

---

---

# Towards Scalable Meta-Learning

---

---

A thesis submitted to  
the University of Manchester  
for the degree of Doctor of Philosophy  
in the Faculty of Humanities

2021

Sebastian Flennerhag  
School of Social Sciences

Supervisors: Prof. Mark Elliot  
Prof. John Keane  
Prof. Hujun Yin

Internal reviewer: Dr. Xiao-Jun Zeng

Examiners: Dr. Joaquin Vanschoren  
Dr. Richard Allmendinger

# Contents

<b>I</b>	<b>Declaration of Authorship</b>	<b>9</b>
<b>II</b>	<b>Copyright</b>	<b>10</b>
<b>III</b>	<b>Abstract</b>	<b>12</b>
<b>IV</b>	<b>Acknowledgements</b>	<b>13</b>
<b>V</b>	<b>Abbreviations</b>	<b>14</b>
<b>VI</b>	<b>Notation</b>	<b>15</b>
<b>VII</b>	<b>Publications</b>	<b>17</b>
<b>1</b>	<b>Introduction</b>	<b>18</b>
1.1	Limitations of Tabula Rasa . . . . .	20
1.2	Limitations of Contemporary Meta-Learning . . . . .	23
1.2.1	Scalability . . . . .	24
1.2.2	Generalisation . . . . .	26
1.2.3	Exogenous Tasks . . . . .	28
1.3	Thesis Outline . . . . .	29
<b>I</b>	<b>Foundations</b>	<b>31</b>
<b>2</b>	<b>Machines that Learn</b>	<b>32</b>
2.1	Types of Machine Learning . . . . .	32
2.2	Machine Learning Problem Definitions . . . . .	35
2.2.1	Maximum Likelihood Estimation . . . . .	36
2.2.2	Empirical Risk Minimization . . . . .	37
2.3	Learning Through Stochastic Gradient Descent . . . . .	39
2.4	Reinforcement Learning . . . . .	41
2.4.1	The Reinforcement Learning Problem . . . . .	41
2.4.2	Generalised Policy Iteration . . . . .	44
2.4.3	Reinforcement Learning with Policy Gradients . . . . .	46
2.5	Summary . . . . .	48

<b>3</b>	<b>Machines that Learn to Learn</b>	<b>49</b>
3.1	The Meta-Learning Problem . . . . .	49
3.2	Related Fields . . . . .	52
3.3	A Historical Perspective . . . . .	54
3.4	The Mechanics of Meta-Learning . . . . .	57
3.5	Contemporary Meta-Learning with Neural Networks . . . . .	59
3.6	Summary . . . . .	63
<b>II</b>	<b>Learning to Learn</b>	<b>65</b>
<b>4</b>	<b>Learning to Dynamically Adapt</b>	<b>66</b>
	Breaking the Activation Function Bottleneck through Adaptive Parameterisation . . . . .	68
4.1	Introduction . . . . .	68
4.2	Adaptive Parameterization . . . . .	69
4.2.1	The Adaptive Feed-Forward Layer . . . . .	71
4.2.2	Adaptation Modules . . . . .	71
4.3	Adaptive Parameterization in RNNs . . . . .	73
4.4	Related Work . . . . .	75
4.5	Experiments . . . . .	76
4.5.1	Extreme Tail Regression . . . . .	76
4.5.2	MNIST . . . . .	77
4.5.3	Penn Treebank . . . . .	77
4.5.4	WikiText-2 . . . . .	79
4.5.5	Ablation Study . . . . .	79
4.5.6	Robustness . . . . .	80
4.6	Conclusions . . . . .	81
	<b>Appendix</b>	<b>83</b>
4.A	NLP Experiment Hyper-Parameters . . . . .	83
<b>5</b>	<b>Scaling up Meta-Learning On First Principles</b>	<b>84</b>
	Transferring Knowledge across Learning Processes . . . . .	86
5.1	Introduction . . . . .	86
5.2	Transferring Knowledge across Learning Processes . . . . .	87
5.2.1	Gradient Paths on Task Manifolds . . . . .	88
5.2.2	Meta Learning across Task Manifolds . . . . .	90
5.2.3	Leap . . . . .	92
5.3	Related Work . . . . .	95
5.4	Empirical Results . . . . .	97

5.4.1	Omniglot . . . . .	97
5.4.2	Multi-CV . . . . .	98
5.4.3	Atari . . . . .	100
5.5	Conclusions . . . . .	101
<b>Appendix</b>		<b>102</b>
5.A	Mathematical Results . . . . .	102
5.B	Ablation Study: Approximating Jacobians . . . . .	106
5.C	Ablation Study: Leap Hyper-Parameters . . . . .	107
5.D	Experiment Details: Omniglot . . . . .	108
5.E	Experiment Details: Multi-CV . . . . .	111
5.F	Experiment Details: Atari . . . . .	113
<b>6</b>	<b>General-Purpose Meta-Learning</b>	<b>116</b>
	Meta-Learning with Warped Gradient Descent . . . . .	117
6.1	Introduction . . . . .	117
6.2	Warped Gradient Descent . . . . .	119
6.2.1	Gradient-Based Meta-Learning . . . . .	119
6.2.2	General-Purpose Preconditioning . . . . .	121
6.2.3	The Geometry of Warped Gradient Descent . . . . .	123
6.2.4	Meta-Learning Warp Parameters . . . . .	124
6.2.5	Integration with Learned Initialisations . . . . .	126
6.3	Related Work . . . . .	127
6.4	Experiments . . . . .	129
6.4.1	Few-Shot Learning . . . . .	129
6.4.2	Multi-Shot Learning . . . . .	129
6.4.3	Complex Meta-Learning . . . . .	131
6.5	Conclusion . . . . .	133
<b>Appendix</b>		<b>134</b>
6.A	WarpGrad Design Principles for Neural Nets . . . . .	134
6.B	WarpGrad Meta-Training Algorithms . . . . .	135
6.C	WarpGrad Optimisers . . . . .	137
6.D	Synthetic Experiment . . . . .	140
6.E	Omniglot . . . . .	141
6.F	Ablation Study: Layers, Objectives, Algorithms . . . . .	143
6.G	Ablation study: Warped and Natural Gradients . . . . .	144
6.H	<i>mini</i> ImageNet and <i>tiered</i> ImageNet . . . . .	146
6.I	Maze Navigation . . . . .	148
6.J	Meta-Learning for Continual Learning . . . . .	150



<b>III</b>	<b>Towards Never-Ending Learning</b>	<b>157</b>
<b>7</b>	<b>Lifelong Learning in Autonomous Agents</b>	<b>158</b>
	Temporal Difference Uncertainties as a Signal for Exploration . . .	160
	7.1 Introduction . . . . .	161
	7.2 Estimating Value Function Uncertainty . . . . .	162
	7.3 Temporal Difference Uncertainties . . . . .	164
	7.4 Implementing TDU with Bootstrapping . . . . .	167
	7.5 Empirical Evaluation . . . . .	169
	7.5.1 Behaviour Suite . . . . .	169
	7.5.2 Atari . . . . .	172
	7.6 Related Work . . . . .	173
	7.7 Conclusion . . . . .	175
	<b>Appendix</b>	<b>176</b>
	7.A Implementation and Code . . . . .	176
	7.B Proofs . . . . .	179
	7.C Binary Tree MDP . . . . .	187
	7.D Behaviour Suite . . . . .	189
	7.D.1 Agents and Hyper-Parameters . . . . .	189
	7.D.2 TDU Experiments . . . . .	191
	7.E Atari with R2D2 . . . . .	194
	7.E.1 Bootstrapped R2D2 . . . . .	194
	7.E.2 Pre-processing . . . . .	196
	7.E.3 Hyper-Parameter Selection . . . . .	196
	7.E.4 Detailed Results: Main Experiment . . . . .	198
	7.E.5 Full Atari suite . . . . .	199
<b>8</b>	<b>Conclusion</b>	<b>202</b>
	8.1 Thesis Summary . . . . .	202
	8.2 Contributions . . . . .	204
	8.2.1 Scalability . . . . .	204
	8.2.2 Generalisation . . . . .	206
	8.2.3 Exogenous Tasks . . . . .	207
	8.3 Limitations and Future Work . . . . .	208

*Word count: 50,129.*

# List of Figures

3.1	Example . . . . .	58
4.1	Adaptation . . . . .	70
4.2	Extreme tail regression . . . . .	77
4.3	PTB . . . . .	79
4.4	Sensitivity analysis . . . . .	82
5.1	Loss surface . . . . .	88
5.2	Leap . . . . .	91
5.3	Omniglot main . . . . .	98
5.4	Atari main . . . . .	100
5.5	Sensitivity analysis . . . . .	106
5.6	Ablation . . . . .	108
5.7	Atari full mean results . . . . .	113
5.8	Atari full results . . . . .	115
6.1	WarpGrad . . . . .	119
6.2	Gradient-based meta-learning . . . . .	121
6.3	Gradient warping . . . . .	122
6.4	WarpGrad main results . . . . .	131
6.5	Continual learning experiment . . . . .	132
6.6	WarpGrad architectures . . . . .	135
6.7	WarpGrad algorithms . . . . .	136
6.8	WarpGrad geometries . . . . .	141
6.9	Omniglot results . . . . .	153
6.10	Omniglot ablation study . . . . .	154
6.11	TieredImageNet results . . . . .	154
6.12	Maze navigation results . . . . .	155
6.13	Continual learning results . . . . .	155
6.14	Continual learning detailed results . . . . .	156
7.1	Deep Sea benchmark . . . . .	170
7.2	Deep Sea results . . . . .	170

7.3	Atari results with distributed training . . . . .	172
7.4	Binary Tree MDP . . . . .	188
7.5	Sensitivity analysis on Binary Tree MDP . . . . .	188
7.6	Bsuite full results . . . . .	189
7.7	Bsuite per-task results . . . . .	194
7.8	Atari ablation I . . . . .	197
7.9	Atari ablation II . . . . .	198
7.10	Atari ablation III . . . . .	198
7.11	Atari full results I . . . . .	199
7.12	Atari full results II . . . . .	200
7.13	Atari full results III . . . . .	200
7.14	Atari full results IV . . . . .	201

## List of Tables

4.1	MNIST . . . . .	78
4.2	PTB . . . . .	80
4.3	WT2 . . . . .	81
4.4	Ablation . . . . .	81
5.1	Multi-CV main . . . . .	99
5.2	Omniglot test error . . . . .	109
5.3	Omniglot hyper-parameters . . . . .	110
5.4	Multi-CV hyper-parameters . . . . .	110
5.5	Multi-CV full results . . . . .	112
5.6	Atari environments . . . . .	114
6.1	WarpGrad experimental results . . . . .	130
6.2	Omniglot results . . . . .	142
6.3	Omniglot ablation study: architecture . . . . .	143
6.4	Omniglot ablation study: random initialisation . . . . .	145
7.1	Atari benchmark . . . . .	174
7.2	Hyper-parameters for Bsuite . . . . .	190
7.3	Hyper-parameter grid searches for Bsuite . . . . .	191
7.4	R2D2 hyper-parameters. . . . .	195
7.5	Atari pre-processing hyperparameters. . . . .	196

# List of Algorithms

5.1	Leap . . . . .	93
6.1	WarpGrad: online meta-training . . . . .	126
6.2	WarpGrad: offline meta-training . . . . .	126
6.1	Online meta-training . . . . .	136
6.3	Continual meta-training . . . . .	136
6.2	Offline meta-training . . . . .	136
7.1	Bootstrapped DQN with TDU . . . . .	169
7.2	Bootstrapped TD-loss with TDU. . . . .	169
7.3	Pseudo-code for generic TDU loss . . . . .	177
7.4	Pseudo-code for $Q$ -learning TDU loss . . . . .	177
7.5	JAX implementation of TDU agent under Bootstrapped DQN . .	178

# List of Theorems & Lemmas

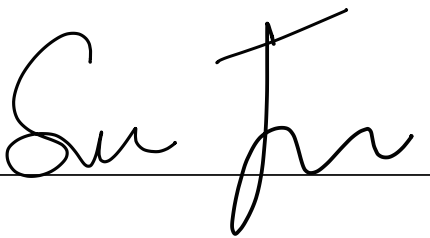
2.1	Theorem (Bellman Optimality) . . . . .	43
4.1	Lemma (IO-adaptation) . . . . .	72
5.1	Theorem (Pull-forward) . . . . .	94
7.1	Lemma (Bellman uncertainty bias) . . . . .	163
7.1	Theorem (Function approximation bias) . . . . .	164
7.2	Theorem (Temporal difference uncertainty estimation) . . . . .	165
7.2	Lemma (Bellman uncertainty bias under diagonal prior) . . . . .	180
7.3	Lemma (Bellman uncertainty bias under factorised prior) . . . . .	182
7.4	Lemma (Bias of mean TD error) . . . . .	183
7.5	Lemma (Bias of TD error variance) . . . . .	184

# Declaration of Authorship

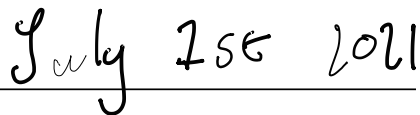
I, Sebastian Flennerhag, declares that this thesis titled, ‘Towards Scalable Meta-Learning’, and the work presented herein are my own. I confirm that:

- i. This work was done wholly while in candidature for a research degree at this University.
- ii. No part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution.
- iii. Where I have consulted the published work of others, this is always clearly attributed.
- iv. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- v. I have acknowledged all main sources of help.
- vi. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: \_\_\_\_\_



Date: \_\_\_\_\_



# Copyright

- i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420>), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.library.manchester.ac.uk/about/regulations/>) and in The University’s policy on Presentation of Theses

*“How small a thought it takes to fill a whole life.”*

— Ludwig Wittgenstein

# Abstract

Artificial intelligence questions our understanding of intelligent behaviour and our own intelligence. Recent advances rely on machine learning, where intelligence arises through statistical learning. Often, machine learning assumes agents approach tasks with no prior knowledge. This stands in stark contrast to how humans approach new problems and is unlikely to yield human-level intelligence that can learn to solve new, unseen problems as they arise. To this end, we need a learning paradigm at a higher level of abstraction.

One alternative is agents that learn *to learn*. Within this framework, agents learn not to solve a given set of tasks, but *how to* solve them. Such agents can generalise prior experiences into abstract concepts for learning and problem solving. Within the context of neural networks, current methods are limited in their ability to generalise and scale in terms of task variation and complexity.

This thesis makes four contributions that tackle these challenges. A novel method is proposed that learns to dynamically adapt an agent’s parameters to increase its expressive capacity and ability to generalise. Further, a framework is proposed that grounds meta-learning in differential geometry by learning shortest solution paths (geodesics) across tasks.

Building on these insights, a novel method for meta-learning is proposed that is simple, scalable, and effective. It is the first gradient-based meta-learner that can be directly applied to any form of learning—including supervised, unsupervised, reinforcement, continual, and online learning—opening up for meta-learning at the scale and at the level of complexity required for sophisticated artificial intelligence.

Finally, while the above methods rely on a predefined task distribution, an artificial intelligence should be able to define its own tasks as needed. To this end, a novel system for exploration in reinforcement learning is proposed that creates intrinsic tasks. These drive an agent to explore experiences where it has high uncertainty and evolves continuously as the agent learns about its world.



# Acknowledgements

I'm deeply grateful to my supervisor Professor Mark Elliot for the opportunity to pursue a PhD and the support Mark has given me to explore freely. I'm also grateful for the support I have received from my co-supervisors Professors John Keane and Hujun Yin, whose valuable insights have improved this thesis many times over.

A distinctive feature of my PhD have been the internships and exchanges I experienced. I'm particularly indebted to Andreas Damianou, whose intellectual curiosity stimulated my own and catapulted my research to much greater heights than what I would otherwise have reached. I'd also like to thank Neil Lawrence and Pablo Moreno, as well as my fellow interns at Amazon's Machine Learning Lab, for making my time there both intellectually stimulating and really fun.

My time at the Alan Turing Institute proved that a PhD journey can be rather action-packed. The amount of diversity and energy at ATI helped push my PhD forward at a critical time; thank you all for making it so easy to go to work.

I've had the great fortune to spend time at DeepMind, whose stimulating research environment has opened up a whole new world for me. Thank you Andrei, Jane, Razvan, and Raia for bringing me along! I continue to be amazed by the creativity and sheer brilliance of my colleagues, as well as the humility, kindness, and openness that permeates the culture. I could not have hoped for a better environment to pursue my research in.

Last but not least, this thesis would not have seen the light of day without the unyielding support of my friends and family. You give me the strength to pursue my passion and fill my days with laughter. Thank you for putting up with my antics, such as this one.

# Abbreviations

AI	Artificial Intelligence
CV	Computer Vision
ERM	Empirical Risk Minimization
BDQN	Bootstrapped Deep $Q$ -Network
DDQN	Double Deep $Q$ -Network
DNN	Deep Neural Network
DQN	Deep $Q$ -Network
FF(L)	Feed-Forward Layer
LSTM	Long Short-Term Memory (model)
MAP	Maximum a Posteriori
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron (Feed-Forward Layer)
NLP	Natural Language Processing
RL	Reinforcement Learning
RHN	Recurrent Highway Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
TD	Temporal Difference

# Notation

## Variables

$x$	variable (scalar or vector)
$x^{(i)}$	$i$ th sample of $x$ from an empirical distribution
$x_i, x_j$	$i$ th or $j$ th element of an ordered set or vector $x$
$x_k, x_t$	$k$ th or $t$ th iteration on a variable $x$
$X$	real-valued matrix
$\mathcal{X}$	set, vector space, or Riemannian manifold
$\theta, \phi, \xi$	model parameters
$\alpha, \beta, \lambda, \eta$	scalar hyper-parameters
$\gamma$	point or curve on a Riemannian manifold

## Operators

$G(x; \phi)$	smoothly varying matrix $G$ as a function of $x$ and $\phi$
$\langle \cdot, \cdot \rangle$ or $x^T v$	Euclidean inner product, $\langle x, v \rangle := \sum_{i=1}^n x_i v_i$
$\  \cdot \ _2$ or $\  \cdot \ $	Euclidean norm, $\ x\ _2 := \sqrt{\langle x, x \rangle}$
$\langle \cdot, \cdot \rangle_G$	inner product under metric tensor $G$ , $\langle x, v \rangle_G := x^T G v$
$\  \cdot \ _G$	norm under metric tensor $G$ , $\ x\ _G := \sqrt{\langle x, x \rangle_G}$
$[ \cdot ; \cdot ]$	concatenation, $[x; v] := (x_1, \dots, x_n, v_1, \dots, v_m)$
$\odot$	Hadamard product, $x \odot u := (x_1 u_1, \dots, x_n u_n)$
$\sigma, \varphi$	element-wise operator, $\sigma(x) := (\sigma(x_1), \dots, \sigma(x_n))$
$\omega, \Omega$	differentiable mapping, $U = \omega(X)$
$f(\cdot; \theta), f_\theta$	a model $f$ parameterised by $\theta$
$F, f^{(i)}$	a Neural Network with layers $f^{(i)}$ , $F := f^{(L)} \circ \dots \circ f^{(1)}$
$\ell$	scalar-valued objective function, e.g. $\ell(f_\theta(x), y) \in \mathbb{R}$
$\nabla, D$	gradient and Jacobian operator (Frechet derivative)
$\nabla_x g(x, u)$	gradient of a function $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ w.r.t. $x$
$\nabla_x^2 g(x, u)$	Hessian of a function $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ w.r.t. $x$
$D_x g(x, u)$	Jacobian of a function $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^k$ w.r.t. $x$
$\text{sg}[\cdot]$	stop-gradient operator defined by $\nabla \text{sg}[g(x)] := 0$

## Statistics

$p$	probability mass or density function
$p(x)$	probability mass or density of a random variable $x$
$p(\cdot; \theta), p_\theta$	probability mass or density function parameterised by $\theta$
$\mathbb{E}_x[\cdot], \mathbb{E}_p[\cdot]$	expectation operator over random variable $x$ or distribution $p$
$\mathbb{E}_{x \sim p(x)}[\cdot]$	expectation over $x$ given probability mass or density $p$
$\mathcal{D}$	dataset of observations, e.g. $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$
$\mathcal{L}, \mathcal{J}$	stochastic objective over $\ell$ for some given data distribution
$\tau$	task index; tasks define distinct optimization objectives

## RL

$s, a, r$	state, action, reward; observed quantity or random variable
$\gamma$	discount factor (not to be confused with geometry notation)
$\pi$	policy mapping state $s$ into distribution over actions, $a \sim \pi(s)$
$\pi(\cdot; \theta), \pi_\theta$	policy parameterised by $\theta$
$V$	Value function, $V(s) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^t r_t \mid s_1 = s]$
$Q$	Action-Value function, $Q(s, a) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^t r_t \mid s_1 = s, a_1 = a]$
$Q(\cdot; \theta), Q_\theta$	function approximation parameterised by $\theta$ (similarly for $V$ )

A mathematical statement should be a clear expression, the notation the minimum required to express the idea. A general philosophy of this thesis is to simplify notation whenever possible.  $f$  denotes a function or model when the distinction to  $F$  is irrelevant, while  $x, u, v$ , and  $z$  denotes arbitrary variables. For random variables, notation is overloaded so that  $x$  denotes both the random variable and a random sample.  $p(x)$  denotes both the distribution and the probability of observing the value  $x$ . In RL,  $s, a, r$  denotes both observations and random variables. Whenever possible, time subscripts are omitted.

In general, when context permits, indices are dropped. For instance, write gradient descent as  $\theta := \theta - \alpha \nabla \mathcal{L}(\theta)$ . Stochastic objectives  $\mathcal{L}$  and  $\mathcal{J}$  subsume data distributions and are written as functions of parameters only; a supervised objective may be defined by  $\mathcal{L}(\theta) := \mathbb{E}_{(x,y) \sim p(x,y)}[\ell(f(x; \theta), y)]$  for some  $p(x, y)$ ,  $f_\theta$ , and  $\ell$ . Thus note that in practice,  $\mathcal{L}(\theta)$  and  $\nabla \mathcal{L}(\theta)$  are estimated.

Occasionally, indexing will be important. The notation  $x_k$  is reserved to denote iterations within optimisation algorithms, such as in gradient descent, while  $x_t$  denote iterations within a data sequence.  $x_i$  or  $x_j$  denotes the  $i$ th or  $j$ th element of a vector  $x$  while  $x^{(i)}$  denotes the  $i$ th observation of  $x$ . For functions,  $f^{(i)}$  denotes the  $i$ th layer in a composition  $F := f^{(L)} \circ \dots \circ f^{(1)}$ . If  $F$  is parameterised by  $\theta$ , denote by  $\theta^{(i)}$  the parameters of  $f^{(i)}$ .

# Publications

The following list of publications comprise the majority of original research presented in this thesis. Contributions of co-authors are detailed below.

1. **Flennerhag, Sebastian**, Yin, Hujun, Keane, John, and Elliot, Mark. Breaking the Activation Function Bottleneck Through Adaptive Parameterization. In *Advances in Neural Information Processing Systems*, 2018.
2. **Flennerhag, Sebastian**, Moreno, Pablo G., Lawrence, Neil D., and Damianou, Andreas. Transferring Knowledge across Learning Processes. In *International Conference on Learning Representations*, 2019.
3. **Flennerhag, Sebastian**, Rusu, Andrei A., Pascanu, Razvan, Visin, Francesco, Yin, Hujun, and Hadsell, Raia. Meta-learning with Warped Gradient Descent. In *International Conference on Learning Representations*, 2020.
4. **Flennerhag, Sebastian**, Wang, Jane, Visin, Francesco, Galashov, Alexandre, Sprechmann, Pablo, Heess, Nicolas, Borsa, Diana, Baretto, André, and Razvan Pascanu. Temporal Difference Uncertainties as a Signal for Exploration. *Under Review*, 2020.

The majority of ideas, text, figures, and experiments originated from the first author, as well as all mathematical results, in all cases. Andreas Damianou had an important supervisory role on the paper *Transferring Knowledge across Learning Processes*. Credit is due to Andrei Rusu for implementing and running the mini-Imagenet experiment and the continual-learning experiment for the paper *Meta-learning with Warped Gradient Descent*. Credit is due to Pablo Sprechmann for implementing and running the Atari experiments for the paper *Temporal Difference Uncertainties as a Signal for Exploration*. All co-authors provided comments and editorial assistance on drafts of each publication.

## 1

Introduction

---

Amid all the hyperbole and sensationalism, the field of artificial intelligence is gradually revolutionising our society. In the past few years, artificial intelligence has reached a level of sophistication that enables us to train *AIs*<sup>1</sup> that outrival grand-masters in challenging games such as Chess, Go, Shogi, Dota 2, and StarCraft II (Silver et al., 2018; Berner et al., 2019; Vinyals et al., 2019). These breakthroughs speak to the inherent potential of modern approaches to artificial intelligence. For instance, in the span of a few days, AlphaZero (Silver et al., 2018) learns to play Go to a level of sophistication that humans have been unable to reach during the game’s 4000 year long history (Britannica, 2021), producing novel, hitherto unfathomable moves by humans observers.

The environments in which these AIs live may seem quite simple compared to the full complexity of the world that we as humans inhabit. However, the problem of *learning*—without detailed domain expertise—how to behave optimally in these environments is profoundly complex. In the board game Go, for instance, there are  $10^{170}$  possible board configurations (Allis, 1994)—more than the number of atoms in the universe. A computer that exhaustively searches for the best next move by planning 8 moves ahead must consider 512 quintillion ( $5.12 \cdot 10^{20}$ ) possible board configurations to make a move, yet a single move can exert influence for hundreds of moves (Allis, 1994). StarCraft II exhibits even greater complexity, with an even greater set of possible states along with multi-modal data streams that the agent must reconcile (Vinyals et al., 2019).

To overcome these challenges, modern artificial intelligence relies on machine learning to extract useful internal representations that allow agents to generalise without exhaustively searching through every possibility, much like humans do (Johnson-Laird, 1980). That machine learning allows AIs to play sophisticated games beyond human capabilities—which they learn primarily

---

<sup>1</sup>Throughout this thesis, the acronym “AI” is reserved to denote a specific instance of a machine learning system.

by playing *against themselves*—hints at the tantalising possibility of deploying autonomous systems that learn from a never-ending stream of experiences as they interact with the world.

Beyond such successes in pure game settings, machine learning has also demonstrated its potential in a host of real-world commercial settings. It is largely responsible for the recent excitement around self-driving cars, which is already offered as a service by reputable car makers (Badue et al., 2019). It has replaced meticulously curated hand-written rules previously used to power Google Translate; instead, a neural network powers the translation service (Wu et al., 2016a), which is used more than 100 billion times per day (Turovsky, 2016). Likewise, Amazon’s popular home-assistant Alexa is powered by a neural network (Kim et al., 2018b) capable of processing voice commands to more than 40 000 distinct third-party applications (Kim, 2018).

Recent breakthroughs in natural language processing have produced machine learning systems that are capable of producing substantial bodies of text that humans cannot distinguish from human-generated text (Radford et al., 2019). Moreover, these systems can learn rich linguistic representations that enable them to perform a wide variety of linguistic tasks, including translation, question-answering, and numerical computations such as arithmetic (Brown et al., 2020). Similarly, in computer vision, machine learning systems are now capable of generating images that are indistinguishable from real photographs (Brock et al., 2018a). These systems are so powerful that they can generate believable footage of real people in fabricated scenarios, which has sparked a heated debate about privacy, integrity, and even national security as artificial intelligence makes its way into everyday life (Chesney & Citron, 2019).

Yet for all its impressive achievements, human-level artificial intelligence is still in its infancy. These—and arguably every other machine learning system to date—represent so-called “narrow” artificial intelligence, in that they are highly capable at one task, but typically unable to adapt to a wider set of tasks that require the AI to adapt in some non-trivial way (Hernández-Orallo, 2017). Part of the inability of current systems to generalise can be explained by the tendency of current systems—most notably those involving neural networks—to overfit to training data (Szegedy et al., 2014; Zhang et al., 2017; Novak et al., 2018). Yet the problem runs deeper, most modern machine learning algorithms train models to solve a specific, narrowly defined task. These AIs are inherently incapable of exhibiting the type of adaptation that we associate with human-like intelligence (Chollet, 2019).

The dominant learning paradigm in machine learning relies on optimising a model to make predictions about a specific task. With sufficient data, we would expect a powerful model to eventually learn to perform that task. However, it does not follow that this model, once trained, will do well on a *different* yet related task. In fact, most machine learning models fail catastrophically even on slight modifications of the task they were trained on (for a contemporary overview, see [Sinz et al., 2019](#)). Yet many agree that such adaptability is a necessary (if not sufficient) condition for intelligence (e.g. [Legg & Hutter, 2007](#); [Thórisson & Helgasson, 2012](#); [Lake et al., 2017](#); [Wang, 2019](#); [Chollet, 2019](#)). To imbue a machine with a notion of intelligence, we would expect it to demonstrate an ability to *learn* new concepts and skills as required when facing new challenging problems.

While the exact definition of artificial intelligence is a long-standing debate, it is safe to say that contemporary machine learning falls short of these requirements. This thesis takes the view that the ability of an AI to learn new tasks in a cumulative manner, building on its previous skills, throughout its life will be central to any AI that exhibits human-level intelligence. This view stands in contrast to the dominant *tabula rasa* paradigm in machine learning, where an AI is trained on each task independently from a blank slate.

## 1.1 Limitations of Tabula Rasa

This thesis focuses on inductive biases that define the agent’s *process* of learning, in particular those that define the AI’s learning rule itself. That we tend to equate contemporary machine learning models that solve a single task with artificial intelligence dates back to classical works in artificial intelligence, which largely conceived of AI as a static assembly of logical constructs from which intelligence would emerge ([Chollet, 2019](#)). [Minsky \(1985\)](#), for instance, defined artificial intelligence as “the ability to solve complex problems.” This definition lends itself to an interpretation of AI as a machine that can “solve a task” ([Hernández-Orallo, 2017](#)). In fact, the narrow view of an AI as a logical construct that can solve complex tasks was so dominant in the past that *learning* was often not even mentioned in textbooks ([Chollet, 2019](#)). In a recent survey of its history, [Hernández-Orallo \(2017\)](#) concluded that “the field of artificial intelligence has been very successful in developing artificial systems that perform these tasks without featuring intelligence.”

Such a view of artificial intelligence is not only narrow, it is also static: it essentially equates artificial intelligence with computation ([Wang, 2019](#)). Mod-



ern machine learning puts considerably more emphasis on learning, but has largely maintained the narrow notion of artificial intelligence (Chollet, 2019). An alternative view, originated by Turing (Turing, 1950), maintains that it is the ability to *learn* to solve unseen tasks that we should strive for in artificial intelligence. In a recent effort to provide a working definition of artificial intelligence, Wang (2019) proposed the following definition of intelligence (the artificial part being uncontroversial):

“intelligence is the capacity of an information-processing system to adapt to its environment while operating with insufficient knowledge and resources.”

Here, intelligence is the capacity *to adapt* within a set of constraints. Notwithstanding certain issues with this definition (Monett et al., 2020), this notion of artificial intelligence is embodied in pioneering works that proposed artificial learning systems that could learn *to learn* (Hinton & Plaut, 1987; Schmidhuber, 1987; Bengio et al., 1991). These early works demonstrated that it is not only possible but also favourable to design algorithms that internalise their own learning rule. Since then, through a growing body of work it is becoming increasingly clear that the paradigm of training a model on a narrowly defined task is unlikely to yield an artificial intelligence except in the narrow sense.

While some argue that defining large enough a task and large enough a neural network could produce a general AI (as opposed to a narrow AI; Brown et al., 2020), this approach is unlikely to yield intelligence as defined above since the system is constrained to a single set of parameters that must contain all knowledge the AI will ever need. This seems very unlikely given that neural networks are brittle (Szegedy et al., 2014), generalise poorly (Zhang et al., 2017; Novak et al., 2018), even to moderately new tasks (Sinz et al., 2019), and learning such networks require a large amount of data (in terms of bits) (Brown et al., 2020). These limitations cast doubt on the feasibility of the tabula rasa approach to artificial intelligence. For instance, Chollet (2019) questions the feasibility of constructing a single training set and then training a model that, once trained on it, can exhibit human-level artificial intelligence without further need for learning.

We can contrast the tabula rasa approach with findings from developmental psychology. Spelke & Kinzler (2007) summarise the literature as refuting both extremes: the human brain is neither a collection of static rules nor a blank slate, but a combination of innate priors *through which* we can learn skills needed throughout our lives, from motor control (Braun et al., 2009) to abstract concept acquisition (Lake et al., 2015). While it is not necessary

for an AI to mimic the human brain, a notion of artificial intelligence relates to human intelligence in the sense that we are the judges—to deem an entity intelligent we typically relate it to ourselves (Turing, 1950). Human intelligence, in turn, is perhaps the best source of inspiration available to us (Jankowski et al., 2011). The developmental view of human intelligence certainly suggests that adaptability is central to intelligence. From this perspective, intelligence refers not to the current capabilities of a system, but its ability to acquire new capabilities as necessitated by its environment and goals (Thórisson & Helgasson, 2012; Wang, 2019). As Chollet (2019, p. 40) observes, “intelligence is, in a way, a conversion rate between information about part of the situation space, and the ability to perform well over a maximal area of future situation space, which will invoke novelty and uncertainty.” The key words here are *novelty* and *uncertainty*; an AI cannot be certain about what knowledge it might need in the future, nor is it guaranteed to have enough experience at any point in time to obtain that knowledge, hence it must always have the capability to learn. In stark contrast to Minsky’s view of an AI as something that solves a complex task, here, an AI is a cumulative, never-ending learning process (Thórisson et al., 2019; Wang, 2019).

