L}_{\mathcal{C}\mathbf{c}} \in \mathbb{R}$.

ii) Substituting the value of the new weight into the objective,

$$
\begin{aligned}
&\sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^m - L(\mathbf{c},\bullet)\mathbf{w}^{\min\top} \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i)||^2_{\mathcal{H}_L} \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} + L(\mathbf{c},\mathbf{c}) \boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i)\, \mathbf{w}^{\min} \mathbf{w}^{\min\top} \boldsymbol{\psi}_{\mathcal{B}}(\mathbf{z}_i) - 2\boldsymbol{\psi}_{\mathcal{B}}^\top(\mathbf{z}_i)\ \mathbf{w}^{\min}\ {}^{(\mathbf{c})}\mathbf{r}_i^d, \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} + \mathrm{Tr}\left(L(\mathbf{c},\mathbf{c}) \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\, \mathbf{w}^{\min} \mathbf{w}^{\min\top} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} - 2\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}}\, \mathbf{w}^{\min} \mathbf{R}_{\mathbf{c}\mathcal{D}} \right), \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} + \mathrm{Tr}\Big(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \\
&\qquad - 2\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} \Big) / L(\mathbf{c},\mathbf{c}), \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} + \mathrm{Tr}\Big(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} \\
&\qquad - 2\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \Big) / L(\mathbf{c},\mathbf{c}), \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} + \mathrm{Tr}\Big(\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} \\
&\qquad - 2\boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}} (\boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \boldsymbol{\Upsilon}_{\mathcal{Z}\mathcal{B}})^{-1} \boldsymbol{\Upsilon}_{\mathcal{B}\mathcal{Z}} \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} \Big) / L(\mathbf{c},\mathbf{c}), \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} - \mathrm{Tr}\Big(\boldsymbol{\Pi}\, \mathbf{R}_{\mathcal{D}\mathbf{c}} \mathbf{R}_{\mathbf{c}\mathcal{D}} / L(\mathbf{c},\mathbf{c}) \Big), \\
&= \sum_{i=1}^{n} ||{}^{(\bullet)}\mathbf{r}_i^d||^2_{\mathcal{H}_L} - \mathbf{R}_{\mathbf{c}\mathcal{D}}\, \boldsymbol{\Pi}\, \mathbf{R}_{\mathcal{D}\mathbf{c}} / L(\mathbf{c},\mathbf{c}),
\end{aligned}
$$

where the trace's cyclic property is exploited. The next basis that maximally minimises the loss is

$$L(\mathbf{c}_{m+1}, \bullet) = \underset{L(\mathbf{c},\bullet)\in\mathcal{G}}{\arg\sup} \Big[\mathbf{R}_{\mathbf{c}\mathcal{D}}\, \mathbf{\Pi}\, \mathbf{R}_{\mathcal{D}\mathbf{c}} / L(\mathbf{c}, \mathbf{c}) \Big].$$

□

Corollary A.1.3. *If the original embedding is* ${}^{(\bullet)}\mu^m(\cdot) = \mathbf{\Phi}_{\mathcal{C}}^\top \mathbf{W} \boldsymbol{\psi}_{\mathcal{B}}(\cdot)$ *then the embedding with* $\mathcal{C}'$ *(which is* $\mathcal{C}$ *augmented by one new basis function using matching pursuit regression) is*

$${}^{(\bullet)}\mu^{m+1}(\cdot) \;=\; \begin{bmatrix} \mathbf{\Phi}_{\mathcal{C}}^\top & L(\mathbf{c}_{m+1}, \cdot) \end{bmatrix} \begin{bmatrix} \mathbf{W} \\ \mathbf{w}_{m+1}^\top \end{bmatrix} \boldsymbol{\psi}_{\mathcal{B}}(\cdot).$$

A.1.4 Sparsification in the vvRKHS Norm

The following is matching pursuit specialised to a vvRKHS function $\mathbf{h} \in \mathcal{H}_{\mathbf{\Gamma}}$ and builds an approximation $\hat{\mathbf{f}} \in \mathcal{H}_{\mathbf{\Gamma}}$ by incrementally adding new basis functions that minimises $||\mathbf{h} - \hat{\mathbf{f}}||_{\mathbf{\Gamma}}$. This is not a solution to a regression problem that requires data as in the previous matching pursuit variants. Instead this is a sparsification algorithm that exploits the RKHS norm: if any two functions are close in their RKHS norm then their point evaluations, over their entire domain, are also close. The following is not necessarily specialised to an embedding.

Lemma A.4 (Vector-Valued Matching Pursuit for Sparsification)**.** *A known vvRKHS function* $\mathbf{h} \in \mathcal{H}_{\mathbf{\Gamma}}$ *is given where* $\mathbf{h} : \mathcal{X} \to \mathcal{Y}$*,* $\mathcal{X}$ *is a non-empty set and* $\mathcal{Y}$ *is Hilbert space. The vvRKHS kernel* $\mathbf{\Gamma}(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}')\mathbf{I}$ *(where* $\mathbf{I} : \mathcal{Y} \to \mathcal{Y}$*) is associated with a scalar-valued RKHS* $\mathcal{H}_K$ *of functions over* $\mathcal{X}$*. If* $\mathbf{h}$ *is a weighted sum of* d' *kernels* $K(\mathbf{x}, \cdot) : \mathcal{X} \to \mathbb{R}$*, then*

$$\mathbf{h}(\cdot) := \sum_{\ell=1}^{d'} \tilde{\mathbf{w}}_\ell K(\tilde{\mathbf{x}}_\ell, \cdot) \in \mathcal{H}_{\mathbf{\Gamma}}, \quad \tilde{\mathbf{w}}_\ell \in \mathcal{Y}.$$

Given a dictionary of candidate basis functions made from the existing d' *kernels* $\mathcal{G} := \{K(\tilde{\mathbf{x}}_1, \cdot), ..., K(\tilde{\mathbf{x}}_{d'}, \cdot)\}$*, then matching pursuit aims to find a sparse approximation to* $\mathbf{h}$ *in the form*

$$\hat{\mathbf{f}}^d(\cdot) := \sum_{\ell=1}^{d} \mathbf{w}_\ell K(\mathbf{x}_\ell, \cdot) \in \mathcal{H}_{\mathbf{\Gamma}}, \quad \tilde{\mathbf{w}}_\ell \in \mathcal{Y},$$

where $d<d'$*,* $\mathcal{B} = \{K(\mathbf{x}_\ell \cdot) \in \mathcal{H}_K\}_{\ell=1}^d \subseteq \mathcal{G}$ *is the sparsified basis and* $\{\mathbf{w}_\ell \in \mathcal{Y}\}_{\ell=1}^d$ *are the basis weights. If matching pursuit greedily adds a new basis element* $K(\mathbf{x}_{d+1}, \cdot)$ *to*

minimise the RKHS norm,

$$K(\mathbf{x}_{d+1},\cdot) = \underset{K(\mathbf{x},\cdot)\in\mathcal{G}}{\arg\min}\min_{\mathbf{w}\in\mathcal{Y}} ||\mathbf{h}(\cdot) - (\hat{\mathbf{f}}^d(\cdot) + \mathbf{w}K(\mathbf{x},\cdot))||^2_{\mathbf{\Gamma}},$$

then the following closed form results are as follows:

i) If the residue for the d^{th} estimator is defined as

$$\mathbf{r}^d(\cdot) = \mathbf{h}(\cdot) - \hat{\mathbf{f}}^d(\cdot) \in \mathcal{H}_{\mathbf{\Gamma}},$$

then the weight of the new basis element $K(\mathbf{x},\cdot)$ is

$$\mathbf{w}_{d+1} = \mathbf{r}^d(\mathbf{x})/K(\mathbf{x},\mathbf{x}) \in \mathcal{Y}, \tag{A.8}$$

where $\mathbf{r}^d(\mathbf{x}) = \mathbf{h}(\mathbf{x}) - \hat{\mathbf{f}}^d(\mathbf{x}) \in \mathcal{Y}$, and

ii) the new basis element $K(\mathbf{x}_{d+1},\cdot)$ is the solution to

$$K(\mathbf{x}_{d+1},\cdot) = \underset{K(\mathbf{x},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{||\mathbf{r}^d(\mathbf{x})||^2_{\mathcal{Y}}}{K(\mathbf{x},\mathbf{x})}\right]. \tag{A.9}$$

Proof. i) Beginning with the optimisation problem,

$$\begin{aligned} g_{d+1} &= \underset{K(\mathbf{x},\cdot)\in\mathcal{G}}{\arg\min}\min_{\mathbf{w}\in\mathcal{Y}} ||\mathbf{h}(\cdot) - (\hat{\mathbf{f}}^d(\cdot) + \mathbf{w}K(\mathbf{x},\cdot))||^2_{\mathbf{\Gamma}}, \\ &= \underset{K(\mathbf{x},\cdot)\in\mathcal{G}}{\arg\min}\min_{\mathbf{w}\in\mathcal{Y}} ||\mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x},\cdot)||^2_{\mathbf{\Gamma}}, \end{aligned}$$

then since $\nabla_{\mathbf{w}}||\mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x},\cdot)||^2_{\mathbf{\Gamma}} = \mathbf{0}$ at the minimum and by lemma A.5, we have

$$\begin{aligned} \mathbf{0} &= \nabla_{\mathbf{w}}\left(-2\langle \mathbf{w}, \mathbf{r}^d(\mathbf{x})\rangle_{\mathcal{Y}} + ||\mathbf{w}||^2_{\mathcal{Y}}\,K(\mathbf{x},\mathbf{x})\right), \\ &= -2\mathbf{r}^d(\mathbf{x}) + 2\mathbf{w}K(\mathbf{x},\mathbf{x}), \\ \Rightarrow \mathbf{w}_{d+1} &= \mathbf{r}^d(\mathbf{x})/K(\mathbf{x},\mathbf{x}) \in \mathcal{Y}. \end{aligned}$$

ii) Substituting the value of the minimiser $\mathbf{w}_{\min} = \mathbf{w}_{d+1}$ into the objective expanded by lemma A.5, then

$$\begin{aligned} &||\mathbf{r}^d(\cdot) - \mathbf{w}_{\min}K(\mathbf{x},\cdot)||^2_{\mathbf{\Gamma}} \\ &= ||\mathbf{r}^d(\cdot)||^2_{\mathbf{\Gamma}} - 2\langle \mathbf{w}_{\min}, \mathbf{r}^d(\mathbf{x})\rangle_{\mathcal{Y}} + ||\mathbf{w}_{\min}||^2_{\mathcal{Y}}\,K(\mathbf{x},\mathbf{x}), \\ &= ||\mathbf{r}^d(\cdot)||^2_{\mathbf{\Gamma}} - \frac{||\mathbf{r}^d(\mathbf{x})||^2_{\mathcal{Y}}}{K(\mathbf{x},\mathbf{x})}, \end{aligned}$$

then the next basis that maximally minimises the loss is

$$K(\mathbf{x}_{d+1},\cdot) = \underset{K(\mathbf{x},\cdot)\in\mathcal{G}}{\arg\sup}\left[\frac{||\mathbf{r}^d(\mathbf{x})||^2_{\mathcal{Y}}}{K(\mathbf{x},\mathbf{x})}\right],$$

which can be directly compared to the matching pursuit regression results in lemma 4. □

Algorithm 24 MatchingPursuitSparsification($\mathbf{h}$, $d_{\max}$, δ_{tol})

Input: vvRKHS function $\mathbf{h} \in \mathcal{H}_{\mathbf{\Gamma}}$, $\mathbf{h} : \mathcal{X} \to \mathcal{Y}$ of the form $\mathbf{h}(\cdot) = \sum_{\ell=1}^{d'} \tilde{\mathbf{w}}_\ell K(\tilde{\mathbf{x}}_\ell, \cdot)$, maximum sparse basis count $d_{\max} \leq d'$, RKHS norm tolerance δ.
Output: $\hat{\mathbf{f}}^d(\cdot) = \sum_{\ell=1}^{d} \mathbf{w}_\ell K(\mathbf{x}_\ell, \cdot)$ where $d \leq d_{\max}$ or $||\mathbf{h} - \hat{\mathbf{f}}^d||_{\mathbf{\Gamma}}^2 < \delta_{\text{tol}}$.
Initialise: $d \leftarrow 0$, $\delta \leftarrow \infty$, $\hat{\mathbf{f}}^0 \leftarrow \emptyset$, $\mathcal{G} := \{K(\tilde{\mathbf{x}}_\ell, \cdot)\}_{\ell=1}^{d'}$, $\mathbf{r}^0(\mathbf{x}) := \mathbf{h}(\mathbf{x})\ \forall \mathbf{x} \in \mathcal{G}$.
while $d < d_{\max}$ **or** $\delta > \delta_{\text{tol}}$ **do**
  $K(\mathbf{x}^*, \cdot) = \underset{K(\mathbf{x},\cdot) \in \mathcal{G}}{\arg\sup} \left[\frac{||\mathbf{r}^d(\mathbf{x})||_{\mathcal{Y}}^2}{K(\mathbf{x},\mathbf{x})} \right]$ ▷ equation (A.9)
  $\mathbf{w}^* = \mathbf{r}^d(\mathbf{x}^*)/K(\mathbf{x}^*, \mathbf{x}^*)$ ▷ equation (A.8)
  $\hat{\mathbf{f}} \leftarrow \hat{\mathbf{f}} + \mathbf{w}^* K(\mathbf{x}^*, \cdot)$
  $\mathcal{G} \leftarrow \mathcal{G} \backslash K(\mathbf{x}^*, \cdot)$ ▷ Remove basis from $\mathcal{G}$
  $d \leftarrow d + 1$
  for each $K(\mathbf{x}, \dot{}) \in \mathcal{G}$ **do**
    $\mathbf{r}^d(\mathbf{x}) \leftarrow \mathbf{r}^d(\mathbf{x}) - \hat{\mathbf{f}}(\mathbf{x})$ ▷ Residue is maintained over the dictionary basis
  end for
  $\delta \leftarrow ||\mathbf{r}^d(\cdot)||^2_{\mathbf{\Gamma}} = \frac{1}{2|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} ||\mathbf{r}^d(\mathbf{x}_i)||_{\mathcal{Y}}^2$ ▷ $\mathbf{r}^d(\cdot) = \mathbf{h} - \hat{\mathbf{f}}$
end while
return $\hat{\mathbf{f}}$

Lemma A.5 (Residue loss in the RKHS norm)**.** *Assume the current residue is some vvRKHS function (as defined in lemma A.4) is decomposed into some expansion* $\mathbf{r}^d(\cdot) := \sum_\ell \bar{\mathbf{w}}_\ell K(\bar{\mathbf{x}}_\ell, \cdot)$ *where* $\bar{\mathbf{w}}_\ell \in \mathcal{Y}$*. Then the squared loss when a new* $\mathbf{w}K(\mathbf{x}_\ell, \cdot)$ *is subtracted is,*

$$||\mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x}, \cdot)||_{\mathbf{\Gamma}}^2 = ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\langle \mathbf{w}, \mathbf{r}^d(\mathbf{x}) \rangle_{\mathcal{Y}} + ||\mathbf{w}||_{\mathcal{Y}}^2 K(\mathbf{x}, \mathbf{x}),$$

where $\mathbf{w} \in \mathcal{Y}$*.*

Proof.

$$\begin{aligned}
||\mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x}, \cdot)||_{\mathbf{\Gamma}}^2 &= \left\langle \mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x}, \cdot),\ \mathbf{r}^d(\cdot) - \mathbf{w}K(\mathbf{x}, \cdot) \right\rangle_{\mathbf{\Gamma}}, \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\langle \mathbf{w}K(\mathbf{x}, \cdot), \mathbf{r}^d(\cdot) \rangle_{\mathbf{\Gamma}} + \langle \mathbf{w}K(\mathbf{x}, \cdot), \mathbf{w}K(\mathbf{x}, \cdot) \rangle_{\mathbf{\Gamma}}, \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\langle \mathbf{w}K(\mathbf{x}, \cdot), \sum_\ell \bar{\mathbf{w}}_\ell K(\bar{\mathbf{x}}_\ell, \cdot) \rangle_{\mathbf{\Gamma}} + \langle \mathbf{w}, \mathbf{w} \rangle_{\mathcal{Y}} \langle K(\mathbf{x}, \cdot), K(\mathbf{x}, \cdot) \rangle_{\mathcal{H}_K}, \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\sum_\ell \langle \mathbf{w}, \bar{\mathbf{w}}_\ell \rangle_{\mathcal{Y}} \langle K(\bar{\mathbf{x}}_\ell, \cdot), K(\mathbf{x}, \cdot) \rangle_{\mathcal{H}_K} + \langle \mathbf{w}, \mathbf{w} \rangle_{\mathcal{Y}} K(\mathbf{x}, \mathbf{x}), \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\sum_\ell \langle \mathbf{w}, \bar{\mathbf{w}}_\ell \rangle_{\mathcal{Y}} K(\bar{\mathbf{x}}_\ell, \mathbf{x}) + \langle \mathbf{w}, \mathbf{w} \rangle_{\mathcal{Y}} K(\mathbf{x}, \mathbf{x}), \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\sum_\ell \langle \mathbf{w}, \bar{\mathbf{w}}_\ell K(\bar{\mathbf{x}}_\ell, \mathbf{x}) \rangle_{\mathcal{Y}} + \langle \mathbf{w}, \mathbf{w} \rangle_{\mathcal{Y}} K(\mathbf{x}, \mathbf{x}), \\
&= ||\mathbf{r}^d(\cdot)||_{\mathbf{\Gamma}}^2 - 2\langle \mathbf{w}, \mathbf{r}^d(\mathbf{x}) \rangle_{\mathcal{Y}} + ||\mathbf{w}||_{\mathcal{Y}}^2 K(\mathbf{x}, \mathbf{x}).
\end{aligned}$$

where the vvRKHS inner product $\langle \cdot, \cdot \rangle_{\mathbf{\Gamma}}$ is expanded by equation (B.69). Notation $\bar{\mathbf{w}}$ and $K(\bar{\mathbf{x}}, \cdot)$ represent weights and basis functions in the residue $\mathbf{r}^d(\cdot)$ respectively. □

The algorithm implementing this sparsification technique only requires maintaining residues $\mathbf{r}^d(\mathbf{x}) \in \mathcal{Y}$ over $\forall K(\mathbf{x}, \cdot) \in \mathcal{G}$. If $\mathcal{Y}$ is an RKHS then tracking the residue basis points is required and adds overhead to the calculation. Although not implemented in this investigation, the next section outlines such a case for the embedding.

A.1.5 Sparsifying Embeddings in the RKHS norm

Embedding notation in equation (A.1.1) is assumed, then by adapting lemma A.4 we set $\mathcal{X}$ as state-actions $\mathcal{Z}$ and $\mathcal{Y}$ as an RKHS $\mathcal{H}_L$ over states $\mathcal{S}$. If the derivative $\nabla_{\mathbf{w}}$ of the loss function is defined then it may be possible to sparsify the embedding in the $\mathcal{B}$ basis by directly applying lemma A.4 as sketched below.

Lemma A.6 (Sparsify $\mathcal{B}$ in the RKHS norm)**.** *Given an existing embedding to be sparsified in the form of equation* (A.2)

$$^{(\bullet)}\mathbf{h}(\cdot) := \sum_{\ell=1}^{d'} \tilde{\mathbf{w}}_\ell K(\tilde{\mathbf{b}}_\ell, \cdot), \quad \tilde{\mathbf{w}}_\ell \in \mathcal{H}_L,$$

from which a dictionary of candidate basis functions $\mathcal{G} := \{K(\tilde{\mathbf{b}}_1, \cdot), ..., K(\tilde{\mathbf{b}}_{d'}, \cdot)\}$ *is defined. Matching pursuit aims to find a sparse approximation to* $\mathbf{h}$ *in the form*

$$^{(\bullet)}\hat{\mathbf{f}}^d(\cdot) := \sum_{\ell=1}^{d} \mathbf{w}_\ell K(\mathbf{b}_\ell, \cdot),$$

where $d<d'$*. If matching pursuit greedily adds a new basis element from* $\mathcal{G}$ *to minimise the residue in the RKHS norm,*

$$K(\mathbf{b}_{d+1}, \cdot) = \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\min} \min_{\mathbf{w}\in\mathcal{H}_L} || ^{(\bullet)}\mathbf{h}(\cdot) - \left(^{(\bullet)}\hat{\mathbf{f}}^d(\cdot) + \mathbf{w}K(\mathbf{b}, \cdot) \right) ||^2_{\mathbf{\Gamma}},$$

then by lemma A.4 the following closed form results are as follows:

i) If the residue for the d^{th} *estimator is defined as*

$$^{(\bullet)}\mathbf{r}^d(\cdot) = {}^{(\bullet)}\mathbf{h}(\cdot) - {}^{(\bullet)}\hat{\mathbf{f}}^d(\cdot) \in \mathcal{H}_{\mathbf{\Gamma}},$$

then the weight of a new basis element $K(\mathbf{b}, \cdot)$ *is*

$$\mathbf{w}_{d+1} = {}^{(\bullet)}\mathbf{r}^d(\mathbf{b}) / K(\mathbf{b}, \mathbf{b}) \in \mathcal{H}_L,$$

and ii) the new basis element is the solution to

$$K(\mathbf{b}_{d+1}, \cdot) = \underset{K(\mathbf{b},\cdot)\in\mathcal{G}}{\arg\sup} \left[\frac{||^{(\bullet)}\mathbf{r}^d(\mathbf{b})||^2_{\mathcal{H}_L}}{K(\mathbf{b}, \mathbf{b})} \right]. \tag{A.10}$$

Corollary A.1.4. *By considering an existing embedding to be sparsified of the form in equation (A.4) and dictionary of candidate basis functions* $\mathcal{G} := \{L(\hat{\mathbf{c}}_1, \bullet), ..., L(\hat{\mathbf{c}}_{m'}, \bullet)\}$*, then by a similar argument it might be possible sparsify* $\mathcal{C}$ *such that*

$$\mathbf{w}_{m+1}^{\top} = {}^{(\mathbf{c})}\mathbf{r}^m(\cdot)/L(\mathbf{c}, \mathbf{c}) \in \mathcal{H}_K,$$

and the new basis function is given by

$$L(\mathbf{c}_{m+1}, \bullet) = \underset{L(\mathbf{c},\bullet)\in\mathcal{G}}{\arg\sup} \left[\frac{||{}^{(\mathbf{c})}\mathbf{r}^m(\cdot)||^2_{\mathcal{H}_K}}{L(\mathbf{c}, \mathbf{c})} \right]. \tag{A.11}$$

Both conditions A.6 and A.11 require tracking residue basis functions because for an embedding $\mathcal{Y}$ is an RKHS. Consider for example when $\mathcal{Y}$ is a regular Hilbert space, then successive residues after each new basis function has been added can be updated quickly as $r^d(\mathbf{x})$ seen in line 11 in algorithm 24 is a vector of real values. However if $\mathcal{Y}$ is an RKHS, then the new residual calculation requires the tracking of basis functions. An efficient way of applying lemma A.6 to an embedding would be to follow a similar approach to section 5.1.1 by approximating the target embedding so $\mathcal{Y}$ is a regular Hilbert space.