Perhaps somewhat surprisingly, we can adopt this perspective within machine learning without much change. Early works that took this perspective formalised the notion of *learning to learn* as an embedded mechanism within the model (Schmidhuber, 1987; Hinton & Plaut, 1987). In this way, the model internalises its own learning rule, but the system itself is trained within the standard machine learning framework. Other approaches use optimisation as an inductive bias and meta-learn certain aspects of the update rule (Bengio et al., 1991). These ideas have since been expanded in various ways under the umbrella of *meta-learning*, often characterised as a bi-level optimisation problem, where the higher level of abstraction *meta-learns* how to learn across a distribution of tasks, while the lower level is the standard machine learning problem of learning to solve a task (Lake et al., 2011; Vinyals et al., 2016; Santoro et al., 2016).

Meta-learning formalises the notion of learning at two levels. Meta-knowledge encodes general knowledge about a class of problems as well as the process of learning such tasks. It refers to the process of accumulating such knowledge so that when the AI is faced with a new (but related) task, it can learn to perform this task rapidly. Hence, in this way, we can understand meta-learning as the problem of learning useful inductive biases from data. These biases are then brought to bear when the AI learns a new task. While the tools of meta-learning largely remain the same as in traditional machine learning, the

perspective and the philosophy shifts from the problem of learning a given task, to the problem of learning to learn over many tasks (Thrun & Pratt, 1998). It therefore has the potential to bring us one step closer to a more general form of artificial intelligence.

Recent works have demonstrated the potential of meta-learning to further advance the state-of-the-art in machine learning; successful demonstrations include few-shot image recognition (Vinyals et al., 2016), continual learning (Javed & White, 2019; Flennerhag et al., 2020a), adversarial robustness (Yin et al., 2018), unsupervised learning (Metz et al., 2019), exploration in reinforcement learning (Alet et al., 2020), and even the possibility to learn the concept of neural networks as well as gradient descent itself purely from data (Real et al., 2020). With that said, contemporary meta-learning faces a set of constraints that must be overcome in order to unlock its potential for artificial intelligence.

## 1.2 Limitations of Contemporary Meta-Learning

Contemporary meta-learning covers a wide range of research topics as different communities have cultivated a notion of meta-learning in their respective contexts and thus assigned slightly different meanings to the term. In its broadest sense, the term is taken to mean any form of meta-knowledge that is brought to bear on a given problem. Meta-learning has been used in the context of transfer learning (Thrun & Pratt, 1998), multi-task learning (Vilalta & Drissi, 2002), and autoML (Yao et al., 2018a; Vanschoren, 2018). In this thesis, meta-learning is restricted to mean learning a learning rule, whereby a learning rule defines the means by which an AI adapts to its current context.

While earlier literature tended to focus on the online learning setting (Schmidhuber, 1987; Hochreiter et al., 2001), where both meta-learning and learning happen concurrently in a single stream of data, contemporary meta-learning has gravitated towards few-shot learning as its canonical problem formulation. In this setting, a “task” is defined as a pair of data sets: a small training set with at most a handful of samples (for instance, few-shot classification often uses 1 or 5 samples per task-specific class) and a test set (Lake et al., 2011; Vinyals et al., 2016). Given a handful of examples of a task, the goal of the learner is to quickly adapt so that it can perform on further data from the new task.

In contrast to a tabula rasa approach, few-shot meta-learning meta-learns an inductive bias for the learner from related tasks (Lake et al., 2015) so that the learner can achieve this form of adaptation on new tasks that has not yet seen. Meta-learning in this context refers to the problem of learning this

inductive bias over a distribution of related tasks (Vinyals et al., 2016). Because most contemporary works in meta-learning, whether implicitly or explicitly, assume a few-shot problem setting, recent works share certain properties that limit their general applicability. First, they tend to scale poorly with the training set size; second, they require expressive models to generalise, rendering them computationally expensive to meta-learn; third, they assume tasks are exogenously given, which introduces a challenging engineering problem.

### 1.2.1 Scalability

A pervasive limitation of contemporary meta-learning is its inability to scale meta-learning systems beyond the few-shot learning paradigm. Because most algorithms are developed for few-shot learning, they accept a high computational complexity in terms of learning, assuming that the learner will only engage in a relatively short burst of learning. To see this more concretely, consider two common approaches to few-shot learning.

One approach treats the meta-learner as a complete black box (Schmidhuber, 1987; Li & Malik, 2016; Ravi & Larochelle, 2017) and is motivated by choosing a Turing complete model as the meta-learner (Schmidhuber, 1987; Hochreiter et al., 2001) so that it can represent any learning rule, at least in theory. Recurrent Neural Networks (RNNs), which are known to be Turing complete (Siegelmann & Sontag, 1995), are often the architecture of choice. In a black-box meta-learner, the update rule is represented by the forward computation of the model, while its output represents the update itself.

For RNN-based meta-learners, this immediately implies quadratic complexity in the number of parameters of the task learner, since the RNN is composed of a set of matrix operations. Further, in the case of RNNs, the cost of training the meta-learner also increases in the number of training steps and can cause various forms of instability issues (Pascanu et al., 2013; Balduzzi & Ghifary, 2016; Miller & Hardt, 2019). An alternative is to rely on convolutional neural networks (Mishra et al., 2018), but these come with their own limitations.

A general challenge for black-box approaches is that the learning rule is defined by a meta-learned model. Hence they rely on the generalisation properties of the model class. Their complexity fundamentally limits black-box approaches to relatively small meta-learners and in practice, they tend to generalise worse to new tasks than meta-learners that are built on top of stronger inductive biases, such as gradient descent (Finn & Levine, 2018).

As an alternative to black-box meta-learning, [Finn et al. \(2017\)](#) proposed Model-Agnostic Meta-Learning (MAML), an algorithm that uses gradient descent as the learning rule and meta-learns the initialisation of the gradient descent process. This initialisation is shared across tasks and meta-learned to maximise the expected final performance across a given task distribution (defined in [Section 3.5](#); see also [Vinyals et al. \(2016\)](#); [Zamir et al. \(2018\)](#); [Triantafillou et al. \(2020\)](#); [Yu et al. \(2020\)](#); [Hospedales et al. \(2020\)](#)). MAML relies on the notion that there exists a good initialisation of gradient descent for related tasks, such that a few gradient steps are sufficient to find optimal task performance.

MAML-based algorithms have been very popular within few-shot learning due to their simplicity and performance ([Yao et al., 2018a](#)). Yet, in recent years, few-shot learning domains designed specifically for domain generalisation, MAML-based approaches have shown themselves to be less efficient than simpler but more scalable alternatives ([Tian et al., 2020](#); [Doersch et al., 2020](#)).

A limitation of MAML-based approaches is that they backpropagate from the final performance on a task to the learner’s initial parameters. As this requires computing Hessians at each parameter point on the trajectory, MAML-based approaches are subject to quadratic complexity both in the number of parameters in the model and in the number of training steps taken on a task. As such, it is rarely applicable to problems where task adaptation requires less than a handful of adaptation steps ([Flennerhag et al., 2019](#)).

This thesis presents a set of gradient-based meta-learning algorithms that do not require this form of backpropagation. This allows them to scale beyond few shot learning and open up to new learning paradigms, such as continual meta-learning ([Javed & White, 2019](#); [Flennerhag et al., 2020a](#)).

The first contribution to this line of work tackles scalability in gradient-based meta-learning of an initialisation. To avoid backpropagating through the final performance, as in MAML, [Flennerhag et al. \(2019\)](#) exploit the Riemannian geometry underlying learning. In particular, this work leverages the fact that rapid adaptation implies fast convergence in parameter space. Hence, rather than looking at final performance, the proposed method meta-learns an initialisation such that adaptation converges as rapidly as possible.

The meta-objective this work introduces can be solved iteratively using only statistics encountered during task adaptation and thus avoids backpropagating through the adaptation process. Meta-updates can therefore be computed almost free of charge. This is the first meta-learner that enjoys both such scalability and guarantees of convergence.

The second contribution to scalable gradient-based meta-learning introduces a framework for meta-learning the update rule, as opposed to the initialisation. This work is motivated by a key limitation of meta-learning the initialisation of a learner: it is a relatively passive form of knowledge transfer that interacts with the task learner only at initialisation. For the meta-learner to have a stronger influence over learning, it needs to interact with the learner at each learning step, as black-box meta-learners do.

In the context of gradient-based meta-learning, a natural way for the meta-learner to interact with the gradient update is to project the gradient using a meta-learned projection operator (in the context of MAML; [Lee & Choi, 2018](#); [Park & Oliva, 2019](#)). For positive-definite preconditioners, gradient preconditioning retains the inductive bias of gradient descent (most notably, guarantees of convergence) while it allows us to re-formulate gradient-based meta-learning in terms of an operator that projects the gradient at every step of learning.

[Flennerhag et al. \(2020a\)](#) show that meta-learning this projection is equivalent to meta-learning a geometry, which is agnostic to the trajectory used to generate data. Thus, meta-learning a gradient-based update rule can be achieved without backpropagation through the adaptation process while avoiding dependence on the initialisation. Leveraging these properties, the proposed meta-learner is guaranteed to converge, scales beyond few-shot learning, and is readily applicable to any form of learning, including supervised, unsupervised, reinforcement, and continual learning.

### 1.2.2 Generalisation

Meta-learning faces two generalisation challenges. The first is the more typical generalisation within a task; to generalise from the training data to new unseen samples from the same task data distribution. The second form is specific to meta-learning and requires the meta-learner to generalise across tasks, in the sense that meta-knowledge learned on a set of training tasks should generalise to learning new tasks from the same task distribution. These are both open research questions in the literature that in part rely on the question of what representations of learners and meta-learner tend to generalise better than others ([Hospedales et al., 2020](#)).

The architectural design matters greatly since this fundamentally determines how task-specific and task agnostic knowledge interact. It also determines an agent’s ability to adapt. More generally, understanding properties of neural networks that cause or prevent generalisation is a very active area of research

(e.g. Canziani et al., 2016; Neyshabur et al., 2017; Zhang et al., 2017; Novak et al., 2018; Achille & Soatto, 2018; Liang et al., 2019). In terms of meta-learning specifically, it has been observed that the meta-learners benefit from greater expressive capacity in the learner (Rusu et al., 2019). A likely reason for this is that it allows the meta-learner to efficiently encode meta-knowledge in the learner’s feature representation (Tseng et al., 2019; Raghu et al., 2020).

Large models for meta-learning are problematic because of the computational cost they are associated with, especially as meta-learning involves training the learner on several tasks. It is therefore pertinent to discover architectures that enable efficient meta-learning. This thesis contributes towards this strand of the literature by investigating the means by which standard neural network layers can be made more expressive for a given parameter count (in the context of meta-learning, this allows a learner to express greater adaptive capacity and thus improve its generalisation across a task distribution, as demonstrated in Flennerhag et al. (2020a)).

In particular, Flennerhag et al. (2018) leverage that neural networks (typically) use fixed activation functions that are linear over large parts of their domain. This limits the degree of non-linearity that can be expressed in any one output dimension, thus throttling the network’s expressive capacity. The proposed method relies on inducing non-linearity in the linear transformation directly. This is achieved by adaptively parameterising a layer, which can be efficiently implemented as a sequence of element-wise gating mechanisms within the layer. Such adaptive parameterisation induces greater expressive capacity in a layer and can outperform carefully tuned baselines. The proposed adaptation mechanism is a prime candidate for a meta-learning mechanism and—because it only needs to output an element-wise scaling vector—it scales gracefully with the number of parameters in the model.

Previously, gradient-based meta-learning had no such option. To meta-learn an initialisation means that the number of meta-parameters is equal to the size of the model. Worse still, while the number of meta-parameters increases linearly in the number of task-adaptable parameters, computational complexity grows quadratically. This thesis provides contributions to the literature that largely remove this limitation. In terms of meta-learning an initialisation, this thesis proposes a method that has almost no computational cost for meta-updates and thus entirely skirts the issue. For meta-learning a gradient-based update rule, the proposed method de-couples meta- and task-parameters, and hence increasing the latter does not increase the former.



### 1.2.3 Exogenous Tasks

A common assumption in contemporary meta-learning is the existence of a task distribution. The role of meta-learning, put simply, is to learn inductive biases that speed up learning over this task distribution. This puts meta-learning at the mercy of the task distribution since a poorly defined task-distribution will effectively prevent the meta-learner from generalising to at least some unseen tasks. Gradient-based meta-learning is somewhat less exposed to the task distribution since its learning rule—gradient descent—is at heart a converging process (this also applies to meta-learners that are based on other forms of optimisation (e.g. [Chen et al., 2017](#))). However, it can still perform poorly if the initialisation is very far away from the global optima causing the learner to be attracted to poor local minima, or if a meta-learned gradient preconditioner projects gradients away from the true solution space. For black-box meta-learners, the situation is more dire. Since a black-box meta-learner defines the update rule, its ability to generalise relies entirely on the properties of the function class underpinning the meta-learner.

Ultimately, a key limitation of current meta-learning is the assumption of an exogenous task distribution. In general, we would expect an AI to learn continuously from its own experiences. Hence the AI is itself responsible for both how to define tasks, and how to sample them. This is still largely untouched in the literature, in part because it is not clear how to define the meta-learning problem. This problem has previously been studied in reinforcement learning, where past experience can be used to construct “tasks” ([Andrychowicz et al., 2017](#); [Sukhbaatar et al., 2018](#)). More generally, there is a long line of work that studies how to automatically generate curricula of learning problems to progressively introduce to the agent during the course of learning ([Schmidhuber, 1991](#); [Oudeyer et al., 2007](#); [Jaderberg et al., 2017c](#); [Riedmiller et al., 2018](#)).

This thesis’s final contribution builds on these works and considers systems that define their own tasks internally. [Flennerhag et al. \(2020b\)](#) consider this problem outside of the current meta-learning paradigm, taking a reinforcement learning perspective. How might an agent propose its own tasks, such that learning from such tasks helps the agent to learn for the future? In this work, this is achieved by defining an agent as composed of two policies. One policy—a form of meta-policy—is tasked with collecting data that another, a form of adaptive policy, uses to learn efficiently. This creates an automatic curriculum of tasks, as the task facing the meta-policy changes as the adaptive policy is learning from the data the meta-policy has collected previously. Prior works



tend to rely on adversarial training (Sukhbaatar et al., 2018; Florensa et al., 2018; Kachalsky et al., 2019), which yield automatic curricula of progressively harder tasks, but not necessarily tasks that are useful for *learning*. The proposed method is evaluated on a set of hard exploration problems and the results demonstrate that agents that learn from their own tasks might be a potential way forward towards large-scale meta-learning.

### 1.3 Thesis Outline

Part I provides the necessary context for original research presented in this thesis. Chapter 2 provides a review of relevant tools from machine learning. It focuses on optimisation as machine learning and provides a brief introduction to reinforcement learning. Chapter 3 serves as a review of the field of meta-learning as viewed in this thesis. The remainder of the thesis presents original research.

Part II contains original research on meta-learning. Chapter 4 presents a novel method for learning how to dynamically adapt the parameterisation of a model conditional on the input (Flennerhag et al., 2018). This can produce a significant increase in model expressiveness for a given parameter count: it achieves state-of-the-art performance using 30% fewer parameters than comparable methods. While the original work is not focused on meta-learning (in the contemporary sense of generalising across task distribution), Chapter 6 uses this method in a proper meta-learning setting, where the adaptive mechanism is meta-learned to facilitate learning when a reinforcement learning agent faces a stream of tasks.

Chapter 5 presents a novel geometric perspective on gradient-based meta-learning and derives an algorithm for meta-learning an initialisation (Flennerhag et al., 2019). The method aims to make gradient descent trajectories as short as possible in terms of the distance learning travelled on the model’s parameter manifold. This approach can scale beyond few-shot learning and achieve strong results on problems hitherto outside the scope of gradient-based meta-learners.

Given these findings, Chapter 6 pursues the geometric perspective further still. While the above method optimises for an initialisation over a distribution of geometries, this work uses Riemannian geometry to derive an algorithm for meta-learning the task geometry (Flennerhag et al., 2020a). The resulting meta-learner is both flexible and scalable. This work demonstrates that a single algorithm can learn-to-learn across several learning problems, including few-shot, supervised, and reinforcement learning. Further, it introduces a novel

form of meta-learning—meta-continual learning (also independently proposed by [Javed & White \(2019\)](#))—and demonstrates that the proposed method can meta-learn how to avoid catastrophic forgetting ([French, 1999](#)).

Finally, Part **III** turns towards the future of meta-learning. Chapter 7 considers an agent that constructs its own tasks as a means of meta-learning (in some sense of the word). This work does not rely on the typical meta-learning formalism and instead considers a pure reinforcement learning setting. It proposes an agent that is composed of two policies ([Flennerhag et al., 2020b](#)). The first policy learns to solve the task given its own experiences and experiences collected by the second system. The second policy learns to collect experience that provides as much learning signal to the first system as possible. This form of a gradual curriculum lets the agent learn efficiently in a host of challenging environments, providing a path towards meta-learning without a task distribution.

Chapter 8 concludes the thesis with a discussion of implications for future research in meta-learning and artificial intelligence and considers how the research presented herein can provide stepping stones towards systems that learn to learn continually by generalising from past experiences.

---

# I

## *Foundations*

## 2 Machines that Learn

---

This chapter formalises the notion of a machine that learns as used in this thesis. Broadly speaking, there are many ways to formalise machine learning, for instance through Bayesian inference, stochastic optimisation, or other means of statistical inference. These approaches offer different viewpoints and bring to bear different tools for learning. For a full review, the reader is referred to one of the many excellent books on the topic (e.g. [Mitchell, 1997](#); [MacKay, 2003](#); [Bishop, 2006](#)). This thesis relies on framing machine learning as an optimisation problem and the goal here is to concisely present and discuss the fundamental tools that underpin original research in later chapters.

The chapter is structured as follows. Section [2.1](#) provides a brief taxonomy of the types of machine learning problems that will feature in this thesis. Section [2.2](#) formalises machine learning as an optimisation problem and introduces the notation used throughout the thesis. In particular, Chapter [3](#) presents historical and contemporary work on meta-learning in light of this formalism. Section [2.3](#) presents the gradient descent framework, which is the main workhorse algorithm used by learners in this thesis to solve machine learning problems. Finally, Section [2.4](#) contains a brief presentation of key concepts from reinforcement learning that will feature in this thesis.

### 2.1 Types of Machine Learning

A common definition of machine learning is due to [Mitchell \(1997\)](#), whereby a machine is said to learn “from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ”. This notion embodies the idea that, because the machine will only have access to a limited amount of experience, the machine must internalise knowledge that generalises to new, unseen data. This requirement of generalisation separates machine learning from classical optimisation and statistics (for a more in-depth comparison, see [Goodfellow](#)

et al., 2016, p. 267). Depending on how these quantities are defined, we obtain distinct machine learning problems. The types of machine learning problems considered in this thesis are presented below, with the exception of reinforcement learning, which is introduced separately in Section 2.4.

**Experience** Throughout this thesis, experience is assumed to be sampled from some unknown data-generating process, defined by a probability density  $p : E \mapsto (0, 1)$  that is independent of the machine learner (henceforth learner). The learner is exposed to experience from this data-generating process in the form of a dataset  $\mathcal{D} := \{e^{(i)}\}_{i=1}^N$ , where  $e^{(i)} \sim p(e)$  is an independent random sample from  $p$ . In this thesis, the experience will take the form of a tuple  $e^{(i)} := (x^{(i)}, y^{(i)})$ , where  $x^{(i)} \in \mathbb{R}^n$  is an input variable and  $y^{(i)} \in \mathbb{R}^c$  is a target variable. Throughout this chapter, unless otherwise stated, it is assumed that samples  $e^{(i)} \sim p(e)$  are independently and identically distributed (i.i.d), meaning that they are independent draws from  $p$ .

**Task** The class of tasks that this thesis is primarily concerned with is that of predicting  $y^{(i)}$  from  $x^{(i)}$ . This class of tasks is often called *supervised learning* because the learner is being supervised by a ground truth  $y^{(i)}$  that tells the learner what it should predict for each  $x^{(i)}$ . Within this space, there are many flavours of tasks. This thesis will consider three specific tasks:

1. *Classification*: In this task,  $y$  is a categorical variable that can take on one of  $c$  values;  $y \in \{1, 2, \dots, c\}$ . When class membership is mutually exclusive—as in this thesis—classification can be formalised by introducing a ground truth map  $g : x \mapsto y$  that assigns a label  $y$  to  $x$ . The goal in classification is to learn an approximation  $f$  of  $g$  from a dataset  $\mathcal{D}$ .
2. *Regression*. Is a similar task to classification, but differs in that the target  $y \in \mathbb{R}^c$  is a continuous variable. Regression can be also be formalised by assuming a ground truth map  $g : x \mapsto y$  that the learner must learn to approximate under some appropriate metric over  $\mathbb{R}^c$ .
3. *Density estimation*. Differs slightly from the previous two tasks in that the goal of the learner is to approximate the conditional probability distribution  $p(y | x)$ . Density estimation is more general in that the goal is to model the full conditional density, or even the full joint density  $p(x, y)$ . Given an approximation to the full conditional density  $p(y | x)$ , a point prediction of  $y$  at  $x$  for regression or classification can be obtained from the learner by choosing the mean, median, or mode of the learned model.

A closely related class of tasks is unsupervised learning. In this setting, there is no target variable (i.e.  $e^{(i)} := x^{(i)}$ ) and instead the goal of the learner is to extract useful knowledge from the data. These are often harder problems because it is not always clear how to quantify useful knowledge. While many techniques for supervised learning have analogues in unsupervised learning—for instance, density estimation can be applied to estimate  $p(x)$ —other approaches are distinct from supervised learning. For a full treatment of unsupervised learning, see for instance [Ghahramani \(2003\)](#).

**Performance measure** In general, given a learned model  $f$ , a performance measure  $P$  is constructed by measuring the “closeness” between a prediction  $f(x)$  and the ground truth  $y$ . The specifics of how to measure the quality of the prediction differ based on the specific task.

1. *Classification.* The performance measure used in this thesis is the accuracy of the learned model. The accuracy measures whether the predicted class assignment  $f(x)$  matches with the true class assignment  $y$ . More nuanced measures of performance are often considered when classification is the primary object of study (e.g. [Fawcett, 2006](#)).
2. *Regression.* The performance measure is typically a distance measure  $d(f(x), y) \in [0, \infty)$  between the target  $y$  and the prediction  $f(x)$ . There are many choices for  $d$ , such as any metric function. Other divergence measures are also useful and the most common choice is the squared error,  $\|f(x) - y\|_2^2$ , between the prediction  $f(x)$  and the target  $y$ . When  $f$  is a probabilistic model, regression is closely related to density estimation ([Murphy, 2012](#), pp. 247).
3. *Density estimation.* As the goal is to approximate the density  $p(y | x)$ , performance is measured by the distributional similarity of  $p$  and  $f$ . There are numerous ways of measuring (dis)similarity, but when  $p$  is not known, the metric must rely on empirical data. A common choice is the Cross-Entropy of  $f$  relative to  $p$  (as measured by  $\mathcal{D}$ ). This measure is closely related to the (log) likelihood function (cf. [Berger & Wolpert, 1988](#)) and the Kullback-Liebler Divergence. In this thesis, these objects are mainly used for learning, in which case they reduce to the same optimisation problem (see Section 2.2).

Importantly, the data used for learning and evaluation are separated. Thus the performance measure—which is an evaluation metric—is defined over *unseen* data sampled from  $p$ , collected in a test set  $\mathcal{V} = \{e^{(i)} \mid e^{(i)} \notin \mathcal{D}\}_{i=1}^M$ .

Machine learning therefore differs from classical optimisation and statistics in two vital respects. Firstly, the learner is generally unaware of  $P$  when learning. In other words, the learner might be optimising a different quantity. In contrast, statistics and optimisation typically both learn and evaluate the model under  $P$ . Secondly,  $P$  is evaluated on data the learner has not seen. In contrast, statistics typically make use of all available data and is more concerned with analytical properties of the estimator (Bzdok et al., 2018), while classical optimisation studies methods to find the function  $f$  that maximises  $P$  over some fixed dataset (cf. Nocedal & Wright, 2006).

**Example** To illustrate a typical machine learning problem in this thesis, consider the regression example from Chapter 4. The problem is defined by an input variable  $x \in \mathbb{R}^2$  distributed according to  $\mathcal{N}((0, 0), I_2)$ , i.e. the multi-variate normal distribution. It introduces a ground truth map  $g : x \mapsto y$  given by  $g(x) = (2x_1)^2 - (3x_2)^4$ . The target variable is sampled from  $p(y \mid x) = g(x) + \epsilon$ , with white noise  $\epsilon \sim \mathcal{N}(0, 1)$ . The goal of the learner is to find a model  $f$  that minimises the squared error between observed targets  $y^{(i)}$  and predictions  $f(x^{(i)})$ , i.e. to minimise  $\frac{1}{N} \sum_{i=1}^N \|y^{(i)} - f(x^{(i)})\|_2^2$ . The learned model is evaluated under the same metric, but on newly sampled data.

This summarises the taxonomy of machine learning problems a learner will face in this thesis. Next, these are cast into a general optimisation problem to compactly express the notion of a machine that learns.

## 2.2 Machine Learning Problem Definitions

Fundamental to this thesis is the view of machine learning as the process of solving an optimisation problem. This is by no means the only way of viewing machine learning (see for instance MacKay, 2003; Bishop, 2006; Murphy, 2012), but it has proven to be a powerful framework for meta-learning over a distribution of tasks (to be introduced in Chapter 3). When casting machine learning problems into optimisation problems, learning is defined by the process of iteratively searching for a solution given observations from the data-generating process.

There are several ways of deriving an optimisation problem for a given type of machine learning problem (Section 2.1). Two common ways are the Maximum Likelihood Estimation approach and the Empirical Risk Minimization framework, presented below.

### 2.2.1 Maximum Likelihood Estimation

This approach is best understood as a means of solving a probability density estimation problem. The goal of the learner is to approximate the unknown density  $p(y | x)$  (or some other density, such as  $p(x)$  and  $p(x, y)$ ) given observed data  $\mathcal{D}$ . In particular, the learner must choose the parameters  $\theta$  for a model  $f_\theta$  from a family of parametric models  $\mathcal{F} = \{f_\theta \mid \theta \in \Theta\}$ .

Maximum Likelihood Estimation (MLE) relies on the Likelihood Principle, which states that inference should be based only on observed data—in particular, it asserts that the likelihood function contains all available information about the data (Berger & Wolpert, 1988, p. 19). The likelihood function for a model  $f_\theta$  is defined as  $L(\theta; \mathcal{D}) := f_\theta(\mathcal{D})$ , where  $f_\theta(\mathcal{D})$  denotes the probability density of  $\mathcal{D}$  under  $f_\theta$ . Assuming samples are i.i.d, it follows from standard laws of probability that  $f_\theta(\mathcal{D}) = \prod_{i=1}^N f_\theta(x^{(i)})$ . From an optimisation point of view, it is more convenient to use the log-likelihood,  $\log f_\theta(\mathcal{D}) = \sum_{i=1}^N \log f_\theta(x^{(i)})$ .

An immediate consequence of the Likelihood Principle is that a model that fits the data should have a high likelihood of having generated the data. Intuitively speaking MLE seeks the model  $f_\theta$  that is most likely to have generated the data. A learner solves this problem by maximising  $L(\theta; \mathcal{D})$ . Assuming i.i.d. data, the learner's problem is often formulated as minimizing the negative log-likelihood function. These are equivalent formulations as  $\min_x -g(x) = \max_x g(x)$  and the maximiser of a function  $g$  is also the maximiser of the function  $\log g$ . Thus, the form of maximum likelihood estimation used in this thesis is defined by the problem

$$\min_{\theta \in \Theta} \sum_{i=1}^N -\log f_\theta(x^{(i)}). \quad (2.1)$$

The maximum likelihood estimate—i.e. the solution to Eq. 2.1—need not exist nor be unique. In machine learning, these are not serious issues because Eq. 2.1 might still be useful for a learner to improve its model, even if it cannot compute an exact solution. With that said, the likelihood function and the maximum likelihood estimator are surprisingly profound statistical objects (for a summary, see Wasserman, 2013, pp. 153). Assuming there is some  $\theta^*$  such that  $f_{\theta^*} = p$ , under certain regularity conditions on the model, the solution to Eq. 2.1 can be shown to converge in probability to  $\theta^*$  as  $N \rightarrow \infty$  (Wald, 1949). It can also be shown to be asymptotically optimal, meaning that it has the smallest possible asymptotic variance among all well-behaved estimators.



MLE is also related to Bayesian inference. The likelihood function influences the posterior distribution, as  $p(\theta \mid \mathcal{D}) = f_\theta(\mathcal{D})p(\theta)/p(\mathcal{D})$ . Thus, Bayesian inference is implicitly compliant with the likelihood principle (Berger & Wolpert, 1988, p. 23). In particular, MLE is closely related to the maximum a posteriori (MAP) estimate, defined by  $\max_\theta f_\theta(\mathcal{D})p(\theta)/p(\mathcal{D})$ . For a uniform prior  $p(\theta)$ , MLE coincides with MAP estimation; otherwise, the MAP estimate converges to the maximum likelihood estimate as  $N \rightarrow \infty$  (Murphy, 2012, p. 71).

MLE is one way of casting a machine learning problem into an optimisation problem by means of the likelihood principle. Another way to cast a machine learning problem as an optimisation problem is through Empirical Risk Minimization (ERM; Vapnik, 1999). This framework does not assume that the goal is to estimate a density, and is thus more broadly applicable, but the gain generality comes at the cost of requiring the user to specify a loss function.

### 2.2.2 Empirical Risk Minimization

In ERM, a learner is faced with a generative process  $p$  from which it observes samples  $(x^{(i)}, y^{(i)}) \sim p(x, y)$ . The goal of the learner is to choose a model  $f_\theta$  from some hypothesis space  $\mathcal{F} = \{f_\theta \mid \theta \in \Theta\}$  (this presentation restricts attention to parametric hypothesis spaces, but ERM applies equally to functional hypothesis spaces) that approximates the true relationship  $x \mapsto y$ .

This framework requires a user-specified loss function  $\ell : \mathbb{R}^c \times \mathbb{R}^c \rightarrow \mathbb{R}$  that measures the fit of the model's prediction  $f_\theta(x)$  and the ground truth  $y$ . The loss function is typically required to satisfy  $\min_\theta \ell(f_\theta(x), y) \geq \ell(y, y)$  and  $\ell(f_\theta(x), y) = \ell(y, y) \implies f_\theta(x) = y$ . Given  $\ell$  and the probability space on which  $p$  is defined, the *risk* of a model  $f_\theta \in \mathcal{F}$  is defined by

$$\mathcal{R}(\theta) := \mathbb{E}_{(x,y) \sim p(x,y)} [\ell(f_\theta(x), y)]. \quad (2.2)$$

Intuitively, the risk of  $f_\theta$  measures the expected loss the model incurs when it fails to predict  $y$  given  $x$ . A model with no risk represents the true relationship  $x \mapsto y$ . To compare imperfect models,  $\ell$  must additionally be defined such that the larger the risk, the less likely the model is to be accurate. In practice, the true risk cannot be measured since the generative process  $p$  is not accessible to the learner. Consequently, the object of interest is the empirical risk, which

is defined over a dataset  $\mathcal{D}$  of samples from  $p$ :

$$\mathcal{L}(\theta; \mathcal{D}) := \sum_{i=1}^N \ell \left( f_{\theta}(x^{(i)}), y^{(i)} \right). \quad (2.3)$$

This measure is motivated by Monte-Carlo estimation: if samples in  $\mathcal{D}$  are identically and independently distributed (i.i.d.), meaning that they are independent draws from  $p$ , then  $\mathcal{L}$  is an unbiased and consistent estimator of  $\mathcal{R}$  (Vapnik, 1999). The learner's problem is therefore to find a model  $f_{\theta} \in \mathcal{F}$  with minimal empirical risk, resulting in the optimisation problem

$$\min_{\theta \in \Theta} \mathcal{L}(\theta; \mathcal{D}). \quad (2.4)$$

All machine learning problems in this thesis can be written on this form for some choice of  $\ell$ . In particular, MLE and ERM coincide if  $\ell$  is chosen to be the negative log-likelihood of  $f_{\theta}$  at  $(x^{(i)}, y^{(i)})$ .<sup>2</sup> Thus, from the point of view of this thesis, the learner is indifferent as to whether the problem is formulated from the point of view of MLE or ERM.

For classification problems,  $\ell$  is indeed typically the negative log-likelihood function. For regression problems,  $\ell$  can either be the negative log-likelihood or some geometric measure of distance between the target  $y$  and the prediction  $f_{\theta}(x)$ , generally defined by

$$\ell(f_{\theta}(x), y) = \|f_{\theta}(x) - y\|_{G(y)}. \quad (2.5)$$

Often, the metric space is Euclidean, so that  $\|\cdot\|_G = \|\cdot\|_2$ . To ease optimisation, this can be simplified to the so-called *mean-squared error* objective,  $\|\cdot\|_2^2$ , in which case Eq. 2.4 is written

$$\min_{\theta \in \Theta} \frac{1}{2} \sum_{i=1}^N \left( f_{\theta}(x^{(i)}) - y^{(i)} \right)^2. \quad (2.6)$$

While derived from a metric point of view, one can show that the minimiser

---

<sup>2</sup>Similarly, in the machine learning problems defined in this thesis, i.e. under a target function  $g : x \mapsto y$ , direct algebraic manipulation shows that minimizing the negative log-likelihood is equivalent to minimizing the Kullback-Leibler Divergence between the model  $f_{\theta}$  and  $g$  (Wasserman, 2013, p. 154). Further, both of these objectives are equivalent to minimizing the Cross-Entropy between  $f_{\theta}$  and  $g$ .

of the mean-squared error in Eq. 2.6 is the MLE estimate under a Gaussian model  $f_\theta$  (Murphy, 2012, pp. 200). Other metrics  $\|f_\theta(x) - y\|_{G(y)}$  can also be shown to have sound statistical underpinnings (Hastie et al., 2009, pp. 43).

Going forward, the general definition of a machine learning problem will be taken to be Eq. 2.4, with the understanding that the problem formulation relies on sound statistical foundations for appropriate choices of  $\ell$ . Note that—in contrast to statistics and optimisation—the loss function  $\ell$  need not coincide with the performance measure  $P$  (Section 2.1), as previously discussed.

## 2.3 Learning Through Stochastic Gradient Descent

Both MLE and ERM provide principled frameworks for learning parameters  $\theta$  of a model  $f_\theta$  given data  $\mathcal{D}$ . For the optimisation problem in Eq. 2.3 to involve an amount of learning, the learner must not be able to directly compute an exact solution, or the problem reduces to mere computation. Instead, the learning aspect arises as the learner must search through the space  $\mathcal{F}$  of models and find an *approximately* optimal model. Learning in this setting is the *process* of iteratively searching for a better model (Rumelhart et al., 1986; Mangasarian & Solodov, 1994; Cortes & Vapnik, 1995).

As with the problem formulation, there are many ways to formalise such iterative processes. This thesis focuses on gradient-based learning, as this has proven to be a particularly efficient class of algorithms for learning when the model  $f_\theta$  is a neural network (Bennett & Parrado-Hernández, 2006; Goodfellow et al., 2016). Similarly, as contemporary meta-learning goes beyond hyperparameter tuning and involves high-dimensional parameters space, often just as large the task learner’s parameter space (Finn et al., 2017), we focus on gradient-based meta-learning algorithms. While other forms of meta-learning are possible (Bergstra et al., 2011; Chen et al., 2017; Jaderberg et al., 2017b; Falkner et al., 2018; Drake et al., 2020), they are generally less efficient in such high-dimensional search spaces.

With that said, a particular challenge with neural networks is that they define complex non-linear functions. Thus, optimisation problems under neural networks are generally not convex, ruling out many efficient optimisation algorithms that have been designed for convex optimisation (Sutskever et al., 2013; Martens, 2010). Instead, they exhibit complex loss surfaces (Li & Hoiem, 2016; Freeman & Bruna, 2017; Garipov et al., 2018) with intriguing properties. For instance, it seems likely that—due to the high-dimensionality of the parameter space of neural networks—local minima are relatively low cost

compared to the global minima (e.g. [Saxe et al., 2013](#)). Even so, solutions found by neural networks are easily perturbed, to the point that  $\epsilon$ -small perturbations can completely destroy a model ([Szegedy et al., 2014](#)). [Czarnecki et al. \(2019\)](#) showed that a sufficiently deep and wide neural network contains every low-dimensional pattern, suggesting that deep neural networks have a fractal-like quality to them. [Arora et al. \(2018\)](#) showed that over-parameterising neural networks induces momentum-like dynamics during optimisation, allowing over-parameterised models to converge faster. [Izmailov et al. \(2018\)](#) demonstrates that stochastic Polyak Averaging during optimisation finds flatter minimas than conventional parameter updates, further hinting at the sheer variety and complexity of local geometries in a neural network's loss surfaces. Due to this complexity, neural networks have thus far defied more advanced optimisation algorithms and tend to favour first-order methods such as gradient descent ([Goodfellow et al., 2016](#), p. 298). This algorithm is defined by a randomly chosen initialisation  $\theta_0$ , after which parameters evolve by

$$\theta_{k+1} = \theta_k - \alpha_k \nabla \mathcal{L}(\theta_k; \mathcal{B}_k), \quad (2.7)$$

where  $\{\alpha_k\}_{k \in \mathbb{N}}$  is a sequence of learning rates and  $\nabla \mathcal{L}$  is the gradient with respect to  $\theta$  evaluated at  $\theta_k$  under data  $\mathcal{B}_k$ . Different choices of data result in slightly different types of algorithms. When  $\mathcal{B}_k = \mathcal{D}$ , the algorithm is known as batch gradient descent. When  $\mathcal{B}_k \subset \mathcal{D}$  is a randomly sampled subset, the algorithm is known as stochastic gradient descent, since the gradient becomes a random variable. If  $\mathcal{B}_k$  is an i.i.d. sample from  $\mathcal{D}$ , then stochastic gradient descent is an unbiased estimate of the true gradient and can lead to faster convergence for the same computational budget ([Bottou, 1998](#)).

The exact nature of solutions found by neural networks is subject to much debate (e.g. [Kawaguchi, 2016](#); [Jacot et al., 2018](#); [Allen-Zhu et al., 2018](#); [Arora et al., 2019](#)), but it is relatively clear that a sufficiently large neural network can find a perfect solution to the data it is trained on ([Du et al., 2018](#)). A more pressing concern is the ability of a trained neural network to *generalise* to unseen data from the same data generating process ([Zhang et al., 2017](#)). Because of the large number of parameters in a neural network, the amount of data (in terms of bits) required to learn a parameterisation that generalises reasonably well can be extremely large ([Brown et al., 2020](#)).

A reason for this is that the learning process is entirely domain agnostic: the initialisation  $\theta_0$  has no prior knowledge built into it. Similarly, each parameter update relies only on the current knowledge embedded in the gradient itself

but is otherwise agnostic to the task at hand. This level of generality allows gradient descent to learn on a wide range of problems, but also makes it relatively inefficient on each of them and sensitive to manual tuning hyper-parameters, most notably the learning rate. Meta-learning offers a means of infusing useful inductive biases into the learning process, for instance by choosing a more efficient initialisation (Finn et al., 2017) or by tuning the learning rate (Li et al., 2017).

## 2.4 Reinforcement Learning

Reinforcement Learning (RL) differs from other machine learning problems in that it does not rely on an exogenously given dataset  $\mathcal{D}$ . Instead, the learner has access to a dataset  $\mathcal{D}_t$  at time  $t$  that depends on its own behaviour through some mapping  $g$  such that  $\mathcal{D}_t = g(f_\theta(x^{(i)})) \cup \mathcal{D}_{t-1}$ . Consequently, in contrast to other forms of machine learning, in RL the learner must account for how its behaviour affects the data it receives, fundamentally changing the nature of the learner’s problem. This Section presents the type of RL problem considered in this thesis, along with tools for learning. It is structured as follows: Section 2.4.1 presents the overall problem setting used in this thesis. Section 2.4.2 presents the generalised value improvement framework and specifically the  $Q$ -learning algorithm. Finally, Section 2.4.3 presents the policy-gradient approach.

### 2.4.1 The Reinforcement Learning Problem

RL is a behavioural approach to artificial intelligence with strong links to constructivist thinking in neurological science (Silvetti & Verguts, 2012; Botvinick et al., 2020). The central object in RL is a goal-directed *agent*, which plays the role of a learner. The agent interacts with an *environment* by receiving observations from the environment and in turn taking actions that influences how the environment’s internal state evolves. The agent must take actions such that it achieves its predefined goal, defined in terms of the state of the environment.

A core tenet of reinforcement learning is that the agent’s goal can be encoded in a scalar reward function. The agent receives rewards conditioned on what action it takes in a given state so that the goal of the agent can be formalised as maximising rewards by taking appropriate actions (Sutton & Barto, 1998, p. 6).

This introduces a fundamental trade-off for an RL agent that does not (typically) arise in other forms of machine learning. On the one hand, the agent should take actions that it believes are going to yield the most rewards; on the other hand, the agent must also explore other actions so that it can learn what

behaviours lead to the highest amount of rewards. This trade-off between exploiting what the agent already knows and exploring what it does not is known—known as the exploration-exploitation dilemma—is central to RL and separates it from the typical supervised or unsupervised learning problem.

There are several ways to formalise an RL problem. By far the most common is to rely on formalism from dynamic programming and define the RL problem as a Markov decision process (for an in-depth treatment, see [Puterman, 2014](#)). The difference between RL and dynamic programming comes down to how estimates are obtained: dynamic programming assumes the agent has access to a perfect model of the environment while RL relies on trial-and-error to construct Monte-Carlo approximations ([Sutton & Barto, 1998](#), p. 91).

This thesis considers fully-observed Markov Decision Processes (MDPs). These are defined by a tuple  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, d)$  comprising a state-space  $\mathcal{S}$ , an action-space  $\mathcal{A}$ , a transition kernel  $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ , a reward function  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \rightarrow [0, 1]$ , a discount factor  $\gamma \in (0, 1]$ , and an initial state distribution  $d : \mathcal{S} \rightarrow [0, 1]$ . A stationary policy  $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  maps a state  $s \in \mathcal{S}$  and an action  $a \in \mathcal{A}$  to a probability  $p(a | s) = \pi(a | s)$ . Stationarity is not a major restriction because the state can always be extended to include the relevant time-varying information ([Bertsekas, 1995](#), p. 12). When the policy is deterministic, it can be written as a mapping from state to action,  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ .

The relevant class of MDPs in this thesis are finite-horizon episodic MDPs. In this setting, the agent interacts with the environment through episodes. Each episode lasts for  $T$  time-steps, after which the state of the environment is reset through the initial state distribution. An episode starts with the environment sampling an initial state  $s_0 \sim d(s_0)$ . The agent observes the state  $s_0$  and responds by taking an action  $a_0 \sim \pi(a_0 | s_0)$  according to its policy. Given  $a_0$ , the environment evolves a new state according to  $s_1 \sim \mathcal{P}(s_0, a_0, s_1)$ , after which the agent receives a reward  $r_1 \sim \mathcal{R}(s_0, a_0, s_1)$ . This process is repeated for  $T$  steps. The episode is represented by a trajectory  $\tau := (s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T)$  and an *episodic return*, defined as the cumulative discounted sum of rewards:

$$G(\tau) := \sum_{t=1}^T \gamma^{t-1} r_t. \quad (2.8)$$

Note that  $G(\tau)$  is a random variable and depends the initial state distribution, the transition kernel, the reward function, and the policy. The agent’s problem

is to find the policy  $\pi$  from a set  $\Pi$  that maximise the expected return:

$$\max_{\pi \in \Pi} \mathbb{E}_{\pi, \mathcal{P}, \mathcal{R}, d}[G(\tau)]. \quad (2.9)$$

This thesis is restricted to finite MDPs where that the state-space and the action space are countably finite. Define by  $R(s_t, a_t, s_{t+1}) = \mathbb{E}[\mathcal{R}(s_t, a_t, s_{t+1})]$  the expected reward. Under these assumptions, Eq. 2.9 takes the form

$$\max_{\pi \in \Pi} \sum_{s_0 \in \mathcal{S}} d(s_0) \sum_{t=1}^T \gamma^{t-1} \sum_{a_t \in \mathcal{A}} \pi(s_t, a_t) \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1}). \quad (2.10)$$

An important fact is that Eq. 2.10 can be written recursively. Define the value function of  $\pi$  at time  $i$  and state  $s_i$  as

$$V_\pi(s_i) := \sum_{t=i}^T \gamma^{t-i} \sum_{a_t \in \mathcal{A}} \pi(s_t, a_t) \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1}). \quad (2.11)$$

It follows by recursion that  $V_\pi(s_i) = \mathbb{E}_{a_i, s_{i+1}} [R(s_i, a_i, s_{i+1}) + \gamma V_\pi(s_{i+1})]$ , which is known as the Bellman (consistency) Equation. This is a profound result in dynamic programming in that there exists a stationary policy with maximal value.

**Theorem 2.1** (Bellman Optimality). *Let  $\Pi$  be the set of all non-stationary and randomized policies on a given MDP  $\mathcal{M}$ . Define  $V^*$  by  $V^*(s) := \sup_{\pi \in \Pi} V_\pi(s) \forall s \in \mathcal{S}$ , which is finite for finite reward functions. There exists a stationary and deterministic policy  $\pi$  such that  $V_\pi(s) = V^*(s) \forall s \in \mathcal{S}$ .*

This version of the theorem is taken from [Agarwal et al. \(2020\)](#). The core result is that there exists an optimal policy that is stationary and deterministic. As a consequence, algorithms in dynamic programming and in RL focus on learning stationary policies by estimation of value functions, which involves two distinct learning problems. The first problem is the problem of *policy evaluation*: efficiently estimating  $V_\pi$ ; the second is that of *policy improvement*: efficiently improving  $\pi$ . In an RL setting, policy evaluation is made challenging by the random nature of the environment and the policy ([Sutton & Barto, 1998](#), p. 91). Policy improvement becomes challenging in the function approximator setting where convergence guarantees cannot generally be obtained.



### 2.4.2 Generalised Policy Iteration

A general strategy for finding an optimal policy is to fix some reference policy  $\pi$ , evaluate its value function, then derive a new policy  $\pi'$  that is at least as good as  $\pi$ , then repeat the process. This class of algorithms is typically called Generalised Policy Iteration. This Section briefly presents this framework; see [Sutton & Barto \(1998, p. 76\)](#) for a more detailed presentation. Central to this class of algorithms is a means of deriving an improved policy. This is done through the *action-value function*, which is defined similarly to the value function:

$$Q_\pi(s_i, a) := \mathbb{E}_{s_{i+1}}[R(s_i, a, s_{i+1}) + \gamma V_\pi(s_{i+1})] \quad \forall (s_i, a) \in \mathcal{S} \times \mathcal{A}. \quad (2.12)$$

The action-value function, or  $Q$ -function, evaluates a strategy of taking action  $a$  in state  $s_i$ , and thereafter following policy  $\pi$ . The policy improvement theorem states that if there exists a  $\pi'$  such that  $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$  for all  $s \in \mathcal{S}$ , then  $V_{\pi'}(s) \geq V_\pi(s)$  for all  $s \in \mathcal{S}$ . Thus, policy improvement amounts to finding such a  $\pi'$ , which is straightforward under the  $Q$ -function. Define the deterministic policy

$$\pi'(s_i) := \arg \max_a Q_\pi(s_i, a) \quad \forall s \in \mathcal{S}. \quad (2.13)$$

By construction,  $\pi'$  must be equal to or better than  $\pi$  since it takes the actions that maximise the immediate reward and the future value under  $\pi$  (Eq. 2.12). Now, suppose  $V_{\pi'} = V_\pi$ . If this is the case, then  $\pi$  must be the optimal policy. This follows because, for any  $s_i \in \mathcal{S}$ , the following must be true:

$$V_\pi(s_i) = Q_\pi(s_i, \pi'(s_i)) = \max_a \mathbb{E}_{s_{i+1}}[R(s_i, a, s_{i+1}) + \gamma V_\pi(s_{i+1})]. \quad (2.14)$$

Thus, for any  $s_i$ , the value of  $\pi$  is given by taking the action that yields the highest immediate reward and the highest future value under  $\pi$ . But since  $\pi$  is defined by this in every state  $s \in \mathcal{S}$ , by divide-and-conquer  $\pi$  must be optimal. More generally, Eq. 2.14 is known as the Bellman Optimality Equation and a fundamental theorem of dynamic programming is that a stationary policy that satisfies Eq. 2.14 must be optimal in the sense of Theorem 2.1.

These results assume value functions are computed exactly. In RL, this is never true. Instead, the agent must estimate  $V_\pi$  by interacting with the environment. In particular, both  $V_\pi$  and  $Q_\pi$  can be written as an expectation under  $\pi$  for a given state (and action) and can be estimated using Monte-Carlo sampling.



Specific algorithms differ in how they estimate  $V$  or  $Q$  and how they define  $\pi'$ . The literature on this class of algorithms is large and constantly growing; a full review is beyond the scope of this thesis; comprehensive overviews can be found in (Sutton & Barto, 1998; Szepesvári, 2010; Agarwal et al., 2020). This thesis is exclusively concerned with a subclass of algorithms that rely on  $Q$ -learning (Watkins & Dayan, 1992).

In  $Q$ -learning policy evaluation and policy improvement are compressed into a single computation that takes place online as the agent is interacting with the environment. In a finite MDP, the  $Q$ -function can be thought of as a table that stores action-values for each pair  $(s, a) \in \mathcal{S} \times \mathcal{A}$ . Specifically, at any time  $t$ , given state  $s_t$ , a greedy action is taken under  $\pi'$  in Eq. 2.13. The resulting transition  $(s_t, a_t, r_t, s_{t+1})$  is used to update the  $Q$ -value at  $(s_t, a_t)$  by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( r_t + \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right). \quad (2.15)$$

By inspection, this update can be seen as the difference between the left-hand side and the right-hand side of the Bellman Optimality Equation (Eq. 2.14). Hence,  $Q$ -learning can be thought of as directly estimating the optimal  $Q^*$ , rather than separating policy evaluation and policy improvement into two distinct steps. Watkins & Dayan (1992) showed that if each action-state pair is visited infinitely often, then  $Q$ -learning converges to the optimal policy.

In large MDPs,  $Q$  cannot be represented in tabular format and some form of function approximation is required. Function approximation complicates learning considerably because parameter updates change the value estimate in every, not only the state used to compute the update. As such,  $Q$ -learning under function approximation is generally not guaranteed to converge.

The standard  $Q$ -learning algorithm for deep neural networks, and hence of particular interest in this thesis, is the DQN algorithm (Mnih et al., 2013). A neural network  $Q_\theta$  serves as an approximation of  $Q$ . A greedy policy is derived in the usual manner; what differs is the updates to  $Q_\theta$ , which now takes the form of a stochastic gradient update. To ensure that updates to  $\theta$  stay true to value estimates in as many states as possible, updates are computed on mini-batches of transitions. These are sampled from a replay buffer of historical data, giving rise to the update

$$\theta_{k+1} = \theta_k + \alpha \sum_{i=1}^N \left( \max_{a'} \gamma Q_{\bar{\theta}}(s_{i+1}, a') + r_i - Q_\theta(s_i, a_i) \right) \nabla Q_{\theta_k}(s_i, a_i), \quad (2.16)$$

where  $\bar{\theta}$  are target parameters that follow  $\theta$  at a slower rate to ensure stability. This is not a proper gradient-descent algorithm and hence lack any guarantees of convergence. Because of this, deep  $Q$ -learning is susceptible to unstable learning dynamics, for which a variety of improvements have been proposed (Mnih et al., 2015; Schaul et al., 2015; Van Hasselt et al., 2016; Wang et al., 2016b).

### 2.4.3 Reinforcement Learning with Policy Gradients

Generalised Policy Improvement relies on first constructing a value estimate and then deriving a policy from the estimate. Another approach—that is closer in spirit to other forms of machine learning—is to parameterise the policy directly, denoted  $\pi_\theta$ , and formulate the agent’s problem (Eq. 2.9) as an optimisation problem over  $\theta$ :

$$\max_{\theta \in \Theta} \mathbb{E}_{\pi_\theta, \mathcal{P}, \mathcal{R}, d}[G(\tau)]. \quad (2.17)$$

This optimisation problem is similar to the machine learning problem defined in Eq. 2.3, but differs in that  $\theta$  features in the expectation. Gradient based optimisation is therefore more challenging, as the derivative must factor in how the data distribution changes with  $\theta$ , which is typically intractable unless the agent has a perfect (differentiable) model of the environment.

Fortunately, Williams & Peng (1991) showed the gradient of the objective function in Eq. 2.17 can be computed under a Monte-Carlo estimate, known as the REINFORCE algorithm. This algorithm leverages that the objective function can be written in terms of the probability of a trajectory, i.e.  $\mathbb{E}[G(\tau)] = \sum_\tau p(\tau)G(\tau)$ . Since  $G(\tau)$  is a sum of rewards,  $\theta$  features only in the probability of the trajectory. This means that the gradient can be written as  $\nabla_\theta \mathbb{E}_\tau[G(\tau)] = \mathbb{E}_\tau[G(\tau) \nabla_\theta \log p(\tau)]$ , using that  $\nabla f(x) = f(x) \log \nabla f(x)$ . Since  $p(\tau) := d(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$ , the policy-gradient can be written as

$$\nabla_\phi \mathbb{E}_{\pi_\phi, \mathcal{P}, \mathcal{R}, d}[G(\tau)] = \mathbb{E}_{\pi_\phi, \mathcal{P}, \mathcal{R}, d} \left[ G(\tau) \sum_{(s_t, a_t) \in \tau} \nabla_\phi \log \pi_\phi(a_t | s_t) \right]. \quad (2.18)$$

REINFORCE estimates the gradient by sampling trajectories under  $\pi_\theta$  to compute the right-hand side of Eq. 2.18. While unbiased, it can have high variance. For this reason, it is often more convenient to instead estimate the

policy gradient using the equivalent formulations (cf. [Agarwal et al., 2020](#))

$$\nabla_{\phi} \mathbb{E}_{\pi_{\phi}, \mathcal{P}, \mathcal{R}, d}[G(\tau)] = \frac{1}{1-\gamma} \mathbb{E}_{(s,a) \sim \mu_{\pi}} [Q_{\pi_{\theta}}(s, a) \nabla_{\phi} \log \pi_{\phi}(a | s)] \quad (2.19)$$

$$= \frac{1}{1-\gamma} \mathbb{E}_{(s,a) \sim \mu_{\pi}} [A_{\pi_{\theta}}(s, a) \nabla_{\phi} \log \pi_{\phi}(a | s)], \quad (2.20)$$

where  $A_{\pi_{\theta}}(s, a) := Q_{\pi_{\theta}}(s, a) - V_{\pi_{\theta}}(s)$  is the advantage function and  $\mu_{\pi}$  is the stationary state distribution under  $\pi$  (the fraction of times the agent is in state  $s$ ). When the value function is not known, estimating the policy gradient involves an estimate of the (action-)value function ([Sutton et al., 1999](#)). This typically introduces a bias, but leads to significantly lower variance and is often preferred. This class of algorithms is collectively referred to as actor-critic algorithms ([Konda & Tsitsiklis, 2000](#)).

This concludes the presentation of reinforcement learning methods used in this thesis. There are many more approaches to reinforcement learning and several fundamental challenges not mentioned in this brief presentation. For a broader view of the topic, see for instance [Sutton & Barto \(1998\)](#) or [Agarwal et al. \(2020\)](#).

Irrespective of learning algorithm, reinforcement learning is fundamentally a problem of delayed feedback. The agent must take a full sequence of actions  $(a_0, \dots, a_{t-1})$  to obtain a full sequence of rewards  $(r_1, \dots, r_T)$  before it has complete information to evaluate the return its policy is yielding. Thus a fundamental learning problem in reinforcement learning is the *credit-assignment* problem, meaning the agent must infer what actions in the sequence lead to the return it observed. Similarly, by committing to a certain sequence of actions, the agent foregoes experiences that it could have obtained under another action sequence. This results in an exploration problem, in the sense that the agent must search for the experience required to learn a useful policy.

Both these problems are amenable to meta-learning. For instance, credit assignment can be tackled by meta-learning the learning rule itself ([Xu et al., 2018b](#)). Similarly, given a distribution of tasks, one can meta-learn an exploration strategy ([Alet et al., 2020](#)). There are many more approaches one can take to meta-learning in the reinforcement learning setting, most of which bears close resemblance to the typical machine-learning problem. In particular, reinforcement learning can be viewed as a maximisation problem in some parameters, allowing a unified view of meta-learning as learning over a distribution of optimisation problems.

## 2.5 Summary

To provide a coherent and unified view of a machine that learns, this thesis takes the perspective of machine learning as an optimisation problem. In terms of machine learning, this view can be motivated both from the point of view of maximum likelihood estimation and empirical risk minimisation. In terms of reinforcement learning, the problem itself is defined as a maximisation problem, while algorithms differ in terms of how they go about solving the problem.

Given a unified view of a machine that learns as an optimisation problem, meta-learning can be formalised as the problem of learning *how to learn* over a distribution of such problems. Next, Chapter 3 formalises this notion into a framework amenable to machine learning, as well as place meta-learning in context of relevant historical and contemporary research.

## 3 Machines that Learn to Learn

---

As Chapter 2 formalises the notion of a machine that learns, this chapter formalises the notion of a machine that meta-learns. This chapter builds on the formalism of a machine learning problem and frames meta-learning as a learning problem over a distribution of learning problems. From this view, meta-learning can be formalised as a bi-level optimisation. While not all of meta-learning fits within this framework, it provides a unifying language from which to discuss meta-learning as viewed in this thesis.

The chapter is structured as follows. Section 3.1 provides a high-level framework for meta-learning. Section 3.2 surveys related fields while Section 3.3 provides an overview of the literature on meta-learning from a historical perspective. Section 3.4 discusses the mechanics of meta-learning, while Section 3.5 details the specific meta-learning setting used in most contemporary meta-learning and in this thesis.

### 3.1 The Meta-Learning Problem

The previous chapter highlights that an important consideration in a machine learning problem is the inductive bias brought to bear on a learning problem. Meta-learning can be viewed as learning inductive biases from data—broadly speaking—act on a learning problem by priming the learner towards certain solutions. The type of model, loss function, learning rule, and problem formulation are all inductive biases that can be meta-learned.

At the heart of meta-learning lies the no free-lunch theorem (Wolpert & Macready, 1997); there is no unique set of inductive biases that will be optimal for every problem. Rather, each class of problems might have a distinct set of inductive biases that makes learning efficient. Some of these can be specified manually by us as designers—itself a form of meta-learning. Others can be meta-learned from data without human intervention. Broadly speaking, there are two levels to the meta-learning problem.

At the highest level, the meta-level, meta-learning is a machine learning problem defined over a distribution of machine-learning problems. Taking an optimisation-based view of machine learning, as per Chapter 3, meta-learning can be formulated in the same form as the canonical machine learning problem (Eq. 2.4). First, let  $\omega$  represent the learner’s meta-knowledge. Second, let  $\mathcal{T}$  denote a set of tasks, where each task is a machine learning problem (Eq. 2.3). Meta-learning can then be formalised as a machine learning problem defined over machine learning problems:

$$\min_{\omega \in \Omega} \mathcal{J}(\omega, \mathcal{T}). \quad (3.1)$$

This learning problem is akin to Eq. 2.4, but differs in that learns  $\omega$  over  $\mathcal{T}$ . For learning to be well-defined,  $\omega$  must be able to influence the solution to machine-learning problems  $\tau$  in  $\mathcal{T}$ . Thus, at the lowest level of the meta-learning problem,  $\omega$  alters how a solution to a machine-learning problem is found.

Specifically, a machine learning problem can be defined as a tuple  $\tau = (\mathcal{L}, \mathcal{D}, \mathcal{F})$  that specifies the objective  $\mathcal{L}$ , the data available  $\mathcal{D}$ , and the search space  $\mathcal{F}$  to select a model  $f_\theta$  from. A solution is given by an algorithm  $\mathcal{A} : \tau \rightarrow \theta^\tau$  that maps a task  $\tau$  to a solution  $\theta^\tau$ . Meta-learning influences the machine learning problem by acting on the solution; either the map  $\mathcal{A}$  itself or its inputs.

For instance, the meta-learner can select the best algorithm from a set of known algorithms to fit the task at hand (Leite et al., 2012; van Rijn et al., 2015), or it can learn an algorithm from scratch (Andrychowicz et al., 2016; Ravi & Larochelle, 2017), or tune a given algorithm’s parameters (Finn et al., 2017). The learning objective  $\ell$  can be meta-learned (Sung et al., 2018; Kirsch et al., 2019; Oh et al., 2020), as can the data  $\mathcal{D}$ , for instance by meta-learning preprocessing pipelines (Vilalta et al., 2004) or a data-generative process to expand the amount of training data (Zhang et al., 2018; Seo et al., 2018). Meta-learning can restrict the search space, for instance by defining the model  $f_\theta$  in a meta-learned metric space (Vinyals et al., 2016; Snell et al., 2017) or by meta-learning feature representations (Lee & Choi, 2018).

In summary, meta-learning acts on standard machine learning through the learning process. Different meta-learners will affect different aspects of the learning process. In its most general form, a meta-learner  $\omega = (\omega_{\mathcal{A}}, \omega_{\mathcal{L}}, \omega_{\mathcal{D}}, \omega_{\mathcal{F}})$  act on a machine learning problem in Eq. 2.4 through

$$\arg \min_{\theta \in \Theta} \mathcal{L}(\theta; \mathcal{D}) \approx \theta(\omega) = \omega_{\mathcal{A}}(\mathcal{A}) \left( \omega_{\mathcal{L}}(\mathcal{L}), \omega_{\mathcal{D}}(\mathcal{D}), \omega_{\mathcal{F}}(\mathcal{F}) \right). \quad (3.2)$$

Intuitively, the goal of meta-learning is to provide the learner with useful meta-knowledge that facilitates learning. For instance, meta-learning can help improve the speed of learning (Andrychowicz et al., 2016; Flennerhag et al., 2019), the final performance of a learned model (Vinyals et al., 2016; Finn et al., 2017), or the model’s robustness (O’Sullivan et al., 2000; Zhang et al., 2018). Meta-learning differs from conventional machine learning in that it contains two nested learning problems, often referred to as an *inner loop* and an *outer loop*.

The inner loop (Eq. 3.2) solves a specific task, i.e. a machine learning problem. Tasks can differ both in terms of objective, data, solution space, and learning algorithm. A common assumption in the literature is that tasks differ only in terms of data, so that a task  $\tau$  can be identified with its data-generating process  $p^\tau : (x, y) \mapsto (0, 1)$ . Methods developed in this thesis do not make such a stringent assumption. However, it is assumed that the learner’s model class (i.e.  $f$ ) is identical across tasks so that  $\theta$  is of fixed size.

The *outer-loop* (Eq. 3.4) is distinct from other forms of machine learning in that it is a learning problem over some set of tasks. While the inner loop is concerned with learning a task-specific model, the outer loop is concerned with finding *meta-knowledge* that facilitates learning such models on any task in the designated task space  $\mathcal{T}$ . Thus the meta-learning problem is akin to conventional machine learning but operates on a higher level of abstraction. Whereas the goal of conventional machine learning (e.g. the inner loop) is to learn a model  $f_\theta$  that generalises from a set of training data  $\mathcal{D}^\tau$ , the goal of meta-learning is to learn an algorithm  $\mathcal{A}_\omega$  that generalises from a set of training tasks  $\mathcal{T}$ .

Not all of meta-learning can be formulated in this way, but this characterisation provides a clear distinction between conventional machine learning and meta-learning. In particular, it highlights the role of meta-knowledge: to facilitate the learning problem faced by the learner. Hence meta-learning is the problem of learning what this meta-knowledge should be. This can be achieved by, for instance, meta-learning the initialisation of gradient descent (Finn et al., 2017), the learning rate (Li et al., 2017) or similar hyper-parameters (Xu et al., 2018b), meta-learning causal inference mechanisms (Bengio et al., 2019), exploration strategies for reinforcement learning (Alet et al., 2020), or meta-learn fundamental machine learning methods such as gradient descent and neural networks (Real et al., 2020).

## 3.2 Related Fields

**Transfer learning** A related but distinct field, where the goal is to transfer knowledge from a pre-trained model on a source task to a new model being trained on a target task (Pratt, 1993; Caruana, 1997). While similar to meta-learning, it differs in that the source model is trained on the source task without taking into account that it will later be used for knowledge transfer.

Hence, transfer learning is not a learning to learn paradigm since the source model does not take into account what information might be useful when training the target model. While ignoring the transfer process can limit knowledge transfer (Higgins et al., 2017; Achille et al., 2018), transfer learning has enjoyed widespread success by training a large model on a very large source task that forces the model to encode relatively general knowledge about the domain that can be useful in a range of related tasks. This technique can be highly efficient in computer vision (e.g. Deng et al., 2009; Donahue et al., 2014; Sharif Razavian et al., 2014; Russakovsky et al., 2015) and is also behind recent advances in natural language processing (e.g Howard & Ruder, 2018; Yang et al., 2019; Brown et al., 2020).