Approximate Matching Pursuit in the RKHS Norm for Maintaining $\mathcal{B}$

The embedding is approximated in the same way as for approximate matching pursuit regression in section 5.1.1. Using notation in section 5.1.1, then a target embedding $\mathbf{h} \in \mathcal{H}_{\mathbf{\Gamma}}$ is approximated by making a Cholesky decomposition $\mathbf{\Phi}_{\mathcal{C}}^{\top} \approx \mathbf{P} = [\mathbf{p}_1, ..., \mathbf{p}_m] \in \mathbb{R}^{m_{\text{chol}} \times m}$,

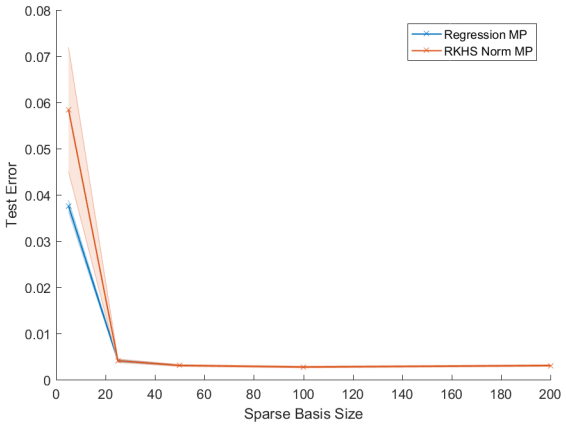
$$\begin{aligned}
{}^{(\bullet)}\mathbf{h}(\cdot) &= \sum_{\ell=1}^{d} \tilde{\mathbf{w}}_\ell K(\mathbf{b}_\ell, \cdot), \quad \tilde{\mathbf{w}}_\ell \in \mathcal{H}_L \\
&= \mathbf{\Phi}_{\mathcal{C}}^{\top} \mathbf{W} \boldsymbol{\psi}_{\mathcal{B}}(\cdot) \in \mathcal{H}_{\mathbf{\Gamma}}, \\
&\approx \mathbf{P}\mathbf{W}\boldsymbol{\psi}_{\mathcal{B}}(\cdot), \\
&= \mathbf{Q}\boldsymbol{\psi}_{\mathcal{B}}(\cdot), \quad \mathbf{Q} \in \mathbb{R}^{m_{\text{chol}} \times d}, \\
&= \sum_{\ell=1}^{d} \mathbf{q}_\ell K(\mathbf{b}_\ell, \cdot), \quad \mathbf{q}_\ell \in \mathbb{R}^{m_{\text{chol}}}, \\
&= \tilde{\mathbf{h}}(\cdot).
\end{aligned}$$

In doing so, algorithm 24 can be used to sparsify an existing approximate embedding $\hat{\mathbf{h}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{m_{\text{chol}}}$. Once the new sparse set $\mathcal{B}$ has been found, it replaces the original RKHS-valued embedding's previous basis and the weights are back-fitted in the primal as described in section 5.1.1.

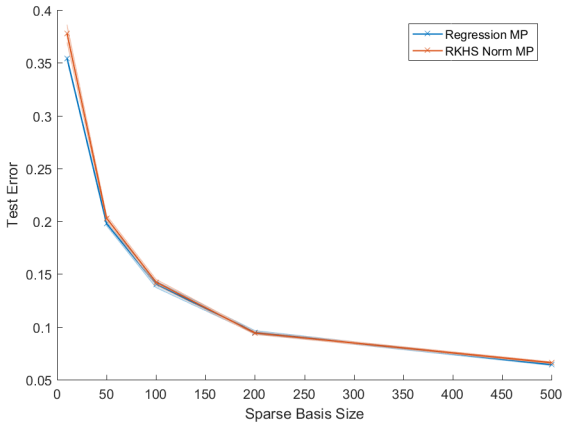
Experiments were carried out to analyse the test errors of embeddings sparsified in this way and compared to the approximate matching pursuit regression whose results are shown in fig. A.1. Trajectory data with 2000 samples for both mountain car and cart-pole was collected during policy iteration. Using approximate matching pursuit regression with backfitting, embeddings were trained on 10 independently sampled sets of 1000 samples for sparse basis sizes $|\mathcal{B}| \in \{5, 25, 50, 100, 200\}$ (mountain car) and $|\mathcal{B}| \in \{10, 50, 100, 200, 500\}$ (cart-pole). Test errors were calculated on a separate test set. Approximate matching pursuit in the RKHS norm is also carried out to sparsify existing $|\mathcal{B}| = 300$ (mountain car) and $|\mathcal{B}| = 600$ (cart-pole) embeddings to the sparse basis sizes.

Figure A.1 Approximate regression matching pursuit (Regression MP) vs. approximate RKHS norm matching pursuit (RKHS Norm MP) for sparsifying $\mathcal{B}$ in an embedding. Each method was run 10 independent times from existing trajectory data for both mountain car and cart-pole.

(a) Mountain Car



(b) Cart-Pole



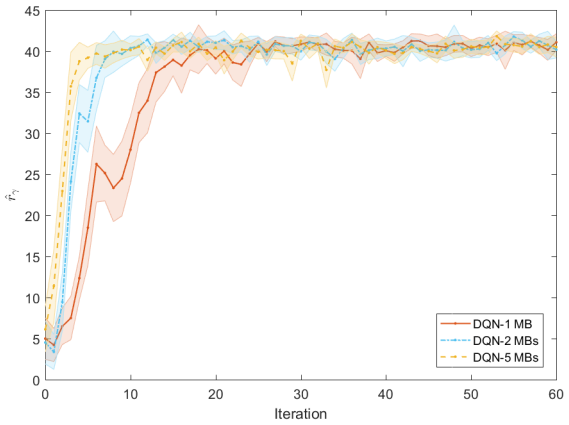
Both methods produce sparse embeddings with very similar test errors. Predictably as the sparse basis size falls, the test error increases. Further experiments are required to explore if RKHS norm sparsified embeddings can work in a CCME policy iteration algorithm. Given the out of sample errors it would seem that such an approach would work. However the disadvantage is that a non-sparse embedding has to be trained first before it is sparsified, which is avoided during approximate matching pursuit regression. Future work should also investigate the matching pursuit variants described in section A.1.3.

A.2 DQN Experiments

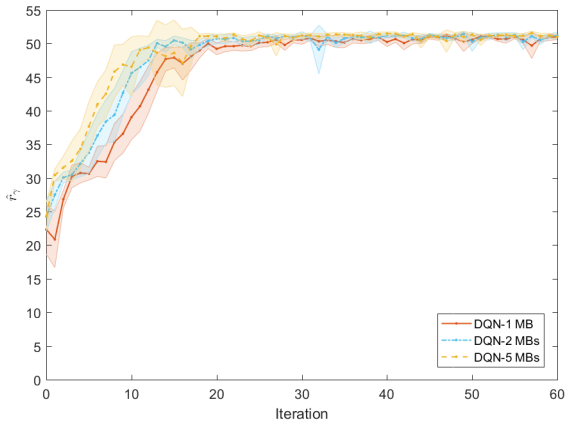
DQN experiments were performed to establish a model-free benchmark. DQN architecture as described by Mnih et al. (2015) and parameters were searched for in order to maximise performance and implemented . Like the ACCME architecture,

Figure A.2 DQN: Varying the number of minibatches per transition affects sample efficiency. Empirical discounted return is presented for each MDP.

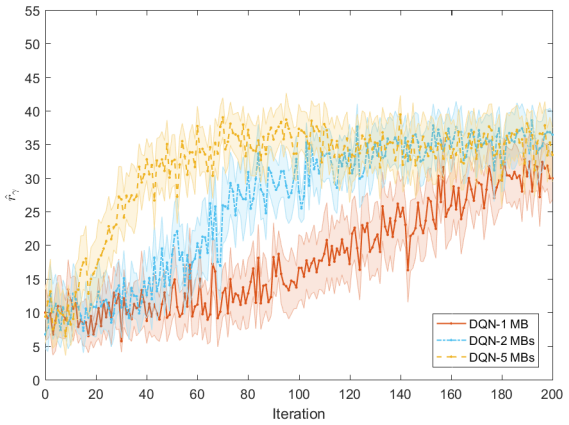
(a) Mountain car



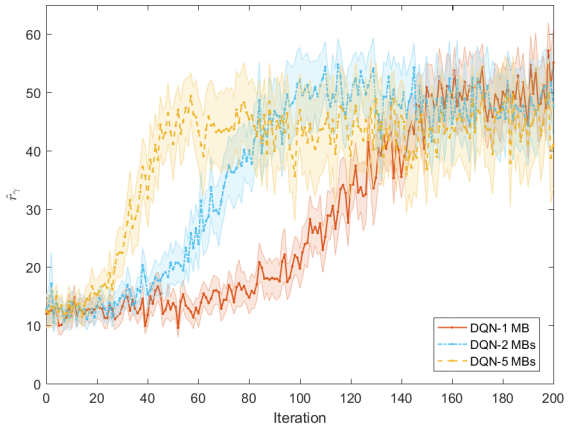
(b) Cart-pole



(c) Quadrocopter navigation



(d) Quadrocopter holding pattern



DQN is implemented as a ReLU hidden layered feed-forward (but not convolutional) architecture and trained using the Adam optimisation algorithm (with initial learning rate $\eta = 1\times10^{-3}$ with all other parameters set to default values) as described in section B.5. For mountain car and cart-pole, the hidden node counts are $\{25, 50, 100, 200, 300\}$, for the quadrocopter experiments this was increased to $\{25, 50, 100, 200, 300, 500\}$. The input layer is $\dim(\mathcal{S})$ and last layer is $|\mathcal{A}|$.

Value function updates were made online after each new sample was experienced by drawing minibatches from an experience replay memory. Crucially it was important to extract as much information out of each sample as possible such that making 5 minibatch updates per transition was optimal. Reducing the number of updates per transition drastically decreased DQN's sample efficiency and can be seen in fig. A.2. Target function approximator freezing (Mnih et al., 2015) was implemented as a function of the number of value function updates. For mountain car and cart-pole the

target function approximator was updated with the current approximator every 100 minibatches, for the quadrocopter experiments this was increased to 2000.

Appendix B

Literature Review Supplemental

B.1 Bellman Sup-Norm Contractions

Well-known results reproduced here e.g. Szepesvári (2010).

Lemma B.1 (Bellman operator contraction)**.** *Given $u, v \in \mathcal{B}(\mathcal{S})$, $\gamma \in [0, 1)$, deterministic policy π, $P^\pi v := \sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))v(\mathbf{s}')$, reward function r^π for actions defined by π, then $T^\pi v := r^\pi + \gamma P^\pi v$ is a sup-norm contraction.*

Proof.

$$\begin{aligned}
||T^\pi u - T^\pi v||_\infty &= \gamma \sup_{\mathbf{s} \in \mathcal{S}} \Big[| \sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s})) \Big(u(\mathbf{s}') - v(\mathbf{s}') \Big) | \Big], \\
&\leq \gamma \sup_{\mathbf{s} \in \mathcal{S}} \Big[\sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s})) |u(\mathbf{s}') - v(\mathbf{s}')| \Big], \\
&\leq \gamma \sup_{\mathbf{s} \in \mathcal{S}} \Big[\sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s})) ||u - v||_\infty \Big], \\
&= \gamma ||u - v||_\infty,
\end{aligned}$$

where $\sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s})) = 1$ and $|\sum_i x_i| \leq \sum_i |x_i|$ for $x_i \in \mathbb{R}$. □

Lemma B.2. $\Big| \sup_{\mathbf{a} \in \mathcal{A}} [f(\mathbf{a})] - \sup_{\mathbf{a} \in \mathcal{A}} [g(\mathbf{a})] \Big| \leq \sup_{\mathbf{a} \in \mathcal{A}} \Big[|f(\mathbf{a}) - g(\mathbf{a})| \Big]$.

Proof. Let $\mathbf{a}' = \operatorname*{argsup}_{\mathbf{a} \in \mathcal{A}} [f(\mathbf{a})]$, then

$$\begin{aligned}
\Big| \sup_{\mathbf{a} \in \mathcal{A}} [f(\mathbf{a})] - \sup_{\mathbf{a}' \in \mathcal{A}} [g(\mathbf{a}')] \Big| &< \Big| \sup_{\mathbf{a} \in \mathcal{A}} [f(\mathbf{a})] - g(\mathbf{a}') \Big| \\
&= |f(\mathbf{a}') - g(\mathbf{a}')| \\
&\leq \sup_{\mathbf{a} \in \mathcal{A}} \Big[|f(\mathbf{a}) - g(\mathbf{a})| \Big].
\end{aligned}$$

□

Lemma B.3 (Bellman optimality operator contraction)**.** *Given $u, v \in \mathcal{B}(\mathcal{S})$ and $\gamma \in [0, 1)$, then T^* is a sup-norm contraction.*

Proof.

$$
\begin{aligned}
||T^*u - T^*v||_\infty &= \sup_{\mathbf{s}\in\mathcal{S}}\Big[\Big|\sup_{\mathbf{a}\in\mathcal{A}}\Big[|\sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})u(\mathbf{s}')|\Big] - \sup_{\mathbf{a}\in\mathcal{A}}\Big[|\sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})v(\mathbf{s}')|\Big]\Big|\Big], \\
&\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}}\Big[|\sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})\Big(u(\mathbf{s}') - v(\mathbf{s}')\Big)|\Big], \\
&\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}}\Big[\sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})|u(\mathbf{s}') - v(\mathbf{s}')|\Big], \\
&\leq \gamma \sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}}\Big[\sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})||u - v||_\infty\Big], \\
&= \gamma||u - v||_\infty,
\end{aligned}
$$

where in the second line lemma B.2 is used. □

Lemma B.4 (Fixed point of the exponentially-weighted Bellman operator)**.** *Given the Bellman operator T^π and $\lambda \in [0,1)$, then v^π is the fixed point of the exponentially-weighted operator $T^\lambda : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$; $T^\lambda v := \lim_{n\to\infty}(1-\lambda)\sum_{k=1}^n \lambda^{k-1}(T^\pi)^k v$.*

Proof. The n-step weighted operator is $T^{\lambda[n]}v := (1-\lambda)\sum_{k=1}^n \lambda^{k-1}(T^\pi)^k v$ and we have $\lambda T^\pi T^{\lambda[n]}v = (1-\lambda)\sum_{k=1}^n \lambda^k (T^\pi)^{k+1} v$, then

$$
\begin{aligned}
T^{\lambda[n]}v - \lambda T^\pi T^{\lambda[n]}v &= (1-\lambda)\sum_{k=1}^{n}\Big(\lambda^{k-1}(T^\pi)^k - \lambda^k(T^\pi)^{k+1}\Big)v, \\
&= (1-\lambda)\Big(T^\pi - \lambda^n (T^\pi)^{n+1}\Big)v.
\end{aligned}
$$

Taking the limit $n \to \infty$,

$$
\begin{aligned}
T^\lambda v - \lambda T^\pi T^\lambda v &:= \lim_{n\to\infty}(1-\lambda)\Big(T^\pi - \lambda^n (T^\pi)^{n+1}\Big)v, \\
&= (1-\lambda)T^\pi v, \\
\Rightarrow T^\lambda v - \lambda r^\pi - \lambda\gamma P^\pi T^\lambda v &= T^\pi v - \lambda r^\pi - \lambda\gamma P^\pi v,
\end{aligned}
$$

then cancelling/rearranging terms and subtracting v from both sides,

$$
\begin{aligned}
T^\lambda v - \lambda\gamma P^\pi T^\lambda v + \lambda\gamma P^\pi v - v &= T^\pi v - v, \\
(I - \lambda\gamma P^\pi)\Big(T^\lambda v - v\Big) &= T^\pi v - v, \\
\Rightarrow T^\lambda v &= v + (I - \gamma\lambda P^\pi)^{-1}\Big(T^\pi v - v\Big).
\end{aligned}
$$

Setting $v = v^\pi$ and we know by definition $T^\pi v^\pi = v^\pi$, then

$$
\begin{aligned}
(T^\lambda v^\pi) &= v^\pi + (I - \gamma\lambda P^\pi)^{-1}\Big(v^\pi - v^\pi\Big), \\
&= v^\pi.
\end{aligned}
$$

□

Lemma B.5 (Online Average)**.** *For a random variable Y_{t-1} whose empirical average is $\bar{Y}_{t-1} := \frac{1}{t-1}\sum_{i=1}^{t-1} y_i$, then the online (iterative) mean after seeing sample y_t is,*

$$\bar{Y}_t = \bar{Y}_{t-1} + \frac{1}{t}(y_t - \bar{Y}_{t-1})$$

Proof.

$$\begin{aligned} \bar{Y}_t &= \frac{1}{t}\sum_{i=1}^{t} y_i \\ &= \frac{1}{t}((t-1)\bar{Y}_{t-1} + y_t) \\ \implies \bar{Y}_t &= \bar{Y}_{t-1} + \frac{1}{t}(y_t - \bar{Y}_{t-1}). \end{aligned}$$

□

B.2 Block Matrix Inversion

A block matrix inverse derived using the Schur complement is reproduced below, as detailed in (Zhang, 2005). Let

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}, \tag{B.1}$$

then assuming $\mathbf{A}$ is non-singular, the block LDU decomposition for $\mathbf{M}$ is

$$\mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{A}^{-1} & \mathbf{A} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}/\mathbf{A} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{B.2}$$

where $\mathbf{M}/\mathbf{A} := (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$ is the Schur complement of block $\mathbf{A}$. Similarly if $\mathbf{D}$ is assumed non-singular, the LDU decomposition can be written as

$$\mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{M}/\mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{pmatrix}, \tag{B.3}$$

where $\mathbf{M}/\mathbf{D} := (\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})$ is the Schur compliment of block $\mathbf{D}$. It follows that $\mathbf{M}^{-1}$ can be written by taking the inverse of either LDU decomposition respectively,

$$\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{M}/\mathbf{A})^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{A} \end{pmatrix} \tag{B.4}$$

$$= \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{pmatrix} \begin{pmatrix} (\mathbf{M}/\mathbf{D})^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{B.5}$$

where the following inversion identities were used for lower block diagonal $\mathbf{L}$, upper block diagonal $\mathbf{U}$

$$\mathbf{L} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{P} & \mathbf{I} \end{pmatrix} \Rightarrow \mathbf{L}^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{P} & \mathbf{I} \end{pmatrix}$$
$$\mathbf{U} = \begin{pmatrix} \mathbf{I} & \mathbf{Q} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \Rightarrow \mathbf{U}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{Q} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

and for any square matrices $\mathbf{V}$, $\mathbf{W}$, the inverse of their multiplication is $(\mathbf{VW})^{-1} = \mathbf{W}^{-1}\mathbf{V}^{-1}$. By multiplying out (B.4) and (B.5) respectively,

$$\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1} \\ (\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{M}/\mathbf{A})^{-1} \end{pmatrix} \tag{B.6}$$
$$= \begin{pmatrix} (\mathbf{M}/\mathbf{D})^{-1} & -(\mathbf{M}/\mathbf{D})^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{M}/\mathbf{D})^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}(\mathbf{M}/\mathbf{D})^{-1}\mathbf{B}\mathbf{D}^{-1} \end{pmatrix}. \tag{B.7}$$

If we have access to $\mathbf{A}^{-1}$ already, then equation (B.6) is used to perform the block inverse efficiently.

B.3 Some Mathematical Definitions

B.3.1 Some Measure Theory

Definitions adapted from Grimmett and Stirzaker (2001) and Rudin (1976, chapter 11).

Definition B.1 (Measurable Space)**.** *A measurable space is the ordered pair* $(\mathcal{X}, \Sigma)$ *consisting of a non-empty set* $\mathcal{X}$ *equipped with a sigma-algebra* Σ *on* $\mathcal{X}$*.* Σ *defines a collection whose elements are subsets* $E_i \subset \mathcal{X}$ *where the following hold; i)* $\emptyset \in \Sigma$ *and* $\mathcal{X} \in \Sigma$*, ii) if* $E \in \Sigma$ *then so is the complement* $E^c \in \mathcal{F}$*, iii) if* $E_i \in \Sigma$ *for every* $i \in \mathbb{N}$*, then* $\bigcup_{i=1}^{\infty} E_i \in \Sigma$*.*

Conditions ii) and iii) are known as closed under complement and closed under countable unions respectively. Elements of Σ are known as measurable subsets of $\mathcal{X}$ and are candidates for having a *size* assigned to each of them as defined by a *measure*. If $\mathcal{X} = \mathbb{R}$ (as an example of an uncountable set) then a fundamental sigma-algebra is the Borel sigma-algebra; a collection containing subsets of $\mathbb{R}$ that are the open sets defined on an interval.

Definition B.2 (Measure)**.** *Given a measurable space* $(\mathcal{X}, \Sigma)$*, then* $m : \Sigma \to \mathbb{R}^+$ *is called a measure on* $(\mathcal{X}, \Sigma)$ *which assigns a size to elements of* Σ *if i)* $m(\emptyset) = 0$ *and* $m(E) \geq 0 \ \forall E \in \Sigma$ *(non-negative), ii)* $m(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} m(E_i)$ *for any countable family* $(E_i)_i$ *of measurable disjoint sets of* $\mathcal{X}$ *(countably additive).*

Definition B.3 (Probability Measure)**.** *A probability measure is* $m : \Sigma \to [0, 1]$ *with the added condition that* $m(\Sigma) = 1$*.*

Definition B.4 (Signed Measure)**.** *A signed measure relaxes its range to include negative values such that* $m : \Sigma \to \mathbb{R}$ *and* $m(\Sigma)$ *does not necessarily need to be* 1*.*

Definition B.5 (Measure Space)**.** *A measurable space* $(\mathcal{X}, \Sigma)$ *equipped with a measure* m *is known as a measure space* $(\mathcal{X}, \Sigma, m)$*.*

B.3.2 Some Functional Analysis

Definition B.6 (Vector Space (Kreyszig, 1978, p 129))**.** *A real vector space (or linear space) over a field* $\mathbb{R}$ *is a non-empty set* $\mathcal{X}$ *of elements* $\mathbf{x}_1, \mathbf{x}_2, ...$ *together with two algebraic operations of vector addition and multiplication by scalars.*

Definition B.7 (Metric Space (Kreyszig, 1978, def. 1.1-1))**.** *A metric space is a pair* $(\mathcal{X}, d)$*; where* $\mathcal{X}$ *is a set and* d *is a metric (or distance function) on* $\mathcal{X}$*, that is* $d : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ *i.e.* d *is i) real, finite and non-negative, ii)* $d(\mathbf{x}, \mathbf{x}') = 0 \Leftrightarrow \mathbf{x} = \mathbf{x}$*, iii) symmetric and iv) obeys the triangle inequality,* $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$*.*

Definition B.8 (Normed and Banach Space (Kreyszig, 1978, def. 2.1-1))**.** *A normed space is the pair* $(\mathcal{X}, ||\cdot||)$*; a vector space whose metric is induced by a norm* $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||$*,* $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$*. The concept of a norm is motivated by quantifying element length or distance between elements and is a real-valued function obeying a specific set of properties as detailed in Kreyszig (1978, def. 2.1-1, eqns N1-4). A Banach space* $\mathcal{B} := (\mathcal{X}, ||\cdot||)$ *is a complete normed space.*

Completeness refers to a property that every Cauchy sequence $(\mathbf{x}_n)_{n \geq 1}$ of elements $\mathbf{x}_n \in \mathcal{X}$ converges to an element $\mathbf{x} \in \mathcal{X}$. A Cauchy sequence, or sequence of *diminishing oscillations* satisfies the following property (Shawe-Taylor and Cristianini, 2004, p 49),

$$\lim_{n \to \infty} \left[\sup_{m \geq n} [||\mathbf{x}_n - \mathbf{x}_m||] \right] = 0. \tag{B.8}$$

Completeness intuitively means that there are no 'missing points' or holes in a space such that every Cauchy sequence has a limit in the space itself. Certain algebraic properties that one may want to operate on a space may be unavailable if it is incomplete e.g. well-known examples include i) when working in $\mathbb{Q}$, the solution to $x^2 - 2 = 0$ does not exist and requires a completion in $\mathbb{R}$ because the solution is $\pm\sqrt{2}$, or ii) when working in $\mathbb{R}$, the solution to $x^2 + 1 = 0$ lies in $\mathbb{C}$.

Definition B.9 (Hilbert Space (Kreyszig, 1978, def. 3.1-1))**.** *A Hilbert space* $\mathcal{H} := (\mathcal{B}, \langle \cdot, \cdot \rangle)$ *is a Banach space endowed with an inner product. An inner product (Kreyszig, 1978, p 129) is a real-valued symmetric bilinear (linear in each argument) map* $\langle \cdot, \cdot \rangle$ *that satisfies* $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$*. It is also* strict *if* $\langle \mathbf{x}, \mathbf{x} \rangle = 0 \iff \mathbf{x} = 0$*. This is equivalent to an inner product (pre-Hilbert) space (Shawe-Taylor and Cristianini, 2004, p 48), complete in the metric induced by its inner product,* $||\mathbf{v}|| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$*.*

B.4 Function Approximation Review

Below is a brief review of function approximation and regression from a frequentist's perspective, briefly touching on statistical learning theory (Vapnik, 1998, 1999).