**Hyper-parameter tuning** A closely related but distinct field that specialises on hyper-parameter tuning in the single-task setting (Bergstra et al., 2011). Some work in meta-learning (Xu et al., 2018b; Zahavy et al., 2020) are in fact closely related to hyper-parameter tuning algorithms (Bengio, 2000; Maclaurin et al., 2015; van Erven & Koolen, 2016). Hyper-parameter tuning differs from meta-learning in that it considers a smaller search space, thus allowing a broader range of optimisation methods, such as evolutionary strategies (Jaderberg et al., 2017b; Drake et al., 2020) or Bayesian modeling (Feurer et al., 2019; Klein et al., 2017; Falkner et al., 2018). These methods are generally not applied in meta-learning with neural networks, where gradient-based methods have shown greater empirical success to date (Hospedales et al., 2020).

**Multi-task learning** This field considers a model that must jointly solve several tasks (Caruana, 1997). This forces the model to learn representations that generalise across tasks (Rebuffi et al., 2017), as with meta-learning. However, in contrast to meta-learning, the focus in multi-task learning is the final model and not the learning rule (Li & Hoiem, 2016; Bilen & Vedaldi, 2017). Instead, the literature place more emphasis on the structure of tasks, such as task asymmetry (e.g. Allmendinger et al., 2015). Such methods offer opportunities for further research in the context of meta-learning.



**Domain adaptation and generalisation** These are related to transfer learning but focus primarily on methods that deal with shifts in the data distribution. Domain adaptation is concerned with cases where the input or target distribution changes over time when training a model in an online learning fashion, while domain generalisation is concerned with domain shifts that arise when a trained model is deployed to a different data set (Pan & Yang, 2009). Domain generalisation focuses on methods that train the model to be robust to domain shifts (Muandet et al., 2013), which can be learned through meta-learning (Li et al., 2018), for instance by meta-learned regularisation (Balaji et al., 2018), loss functions (Li et al., 2019), or model augmentations such as parametric noise (Tseng et al., 2019). Domain adaptation has only recently received attention within meta-learning from an online learning perspective (Finn et al., 2019).

**Bayesian modelling** Because meta-knowledge is typically thought of as some *a priori* accumulated knowledge, meta-learning also has parallels to Bayesian modelling, insofar as we think of the meta-knowledge as constituting a “prior”. While some meta-learning frameworks do have a Bayesian interpretation (Fei-Fei et al., 2003; Lawrence & Platt, 2004; Edwards & Storkey, 2017; Garnelo et al., 2018; Ravi & Beaton, 2018), in general this is not the case. From this perspective, meta-learning is an inference process that learns a prior in a hierarchical Bayesian model. A more common approach in contemporary meta-learning is to view it as a hierarchical optimisation problem (Grant et al., 2018).

**Continual and lifelong learning** Typically formulated as a problem of learning a sequence of tasks continually (Parisi et al., 2019). Continual learning shares the notion of a task distribution with meta-learning, but differs in that there is no distinction between meta-training and meta-testing; instead, tasks are exposed sequentially and the goal is to maximise performance on all tasks, past and present, simultaneously. The characteristic problem in continual learning is that of catastrophic forgetting (French, 1999), by which is meant a drastic loss of performance on past tasks as the agent is training on the current task.

Solutions to this problem are often based on attempting to incrementally build a multi-task solution, for instance by incrementally growing the network (Rusu et al., 2016) for each new task (that can be intermittently distilled (Schwarz et al., 2018)), replaying data from past tasks (Riemer et al., 2018), constraining parameter updates to lie close to past solutions (Kirkpatrick et al., 2017; Zenke et al., 2017) or in other ways prevent changing parameters that are important for past tasks (Serrá et al., 2018), or by learning generative models of task distributions for replay (Shin et al., 2017), along with Bayesian methods for

keeping an online posterior over tasks (Nguyen et al., 2018; Ritter et al., 2018). Continual learning differs from meta-learning in that it is not formulated as a learning-to-learn problem, though recent work has extended meta-learning to include tasks that are continual in nature, in which case the meta-learning is learning to continually learn (Javed & White, 2019; Flennerhag et al., 2020a).

**AutoML** Broadly speaking, AutoML covers topics that involve automatically discovering and tuning a model on a single task (Vilalta & Drissi, 2002; Yao et al., 2018a). As such, AutoML covers topics of automated feature-engineering and data mining (Vilalta et al., 2004), model selection (Kotthoff et al., 2017), and hyper-parameter optimisation (Shahriari et al., 2016), to name a few. These problems are distinct from meta-learning in that they are primarily focused on automating manual aspects of the machine-learning pipeline on a given problem. Most of AutoML framed as learning to learn, in the sense that the meta-learner is trained and evaluation on a single task. However, recent work lie at the intersection of the two (Vanschoren, 2018; Real et al., 2020).

### 3.3 A Historical Perspective

Early works in meta-learning, often referred to as algorithms for *learning to learn*, thought of meta-learning as a general (i.e. Turing Complete) machine that would take a description of the current model and current experience as input and output a new model (Schmidhuber, 1987; Thrun & Pratt, 1998). Implemented as a Recurrent Neural Network (RNN),  $f_\theta$ , the parameters of the network  $\theta$  corresponds to meta-knowledge while the hidden state  $h$  is an abstract representation of both its history and the current model (Hochreiter et al., 2001). If the input to this RNN contains its most recent output  $y$ , the RNN can be interpreted as self-referential in the sense that it can learn to implement a loss function and an update rule (Schmidhuber et al., 1996; Hochreiter et al., 2001). What characterises this form of meta-learning, or learning to learn, is that there is no real distinction between the learning rule and the model; the model embodies the learning rule. Meta-learning in this context amounts to unrolling the RNN over a tasks's dataset. Given a current hidden state  $h$ , a past input  $x$  with target  $y$ , the meta-learner adapts to the task by updating its hidden state  $h'$  while making a prediction  $\hat{y}'$  for a new datapoint  $x'$  from the task:

$$\hat{y}', h' = f_\theta(x', y, h). \quad (3.3)$$

Meta-learning arises by learning  $\theta$  over a dataset; this allows the network to

associate a past input  $x$  with its target  $y$  and relate that to the prediction it made at the time,  $\hat{y}$ . This knowledge is captured in the hidden state  $h$  and can be meta-learned so that the network learns to adapt its predictions to fit the task. [Duan et al. \(2016\)](#) and [Wang et al. \(2016a\)](#) independently extended this framework to reinforcement learning. In this setting, there is no target  $y$ . Instead, the input to the meta-learner is its previous action  $a$  along with the reward  $r$  it obtained for the transition  $(s, a, r, s')$ :

$$a', h' = f_{\theta}(s', a, r, h). \quad (3.4)$$

This system can—in principle—learn efficient policy evaluation and policy improvement strategies, as well as how to trade off exploration and exploitation. Unrolling  $f_{\theta}$  as defined above in an environment corresponds to running its learned algorithm ([Duan et al., 2016](#); [Wang et al., 2016a](#)). However, a meta-learned RNN algorithm have no guarantees of convergence.

The extent to which a meta-learner  $f_{\theta}$  generalises depends both on the properties of the function approximator and the data it was trained on. If the data it was trained on is unrepresentative in some way, the meta-learner can be arbitrarily bad. While this can be addressed by increasing diversity of the meta-training set ([Triantafillou et al., 2020](#)), RNN-based meta-learning is ultimately at the mercy of the function approximator. In particular, an RNN does not contain any useful inductive biases for learning, but must learn these from data.

An alternative view on meta-learning that partially circumvent this issue makes a distinction between the learning rule,  $\omega$  (e.g. gradient descent), and the model  $f_{\theta}$ . Meta-learning in this form arises by separating meta parameters  $\phi$  from rapidly adaptable parameters  $\theta$ . The meta-parameters are learned at a slower time-scale to account for the dynamics that govern how  $\theta$  evolves under the learning rule  $\omega$ . From this perspective, meta-learning arises as a partitioning of learning into different time-scales ([Thrun & Pratt, 1998](#)).

In the original proposal by [Hinton & Plaut \(1987\)](#), both  $\theta$  and  $\phi$  resides within the model and forms a system where each “synapse” has two parameters, one that is rapidly adapting (i.e.  $\theta$ ) and that decays towards zero, and one that slowly adapts (i.e.  $\phi$ ) without decay. The model is parameterised by summation,  $f_{\theta+\phi}$ . The rapidly changing  $\theta$  reflects rapid adaptation to current experiences. The slowly adapting evolving  $\phi$  integrates over a longer horizon to capture general knowledge. In a manner akin to plasticity,  $\theta$  can override default behaviour in  $\phi$  as and when local context dictates but will fade from memory

if the effect is not persistent. The idea of fast and slow weights has since been explored further in a variety of settings (e.g. Schmidhuber, 1992; Gomez & Schmidhuber, 2005; Ba et al., 2016; Ha et al., 2017; Munkhdalai & Yu, 2017).

Further removed from the self-referential learning to learn system (Eq. 3.3), Bengio et al. (1991) proposed a form of meta-learning that extricates meta-parameters from the model entirely and instead places them within a learning rule  $\omega_\phi$ . In this setting, the role of adaptable parameters  $\theta$  and meta-parameters  $\phi$  are disjoint, as meta-parameters only affect the learning rule but not the model. In this regime, which is sometimes referred to as learning *how to learn*, meta-learning is a means of inferring a learning rule.

While a learning rule can be learned online in a single stream of data (Chen et al., 2016; Andrychowicz et al., 2016; Xu et al., 2018a), it is typically studied as the problem of meta-learning a learning rule that generalises across tasks, as described in Section 3.1. From this perspective, the learning rule acts on the algorithm,  $\mathcal{A}$ ; typically,  $\mathcal{A}$  is defined as iteratively applying the learning rule:

$$\mathcal{A}_{\omega_\phi} := \omega_\phi \cdot \dots \cdot \omega_\phi. \quad (3.5)$$

Meta-learning the learning rule therefore amounts to learning  $\phi$  such that repeated application of  $\omega_\phi$  yields good learning on a given task. In particular, given a task distribution  $p(\tau)$ , the meta-learning problem is typically formulated as (c.f. Hospedales et al., 2020):

$$\begin{aligned} \min_{\phi} \quad & \mathbb{E}_{\tau \sim p(\tau)} [\mathcal{L}^\tau(\theta^\tau; \mathcal{D}^\tau)] \\ \text{s.t.} \quad & \theta^\tau = \mathcal{A}_{\omega_\phi}(\tau). \end{aligned} \quad (3.6)$$

A limiting factor of this problem formulation is that the meta-learner is only evaluated in terms of final performance. In complex meta-learning problems, learning often amounts to thousands of repeated applications of  $\omega_\phi$ . This gives rise to a credit assignment problem, where a signal at the very end must be attributed across many applications of the learning rule. A core tenet of this thesis is that meta-learning can be directly formulated in terms of the *process* of learning, as opposed to the end result. Formulating meta-learning in terms of the process of learning provides significant benefits of scale (Chapter 5) and provides a more efficient and flexible form of meta-learning as it does not need to wait for final task performance (Chapter 6).

Finally, it is worth mentioning a third dominant approach to meta-learning, which focuses on meta-learning metric spaces for non-parameteric inference (Koch, 2015; Vinyals et al., 2016; Snell et al., 2017). In this setting, meta-knowledge is represented as a transformation  $g_\phi$  that induces a metric space  $\langle \cdot, \cdot \rangle_{g_\phi}$ . To adapt to a new task, the learner is given a so-called support set of task data. Predictions for other data points, so-called query points, are constructed in a non-parametric fashion by finding nearest neighbours under the meta-learned metric. To illustrate in the context of classification, when faced with a new task for which we have, let's say, one example of each class,  $\mathcal{D}^\tau = \{(x^{(i)}, y^{(i)})\}_{i=1}^C$  where  $y^{(i)} = i$ . The model makes predictions by making comparison under the meta-learned metric using some inference procedure, such as the Boltzmann distribution (Vinyals et al., 2016; Snell et al., 2017):

$$p(y' = i | x', \mathcal{D}^\tau) \propto \exp \left( g_\phi(x')^T g_\phi(x^{(i)}) \right). \quad (3.7)$$

Few-shot metric-learning methods akin to Eq. 3.7 have been an important driver meta-learning research in the last few years and appear likely to remain so for the foreseeable future.

### 3.4 The Mechanics of Meta-Learning

Contemporary works in meta-learning tend to view meta-learning as a means of abstracting away design choices and to automate as much of learning as possible (Clune, 2019). This view is partially inspired by computational evolution (Wang et al., 2019; Arulkumaran et al., 2019), procedural generation of complexity (Schmidhuber, 2013; Sukhbaatar et al., 2018; Racaniere et al., 2020; Risi & Togelius, 2020), and partially inspired by the human brain's ability to acquire new skills from experience (Jankowski et al., 2011; Lake et al., 2011). While there is truth to this view, it is important to emphasise that meta-learning is, by virtue of being a machine learning problem, also subject to the No Free Lunch Theorem. Hence, the power of meta-learning is not that it expands the number of solutions a learner can reach, but that it *restricts* the number of possible solutions a learner can achieve on a given task by providing inductive biases that primes the learner towards certain solutions—at the exclusion of other possibilities. As Schmidhuber (1987, p. 5) put it,

“Meta-capacity probably is also essential for truly flexible learning systems. The more parts of a system are accessible by the system itself (in a non-destructive manner), the more senseful self-modification may take place.”

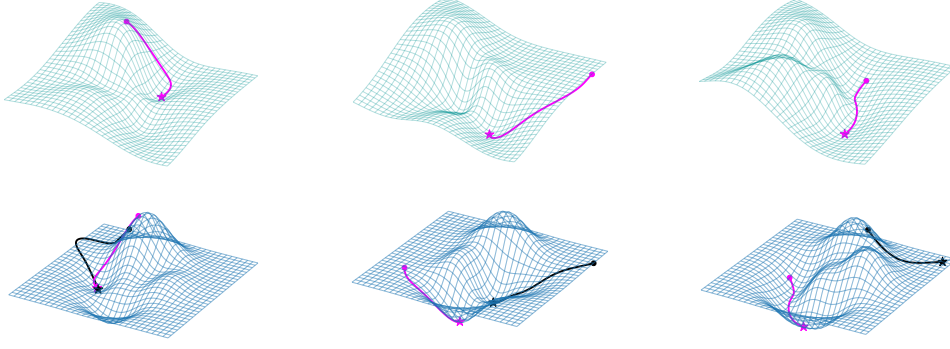


Figure 3.1: Example of how meta-learning an update rule can facilitate learning (Flennerhag et al., 2020a, Appendix 6.D). Each task  $\mathcal{L} \sim p(\mathcal{L})$  defines a distinct loss surface (bottom row). Gradient descent (black) on these surfaces struggles to find a solution. A meta-learned update rule exploits regularities across tasks to learn better initialisations and update directions (magenta) by learning a dual space (top row) where gradient descent is well behaved.

By restricting how a learner interacts with its search space, meta-learning can discover highly effective inductive biases that allow the learner to tackle problems of greater complexity. To see this in action, consider the following problem.<sup>3</sup> The meta-learner is facing a distribution  $p(\mathcal{L})$  of loss functions  $\mathcal{L} : \mathbb{R}^n \rightarrow \mathbb{R}$ . While each loss function is different, they share some underlying structure; Figure 3.1 shows three examples drawn from this distribution. For each learning problem  $\mathcal{L}$ , the learner must solve

$$\min_{\theta} \mathcal{L}(\theta). \quad (3.8)$$

The loss functions  $\mathcal{L}$  are designed such that the loss surfaces are ill-conditioned, causing gradient descent to struggle, as can be seen by the black curves in (Figure 3.1). Meta-learning can provide the learner with an inductive bias that corrects for this ill-conditioning by meta-learning how to correct for curvature over the distribution of learning problems. One way of doing this is to introduce a projection  $\Omega_{\phi} : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$  that corrects for the loss surface’s curvature. With this projection, a meta-learned update rule takes the form  $\theta = \theta - \alpha \Omega_{\phi}(\theta) \nabla \mathcal{L}(\theta)$ . Meta-parameters  $\phi$  control how the learning rule corrects for curvature. Meta-learning  $\phi$  across  $p(\mathcal{L})$  can lead to a learned projection  $\Omega_{\phi}$  that provides a smoother loss surfaces to learn on across  $p(\mathcal{L})$ , even those we have yet to see, as shown on the top row in Figure 3.1.

<sup>3</sup>Example from (Flennerhag et al., 2020a); see Section 6.D

This example illustrates the two sides of meta-learning. Introducing  $\Omega_\phi$  expands the set of learning rules that can be applied to the task distribution. On the other hand, for any one task, the meta-learned learning rule  $\Omega_\phi$  introduces a strong inductive bias that will dictate the process of learning on any task  $\mathcal{L}$ . In this way, meta-learning allows greater generality by automating certain high-level choices. Meanwhile, it provides each task-specific machine learning problem with a stronger inductive bias for learning. The cost of this is that meta-learning relies on generalisation: it inherently faces a challenge when prior experience is *not* representative of a new task. A task  $\mathcal{L}'$  that is not from  $p(\mathcal{L})$  can instead suffer from the meta-learned learning rule if that rule excludes certain solutions relevant to  $\mathcal{L}'$ . How to bridge this gap and allow the meta-learner to generalise to new task *distributions* is as of yet an open and exciting research question (Hospedales et al., 2020).

It is important to note that this property is not specific to meta-learning, the same limitation applies to any statistical process, including us humans (Griffiths et al., 2019); we are—through evolution—primed to learn certain sets of skills, while others are beyond our capacity to learn. Meta-learning should not be understood as a silver-bullet that lets an AI learn anything from data. Rather the opposite, it is a means by which an AI learns—formalised by some learning algorithm—can be restricted through relevant inductive biases that enables the AI to learn and adapt to experiences as it interacts with its environment. As such, meta-learning is a powerful framework for learning models that generalise across a class of problems. The relevant class can be playing board-games, driving cars, or the problems we as humans can solve, or beyond that.

### 3.5 Contemporary Meta-Learning with Neural Networks

Contemporary meta-learning is distinguished by its heavy use of a task distribution. The notion of a task distribution arose in the setting of *N-way-K-shot* classification, where a classifier is tasked with solving an *N*-way classification problem given *K* examples of each class (Larochelle et al., 2008; Lake et al., 2011). Earlier works used various forms of pre-training without matching training and testing conditions (Fei-Fei et al., 2003; Lawrence & Platt, 2004; Lake et al., 2011; Norouzi et al., 2014).

**Protocol** Vinyals et al. (2016) proposed to deliberately structure the training set such that it matched test-time conditions by referring to a task as tuple  $\tau := (\mathcal{D}_{\text{train}}^\tau, \mathcal{D}_{\text{test}}^\tau)$  of data sets; a training set and a test set (also known as support and query sets, respectively). The meta-learner is trained over tasks



sampled from a meta-training set  $\mathcal{T}_{\text{train}} := \{\tau^{(i)}\}_{i=1}^N$  and later evaluated on unseen tasks in the meta-test set,  $\mathcal{T}_{\text{test}} := \{\tau^{(j)}\}_{j=1}^M$ .

Beyond proposing this paradigm, they also introduced two meta-learning benchmarks that would become canonical to few-shot learning based on the Omniglot dataset (Lake et al., 2011) and the ImageNet dataset (Deng et al., 2009). These benchmarks have been used in a variety of influential works (Snell et al., 2017; Ravi & Larochelle, 2017; Finn et al., 2017, e.g) that has served to popularise the view of meta-learning as a map  $\mathcal{A}_{\omega_\phi}$  from  $\mathcal{D}_{\text{train}}^\tau$  to a task-specific model  $f_{\theta^\tau}$  that is evaluated on  $\mathcal{D}_{\text{test}}^\tau$ .

Meta-learning under this protocol is a specific instance of Eq. 3.6 where  $\mathcal{L}^\tau$  is the test performance:

$$\begin{aligned} \min_{\phi} \quad & \sum_{\tau \in \mathcal{T}_{\text{train}}} \mathcal{L}(\theta^\tau; \mathcal{D}_{\text{test}}^\tau) \\ \text{s.t.} \quad & \theta^\tau = \mathcal{A}_{\omega_\phi}(\mathcal{D}_{\text{train}}^\tau). \end{aligned} \tag{3.9}$$

This type of meta-learning can take a wide variety of forms depending on how  $\omega_\phi$  is represented and how it produces task-specific parameters  $\theta^\tau$ . In broad strokes, contemporary research on methods for meta-learning can be divided into works that view meta-learning as a parameter prediction problem, a means of learning feature representations, as a black-box function approximator, or that meta-learn aspects of a known update rule—typically gradient descent.

**Meta-learning as parameter prediction** Because of the representational capacity of neural networks, a common approach is to identify  $\omega_\phi$  with a neural network that outputs either a description of a model (as in Eq. 3.3), or predicts the model’s parameters (Bertinetto et al., 2016; Ha et al., 2017; Shaban et al., 2017; Munkhdalai et al., 2018; Gidaris & Komodakis, 2018; Qiao et al., 2018; Lee et al., 2019b). The number of parameters in the meta-learner’s neural network grows exponentially in the number of task-specific parameters to predict. Thus, these methods pretrain a feature extractor and predict a small subset of the parameters, often a final linear classifier.

**Meta-learned feature representations** An alternative approach is to use neural networks to meta-learn a feature embedding for non-parametric learning (Koch, 2015; Vinyals et al., 2016; Snell et al., 2017; Sung et al., 2018; Rodríguez et al., 2020). Others have studied how meta-learning can affect relational reasoning through attention mechanisms (Mishra et al., 2018; Ren et al., 2019;



Hou et al., 2019) or explicit memory (Graves et al., 2014) that the meta-learner learns to make use of (Santoro et al., 2016; Munkhdalai & Yu, 2017).

**Black-box meta-learning** These methods parameterise  $\omega_\phi$  as a deep neural network such as a recurrent neural network (Hochreiter et al., 2001; Andrychowicz et al., 2016; Li & Malik, 2016; Ravi & Larochelle, 2017), Neural Turing Machine (Santoro et al., 2016), or some other form of memory-based model (Ba et al., 2016; Mishra et al., 2018; Munkhdalai & Yu, 2017). As mentioned previously, the characteristic feature of these approaches is that  $\omega_\phi$  is a parameterised map from a task training set  $\mathcal{D}^\tau$  that either implicitly encode a task model in its hidden state (Hochreiter et al., 2001; Wang et al., 2016b; Duan et al., 2016) or explicitly predict its parameters (Andrychowicz et al., 2016; Li & Malik, 2016; Ravi & Larochelle, 2017; Ha et al., 2017). Many of these have the capacity to learn any learning rule by virtue of being universal function approximators (Cybenko, 1989; Hornik, 1991; Siegelmann & Sontag, 1995; Schäfer & Zimmermann, 2007), but it places the generalisation of that learning rule at the mercy of the properties of the neural network. Theoretical work on generalisation properties of neural networks suggests that they are local approximators primarily (Jo & Bengio, 2017; Justesen et al., 2018). Their complex loss surfaces (Li & Hoiem, 2016; Freeman & Bruna, 2017; Garipov et al., 2018) mean that they can easily get attracted to local minima with respect to the meta-training tasks, preventing them from learning representations that generalising deeper concepts like physics and grammar (Lake et al., 2017). Empirically, such full black-box approaches to meta-learning tend to generalise less well than other methods (Mishra et al., 2018; Finn et al., 2017), which have provoked questions as to their general feasibility (Finn & Levine, 2018).

**Gradient-based meta-learning** In contrast to these approaches, an alternative approach defines  $\omega_\phi$  as a *gradient-based* update rule and meta-learns a set of shared parameters of the gradient descent algorithm, most commonly the initialisation (Finn et al., 2017; Nichol et al., 2018; Flennerhag et al., 2019). In contrast to black-box methods, this form of meta-learning does have an inductive bias that is valid even for tasks that may be distinctly different from the meta-training set. This approach was popularised by the Model Agnostic Meta-Learner (MAML; Finn et al., 2017), which is a special form of Eq. 3.9, where  $\omega_\phi$  is defined as gradient descent and  $\phi := \theta_0$  is equal to an initialisation  $\theta_0$  that is shared across tasks. Under this notation, in the case of one gradient

step of learning per task, MAML is defined by

$$\min_{\theta_0} \mathbb{E}_{\tau \sim p(\tau)} \left[ \mathcal{L} \left( \theta_0 - \alpha \nabla \mathcal{L}(\theta_0; \mathcal{D}_{\text{train}}^\tau); \mathcal{D}_{\text{test}}^\tau \right) \right]. \quad (3.10)$$

Subsequent works have shown that it is beneficial to more directly control gradient descent by meta-learning parameters that are active at each step of adaptation. Meta-SGD (MSGD; [Li et al., 2017](#)) learns a vector of learning rates, Meta-Curvature (MC; [Park & Oliva, 2019](#)) defines a block-diagonal preconditioning matrix  $B$ , and T-Nets ([Lee & Choi, 2018](#)) embed block-diagonal preconditioning in feed-forward learners via linear projections. These differ in how the gradient-update rule is defined:

$\omega(\theta_k; \theta_0) := \theta_k - \alpha \nabla \mathcal{L}(\theta_k)$	MAML
$\omega(\theta_k; \theta_0, \phi) := \theta_k - \alpha \text{diag}(\phi) \nabla \mathcal{L}(\theta_k)$	MSGD
$\omega(\theta_k; \theta_0, \phi) := \theta_k - \alpha B(\phi) \nabla \mathcal{L}(\theta_k)$	MC
$\omega(\theta_k; \theta_0, \phi) := \theta_k - \alpha \nabla \mathcal{L}(\theta_k; \phi)$	T-Nets.

A common limitation of MAML-based meta-learners is that they backpropagate through the gradient descent process in Eq. 3.10, which results in a trajectory dependence in the meta-objective. This dependence limits them to few-shot learning as they become (1) computationally expensive, (2) susceptible to exploding/vanishing gradients, and (3) subject to a credit assignment problem ([Wu et al., 2018](#); [Antoniou et al., 2019](#); [Liu et al., 2019](#)). These limitations are discussed at depth in Chapter 6.

MAML based approaches have since proliferated. [Grant et al. \(2018\)](#) showed that MAML can be thought of as an Hierarchical Bayesian prior, which has spurred several extensions of MAML to imbue it with a Bayesian interpretation ([Kim et al., 2018a](#); [Finn et al., 2018](#); [Ravi & Beatson, 2018](#)). MAML has also been used to meta-learn latent spaces for concept or task inference, which is an effective approach to blending gradient-based and inference-based adaptation ([Zhou et al., 2018](#); [Oreshkin et al., 2018](#); [Rusu et al., 2019](#); [Lee et al., 2019a](#)). It has been extended to continual learning ([Javed & White, 2019](#)) and online meta-learning ([Finn et al., 2019](#)), and gradient-based meta-learning has recently been introduced in the context of graph neural networks ([Zhou et al., 2019](#); [Huang & Zitnik, 2020](#)), where they are used to learn metric spaces for rapid graph inference.

## 3.6 Summary

Contemporary meta-learning is typically viewed as a bi-level optimisation problem. This has proven a powerful framework for understanding the nature of meta-learning and for developing new methods that tackle increasingly challenging problems. In the few-shot learning scenario, a natural formulation is to view the meta-learner as a mapping from task description to a trained model and to meta-learn this mapping with respect to the trained model’s generalisation performance. While elegant and powerful, this approach to meta-learning comes with certain limitations and open research questions.

**Scalability** In particular, it is an “asymptotic” evaluation that only scores the meta-learner in terms of the final performance of the trained model. For meta-learning beyond few-shot learning, this poses serious challenges. First, because of an intractable credit assignment problem, where the final performance must be attributed over thousands (if not millions) of learning steps. Second, the notion of final performance may not be relevant, or even ill-defined. For long learning processes, one might equally care about the performance *during* learning; for never-ending learning process, the *only* relevant metric is the learner’s performance while learning.

**Generalisation** Contemporary meta-learning introduces a complicated inter-relationship between the algorithm’s parameters and the learner’s parameters. This interaction is dictated by the properties of the function approximators used in learning. Currently, it is an open research question as to what representation of meta-knowledge allows efficient generalisation in meta-learning.

**Exogenous Tasks** This form of meta-learning relies heavily on the notion of a task distribution. It is becoming increasingly clear that engineering sufficiently complex task distribution might not be feasible (Hospedales et al., 2020), except in certain domains such as natural language processing (Brown et al., 2020). It is yet unclear what alternatives there are to manually designed task distributions.

Inspired by these challenges, this thesis aims to provide scalable methods that take steps towards a truly general meta-learner. Chapter 4 presents a neural network architecture for rapid adaptation, with the aim of increasing the flexibility in the task learner. Chapter 5 proposes a geometric view of meta-learning as an alternative to evaluating a meta-learner in terms of final performance. The framework provides a principled path to scale up meta-

learning; Chapter 6 pushes this line of research further by proposing a meta-learner that avoids dependence on parameter trajectories in the objective. Instead, a meta-learner is evaluated in expectation over each of its learning steps, thereby encouraging meta-learners that are efficient through the learning process. Finally, Chapter 7 takes aim at task distributions and proposes a method for implicitly defining tasks within a reinforcement learning agent. Chapter 8 discusses how these strands of research may provide a potential avenue towards scalable meta-learning.

---

## II

---

### *Learning to Learn*

## 4 Learning to Dynamically Adapt

---

The first contribution of this thesis is architectural: this work proposes a novel method for adaptively parameterising a neural network. While neural networks are powerful function approximators, they are also highly inefficient in terms of parameter counts. Often, to obtain a model with good performance, the number of learnable parameters must be greater than the number of datapoints in the training set (Zhang et al., 2017). This paper focuses on one potential reason why this may be; the activation function.

Most architectures rely on weakly non-linear activation function, in that they are in fact linear over large parts of their domain. As a consequence, models need more and wider layers to increase that perhaps necessary to achieve high expressive capacity. This work proposes a mechanism for increasing the expressive capacity of a feed-forward layer by adaptively parameterising the layer.

While this paper is not explicitly concerned with meta-learning, thought the architecture is designed for rapid adaptation. Thus, in Chapter 6, this architecture is a critical component of a very efficient meta-reinforcement learning agent. From the perspective of learning to learn, adaptive layers are meta-learners, they describe how to change the model conditional on the input, much like how Schmidhuber (1987) envisioned a meta-learner. If these adaptive layers have low capacity, as is the case with standard activation function, then the meta-learner has little capacity to adapt to the context.

This reasoning can be applied to LSTMs (Hochreiter & Schmidhuber, 1997). A hierarchy of LSTMs, where one learns to adapt the parameters of the other, does significantly better than a standard LSTM. In fact, our adaptive LSTM even outperform a standard LSTM that has a larger representation space and 30% more learnable parameters.

The results presented herein suggests that learning to adapt can have significant impact on performance. We will later use these ideas in Chapter 6, where

$\omega$  is explicitly meta-learned across a distribution of tasks. In that context,  $f$  is a learning rule for the task at hand in the way imagined by Schmidhuber (1987). This learning rule is being adapted by  $\omega$  to be as effective on the current task as possible. And so rather than compressing a task distribution into a single learning rule, the architectural advances we make in this paper allow us to meta-learn a distribution over learning rules that can be learned online while adapting to a task.

## Breaking the Activation Function Bottleneck through Adaptive Parameterisation

Peer-reviewed publication ([Flennerhag et al., 2018](#)); notational modifications.

Published in *Advances in Neural Information Processing Systems*. 2018.

Authors: **Sebastian Flennerhag**, University of Manchester,  
Hujun Yin, University of Manchester,  
John Keane, University of Manchester,  
Mark Elliot, University of Manchester.

*Abstract. Standard neural network architectures are non-linear only by virtue of a simple element-wise activation function, making them both brittle and excessively large. In this paper, we consider methods for making the feed-forward layer more flexible while preserving its basic structure. We develop simple drop-in replacements that learn to adapt their parameterisation conditional on the input, thereby increasing statistical efficiency significantly. We present an adaptive LSTM that advances the state of the art for the Penn Treebank and WikiText-2 word-modeling tasks while using fewer parameters and converging in less than half the number of iterations.*

---

### 4.1 Introduction

While a two-layer feed-forward neural network is sufficient to approximate any function ([Cybenko, 1989](#); [Hornik, 1991](#)), in practice much deeper networks are necessary to learn a good approximation to a complex function. In fact, a network tends to generalise better the larger it is, often to the point of having more parameters than there are data points in the training set ([Canziani et al., 2016](#); [Novak et al., 2018](#); [Frankle & Carbin, 2019](#)).