B.4.1 Learning Prediction Functions from Data

Let independent variables $\mathcal{X} = \mathbb{R}^d$ and dependent $\mathcal{Y} = \mathbb{R}$ be modelled by an unknown deterministic scalar-valued function $f_{\text{targ}} : \mathcal{X} \to \mathcal{Y}$ corrupted by noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, $Y = f_{\text{targ}}(X) + \varepsilon$ i.e. $Y \sim \mathcal{N}(f_{\text{targ}}(X), \sigma^2)$. Let data pairs be drawn from an unknown joint distribution $(X, Y) \sim p(\cdot, \cdot)$. We search for the prediction function $f \in \mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ that best generalises (out of sample) the target function f_{targ}. The hypothesis class $\mathcal{H}$ may or may not contain the target function, but the learning algorithm will not be aware of this. It is easy to see that the best hypothesis class contains only the target function i.e. $\mathcal{H} = \{f\}$ and learning would proceed by plucking the function from its hypothesis class. In reality this is not possible and instead $\mathcal{H}$ is picked and a finite data set $\mathcal{D} := \{(\mathbf{x}, y)_i\}_{i=1}^n$ is sampled i.i.d (independently and identically distributed) from $p(\mathbf{x}, y)$. By defining a loss function, a model $\hat{f}$ is trained with the goal of minimising out of sample error so that f generalises the real signal f_{targ} well but does not capture high frequency noise (also known as overfitting).

Empirical Risk Minimisation

A non-negative loss (also known as risk) function $V : \mathcal{Y} \times \mathcal{Y} \to [0, +\infty)$ penalises hypothesis prediction errors. The expected loss (also known as expected prediction error (EPE) (Hastie et al., 2001) or expected loss), is defined as

$$\begin{aligned} \mathcal{L}(V, f) :&= \mathbb{E}_{(X,Y)\sim p(\cdot,\cdot)}[V(f(X), Y)], \\ &= \mathbb{E}_{X\sim p(\cdot)}\mathbb{E}_{Y\sim p(\cdot|X)}[V(f(X), Y))]. \end{aligned} \tag{B.9}$$

By placing equal probability mass $p_i = \frac{1}{2n}$ on the i^{th} training sample (Barber, 2012) such that $p(\mathbf{x}, y) = \sum_{i=1}^{n} p_i \delta(\mathbf{x} - \mathbf{x}_i)\delta(y - y_i)$, the empirical estimate for (B.9) is

$$\begin{aligned}\hat{\mathcal{L}}(V, f, \mathcal{D}_n) &= \int_{\mathcal{X}\times\mathcal{Y}} \sum_{i=1}^{n} p_i \delta(\mathbf{x} - \mathbf{x}_i)\delta(y - y_i) V(f(\mathbf{x}), y) \mathrm{d}\mathbf{x}\mathrm{d}y, \\ &= \sum_{i=1}^{n} p_i V(f(\mathbf{x}_i), y_i) = \frac{1}{2n} \sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i), \end{aligned} \tag{B.10}$$

leading to the empirical risk minimisation (ERM) problem,

$$\hat{f} = \arg\min_{f\in\mathcal{H}} [\frac{1}{2n} \sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i)]. \tag{B.11}$$

Mean Squared Error

The loss function V is assumed to be continuous, strongly convex and smooth so that a unique solution can be found by differentiation. For function approximation the squared error (SE) loss $V_{\text{SE}}(f(X), Y) = (f(X) - Y)^2$ is a common choice such that the expected loss is

$$\mathcal{L}(V_{\text{SE}}, f) = \mathbb{E}_{X\sim p(\cdot)} \mathbb{E}_{Y\sim p(\cdot|X)} \Big[(f(X) - Y)^2 \Big], \tag{B.12}$$

whose empirical minimisation problem is

$$\hat{f} = \arg\min_{f\in\mathcal{H}} \Big[\frac{1}{2n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2 \Big]. \tag{B.13}$$

The conditional distribution component of the expected loss (B.12) makes it possible to work with $p(y|\mathbf{x})$ instead of having to deal with the unknown joint distribution $p(\mathbf{x}, y)$. The solution to the least squares expected risk is now reduced to a point-wise minimisation problem,

$$f(\mathbf{x}) = \arg\min_{c} \Big[\mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})} [(c - Y)^2] \Big].$$

Lemma B.6 (Regression Function)**.** *The minimiser of V_{SE} is the conditional mean of the target variable, $f(\mathbf{x}) = \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})}[Y]$.*

Proof. Expand the square term in the expectation

$$\begin{aligned}\mathcal{L}(V_{\text{SE}}, f \,|\, \mathbf{x}) &= \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})} \Big[(f(\mathbf{x}) - Y)^2 \Big], \\ &= \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})} \Big[f(\mathbf{x})^2 - 2f(\mathbf{x})Y + Y^2 \Big], \\ &= f(\mathbf{x})^2 - 2f(\mathbf{x}) \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})}[Y] + \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})}[Y^2]. \end{aligned}$$

By taking the derivative with respect to $f(\mathbf{x})$ (see Bishop (2006, p 46)) and setting to zero,

$$2f(\mathbf{x}) - 2\mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})}[Y] = 0,$$

clearly identifies $f(\mathbf{x}) = \mathbb{E}_{Y\sim p(\cdot|X=\mathbf{x})}[Y]$ as the minimiser for the expected square loss[1] also known as the *regression function*. □

Linear OLS Regression

How does ordinary least squares (OLS) linear regression fit into this point-wise EPE minimisation? Following Hastie et al. (2001, p 19), by assuming a linear approximation scheme $\mathcal{H} := \{f(\mathbf{x}) = \mathbf{x}^\top\boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathbb{R}^d\}$, then the solution $\boldsymbol{\beta}$ can be solved theoretically by substituting $f(X)=X^\top\boldsymbol{\beta}$ (where X is a random variable) back into equation (B.12), taking the derivative w.r.t $\boldsymbol{\beta}$ and setting to zero,

$$\begin{aligned}\mathbb{E}_{X,Y\sim p(\cdot,\cdot)}[XX^\top\boldsymbol{\beta} - XY] &= 0,\\ \Rightarrow \boldsymbol{\beta} &= \mathbb{E}_{X\sim p(\cdot)}[XX^\top]^{-1}\mathbb{E}_{(X,Y)\sim p(\cdot,\cdot)}[XY], \qquad \text{(B.14)}\\ &= \mathrm{var}(X)^{-1}\mathrm{cov}(X,Y).\end{aligned}$$

The empirical least squares estimate $\hat{\boldsymbol{\beta}}$ can be calculated by replacing the expectations in (B.14) by sample averages over the training data. This is equivalent to accumulating samples in matrix $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]^\top \in \mathbb{R}^{n\times d}$ (where implicitly a bias column is included as $\mathbf{x}_0 = \mathbf{1}$) and vector $\mathbf{y} \in [y_1, ..., y_n]^\top \in \mathbb{R}^n$; then the least squares minimiser $\hat{\boldsymbol{\beta}}_{\text{ols}} \in \mathbb{R}^d$ of (B.13) is

$$\begin{aligned}0 = \nabla_f\Big(\frac{1}{2n}\sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i)\Big) &= \nabla_\beta\Big[\frac{1}{2n}||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||_2^2\Big],\\ &= -\frac{1}{n}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}), \qquad \text{(B.15)}\\ \Rightarrow \hat{\boldsymbol{\beta}}_{\text{ols}} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}. \qquad \text{(B.16)}\end{aligned}$$

Note that 'mean squared error' (MSE) usually refers to the empirical average 'residual sum of squares' (RSS), which is the sum of the *residuals* (or *realised* errors) of observations under the assumed statistical model. The residual vectors $\mathbf{e} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$ are assumed to belong to a Hilbert space $\mathbb{R}^n$ endowed with an inner product induced by its norm $||\cdot||_2^2 = \langle\cdot,\cdot\rangle_2$, such that $RSS := ||\mathbf{e}||_2^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$.

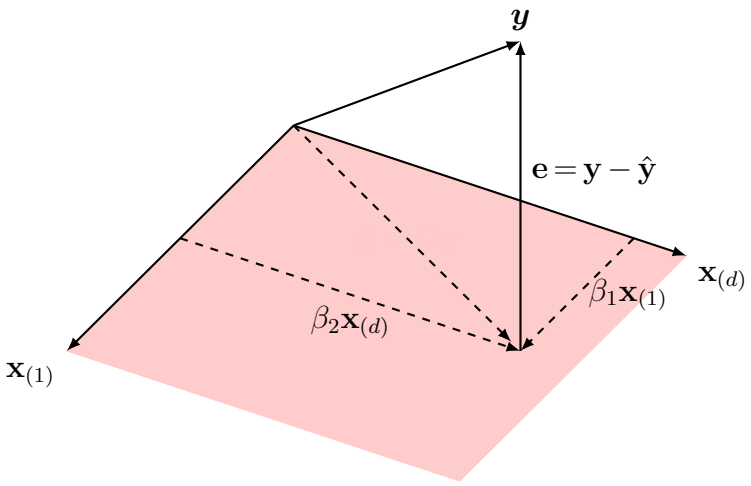
The set of columns of the design matrix $\mathbf{X} = [\mathbf{x}_{(1)}, .., \mathbf{x}_{(d)}]$ are considered to form a column space of $\mathbf{X}$ labelled span($\mathbf{X}$). The OLS solution minimises the RSS by choosing the shortest vector $\mathbf{y} - \hat{\mathbf{y}}$ to span($\mathbf{X}$) which is the normal vector and geometrically is

[1] Note that if $V(Y, f(X)) = |f(X) - Y|$, then the optimal solution is $f(\mathbf{x}) = \text{median}_{Y\sim p(\cdot|X=\mathbf{x})}[Y]$ where V is more robust to outliers, however it lacks smoothness of the squared loss.

an orthogonal projection $\mathbf{\Pi} : \mathbb{R}^n \to \mathbb{R}^d$ (see figure B.1),

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\hat{\boldsymbol{\beta}}_{\text{ols}}, \\ &= \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{\Pi}\mathbf{y}, \\ &= \sum_{i=1}^{d} \beta_i \mathbf{x}_{(i)}. \end{aligned} \tag{B.17}$$

Figure B.1 Geometric interpretation of regression and Π.



Theorem B.4.1 (Orthogonal Complement (Kreyszig, 1978, thm 3.3-4))**.** *Given a Hilbert space $\mathcal{H}$ and a closed subspace $\hat{\mathcal{H}} \subseteq \mathcal{H}$ with elements $\hat{h}$, then the orthogonal complement $\hat{\mathcal{H}}^\perp$ of $\hat{\mathcal{H}}$ is*

$$\hat{\mathcal{H}}^\perp := \{h \in \mathcal{H} \mid \langle h, \hat{h} \rangle_{\mathcal{H}} = 0, \forall \hat{h} \in \hat{\mathcal{H}}\},$$

then the orthogonal decomposition is the internal direct sum $\mathcal{H} = \hat{\mathcal{H}} \oplus \hat{\mathcal{H}}^\perp$.

The OLS analysis makes the decomposition $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{e}$ such that $\langle \hat{\mathbf{y}}, \mathbf{e} \rangle_2 = 0$.

Unique Solution

For a unique solution $\boldsymbol{\beta}_{\text{ols}}$ to exist depends on the properties of the design matrix $\mathbf{X}$ and its corresponding non-centred sample covariance matrix $\mathbf{X}^\top\mathbf{X}$ as outlined below. Equation (B.17) states that the predicted $\hat{\mathbf{y}}$ lies within the basis defined by the columns of $\mathbf{X}$.

Definition B.10 (Rank (Lang, 1987, p 113))**.** *Let $\mathbf{M} \in \mathbb{R}^{m \times n}$ then $M = [\mathbf{c}_1, ..., \mathbf{c}_n]$ are columns that generate a subspace whose dimension is called the column rank of $\mathbf{M}$. Similarly the rows $\mathbf{M} = [\mathbf{r}_1, ..., \mathbf{r}_m]^\top$ form a subspace whose dimension is called the row*

rank. The column (row) rank is the maximum number of linearly independent columns (rows) of $\mathbf{M}$.

Following Hastie et al. (2001, p 64) and Lang (1987, ch 8), the singular value decomposition (SVD) of the design matrix is defined $\mathbf{X}=\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top \in \mathbb{R}^{n\times d}$ where $\mathbf{U}\in\mathbb{R}^{n\times n}$ and $\mathbf{V}\in\mathbb{R}^{d\times d}$ are orthogonal matrices such that $\mathbf{U}^\top\mathbf{U}=\mathbf{I}_n$ and $\mathbf{V}^\top\mathbf{V}=\mathbf{I}_d$. The columns of $\mathbf{U}$ and $\mathbf{V}$ span the column space and row space of $\mathbf{X}$ respectively. $\boldsymbol{\Sigma}\in\mathbb{R}^{n\times d}$ is a rectangular diagonal matrix whose diagonal values are the *singular values* $\sigma_i\in\mathbb{R}^+$ of decreasing order for $i=1,...,\min(n,d)$.

If there are linearly *dependent* columns in the design matrix $\mathbf{X}\in\mathbb{R}^{n\times d}$ then $\text{Rank}(\mathbf{X})<n$. This means that at least one basis vector (explanatory variable or column $\mathbf{x}_{(i)}$ in equation (B.17)) of the subspace spanned by $\text{col}(\mathbf{X})$ is a linear combination of another basis vector $\mathbf{x}_{(j)}$. Such column rank deficiencies can occur by i) exact multicollinearity where two or more columns are linear combinations of each other, ii) numerical multicollinearity where columns are highly correlated at machine precision or iii) the regression problem is underdetermined where dimensionality of the data exceeds the number of samples collected $d>n$. Rank deficiency implies that the column basis is insufficient to span the space that the raw data resides in.

The eigen-decomposition (Lang, 1987, ch 8, §1) and (Shawe-Taylor and Cristianini, 2004, chapter 3) of a square real symmetric matrix is $\mathbf{A}=\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$ where orthonormal eigenvectors are collected into columns $\mathbf{V}:=[\mathbf{v}_1,...,\mathbf{v}_d]$ and eigenvalues are arranged by $\boldsymbol{\Lambda}=\text{diag}(\lambda_1,...,\lambda_d)$. The inverse is therefore $\mathbf{A}^{-1}=\mathbf{V}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\top$. Comparing this with the SVD of the uncentered sample covariance matrix can is therefore decomposed

$$\begin{aligned}\mathbf{X}^\top\mathbf{X} &= \mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top,\\ &= \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^\top,\\ &= \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top,\end{aligned} \tag{B.18}$$

then $\boldsymbol{\Lambda}^{-1}=\boldsymbol{\Sigma}^{-2}$ such that $\text{diag}(\lambda_1^{-1},...,\lambda_d^{-1})=\text{diag}(\sigma_1^{-2},...,\sigma_d^{-2})$. If any singular values (and therefore eigenvalues) are zero then $\mathbf{X}^\top\mathbf{X}$ is known as a *singular* matrix. Conversely if all singular values (and therefore all eigenvalues) are non-zero then $\mathbf{X}^\top\mathbf{X}$ is *non-singular*. The number of non-zero singular values is equal to $\text{rank}(\mathbf{X})$. Then if $n<d$, the covariance matrix $\mathbf{X}^\top\mathbf{X}$ has at least one zero eigenvalue or singular value, rendering it a *singular* matrix that is not invertible. The solution $\hat{\boldsymbol{\beta}}$ is therefore not uniquely defined such that there is more than one way to project $\hat{\mathbf{y}}$ onto the column space of $\mathbf{X}$. Symmetric positive semi-definite matrices have non-negative eigenvalues and symmetric strictly positive definite matrices have positive eigenvalues (Shawe-Taylor and Cristianini, 2004, p 57, thm. 3.6). By the eigen-decomposition of $\mathbf{X}^\top\mathbf{X}$, it is easy to see that an inverse does not exist if any of the eigenvalues are zero. It is therefore possible to see if there is a least squares solution by checking if

the design matrix $\mathbf{X}$ is full column rank. Given equation (B.17), then full rank is achieved if $\mathbf{X}$ has linearly independent columns. The same properties of the design matrix directly relate to the convexity properties of the loss function (Murphy, 2012, section 7.3.3). Adding a diagonal (ridge term) constant to a matrix mitigates zero eigenvalues, making it invertible (see regularisation below).

Bias-Variance Decomposition of the Expected Loss

The expected loss can be decomposed into different sources of error using the bias-variance decomposition (see Hastie et al. (2001, p.223) or Murphy (2012, p.202)). The decomposition describes how the sources of error change depending on the richness of the hypothesis class $\mathcal{H}$ from which functions are fitted under a fixed training set. The same analysis motivates how to pick functions that best fit out-of-sample data by mitigating *overfitting*. For clarity (whose steps are usually omitted in the literature) the decomposition will be reproduced here.

Lemma B.7 (Bias-variance decomposition)**.** *Given a fixed hypothesis set $\mathcal{H}$, data model $Y = f_{\text{targ}}(X) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ and a budget of n training samples drawn (as a random variable) from the joint distribution $\mathcal{D}_n := \{(X,Y)_i\}_{i=1}^n \sim p(X,Y)$, then the expected out-of-sample error $\mathcal{L}_{out}$ is decomposed into* $\text{bias}^2[h(X)] + \text{var}_{\mathcal{D}}[h^{(\mathcal{D}_n)}(X)] + \sigma^2$.

Proof. Let the out of sample expected error, averaged over training sets be

$$\begin{aligned}\mathcal{L}_{\text{out}}(V_{\text{SE}}, h) &= \mathbb{E}_{\mathcal{D}_n}\Big[\mathbb{E}_{X\sim p(\cdot)}\Big[\mathbb{E}_{Y\sim p(\cdot|X)}[(h^{(\mathcal{D}_n)}(X) - Y)^2]\Big]\Big], \\ &= \mathbb{E}_{X\sim p(\cdot)}\Big[\mathbb{E}_{\mathcal{D}_n}\Big[\mathbb{E}_{Y\sim p(\cdot|X)}[(h^{(\mathcal{D}_n)}(X) - Y)^2]\Big]\Big], \end{aligned} \tag{B.19}$$

where $h^{(\mathcal{D}_n)} \in \mathcal{H}$ is the hypothesis from $\mathcal{H}$ created (i.e. selected from $\mathcal{H}$) by being trained on the finite training set $\mathcal{D}_n$. The inner most term is

$$\begin{aligned}&\mathbb{E}_{Y\sim p(\cdot|X)}\Big[\Big(h^{(\mathcal{D}_n)}(X) - Y\Big)^2\Big] \\ &\quad= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\sigma^2)}\Big[\Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X) - \varepsilon\Big)^2\Big], \\ &\quad= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\sigma^2)}\Big[\Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X)\Big)^2 - 2\varepsilon\Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X)\Big) + \varepsilon^2\Big], \\ &\quad= \Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X)\Big)^2 + \sigma^2, \end{aligned} \tag{B.20}$$

where in the last line $\mathbb{E}_\varepsilon[\varepsilon] = 0$ and $\mathbb{E}_\varepsilon[\varepsilon^2] = \text{var}(\varepsilon) + \mathbb{E}_\varepsilon[\varepsilon]^2 = \text{var}(\varepsilon) = \sigma^2$. Substituting (B.20) into (B.19),

$$\mathcal{L}_{\text{out}}(V_{\text{SE}}, h) = \mathbb{E}_{X\sim p(\cdot)}\Big[\mathbb{E}_{\mathcal{D}_n}\Big[\Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X)\Big)^2\Big] + \sigma^2\Big]. \tag{B.21}$$

The *average* hypothesis over all training sets of size n is $\bar{h}(X) := \mathbb{E}_{\mathcal{D}_n}[h^{(\mathcal{D}_n)}(X)] = \lim_{N\to\infty} \frac{1}{N}\sum_{j=1}^N h^{\mathcal{D}_n^{(j)}}(X)$ where $\{\mathcal{D}_n^{(1)}, ..., \mathcal{D}_n^{(N)}\}$ is a collection of N unique datasets

all with n samples. Introducing $\bar{h}$ into the inner term of equation (B.21),

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}_n}\Big[\Big(h^{(\mathcal{D}_n)}(X) - f_{\text{targ}}(X)\Big)^2\Big] \\
&\quad = \mathbb{E}_{\mathcal{D}_n}\Big[\Big(\bar{h}(X) - f_{\text{targ}}(X) + h^{(\mathcal{D}_n)}(X) - \bar{h}(X)\Big)^2\Big], \\
&\quad = \mathbb{E}_{\mathcal{D}_n}\Big[\Big(\bar{h}(X) - f_{\text{targ}}(X)\Big)^2 + \Big(h^{(\mathcal{D}_n)}(X) - \bar{h}(X)\Big)^2\Big], \\
&\quad = \Big(\bar{h}(X) - f_{\text{targ}}(X)\Big)^2 + \mathbb{E}_{\mathcal{D}_n}\Big[\Big(h^{(\mathcal{D}_n)}(X) - \bar{h}(X)\Big)^2\Big], \\
&\quad = \text{bias}^2\Big[h(X)\Big] + \text{var}_{\mathcal{D}_n}\Big[h^{(\mathcal{D}_n)}(X)\Big], \qquad \text{(B.22)}
\end{aligned}$$

where in line three the cross-terms are zeroed out due to the expectation. The average hypothesis $\bar{h}$ can be considered as the *best* hypothesis given the chosen $\mathcal{H}$ and budget n, but in all practical settings it is not accessible and used only as a tool to express the bias-variance trade-off. Substituting (B.22) into (B.21),

$$\mathcal{L}_{\text{out}}(V_{\text{SE}}, h) = \mathbb{E}_{X\sim p(\cdot)}\left[\text{bias}^2\Big[h(X)\Big] + \text{var}_{\mathcal{D}_n}\Big[h^{(\mathcal{D}_n)}(X)\Big]\right] + \sigma^2. \qquad \text{(B.23)}$$

□

Theorem B.4.2 (Gauss Markov Theorem (Graybill, 1961))**.** *Assuming the unobserved errors i) satisfy* $\mathbb{E}[\varepsilon] = 0$*, ii) they are homoscedastic across all* $\mathcal{X}$*, iii)* $var[\varepsilon] = \sigma^2 < \infty$ *and uncorrelated across* $\mathcal{X}$*, then the linear least squares estimate* $h(\mathbf{x}) = \langle \mathbf{x}, \hat{\boldsymbol{\beta}}_{ols} \rangle_2$ *is an unbiased estimator and has the lowest variance for a linear estimator.*

Bias-Variance Trade-Off and Overfitting

A good function approximator that adequately captures the signal from f_{targ} will have a low out of sample error $\mathcal{L}_{\text{out}}$, therefore how does one achieve this in practice? Given a budget of n samples per training set, a practitioner only has the freedom to choose the hypothesis class $\mathcal{H}$. The bias-variance decomposition (B.23) tells us how the choice $\mathcal{H}$ affects $\mathcal{L}_{\text{out}}$ by decomposing it into three sources of error. Two of these are sources of *noise*; i) *stochastic* noise generated from ε and ii) *deterministic* noise represented by the expected bias term (over $\mathcal{X}$) which is the signal generated by the target function that the chosen $\mathcal{H}$ cannot capture. Stochastic noise cannot be minimised, but deterministic noise can be reduced if $f$targ is included in $\mathcal{H}$ which can be achieved by enriching $\mathcal{H}$.

A final third source of error is due to the term $\text{var}_{\mathcal{D}_n}[h^{(\mathcal{D}_n)}(X)]$ which represents *overfitting*. Data is the only resource that helps guide the training process to an h which is hopefully close to the average hypothesis $\bar{h}$ (achievable given $\mathcal{H}$). A high variance term is typically induced by a rich $\mathcal{H}$ being deployed in a training process when n is limited. In this scenario, the evaluation of a trained $h^{(\mathcal{D}_n)}$ for any instance of training data $\mathcal{D}_n$ will frequently lie far from the average hypothesis' evaluation i.e. the

limited training data available is not good enough to *consistently* guide the learning algorithm through the rich $\mathcal{H}$ to a good hypothesis close to $\bar{h}$. This effect can be seen as the hypothesis capturing the high frequency ε noise. In the limit of infinite data the variance term will tend to zero as there is adequate data resource to *consistently* select a good hypothesis.

Increasing n decreases overfitting, increasing $\mathcal{H}$ decreases bias however a too complex $\mathcal{H}$ (when n is a finite budget) will induce overfitting. Trading an increase of bias for a decrease in variance (B.23) can be better understood by an additional bias decomposition (Hastie et al., 2001, p 224-226). The best unconstrained hypothesis can be thought of as trained over the entire domain $\mathcal{X}$ and whose targets are the actual target function, $h^* := \arg\min_{h \in \mathcal{H}} \Big[\mathbb{E}_X[(f_{\text{targ}}(X) - h(X))^2]\Big]$ into the bias term,

$$\begin{aligned}
\mathbb{E}_{X \sim p(\cdot)}\Big[\Big(\bar{h}(X) - f_{\text{targ}}(X)\Big)^2\Big] &= \mathbb{E}_X\Big[\Big(\bar{h}(X) - h^* + h^* - f_{\text{targ}}(X)\Big)^2\Big], \\
&= \mathbb{E}_X\Big[\Big(\bar{h}(X) - h^*\Big)^2\Big] + \mathbb{E}_X\Big[\Big(h^* - f_{\text{targ}}(X)\Big)^2\Big], \\
&= \mathbb{E}_X\Big[\text{estimationBias}^2[X]\Big] + \mathbb{E}_X\Big[\text{modelBias}^2[X]\Big].
\end{aligned} \tag{B.24}$$

More precisely in the context of the bias-variance trade-off, if $\mathcal{H}$ is an unconstrained set of ordinary least squares linear approximators, then the *estimation bias* in (B.24) is zero as stated by the Gauss-Markov theorem B.4.2. It is possible to trade off an increase in *estimation bias* for a reduction in variance by constraining $\mathcal{H}$. If the reduction in variance is greater than the increase in estimation bias then the out-of-sample error is lower, which is known as the 'bias-variance' trade-off. *Model bias* can only be reduced by enriching $\mathcal{H}$.

Practitioners do not have access to the out-of-sample error because any data made available will always make it into the training set. Validation error is therefore used in a cross-validation scheme to search for the constrained $\mathcal{H}$ that strikes the best bias-variance trade-off.

Penalised Empirical Risk Minimisation

Restricting the hypothesis space $\mathcal{H}$ is made possible by the constrained optimisation problem,

$$\begin{aligned}
\underset{f}{\text{minimise}} \quad & \frac{1}{2n}\sum_{i=1}^{n} V(f(\mathbf{x}_i) - y_i) \\
\text{subject to} \quad & g(f) \leq c
\end{aligned}$$

which has a one-to-one correspondence (Hastie et al., 2015, sec 2.2) with the regularised risk minimisation problem

$$\hat{f} = \arg\min_{f \in \mathcal{H}} [\frac{1}{2n} \sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i) + \lambda g(f)], \tag{B.26}$$

where $g(\cdot)$ is a convex penalty function, shrinkage parameter is $\lambda \geq 0$ and V is a convex loss function. Ridge regression (also known as Tikonov regularisation) penalises ordinary least squares regressors (where V is the squared loss) with the L2 norm $g(f) := \frac{1}{2}||\boldsymbol{\beta}||_2^2$ (Hastie et al., 2001, p 61) such that the restricted hypothesis set to search through is $\mathcal{H} := \{f(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathbb{R}^d, \ ||\boldsymbol{\beta}||_2^2 \leq 2c\}$[2].

The ERM problem for regularised ridge regression is solved by setting $f = \boldsymbol{\beta} \in \mathbb{R}^d$, $V = V_{\text{SE}}$, $g(f) = \lambda||\boldsymbol{\beta}||_2^2$ which is both convex and smooth. Therefore $\boldsymbol{\beta}$ can be solved in closed form,

$$0 = \nabla_f \Big(\frac{1}{2n} \sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i) + g(f) \Big) = \nabla_\beta \Big[\frac{1}{2n} ||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\beta}||_2^2 \Big], \tag{B.27}$$

$$= -\frac{1}{n} \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}, \tag{B.28}$$

$$\Rightarrow \hat{\boldsymbol{\beta}}_{\text{ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{y}. \tag{B.29}$$

Solution (B.29) is known as the *primal* solution (with n absorbed into λ). The *estimation bias* in equation (B.24) quantifies the distance between the achievable average hypothesis $\bar{h}$ in the *constrained* hypothesis set and the closest fit h^* in the *unconstrained* hypothesis set. The gradient of *decreasing* penalised loss (equation (B.28)) has the term $-\lambda\boldsymbol{\beta}$ (excluding the bias term β_0) from the regulariser which is clearly a shrinkage operation. Informally the addition of the L2 regulariser prevents elements of $\boldsymbol{\beta}$ growing uncontrollably (Bishop, 2006, p 11) and leads to smoother regressors that are less-susceptible to capturing high frequency noise. The *regularised* empirical loss proves to be a better proxy to the out-of-sample error and leads to better hypothesis generalisation.

Another perspective is to observe a convex loss surface for OLS where the minimum is not obvious, such as for a long, almost level and very flat loss surface valley. In such a case, many OLS candidate solutions would have a very similar loss value. By adding an additional convex penalty (such as the L2 norm) modifies the loss surface such that a well-defined minimum exists. Following (Hastie et al., 2001, p 66), the SVD decomposition for OLS and ridge regression is,

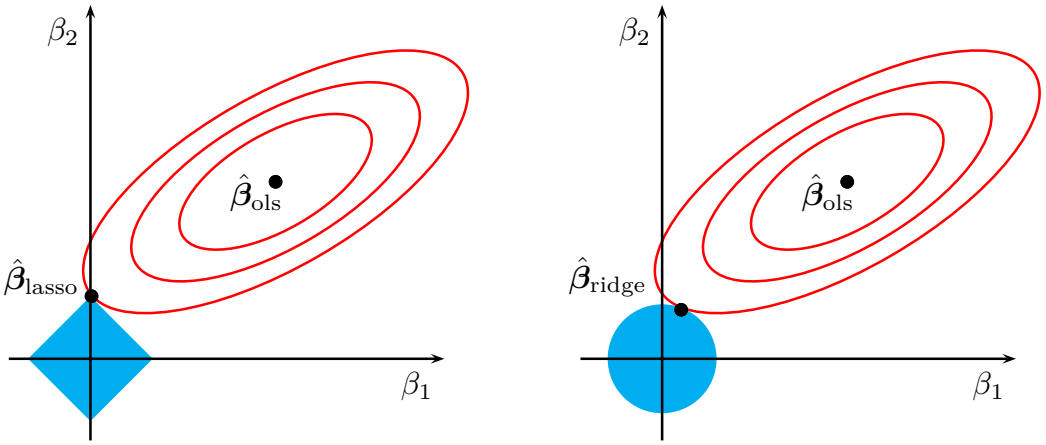
[2]For brevity the notation doesn't explicitly include bias terms and should be assumed depending on context. All function evaluations include the bias term, however regularisers exclude acting on bias terms during penalisation.

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\hat{\boldsymbol{\beta}}_{\text{ols}}, \\ &= \mathbf{U}\boldsymbol{\Sigma}(\boldsymbol{\Sigma}^2)^{-1}\boldsymbol{\Sigma}\mathbf{U}^\top\mathbf{y} \\ &= \mathbf{U}\mathbf{U}^\top\mathbf{y} \\ &= \sum_{i=1}^{d}\mathbf{u}_i\mathbf{u}_i^\top\mathbf{y}. \end{aligned} \tag{B.30}$$

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\hat{\boldsymbol{\beta}}_{\text{ridge}}, \\ &= \mathbf{U}\boldsymbol{\Sigma}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}\mathbf{U}^\top\mathbf{y} \\ &= \sum_{i=1}^{d}\mathbf{u}_i\frac{\sigma_i^2}{\sigma_i^2+\lambda}\mathbf{u}_i^\top\mathbf{y}, \end{aligned} \tag{B.31}$$

where the columns in $\mathbf{U} := [\mathbf{u}_1, ..., \mathbf{u}_d] \in \mathbb{R}^{n\times d}$ form an orthogonal basis. Both equations (B.30) and (B.31) take the form $\sum_{i=1}^{d}\mathbf{u}_i\tilde{\beta}_i$ where $\tilde{\beta}_i$ weights the coordinates of $\hat{\mathbf{y}}$ in span($\mathbf{U}$). The ridge estimate re-scales this weight by shrinking those $\tilde{\beta}_i$ with the smallest σ_i the most. Hastie et al. (2001, p 66) argue that in this way ridge-regression shrinks the coefficients for low variance directions (whose singular terms σ_i are smallest) in the data $\mathbf{X}$ more than large variance directions, effectively reducing the effective degrees of freedom (Hastie et al., 2001, p 68) which is normally d+1 free parameters (which includes the bias term) for ordinary least squares. Informally this leads to smoother function approximators.

Figure B.2 L1 (LASSO) and L2 (ridge regression) regularisation for $\boldsymbol{\beta} \in \mathbb{R}^2$.



Choice of regularisation norms lead to different effects on $\boldsymbol{\beta}$ during the optimisation procedure. By choosing $g(f) := ||\boldsymbol{\beta}||_1$ (which is the L1-norm) leads to lasso (Tibshirani, 1996) sparse regression where elements of $\boldsymbol{\beta}$ are driven to zero. However the L1 regulariser is not smooth as it possesses a discontinuity in its first derivative and therefore a LASSO-based ERM cannot be solved directly. A cannon of convex optimisation algorithms exist that deal with solving the LASSO and a good summary is made by Hastie et al. (2015, sec 5). The weight space when optimising with both L1 and L2 regularisers is visualised in fig. B.2, noting that the 'angular' geometry of the L1-norm induces sparsity in the weight vector.

B.4.2 Reproducing Kernel Hilbert Spaces of Functions

The following summary of RKHS theory takes elements from Shawe-Taylor and Cristianini (2004, chapter 3), Steinwart and Christmann (2008, chapter 4) and Sejdinovic and Gretton (2012). We briefly re-visit penalised risk minimisation for fitting linear functions. Given a Hilbert space of scalar-valued functions $\mathcal{F} \subset \mathbb{R}^{\mathcal{X}}$ whose function evaluation is $f(\mathbf{x}) = \langle f, \boldsymbol{\phi}(\mathbf{x}) \rangle_{\mathcal{F}}$ with a feature representation $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{F}$ then the penalised empirical risk problem is

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \Big[\sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i) + g(||f||_{\mathcal{H}}) \Big], \tag{B.32}$$

with loss function $V : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R} \cup \{\infty\}$ and a strictly monotonically increasing real-valued function $g(||\cdot||_{\mathcal{H}}) : [0, \infty) \to \mathbb{R}$ that restricts the hypothesis space $\mathcal{H} \subseteq \mathcal{F}$. Choosing V as the squared loss and $g(||\cdot||_{\mathcal{H}}) = \frac{\lambda}{2}||\cdot||_2^2$ then the penalised ERM (B.32) becomes the ridge regression problem that is solved by (B.29) with a substitution $\mathbf{X} = \boldsymbol{\Phi}$ and $\boldsymbol{\Phi} := [\boldsymbol{\phi}(\mathbf{x}_1), .., \boldsymbol{\phi}(\mathbf{x}_n)]^\top$. The introduction of the feature representation identifies $\mathcal{F}$ as the *feature space* in which function evaluation is *linear* and provides a mechanism to model non-linear target functions f_{targ}.

Primal and Dual Representation

Ridge regression PERM (B.27) has already been solved by the *primal* solution (B.29) whose feature-based version is

$$0 = \nabla_f \Big(\frac{1}{2n} \sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i) + g(f) \Big) = \nabla_\beta \Big[\frac{1}{2n} ||\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\beta}||_2^2 \Big], \tag{B.33}$$

$$= -\frac{1}{n} \boldsymbol{\Phi}^\top (\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}, \tag{B.34}$$

$$\Rightarrow \hat{\boldsymbol{\beta}}_{\text{ridge}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \mathbf{I}_{d'})^{-1} \boldsymbol{\Phi}^\top \mathbf{y}, \tag{B.35}$$

where $\dim(\mathcal{F}) = d'$ is the dimensions of the feature space. The regression solution is computed with a cost that scale cubically with d' which can be impractical if very rich representation is required or impossible for infinite dimensional $\mathcal{F}$. Parametric regression requires either $\boldsymbol{\phi}$ to be provided a priori or learnt before $\boldsymbol{\beta}_{\text{ridge}}$ is calculated.

Following Shawe-Taylor and Cristianini (2004, p 31, sec. 2.2.2) (see also Hastie et al. (2001, section 12.3.7)), by making the observation that equation (B.34) contains a term that is a linear combination of feature vectors, then substituting $\boldsymbol{\alpha} := \frac{1}{n\lambda}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})$ into equation (B.34) gives

$$0 = -\lambda \boldsymbol{\Phi}^\top \boldsymbol{\alpha} + \lambda \boldsymbol{\beta},$$

$$\Rightarrow \boldsymbol{\beta} = \boldsymbol{\Phi}^\top \boldsymbol{\alpha} = \sum_{j=1}^{n} \alpha_j \boldsymbol{\phi}(\mathbf{x}_j). \tag{B.36}$$

This specifies that the solution of the PERM problem to be a weighted sum of features over each training point. Substituting equation (B.36) into the new definition for $\boldsymbol{\alpha}$,

$$\begin{aligned} \boldsymbol{\alpha} &= \frac{1}{n\lambda}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \boldsymbol{\alpha}), \\ \Rightarrow \boldsymbol{\alpha} &= (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \lambda \mathbf{I}_n)^{-1}\mathbf{y}, \\ &= (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \end{aligned} \tag{B.37}$$

where $\mathbf{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ and $\mathbf{K}_{ij} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}}$. This is known as the *dual* solution to the PERM problem which can now be solved without searching for $\boldsymbol{\beta}$ (which may exist in an infinite feature space), and instead search for a vector $\boldsymbol{\alpha}$ whose size is invariant to the feature representation which scales linearly with the size of the training set,

$$0 = \nabla_{\boldsymbol{\alpha}} \Big(\sum_{i=1}^{n} V((\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \boldsymbol{\alpha})_i, y_i) + g(\boldsymbol{\Phi}^\top \boldsymbol{\alpha}) \Big) = \nabla_{\boldsymbol{\alpha}} \Big[\frac{1}{2n} ||\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \boldsymbol{\alpha}||_2^2 + \lambda ||\boldsymbol{\Phi}^\top \boldsymbol{\alpha}||_{\mathcal{H}}^2 \Big], \tag{B.38}$$

$$= -\frac{1}{n}(\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}) + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha},$$

whose solution is equation (B.37). Solving the penalised ERM problem in the dual therefore requires and inversion of the kernel matrix which has a computational cost that scales *cubically* with the number of data points n. Reducing a possibly infinite dimensional search problem in $\mathcal{F}$ to a problem whose solution scales only with the number of training samples is an example of the *representer theorem*. Elaboration on this will be delayed until further formalisation of the dual solution is made. Both the the primal and dual solutions can now be compared,

$$\hat{f}(\mathbf{x}) = \boldsymbol{\phi}^\top(\mathbf{x})\hat{\boldsymbol{\beta}} \tag{B.39}$$

$$= \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\Phi}^\top \hat{\boldsymbol{\alpha}} \rangle_{\mathcal{F}} = \langle \boldsymbol{\phi}(\mathbf{x}), \sum_{j=1}^{n} \hat{\alpha}_j \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}} = \sum_{j=1}^{n} \hat{\alpha}_j \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}} = \sum_{j=1}^{n} \hat{\alpha}_j K(\mathbf{x}_j, \mathbf{x}),$$

$$= \mathbf{k}_{\mathbf{x}}^\top \hat{\boldsymbol{\alpha}}, \tag{B.40}$$

where $\mathbf{k}_{\mathbf{x}} := [K(\mathbf{x}, \mathbf{x}_1), ..., K(\mathbf{x}, \mathbf{x}_n)]^\top$ and K is a positive definite *kernel function* representing inner products in feature space. Hilbert space dual solutions, definition of kernels and their relationship is formalised by reproducing kernel Hilbert spaces (RKHS) of functions in the following section.

Reproducing Kernel Hilbert Spaces

In the previous section it was shown that function evaluation is tantamount to evaluating inner products of points in feature space. An RKHS is a non-parametric function class (because its solution grows with the number of data points) which assumes that functions exist in a Hilbert space endowed with additional structure related to the

smoothness of evaluation and function similarity. Function evaluation of this powerful function class replaces explicit inner products in $\mathcal{F}$ with *kernel* evaluations, implicitly representing inner products in potentially an infinite dimensional feature space.

Figure B.3 Equivalent concepts in RKHS theory.

$$\textbf{RKHS of functions}, \mathcal{H}_K \text{ (definition B.11):} \tag{B.41}$$
$$\text{Linear function evaluation operators } \delta_{\mathbf{x}} f = f(\mathbf{x}) \text{ are bounded,}$$
$$\text{where } \mathbf{x} \in \mathcal{X},\ f : \mathcal{X} \to \mathbb{R},\ f \in \mathcal{H}_K,\ \delta_{\mathbf{x}} : \mathcal{H}_K \to \mathbb{R},\ \delta_{\mathbf{x}} \in \mathcal{H}_K^* .$$
$$\Updownarrow \text{ (Theorem B.4.4)}$$
$$\textbf{Reproducing Kernel/Property} \text{ (definition B.12):} \tag{B.42}$$
$$\text{For an } \mathbb{R}\text{-Hilbert space } \mathcal{H}, \text{ function } K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \text{ is a reproducing kernel,}$$
$$\text{if } K(\cdot, \mathbf{x}) \in \mathcal{H} \text{ and } f(\mathbf{x}) = \langle f, K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} .$$
$$\Updownarrow \text{ (Lemma B.8)}$$
$$\textbf{A Kernel as an Inner Product} \text{ (definition B.13):} \tag{B.43}$$
$$\text{A kernel is an inner product in feature space } \mathcal{H}, \text{ of feature maps } \boldsymbol{\phi} : \mathcal{X} \to \mathcal{H},$$
$$K(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}} .$$
$$\Updownarrow \text{ (Lemma B.9)}$$
$$\textbf{Positive semi-definite Function:} \text{ (definition B.14):} \tag{B.44}$$
$$\text{A symmetric function } K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \text{ is positive semi-definite if}$$
$$\mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0, \quad \mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j),\ \forall \boldsymbol{\alpha} \in \mathbb{R}^n.$$

Definition B.11 (Reproducing Kernel Hilbert Space (Schölkopf and Smola, 2002))**.** *Given a non-empty set $\mathcal{X}$, an RKHS $\mathcal{H}_K$ is an $\mathbb{R}$-Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ whose function evaluation operator $\delta_{\mathbf{x}}$ is bounded,*

$$|f(\mathbf{x})| = |\delta_{\mathbf{x}} f| \leq \lambda_{\mathbf{x}} ||f||_{\mathcal{H}_K}, \quad \forall f \in \mathcal{H}_K,\ \forall \mathbf{x} \in \mathcal{X},\ \lambda_{\mathbf{x}} \geq 0. \tag{B.45}$$

If follows that two functions that agree at every point are identical in the RKHS norm $||\cdot||_{\mathcal{H}_K} = \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{H}_K}}$ (Shawe-Taylor and Cristianini, 2004, p 62) i.e.

$$|f(\mathbf{x}) - g(\mathbf{x})| \leq \lambda_{\mathbf{x}} ||f - g||_{\mathcal{H}_K},\ \forall f, g \in \mathcal{H}_K,\ \forall \mathbf{x} \in \mathcal{X} \tag{B.46}$$

Point evaluation as a continuous linear functional (which is equivalent to a bounded linear functional in this setting) is the defining property of RKHS functions that differentiates them from other Hilbert spaces of functions, such as the set of square integrable functions $L_2(\mathcal{X})$ (Kreyszig, 1978, def. 3.1-5) i.e. it is possible to construct two functions $f, g \in L_2(\mathcal{X})$ that when evaluated over $\mathcal{X}$ are very different, but are identical in their norm induced by their inner product (Hofmann et al., 2008, p 1176). For an RKHS, if $f, g \in \mathcal{H}_K$ are close in the norm $||\cdot||_{\mathcal{H}_K}$, then $f(\mathbf{x}) \approx g(\mathbf{x})$. Evaluation

similarity and function similarity are directly related and make RKHS function classes ideal for solving the penalised ERM problem where optimisation occurs over $f \in \mathcal{H}_K$ at each sample in the training set.

Theorem B.4.3 (Riesz Representation Theorem (Kreyszig, 1978, 3.8-1))**.** *Let $\mathcal{H}$ be a Hilbert space and $\mathcal{H}^*$ be its dual space consisting of all bounded linear functionals $\delta_{\mathbf{x}} : \mathcal{H} \to \mathbb{R}$, then for every $f \in \mathcal{H}$ there exist some unique element (known as the representer of evaluation) $g_{\mathbf{x}} \in \mathcal{H}$ such that*

$$\delta_{\mathbf{x}} f = \langle f, g_{\mathbf{x}} \rangle_{\mathcal{H}}.$$

Riesz tells us that all bounded (i.e. continuous) evaluation functionals on a Hilbert space are also using inner-products (a bilinear, positive-definite and symmetric function (Kreyszig, 1978, sec 3.1)) with respect to a *unique* element (or representer of evaluation) in the Hilbert space.