One reason why neural networks are so large is that they bias towards linear behavior: if the activation function is largely linear, so will the hidden layer be. Common activation functions, such as the Sigmoid, Tanh, and ReLU all behave close to linear over large ranges of their domain. Consequently, for a randomly sampled input to break linearity, layers must be wide and the network deep to ensure some elements lie in non-linear regions of the activation function. To overcome the bias towards linear behavior, more sophisticated activation functions have been designed ([Clevert et al., 2015](#); [He et al., 2015](#); [Klambauer et al., 2017](#); [Dauphin et al., 2017](#)). However, these still limit all non-linearity to sit in the activation function.

We instead propose *adaptive parameterisation*, a method for learning to adapt the



parameters of the affine transformation to a given input. We present a generic adaptive feed-forward layer that retains the basic structure of the standard feed-forward layer while significantly increasing the capacity to model non-linear patterns. We develop specific instances of adaptive parameterisation that can be trained end-to-end jointly with the network using standard backpropagation, are simple to implement, and run at minimal additional cost.

Empirically, we find that adaptive parameterisation can learn non-linear patterns where a non-adaptive baseline fails, or outperform the baseline using 30–50% fewer parameters. In particular, we develop an adaptive version of the Long Short-Term Memory model (LSTM; Hochreiter & Schmidhuber, 1997; Gers et al., 2000) that enjoys both faster convergence and greater statistical efficiency.

The adaptive LSTM advances the state of the art for the Penn Treebank and WikiText-2 word modeling tasks using ~20–30% fewer parameters and converging in less than half as many iterations.<sup>4</sup> Section 4.2 presents the adaptive feed-forward layer, Section 4.3 develops the adaptive LSTM, Section 4.4 discusses related work and Section 4.5 presents empirical results.

## 4.2 Adaptive Parameterization

To motivate adaptive parameterisation, we show that deep neural networks learn a family of compositions of linear maps and because the activation function is static, the inherent flexibility in this family is weak. Adaptive parameterisation is a means of increasing this flexibility and thereby increasing the model’s capacity to learn non-linear patterns. We focus on the feed-forward layer,  $f(x) := \sigma(Wx+b)$ , for some activation function  $\sigma$ . Define the pre-activation layer as  $a = A(x) := Wx+b$  and denote by  $g(a) := \sigma(a)/a$  the activation effect of  $\sigma$  given  $a$ , where division is element-wise. Let  $G := \text{diag}(g(a))$ . We then have  $f(x) = g(a) \odot a = Ga$ , where “ $\odot$ ” denotes element-wise multiplication.<sup>5</sup>

For any pair  $(x, y) \in \mathbb{R}^n \times \mathbb{R}^k$ , a deep feed-forward network with  $L \in \mathbb{N}$  layers,  $F := f^{(L)} \circ \dots \circ f^{(1)}$ , approximates the relationship  $x \mapsto y$  by a composition of linear maps because  $x$  is sufficient to determine all activation effects  $\mathcal{G} = \{G^{(l)}\}_{l=1}^L$ . Given transformations  $\mathcal{A} = \{A^{(l)}\}_{l=1}^L$ , the network can be written

$$F(x) = (f^{(N)} \circ \dots \circ f^{(1)})(x) = (G^{(N)} \circ A^{(N)} \circ \dots \circ G^{(1)} \circ A^{(1)})(x). \quad (4.1)$$

<sup>4</sup>Code available at <https://github.com/flennerhag/alstm>.

<sup>5</sup>We ignore the measure 0 set of exceptions  $\{a \mid a_i = 0, a_i \in a\}$ .

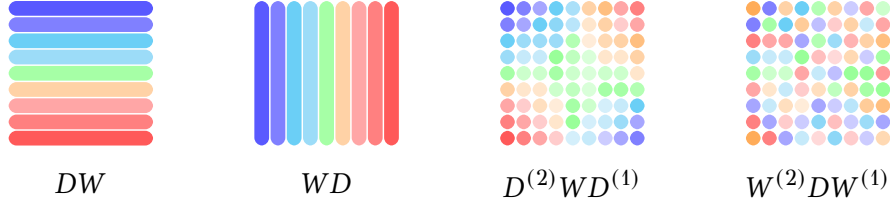


Figure 4.1: Adaptation. *Left*: output adaptation shifts the mean row-wise; *center left*: input adaptation shifts the mean column-wise; *center right*: IO-adaptation shifts mean and variance across sub-matrices; *Right*: SVA scales singular values.

A neural network can therefore be understood as learning a “prior”  $\mathcal{A}$  in parameter space around which it constructs a family of compositions of linear maps (as  $\mathcal{G}$  varies across inputs). The neural network adapts to inputs through the set of activation effects  $\mathcal{G}$ . This adaptation mechanism is weak: if  $\sigma$  is close to linear over the distribution of  $a$ , as is often the case, little adaptation can occur. Moreover, because  $\mathcal{G}$  does not have any learnable parameters itself, the fixed prior  $\mathcal{A}$  must learn to encode both global input-invariant information as well as local contextual information. We refer to this as the *activation function bottleneck*. Adaptive parameterisation breaks this bottleneck by parameterising the adaptation mechanism in  $\mathcal{G}$ , thereby circumventing these issues.

To see how the activation function bottleneck arises, note that  $\sigma$  is redundant whenever it is closely approximated by a linear function over some non-trivial segment of the input distribution. For these inputs,  $\sigma$  has no non-linear effect and such lost opportunities imply that the neural network must be made larger than necessary to fully capture non-linear patterns. For instance, both the Sigmoid and the Tanh are closely approximated around 0 by a linear function, rendering them redundant for inputs close to 0. Consequently, the network must be made deeper and its layers wider to mitigate the activation function bottleneck. In contrast, adaptive parameterisation places the layer’s non-linearity within the parameter matrix itself, thereby circumventing the activation function bottleneck. Further, by relaxing the element-wise non-linearity constraint imposed on the standard feed-forward layer, it can learn behaviours that would otherwise be very hard or impossible to model, such as contextual rotations and shears, and adaptive feature activation.

### 4.2.1 The Adaptive Feed-Forward Layer

Our goal is to break the activation function bottleneck by generalising  $\mathcal{G}$  into a parameterised *adaptation module*, thereby enabling the network to specialize parameters in  $\mathcal{A}$  to encode global, input invariant information while the adaptation module encode local, contextual information.

Consider the standard feed-forward layer, defined by one adaptation block  $f(x) = (G \circ A)(x)$ . As described above, we increase the capacity of the adaptation mechanism  $G$  by replacing it with a parameterised adaptation mechanism  $\Omega^{(j)} := \text{diag}(\omega^{(j)}(x))$ , where  $\omega^{(j)}$  is a learnable adaptation module. Note that  $\omega^{(j)}$  can be made arbitrarily complex. In particular, even if  $\omega^{(j)}$  is linear, the adaptive mechanism  $\Omega^{(j)}a$  is quadratic in  $x$ , and as such escapes the bottleneck. To ensure that the adaptive feed-forward layer has sufficient capacity, we generalise it to  $q \in \mathbb{N}$  adaptation blocks,<sup>6</sup>

$$f_\omega(x) := \sigma \left( \Omega^{(q)} W^{(q-1)} \dots W^{(1)} \Omega^{(1)} x + \Omega^{(0)} b \right). \quad (4.2)$$

We refer to the number of adaptation blocks  $q$  as the order of the layer. Strictly speaking, the adaptive feed-forward layer does not need an activation function, but it can provide desirable properties depending on the application. It is worth noting that the adaptive feed-forward layer places no restrictions on the form of the adaptation module  $\omega = (\omega^{(0)}, \dots, \omega^{(q)})$  or its training procedure. In this paper, we parameterise  $\omega$  as a neural network trained jointly with the main model. Next, we discuss adaptive feed-forward layers.

### 4.2.2 Adaptation Modules

Higher-order adaptation (i.e.  $q$  large) enables expressive adaptation modules, but because the adaptation module depends on  $x$ , high-order layers are less efficient than a stack of low-order layers. Low-order layers are surprisingly powerful; a module of order 2 that can express any other adaptation module.

**Partial Adaptation** The simplest adaptation module ( $q = 1$ ) is given by  $f_\omega(x) = W\Omega^{(1)}x + \Omega^{(0)}b$ . This module is equivalent to a mean shift and a re-scaling of the columns of  $W$ , or alternatively re-scaling the input. It can be thought of as a learned contextualized standardization mechanism that conditions the effect on the specific input. As such, we refer to this module as *input adaptation*. Its mirror image, *output adaptation*, is given by

---

<sup>6</sup>The ordering of  $W$  and  $D$  matrices can be reversed by setting the first and / or last adaptation matrix to be the identity matrix.

$f_\omega(x) = \Omega^{(1)}Wx + \Omega^{(0)}b$ . This is a special case of second-order adaptation modules, where  $\Omega^{(1)} = I$ , where  $I$  denotes the identity matrix. Both these modules are restrictive in that they only operate on either the rows or the columns of  $W$  (Figure 4.1).

**IO-adaptation** The general form of second-order adaptation modules integrates input- and output-adaptation into a jointly learned adaptation module. As such we refer to this as *IO-adaptation*,

$$f_\omega(x) = \Omega^{(2)}W\Omega^{(1)}x + \Omega^{(0)}b. \quad (4.3)$$

IO-adaptation is much more powerful than either input- or output-adaptation alone, which can be seen by the fact that it essentially learns to identify and adapt sub-matrices in  $W$  by sharing adaptation vectors across rows and columns (Figure 4.1). In fact, assuming  $\omega$  is sufficiently powerful, IO-adaptation can express any mapping from input to parameter space.

**Lemma 4.1** (IO-adaptation). *Let  $W$  be given and fix  $x$ . For any  $G$  of same dimensionality as  $W$ , there are arbitrarily many  $(\Omega^{(1)}, \Omega^{(2)})$  such that  $Gx = \Omega^{(2)}W\Omega^{(1)}x$ .*

*Proof.* Let  $y^G = Gx = \sum_i x_i g_{:,i}$ , where  $g_{:,i}$  is the  $i$ th column of  $G$ . Recall that  $\Omega = \text{diag}(\omega)$ ; for  $s \in \{1, 2\}$ , let  $\omega_i^{(s)} = \Omega_{ii}^{(s)}$  denote the diagonal entry of  $\Omega^{(s)}$ . Write

$$\Omega^{(2)}W\Omega^{(1)}x = \sum_i x_i \omega_i^{(1)} \begin{bmatrix} \omega_1^{(2)} w_{1i} \\ \vdots \\ \omega_m^{(2)} w_{mi} \end{bmatrix}. \quad (4.4)$$

Choose some  $k$  for which  $x_k \neq 0$  and set  $\omega_k^{(1)} = 1/x_k$ , with other  $\omega_{i \neq k}^{(1)} = 0$ . For each  $j \in \{1, \dots, m\}$ , set  $\omega_j^{(2)} = y_j^G / w_{jk}$ . We have

$$\sum_i x_i \omega_i^{(1)} \begin{bmatrix} \omega_1^{(2)} w_{1i} \\ \vdots \\ \omega_m^{(2)} w_{mi} \end{bmatrix} = \frac{x_k}{x_k} \begin{bmatrix} (y_1^G / w_{1k}) w_{1k} \\ \vdots \\ (y_m^G / w_{mk}) w_{mk} \end{bmatrix} = y^G. \quad (4.5)$$

This proves existence of at least one solution. Finally, since this holds for any  $x_k \neq 0$ , every linear combination of such solutions are also solutions. ■

**Singular Value Adaptation (SVA)** An interesting module arises as a special case of a third-order module, where  $\Omega^{(1)} = I$  as before. The resulting module,

$$f_{\omega}(x) = W^{(2)}\Omega^{(2)}W^{(1)}x + \Omega^{(0)}b, \quad (4.6)$$

is reminiscent of Singular Value Decomposition. However, rather than being a decomposition, it composes a projection by adapting singular values to the input. In particular, letting  $W^{(1)} = V^T A$  and  $W^{(2)} = BU$ , with  $U$  and  $V$  appropriately orthogonal, Eq. 4.6 can be written as  $B(U\Omega V^T)Ax$ , with  $U\Omega V^T$  adapted to  $x$  through its singular values. In our experiments with this form of adaptation, we initialise weight matrices as semi-orthogonal (Saxe et al., 2013), but do not enforce orthogonality thereafter.

One potential limitation of SVA is that it requires learning two separate matrices of relatively high rank. For problems where the dimensionality of  $x$  is large, the dimensionality of the adaptation space has to be made small to control parameter count. This limits the model’s capacity by enforcing a low-rank factorization, which also tends to impact training negatively (Denil et al., 2013).

SVA and IO-adaptation are simple but flexible modules that can be used as drop-in replacements for any feed-forward layer. Because they are differentiable, they can be trained using standard backpropagation. Next, we demonstrate adaptive parameterisation in the context of Recurrent Neural Networks (RNNs), where feed-forward layers are predominant.

### 4.3 Adaptive Parameterization in RNNs

RNNs are common in sequence learning, where the input is a sequence  $\{x_1, \dots, x_t\}$  and the target variable either itself a sequence or a single point or vector. In either case, the computational graph of an RNN, when unrolled over time, will be of the form in Eq. 4.1, making it a prime candidate for adaptive parameterisation. Moreover, in sequence-to-sequence learning, the model estimates a conditional distribution  $p(y_t | x_1, \dots, x_t)$  that changes significantly from one time step to the next. Because of this variance, an RNN must be very flexible to model the conditional distribution. By embedding adaptive parameterisation, we can increase flexibility for a given model size. We consider the LSTM (Hochreiter & Schmidhuber, 1997; Gers et al., 2000), where a

recurrent layer  $h_t, c_t = f_\theta(x_t, h_{t-1}, c_{t-1})$  is defined by the gating mechanism

$$\begin{aligned} c_t &= \sigma(u_t^f) \odot c_{t-1} + \sigma(u_t^i) \odot \varphi(u_t^z) \\ h_t &= \sigma(u_t^o) \odot \varphi(c_t), \end{aligned} \quad (4.7)$$

where  $\sigma$  and  $\varphi$  represent Sigmoid and Tanh activation functions respectively and each  $u_t^{s \in \{i, f, o, z\}}$  is a linear transformation of the form  $u_t^s = W^{(s)}x_t + V^{(s)}h_{t-1} + b^{(s)}$ . Adaptation in the LSTM can be derived directly from the adaptive feed-forward layer (Eq. 4.2). We focus on IO-adaptation as this adaptation module performed better in our experiments. For  $\omega$ , we use a small neural network to output a latent variable  $z_t$  that we map into each sub-module with a projection  $U^{(j)}$ :  $\omega^{(j)}(z_t) = \varphi(U^{(j)}z_t)$ . We test a feed-forward and a recurrent network as models for the latent variable,

$$z_t = \text{RELU}(Wv_t + b), \quad (4.8)$$

$$z_t = \text{LSTM}(v_t, z_{t-1}), \quad (4.9)$$

where  $v_t$  a summary variable of the state of the system, normally  $v_t = [x_t; h_{t-1}]$ . The potential benefit of using a recurrent model is that it is able to retain a separate memory that facilitates learning of local, sub-sequence specific patterns (Ha et al., 2017). Generally, we find that the recurrent model converges faster and generalises marginally better. To extend the adaptive feed-forward layer to the LSTM, index sub-modules with a tuple  $(s, j) \in \{i, f, o, z\} \times \{0, 1, 2, 3, 4\}$  such that  $\Omega_t^{(s, j)} = \text{diag}(\omega^{(s, j)}(z_t))$ . At each time step  $t$  we adapt the LSTM's linear transformations through IO-adaptation,

$$u_t^s = \Omega_t^{(s, 4)} W^{(s)} \Omega_t^{(s, 3)} x_t + \Omega_t^{(s, 2)} V^{(s)} \Omega_t^{(s, 1)} h_{t-1} + \Omega_t^{(s, 0)} b^{(s)}. \quad (4.10)$$

An undesirable side-effect of the formulation in Eq. 4.10 is that each linear transformation requires its own modified input, preventing a vectorised implementation of the LSTM. We avoid this by tying all input adaptations across  $s$ : that is,  $\Omega^{(s', j)} = \Omega^{(s, j)}$  for all  $(s', j) \in \{i, f, o, z\} \times \{1, 3\}$ . Doing so approximately halves the computation time and speeds up convergence considerably. When stacking multiple aLSTM layers, the computational graph of the model becomes complex in that it extends both in the temporal dimension and along the depth of the stack. For the recurrent adaptation module (Eq. 4.9) to be consistent, it should be conditioned not only by the latent variable in its own layer, but also on that of the preceding layer, or it will not have a full memory of the

computational graph. To achieve this, for a layer  $l \in \{1, \dots, L\}$ , we define

$$v_t^{(l)} = \left[ h_t^{(l-1)}; h_{t-1}^{(l)}; z_t^{(l-1)} \right], \quad (4.11)$$

where  $h_t^{(0)} = x_t$  and  $z_t^{(0)} = z_{t-1}^{(L)}$ . In doing so, the credit assignment path of adaption module visits all nodes in the computational graph. The resulting adaptation model becomes a blend of a standard LSTM and a Recurrent Highway Network (RHN; Zilly et al., 2017).

## 4.4 Related Work

Adaptive parameterisation is a special case of having a relatively inexpensive learning algorithm search a vast parameter space in order to parameterise the larger main model (Stanley et al., 2009; Fernando et al., 2016). The notion of using one model to generate context-dependent parameters for another was suggested by Schmidhuber (1992); Gomez & Schmidhuber (2005). Building on this idea, Ha et al. (2017) proposed to jointly train a small network to generate the parameters of a larger network; such HyperNetworks have achieved impressive results in several domains (Suarez, 2017; Ha & Eck, 2018; Brock et al., 2018b). The general concept of learning to parameterise a model has been explored in a variety of contexts, for example Schmidhuber (1992); Gomez & Schmidhuber (2005); Denil et al. (2013); Jaderberg et al. (2017a); Andrychowicz et al. (2016); Yang et al. (2018).

Parameter adaptation has also been explored in meta-learning, usually in the context of few-shot learning, where a meta-learner is trained across a set of tasks to select task-specific parameters of a downstream model (Bengio et al., 1991, 1995; Schmidhuber, 1992). Similar to adaptive parameterisation, Bertinetto et al. (2016) directly tasks a meta learner with predicting the weights of the task-specific learner. Ravi & Larochelle (2017) defines the adaptation module as a gradient-descent rule, where the meta learner is an LSTM tasked with learning the update rule to use. An alternative method pre-defines the adaptation module as gradient descent and meta-learns an initialisation such that performing gradient descent on a given input from some new task yields good task-specific parameters (Finn et al., 2017; Lee & Choi, 2018; Al-Shedivat et al., 2018).

Using gradient information to adjust parameters has also been explored in sequence-to-sequence learning, where it is referred to as dynamic evaluation (Mikolov, 2012; Graves, 2013; Krause et al., 2017). This form of adaptation



relies on the auto-regressive property of RNNs to adapt parameters at each time step by taking a gradient step with respect to previous time steps.

Many extensions have been proposed to the RNN and the LSTM (Hochreiter & Schmidhuber, 1997; Gers et al., 2000), some of which can be seen as implementing a form of constrained adaptation module. The multiplicative RNN (mRNN; Sutskever et al., 2011) and the multiplicative LSTM (mLSTM; Krause et al., 2016) can be seen as implementing an SVA module for the hidden-to-hidden projections. The mRNN improves upon RNNs in language modeling tasks (Sutskever et al., 2011; Mikolov et al., 2012), but tends to perform worse than the standard LSTM (Cooijmans et al., 2016). The mLSTM has been shown to improve upon RNNs and LSTMs on language modeling tasks (Krause et al., 2017; Radford et al., 2017). Finally, the multiplicative-integration RNN and LSTM (Wu et al., 2016b) implement a constrained output-adaptation module.

The implicit modules in the above models condition only on the input, ignoring the state of the system. In contrast, the GRU (Cho et al., 2014; Chung et al., 2014) can be interpreted as implementing an input-adaptation module on the input-to-hidden matrix that conditions on both the input and the state of the system. Most closely related to the aLSTM are HyperNetworks (Ha et al., 2017; Suarez, 2017); these implement output adaptation conditioned on both the input and the state of the system using a recurrent adaptation module. HyperNetworks have attained impressive results on character level modeling tasks and sequence generation tasks, including hand-writing and drawing sketches (Ha et al., 2017; Ha & Eck, 2018). They have also been used in neural architecture search by generating weights conditional on the architecture (Brock et al., 2018b), demonstrating that adaptive parameterisation can be conditioned on some arbitrary context, in this case the architecture itself.

## 4.5 Experiments

We analyse adaptive feed-forward networks in a controlled regression problem and on MNIST (LeCun et al., 1998). The aLSTM is tested on the Penn Treebank and WikiText-2 word modeling tasks. We use the Adam optimiser (Kingma & Ba, 2015) unless otherwise stated.

### 4.5.1 Extreme Tail Regression

To study the flexibility of the adaptive feed-forward layer, we sample  $x = (x_1, x_2)$  from  $\mathcal{N}(0, I)$  and construct the target variable as  $y = (2x_1)^2 - (3x_2)^4 + \epsilon$  with  $\epsilon \sim \mathcal{N}(0, 1)$ . Most of the data lies on a hyperplane, but the target variable grows



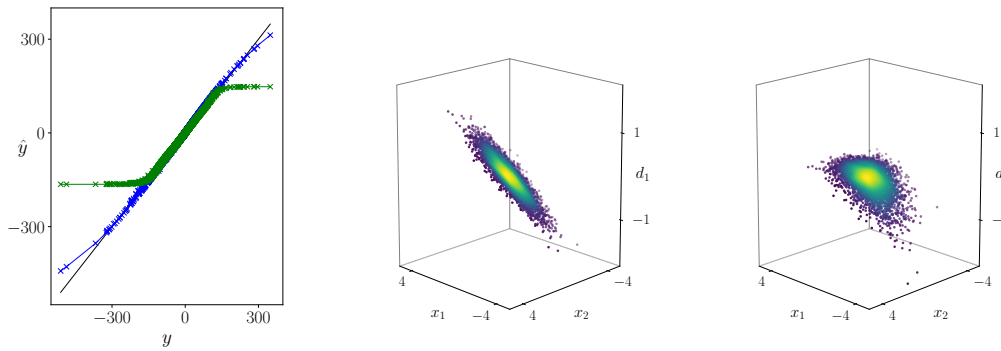


Figure 4.2: Extreme Tail Regression. *Left*: Predictions of the adaptive model (blue) and the baseline model (green) against ground truth (black). *Center & Right*: distribution of adaptive singular values.

or shrinks exponentially as  $x_1$  or  $x_2$  moves away from 0. We compare a 3-layer feed-forward network with 10 hidden units to a 2-layer model with 2 hidden units, where the first layer is adaptive and the final layer is static. We use an SVA module where  $\omega$  is a gated linear unit (Dauphin et al., 2017). Models are trained for 10 000 steps with batch size 50 and a learning rate of 0.003.

The baseline model fails to represent the tail of the distribution despite being three times larger. In contrast, the adaptive model does a remarkably good job given how small the model is and the extremity of the distribution. It is worth noting how the adaptation module encodes local information through the distribution of its singular values (Figure 4.2).

### 4.5.2 MNIST

We compare performance of a 3-layer feed-forward model against (a) a single-layer SVA model and (b) a 3-layer SVA model. We train all models with Stochastic Gradient Descent with a learning rate of 0.001, a batch size of 128, and train for 50 000 steps. The single-layer adaptive model reduces to a logistic regression conditional on the input. In comparison to a logistic regression, the marginal benefit of SVA is  $\sim 1$  percentage point gain in accuracy. A sufficiently large one-layer SVA model can even outperform a deep feed-forward baseline.

### 4.5.3 Penn Treebank

The Penn Treebank corpus (PTB; Marcus et al., 1993; Mikolov et al., 2010) is a widely used benchmark for language modeling. It consists of heavily processed news articles and contains no capital letters, numbers, or punctuation. As such, the vocabulary is relatively small at 10 000 unique words. We evaluate

Table 4.1: Train and test set accuracy on MNIST

Model	Size	Train	Test
Logistic Regression	8K	92.00%	92.14%
3-layer feed-forward	100K	97.57%	97.01%
1-layer SVA	8K	94.05%	93.86%
1-layer SVA	100K	98.62%	97.14%
3-layer SVA	100K	<b>99.99%</b>	<b>97.65%</b>

the aLSTM on word-level modeling following standard practice in training setup (e.g. Zaremba et al., 2015). As we are interested in statistical efficiency, we fix the number of layers to 2, though more layers tend to perform better, and use a module latent variable size of 100. For details see Appendix 4.A. As we are evaluating underlying architectures, we do not compare against bolt-on methods (Grave et al., 2017; Yang et al., 2018; Mikolov, 2012; Graves, 2013; Krause et al., 2017). These are equally applicable to the aLSTM.

The aLSTM improves upon baselines using roughly 30% fewer parameters, a smaller hidden state size, and fewer layers while converging in fewer iterations (Table 4.2). For the standard LSTM to converge at all, gradient clipping is required and dropout rates must be reduced by  $\sim 25\%$ ; a percentage point change to these rates caused either severe overfitting or failure to converge. Taken together, this indicates that adaptive parameterisation exhibits both superior stability and increased model capacity; we explore both properties further in Sections 4.5.5 and 4.5.6. Melis et al. (2018) applies a large-scale hyper-parameter search to an LSTM version with tied input and forget gates and inter-layer skip-connections (TG-SC LSTM), making it a challenging baseline that the aLSTM improves upon by a considerable margin.

Previous state-of-art performance was achieved by the ASGD Weight-Dropped LSTM (AWD-LSTM; Merity et al., 2018), which uses regularisation, optimisation, and fine-tuning techniques designed specifically for language modeling<sup>7</sup>. The AWD-LSTM requires approximately 500 epochs to converge to optimal performance; the aLSTM outperforms the AWD-LSTM after 144 epochs and converges to optimal performance in 180 epochs. Consequently, even if the AWD-LSTM runs on top of the CuDNN implementation of the LSTM, the aLSTM converges approximately  $\sim 25\%$  faster in wall-clock time. In summary, any

<sup>7</sup>Public release of their code at <https://github.com/salesforce/awd-lstm-lm>

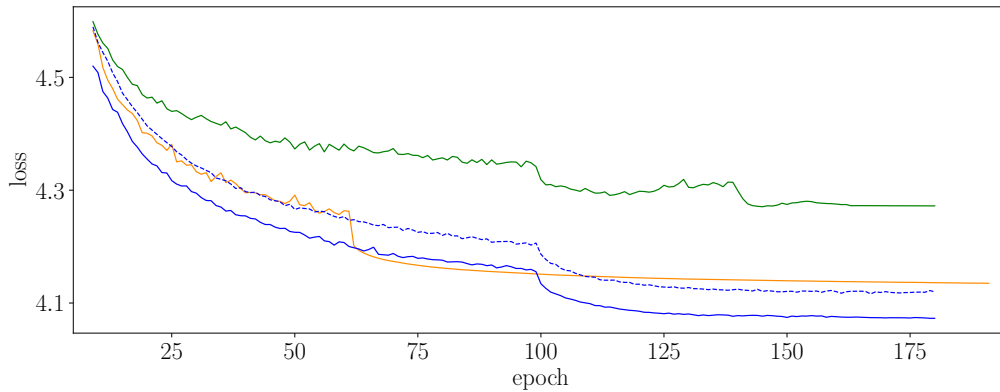


Figure 4.3: Validation loss on PTB for our LSTM (green), aLSTM (blue), aLSTM with feed-forward module (dashed), and the AWD-LSTM (orange; Merity et al., 2018). Drops correspond to learning rate cuts. 1-seed runs.

form of adaptation is beneficial, and a recurrent adaptation model (Eq. 4.9) enjoys both fastest convergence rate and best final performance in this experiment.

#### 4.5.4 WikiText-2

WikiText-2 (WT2; Merity et al., 2017) is a corpus curated from Wikipedia articles with lighter processing than PTB. It is about twice as large with three times as many unique tokens. We evaluate the aLSTM using the same settings as on PTB, and additionally test a version with larger hidden state size to match the parameter count of current state of the art models. Without tuning for WT2, both outperform previously published results in 150 epochs (Table 4.3) and converge to new state of the art performance in 190 epochs. In contrast, the AWD-LSTM requires 700 epochs to reach optimal performance. As such, the aLSTM trains  $\sim 40\%$  faster in wall-clock time. The TG-SC LSTM in Melis et al. (2018) uses fewer parameters, but its hyper-parameters are tuned for WT2, in contrast to both the AWD-LSTM and aLSTM. We expect that tuning hyper-parameters specifically for WT2 would yield further gains.

#### 4.5.5 Ablation Study

We isolate the effect of each component in the aLSTM through an ablation study on PTB. We adjust the hidden state so that every configuration has approximately 17M learnable parameters. We use the same hyper-parameters for all models except for (a) the standard LSTM (see above) and (b) the aLSTM with an output-adaptation module and feed-forward adaptation (this configuration needed slightly lower dropout rates to converge to good performance).

Table 4.2: Validation and test set perplexities on Penn Treebank. All results except those from Zaremba et al. (2015) use tied input and output embeddings (Press & Wolf, 2017).

Model	Size	Depth	Valid	Test
LSTM, Zaremba et al. (2015)	24M	2	82.2	78.4
RHN, Zilly et al. (2017)	24M	10	67.9	65.4
NAS, Zoph & Le (2017)	54M	—	—	62.4
TG-SC LSTM, Melis et al. (2018)	10M	4	62.4	60.1
TG-SC LSTM, Melis et al. (2018)	24M	4	60.9	58.3
AWD-LSTM, Merity et al. (2018)	24M	3	60.0	57.3
LSTM	20M	2	71.7	68.9
aLSTM, feed-forward module (Eq. 4.8)	17M	2	60.2	58.0
aLSTM, recurrent module (Eq. 4.9)	14M	2	59.6	57.2
aLSTM, recurrent module (Eq. 4.9)	17M	2	58.7	56.5
aLSTM, recurrent module (Eq. 4.9)	24M	2	57.6	55.3

As Table 4.4 shows, any form of adaptation yields a significant performance gain. Going from a feed-forward adaptation model (Eq. 4.8) to a recurrent adaptation model (Eq. 4.9) yields a significant improvement irrespective of module, and our hybrid RHN-LSTM (Eq. 4.11) provides a further boost. Similarly, moving from a partial adaptation module to IO-adaptation leads to significant performance improvement under any adaptation model. These results indicate that the LSTM is constrained by the activation function bottleneck and increasing its adaptive capacity breaks the bottleneck. While we have focused on LSTM as the base model, Eq. 4.10 applies to any recurrent network. A promising avenue for further research is to incorporate this form of adaptation with other architectures, such as convolutional layers (LeCun et al., 1995) and attention mechanisms (Bahdanau et al., 2015; Vaswani et al., 2017).

#### 4.5.6 Robustness

We further study the robustness of the aLSTM with respect to hyper-parameters. We limit ourselves to dropout rates and train for 10 epochs on PTB. All other hyper-parameters are held fixed (c.f. Appendix 4.A). For each model, we draw 100 random samples uniformly from intervals of the form  $[r - 0.1, r + 0.1]$ , with  $r$  being the optimal rate found through previous hyper-parameter tuning. The two models exhibit very different distributions (Figure 4.4). The distribution of the aLSTM is tight, reflecting robustness with respect to hyper-parameters. In

Table 4.3: Validation and test set perplexities on WikiText-2.

Model	Size	Depth	Valid	Test
LSTM, <a href="#">Grave et al. (2017)</a>	—	—	—	99.3
LSTM, <a href="#">Inan et al. (2017)</a>	22M	3	91.5	87.7
AWD-LSTM, <a href="#">Merity et al. (2018)</a>	33M	3	68.6	65.8
TG-SC LSTM, <a href="#">Melis et al. (2018)</a>	24M	2	69.1	65.9
aLSTM, recurrent module (Eq. 4.9)	27M	2	68.1	65.5
aLSTM, recurrent module (Eq. 4.9)	32M	2	<b>67.5</b>	<b>64.5</b>

Table 4.4: Ablation study: perplexities on Penn Treebank.<sup>†</sup>Equivalent to the HyperNetwork ([Ha et al., 2017](#)), except the aLSTM uses one projection from  $z$  to  $\omega$  instead of nesting two projections.

Model	Adaptation model	Adaptation module	Valid	Test
LSTM	—	—	71.7	68.9
aLSTM	feed-forward	output-adaptation	66.0	63.1
aLSTM <sup>†</sup>	LSTM	output-adaptation	59.9	58.2
aLSTM	LSTM-RHN	output-adaptation	59.7	57.3
aLSTM	feed-forward	IO-adaptation	61.6	59.1
aLSTM	LSTM	IO-adaptation	59.0	56.9
aLSTM	LSTM-RHN	IO-adaptation	<b>58.5</b>	<b>56.5</b>

fact, no sampled model fails to converge. In contrast, approximately 25% of the population of LSTM configurations fail to converge. In fact, fully 45% of the LSTM population fail to outperform the *worst* aLSTM configuration; the 90<sup>th</sup> percentile of the aLSTM distribution is on the same level as the 10<sup>th</sup> percentile of the LSTM distribution. On WT2 these results are amplified, with half of the LSTM population failing to converge and 80% of the LSTM population failing to outperform the worst-case aLSTM configuration.