Definition B.12 (Reproducing Kernel/Property (Aronszajn, 1950) and (Steinwart and Christmann, 2008, def 4.18 part 1))**.** *An $\mathbb{R}$-Hilbert space of functions has a reproducing kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ if i) for every $\mathbf{x} \in \mathcal{X}$, $g_{\mathbf{x}} = K(\cdot, \mathbf{x}) \in \mathcal{H}_K$ and ii) the reproducing property (defining point-wise evaluation) is,*

$$f(\mathbf{x}) = \langle f, K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_K}, \qquad \forall f \in \mathcal{H}_K, \, \mathbf{x} \in \mathcal{X}. \tag{B.47}$$

Theorem B.4.4 (A Hilbert Space of functions with a reproducing kernel is equivalent to an RKHS (Steinwart and Christmann, 2008, lemma 4.19))**.** *An $\mathbb{R}$-Hilbert space of functions with a reproducing kernel is equivalent to it having linear evaluation operators $\delta_{\mathbf{x}}$ that are bounded.*

Proof. By expanding equation (B.47), it can be shown that if a Hilbert space of functions $\mathcal{H}$ is endowed with a *reproducing kernel* then $\delta_{\mathbf{x}}$ are bounded,

$$\begin{aligned} |f(\mathbf{x})| &= |\langle f, K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}}|, \\ &\leq ||K(\cdot, \mathbf{x})||_{\mathcal{H}} ||f||_{\mathcal{H}}, \\ &= \lambda_{\mathbf{x}} ||f||_{\mathcal{H}}, \end{aligned} \tag{B.48}$$

where in the second line the Cauchy-Schwarz inequality (Shawe-Taylor and Cristianini, 2004, p 51, prop. 3.5) is used. Completing the equivalence the other way (where every RKHS determines a *unique* reproducing kernel) is described by Steinwart and Christmann (2008, theorem 4.20). □

Definition B.13 (Kernel)**.** *Given a Hilbert space $\mathcal{H}$ and map $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$, then a kernel is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined as*

$$K(\mathbf{x}, \mathbf{x}') := \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle_{\mathcal{H}}, \qquad \mathbf{x}, \mathbf{x}' \in \mathcal{X}.$$

Lemma B.8 (Reproducing kernels are kernels (Steinwart and Christmann, 2008, lemma 4.19 part 2)). *The reproducing kernel K associated with RKHS $\mathcal{H}_K$ is also kernel that is an inner product in $\mathcal{H}_K$.*

Proof. The canonical feature mapping $\boldsymbol{\phi}: \mathcal{X} \to \mathcal{H}_K$ can be defined using the reproducing kernel of an RKHS $\mathcal{H}_K$ as $\boldsymbol{\phi}(\mathbf{x}) := K(\cdot, \mathbf{x})$, $\mathbf{x} \in \mathcal{X}$. By choosing a specific $\mathbf{x}' \in \mathcal{X}$ then the element $f := K(\cdot, \mathbf{x}')$ is fixed and the reproducing property can be written as

$$\begin{aligned} \langle f, K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_K} &= \langle K(\cdot, \mathbf{x}'), K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_K} \\ &= \langle \boldsymbol{\phi}(\mathbf{x}'), \boldsymbol{\phi}(\mathbf{x}) \rangle_{\mathcal{H}_K} \\ &= K(\mathbf{x}, \mathbf{x}'), \end{aligned} \tag{B.49}$$

which is a kernel. □

Corollary B.4.1. *From equation (B.48), the boundedness of an RKHS evaluation operator can be expressed in terms of the kernel,*

$$\begin{aligned} |f(\mathbf{x})| &\leq \sqrt{\langle K(\cdot, \mathbf{x}'), K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_K}} ||f||_{\mathcal{H}_K}, \\ &= \sqrt{K(\mathbf{x}, \mathbf{x})} ||f||_{\mathcal{H}_K}, \\ &= \lambda_{\mathbf{x}} ||f||_{\mathcal{H}_K}. \end{aligned}$$

Definition B.14 (Positive Semi-Definite Function (Steinwart and Christmann, 2008, lemma 4.15)). *A symmetric function $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive definite if $\forall n \geq 1$, $\forall \{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathcal{X}^n$, $\forall \mathbf{a} := [a_1, ..., a_n]^\top \in \mathbb{R}^n$ then*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0, \tag{B.50}$$

where $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. The function $K(\cdot, \cdot)$ is **strictly**[3] **positive definite** if for mutually distinct $\{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathcal{X}^n$, $\mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0$ *and* $\mathbf{a}^\top \mathbf{K} \mathbf{a} = 0 \Leftrightarrow \mathbf{a} = \mathbf{0}$.

Lemma B.9 (A kernel is equivalent to a positive semi-definite function).

Proof. Every kernel function is a positive semi-definite function follows from definition B.14 such that we can write,

$$\begin{aligned} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{i=1}^{n} \sum_{j=1}^{n} \langle a_i \boldsymbol{\phi}(\mathbf{x}_i), a_j \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{H}}, \\ &= \langle \sum_{i=1}^{n} a_i \boldsymbol{\phi}(\mathbf{x}_i), \sum_{j=1}^{n} a_j \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{H}}, \\ &= \Big|\Big| \sum_{i=1}^{n} a_i \boldsymbol{\phi}(\mathbf{x}_i) \Big|\Big|_{\mathcal{H}}^2 \geq 0, \end{aligned}$$

[3] We define "positive definite" and "positive semi-definite" as equivalent, using only the term "strict" to define a strict inequality.

which uses the linearity of the inner product and definition of the Hilbert space norm. The definition of the **kernel matrix** $\mathbf{K} \in \mathbb{R}^{n \times n}$ follows where $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. Every positive definite function is a kernel is proven in Steinwart and Christmann (2008, lemma 4.16) or Shawe-Taylor and Cristianini (2004, theorem 3.11). □

Theorem B.4.5 (Moore-Aronszajn (Aronszajn, 1950))**.** *Every symmetric, positive definite kernel constructs a unique RKHS.*

As we have seen, a unique RKHS is associated with a positive semi-definite kernel. The Moore-Aronszajn explicitly constructs the associated RKHS from the positive definite kernel $K(\cdot, \cdot)$ on $\mathcal{X} \times \mathcal{X}$. Informally (see e.g. Hofmann et al. (2008)) a Hilbert space of functions $\mathcal{H}_0 \subseteq \mathbb{R}^{\mathcal{X}}$ associated with K is then constructed containing all finite linear spans that take the form $f = \sum_{i=1}^{m} \alpha_i K(\mathbf{x}_i, \cdot) \in \mathcal{H}_0$. By defining $g = \sum_{j=1}^{n} \beta_j K(\mathbf{x}'_j, \cdot) \in \mathcal{H}_0$ then for any $f, g \in \mathcal{H}_0$, the inner product (whose bilinearity, symmetric and positive definite properties need to be confirmed) between any two functions is defined as

$$\begin{aligned} \langle f, g \rangle_{\mathcal{H}_0} &= \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \langle K(\mathbf{x}_i, \cdot), K(\mathbf{x}'_j, \cdot) \rangle_{\mathcal{H}_0}, \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j K(\mathbf{x}_i, \mathbf{x}'_j). \end{aligned}$$

This pre-RKHS $\mathcal{H}_0$ is then further shown to possess all of the characteristics of an RKHS such as the reproducing property (definition B.12). To show that $\mathcal{H}_0$ is in fact an RKHS $\mathcal{H}_K$ requires that $\mathcal{H}_0$ also contains the limits of all f under the norm induced by the inner product. This is known as *completing* $\mathcal{H}_0$ *w.r.t its norm* and a formal analysis can be found in Steinwart and Christmann (2008, theorem 4.21). As before, explicit $\boldsymbol{\phi}$ representation is never required, **only a symmetric positive semi-definite kernel is needed to specify a unique RKHS**. As a consequence, a function is formed from a summation of kernel evaluations,

$$\begin{aligned} f(\mathbf{x}) &= \langle f, K(\cdot, \mathbf{x}) \rangle_K \ , \\ &= \Big\langle \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \cdot), K(\cdot, \mathbf{x}) \Big\rangle_K \ , \\ &= \sum_{j=1}^{n} \alpha_j \langle K(\mathbf{x}_j, \cdot), K(\cdot, \mathbf{x}) \rangle_K \ , \\ &= \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \mathbf{x}), \end{aligned} \tag{B.51}$$

which concludes the dual function representation as specified by equation (B.40).

Choosing Kernels

Theorem B.4.5 states that all one needs to do in order to specify an RKHS is to choose a kernel that is a symmetric and positive definite function. It is this condition that guarantees the kernel to be an inner product in a feature space (that is a Hilbert space). Common examples include linear kernels $K(\mathbf{x},\mathbf{x}') = \langle \mathbf{x},\mathbf{x}'\rangle_{\mathcal{X}}$, polynomial kernels $k_b(\mathbf{x},\mathbf{x}') = (\langle \mathbf{x},\mathbf{x}'\rangle_{\mathcal{X}} + c)^b$ (where $c \geq 0$, $b > 0$), Gaussian kernels $k_\sigma(\mathbf{x},\mathbf{x}') = \exp(-\frac{1}{2\sigma^2}||\mathbf{x}-\mathbf{x}'||^2_{\mathcal{X}})$ (where bandwidth $\sigma > 0$) and more complex kernel construction techniques all detailed by Shawe-Taylor and Cristianini (2004, chapter 9). At no point is an explicit feature representation required. The *kernel trick* also originates from the Moore-Aronszajn theorem where linear algorithms can be transformed into non-linear forms by replacing all inner products between variables $\mathbf{x},\mathbf{x}' \in \mathcal{X}$ (including inner products between features) with kernel evaluations $K(\mathbf{x},\mathbf{x}')$. The non linearity being achieved by the rich implicit feature representations defined by the kernel.

An order $b=2$ polynomial kernel (where $c=0$), over $\mathcal{X}=\mathbb{R}^2$ input variables $\mathbf{x}=[x_1,x_2]^\top$ and $\mathbf{y}=[y_1,y_2]^\top$, does not have a unique feature representation. Both $\boldsymbol{\phi}:\mathbb{R}^2\to\mathbb{R}^3$ and $\boldsymbol{\phi}:\mathbb{R}^2\to\mathbb{R}^4$ are equivalent to this kernel,

$$\begin{aligned} K(\mathbf{x},\mathbf{y}) &= \langle \mathbf{x},\mathbf{y}\rangle^2_{\mathbb{R}^2}, \\ &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2, \\ &= \left\langle \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}, \begin{bmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}y_1y_2 \end{bmatrix} \right\rangle_{R^3} = \left\langle \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \\ x_1x_2 \end{bmatrix}, \begin{bmatrix} y_1^2 \\ y_2^2 \\ y_1y_2 \\ y_1y_2 \end{bmatrix} \right\rangle_{R^4}. \end{aligned}$$

It is the kernel that is unique and by theorem B.4.5, this is equivalent to a unique RKHS. In order to quantify the relationship of the polynomial feature representation dimension as a function of b requires the binomial theorem (Shawe-Taylor and Cristianini, 2004, p 292),

$$k_b(\mathbf{x},\mathbf{y}) = \sum_{s=0}^{b} \binom{b}{s} c^{b-s} \langle \mathbf{x},\mathbf{y}\rangle_2^s, \qquad \dim(\phi) = \binom{\dim(\mathcal{X})+b}{b} \sim \mathcal{O}(\dim(\mathcal{X})^b). \quad \text{(B.52)}$$

The cost of the explicit inner product in feature space is exponential in b, but the implicit inner product (kernel) calculation is only $\sim\mathcal{O}(\dim(\mathcal{X}))$. If very high order polynomial features are required to make the hypothesis set rich enough, then a kernel-based function class has distinct cost advantage.

Gaussian Kernels and Infinite Dimensional Feature Space

Gaussian kernels are used throughout this thesis and can be written as the expansion (Shawe-Taylor and Cristianini, 2004, p 64)

$$\begin{aligned} k_\sigma(\mathbf{x},\mathbf{x}') &= \exp\Big(-\frac{1}{2\sigma}||\mathbf{x}-\mathbf{x}'||_2^2\Big), \\ &= \exp\Big(-\frac{1}{2\sigma}(||\mathbf{x}||_2^2+||\mathbf{x}'||_2^2)\Big)\exp\Big(\frac{1}{\sigma}\langle\mathbf{x},\mathbf{x}'\rangle_2\Big), \\ &= \exp\Big(-\frac{1}{2\sigma}(||\mathbf{x}||_2^2+||\mathbf{x}'||_2^2)\Big)\sum_{j=0}^{\infty}\frac{\langle\mathbf{x},\mathbf{x}'\rangle_2^j}{\sigma^j j!}, \\ &= \exp\Big(-\frac{1}{2\sigma}(||\mathbf{x}||_2^2+||\mathbf{x}'||_2^2)\Big)\sum_{j=0}^{\infty}\frac{k_{b=j}(\mathbf{x},\mathbf{x}')}{\sigma^j j!}, \end{aligned}$$

where the Taylor expansion in line three consists of a series of polynomial kernels of degrees $b=1\rightarrow\infty$ (see equation (B.52)). Given that a polynomial kernel corresponds to explicit features representations, then if the Gaussian kernel is an infinite sum over all degrees of polynomial kernels, then it too not only corresponds to explicit feature representation, but also the feature space dimensionality is infinite. The higher degree terms fall off by a factor of $j!$ thus diminishing non-smooth terms, however in this form no statements are made about the convergence of the infinite series. Mercer's theorem (Mercer, 1909) was originally used to construct valid kernels whose inner products are over Hilbert space feature mappings. The previous analysis on RKHS construction (including the Moore-Aronszajn theorem) somewhat supersedes Mercer's approach because the only assumption on $\mathcal{X}$ is that it is a non-empty set. However Mercer's approach is still useful to describe an RKHS and its smoothness characteristics induced by the RKHS norm. Indeed the RKHS norm as a regulariser constrains empirical risk minimisation - smoothness is induced in the function approximator when the RKHS norm is small. For extensive discussion on this subject with comparison to primal regularised risk minimisation, see Rasmussen et al. (2006, section 4.2), Shawe-Taylor and Cristianini (2004, pp 64-68) and Evgeniou et al. (2000, sec 3).

Kernel Matrix Decompositions

Factorising techniques are useful in reducing the exposure of computational complexity, w.r.t the number of data samples, of parent algorithms that extensively use and manipulate kernel matrices.

Gram-Schmidt Orthonormalisation and QR Decomposition SVD used in equation (B.30) demonstrates that projecting a vector $\phi(\mathbf{x})\in\mathcal{F}$ onto an orthonormal basis $\text{col}(\mathbf{Q})$, denoted by $\text{Proj}_{\mathbf{Q}}\boldsymbol{\phi}(\mathbf{x}):=\mathbf{Q}\mathbf{Q}^\top\boldsymbol{\phi}(\mathbf{x})$. A vector orthogonal to this projection by simple geometry is $\text{Proj}_{\mathbf{Q}}^{\perp}\boldsymbol{\phi}(\mathbf{x})=\nu^{-1}(\mathbf{I}-\mathbf{Q}\mathbf{Q}^\top)\boldsymbol{\phi}(\mathbf{x})$ with appropriate

normalisation constant ν. Following Shawe-Taylor and Cristianini (2004, p 124) then given data $\mathbf{X} := [\boldsymbol{\phi}(\mathbf{x}_1), ..., \boldsymbol{\phi}(\mathbf{x}_n)]^\top$, Gram-Schmidt orthonormalisation incrementally builds an orthonormal basis $\mathbf{Q}_n := [\mathbf{q}_1, ..., \mathbf{q}_n]$ as follows. For each $i = 1, ...n$ take new data point $\boldsymbol{\phi}(\mathbf{x}_i)$, and

i find orthogonal vector $\mathbf{q}_i = \text{Proj}^{\perp}_{\mathbf{Q}_{i-1}} \boldsymbol{\phi}(\mathbf{x}_i) = \nu^{-1}(\mathbf{I} - \mathbf{Q}_{i-1}\mathbf{Q}_{i-1}^\top)\boldsymbol{\phi}(\mathbf{x}_i)$,

ii $\mathbf{Q}_i \leftarrow \mathbf{Q}_{i-1} \cup \mathbf{q}_i$.

The data can therefore be represented as the QR-decomposition,

$$\mathbf{X}^\top = \mathbf{QR},$$

where $\mathbf{R} = [\mathbf{r}_1, ..., \mathbf{r}_n] \in \mathbb{R}^{n \times n}$ is an upper triangular matrix and i^{th} column $\mathbf{r}_i$ contains the coordinates of $\boldsymbol{\phi}(\mathbf{x}_i)$ in the orthonormal basis i.e. for a basis of n coordinates then $\boldsymbol{\phi}(\mathbf{x}_i) = \sum_{j=1}^{n} \mathbf{q}_j \mathbf{R}_{ji} = \mathbf{Q}_n \mathbf{r}_i$.

Cholesky Decomposition of a Kernel Matrix Performing a Cholesky decomposition on a positive semi-definite matrix is unique and if $\mathcal{F}$ is some feature space, then a Cholesky decomposition on a kernel matrix is equivalent to performing Gram-Schmidt orthonormalisation in this feature space (Shawe-Taylor and Cristianini, 2004, def 5.10). This results in the following decomposition,

$$\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \mathbf{R}^\top \mathbf{Q}_n^\top \mathbf{Q}_n \mathbf{R} = \mathbf{R}^\top \mathbf{R},$$

where $\mathbf{Q}_n^\top \mathbf{Q}_n = \mathbf{I}_n$ and each column $\mathbf{r}_i \in \mathbf{R}$ is a new mapping $\hat{\boldsymbol{\phi}} : \mathbf{x}_i \rightarrow \mathbf{r}_i$ i.e.

$$\begin{aligned} \mathbf{K}_{ij} &= \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}}, \\ &= \langle \mathbf{r}_i, \mathbf{r}_j \rangle_2 . \end{aligned}$$

The Cholesky implementation doesn't require explicitly representing the basis $\mathbf{Q}$ and instead constructs $\mathbf{R}$ directly from $\mathbf{K}$ where $\mathbf{R}_{ji} = \langle \mathbf{q}_j, \boldsymbol{\phi}(\mathbf{x}_i) \rangle_{\mathcal{F}}$ for $i > j$. By processing the data in order of the size of their residual norms (associated with ν), then a basis can be constructed that captures the most important dimensions of the data. An *incomplete* Cholesky decomposition limits increasing the basis by adding a tolerance on the residual norm, under which any newly processed data point won't be used to augment the basis. This investigation regularly limits the basis to $m_{\text{chol}} < n$ such that $\mathbf{R} \in \mathbb{R}^{m_{\text{chol}} \times n}$ in order to place a hard limit on computational costs associated with parent algorithms that are manipulating kernel matrices. This is achieved with a minor modification to Shawe-Taylor and Cristianini (2004, code fragment 5.4) whose decomposition computational complexity is linear in the size of the data $\sim \mathcal{O}(n\, m_{\text{chol}}^2)$, assuming an existing kernel matrix is already calculated. This subroutine is referred

to as IncompleteCholesky($\mathbf{K}, m_{\text{chol}}, \xi$) where factorisation of $\mathbf{K}$ will stop if either the residual norm falls below a threshold ξ or the basis exceeds m_{chol}.

B.4.3 Penalised Empirical Risk Revisited

The primal $\hat{\mathcal{L}}(\mathbf{w})$ (equation (B.33)) and dual $\hat{\mathcal{L}}(\boldsymbol{\alpha})$ (equation (B.38)) penalised ERM problems have both been posed and solutions given. The celebrated *representer theorem* proves that a minimiser over the finite training set is the minimiser to the full penalised empirical risk minimisation problem.

Representer Theorem

By explicitly choosing the hypothesis space as an RKHS $\mathcal{H}{=}\mathcal{H}_K$ in the penalised empirical risk minimisation problem (B.32), then the *representer theorem* formalises working in the dual (see section B.4.2).

Theorem B.4.6 (Representer Theorem (Schölkopf et al., 2001)[Thm. 1])**.** *Given a non-empty set $\mathcal{X}$, an RKHS $\mathcal{H}_K$ with kernel $K : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$, $\alpha_j \in \mathbb{R}$, training data $\mathcal{D}_n = \{\mathbf{x}_i, y_i\}_{i=1}^n$, with targets $\mathcal{Y} \in \mathbb{R}$, a strictly monotonically increasing real-valued function g on $[0,\infty]$, cost function $V : \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}\cup\{\infty\}$, then the minimiser $\hat{f} \in \mathcal{H}_K$ of the regularised risk functional*

$$\hat{f} = \underset{f\in\mathcal{H}_K}{\arg\min}\Big[\sum_{i=1}^{n} V(f(\mathbf{x}_i), y_i) + g(||f||_{\mathcal{H}_K})\Big],$$

*is a linear combination of kernel evaluations over **training samples**,*

$$\hat{f}(\cdot) = \sum_{j=1}^{n} \hat{\alpha}_j K(\mathbf{x}_j, \cdot) \in \hat{\mathcal{H}}_k \subseteq \mathcal{H}_K.$$

Kimeldorf and Wahba (1971) first proposed this theorem with when V is a squared loss and $g(||f||_{\mathcal{H}_K}) := \lambda||f||^2_{\mathcal{H}_K}$. Schölkopf et al. (2001) relaxed the squared loss and regulariser assumption to those assumptions stated in theorem B.4.6. Further broadening of the conditions under which the representer theorem hold can be found in Yu et al. (2013). The proof of theorem B.4.6 is based on the orthogonal decomposition (see theorem B.4.1) of the RKHS $\mathcal{H}_K = \hat{\mathcal{H}}_k \oplus \hat{\mathcal{H}}_k^{\perp}$ where $\hat{\mathcal{H}}_k := \{\hat{f} \in \mathcal{H}_K \mid \hat{f} = \sum_{j=1}^n \alpha_j K(\mathbf{x}_j, \cdot),\ \alpha_j \in \mathbb{R}\}$ is the space of functions that span the training samples in feature space $\mathcal{H}_K$. Thus any RKHS function evaluation is decomposed into

$$\begin{aligned} f(\mathbf{x}) &= (\hat{f} + \hat{f}^{\perp})(\mathbf{x}), \\ &= \hat{f}(\mathbf{x}) + \hat{f}^{\perp}(\mathbf{x}), \\ &= \langle \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \cdot), K(\cdot, \mathbf{x})\rangle_{\mathcal{H}_K} + \langle \hat{f}^{\perp}, K(\cdot, \mathbf{x})\rangle_{\mathcal{H}_K}, \end{aligned} \tag{B.53}$$

Table B.1 Least-squares computational complexity (scalar-valued $\hat{f}$) for training and function evaluation where n is the size of the training set.

Algorithm	Operations	
	Closed-form training	$\hat{f}$ evaluation
Primal	$\mathcal{O}(d'^3)$	$\mathcal{O}(d')$
Dual	$\mathcal{O}(n^3)$	$\mathcal{O}(n)$

where in the last line the reproducing property is invoked. The key point is that evaluating function (B.53) at any training point $\mathbf{x}_i \in \mathcal{D}_n$ results in $\langle \hat{f}^{\perp}, K(\cdot, \mathbf{x}_i)\rangle_{\mathcal{H}_K} = 0$ due to orthogonality. The value of the loss function V in the penalised ERM is therefore independent of $\hat{f}^{\perp}$, which is consistent with replacing f with $\hat{f}$ in the loss function. However the orthogonal component in the ERM's penalising norm still exists. By Pythagoras' theorem,

$$\begin{aligned} g(||f||_{\mathcal{H}_K}) &= g(\sqrt{||\hat{f}||^2_{\mathcal{H}_K} + ||\hat{f}^{\perp}||^2_{\mathcal{H}_K}}), \\ &\geq g(||\hat{f}||_{\mathcal{H}_K}), \end{aligned}$$

then the best choice for f that minimises the regulariser term is when $||\hat{f}^{\perp}||_{\mathcal{H}_K} = 0$. Thus by choosing $\mathcal{H} = \hat{\mathcal{H}}_k$ for the penalised empirical risk minimisation problem, the solution is $\hat{f} = \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \cdot) \in \hat{\mathcal{H}}_k$.

Tension in Complexity

The table B.1 describes the complexity of learning primal and dual functions in the closed-form regression problem where $d' = \dim(\mathcal{F})$ is the dimensionality of the explicit feature space.

B.4.4 Vector-Valued Primal Regression

The general vector-valued minimisation problem is a simple extension to the scalar penalised empirical risk problem (B.26),

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \Big[\sum_{j=1}^{D} \frac{1}{2n} \sum_{i=1}^{n} V(f_j(\mathbf{x}_i), y_{ij}) + \lambda g(f) \Big],$$

where L is assumed to be the squared loss, $D = \dim(\mathcal{Y})$, $f : \mathcal{X} \to \mathcal{Y}$ and $f_j(\mathbf{x})$ is the prediction of the j^{th} output dimension at $\mathbf{x}$. The choice of $\mathcal{H}$ is the interesting component of this problem and below both the linear primal and RKHS settings are summarised.