## 4.6 Conclusions

By viewing deep neural networks as adaptive compositions of linear maps, we have showed that standard activation functions induce an activation function bottleneck because they fail to have significant non-linear effect on a non-trivial subset of inputs. We break this bottleneck through adaptive parameterization, which allows the model to adapt the affine transformation to the input.

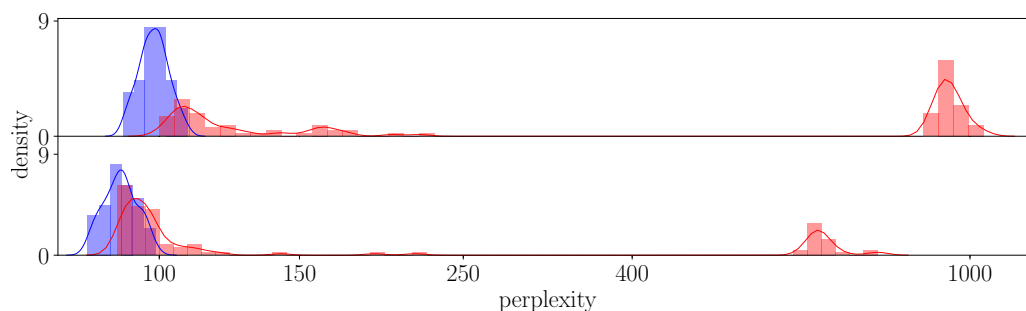


Figure 4.4: Distribution of validation scores on WikiText-2 (top) and Penn Treebank (bottom) for randomly sampled hyper-parameters. The aLSTM (blue) is more robust than the LSTM (red).

We have developed an adaptive feed-forward layer and showed empirically that it can learn patterns where a deep feed-forward network fails whilst also using fewer parameters. Extending the adaptive feed-forward layer to RNNs, we presented an adaptive LSTM that significantly increases model capacity and statistical efficiency while being more robust to hyper-parameters. In particular, we obtain new state of the art results on the Penn Treebank and the WikiText-2 word-modeling tasks, using  $\sim 20\text{--}30\%$  fewer parameters and converging in less than half as many iterations.

## 4

## Appendix

## 4.A NLP Experiment Hyper-Parameters

We base our experiments on the publicly available code-base of the AWD-LSTM (Merity et al., 2018). In particular, for our aLSTM, we use tied embedding weights (Press & Wolf, 2017; Inan et al., 2017) and a weight decay rate of  $10^{-6}$ . The initial learning rate is set to 0.003 with decay rates  $\beta_1 = 0$  and  $\beta_2 = 0.999$ . The learning rate is cut after epochs 100 and 160 by a factor of 10. We use a batch size of 20 and variable truncated backpropagation through time centered at 70, as in Merity et al. (2018). Dropout rates were tuned with a coarse random search. We apply variational dropout (Gal & Ghahramani, 2016) to word and word embedding vectors, the policy latent variable, the hidden state of the aLSTM and the final aLSTM output with drop probabilities 0.16, 0.6, 0.1, 0.25, and 0.6 respectively.

## 5 Scaling up Meta-Learning On First Principles

---

The previous chapter paved the way for meta-learning components of an architecture. This chapter takes a step towards scalable meta-learning. This work introduces Riemannian geometry to gradient-based meta-learning and shows that a distribution of tasks is homeomorphic to a distribution over manifolds and, ultimately, a distribution over learning trajectories.

A critical advantage of this perspective is that it provides tools from Riemannian geometry that allow us to reason about task adaptation from a geometrical perspective. It turns out that this can be used to entirely circumvent the issue of backpropagating through the adaptation process, which is currently a main limitation holding gradient-based meta-learning back from scaling beyond few-shot adaptation.

Most gradient-based meta-learners to date skirt this issue by taking fewer adaptation steps during meta-training than what is required at meta-test time. This introduces a short-horizon bias (Wu et al., 2018) that causes the meta-learner to optimise for the wrong time-horizon. This work shows that such biases lead to a catastrophic loss of performance when the number of meta-test time task adaptation steps are an order of magnitude larger than the number of task adaptation steps taken at meta-training time.

One solution to this issue is a meta-learner that reasons about *the process* of adaptation, as opposed to the final performance. Riemannian geometry provides a principled framework for developing such meta-learners. This work introduces a framework that relies on a different philosophy from previous gradient-based meta-learner, which superimpose a computational budget of  $K$  steps. The perspective taken here is that any task in a given task distribution can be solved if we can train sufficiently long on it. Hence the quantity to minimise during meta-training is not that final performance, but how long



it takes to reach convergence. To make this notion concrete, the framework relies on differential geometry to define “length” in terms of the distance travelled on a loss surface.

A key result of this work is to rigorously show that task distributions induce a distribution over gradient trajectories. This provides a powerful and flexible perspective for gradient-based meta-learning. On the one hand, it can be used to disentangle task adaptation and meta-gradients, so that we no longer need to backpropagate through task optimisation problems. On the other hand, it provides a more general understanding of “task”, so that meta-learning can be applied to almost any learning problem.

## Transferring Knowledge across Learning Processes

Peer-reviewed publication (Flennerhag et al., 2019); notational modifications.

Published in *International Conference on Learning Representations (Oral)*. 2019.

Authors: **Sebastian Flennerhag**, University of Manchester,  
Pablo G. Moreno, Amazon,  
Neil D. Lawrence, Amazon,  
Andreas Damianou, Amazon.

*Abstract. In complex transfer learning scenarios new tasks might not be tightly linked to previous tasks. Approaches that transfer information contained only in the final parameters of a source model will therefore struggle. Instead, transfer learning at a higher level of abstraction is needed. We propose Leap, a method that achieves this by transferring knowledge across learning processes. We associate each task with a manifold on which the training process travels from initialization to final parameters and construct a meta-learning objective that minimizes the expected length of this path. Leap leverages only information obtained during training and can be computed on the fly at negligible cost. We demonstrate that Leap outperforms competing methods, both in meta-learning and transfer learning, on a set of computer vision tasks. Finally, we demonstrate that Leap can transfer knowledge across learning processes in demanding reinforcement learning environments (Atari) that involve millions of gradient steps.*

---

### 5.1 Introduction

Transfer learning is the process of transferring knowledge encoded in one model trained on one set of tasks to another model that is applied to a new task. Since a trained model encodes information in its learned parameters, transfer learning typically transfers knowledge by encouraging the target model’s parameters to resemble those of a previous (set of) model(s) (Pan & Yang, 2009). This approach limits transfer learning to settings where good parameters for a new task can be found in the neighborhood of parameters that were learned from a previous task. For this to be a viable assumption, the two tasks must have a high degree of structural affinity, such as when a new task can be learned by extracting features from a pretrained model (Girshick et al., 2014; He et al., 2017; Mahajan et al., 2018). If not, this approach has been observed to limit knowledge transfer since the training process on one task will discard information that was irrelevant for the task at hand, but that would be relevant for another task (Higgins et al., 2017; Achille et al., 2018).

We argue that such information can be harnessed, even when the downstream

task is unknown, by transferring knowledge of the learning process itself. In particular, we propose a meta-learner for aggregating information across task geometries as they are observed during training. These geometries, formalized as the loss surface, encode all information seen during training and thus avoid catastrophic information loss. Moreover, by transferring knowledge across learning processes, information from previous tasks is distilled to explicitly facilitate the learning of new tasks.

Meta learning frames the learning of a new task as a learning problem itself, typically in the few-shot learning paradigm (Lake et al., 2011; Santoro et al., 2016; Vinyals et al., 2016). In this environment, learning is a problem of rapid adaptation and can be solved by training a meta-learner by backpropagating through the entire training process (Ravi & Larochelle, 2017; Andrychowicz et al., 2016; Finn et al., 2017). For more demanding tasks, meta-learning in this manner is challenging; backpropagating through thousands of gradient steps is both impractical and susceptible to instability. On the other hand, truncating backpropagation to a few initial steps induces a short-horizon bias (Wu et al., 2018). We argue that as the training process grows longer in terms of the distance traversed on the loss landscape, the geometry of this landscape grows increasingly important. When adapting to a new task through a single or a handful of gradient steps, the geometry can largely be ignored. In contrast, with more gradient steps, it is the dominant feature of the training process.

To scale meta-learning beyond few-shot learning, we propose *Leap*, a light-weight method for meta-learning over task manifolds that does not need any forward- or backward-passes beyond those already performed by the underlying training process. We find empirically that *Leap* is superior to similar methods when learning a task requires more than a handful of training steps. Finally, we evaluate *Leap* in a RL environment (Atari 2600; Bellemare et al., 2013), demonstrating that it can transfer knowledge across learning processes that require millions of gradient steps to converge.

## 5.2 Transferring Knowledge across Learning Processes

We start by introducing gradient descent from a geometric perspective in Section 5.2.1. We develop a method for transfer learning in Section 5.2.2 and explain how we can transfer knowledge across learning processes through geometrical quantities. Section 5.2.3 presents the novel meta-learner *Leap*.

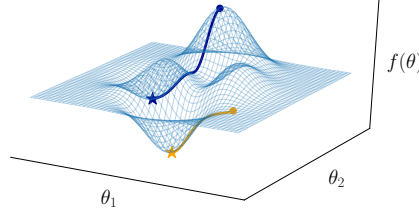


Figure 5.1: Gradient paths on a loss surface. Leap learns an initialization with shorter expected gradient path that improves performance.

### 5.2.1 Gradient Paths on Task Manifolds

Central to our method is the notion of a learning process; the harder a task is to learn, the harder it is for the learning process to navigate on the loss surface (Figure 5.1). Our method is based on the idea that transfer learning can be achieved by leveraging information contained in similar learning processes. Exploiting that this information is encoded in the geometry of the loss surface, we leverage geometrical quantities to facilitate the learning process with respect to new tasks. We focus on the supervised learning setting for simplicity, though our method applies more generally. Given a learning objective  $\mathcal{L}$  that consumes an input  $x$  and a target  $y$  and maps a parameterization  $\theta$  to a scalar loss value,  $\mathcal{L}(\theta) := \mathbb{E}_{(x,y) \sim p(x,y)} [\ell(f_\theta(x), y)]$  for some model  $f$  and loss  $\ell$ , we define gradient descent by

$$\theta := \theta - \alpha \nabla \mathcal{L}(\theta). \quad (5.1)$$

We take the learning rate  $\alpha$  as given, but our method can be extended to learn optimiser hyper-parameters jointly with the initialization. Our method is well defined under any gradient-based update rule; we focus on (stochastic) gradient descent for simplicity. Regardless of optimiser, we assume the process in Eq. 5.1 converges to a stationary point after  $K \in \mathbb{N}$  gradient steps.

To distinguish different learning processes originating from the same initialization, we need a notion of their length. The longer the process, the worse the initialization is (conditional on reaching equivalent performance, discussed further below). Measuring the Euclidean distance between initialization and final parameters is misleading as it ignores the actual path taken. This becomes crucial when we compare paths from different tasks, as gradient paths from different tasks can originate from the same initialization but take very different paths. Therefore, to capture the length of a learning process we must associate it with the loss surface it traversed.

The process of learning a task can be seen as a curve on a specific task manifold  $\mathcal{M}$ . While this manifold can be constructed in a variety of ways, here we exploit that, by definition, any learning process traverses the loss surface of  $\mathcal{L}$ . To accurately describe the length of a gradient-based learning process, it is sufficient to define the task manifold as the loss surface. In particular, because the learning process in Eq. 5.1 follows the gradient trajectory, it constantly provides information about the geometry of the loss surface. Gradients that largely point in the same direction indicate a well-behaved loss surface, whereas gradients with frequently opposing directions indicate an ill-conditioned loss surface—something we would like to avoid. Leveraging this insight, we propose a method for transfer learning that exploits the accumulation of geometric information by constructing a meta objective that minimizes the expected length of the gradient descent path *across tasks*. In doing so, the meta objective intrinsically balances local geometries across tasks and encourages an initialization that makes the learning process as short as possible.

To formalize the notion of the distance of a learning process, we define a task manifold  $\mathcal{M}$  as a submanifold of  $\mathbb{R}^{n+1}$  given by the graph of  $\mathcal{L}$ . Every point  $\gamma := [\theta; \mathcal{L}(\theta)] \in \mathcal{M}$  is locally homeomorphic to a Euclidean subspace, described by the tangent space  $T_\gamma \mathcal{M}$ . Taking  $\mathbb{R}^{n+1}$  to be Euclidean, it is a Riemann manifold. By virtue of being a submanifold of  $\mathbb{R}^{n+1}$ ,  $\mathcal{M}$  is also a Riemann manifold. As such,  $\mathcal{M}$  comes equipped with an smoothly varying inner product  $g_\gamma : T_\gamma \mathcal{M} \times T_\gamma \mathcal{M} \rightarrow \mathbb{R}$  on tangent spaces, allowing us to measure the length of a path on  $\mathcal{M}$ . In particular, the length (or energy) of any curve  $\gamma : [0, 1] \rightarrow \mathcal{M}$  is defined by accumulating infinitesimal changes along the trajectory,

$$\text{Length}(\gamma) := \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))} dt, \quad (5.2)$$

$$\text{Energy}(\gamma) := \int_0^1 g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t)) dt, \quad (5.3)$$

where  $\dot{\gamma}(t) := \frac{d}{dt}\gamma(t) \in T_{\gamma(t)} \mathcal{M}$  is a tangent vector of  $\gamma(t) := [\theta(t); \mathcal{L}(\theta(t))] \in \mathcal{M}$ . We use parentheses (i.e.  $\gamma(t)$ ) to differentiate discrete and continuous domains. With  $\mathcal{M}$  being a submanifold of  $\mathbb{R}^{n+1}$ , the induced metric on  $\mathcal{M}$  is defined by  $g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t)) = \langle \dot{\gamma}(t), \dot{\gamma}(t) \rangle$ . Different constructions of  $\mathcal{M}$  yield different Riemann metrics. In particular, if the model  $f_\theta$  admits a predictive probability distribution  $p_\theta(y | x)$ , the task manifold can be given an information geometric interpretation by choosing the Fisher matrix as Riemann metric, in which case

the task manifold is defined over the space of probability distributions (Amari & Nagaoka, 2007). If Eq. 5.1 is defined as natural gradient descent, the learning process corresponds to gradient descent on this manifold (Amari, 1998; Martens, 2010; Pascanu & Bengio, 2014; Luk & Grosse, 2018).

Having a complete description of a task manifold, we can measure the length of a learning process by noting that gradient descent can be seen as a discrete approximation to the gradient flow  $\dot{\theta}(t) = -\nabla \mathcal{L}(\theta(t))$ . This flow describes a curve that originates in  $\gamma(0) = [\theta_0; \mathcal{L}(\theta_0)]$  and follows the gradient at each point. We define  $\gamma$  to be this unique curve and refer to it as the *gradient path* from  $\theta_0$  on  $\mathcal{M}$ . The metrics in Eqs. 5.2 and 5.3 can be computed exactly, but in practice we observe a discrete learning process. Analogously to how gradient descent approximates the gradient flow, the gradient path length or energy can be approximated by the cumulative chordal distance (Ahlberg et al., 1967),

$$d_p(\theta_0, \mathcal{M}) := \sum_{k=0}^{K-1} \|\gamma_{k+1} - \gamma_k\|_2^p, \quad p \in \{1, 2\}. \quad (5.4)$$

We write  $d$  when the distinction between length and energy is immaterial. Using the energy (Eq. 5.3) yields a slightly simpler objective, but the length (Eq. 5.2) normalizes each length segment and as such protects against differences in scale between task objectives. In Appendix 5.C, we conduct an ablation study and find that they perform similarly, though using the length leads to faster convergence. Importantly,  $d$  involves only terms seen during task training. We exploit this later when we construct the meta gradient, enabling us to perform gradient descent on the meta objective at negligible cost (Eq. 5.9).

We now turn to meta-learning, where we face a set of tasks, each with a distinct task manifold. Our method builds on the idea that we can transfer knowledge across learning processes via the local geometry by aggregating information obtained along observed gradient paths. In essence, Leap finds an initialization from which learning trajectories have minimal expected length.

### 5.2.2 Meta Learning across Task Manifolds

We define a task  $\tau := (\mathcal{L}^\tau, p^\tau, u^\tau)$  as the process of learning to approximate the relationship  $x \mapsto y$  through samples from the data distribution  $p^\tau(x, y)$ . This process is defined by a gradient update rule  $u^\tau$  (e.g. Eq. 5.1), applied  $K$  times to minimize the task objective  $\mathcal{L}^\tau$ . A learning process starts at  $\theta_0^\tau = \theta_0$  and progresses via  $\theta_{k+1}^\tau = u^\tau(\theta_k^\tau)$ . The sequence  $\{\theta_k^\tau\}_{k=0}^K$  defines an approximate gradient path on the task manifold  $\mathcal{M}^\tau$  with distance  $d(\theta_0; \mathcal{M}^\tau)$ .

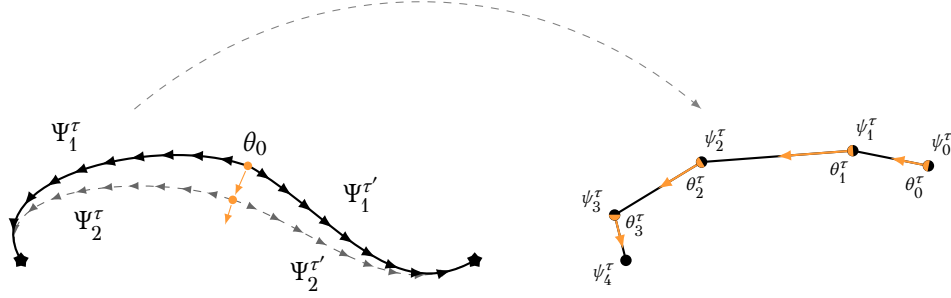


Figure 5.2: *Left*: illustration of Leap (Algorithm 5.1) for two tasks,  $\tau$  and  $\tau'$ . From an initialization  $\theta_0$ , the learning process of each task generates gradient paths,  $\Psi^\tau$  and  $\Psi^{\tau'}$ , which Leap uses to minimize the expected path length. Iterating the process, Leap converges to a locally Pareto optimal initialization. *Right*: the pull-forward objective (Eq. 5.7) minimizes the expected gradient path length by pulling each  $\theta_k^\tau$  towards  $\psi_{k+1}^\tau$ .

To understand how  $d$  transfers knowledge across learning processes, consider two distinct tasks. We can transfer knowledge across these tasks' learning processes by measuring how good a shared initialization is. Assuming two candidate initialisations converge to limit points with equivalent performance on each task, the initialization with shortest expected gradient path distance encodes more knowledge sharing. In particular, if both tasks have convex loss surfaces a unique optimal initialization exists that achieves Pareto optimality in terms of total path distance. This can be crucial in data sparse regimes where rapid convergence can avoid overfitting (Finn et al., 2017).

Given a distribution of tasks  $p(\tau)$ , each candidate initialization  $\theta_0$  is associated with a measure of its expected gradient path distance,  $\mathbb{E}_{\tau \sim p(\tau)}[d(\theta_0; \mathcal{M}^\tau)]$ , that summarizes the suitability of the initialization to the task distribution. The initialization (or a set thereof) with shortest expected gradient path distance maximally transfers knowledge across learning processes and is Pareto optimal in this regard. Above, we have assumed that all candidate initializations converge to limit points of equal performance. If the task objective  $\mathcal{L}^\tau$  is non-convex this is not a trivial assumption and the gradient path distance itself does not differentiate between different levels of final performance.

As such, it is necessary to introduce a feasibility constraint to ensure only initializations with some minimum level of performance are considered. We leverage that transfer learning never happens in a vacuum; we always have a second-best option, such as starting from a random initialization or a pretrained model. This “second-best” initialization,  $\psi_0$ , provides us with the performance we would obtain on a given task in the absence of knowledge transfer. As such, performance obtained by initializing from  $\psi_0$  provides us with an upper

bound for each task: a candidate solution  $\theta_0$  must achieve at least as good performance to be a viable solution. Formally, this implies the task-specific requirement that a candidate  $\theta_0$  must satisfy  $\mathcal{L}^\tau(\theta_K^\tau) \leq \mathcal{L}^\tau(\psi_K^\tau)$ . As this must hold for every task, we obtain the canonical meta objective

$$\begin{aligned} \min_{\theta_0} \quad & \mathcal{J}(\theta_0) := \mathbb{E}_{\tau \sim p(\tau)} [d(\theta_0; \mathcal{M}^\tau)] \\ \text{s.t.} \quad & \theta_{k+1}^\tau = u^\tau(\theta_k^\tau), \quad \theta_0^\tau = \theta_0, \\ & \theta_0 \in \Theta := \cap_{\tau} \{ \theta_0 \mid \mathcal{L}^\tau(\theta_K^\tau) \leq \mathcal{L}^\tau(\psi_K^\tau) \}. \end{aligned} \tag{5.5}$$

This meta objective is robust to variations in loss surfaces, as it balances complementary and competing learning processes (Figure 5.2). For instance, there may be an initialization that can solve a small subset of tasks in a handful of gradient steps, but would be catastrophic for other related tasks. When transferring knowledge via the initialization, we must trade off commonalities and differences between gradient paths. In Eq. 5.5 these trade-offs arise naturally. As the number of tasks whose gradient paths move in the same direction increases, so does their pull on the initialization. Conversely, as the updates to the initialization renders some gradient paths longer, these act as springs that exert increasingly strong pressure on the initialization. The solution to Eq. 5.5 thus achieves an equilibrium between these competing forces.

Solving Eq. 5.5 naively requires training to convergence on each task to determine whether an initialization satisfies the feasibility constraint, which can be very costly. Because we have access to a second-best initialization, we can solve Eq. 5.5 more efficiently by obtaining gradient paths from  $\psi_0$  and use these as baselines that we incrementally improve upon. This improved initialization converges to the same limit points, but with shorter expected gradient paths (Theorem 5.1). As such, it becomes the new second-best option; Leap (Algorithm 5.1) repeats this process of improving upon increasingly demanding baselines, ultimately finding a solution to the canonical meta objective.

### 5.2.3 Leap

Leap starts from a given second-best initialization  $\psi_0$ , shared across all tasks, and constructs baseline gradient paths  $\Psi^\tau := \{\psi_k^\tau\}_{k=0}^K$  for each task  $\tau$  in a batch  $\mathcal{B}$ . These provide a set of baselines  $\Psi := \{\Psi^\tau\}_{\tau \in \mathcal{B}}$ . Recall that all tasks share the same initialization,  $\psi_0^\tau = \psi_0 \in \Theta$ . We use these baselines, corresponding to task-specific learning processes, to modify the gradient path distance metric



**Algorithm 5.1** Leap: Transferring Knowledge over Learning Processes**Require:**  $p(\tau)$ ,  $\tau := (\mathcal{L}^\tau, u^\tau, p^\tau)$ : distribution over tasks**Require:**  $\beta$ : step size

---

```

1: randomly initialize  $\theta_0$ 
2: while not done do
3:    $\nabla \tilde{\mathcal{J}} \leftarrow 0$ : initialize meta gradient
4:   sample task batch  $\mathcal{B}$  from  $p(\tau)$ 
5:   for all  $\tau \in \mathcal{B}$  do
6:      $\psi_0^\tau \leftarrow \theta_0$ : initialize task baseline
7:     for all  $i \in \{0, \dots, K-1\}$  do
8:        $\psi_{k+1}^\tau \leftarrow u^\tau(\psi_k^\tau)$ : update baseline
9:        $\theta_k^\tau \leftarrow \psi_k^\tau$ : follow baseline (recall  $\psi_0^\tau = \theta_0$ )
10:      increment  $\nabla \tilde{\mathcal{J}}$  using the pull-forward gradient (Eq. 5.9)
11:    end for
12:  end for
13:   $\theta_0 \leftarrow \theta_0 - \frac{\beta}{|\mathcal{B}|} \nabla \tilde{\mathcal{J}}$ : update initialization
14: end while

```

---

in Eq. 5.4 by freezing the forward point  $\gamma_{k+1}^\tau$  in all norms,

$$\tilde{d}_p(\theta_0; \mathcal{M}^\tau, \Psi^\tau) := \sum_{k=0}^{K-1} \|\tilde{\gamma}_{k+1}^\tau - \gamma_k^\tau\|_2^p, \quad (5.6)$$

where  $\tilde{\gamma}_k^\tau := [\psi_k^\tau; \mathcal{L}(\psi_k^\tau)]$  represents the frozen forward point from the baseline and  $\gamma_k^\tau := [\theta_k^\tau; \mathcal{L}(\theta_k^\tau)]$  the point on the gradient path originating from  $\theta_0$ . This surrogate distance metric encodes the feasibility constraint; optimizing  $\theta_0$  with respect to  $\Psi$  pulls the initialization forward along each task-specific gradient path in an unconstrained variant of Eq. 5.5 that replaces  $\Theta$  with  $\Psi$ ,

$$\begin{aligned} \min_{\theta_0} \quad & \tilde{\mathcal{J}}(\theta_0; \Psi) := \mathbb{E}_{\tau \sim p(\tau)} [\tilde{d}(\theta_0; \mathcal{M}^\tau, \Psi^\tau)], \\ \text{s.t.} \quad & \theta_{k+1}^\tau = u^\tau(\theta_k^\tau), \quad \theta_0^\tau = \theta_0. \end{aligned} \quad (5.7)$$

We refer to Eq. 5.7 as the *pull-forward* objective. Incrementally improving  $\theta_0$  over  $\psi_0$  leads to a new second-best option that Leap uses to generate a new set of more demanding baselines, to further improve the initialization. Iterating this process, Leap produces a sequence of candidate solutions to Eq. 5.5, all in  $\Theta$ , with incrementally shorter gradient paths. While the pull-forward objective can be solved with any optimization algorithm, we consider gradient-based methods. In Theorem 5.1, we show that gradient descent on  $\tilde{\mathcal{J}}$  yields solutions that always lie in  $\Theta$ . In principle,  $\tilde{\mathcal{J}}$  can be evaluated at any  $\theta_0$ , but a more

efficient strategy is to evaluate  $\theta_0$  at  $\psi_0$ . In this case,  $\tilde{d} = d$ , so that  $\tilde{\mathcal{J}} = \mathcal{J}$ .

**Theorem 5.1** (Pull-forward). *Define a sequence of initializations  $\{\theta_0^s\}_{s \in \mathbb{N}}$  by*

$$\theta_0^{s+1} := \theta_0^s - \beta^s \nabla \tilde{\mathcal{J}}(\theta_0^s; \Psi^s), \quad (5.8)$$

with  $\theta_0^0 \in \Theta$  and  $\psi_0^s = \theta_0^s$  for all  $s$ . For  $\beta^s > 0$  sufficiently small, there exist learning rates schedules  $\{\alpha_k^\tau\}_{k=1}^K$  for all tasks such that  $\theta_0^{k \rightarrow \infty}$  is a limit point in  $\Theta$ .

**Proof:** Appendix 5.A. Because the meta gradient requires differentiating the learning process, we adopt an approximation to obtain a meta-gradient that can be computed analytically on the fly during training. Let  $\tilde{\mathcal{J}}_p$  denote the distance under either the length or the energy. Differentiating  $\tilde{\mathcal{J}}_p$ , we have

$$\nabla \tilde{\mathcal{J}}_p(\theta_0, \Psi) := -p \mathbb{E}_{\tau \sim p(\tau)} \left[ \sum_{k=0}^{K-1} \frac{[D_{\theta_0} \theta_k^\tau]^T \left( \Delta \mathcal{L}_k^\tau \nabla \mathcal{L}^\tau(\theta_k^\tau) + \Delta \theta_k^\tau \right)}{\left( \|\tilde{y}_{k+1}^\tau - y_k^\tau\|_2^p \right)^{2-p}} \right], \quad (5.9)$$

where  $D_{\theta_0} \theta_k^\tau$  denotes the Jacobian of  $\theta_k^\tau$  with respect to the initialisation; we define the loss delta  $\Delta \mathcal{L}_k^\tau := \mathcal{L}^\tau(\psi_{k+1}^\tau) - \mathcal{L}^\tau(\theta_k^\tau)$  and the parameter delta  $\Delta \theta_k^\tau := \psi_{k+1}^\tau - \theta_k^\tau$ . To render the meta gradient tractable, we need to approximate the Jacobians, as these are costly to compute. Empirical evidence suggest that they are largely redundant (Finn et al., 2017; Nichol et al., 2018). Nichol et al. (2018) further shows that an identity approximation yields a meta-gradient that remains faithful to the original meta objective. We provide some further support for this approximation (see Appendix 5.B). First, we note that the learning rate directly controls the quality of the approximation; for any  $K$ , the identity approximation can be made arbitrarily accurate by choosing a sufficiently small learning rates. We conduct an ablation study to ascertain how severe this limitation is and find that it is relatively loose. For the best-performing learning rate, the identity approximation is accurate to four decimal places and shows no signs of significant deterioration as the number of training steps increases. As such, we assume  $D_{\theta_0} \theta_k^\tau \approx I_n$  for all  $k$  throughout. Finally, by evaluating  $\nabla \tilde{\mathcal{J}}$  at  $\theta_0 = \psi_0$ , the meta gradient contains only terms seen during standard training and can be computed asynchronously on the fly at negligible cost.

In practice, we use stochastic gradient descent during task training. This injects noise in  $\mathcal{L}$  as well as in its gradient, resulting in a noisy gradient path. Noise in the gradient path does not prevent Leap from converging. However, noise reduces the rate of convergence, in particular when a noisy gradient step results in  $\mathcal{L}^\tau(\psi_{s+1}^\tau) - \mathcal{L}^\tau(\theta_k^\tau) > 0$ . If the gradient estimator is reasonably accurate, this causes the term  $\Delta \mathcal{L}_k^\tau \nabla \mathcal{L}^\tau(\theta_k^\tau)$  in Eq. 5.9 to point in the steepest ascent direction. We found that adding a stabilizer to ensure we always follow the descent direction significantly speeds up convergence and allows us to use larger learning rates. In this paper, we augment  $\tilde{\mathcal{J}}$  with a stabilizer

$$\mu(\mathcal{L}^\tau(\theta_k^\tau); \mathcal{L}^\tau(\psi_{k+1}^\tau)) := \begin{cases} 0 & \text{if } \Delta \mathcal{L}_k^\tau < 0, \\ -2 [\Delta \mathcal{L}_k^\tau]^2 & \text{else.} \end{cases}$$

We add  $\nabla \mu$  (re-scaled if  $p = 2$ ) to the meta-gradient, which is equivalent to replacing  $\Delta \mathcal{L}_k^\tau$  with  $-|\Delta \mathcal{L}_k^\tau|$  in Eq. 5.9. This ensures that we never follow  $\nabla \mathcal{L}^\tau(\theta_k^\tau)$  in the ascent direction, instead reinforcing the descent direction at that point. This stabilizer is a heuristic, others may prove helpful. In Appendix 5.C we perform an ablation study and find that the stabilizer is not necessary for Leap to converge, but it does speed up convergence significantly.

### 5.3 Related Work

Transfer learning has been explored in a variety of settings, typically by infusing knowledge in a target model’s parameters by encouraging them to lie close to those of a pretrained source model (Pan & Yang, 2009). Because such approaches can limit knowledge transfer (Higgins et al., 2017; Achille et al., 2018), applying standard transfer learning techniques leads to *catastrophic forgetting*, by which the model is rendered unable to perform a previously mastered task (McCloskey & Cohen, 1989; Goodfellow et al., 2013). These problems are further accentuated when there is a larger degree of diversity among tasks that push optimal parameterizations further apart. In these cases, transfer learning can in fact be worse than training from scratch.

Recent approaches extend standard finetuning by adding regularizing terms to the training objective. These encourage the model to learn parameters that both solve a new task and retain high performance on previous tasks and operate by protecting the parameters that affect the loss function the most (Miconi et al., 2018; Zenke et al., 2017; Kirkpatrick et al., 2017; Lee et al., 2017; Serrá et al., 2018). Because these approaches use a single model to encode both

global task-general information and local task-specific information, they can over-regularize, preventing the model from learning further tasks. [Schwarz et al. \(2018\)](#) found that while these approaches mitigate catastrophic forgetting, they are unable to facilitate knowledge transfer on the benchmark they considered. Ultimately, if a single model must encode both task-generic and task-specific information, it must either saturate or grow in size ([Rusu et al., 2016](#)).

In contrast, meta-learning aims to learn the learning process itself ([Schmidhuber, 1987](#); [Bengio et al., 1991](#); [Santoro et al., 2016](#); [Ravi & Larochelle, 2017](#); [Andrychowicz et al., 2016](#); [Vinyals et al., 2016](#); [Finn et al., 2017](#)). The literature focuses primarily on few-shot learning, where a task is some variation on a common theme, such as subsets of classes drawn from a shared pool of data ([Lake et al., 2015](#); [Vinyals et al., 2016](#)). The meta-learning algorithm adapts a model to a new task given a handful of samples. Recent attention has been devoted to three main approaches. One trains the meta-learner to adapt to a new task by comparing an input to samples from previous tasks ([Vinyals et al., 2016](#); [Mishra et al., 2018](#); [Snell et al., 2017](#)). More relevant to our framework are approaches that parameterize the training process through a recurrent neural network that takes the gradient as input and produces a new set of parameters ([Ravi & Larochelle, 2017](#); [Santoro et al., 2016](#); [Andrychowicz et al., 2016](#); [Hochreiter et al., 2001](#)). The approach most closely related to us learns an initialization such that the model can adapt to a new task through one or a few gradient updates ([Finn et al., 2017](#); [Nichol et al., 2018](#); [Al-Shedivat et al., 2018](#); [Lee & Choi, 2018](#)). In contrast to our work, these methods focus exclusively on few-shot learning, where the gradient path is trivial as only a single or a handful of training steps are allowed, limiting them to settings where the current task is closely related to previous ones.

It is worth noting that the Model Agnostic Meta Learner (MAML: [Finn et al., 2017](#)) can be written as  $\mathbb{E}_{\tau \sim p(\tau)} [\mathcal{L}^\tau(\theta_K^\tau)]$ .<sup>8</sup> The meta-gradient of this objective takes the form  $\mathbb{E}_{\tau \sim p(\tau)} \left[ \left[ D_{\theta_0} \theta_K^\tau \right]^T \nabla \mathcal{L}^\tau(\theta_K^\tau) \right]$ . As such, it arises as a special case of Leap where only the final parameterization is evaluated in terms of its final performance. Similarly, the Reptile algorithm ([Nichol et al., 2018](#)), which proposes to update rule  $\theta^0 \leftarrow \theta^0 + \epsilon (\mathbb{E}_{\tau \sim p(\tau)} [\theta_K^\tau] - \theta^0)$ , can be seen as a naive version of Leap that assumes all task geometries are Euclidean. In particular, Leap reduces to Reptile if  $\mathcal{L}_\tau$  is removed from the task manifold and the energy metric without stabilizer is used. We find this configuration to perform significantly worse than any other (see Section 5.4.1 and Appendix 5.C).

<sup>8</sup>MAML differs from Leap in that it evaluates the meta objective on a held-out test set.

Related work studying models from a geometric perspective have explored how to interpolate in a generative model’s learned latent space (Tosi et al., 2014; Shao et al., 2018; Arvanitidis et al., 2018; Chen et al., 2018; Kumar et al., 2017). Riemann manifolds have also garnered attention in the context of optimization where preconditioning can be understood as the instantiation of some Riemann metric (Amari & Nagaoka, 2007; Abbati et al., 2018; Luk & Grosse, 2018).

## 5.4 Empirical Results

We consider three experiments with increasingly complex knowledge transfer. We measure transfer learning in terms of final performance and speed of convergence, where the latter is defined as the area under the training error curve. We compare Leap to competing meta-learning methods on the Omniglot dataset by transferring knowledge across alphabets (Section 5.4.1). We study Leap’s ability to transfer knowledge over more complex and diverse tasks in a Multi-CV experiment (Section 5.4.2) and finally evaluate Leap on in a demanding reinforcement environment (Section 5.4.3).

### 5.4.1 Omniglot

The Omniglot (Lake et al., 2015) dataset consists of 50 alphabets, which we define to be distinct tasks. We hold 10 alphabets out for final evaluation and use subsets of the remaining alphabets for meta-learning or pretraining. We vary the number of alphabets used for meta-learning / pretraining from 1 to 25 and compare final performance and rate of convergence on held-out tasks. We compare against no pretraining, multi-headed finetuning, MAML, the first-order approximation of MAML (FOMAML; Finn et al., 2017), and Reptile. We train on a given task for 100 steps, with the exception of MAML where we backpropagate through 5 training steps during meta-training. For Leap, we report performance under the length metric ( $d_1$ ); see Appendix 5.C for an ablation study on Leap parameters. For further details, see Appendix 5.D.

All baselines significantly improves upon a random initialization. MAML exhibits a considerable short-horizon bias (Wu et al., 2018). While FOMAML is trained full trajectories, but because it only leverages gradient information at final iteration, which may be arbitrarily uninformative, it does worse. Multi-headed finetuning is a tough benchmark to beat as tasks are very similar. Nevertheless, for sufficiently rich task distributions, both Reptile and Leap outperform finetuning, with Leap outperforming Reptile as the complexity grows. That Leap and Reptile achieve similar performance underscores that Reptile is a special case under a simplified geometry. That the AUC gap between

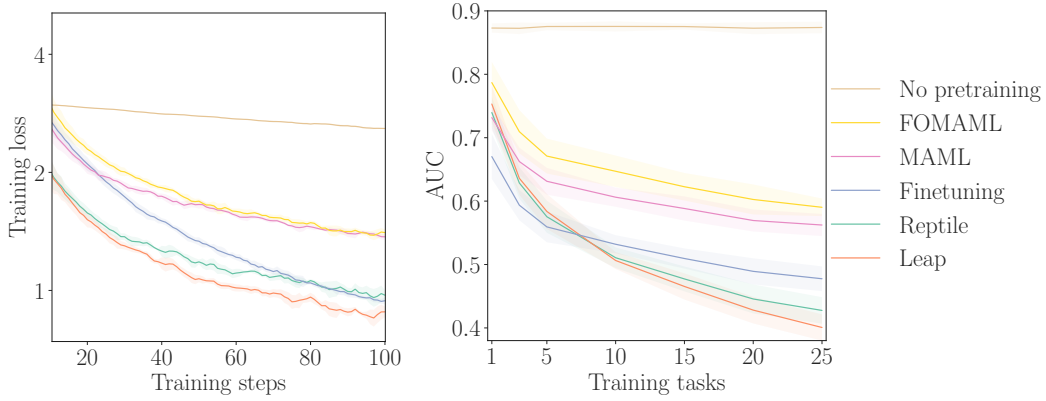


Figure 5.3: Results on Omniglot. *Left*: Comparison of average learning curves on held-out tasks (across 10 seeds) for 25 tasks in the meta-training set. Curves are moving averages with window size 5. Shading: standard deviation within window. *Right*: AUC across number of tasks in the meta-training set. Shading: standard deviation across 10 seeds.

Reptile and Leap grows in the number of meta-training tasks (i.e. meta-learning complexity; see Figure 5.3)—reaching a 4 percentage point difference in final validation error for 25 meta-training tasks (Table 5.2)—supports our hypothesis that accounting for learning dynamics via geometric information can facilitate meta-learning as the meta-learning problem grows increasingly complex.

#### 5.4.2 Multi-CV

Inspired by Serrá et al. (2018), we consider a set of computer vision datasets as distinct tasks. We pretrain on all but one task, which is held out for final evaluation. For details, see Appendix 5.E. To reduce the computational burden during meta training, we pretrain on each task in the meta batch for one epoch using the energy metric ( $d_2$ ). We found this to reach equivalent performance to training on longer gradient paths or using the length metric. This indicates that it is sufficient for Leap to see a partial trajectory to correctly infer shared structures across task geometries.

We compare Leap against a random initialization, multi-headed finetuning, a non-sequential version of HAT (Serrá et al., 2018) (i.e. allowing revisits) and a non-sequential version of Progressive Nets (Rusu et al., 2016), where we allow lateral connection between every task. Note that this makes Progressive Nets over 8 times larger in terms of learnable parameters (leading to a computational and memory overhead of approximately 8x).

Table 5.1: Results on Multi-CV benchmark. All methods are trained until convergence on held-out tasks. Finetuning is multiheaded. <sup>†</sup> Area under training error curve; scaled to 0–100. <sup>‡</sup> Our implementation. MNIST results omitted; see Appendix 5.E, Table 5.5.

Held-out task	Method	Test (%)	Train (%)	AUC <sup>†</sup>
Facescrub	Leap	19.9	0.0	11.6
	Finetuning	32.7	0.0	13.2
	Progressive Nets <sup>‡</sup>	<b>18.0</b>	0.0	<b>8.9</b>
	HAT <sup>‡</sup>	25.6	0.1	14.6
	No pretraining	18.2	0.0	10.5
Cifar10	Leap	<b>21.2</b>	<b>10.8</b>	<b>17.5</b>
	Finetuning	27.4	13.3	20.7
	Progressive Nets <sup>‡</sup>	24.2	15.2	24.0
	HAT <sup>‡</sup>	27.7	21.2	27.3
	No pretraining	26.2	13.1	23.0
SVHN	Leap	<b>8.4</b>	<b>5.6</b>	<b>7.5</b>
	Finetuning	10.9	6.1	10.5
	Progressive Nets <sup>‡</sup>	10.1	6.3	13.8
	HAT <sup>‡</sup>	10.5	5.7	8.5
	No pretraining	10.3	6.9	11.5
Cifar100	Leap	<b>52.0</b>	<b>30.5</b>	<b>43.4</b>
	Finetuning	59.2	31.5	44.1
	Progressive Nets <sup>‡</sup>	55.7	42.1	54.6
	HAT <sup>‡</sup>	62.0	49.8	58.4
	No pretraining	54.8	33.1	50.1
Traffic Signs	Leap	<b>2.9</b>	0.0	<b>1.2</b>
	Finetuning	5.7	0.0	1.7
	Progressive Nets <sup>‡</sup>	3.6	0.0	4.0
	HAT <sup>‡</sup>	5.4	0.0	2.3
	No pretraining	3.6	0.0	2.4

The Multi-CV experiment is more challenging both due to greater task diversity and greater complexity among tasks. We report results on held-out tasks in Table 5.1. Leap outperforms all baselines on all but one transfer learning tasks (Facescrub), where Progressive Nets does marginally better than a random initialization owing to its increased parameter count. Notably, while Leap does marginally worse than a random initialization, finetuning and HAT leads to a substantial drop in performance. On all other tasks, Leap converges faster to optimal performance and achieves superior final performance.



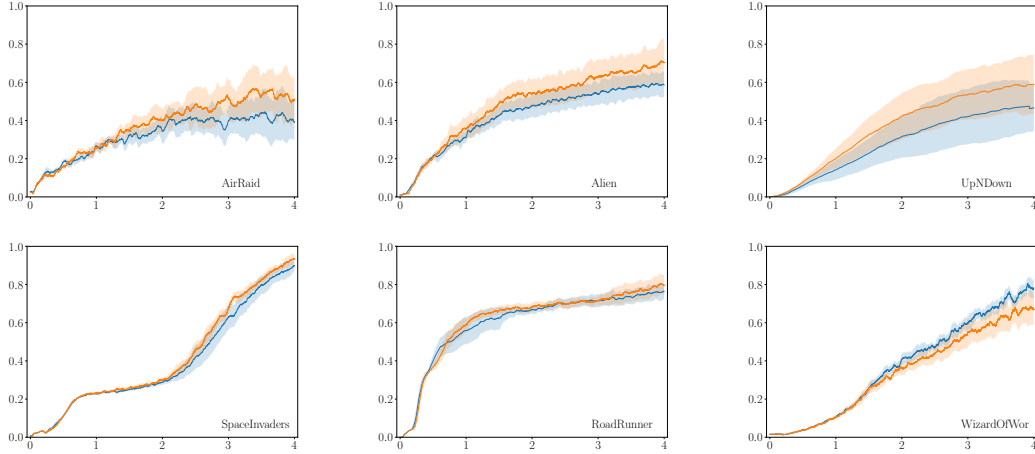


Figure 5.4: Mean normalized episode scores on Atari games across training steps. Shaded regions depict two standard deviations across ten seeds. Leap (orange) generally outperforms a random initialization (blue), even when the action space is twice as large as during pretraining (Table 5.6, Appendix 5.F).

### 5.4.3 Atari

To demonstrate that Leap can scale to large problems, both in computational terms and in task complexity, we apply it in a reinforcement learning environment, specifically Atari 2600 games (Bellemare et al., 2013). We use an actor-critic architecture (Sutton & Barto, 1998) with the policy and the value function sharing a convolutional encoder. We apply Leap to the encoder using the energy metric ( $d_2$ ). During meta training, we sample mini-batches from 27 games that have an action space dimensionality of at most 10, holding out two games with similar action space dimensionality for evaluation, as well as games with larger action spaces (Table 5.6). During meta-training, we train on each task for five million training steps. See Appendix 5.F for details.

Due to high computational cost, we are restricted to train for 100 meta training steps. This is sufficient to see a distinct improvement over the baseline, but with only 100 meta-training steps we do not generally obtain statistical significance. We expect a longer meta-training phase to yield further gains; our results should be interpreted as an initial indication of the general feasibility of meta-learning in large-scale environments.

We find that Leap generally outperforms a random initialization. This performance gain is primarily driven by less volatile exploration, as seen by the confidence intervals in Figure 5.4 (see also Figure 5.8). Leap finds a useful exploration space faster and more consistently, suggesting that it can find shared structures across a diverse set of complex learning processes.



These gains may not cater equally to all tasks. In the case of WizardOfWor (meta-training task), Leap exhibits two modes: in one it performs on par with the baseline, in the other exploration is protracted (Figure 5.8). This stems from noisy gradients, which renders an observed gradient path less representative. Such randomness can be marginalized by training for longer.

That Leap can outperform a random initialization on the pretraining set (AirRaid, UpNDown) is perhaps not surprising. More striking is that it exhibits the same behavior on out-of-distribution tasks. In particular, Alien, Gravitar and RoadRunner all have at least 50% larger state space than anything encountered during pretraining (Appendix 5.F, Table 5.6), yet Leap outperforms a random initialization. This suggests that transferring knowledge at a higher level of abstraction, such as in the space of gradient paths, generalizes to unseen task variations as long as underlying learning dynamics agree.

## 5.5 Conclusions

Transfer learning typically ignores the learning process itself, restricting knowledge transfer to scenarios where target tasks are very similar to source tasks. In this paper, we present Leap, a framework for knowledge transfer at a higher level of abstraction. By formalizing knowledge transfer as minimizing the expected length of gradient paths, we propose a method for meta-learning that scales to highly demanding problems. We find empirically that Leap can provide useful inductive biases that improve the rate of learning, as well as final performance, both compared to a random initialisation, finetuning, and similar meta-learners. We substantive support in the supervised learning setting and our results on the extremely challenging Atari suite provides indicative evidence that meta-learning is feasible in this regime. Delving deeper in this direction is an exciting area for future research.

## 5

## Appendix

## 5.A Mathematical Results

We now establish the proof of Theorem 5.1, re-stated below for convenience:

**Theorem 5.2.** *Define a sequence of initializations  $\{\theta_0^s\}_{s \in \mathbb{N}}$  by*

$$\theta_0^{s+1} := \theta_0^s - \beta^s \nabla \tilde{\mathcal{J}}(\theta_0^s; \Psi^s), \quad (5.10)$$

with  $\theta_0^0 \in \Theta$  and  $\psi_0^s = \theta_0^s$  for all  $s$ . For  $\beta^s > 0$  sufficiently small, there exist learning rates schedules  $\{\alpha_k^r\}_{k=1}^K$  for all tasks such that  $\theta_0^{k \rightarrow \infty}$  is a limit point in  $\Theta$ .

*Proof.* We proceed as follows. First, we establish that, for all  $s$ ,

$$\mathbb{E}_{\tau \sim p(\tau)} [d(\theta_0^{s+1}, \mathcal{M}^\tau)] = \mathcal{J}(\theta_0^{s+1}) \quad (5.11)$$

$$= \tilde{\mathcal{J}}(\theta_0^{s+1}; \Psi^{s+1}) \quad (5.12)$$

$$\leq \tilde{\mathcal{J}}(\theta_0^s; \Psi^s) \quad (5.13)$$

$$= \mathcal{J}(\theta_0^s) \quad (5.14)$$

$$= \mathbb{E}_{\tau \sim p(\tau)} [d(\theta_0^s, \mathcal{M}^\tau)], \quad (5.15)$$

with strict inequality for at least some  $s$ ; note that Eqs. 5.11 and 5.15 follow by definition (Eqs. 5.5 and 5.7), while Eqs. 5.12 and 5.14 hold by assumption  $\psi_0^s = \theta_0^s \forall s$  (in which case  $\tilde{d}$  and  $d$  coincide; see Eqs. 5.4 and 5.6).

Next, because  $\{\beta^s\}_{s=1}^\infty$  satisfies the gradient descent criteria, the sequence  $\{\theta_0^s\}_{s=1}^\infty$  is convergent. To complete the proof we show its limit point lies in  $\Theta$ : for  $\beta^s$  sufficiently small, for all  $s$ ,  $\lim_{k \rightarrow \infty} \theta_k^{s+1} = \lim_{k \rightarrow \infty} \theta_k^s$ . That is, each updated initialization incrementally reduces the expected gradient path length while converging to the same limit point as  $\theta_0^0$ . Since  $\theta_0^0 \in \Theta$  by assumption, we obtain  $\theta_0^s \in \Theta$  for all  $s$  as an immediate consequence.

To establish Eq. 5.13, with strict inequality for some  $s$ , we define need to keep track of two sequences, the inner loop iterates in  $k$  while the outer loop is indexed by  $s$ . To simplify notation, we use  $z$  and  $h$  to map into the point at on task  $\tau$ , iteration  $k$  and  $k + 1$ , respectively, for a given  $s$ . Similarly, we use  $x$  and  $y$  for the corresponding points at iteration  $s + 1$ . These are defined by

$$\begin{aligned} z(\tau, k) &:= \left[ \theta_{s,k}^\tau; \mathcal{L}^\tau(\theta_{s,k}^\tau) \right] & x(\tau, k) &:= \left[ \theta_{s+1,k}^\tau; \mathcal{L}^\tau(\theta_{s+1,k}^\tau) \right] \\ h(\tau, k) &:= \left[ \psi_{s,k+1}^\tau; \mathcal{L}^\tau(\psi_{s,k+1}^\tau) \right] & y(\tau, k) &:= \left[ \psi_{s+1,k+1}^\tau; \mathcal{L}^\tau(\psi_{s+1,k+1}^\tau) \right]. \end{aligned} \quad (5.16)$$

Recall that  $\psi_{s,k+1}^\tau = \theta_{s,k+1}^\tau$  by assumption. We drop arguments and it is implicitly understood that  $z$  refers to  $z(\tau, k)$  and similarly for  $h, x, y$ . Denote by  $\mathbb{E}_k^\tau := \mathbb{E}_{\tau \sim p(\tau)} \sum_{k=1}^K$  the expectation over gradient paths. We write

$$\begin{aligned} \tilde{\mathcal{J}}(\theta_0^s, \Psi^s) &= \mathbb{E}_k^\tau \|h - z\|_2^p & \tilde{\mathcal{J}}(\theta_0^s, \Psi^{s+1}) &= \mathbb{E}_k^\tau \|y - z\|_2^p \\ \tilde{\mathcal{J}}(\theta_0^{s+1}, \Psi^s) &= \mathbb{E}_k^\tau \|h - x\|_2^p & \tilde{\mathcal{J}}(\theta_0^{s+1}, \Psi^{s+1}) &= \mathbb{E}_k^\tau \|y - x\|_2^p, \end{aligned} \quad (5.17)$$

where the notation  $\mathbb{E}_k^\tau \|h - z\|_2^p$  means  $\mathbb{E}_{\tau \sim p(\tau)} \sum_{k=1}^K \|h(\tau, k) - z(\tau, k)\|_2^p$  and  $p = 2$  defines the meta objective in terms of the gradient path energy and  $p = 1$  in terms of the gradient path length. As we are exclusively concerned with the Euclidean norm, we omit the subscript. By assumption, every  $\beta^s$  is sufficiently small to satisfy the gradient descent criteria  $\tilde{\mathcal{J}}(\theta_0^s; \Psi^s) \geq \tilde{\mathcal{J}}(\theta_0^{s+1}; \Psi^s)$ . Adding and subtracting  $\tilde{\mathcal{J}}(\theta_0^{s+1}, \Psi^{s+1})$  to the RHS, we have

$$\begin{aligned} \mathbb{E}_k^\tau \|h - z\|^p &\geq \mathbb{E}_k^\tau \|h - x\|^p \\ &= \mathbb{E}_k^\tau \|y - x\|^p + \|h - x\|^p - \|y - x\|^p. \end{aligned} \quad (5.18)$$

It follows that  $\mathbb{E}^\tau d(\theta_0^s, \mathcal{M}^\tau) \geq \mathbb{E}^\tau d(\theta_0^{s+1}, \mathcal{M}^\tau)$  if  $\mathbb{E}_k^\tau \|h - x\|^p \geq \mathbb{E}_k^\tau \|y - x\|^p$ . As our main concern is existence, we will show something stronger, namely that there exists  $\alpha(\tau, k) := \alpha_k^\tau$  such that

$$\|h - x\|^p \geq \|y - x\|^p \quad \forall \tau, k, s, p \quad (5.19)$$

with at least one such inequality strict for some  $\tau, k, s$  (and all  $p$ ), in which case  $d_p(\theta_0^{s+1}, \mathcal{M}^\tau) < d_p(\theta_0^s, \mathcal{M}^\tau)$  for any  $p \in \{1, 2\}$ . We proceed by establishing the inequality for  $p = 2$  and obtain  $p = 1$  as an immediate consequence of

monotonicity of the square root. Expanding  $\|h - x\|^2$  we have

$$\begin{aligned}\|h - x\|^2 - \|y - x\|^2 &= \|(h - z) + (z - x)\|^2 - \|y - x\|^2 \\ &= \|h - z\|^2 + 2\langle h - z, z - x \rangle + \|z - x\|^2 - \|y - x\|^2.\end{aligned}\quad (5.20)$$

Every term except  $\|z - x\|^2$  can be minimized by choosing  $\alpha$  small, whereas  $\|z - x\|^2$  is controlled by  $\beta^s$ . Thus, our strategy is to make all terms except  $\|z - x\|^2$  small, for a given  $\beta^s$ , by placing an upper bound on  $\alpha$ . We first show that  $\|h - z\|^2 - \|y - x\|^2 = O(\alpha^2)$ . Some care is needed as the  $(n+1)$ th dimension is the loss value associated with the other  $n$  dimensions. Define  $\hat{z}(\tau, k) := \theta_{s,k}^\tau$ , so that  $z = [\hat{z}; \mathcal{L}^\tau(\hat{z})]$ . Similarly define  $\hat{x}, \hat{h}$ , and  $\hat{y}$  to obtain

$$\begin{aligned}\|h - z\|^2 &= \|\hat{h} - \hat{z}\|^2 + (\mathcal{L}^\tau(\hat{h}) - \mathcal{L}^\tau(\hat{z}))^2 \\ \|y - x\|^2 &= \|\hat{y} - \hat{x}\|^2 + (\mathcal{L}^\tau(\hat{y}) - \mathcal{L}^\tau(\hat{x}))^2 \\ 2\langle h - z, z - x \rangle &= 2\langle \hat{h} - \hat{z}, \hat{z} - \hat{x} \rangle + (\mathcal{L}^\tau(\hat{h}) - \mathcal{L}^\tau(\hat{z}))(\mathcal{L}^\tau(\hat{z}) - \mathcal{L}^\tau(\hat{x})).\end{aligned}\quad (5.21)$$

Consider  $\|\hat{h} - \hat{z}\|^2 - \|\hat{y} - \hat{x}\|^2$ . Since  $\hat{h} = \hat{z} - \alpha \nabla \mathcal{L}(\hat{z})$  and  $\hat{y} = \hat{x} - \alpha \nabla \mathcal{L}(\hat{x})$ , it follows that  $\|\hat{h} - \hat{z}\|^2 = \alpha^2 \|\nabla \mathcal{L}(\hat{z})\|^2$  and similarly for  $\|\hat{y} - \hat{x}\|^2$ , so

$$\|\hat{h} - \hat{z}\|^2 - \|\hat{y} - \hat{x}\|^2 = \alpha^2 \left( \|\nabla \mathcal{L}(\hat{z})\|^2 - \|\nabla \mathcal{L}(\hat{x})\|^2 \right) = O(\alpha^2). \quad (5.22)$$

Now consider  $(\mathcal{L}^\tau(\hat{h}) - \mathcal{L}^\tau(\hat{z}))^2 - (\mathcal{L}^\tau(\hat{y}) - \mathcal{L}^\tau(\hat{x}))^2$ . Using the above identities and first-order Taylor series expansion, we have

$$\begin{aligned}(\mathcal{L}^\tau(\hat{h}) - \mathcal{L}^\tau(\hat{z}))^2 &= \left( \langle \nabla \mathcal{L}^\tau(\hat{z}), (\hat{h} - \hat{z}) \rangle + O(\alpha) \right)^2 \\ &= \left( -\alpha \|\nabla \mathcal{L}^\tau(\hat{z})\|^2 + O(\alpha) \right)^2 = O(\alpha^2),\end{aligned}\quad (5.23)$$

and similarly for  $(\mathcal{L}^\tau(\hat{y}) - \mathcal{L}^\tau(\hat{x}))^2$ . As such,  $\|h - z\|^2 - \|y - x\|^2 = O(\alpha^2)$ .

From this it follows that  $\langle h - z, z - x \rangle = -\alpha \langle \nabla \mathcal{L}(\hat{z}), \hat{z} - \hat{x} \rangle + O(\alpha^2)$ . Returning to the quantity of interest,  $\|h - x\|^2 - \|y - x\|^2$ , we have

$$\begin{aligned}\|h - x\|^2 - \|y - x\|^2 &= \|z - x\|^2 + 2\langle h - z, z - x \rangle + O(\alpha^2) \\ &= \|z - x\|^2 - 2\alpha \langle \nabla \mathcal{L}(\hat{z}), \hat{z} - \hat{x} \rangle + O(\alpha^2).\end{aligned}\quad (5.24)$$

The first term is non-negative, and importantly, always non-zero whenever  $\beta^s \neq 0$ . Furthermore,  $\alpha$  can always be made sufficiently small for  $\|z - x\|^2$  to dominate the residual, so we can focus on the inner product  $\langle \nabla \mathcal{L}(\hat{z}), \hat{z} - \hat{x} \rangle$ . If it is negative, all terms are positive and we have  $\|h - x\|^2 \geq \|y - x\|^2$  as desired. If not,  $\|z - x\|^2$  dominates if

$$\alpha \leq \frac{\|z - x\|^2}{2\langle \nabla \mathcal{L}(\hat{z}), \hat{z} - \hat{x} \rangle} \in (0, \infty). \quad (5.25)$$

Thus, for  $\alpha$  sufficiently small, we have  $\|h - x\|^2 \geq \|y - x\|^2 \forall \tau, k, s$ , with strict inequality whenever  $\langle \nabla \mathcal{L}(\hat{z}), \hat{z} - \hat{x} \rangle < 0$  or the bound on  $\alpha$  holds strictly. This establishes  $d_2(\theta_0^{s+1}, \mathcal{M}^\tau) \leq d_2(\theta_0^s, \mathcal{M}^\tau)$  for all  $\tau, s$ , with strict inequality for at least some  $\tau, s$ . To also establish it for the gradient path length ( $p = 1$ ), taking square roots on both sides of  $\|h - x\|^2 \geq \|y - x\|^2$  yields the desired results, and so  $\|h - x\|^p \geq \|y - x\|^p$  for  $p \in \{1, 2\}$ , and therefore

$$d(\theta_0^{s+1}, \mathcal{M}^\tau) = \tilde{\mathcal{J}}(\theta_0^{s+1}; \Psi^{s+1}) \leq \tilde{\mathcal{J}}(\theta_0^s; \Psi^s) = d(\theta_0^s, \mathcal{M}^\tau) \quad \forall \tau, s \quad (5.26)$$

with strict inequality for at least some  $\tau, s$ ; in particular, for some  $\beta^s \neq 0$  and some  $\alpha_k^\tau$  sufficiently small.

Then, to see that the limit point of  $\Psi^{s+1}$  is the same as that of  $\Psi^s$  for  $\beta^s$  sufficiently small, we use that the distance between  $h \in \Psi^s$  and  $z \in \Psi^s$  is greater than between  $h$  and  $x \in \Psi^{s+1}$  by virtue of gradient descent (Eq. 5.18). Hence  $\Psi^{s+1}$  is bound to  $\Psi^s$  and the inner loop is convergent, this means that  $\Psi^{s+1}$  must be converging to the same limit point. To establish this, note that  $x(\tau, k) = y(\tau, k - 1)$  and write Eq. 5.18 as

$$\mathbb{E}_k^\tau \|h(\tau, k) - x(\tau, k)\|^p = \mathbb{E}_k^\tau \|h(\tau, k) - y(\tau, k - 1)\|^p \quad (5.27)$$

$$\leq \mathbb{E}_k^\tau \|h(\tau, k) - z(\tau, k)\|^p \quad (5.28)$$

$$= \mathbb{E}_k^\tau \alpha^p \|\nabla \mathcal{L}(\hat{z}(\tau, k))\|^p + O(\alpha^p). \quad (5.29)$$

Define  $\epsilon(\tau, k)$  as the noise residual from the expectation; each  $y(\tau, k - 1)$  is bounded by  $\|h(\tau, k) - y(\tau, k - 1)\|^p \leq \alpha^p \|\nabla \mathcal{L}(\hat{z})\|^p + \epsilon(\tau, k)$ . For  $\beta^s$  small this noise component vanishes, and since  $\{\alpha_k^\tau\}_{k=1}^\infty$  is a converging sequence, the bound on  $y(\tau, k - 1)$  grows increasingly tight in  $k$ . Hence  $\{\theta_k^{s+1}\}_{k=1}^\infty$  converges to the same limit point as  $\{\theta_k^s\}_{k=1}^\infty$ , yielding  $\theta_0^{s+1} \in \Theta$  for all  $s$ , as desired. ■

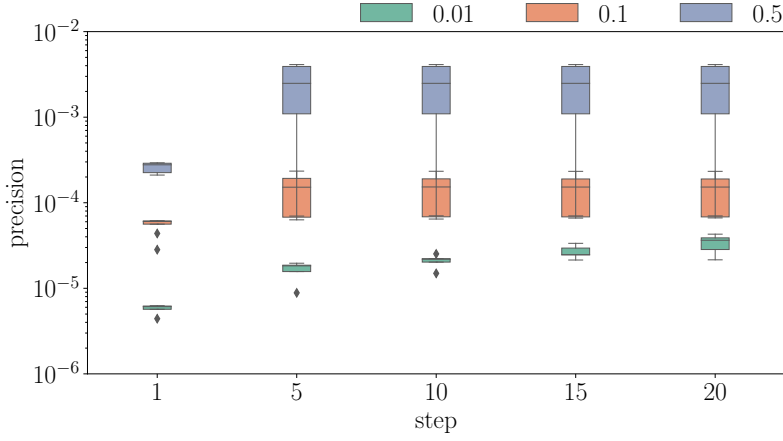


Figure 5.5: Precision of Jacobian approximation. Precision is calculated for the Jacobian of the first layer across learning rates (colors) and gradient steps.

## 5.B Ablation Study: Approximating Jacobians

First, note that (we drop task subscripts for simplicity)

$$D_{\theta_0} \theta_{k+1} = \left( I_n - \alpha \nabla_{\theta_k}^2 \mathcal{L}(\theta_k) \right) D_{\theta_0} \theta_k \quad (5.30)$$

$$= \prod_{s=0}^k \left( I_n - \alpha \nabla_{\theta_s}^2 \mathcal{L}(\theta_s) \right) \quad (5.31)$$

$$= I_n - \alpha \left( \sum_{s=0}^k \nabla_{\theta_s}^2 \mathcal{L}(\theta_s) \right) + O(\alpha^2). \quad (5.32)$$

Thus, we can directly control the quality of the approximation via  $\alpha$  (or more generally,  $\{\alpha_k\}_{k=1}^K$ ). In particular, changes to  $\theta_{k+1}$  are translated into  $\theta_0$  via intermediary Hessians. This makes these Jacobians memoryless up to second-order curvature, motivating this approximation beyond the fact that it works well in practice (c.f. Finn et al., 2017; Nichol et al., 2018). As a practical matter, if the alternative is some other approximation to the Hessians, the amount of noise injected grows exponentially with every iteration. The problem of devising an accurate low-variance estimator for the  $\nabla_{\theta_k}^2 \mathcal{L}(\theta_k)$  is highly challenging and beyond the scope of this paper.

To understand how this approximation limits our choice of learning rates  $\alpha - k$ , we conduct an ablation study in the Omniglot experiment setting. We are

interested in the relative precision of the identity approximation under different learning rates and across time steps, defined under the Schatten-1 norm as

$$\rho(k, \alpha) = \frac{\|I_n - D_{\theta_0} \theta_k\|_1}{\|D_{\theta_0}(\theta_k)\|_1}. \quad (5.33)$$

We use the same four-layer convolutional neural network as in the Omniglot experiment (Appendix 5.D). For each choice of learning rate, we train a model from a random initialization for 20 steps and compute  $\rho$  every 5 steps. Due to exponential growth of memory consumption, we were unable to compute  $\rho$  for more than 20 gradient steps. We report the relative precision of the first convolutional layer. We do not report the Jacobian with respect to other layers, all being considerably larger, as computing their Jacobians was too costly. We computed  $\rho$  for all layers on the first five gradient steps and found no significant variation in precision across layers. Consequently, we prioritize reporting how precision varies with the number of gradient steps. As in the main experiments, we use stochastic gradient descent. We evaluate  $\alpha \in \{0.01, 0.1, 0.5\}$  across 5 different tasks. Figure 5.5 summarizes our results.