Vector-Valued Primal Ridge Regression

Let $\mathcal{X} \in \mathbb{R}^d$ and vector dependent variables be in $\mathcal{Y} \in \mathbb{R}^D$, then assuming that all D dimensions of $\mathcal{Y}$ are independent then the penalised empirical risk minimisation problem is tantamount to solving each scalar problem (Hastie et al., 2001; Rosenblatt, 1956),

$$\hat{\boldsymbol{\beta}}^{(j)} = \arg\min_{\boldsymbol{\beta}^{(j)}} \Big[\frac{1}{2n} \sum_{i=1}^{n} (y_{ij} - \mathbf{x}_i^\top \boldsymbol{\beta}^{(j)})^2 + \frac{\lambda}{2} ||\boldsymbol{\beta}^{(j)}||_2^2 \Big].$$

By defining the regressor $\mathbf{W} := [\boldsymbol{\beta}^{(1)}, .., \boldsymbol{\beta}^{(D)}] \in \mathbb{R}^{d \times D}$, then the minimisation problem is characterised by choosing $g(\mathbf{f}) = \frac{1}{2} \sum_j ||\boldsymbol{\beta}^{(j)}||_2^2 = \frac{1}{2} ||\mathbf{W}||_{\text{Fr}}^2 = \frac{1}{2} \sum_{ij} |\beta_{ij}|^2$ then

$$\sum_{j=1}^{D} \Big(\frac{1}{2n} ||\mathbf{y}^{(j)} - \mathbf{X}\boldsymbol{\beta}^{(j)}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\beta}^{(j)}||_2^2 \Big) = \frac{1}{2n} ||\mathbf{Y} - \mathbf{X}\mathbf{W}||_{\text{Fr}}^2 + \frac{\lambda}{2} ||\mathbf{W}||_{\text{Fr}}^2.$$

The solution is found by solving D regression problems, or in compact matrix form

$$\begin{aligned}
\mathbf{0} &= \nabla_{\mathbf{W}} \Big[\frac{1}{2n} ||\mathbf{Y} - \mathbf{X}\mathbf{W}||_{\text{Fr}}^2 + \frac{\lambda}{2} ||\mathbf{W}||_{\text{Fr}}^2 \Big], \\
&= \nabla_{\mathbf{W}} \Big[\frac{1}{2n} \text{Tr}\Big((\mathbf{Y} - \mathbf{X}\mathbf{W})^\top (\mathbf{Y} - \mathbf{X}\mathbf{W}) \Big) + \frac{\lambda}{2} \text{Tr}\Big(\mathbf{W}^\top \mathbf{W} \Big) \Big], \\
&= -\frac{1}{n} \mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\mathbf{W}) + \lambda \mathbf{W}, \\
\Rightarrow \hat{\mathbf{W}}_{\text{ridge}} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{Y},
\end{aligned} \tag{B.54}$$

where line three uses the identities (B.80) and (B.82). The minimiser is a mapping $\hat{\mathbf{W}}_{\text{ridge}}^\top : \mathcal{X} \to \mathcal{Y}$ such that predictions are made by $\hat{\mathbf{f}}(\mathbf{x}) = \hat{\mathbf{W}}_{\text{ridge}}^\top \mathbf{x}$.

Group Lasso Penalisation

Soft Thresholding The general mixed norm is used to define the L21 norm on the matrix $\mathbf{W} \in \mathbb{R}^{d \times D}$,

$$\begin{aligned}
||\mathbf{W}||_{pq} &:= \left(\sum_{j=1}^{d} \left(\sum_{\ell=1}^{D} |w_{j\ell}|^p \right)^{q/p} \right)^{1/q}, \\
\Rightarrow ||\mathbf{W}||_{21} &= \sum_{j=1}^{d} \left(\sum_{\ell=1}^{D} |w_{j\ell}|^2 \right)^{1/2}, \\
&= \sum_{j=1}^{d} ||\mathbf{w}_{j:}||_2,
\end{aligned} \tag{B.55}$$

where $\mathbf{w}_{j:}$ is the j^{th} row of $\mathbf{W}$. Incorporating this into the vector-valued regression problem describes the well-known group lasso (Yuan and Lin, 2006) problem specialised specifically for inducing row sparsity in $\mathbf{W}$ (i.e. diminishes less important dimensions

Algorithm 25 vvRegression-Primal($\mathcal{D}$)

Input: Data $\mathcal{D} := \{\mathbf{X}, \mathbf{Y}\}$, $\mathbf{X} := [\mathbf{x}_1, ..., \mathbf{x}_n]^\top$, $\mathbf{Y} := [\mathbf{y}_1, ..., \mathbf{y}_n]^\top$, $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$.
Output: linear mapping $\mathbf{W}^\top : \mathcal{X} \to \mathcal{Y}$.
Initialise: $\epsilon_{\text{bestTest}} \leftarrow \infty$, $n_{\text{folds}} \leftarrow 5$, $n_{\text{test}} \leftarrow n/n_{\text{folds}}$, $d \leftarrow \dim(\mathcal{X})$, partitions $\{\mathcal{D}_1, ..., \mathcal{D}_{n_{\text{folds}}}\} \leftarrow \mathcal{D}$, $\{\lambda\} \leftarrow \{10^{-10}, ..., 10^0\}$.
Cross-validate regulariser:
for each $\lambda \in \{\lambda\}$ **do**
    $\epsilon_{\text{sumTest}} \leftarrow 0$
    for $k = 1$ **to** n_{folds} **do**
        $\mathcal{D}_{\text{train}} \leftarrow (\dot{\bigcup}_{i=1}^{n_{\text{folds}}} \mathcal{D}_i)_{i \neq k}$ ▷ $\mathcal{D}_{\text{train}} := \{\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}\}$
        $\mathcal{D}_{\text{test}} \leftarrow \mathcal{D}_k$ ▷ $\mathcal{D}_{\text{test}} := \{\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}\}$
        $\mathbf{W}_{\text{train}} = (\mathbf{X}_{\text{train}}^\top \mathbf{X}_{\text{train}} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}_{\text{train}}^\top \mathbf{Y}_{\text{train}}$ ▷ equation (4.8)
        $\epsilon_{\text{sumTest}} \leftarrow \epsilon_{\text{sumTest}} + \frac{1}{2n_{\text{test}}} ||\mathbf{X}_{\text{test}} \mathbf{W}_{\text{train}} - \mathbf{Y}_{\text{test}}||_{\text{Fr}}^2$
    end for
    $\epsilon_{\text{avTest}} \leftarrow \epsilon_{\text{sumTest}}/n_{\text{folds}}$
    if $\epsilon_{\text{avTest}} < \epsilon_{\text{bestTest}}$ **then** ▷ compare estimate of out-of-sample error
        $\lambda^* \leftarrow \lambda$, $\epsilon_{\text{bestTest}} \leftarrow \epsilon_{\text{avTest}}$.
    end if
end for
$\mathbf{W} \leftarrow (\mathbf{X}^\top \mathbf{X} + \lambda^* \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{Y}$ ▷ equation (4.8)
return W

of the input variable),

$$\frac{1}{2n}||\mathbf{Y} - \mathbf{X}\mathbf{W}||_{\text{Fr}}^2 + \lambda||\mathbf{W}||_{21} = \frac{1}{2n}||\mathbf{Y} - \sum_{j=1}^{d} \mathbf{x}^{(j)}\mathbf{w}_{j:}||_{\text{Fr}}^2 + \lambda \sum_{j=1}^{d} ||\mathbf{w}_{j:}||_2,$$

where $\mathbf{w}_{j:} \in \mathbb{R}^{1 \times D}$, $\mathbf{X} := [\mathbf{x}^{(1)}, ..., \mathbf{x}^{(d)}]$ and $\mathbf{x}^{(j)} \in \mathbb{R}^n$. This loss can be solved by separately differentiating w.r.t. each $\mathbf{w}_{j:}$ and solving

$$\mathbf{0}^\top = -\frac{1}{n}\mathbf{x}^{(j)\top}(\mathbf{Y} - \sum_{j'=1}^{d} \mathbf{x}^{(j')}\mathbf{w}_{j':}) + \lambda \nabla_{\mathbf{w}_{j:}}||\mathbf{w}_{j:}||_2,$$

where unfortunately the derivative for the L2 norm is not defined everywhere. Proceeding with

$$\begin{aligned}
\nabla_{\mathbf{w}_{j:}}||\mathbf{w}_{j:}||_2 &= \nabla_{\mathbf{w}_{j:}} \left(\sum_{\ell=1}^{D} |w_{j\ell}|^2 \right)^{1/2} = \frac{\nabla_{\mathbf{w}_{j:}} \sum_{\ell=1}^{D} |w_{j\ell}|^2}{2||\mathbf{w}_{j:}||_2}, \\
\Rightarrow \frac{\partial}{\partial w_{j\ell}}||\mathbf{w}_{j:}||_2 &= \frac{|w_{j\ell}| \operatorname{sign}(w_{j\ell})}{||\mathbf{w}_{j:}||_2} = \frac{w_{j\ell}}{||\mathbf{w}_{j:}||_2}, \\
\Rightarrow \nabla_{\mathbf{w}_{j:}}||\mathbf{w}_{j:}||_2 &= \frac{\mathbf{w}_{j:}}{||\mathbf{w}_{j:}||_2} \in \mathbb{R}^D,
\end{aligned}$$

then clearly if $||\mathbf{w}_{j:}||_2 = 0$ then the derivative is undefined and therefore we must define the subderivative (Hastie et al., 2015, p. 63),

$$\partial||\mathbf{w}_{j:}||_2 = \begin{cases} \mathbf{w}_{j:}/||\mathbf{w}_{j:}||_2 & ||\mathbf{w}_{j:}|| \neq 0, \\ \mathbf{u} \in \mathbb{R}^D : ||\mathbf{u}||_2 \leq 1 & ||\mathbf{w}_{j:}|| = 0. \end{cases} \tag{B.56}$$

The minimisation problem that we are now faced with is

$$\hat{\mathbf{W}} = \underset{\mathbf{W} \in \mathbb{R}^{d \times D}}{\arg\min} \Big[\mathcal{L}(\mathbf{W}) + \Omega(\mathbf{W}) \Big]$$

where $\mathcal{L}$ is the usual convex differentiable (smooth) loss function and Ω is the regulariser which is non differentiable (non-smooth). A common approach to solve this problem is to decouple the minimisation of both functions into separate operations in an iterative proximal gradient (Hastie et al., 2015, section 5.3.3) (also known as a projected gradient) descent where for the group lasso example

$$\mathbf{W}^{(t+1)} = \text{prox}_{\lambda||\cdot||_{21}} \Big[\mathbf{W}^{(t)} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}^{(t)}) \Big]. \tag{B.57}$$

The prox operator is a generalisation of a projection operation that projects $\mathbf{W}$ to a solution constrained by $||\cdot||_{21}$. In general if $\mathbf{x} \in \mathcal{H}$ then prox operator $\text{prox}_{\lambda||\cdot||} : \mathcal{H} \to \mathcal{H}$ is defined (Hastie et al., 2015, p. 63) for some $\mathcal{H}$ is a Hilbert space as

$$\text{prox}_{\lambda||\cdot||}[x] := \underset{\hat{\mathbf{x}} \in \mathcal{H}}{\arg\min} \Big[\frac{1}{2}||\hat{\mathbf{x}} - \mathbf{x}||^2_{\mathcal{H}} + \lambda||\hat{\mathbf{x}}|| \Big].$$

Given that the norm (equation (B.55)) is separable w.r.t. each individual group, then $\Big[\text{prox}_{\lambda||\cdot||_{21}}[\mathbf{W}^{(t)}]\Big]_{j:} = \text{prox}_{\lambda||\cdot||_2}[\mathbf{w}^{(t)}_{j:}]$ such that the operation can be carried out on each weight group $\mathbf{w}_{j:}$ independently. By solving the definition of the prox operation for the L21 norm then $\text{prox}_{\eta\lambda||\cdot||_{21}}[\mathbf{w}_{j:}]$ is in fact the group soft-thresholding operator (Hastie et al., 2015, eqn 4.16b)

$$\Big[S^{21}_{\eta\lambda}(\mathbf{w}_{j:})\Big]_\ell = \begin{cases} \Big(1 - \frac{\eta\lambda}{||\mathbf{w}_{j:}||_2}\Big) w_{j\ell} & ||\mathbf{w}_{j:}||_2 > \eta\lambda, \\ 0 & ||\mathbf{w}_{j:}||_2 \leq \eta\lambda, \end{cases} \tag{B.58}$$

which is succinctly $\Big[S^{21}_{\eta\lambda}(\mathbf{w}_{j:})\Big]_\ell = \Big[1 - \frac{\eta\lambda}{||\mathbf{w}_{j:}||_2}\Big]_+ w_{j\ell}$, where $[\,\cdot\,]_+ := \max[\cdot, 0]$. For the special case of each group being exactly each ℓ^{th} element of $\mathbf{w}_{j:}$ (i.e. the $j\ell^{\text{th}}$ element of $\mathbf{W}$) then the group lasso collapses to the lasso $\Big[\text{prox}_{\lambda||\cdot||_2}[\mathbf{w}^{(t)}_{j:}]\Big]_\ell = \Big[\text{prox}_{\lambda||\cdot||_1}[\mathbf{w}^{(t)}_{j:}]\Big]_\ell$ whose soft-thresholding operator is

$$\Big[S^{1}_{\eta\lambda}(\mathbf{w}_{j:})\Big]_\ell = \begin{cases} (|w_{j\ell}| - \eta\lambda)\,\text{sign}(w_{j\ell}) & |w_{j\ell}| > \eta\lambda, \\ 0 & |w_{j\ell}| \leq \eta\lambda. \end{cases} \tag{B.59}$$

which is succinctly $\left[S^1_{\eta\lambda}(\mathbf{w}_{j:})\right]_\ell = \left[|w_{j\ell}| - \eta\lambda\right]_+ \text{sign}(w_{j\ell})$. This can easily be derived by noticing the L2 norm in equation (B.58) of the group is replaced by $|w_{j\ell}|$ and by making the substitution $w_{j\ell} := |w_{j\ell}| \,\text{sign}(w_{j\ell})$.

Optimising Lasso-Type Problems Iterative soft-thresholding (ISTA) (see Beck and Teboulle (2009) and references therein) solves this optimisation problem by iteratively making a gradient descent step followed by a projection (or soft-thresholding operation). The celebrated FISTA (Beck and Teboulle, 2009) algorithm is a variant of this approach that combines accelerated gradient descent (Nesterov, 1983). However from experience with these methods in the experimental setting, their cost is high if one wants to maintain sparse regressors in an online fashion. Solving these lasso-type problems in the SGD domain instead proves beneficial for the algorithm developed in Chapter 6. A review of induced structural sparsity in the stochastic setting is found in Appendix B.5.3.

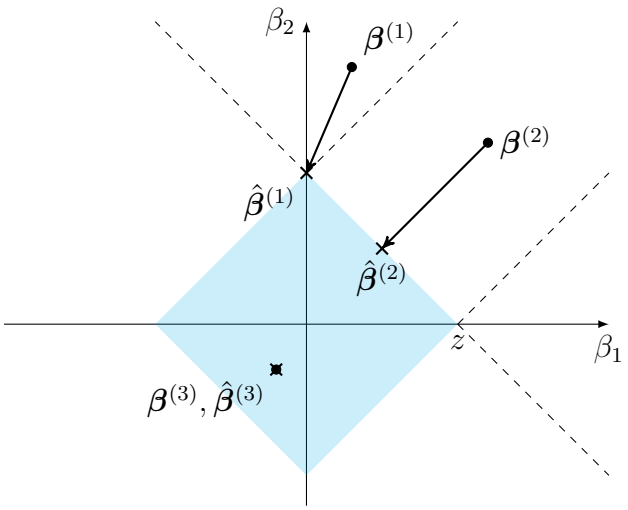
B.4.5 Sparse Projections

A Euclidean projection onto the L1-ball (see fig. B.4) is described by the following optimisation problem,

$$\begin{aligned} \underset{\hat{\boldsymbol{\beta}}}{\text{minimise}} \quad & ||\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}||_2^2 \\ \text{subject to} \quad & ||\hat{\boldsymbol{\beta}}||_1 \leq z, \end{aligned}$$

which can be solved in linear time $\sim\mathcal{O}(\dim(\boldsymbol{\beta}))$ by Duchi et al. (2008). By setting $z = 1$ then any point lying outside the unit ball will be projected onto its surface. Points occupying the interior are not projected and sparsity is induced when a vector is projected to any of the ball's vertices.

Figure B.4 Euclidean projections onto the L1-ball for $\boldsymbol{\beta} \in \mathbb{R}^2$.



B.4.6 Vector-Valued RKHS Regression

Vector-Valued RKHS

By following (Grünewälder et al., 2012b; Micchelli and Pontil, 2005), a vector-valued RKHS (vvRKHS) can be formulated by extending the scalar RKHS (definition B.11) and reproducing property (definition B.12) to the vector-valued setting as summarised below.

Definition B.15 (Vector-Valued Reproducing Kernel Hilbert Space (Grünewälder et al., 2012b))**.** *Let $\mathcal{X}$ be a non-empty set, let $\mathcal{Y}$ be a Hilbert space with inner product $\langle\cdot,\cdot\rangle_{\mathcal{Y}}$ and let $(\mathcal{H},\langle\cdot,\cdot\rangle_{\mathbf{\Gamma}})$ be a Hilbert space of functions $\mathbf{f}:\mathcal{X}\to\mathcal{Y}$. Then $\mathcal{H}$ is a $\mathcal{Y}$-valued RKHS $\mathcal{H}_{\mathbf{\Gamma}}$, if for all $\mathbf{x}\in\mathcal{X}$ and $\mathbf{y}\in\mathcal{Y}$ the linear functional $\mathbf{f}\to\langle\mathbf{y},\mathbf{f}(\mathbf{x})\rangle_{\mathcal{Y}}$ is continuous.*

By the Riesz representation theorem B.4.3, there exists for every $\mathbf{x}\in\mathcal{X}$ and $\mathbf{y}\in\mathcal{Y}$, an element $\mathbf{\Gamma}(\mathbf{x}|\mathbf{y})(\cdot)\in\mathcal{H}_{\mathbf{\Gamma}}$ for all $\mathbf{f}\in\mathcal{H}_{\mathbf{\Gamma}}$ such that

$$\langle\mathbf{y},\mathbf{f}(\mathbf{x})\rangle_{\mathcal{Y}}=\langle\mathbf{\Gamma}(\mathbf{x}|\mathbf{y})(\cdot),\mathbf{f}\rangle_{\mathbf{\Gamma}}. \tag{B.61}$$

Recall in scalar-valued setting, Riesz implies $yf(x)=\langle K(\mathbf{x},\cdot)y,f\rangle_K$. In the vector-valued setting this implies that there exists a linear operator $\mathbf{\Gamma}(\mathbf{x},\cdot):\mathcal{Y}\to\mathcal{H}_{\mathbf{\Gamma}}$ such that $\mathbf{\Gamma}(\mathbf{x}|\mathbf{y})(\cdot):=\mathbf{\Gamma}(\mathbf{x},\cdot)\mathbf{y}\in\mathcal{H}_{\mathbf{\Gamma}}$. The vector-valued reproducing property (cf. the scalar definition B.12) is therefore defined as

$$\langle\mathbf{y},\mathbf{f}(\mathbf{x})\rangle_{\mathcal{Y}}=\langle\mathbf{\Gamma}(\mathbf{x},\cdot)\mathbf{y},\mathbf{f}\rangle_{\mathbf{\Gamma}}, \tag{B.62}$$

where $\mathbf{\Gamma}$ is the reproducing kernel.

The reproducing kernel in the vector-valued setting is related to $\mathbf{\Gamma}(\cdot,\mathbf{x})$ in the following way. Define $\mathcal{L}(\mathcal{Y})$ as the space of bounded linear operators that map $\mathcal{Y}\to\mathcal{Y}$, then the *operator-valued* reproducing kernel $\mathbf{\Gamma}:\mathcal{X}\times\mathcal{X}\to\mathcal{L}(\mathcal{Y})$ indexed at $\mathbf{x},\mathbf{x}'\in\mathcal{X}$ is $\mathbf{\Gamma}(\mathbf{x},\mathbf{x}')\in\mathcal{L}(\mathcal{Y})$, leading to the definition

$$\mathbf{\Gamma}(\mathbf{x},\mathbf{x}')\mathbf{y}=(\mathbf{\Gamma}(\mathbf{x}|\mathbf{y}))(\mathbf{x}'). \tag{B.63}$$

If $\mathbf{\Gamma}:\mathcal{X}\times\mathcal{X}\to\mathcal{L}(\mathcal{Y})$ is defined by both equation (B.63) and equation (B.62) then it is a kernel i.e. it satisfies conditions set out in Micchelli and Pontil (2005, proposition 2.1). These conditions specify a valid kernel and can be compared to the scalar RKHS setting regarding lemma B.9 and definition B.14. One of these conditions is the inner product. By choosing $\mathbf{f}(\cdot)=\mathbf{\Gamma}(\mathbf{x}'|\mathbf{y}')(\cdot)$ then from equations (B.61) and (B.62), the inner product is defined by

$$\begin{aligned}\langle\mathbf{\Gamma}(\mathbf{x}|\mathbf{y})(\cdot),\mathbf{\Gamma}(\mathbf{x}'|\mathbf{y}')(\cdot)\rangle_{\mathbf{\Gamma}}&=\langle\mathbf{y},\mathbf{\Gamma}(\mathbf{x}'|\mathbf{y}')(\mathbf{x})\rangle_{\mathcal{Y}}\ ,\\&=\langle\mathbf{y},\mathbf{\Gamma}(\mathbf{x}',\mathbf{x})\mathbf{y}'\rangle_{\mathcal{Y}}\ ,\quad \mathbf{x},\mathbf{x}'\in\mathcal{X},\ \mathbf{y},\mathbf{y}'\in\mathcal{Y}.\end{aligned} \tag{B.64}$$

In the spirit of theorem B.4.4 for scalar RKHS, if a kernel $\mathbf{\Gamma}$ satisfies Grünewälder et al. (2012b, proposition 2.1), then it is associated with a unique vvRKHS where $\mathbf{\Gamma}$ is its reproducing kernel. In the spirit of Moore-Aronszajn (theorem B.4.5) for scalar-valued functions, it is possible to (conversely from Riesz) define a valid kernel and explicitly construct its unique vvRKHS (see e.g. Minh et al. (2011)).

vvRKHS Penalised Empirical Risk

By considering a data set $\mathcal{D}_n := \{(\mathbf{x}, \mathbf{y})_i\}_{i=1}^n$ then Grünewälder et al. (2012b) solve the PERM problem by choosing a hypothesis class $\mathcal{H}_{\mathbf{\Gamma}}$ of vvRKHS functions,

$$\hat{\mathbf{f}} = \underset{\mathbf{f} \in \mathcal{H}_{\mathbf{\Gamma}}}{\arg\min} \Big[\frac{1}{2n} \sum_{i=1}^{n} ||\mathbf{f}(\mathbf{x}_i) - \mathbf{y}_i||_{\mathcal{Y}}^2 + \lambda ||\mathbf{f}||_{\mathbf{\Gamma}}^2 \Big]. \tag{B.65}$$

The solution takes the form (Grünewälder et al., 2012b, theorem 2.2)

$$\hat{\mathbf{f}}(\cdot) = \sum_{i=1}^{n} \mathbf{\Gamma}(\mathbf{x}_i, \cdot)\mathbf{c}_i \in \mathcal{H}_{\mathbf{\Gamma}}, \qquad \mathbf{c}_i \in \mathcal{Y},$$

where coefficients $\{\mathbf{c}_i\}_{i \leq n}$ are to be found. Micchelli and Pontil (2005, theorem 4.1) show that these coefficients can be found in closed form as they are the unique solution to the linear system of equations,

$$\sum_{i=1}^{n} \Big(\mathbf{\Gamma}(\mathbf{x}_j, \mathbf{x}_i) + \lambda \delta_{ij} \Big) \mathbf{c}_j = \mathbf{y}_i \in \mathcal{Y}, \quad 1 \leq j \leq n.$$

Kernel Choice

Equation B.64 demonstrates that relationship between kernel evaluation in the vvRKHS $\mathbf{\Gamma}$ and the output space $\mathcal{Y}$. By defining function elements as

$$\begin{aligned} \mathbf{\Gamma}(\mathbf{x}|\mathbf{y})(\cdot) &= \mathbf{\Gamma}(\mathbf{x}, \cdot)\mathbf{y}, \\ &= K(\mathbf{x}, \cdot)\mathbf{I}\mathbf{y}, \\ &= K(\mathbf{x}, \cdot)\mathbf{y} \in \mathcal{H}_{\mathbf{\Gamma}}, \end{aligned}$$

where $\mathbf{I} : \mathcal{Y} \to \mathcal{Y}$ is an identity map and scalar kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, then equation (B.64) becomes

$$\begin{aligned} \langle \mathbf{y}K(\mathbf{x}, \cdot), \mathbf{y}'K(\mathbf{x}', \cdot) \rangle_{\mathbf{\Gamma}} &= \langle \mathbf{y}, \mathbf{y}'K(\mathbf{x}', \mathbf{x}) \rangle_{\mathcal{Y}} \\ &= K(\mathbf{x}', \mathbf{x}) \langle \mathbf{y}', \mathbf{y} \rangle_{\mathcal{Y}}. \end{aligned}$$

This is tantamount to choosing kernel $\mathbf{\Gamma}(\mathbf{x}, \mathbf{x}') := K(\mathbf{x}, \mathbf{x}')\mathbf{I} \in \mathcal{L}(\mathcal{Y})$ and is valid because it can be shown to satisfy Grünewälder et al. (2012b, proposition 2.1). The solution

to the regression problem is therefore the solution to

$$\sum_{i=1}^{n} \Big(K(\mathbf{x}_j, \mathbf{x}_i)\mathbf{I} + \lambda\delta_{ij}\Big)\mathbf{c}_i = \mathbf{y}_j \in \mathcal{Y}, \quad 1 \leq j \leq n,$$
$$\Rightarrow \mathbf{c}_i = \sum_{j=1}^{n} W_{ij}\mathbf{y}_j,$$

where W_{ij} is an element in $\mathbf{W} = (\mathbf{K} + \lambda\mathbf{I}_n)^{-1}$ and $K_{ji} = K(\mathbf{x}_j, \mathbf{x}_i)$ is an element in $\mathbf{K}$. The function approximator therefore takes the form

$$\begin{aligned}
\hat{\mathbf{f}}(\mathbf{x}) &= \sum_{i=1}^{n} \boldsymbol{\Gamma}(\mathbf{x}_i, \mathbf{x})\mathbf{c}_i \in \mathcal{Y}, \\
&= \sum_{i=1}^{n} K(\mathbf{x}_i, \mathbf{x})\mathbf{I} \sum_{j=1}^{n} W_{ij}\mathbf{y}_j, \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} K(\mathbf{x}_i, \mathbf{x})W_{ij}\mathbf{y}_j, \\
&= \mathbf{Y}^\top \mathbf{W}\boldsymbol{\psi}(\mathbf{x}), \qquad \text{(B.66)}
\end{aligned}$$

where $\mathbf{Y} := [\mathbf{y}_1, ..., \mathbf{y}_n]^\top$ and $\boldsymbol{\psi}(\mathbf{x}) := [K(\mathbf{x}_1, \mathbf{x}), ..., K(\mathbf{x}_n, \mathbf{x})]^\top$. The function in equation equation (B.66) can be written as

$$\begin{aligned}
\hat{\mathbf{f}}(\cdot) &= \mathbf{Y}^\top \mathbf{W}\boldsymbol{\psi}(\cdot), \\
&= \tilde{\mathbf{W}}\boldsymbol{\psi}(\cdot), \qquad \text{(B.67)} \\
&= \sum_{i=1}^{n} \tilde{\mathbf{w}}_i K(\mathbf{x}_i, \cdot), \qquad \text{(B.68)}
\end{aligned}$$

where $\tilde{\mathbf{W}} := [\tilde{\mathbf{w}}_1, ..., \tilde{\mathbf{w}}_n]$, $\tilde{\mathbf{w}}_i \in \mathcal{Y}$, which can be directly compared to primal vector-valued ridge regression estimator.

vvRKHS Inner Product

Using the standard choice of vvRKHS kernel $\boldsymbol{\Gamma}(\mathbf{x}, \mathbf{x}') := K(\mathbf{x}, \mathbf{x}')\mathbf{I} \in \mathcal{L}(\mathcal{Y})$ where $\mathbf{I} : \mathcal{Y} \to \mathcal{Y}$ and scalar-valued kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, then $\mathcal{H}_{\boldsymbol{\Gamma}}$ has the inner product (Grünewälder et al., 2012a, D2, eqn 21)

$$\Big\langle \mathbf{y}K(\mathbf{x}, \cdot),\, \mathbf{y}'K(\mathbf{x}', \cdot) \Big\rangle_{\boldsymbol{\Gamma}} := \langle \mathbf{y}, \mathbf{y}' \rangle_{\mathcal{Y}} K(\mathbf{x}, \mathbf{x}') \qquad \mathbf{y}, \mathbf{y}' \in \mathcal{Y},\ \mathbf{x}, \mathbf{x}' \in \mathcal{X}. \qquad \text{(B.69)}$$

B.5 Artificial Neural Networks

The following is a brief review of gradient descent, stochastic gradient descent (SGD), the chain rule and backpropagtion.

B.5.1 Gradient-based Empirical Risk Minimisation

Instead of minimising the empirical loss $\hat{\mathcal{L}}$ using a closed-form batch solution, it can instead be minimised by navigating parameters in a vector space Θ, endowed with a norm $||\cdot||_\Theta$, using gradient descent (Bottou, 1998).

Batch Gradient Descent

Gradient descent is an optimisation method deployed to solve (Bottou, 1998) the penalised risk minimisation problem as described in section B.4.1. For $f_\theta : \mathcal{X} \to \mathcal{Y}$ belonging to a hypothesis class $\mathcal{H}$ of parametrised vector-valued multivariate functions, data $\mathcal{D}_n := \{(\mathbf{x}_i, \mathbf{y}_i)_i\}_{i=1}^n$ drawn i.i.d, then the batch objective is defined as

$$\hat{\mathcal{L}}_{\boldsymbol{\theta}}|_{\mathcal{D}_n} := \frac{1}{2n}\sum_{i=1}^{n} V(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) + \frac{\lambda}{2}\Omega(\boldsymbol{\theta}),$$

where for the square loss $V(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) = ||\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i) - \mathbf{y}_i||_{\mathcal{Y}}^2$ and $\Omega(\boldsymbol{\theta})$ is a regulariser on the function parameters $\boldsymbol{\theta}$. In order to form any gradient update requires the application of the chain rule.

Lemma B.10 (Chain/Composite Rule (Haggarty, 1993, sec. 6.1.6))**.** *a) The following is an extension of the scalar case to the multivariate and vector-valued setting. Let a real vector-valued and multivariate function be* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^d$*. Similarly let* $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^d$ *and* $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$ *be differentiable at* $\mathbf{h}{=}\mathbf{h}(\mathbf{x}=\mathbf{a})$ *and* $\mathbf{x}=\mathbf{a}\in\mathbb{R}^n$ *respectively. Then the derivative of the composition* $\mathbf{f}(\mathbf{x}) := (\mathbf{g}\circ\mathbf{h})(\mathbf{x})$ *w.r.t.* $\mathbf{x}$ *at* $\mathbf{x}{=}\mathbf{a}$ *is written as*

$$\begin{aligned}\left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} &= \left.\frac{\partial (\mathbf{g}\circ\mathbf{h})(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{h}}\right|_{\mathbf{h}=\mathbf{h}(\mathbf{a})} \circ \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}}, \\ \Rightarrow \frac{\partial \mathbf{f}}{\partial \mathbf{x}} &= \frac{\partial \mathbf{g}}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{x}},\end{aligned}$$

where the last line is shorthand whose Jacobian matrix elements are

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{ij} = \frac{\partial f_i(\mathbf{x}{=}\mathbf{a})}{\partial x_j}, \quad \left(\frac{\partial \mathbf{g}}{\partial \mathbf{h}}\right)_{ij} = \frac{\partial g_i(\mathbf{h}{=}\mathbf{h}(\mathbf{a}))}{\partial h_j}, \quad \left(\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right)_{ij} = \frac{\partial h_i(\mathbf{x}{=}\mathbf{a})}{\partial x_j}.$$

b) Similarly if function $\mathbf{h}$ *is parametrised by* $\theta \in \boldsymbol{\theta}$*, then*

$$\begin{aligned}\left.\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \theta}\right|_{\mathbf{x}=\mathbf{a}} &= \left.\frac{\partial (\mathbf{g}\circ\mathbf{h})(\mathbf{x})}{\partial \theta}\right|_{\mathbf{x}=\mathbf{a}} = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{h}}\right|_{\mathbf{h}=\mathbf{h}(\mathbf{a})} \circ \left.\frac{\partial \mathbf{h}}{\partial \theta}\right|_{\mathbf{x}=\mathbf{a}}, \\ \Rightarrow \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \theta} &= \frac{\partial \mathbf{g}}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \theta},\end{aligned}$$

Deferring the choice of regulariser until below, a first order gradient update rule is formed by noting that the loss is the composition between the scalar-valued risk

function V and the function approximator,

$$\begin{aligned}\hat{\mathcal{L}}_{\boldsymbol{\theta}}|_{\mathcal{D}_n} &= \frac{1}{2n}\sum_{i=1}^{n} V(\mathbf{y}_i)\circ\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i) + \frac{\lambda}{2}\Omega(\boldsymbol{\theta}),\\ \Rightarrow \left.\frac{\partial\hat{\mathcal{L}}_{\boldsymbol{\theta}}}{\partial\theta}\right|_{\mathcal{D}_n} &= \frac{1}{n}\sum_{i=1}^{n}\left.\frac{\partial V}{\partial\mathbf{f}_{\boldsymbol{\theta}}}\frac{\partial\mathbf{f}_{\boldsymbol{\theta}}}{\partial\theta}\right|_{\mathbf{x}_i,\mathbf{y}_i} + \frac{\lambda}{2}\partial_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta}),\\ &= \frac{1}{2n}\sum_{i=1}^{n}\Big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i)-\mathbf{y}_i\Big)^{\top}\left.\frac{\partial\mathbf{f}_{\boldsymbol{\theta}}}{\partial\theta}\right|_{\mathbf{x}_i} + \frac{\lambda}{2}\partial_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta}).\end{aligned}$$

where $\partial_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta})$ is the subgradient of the regulariser and in the last line the square loss risk function is used. The first-order batch gradient descent update for a weight $\theta\in\boldsymbol{\theta}$ is therefore

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t\left.\frac{\partial\hat{\mathcal{L}}_{\boldsymbol{\theta}}}{\partial\theta}\right|_{\mathcal{D}_n}.$$

Stochastic Gradient Descent

The batch setting is however is computationally expensive as it requires the use of the entire training set to evaluate the gradient step at each t^{th} iteration. Instead stochastic optimisation techniques can drastically decrease computational costs by estimating gradient steps with a subset of samples drawn i.i.d from the the training data. A concise summary can be found in Murphy (2012, sec 8.5.2). At the extreme is *stochastic gradient descent* which only uses one sample drawn from $(\mathbf{x},\mathbf{y})\sim\mathcal{D}_n$ to give an unbiased estimate of the gradient at each t^{th} update. The expected SGD step of the sample gradient is the full batch gradient evaluated over the entire $\mathcal{D}_n$. Stochastic *minibatch* attempts to reduce the variance of the gradient update by estimating each gradient step with a subset or *minibatch* of samples $\hat{\mathcal{D}}\sim\mathcal{D}_n$ i.e.

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t\left.\frac{\partial\hat{\mathcal{L}}_{\boldsymbol{\theta}}}{\partial\theta}\right|_{\hat{\mathcal{D}}}. \tag{B.70}$$

In the stochastic setting one of the main focusses is in deciding on the schedule of the learning rate η_t that guarantees convergence to a local minimum. Robbins and Monro (1951) identify the following sufficient conditions,

$$\sum_{t=1}^{\infty}\eta_t = \infty, \quad \sum_{t=1}^{\infty}\eta_t^2 < \infty. \tag{B.71}$$

Following Murphy (2012, sec 8.5.2.1)), the simplest schedule that satisfy these conditions is $\eta_t{=}1/t$. Bottou (1998) provides additional analysis and proposes a more configurable schedule $\eta_t = \eta_0(1+at)^{-b}$ (Bottou, 2012, sec 5.2) where each constant is a tuned hyperparameter.

A cannon of work exists in the stochastic setting for modifying regular gradient updates in order to improve algorithm convergence. Momentum (Rumelhart et al.,

1986) (also see Nesterov (1983) which has superior convergence properties than vanilla gradient descent) adds an update term to reduce 'zig-zagging' descent paths, AdaGrad (Duchi et al., 2011) and RMSProp (Hinton et al., 2012) calculate adaptive learning rates for each individual weight. RMSProp divides the learning rate with an exponentially decaying square gradient magnitude for each weight. One of the most common methods currently in use is Adam (Kingma and Ba, 2014) which can be thought of as a combination of momentum and RMSProp. Adam maintains exponentially decaying averages of both gradient and square gradient magnitudes for each weight, used to compute bias-corrected first and second order gradient moments in the form of adaptive learning rates.

B.5.2 Training Neural Networks

A brief summary of backpropagation and the architecture choice used by this thesis is summarised below. Schmidhuber (2015) provides a deeper historical summary.

A Brief History of Artificial Neural Networks

Dreyfus (1973) details one of the first explicit examples of applying backprop (see below) to adjust parameters in a function approximator to minimise a loss function. But it was Werbos (1982) who first explicitly deployed backprop to train neural network architectures, Rumelhart et al. (1986) noting that backprop allows representation learning and later LeCun et al. (1998) in the *gradient*-based risk minimisation setting. Only up until very recently have neural networks been notoriously difficult to reliably train with only limited techniques (Lecun, 1998) available. One particular problem was that gradient update signals became vanishingly small when activation functions became saturated (Glorot and Bengio, 2010), thus effectively sending learning rates to zero.

Recent research has mitigated the vanishing gradient problem by techniques such as batch normalisation (Ioffe and Szegedy, 2015) and identifying activation functions known as rectified linear units (ReLUs) (Nair and Hinton, 2010) that are not vulnerable to saturation (Glorot et al., 2011; Maas et al., 2013). Together with recent Adam (Kingma and Ba, 2014) adaptive learning rates, the ReLU-Adam architecture choice is vastly more practical for a wide range of supervised learning problems than previous configurations. Practical deep learning fully entered the scene when image recognition and classification competitions started to be regularly won by hardware-accelerated *deep* convolutional network architectures (Krizhevsky et al., 2012), that although have no convergence guarantees, they frequently outperform *shallow* function approximators like RKHS methods such as SVMs (Schölkopf and Smola, 2002). Deep, highly paramet-

ric architectures are cited as having the ability to learn rich data-driven representations (Bengio et al., 2013) and scaling well with abundant data.

Backprop

Using the notation from before, we express the function approximator of multiple composite functions characterised by multiple layers of a weighted sum $\mathbf{h}^\ell(\mathbf{x}) := \mathbf{W}^\ell \boldsymbol{\sigma}^{\ell-1}(\mathbf{x})$ operation (where $\mathbf{h}^1(\mathbf{x}) := \mathbf{W}^1\mathbf{x}$) and a non-linearity $\boldsymbol{\sigma}^\ell(\mathbf{x}) := \boldsymbol{\sigma}^\ell \circ \mathbf{h}^\ell(\mathbf{x})$, where ℓ is the layer index,

$$\begin{aligned}\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) &:= \boldsymbol{\sigma}^L \circ \mathbf{h}^L \circ \boldsymbol{\sigma}^{L-1} \circ ... \circ \boldsymbol{\sigma}^\ell \circ \mathbf{h}^\ell \circ \boldsymbol{\sigma}^{\ell-1} \circ ... \circ \boldsymbol{\sigma}^1 \circ \mathbf{h}^1(\mathbf{x}),\\ &= \boldsymbol{\sigma}^L \circ \mathbf{h}^L \circ \boldsymbol{\sigma}^{L-1} \circ ... \circ \boldsymbol{\sigma}^\ell \circ \mathbf{h}^\ell \circ \boldsymbol{\sigma}^{\ell-1}(\mathbf{x}),\\ &= \boldsymbol{\sigma}^L \circ \mathbf{h}^L \circ \boldsymbol{\sigma}^{L-1} \circ ... \circ \boldsymbol{\sigma}^\ell \circ \mathbf{h}^\ell(\mathbf{x}),\\ &=: \boldsymbol{\sigma}^L(\mathbf{x}).\end{aligned}$$

This notation specifies $\mathbf{h}^\ell(\mathbf{x})$ as the output layer of function $\mathbf{h}^\ell$ and $\boldsymbol{\sigma}^\ell(\mathbf{x})$ as the output layer of function $\boldsymbol{\sigma}^\ell$, both of which are evaluated when the input to $\mathbf{f}_{\boldsymbol{\theta}}$ is $\mathbf{x}$. For a minibatch $\hat{\mathcal{D}}$ then the empirical loss[4] for the neural function approximator is

$$\hat{\mathcal{L}}_{\boldsymbol{\theta}}|_{\hat{\mathcal{D}}} := \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|} V(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i).$$

The derivative of the loss wrt. the ℓ^{th} layer's $w_{kj} \in \mathbf{W}^\ell$ is calculated using Jacobian multiplication which derives from the chain rule of composite vector-valued functions,

$$\begin{aligned}&\frac{\partial \hat{\mathcal{L}}_{\boldsymbol{\theta}}}{\partial w_{kj}^\ell}\bigg|_{\hat{\mathcal{D}}}\\ &= \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|} \frac{\partial V(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)}{\partial \mathbf{f}_{\boldsymbol{\theta}}}\bigg|_{\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i)} \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \mathbf{h}^\ell}\bigg|_{\mathbf{h}^\ell(\mathbf{x}_i)} \frac{\partial \mathbf{h}^\ell}{\partial w_{kj}^\ell}\bigg|_{\boldsymbol{\sigma}^{\ell-1}(\mathbf{x}_i)},\\ &= \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|} \frac{\partial V}{\partial \boldsymbol{\sigma}^L}\bigg|_{\boldsymbol{\sigma}^L(\mathbf{x}_i)} \frac{\partial \boldsymbol{\sigma}^L}{\partial \mathbf{h}^L}\bigg|_{\mathbf{h}^L(\mathbf{x}_i)} \frac{\partial \mathbf{h}^L}{\partial \boldsymbol{\sigma}^{L-1}}\bigg|_{\boldsymbol{\sigma}^{L-1}(\mathbf{x}_i)} \cdots \frac{\partial \mathbf{h}^{\ell+1}}{\partial \boldsymbol{\sigma}^\ell}\bigg|_{\boldsymbol{\sigma}^\ell(\mathbf{x}_i)} \frac{\partial \boldsymbol{\sigma}^\ell}{\partial \mathbf{h}^\ell}\bigg|_{\mathbf{h}^\ell(\mathbf{x}_i)} \frac{\partial \mathbf{h}^\ell}{\partial w_{kj}^\ell}\bigg|_{\boldsymbol{\sigma}^{\ell-1}(\mathbf{x}_i)},\\ &= \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|} \frac{\partial V}{\partial \mathbf{h}^\ell}\bigg|_{\mathbf{h}^\ell(\mathbf{x}_i)} \frac{\partial \mathbf{h}^\ell}{\partial w_{kj}^\ell}\bigg|_{\boldsymbol{\sigma}^{\ell-1}(\mathbf{x}_i)},\end{aligned} \tag{B.72}$$

where the last term is a zero vector of size $\dim(\mathbf{h}^\ell(\mathbf{x}_i))$ apart from its k^{th} element,

$$\left(\frac{\partial \mathbf{h}^\ell}{\partial w_{kj}^\ell}\bigg|_{\boldsymbol{\sigma}^{\ell-1}(\mathbf{x}_i)}\right)_k = \sigma_j^{\ell-1}(\mathbf{x}_i).$$

[4] For simplicity the regularisation term is ignored, however reintroducing it requires additional considerations as described in the next section

The gradient of the loss function (evaluated for a minibatch) wrt. the entire $\mathbf{W}^\ell$ matrix is therefore

$$\nabla_{\mathbf{W}^\ell}\hat{\mathcal{L}}\Big|_{\hat{\mathcal{D}}} = \frac{1}{2|\hat{\mathcal{D}}|}\sum_{i=1}^{|\hat{\mathcal{D}}|}\left(\frac{\partial V}{\partial \mathbf{h}^\ell}\Big|_{\mathbf{h}^\ell(\mathbf{x}_i)}\right)^\top \boldsymbol{\sigma}^{\ell-1\top}(\mathbf{x}_i), \tag{B.73}$$

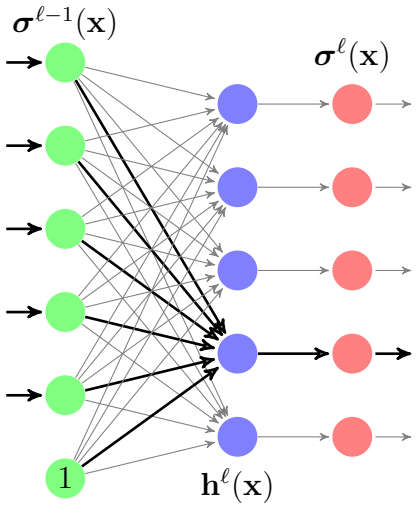
where $\boldsymbol{\sigma}^0(\mathbf{x}_i) := \mathbf{x}_i$. For every $\mathbf{x}_i$, it would be inefficient to evaluate each Jacobian *separately* in equation (B.72) during the calculation of equation (B.73). Instead, backpropagation efficiently calculates $\partial V/\partial \mathbf{h}^\ell$ from $\partial V/\partial \mathbf{h}^{\ell+1}$,

$$\begin{aligned}
&\frac{\partial V}{\partial \mathbf{h}^{\ell+1}}\Big|_{\mathbf{h}^{\ell+1}(\mathbf{x}_i)}\frac{\partial \mathbf{h}^{\ell+1}}{\partial \boldsymbol{\sigma}^\ell}\Big|_{\boldsymbol{\sigma}^\ell(\mathbf{x}_i)}\frac{\partial \boldsymbol{\sigma}^\ell}{\partial \mathbf{h}^\ell}\Big|_{\mathbf{h}^\ell(\mathbf{x}_i)} = \frac{\partial V}{\partial \mathbf{h}^\ell}\Big|_{\mathbf{h}^\ell(\mathbf{x}_i)}, \\
\implies &\frac{\partial V}{\partial \mathbf{h}^{\ell+1}}\Big|_{\mathbf{h}^{\ell+1}(\mathbf{x}_i)}\mathbf{W}^{\ell+1}\frac{\partial \boldsymbol{\sigma}^\ell}{\partial \mathbf{h}^\ell}\Big|_{\mathbf{h}^\ell(\mathbf{x}_i)} = \frac{\partial V}{\partial \mathbf{h}^\ell}\Big|_{\mathbf{h}^\ell(\mathbf{x}_i)},
\end{aligned}$$

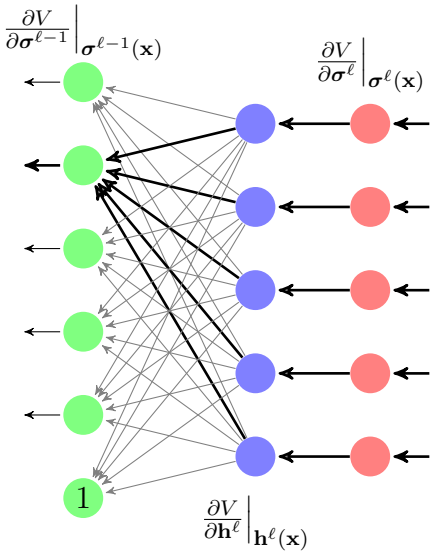
so that equation (B.73) can be efficiently calculated one layer after another in the backwards pass.

Figure B.5 Feedforward neural architecture at the ℓ^{th} layer, with bold lines illustrating information flow.

i) Forward inference.



ii) Backpropagate error signal.



Activation Functions

Softmax For any layer ℓ and positive integer m, then each j^{th} element of $\boldsymbol{\sigma}^\ell(\mathbf{x}) \in \mathbb{R}^m$ is a softmax function,

$$\sigma_j^\ell(\mathbf{x}) := \frac{\exp(h_j^\ell(\mathbf{x}))}{\sum_{i=1}^m \exp(h_i^\ell(\mathbf{x}))}.$$

Typically, including architectures used in this thesis, the softmax is used as the last L^{th} activation function. In this case the computational complexity of the derivative is linear in m *if* the derivative is combined with the loss function derivative (Martins and Astudillo, 2016),

$$\left.\frac{\partial V}{\partial \boldsymbol{\sigma}^{\ell}}\right|_{\boldsymbol{\sigma}^{L}(\mathbf{x})} \left.\frac{\partial \boldsymbol{\sigma}^{L}}{\partial \mathbf{h}^{L}}\right|_{\mathbf{h}^{L}(\mathbf{x})} = \boldsymbol{\sigma}^{L}(\mathbf{x}) \odot (\mathbf{v} - \bar{\mathbf{v}}\mathbf{1}) \in \mathbb{R}^{m},$$

where $\mathbf{v} := \left(\left.\frac{\partial V}{\partial \boldsymbol{\sigma}^{L}}\right|_{\boldsymbol{\sigma}^{L}(\mathbf{x})}\right)^{\top}$ and $\bar{\mathbf{v}} := \sum_{j=1}^{m} \sigma_j^{\ell}(\mathbf{x}) v_j$.

ReLU All other activation functions are set as ReLUs,

$$\sigma_j^{\ell}(\mathbf{x}) := \max[h_j^{\ell}(\mathbf{x}), 0],$$

whose Jacobian is

$$\left.\frac{\partial \boldsymbol{\sigma}^{\ell}}{\partial \mathbf{h}^{\ell}}\right|_{\mathbf{h}^{\ell}(\mathbf{x})} = \operatorname{diag}\left(\left.\frac{\partial \sigma_1^{\ell}}{\partial h_1^{\ell}}\right|_{h_1^{\ell}(\mathbf{x})}, ..., \left.\frac{\partial \sigma_m^{\ell}}{\partial h_m^{\ell}}\right|_{h_m^{\ell}(\mathbf{x})}\right) \in \mathbb{R}^{m \times m},$$

where

$$\left.\frac{\partial \sigma_j^{\ell}}{\partial h_j^{\ell}}\right|_{h_j^{\ell}(\mathbf{x})} = \begin{cases} 1 & h_j^{\ell}(\mathbf{x}) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

B.5.3 Weight Sparsity in the Stochastic Setting

Pruning parameters from neural networks has been shown to not only improve computational/memory costs, but also function approximator generalisation. Optimal brain damage LeCun et al. (1990) use second-order derivative information to rank parameters by their saliency wrt. to the loss function. This approach requires an existing trained network that is then sparsifies rather than maintain structured sparsity as the network is being trained. Other methods use mixed norm approaches to induce sparsity in individual layer parameters or structural sparsity in groups of parameters (Kong et al., 2014; Lebedev and Lempitsky, 2016; Wen et al., 2016; Yoon and Hwang, 2017). Such approaches implement either sub-gradient descent or stochastic proximal gradient methods during network training (see Appendix B.4.4 for the deterministic setting).

Truncated Gradient for Online Lasso Regression

The following summarises existing work that is modified in Chapter 6 to induce structural sparsity in a deep CME. Langford et al. (2009) focus on solving the batch

lasso (cf. equation (B.26))

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^d} \Big[\mathcal{L}(\boldsymbol{\beta}) + \lambda||\boldsymbol{\beta}||_1\Big], \tag{B.74}$$

where $\mathcal{L}(\boldsymbol{\beta}) := \frac{1}{2n}||\mathbf{X}\boldsymbol{\beta} - \mathbf{y}||_2^2$ is a differentiable convex loss function formed from $\mathcal{D} := \{(\mathbf{x}, y)_i\}_{i=1}^n$ and linearly independent features of $\mathbf{x}$ are provided a priori. The aim is such that at any iteration, $\boldsymbol{\beta}$ is sparse with k non-zero entries such that the goal is to incur a cost linear in k and independent of d for each iteration. The naive approach is to simply use subgradient descent

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta\nabla_{\boldsymbol{\beta}}\hat{\mathcal{L}}\Big|_{\hat{\mathcal{D}}} - \eta\lambda\partial_{\boldsymbol{\beta}}||\boldsymbol{\beta}||_1,$$

where $\hat{\mathcal{L}}$ is the loss over a minibatch $\hat{\mathcal{D}} \sim \mathcal{D}$. For element ℓ, the subgradient is the piecewise function

$$\Big[\partial_{\boldsymbol{\beta}}||\boldsymbol{\beta}||_1\Big]_\ell \in \begin{cases} \text{sign}(\beta_\ell) & \beta_\ell \neq 0, \\ [-1, 1] & \beta_\ell = 0. \end{cases} \tag{B.75}$$

Usually practitioners ignore the ambiguous gradient at $\beta_\ell = 0$, however stochastic gradient descent will not produce sparse solutions as it is almost impossible for a β estimate be driven to zero when using floating point arithmetic. Another approach is to iteratively take an SGD step with an additional mapping $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}^d$,

$$\boldsymbol{\beta} \leftarrow \mathcal{T}\left(\boldsymbol{\beta} - \eta\nabla_{\boldsymbol{\beta}}\hat{\mathcal{L}}\Big|_{\hat{\mathcal{D}}}, \boldsymbol{\omega}\right), \tag{B.76}$$

where $\boldsymbol{\omega}$ is a set of hyperparameters. A naive choice for $\mathcal{T} = \mathcal{T}_{\text{round}}$ is to round the gradient steps to zero if they fall within a value ω i.e.

$$\Big[\mathcal{T}_{\text{round}}(\mathbf{u}, \omega)\Big]_\ell = \begin{cases} 0 & |u_\ell| \leq \omega, \\ u_\ell & \text{otherwise.} \end{cases} \tag{B.77}$$

The problem with this approach is that the rounding is a hard threshold and may send too many weights to zero as they pass through $[-\omega, \omega]$. On the other hand, gradient descent may not actively drive weights to this range thus not inducing sparsity. Overall the algorithm is very sensitive to choice of ω. A final approach would be to set $\mathcal{T} = \mathcal{T}_{\text{soft}}$ as the soft thresholding operator in equation (B.59) i.e.

$$\Big[\mathcal{T}_{\text{soft}}(\mathbf{u}, \eta\lambda)\Big]_\ell = \Big[S^1_{\eta\lambda}(\mathbf{u})\Big]_\ell = \Big[|u_\ell| - \eta\lambda\Big]_+ \text{sign}(u_\ell),$$

which is equivalent to a stochastic variant of the iterative shrinkage methods as described in Appendix B.4.4. However Langford et al. (2009) point out that even the soft thresholding method may be too aggressive and drive all weights to zero over the course of the optimisation. Langford et al. (2009) instead combine both $\mathcal{T}_{\text{soft}}$ and $\mathcal{T}_{round}$

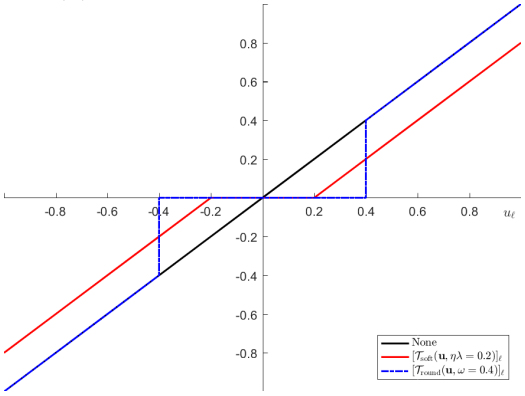
into the truncated $\mathcal{T}_{\text{trunc}}$ by only applying the soft-thresholding in a range $[-\omega, \omega]$,

$$\left[\mathcal{T}_{\text{trunc}}(\mathbf{u}, \eta\lambda, \omega)\right]_\ell = \begin{cases} \left[|u_\ell| - \eta\lambda\right]_+ \text{sign}(u_\ell) & |u_\ell| \leq \omega, \\ u_\ell & \text{otherwise.} \end{cases} \tag{B.78}$$

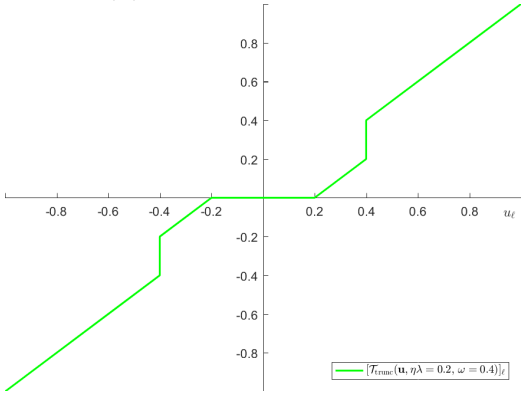
It is claimed that not only will this create sparsity but it restricts driving parameters to zero to a small region around zero. Note that as $\omega \to \infty$ then $\mathcal{T}_{\text{trunc}} \to \mathcal{T}_{\text{soft}}$. All $\mathcal{T}$ variants are be visualised in fig. B.6.

Figure B.6 Individual weight shrinkage

(a) Previous Shrinkage Methods



(b) Truncated Shrinkage



B.6 Matrix Identities

B.6.1 Cook Book

The following identities are used throughout this thesis and are taken from the *Matrix Cookbook* (Petersen and Pedersen, 2012).

$$\text{No. 85:} \quad \nabla_{\mathbf{w}} \mathbf{w}^\top \mathbf{B}\mathbf{w} = 2\mathbf{B}\mathbf{w} \tag{B.79}$$

$$\text{No. 115:} \quad \nabla_{\mathbf{W}} \text{Tr}[\mathbf{W}^\top \mathbf{W}] = \nabla_{\mathbf{W}} \text{Tr}[\mathbf{W}\mathbf{W}^\top] = 2\mathbf{W} \tag{B.80}$$

$$\text{No. 118:} \quad \nabla_{\mathbf{W}} \text{Tr}[\mathbf{W}\mathbf{X}\mathbf{W}^\top \mathbf{Z}] = 2\mathbf{Z}\mathbf{W}\mathbf{X} \tag{B.81}$$

$$\text{No. 119:} \quad \nabla_{\mathbf{W}} \text{Tr}[(\mathbf{X}\mathbf{W}\mathbf{Z} - \mathbf{Y})^\top (\mathbf{X}\mathbf{W}\mathbf{Z} - \mathbf{Y})] = 2\mathbf{X}^\top (\mathbf{X}\mathbf{W}\mathbf{Z} - \mathbf{Y})\mathbf{Z}^\top \tag{B.82}$$

Appendix C

External Work

C.1 CCME Supplemental

The following supplemental summarises contributions external from this thesis made by colleagues in Lever et al. (2016). These contributions are integral to the final operation of the CCME algorithm. However this thesis does contribute to the overall experimentation of the final algorithm in multiple control tasks, therefore a description of this external work is critical. This work is also an inspiration to this thesis' final chapter where the compression set is maintained in an alternative way in the SGD setting.

C.1.1 Compression Set

At the k^{th} policy iteration, a CME is defined over the entire set of successor states $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n$ seen thus far in data $\mathcal{D}_n$,

$$\hat{\mu}(\mathbf{s}, \mathbf{a}) = \sum_{j=1}^{n} \alpha_j(\mathbf{s}, \mathbf{a}) \boldsymbol{\phi}(\mathbf{s}'_j) \in \mathcal{F}, \quad (\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}. \tag{C.1}$$

If $||\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})||_1 \leq 1$ for all $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$ then this is known as a *proper* CME. If n grows uncontrollably, so does i) the cost of training the embedding and ii) the cost of policy evaluation in the finite-induced MDP. The *compressed* CME (CCME) approximates the embedding over a compressed basis $\mathcal{C} := \{\boldsymbol{\phi}(\mathbf{c}_i)\}_{i=1}^m$. The following theory establishes that a compressed embedding can approximate the uncompressed embedding within a tolerance defined by the norm in $\mathcal{F}$ by choosing $\mathcal{C}$ with algorithm 26.

Lemma C.1. *Given a proper CME $\hat{\mu}$, $\mathcal{S}' := \{\mathbf{s}'_i\}_{i=1}^n$ from data $\mathcal{D}_n$ and any small error δ, suppose there exists a compression set $\mathcal{C} = \{\boldsymbol{\phi}(\mathbf{c}_j)\}_{j=1}^m$ such that a basis $\mathbf{b}(\mathbf{s}'_j) \in \mathbb{R}^m$*

exists that can approximate $\boldsymbol{\phi}(\mathbf{s}'_j)$,

$$||\sum_{i=1}^{m} b_i(\mathbf{s}'_j)\phi(\mathbf{c}_i) - \phi(\mathbf{s}'_j)||_{\mathcal{F}} \leq \delta, \quad \forall \mathbf{s}'_j \in \mathcal{S}', \tag{C.2}$$

where the constraint $||\mathbf{b}(\mathbf{s}'_j)||_1 \leq 1$ *holds. In addition, if we define* $\alpha_i^{CCME}(\mathbf{s},\mathbf{a}) := \sum_{j=1}^{n} b_i(\mathbf{s}'_j)\alpha_j(\mathbf{s},\mathbf{a})$, *then*

i)

$$\sum_{i=1}^{m} |\alpha_i^{CCME}(\mathbf{s},\mathbf{a})| \leq 1,$$

and ii) if $\hat{\mu}^{CCME}(\cdot) := \sum_{i=1}^{m} \alpha_i^{CCME}(\cdot)\boldsymbol{\phi}(\mathbf{c}_i)$ *then*

$$||\hat{\mu}(\mathbf{s},\mathbf{a}) - \hat{\mu}^{CCME}(\mathbf{s},\mathbf{a})||_{\mathcal{F}} \leq \delta, \qquad \forall (\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}.$$

Proof. i)

$$\begin{aligned}
\sum_{i=1}^{m} |\alpha_i^{\text{CCME}}(\mathbf{s},\mathbf{a})| &= \sum_{i=1}^{m} |\sum_{j=1}^{n} b_i(\mathbf{s}'_j)\alpha_j(\mathbf{s},\mathbf{a})|, \\
&\leq \sum_{j=1}^{n} |\alpha_j(\mathbf{s},\mathbf{a})| \sum_{i=1}^{m} |b_i(\mathbf{s}'_j)|, \\
&\leq 1.
\end{aligned}$$

ii) For all $(\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}$,

$$\begin{aligned}
||\hat{\mu}(\mathbf{s},\mathbf{a}) - \hat{\mu}^{\text{CCME}}(\mathbf{s},\mathbf{a})||_{\mathcal{F}} &= ||\sum_{j=1}^{n} \alpha_j(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}'_j) - \sum_{i=1}^{m} \alpha_i^{\text{CCME}}(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{c}_i)||_{\mathcal{F}}, \\
&= ||\sum_{j=1}^{n} \alpha_j(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}'_j) - \sum_{i=1}^{m}\sum_{j=1}^{n} b_i(\mathbf{s}'_j)\alpha_j(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{c}_i)||_{\mathcal{F}}, \\
&= ||\sum_{j=1}^{n} \alpha_j(\mathbf{s},\mathbf{a})\Big(\boldsymbol{\phi}(\mathbf{s}'_j) - \sum_{i=1}^{m} b_i(\mathbf{s}'_j)\boldsymbol{\phi}(\mathbf{c}_i)\Big)||_{\mathcal{F}}, \\
&\leq \sum_{j=1}^{n} |\alpha_j(\mathbf{s},\mathbf{a})| \max_j ||\boldsymbol{\phi}(\mathbf{s}'_j) - \sum_{i=1}^{m} b_i(\mathbf{s}'_j)\boldsymbol{\phi}(\mathbf{c}_i)||_{\mathcal{F}}, \\
&\leq \delta. \quad \square
\end{aligned}$$

□

Therefore if condition C.2 is satisfied, then a δ-lossy CCME is guaranteed.

Theorem C.1.1. *Algorithm algorithm 26 with* $\mathcal{C}$ *initialised to* $\emptyset$ *returns a* δ*-lossy compression set of* $\mathcal{S}'$.

Proof: Condition C.2 in lemma C.1 must be satisfied to guarantee a δ*-lossy CCME*

embedding. By construction algorithm 26 solves

$$\max_{1\leq j\leq n}\min_{\mathbf{b}\in\mathbb{R}^m,\,||\mathbf{b}||_1\leq 1}\left[||\sum_{i=1}^{m} b_i\boldsymbol{\phi}(\mathbf{c}_i)-\boldsymbol{\phi}(\mathbf{s}'_j)||_{\mathcal{F}}\right], \tag{C.3}$$

which satisfies this condition, such that the returned set is the δ-lossy compression set. □

The compression algorithm takes each new data point, projects it onto a sparse basis of existing compression points using a constrained optimisation algorithm such as lasso. If the error is over a threshold δ, the successor state is added to the compression set.

Algorithm 26 AugmentCompressionSet(L, $\mathcal{C}$, $\mathcal{S}'$, δ)

1: **Input:** State kernel $L:\mathcal{S}\times\mathcal{S}\to\mathbb{R}$ defining implicit feature map $\boldsymbol{\phi}(\mathbf{s}):=L(\mathbf{s},\cdot)$, existing compression set $\mathcal{C}=\{\boldsymbol{\phi}(\mathbf{c}_1),...,\boldsymbol{\phi}(\mathbf{c}_m)\}$, candidate states $\mathcal{S}'=\{\mathbf{s}'_1,...,\mathbf{s}'_n\}$, tolerance δ.
2: **Output:** Augmented compression set $\mathcal{C}$.
3: **for** $j=1$ **to** n **do**
4: **if** $\min_{\mathbf{b}\in\mathbb{R}^m,||\mathbf{b}||_1\leq 1}||\sum_{i=1}^{m} b_i\boldsymbol{\phi}(\mathbf{c}_i)-\boldsymbol{\phi}(\mathbf{s}'_j)||_{\mathcal{F}}>\delta$ **then**
5: $\mathcal{C}\leftarrow\mathcal{C}\cup\boldsymbol{\phi}(\mathbf{s}'_j)$, $m\leftarrow m+1$ ▷ Augment compression set.
6: **end if**
7: **end for**
8: **return** $\mathcal{C}$

This algorithm is guaranteed to find the compression set for a proper ($||\boldsymbol{\alpha}(s,a)||_1\leq 1$) CME.

Corollary C.1.1 (Performance Guarantee)**.** *Given a proper CME $\hat{\mu}$ and proper CCME $\hat{\mu}^{CCME}$, if*

$$\sup_{(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}}\left[||\hat{\mu}(\mathbf{s},\mathbf{a})-\hat{\mu}^{CCME}(\mathbf{s},\mathbf{a})||_{\mathcal{F}}\right]\leq\delta,$$

and if the policy improvement suboptimality bound in theorem 4 is defined as $B_k(\tilde{v}^*)$, then at the $k=\kappa$ iteration, policy improvement suboptimality using the CCME is upper bounded by

$$||v^{\hat{\pi}_\kappa}-v^*||_\infty\leq B_\kappa(\tilde{v}^*)+\frac{2\gamma}{(1-\gamma)^2}\delta||\tilde{v}^*||_{\mathcal{F}}.$$

By using the compressed embedding for value iteration, the original policy improvement suboptimality bound for a proper vanilla CME is only worsened by $\frac{2\gamma}{(1-\gamma)^2}\delta||\tilde{v}^*||_{\mathcal{F}}$.

Implementation

Gaussian kernel matrices are positive definite and therefore they have a unique symmetric square root. The kernel matrix over the compression set $\mathcal{C}$ and candidate

state $\mathbf{s}'$,

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{\mathcal{C}\mathcal{C}} & \mathbf{L}_{\mathcal{C}\mathbf{s}'} \\ \mathbf{L}_{\mathbf{s}'\mathcal{C}} & \mathbf{L}_{\mathbf{s}'\mathbf{s}'} \end{pmatrix},$$

has a square root of the form $\mathbf{L}^{\frac{1}{2}} = [\tilde{\mathbf{\Phi}}^\top \tilde{\boldsymbol{\phi}}(\mathbf{s}')] = [\tilde{\mathbf{\Phi}}; \tilde{\boldsymbol{\phi}}^\top(\mathbf{s}')] \in \mathbb{R}^{m+1 \times m+1}$ where $\mathbf{L}_{\mathcal{C}\mathcal{C}} \approx \tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^\top \in \mathbb{R}^{m \times m}$, $\mathbf{L}_{\mathcal{C}\mathbf{s}'} \approx \tilde{\mathbf{\Phi}}\tilde{\boldsymbol{\phi}}(\mathbf{s}') \in \mathbb{R}^m$ and $\mathbf{L}_{\mathbf{s}'\mathbf{s}'} \approx \tilde{\boldsymbol{\phi}}^\top(\mathbf{s}')\tilde{\boldsymbol{\phi}}(\mathbf{s}') \in \mathbb{R}$. The constrained minimisation in lemma 26 can therefore be implemented as a lasso (Tibshirani, 1996) optimisation problem

$$\hat{\mathbf{b}} = \min_{\mathbf{b} \in \mathbb{R}^m, \, ||\mathbf{b}||_1 \leq 1} \Big[||\mathbf{X}\mathbf{b} - \mathbf{y}||_2 \Big],$$

with design matrix $\mathbf{X} := \tilde{\mathbf{\Phi}}^\top$ and target $\mathbf{y} := \tilde{\boldsymbol{\phi}}(\mathbf{s}')$.

C.1.2 Contraction Constraint

The following projection can be achieved by a lasso (Tibshirani, 1996) optimisation where for each $(\mathbf{s}, \mathbf{a})$ presented to the model, the projection is

$$\begin{aligned} \text{Proj}\Big(\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})\Big) &= \min_{\boldsymbol{\beta} \in \mathbb{R}^m, \, ||\boldsymbol{\beta}||_1 \leq 1} \Big[|| \sum_{j=1}^{m} \alpha_j(\mathbf{s}, \mathbf{a}) \boldsymbol{\phi}(\mathbf{s}'_j) - \sum_{j=1}^{m} \beta_j \boldsymbol{\phi}(\mathbf{s}_j) ||_{\mathcal{F}} \Big], \\ &= \min_{\boldsymbol{\beta} \in \mathbb{R}^m, \, ||\boldsymbol{\beta}||_1 \leq 1} \Big[||\mathbf{\Phi}^\top \boldsymbol{\alpha}(\mathbf{s}, \mathbf{a}) - \mathbf{\Phi}^\top \boldsymbol{\beta}||_{\mathcal{F}} \Big]. \end{aligned} \tag{C.4}$$

Implementation

An incomplete Cholesky decomposition (see section B.4.2) of the kernel matrix $\mathbf{L}_{\mathcal{C}\mathcal{C}} \approx \tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^\top$ is made to approximate $\mathbf{\Phi}^\top \approx \tilde{\mathbf{\Phi}}^\top \in \mathbb{R}^{m_{\text{chol}} \times m}$ where the constant $m_{\text{chol}} << m$ is the dimensionality of the underlying orthonormal basis that each $\boldsymbol{\phi}(\mathbf{s}')$ is decomposed by. The projection (C.4) is therefore approximated by

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \text{LassoSparse}\Big(\boldsymbol{\alpha}(\mathbf{s}, \mathbf{a})\Big), \\ &= \min_{\boldsymbol{\beta} \in \mathbb{R}^m, ||\boldsymbol{\beta}||_1 \leq 1} \Big[||\tilde{\mathbf{\Phi}}^\top \boldsymbol{\alpha}(\mathbf{s}, \mathbf{a}) - \tilde{\mathbf{\Phi}}^\top \boldsymbol{\beta}||_2 \Big]. \end{aligned} \tag{C.5}$$

The Cholesky decomposition is used to limit the computational complexity of the lasso minimisation problem, which is performance critical as it is executed every time the embedding is evaluated. In a similar way that the L1-projection (in section B.4.5) was used to induce $\boldsymbol{\alpha}$ sparsity for the CME, LassoSparse performs a lasso optimisation on $\boldsymbol{\alpha}$ to find the projected $\hat{\boldsymbol{\beta}}$ constrained by the L1-norm. Note that the $\hat{\boldsymbol{\beta}}$ is normalised with the L1-norm to ensure it lies on the unit ball such that it satisfies the pseudo-MDP contraction constraint.