Reassuringly, we find the identity approximation to be accurate to at least the fourth decimal for learning rates we use in practice, and to the third decimal for the largest learning rate (0.5) we were able to converge with. Importantly, except for the smallest learning rate, the quality of the approximation is constant in the number of gradient steps. The smallest learning rate that exhibits some deterioration on the fifth decimal, however larger learning rates provide an upper bound that is constant on the fourth decimal, indicating that this is of minor concern. Finally, we note that while these results suggest the identity approximation to be a reasonable approach on the class of problems we consider, other settings may put stricter limits on the effective size of learning rates.

## 5.C Ablation Study: Leap Hyper-Parameters

We have several degrees of freedom in specifying a meta learner. In particular, we are free to choose the task manifold structure, the gradient path distance metric,  $d_p$ , and whether to incorporate stabilizers. These are non-trivial choices and to ascertain the importance of each, we conduct an ablation study. We vary (a) the task manifold between using the full loss surface and only parameter space, (b) the gradient path distance metric between using the energy or length, and (c) inclusion of the stabilizer  $\mu$  in the meta objective. We stay as close as possible to the set-up used in the Omniglot experiment (Appendix 5.D),

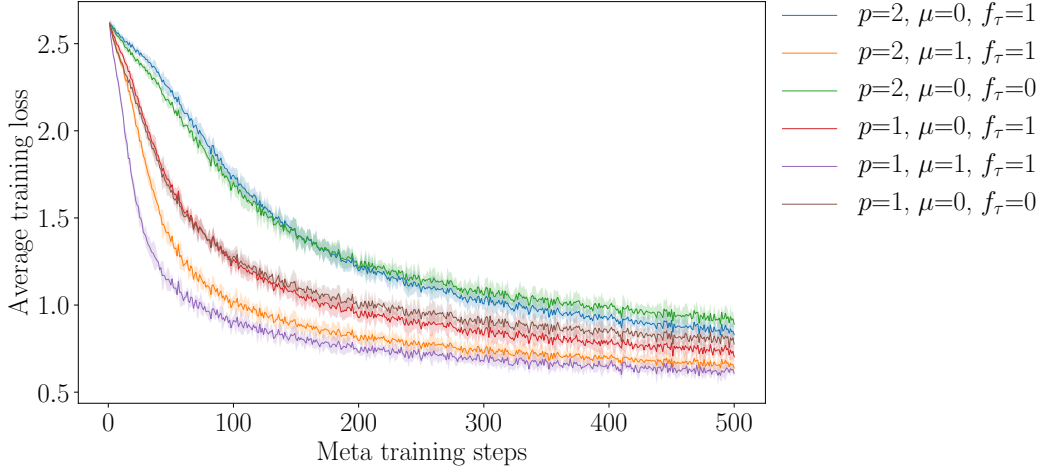


Figure 5.6: Average task training loss over meta-training steps.  $p$  denotes the  $\tilde{d}_p$  used in the meta objective,  $\mu = 1$  the use of the stabilizer, and  $\mathcal{L}^\tau = 1$  the inclusion of the loss in the task manifold.

fixing the number of pretraining tasks to 20. We perform 500 meta gradient updates; other hyper-parameters are the same.

Our ablation study indicates that the richer the task manifold and the more accurate the gradient path length is approximated, the better Leap performs (Figure 5.6). Further, adding a stabilizer has the intended effect and leads to significantly faster convergence. The simplest configuration, defined in terms of the gradient path energy and with the task manifold identifies as parameter space, yields a meta gradient equivalent to the update rule used in Reptile. We find this configuration to be less efficient in terms of convergence and we observe a significant deterioration in performance. Extending the task manifold to the loss surface does not improve meta-training convergence speed, but does cut prediction error in half. Adding the stabilizer significantly speeds up convergence. These conclusions also hold under the gradient path length as distance measure, and in general using the gradient path length does better than using the gradient path energy as the distance measure.

## 5.D Experiment Details: Omniglot

Omniglot contains 50 alphabets, each with a set of characters that in turn have 20 unique samples. We treat each alphabet as a distinct task and pretrain on up to 25 alphabets, holding out 10 out for final evaluation. We use data augmentation on all tasks to render the problem challenging. We augment any image with a random affine transformation by (a) random sampling a



Table 5.2: Mean test error after 100 training steps on held out evaluation tasks.<sup>†</sup>Multi-headed finetuning.<sup>‡</sup>No pretraining. No. tasks refers to number of meta-training tasks.

Method No. tasks	Leap	Reptile	FT <sup>†</sup>	MAML	FOMAML	NP <sup>‡</sup>
1	62.3	59.8	46.5	64.0	64.5	82.3
3	46.5	46.5	36.0	56.2	59.0	82.3
5	40.3	41.4	32.5	50.1	53.0	82.5
10	32.6	35.6	28.7	49.3	49.6	82.9
15	29.6	33.3	26.9	45.5	47.8	82.6
20	26.0	30.8	24.7	41.7	45.4	82.6
25	24.8	29.4	23.5	42.9	44.0	82.8

scaling factor between  $[0.8, 1.2]$ , (b) random rotation between  $[0, 360]$ , and (c) randomly cropping the height and width by a factor between  $[-0.2, 0.2]$  in each dimension. Our setup differs significantly from other protocols (Vinyals et al., 2016; Finn et al., 2017), where tasks are defined by selecting different permutations of characters (irrespective of alphabet) and restricting the number of samples available for each character.

We use a similar convolutional neural network architecture as in previous works (Vinyals et al., 2016; Schwarz et al., 2018). The networks stacks a convolutional module four times, where each model is comprised of a  $3 \times 3$  convolution with 64 filters, followed by  $2 \times 2$  max-pooling, batch-normalization, and ReLU activation. All images are downsampled to  $28 \times 28$ , resulting in a  $1 \times 1 \times 64$  feature map that is passed on to a final linear layer. We define a task as a 20-class classification problem with classes drawn from a distinct alphabet. For alphabets with more than 20 characters, we pick 20 characters at random, alphabets with fewer characters (4) are dropped from the task set. On each task, we train a model using stochastic gradient descent. For each model, we evaluated learning rates in the range  $[0.001, 0.01, 0.1, 0.5]$ ; we found 0.1 to be the best choice in all cases. See Table 5.3 for further hyper-parameters.

We meta-train for 1000 steps unless otherwise noted; on each task we train for 100 steps. Increasing the number of steps used for task training yields similar results, albeit at greater computational expense. For each character in an alphabet, we hold out 5 samples in order to create a task validation set.

Table 5.3: Summary of hyper-parameters for Omniglot. “Meta” refers to the outer training loop, “task” refers to the inner training loop.

	Leap	Finetuning	Reptile	MAML	FOMAML
Meta training					
Learning rate	0.1	—	0.1	0.5	0.5
Training steps	1000	1000	1000	1000	1000
Batch size (tasks)	20	20	20	20	20
Task training					
Learning rate	0.1	0.1	0.1	0.1	0.1
Training steps	100	100	100	5	100
Batch size (samples)	20	20	20	20	20
Task evaluation					
Learning rate	0.1	0.1	0.1	0.1	0.1
Training steps	100	100	100	100	100
Batch size (samples)	20	20	20	20	20

Table 5.4: Summary of hyper-parameters for Multi-CV. “Meta” refers to the outer training loop, “task” refers to the inner training loop.

	Leap	Finetuning	Progressive Nets	HAT
Meta training				
Learning rate	0.01	—	—	—
Training steps	1000	1000	1000	1000
Batch size	10	10	10	10
Task training				
Learning rate	0.1	0.1	0.1	0.1
Max epochs	1	1	1	1
Batch size	32	32	32	32
Task evaluation				
Learning rate	0.1	0.1	0.1	0.1
Training epochs	100	100	100	100
Batch size	32	32	32	32

## 5.E Experiment Details: Multi-CV

Architectures differ between tasks through different final linear layers. We use the same convolutional encoder as in the Omniglot experiment (Appendix 5.D). Leap learns an initialization for the convolutional encoder; on each task, the final linear layer is always randomly initialized. We compare Leap against (a) a baseline with no pretraining, (b) multitask finetuning, (c) HAT (Serrá et al., 2018), and (d) Progressive Nets (Rusu et al., 2016). For HAT, we use the original formulation, but allow multiple task revisits (until convergence). For Progressive Nets, we allow lateral connections between all tasks and multiple task revisits (until convergence). Note that this makes Progressive Nets over 8 times larger in terms of learnable parameters than the other models.

We train using SGD with cosine annealing (Loshchilov & Hutter, 2017). During meta training, we sample a batch of 10 tasks at random from the pretraining set and train until the early stopping criterion is triggered or the maximum amount of epochs is reached (see Table 5.4). We used the same interval for selecting learning rates as in the Omniglot experiment (Appendix 5.D). Only Leap benefited from using more than 1 epoch as the upper limit on task training steps during pretraining.

In the case of Leap, the initialization is updated after all tasks in the meta batch has been trained to convergence; for other models, there is no distinction between initialization and task parameters. On a given task, training is stopped if the maximum number of epochs is reached (Table 5.4) or if the validation error fails to improve over 10 consecutive gradient steps. Meta training is stopped once the mean validation error fails to improve over 10 consecutive meta training batches. We use Adam (Kingma & Ba, 2015) for the meta gradient update with a constant learning rate of 0.01. We use no dataset augmentation. MNIST images are zero padded to have  $32 \times 32$  images; we use the same normalizations as Serrá et al. (2018).

Table 5.5: Transfer learning results on Multi-CV benchmark. All methods are trained until convergence on held-out tasks. <sup>†</sup>Area under training error curve; scaled to 0–100. <sup>‡</sup>Our implementation.

Held-out task	Method	Test (%)	Train (%)	AUC <sup>†</sup>
Facescrub	Leap	19.9	0.0	11.6
	Finetuning	32.7	0.0	13.2
	Progressive Nets <sup>‡</sup>	<b>18.0</b>	0.0	<b>8.9</b>
	HAT <sup>‡</sup>	25.6	0.1	14.6
	No pretraining	18.2	0.0	10.5
NotMNIST	Leap	<b>5.3</b>	<b>0.6</b>	<b>2.9</b>
	Finetuning	5.4	2.0	4.4
	Progressive Nets <sup>‡</sup>	5.4	3.1	3.7
	HAT <sup>‡</sup>	6.0	2.8	5.4
	No pretraining	5.4	2.6	5.1
MNIST	Leap	<b>0.7</b>	0.1	<b>0.6</b>
	Finetuning	0.9	0.1	0.8
	Progressive Nets <sup>‡</sup>	0.8	<b>0.0</b>	0.7
	HAT <sup>‡</sup>	0.8	0.3	1.2
	No pretraining	0.9	0.2	1.0
Fashion MNIST	Leap	<b>8.0</b>	4.2	<b>6.8</b>
	Finetuning	8.9	<b>3.8</b>	7.0
	Progressive Nets <sup>‡</sup>	8.7	5.4	9.2
	HAT <sup>‡</sup>	9.5	5.5	8.1
	No pretraining	8.4	4.7	7.8
Cifar10	Leap	<b>21.2</b>	<b>10.8</b>	<b>17.5</b>
	Finetuning	27.4	13.3	20.7
	Progressive Nets <sup>‡</sup>	24.2	15.2	24.0
	HAT <sup>‡</sup>	27.7	21.2	27.3
	No pretraining	26.2	13.1	23.0
SVHN	Leap	<b>8.4</b>	<b>5.6</b>	<b>7.5</b>
	Finetuning	10.9	6.1	10.5
	Progressive Nets <sup>‡</sup>	10.1	6.3	13.8
	HAT <sup>‡</sup>	10.5	5.7	8.5
	No pretraining	10.3	6.9	11.5
Cifar100	Leap	<b>52.0</b>	<b>30.5</b>	<b>43.4</b>
	Finetuning	59.2	31.5	44.1
	Progressive Nets <sup>‡</sup>	55.7	42.1	54.6
	HAT <sup>‡</sup>	62.0	49.8	58.4
	No pretraining	54.8	33.1	50.1
Traffic Signs	Leap	<b>2.9</b>	0.0	<b>1.2</b>
	Finetuning	5.7	0.0	1.7
	Progressive Nets <sup>‡</sup>	3.6	0.0	4.0
	HAT <sup>‡</sup>	5.4	0.0	2.3
	No pretraining	3.6	0.0	2.4

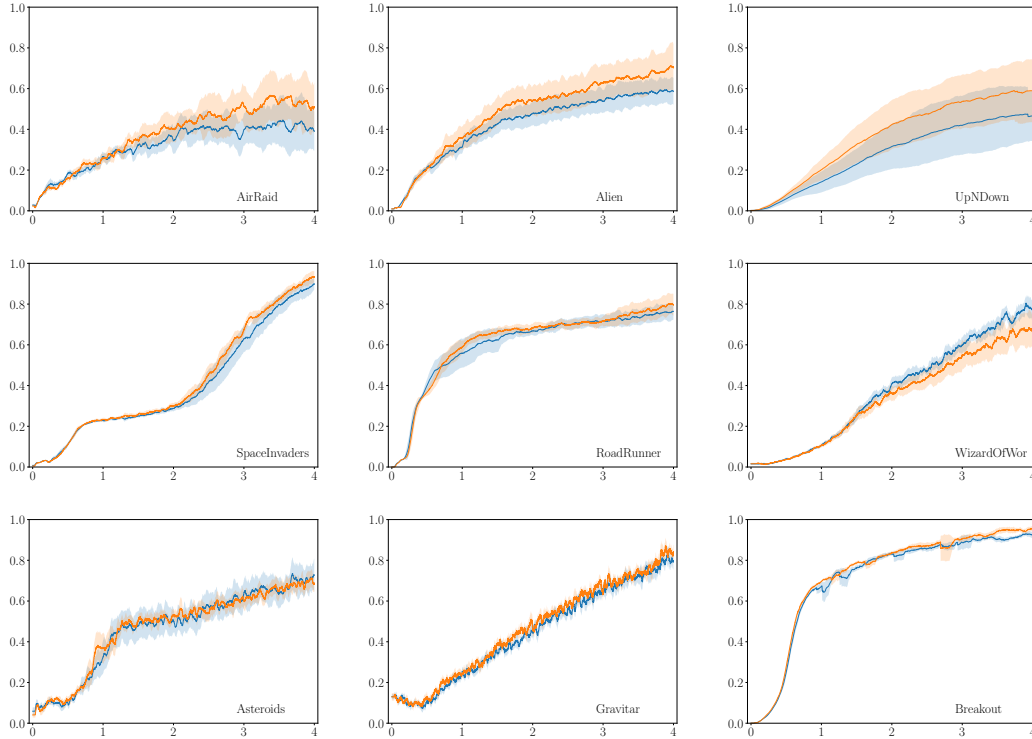


Figure 5.7: Mean normalized episode scores on Atari games across training steps. Scores are reported as moving average over 500 episodes. Shaded regions depict two standard deviations across ten seeds. KungFuMaster, RoadRunner and Krull have action state spaces that are twice as large as the largest action state encountered during pretraining. Leap (orange) generally outperforms a random initialization, except for WizardOfWor, where a random initialization does better on average due to outlying runs under Leap’s initialization.

## 5.F Experiment Details: Atari

We use the same network as in [Mnih et al. \(2013\)](#), adopting it to actor-critic algorithms by estimating both value function and policy through linear layers connected to the final output of a shared convolutional network. Following standard practice, we use downsampled  $84 \times 84 \times 3$  RGB images as input. Leap is applied with respect to the convolutional encoder (as final linear layers vary in size across environments).

We use all environments with an action space of at most 10 as our pretraining pool, holding out Breakout and SpaceInvaders. During meta training, we sample a batch of 16 games at random from a pretraining pool of 27 games. On each game in the batch, a network is initialized using the shared initialization and trained independently for 5 million steps, accumulating the meta gradient across games on the fly. Thus, the baseline and Leap differs only with respect

Table 5.6: Evaluation environment characteristics. <sup>†</sup>Standard Deviation; calculated on baseline (no pretraining) data.

Environment	Action Space	Mean Reward <sup>†</sup>	St. Dev. <sup>†</sup>	Pretraining Env
AirRaid	6	2538	624	Y
UpNDown	6	52417	2797	Y
WizardOfWor	10	2531	182	Y
Breakout	4	338	13	N
SpaceInvaders	6	1065	103	N
Asteroids	14	1760	139	N
Alien	18	1280	182	N
Gravitar	18	329	15	N
RoadRunner	18	29593	2890	N

to the initialization of the convolutional encoder. We trained Leap for 100 steps, equivalent to training 1600 agents for 5 million steps.

The meta learned initialization was evaluated on the held-out games, a random selection of games seen during pretraining, and a random selection of games with action spaces larger than 10 (Table 5.6). On each task, we use a batch size of 32, an unroll length of 5 and update the model parameters with RMSProp (using  $\epsilon = 10^{-4}$ ,  $\alpha = 0.99$ ) with a learning rate of  $10^{-4}$ . We set the entropy cost to 0.01 and clip the absolute value of the rewards to maximum 5.0. We use a discounting factor of 0.99.

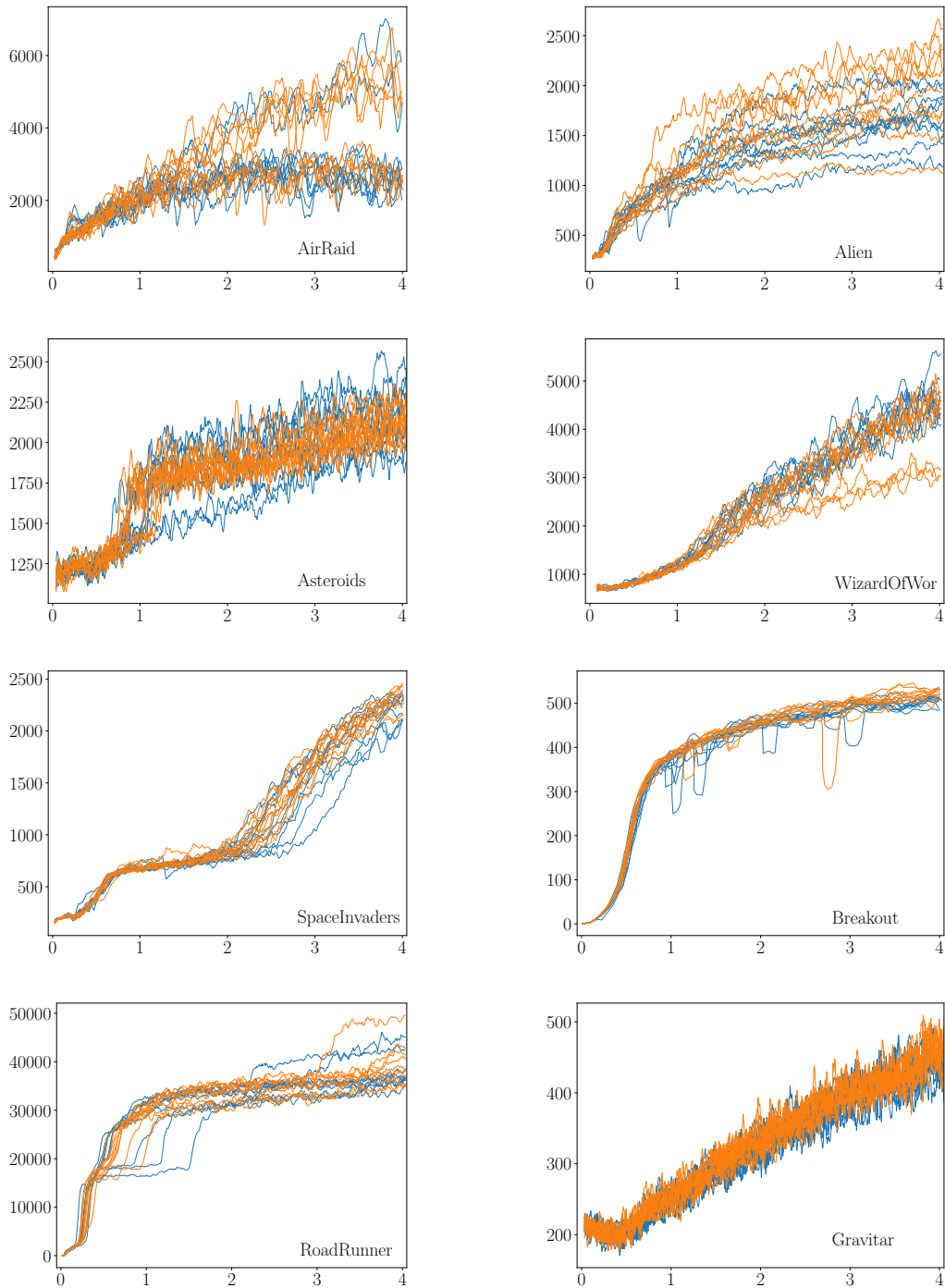


Figure 5.8: Mean episode scores on Atari games across training steps for different runs. Scores are reported as moving average over 500 episodes. Leap (orange) outperforms a random initialization by being less volatile.

## 6 General-Purpose Meta-Learning

---

The previous two chapters proposed a method for adaptive parameterisation and a geometric perspective on gradient-based meta-learning, respectively. In this chapter, these two pieces are put together into a general-purpose gradient-based meta-learner. This work formalises the notion that separate layers in a network serve different purposes. This form of modular meta-learning embeds meta-knowledge into a task learner’s hidden representation that serves as a form of gradient preconditioning.

By preconditioning gradients, learning is projected into a space that is meta-learned to facilitate learning over a certain class of problems. Meta-learned hidden layers in a neural network control this space. These meta-learned layers are equivalent to learning a mapping  $\Omega : \mathcal{P} \rightarrow \mathcal{M}$  from a meta-learned space  $\mathcal{P}$  onto the task’s loss surface  $\mathcal{M}$ . The proposed method maps a task distribution into a shared geometry  $\mathcal{P}$  where task solutions are easier to find.

Because the meta-learner layers act on gradients as preconditioning matrices, this work leverages Riemannian geometry to derive a meta-objective that avoids backpropagation through the task adaptation process on first principles. To the best of our knowledge, this is the only gradient-based meta-learner that enjoys this property. Appealingly, because meta-knowledge is represented as hidden layers in a neural network, our proposed method becomes extremely flexible. It makes very few assumptions on the structure of the task; all that is required is a task objective, a meta-objective defined under the task, and a task learner defined under the meta-learner. It makes no assumption about the type of task learning (online, offline, continual, etc.) or the length of task adaptation trajectories—in fact it does not assume the existence of such trajectories. Instead, it relies on inducing an empirical distribution over parameter space without imposing restriction on the form of the distribution. This paper demonstrates that it is directly applicable to many forms of learning, including few-shot and many-shot learning, as well as continual learning and reinforcement learning.



## Meta-Learning with Warped Gradient Descent

Peer-reviewed publication (Flennerhag et al., 2020a); notational modifications.

Published in *International Conference on Learning Representations (Oral)*. 2020.

Authors: **Sebastian Flennerhag**, University of Manchester,  
 Andrei A. Rusu, DeepMind,  
 Razvan Pascanu, DeepMind,  
 Francesco Visin, DeepMind,  
 Hujun Yin, University of Manchester,  
 Raia Hadsell, DeepMind,

**Abstract.** *Learning an efficient update rule from data that promotes rapid learning of new tasks from the same distribution remains an open problem in meta-learning. Typically, previous works have approached this issue either by attempting to train a neural network that directly produces updates or by attempting to learn better initialisations or scaling factors for a gradient-based update rule. Both of these approaches pose challenges. On one hand, directly producing an update forgoes a useful inductive bias and can easily lead to non-converging behaviour. On the other hand, approaches that try to control a gradient-based update rule typically resort to computing gradients through the learning process to obtain their meta-gradients, leading to methods that can not scale beyond few-shot task adaptation. In this work, we propose Warped Gradient Descent (WarpGrad), a method that intersects these approaches to mitigate their limitations. WarpGrad meta-learns an efficiently parameterised preconditioning matrix that facilitates gradient descent across the task distribution. Preconditioning arises by interleaving non-linear layers, referred to as warp-layers, between the layers of a task-learner. Warp-layers are meta-learned without backpropagating through the task training process in a manner similar to methods that learn to directly produce updates. WarpGrad is computationally efficient, easy to implement, and can scale to arbitrarily large meta-learning problems. We provide a geometrical interpretation of the approach and evaluate its effectiveness in a variety of settings, including few-shot, standard supervised, continual and reinforcement learning.*

---

## 6.1 Introduction

Learning (how) to learn implies inferring a learning strategy from some set of past experiences via a *meta-learner* that a *task-learner* can leverage when learning a new task. One approach is to directly parameterise an update rule via the memory of a recurrent neural network (Andrychowicz et al., 2016; Ravi & Larochelle, 2017; Li & Malik, 2016; Chen et al., 2017). Such *memory-based methods* can, in principle, represent any learning rule by virtue of

being universal function approximators (Cybenko, 1989; Hornik, 1991; Schäfer & Zimmermann, 2007). They can also scale to long learning processes, but they lack an inductive bias as to what constitutes a reasonable learning rule. This renders them hard to train and brittle to generalisation as their parameter updates have no guarantees of convergence.

An alternative approach defines a *gradient-based* update rule and meta-learns a shared initialisation that facilitates task adaptation across a distribution of tasks (Finn et al., 2017; Nichol et al., 2018; Flennerhag et al., 2019). Such methods are imbued with a strong inductive bias—gradient descent—but restrict knowledge transfer to the initialisation. Recent work has shown that it is beneficial to more directly control gradient descent by meta-learning an approximation of a *parameterised matrix* (Li et al., 2017; Lee & Choi, 2018; Park & Oliva, 2019) that *preconditions* gradients during task training, similarly to second-order and Natural Gradient Descent methods (Nocedal & Wright, 2006; Amari & Nagaoka, 2007). To meta-learn preconditioning, these methods backpropagate through the gradient descent process, limiting them to few-shot learning.

In this paper, we propose a novel framework called Warped Gradient Descent (WarpGrad),<sup>9</sup> that relies on the inductive bias of gradient-based meta-learners by defining an update rule that preconditions gradients, but that is meta-learned using insights from memory-based methods. In particular, we leverage that gradient preconditioning is defined point-wise in parameter space and can be seen as a recurrent operator of order 1. We use this insight to define a trajectory agnostic meta-objective over a joint parameter search space where knowledge transfer is encoded in gradient preconditioning.

To achieve a scalable and flexible form of preconditioning, we take inspiration from works that embed preconditioning in task-learners (Desjardins et al., 2015; Lee & Choi, 2018), but we relax the assumption that task-learners are feed-forward and replace their linear projection with a generic neural network  $\omega$ , referred to as a *warp layer*. By introducing non-linearity, preconditioning is rendered data-dependent. This allows WarpGrad to model preconditioning beyond the block-diagonal structure of prior works and enables it to meta-learn over arbitrary adaptation processes.

We evaluate WarpGrad empirically in a variety of learning paradigms, spanning supervised, continual and reinforcement learning. We show that it surpasses baseline gradient-based meta-learners on standard few-shot learning tasks (*miniImageNet*, *tieredImageNet*; Vinyals et al., 2016; Ravi & Larochelle,

<sup>9</sup>Open-source implementation available at <https://github.com/flennerhag/warpgrad>.



agating through  $K$  steps of gradient descent across a task distribution  $p(\tau)$ ,

$$C^{\text{MAML}}(\xi) := \sum_{\tau \sim p(\tau)} \mathcal{L}_{\mathcal{D}_{\text{test}}^{\tau}} \left( \theta_0 - \alpha \sum_{k=0}^{K-1} U_{\mathcal{D}_{\text{train}}^{\tau}}(\theta_k^{\tau}; \xi) \right). \quad (6.1)$$

Subsequent works on gradient-based meta-learning differ in the parameterisation of  $U$ . Meta-SGD (MSGD; [Li & Malik, 2016](#)) learns a vector of learning rates, Meta-Curvature (MC; [Park & Oliva, 2019](#)) defines a block-diagonal preconditioning matrix  $B$ , and T-Nets ([Lee & Choi, 2018](#)) embed block-diagonal preconditioning in feed-forward learners via linear projections,

$$U(\theta_k; \theta_0) := \theta_k - \alpha \nabla \mathcal{L}(\theta_k) \quad \text{MAML} \quad (6.2)$$

$$U(\theta_k; \theta_0, \phi) := \theta_k - \alpha \text{diag}(\phi) \nabla \mathcal{L}(\theta_k) \quad \text{MSGD} \quad (6.3)$$

$$U(\theta_k; \theta_0, \phi) := \theta_k - \alpha B(\phi) \nabla \mathcal{L}(\theta_k) \quad \text{MC} \quad (6.4)$$

$$U(\theta_k; \theta_0, \phi) := \theta_k - \alpha \nabla \mathcal{L}(\theta_k; \phi) \quad \text{T-Nets.} \quad (6.5)$$

These methods optimise meta-parameters  $\xi = \{\theta_0, \phi\}$  by backpropagating through the gradient descent process (Eq. 6.1), which results in a trajectory dependence in the meta-objective. This limits them to few-shot learning as they become (1) computationally expensive, (2) susceptible to exploding/vanishing gradients, and (3) subject to a credit assignment problem ([Wu et al., 2018](#); [Antoniou et al., 2019](#); [Liu et al., 2019](#)).

Our goal is to develop a meta-learner that overcomes all three limitations. To do so, we depart from the paradigm of backpropagating to the initialisation and exploit the fact that learning to precondition gradients can be seen as a Markov Process of order 1 that depends on the state but not the trajectory ([Li et al., 2017](#)). That is,  $U$  can be seen as a recurrent operator  $U : \theta, \phi \mapsto \theta$  and if  $U$  defines gradient preconditioning, this takes the form  $U(\theta; \phi) := \theta - \alpha P(\theta; \phi) \nabla \mathcal{L}(\theta)$ . Hence, learning  $\phi$  does not require backpropagation through the entire gradient descent process, just the expected one-step gradient update.

To develop this notion formally, we first establish a general-purpose form of preconditioning (Section 6.2.2). Based on this, we obtain a canonical meta-objective from a geometrical point of view (Section 6.2.3), from which we derive a trajectory-agnostic meta-objective (Section 6.2.4).

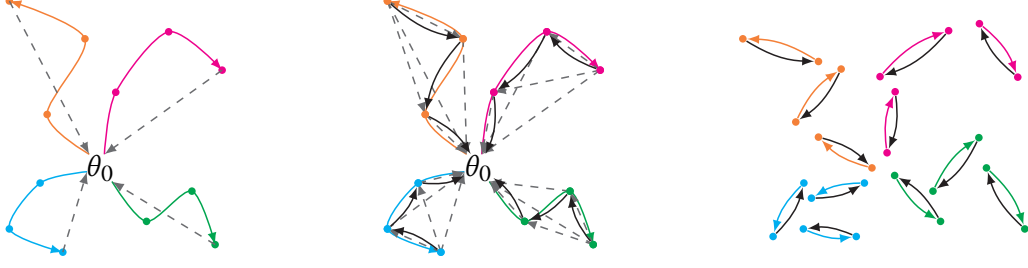


Figure 6.2: Gradient-based meta-learning. Colours denote different tasks ( $\tau$ ), dashed lines denote backpropagation through the adaptation process, and solid black lines denote optimiser parameter ( $\phi$ ) gradients w.r.t. one step of task parameter ( $\theta$ ) adaptation. *Left*: A meta-learned initialisation compresses trajectory information into a single initial point ( $\theta_0$ ). *Middle*: MAML-based optimisers interact with adaptation trajectories at every step and backpropagate through each interaction. *Right*: WarpGrad is trajectory agnostic. Task adaptation defines an empirical distribution  $p(\tau, \theta)$  over which WarpGrad learns a geometry for adaptation by optimising for steepest descent directions.

### 6.2.2 General-Purpose Preconditioning

A preconditioned gradient descent rule,  $U(\theta; \phi) := \theta - \alpha P(\theta; \phi) \nabla \mathcal{L}(\theta)$ , defines a geometry via  $P$ . To disentangle the expressive capacity of this geometry from the expressive capacity of the task-learner  $F$ , we take inspiration from T-Nets that embed linear projections  $T$  in feed-forward layers,  $f = \sigma(TWx + b)$ . This in itself is not sufficient to achieve disentanglement due to linearity in  $T$  and  $W$ , but it can be achieved under non-linear preconditioning.

To this end, we relax the assumption that the task-learner is feed-forward and consider an arbitrary neural network,  $F = f^{(L)} \circ \dots \circ f^{(1)}$ . We insert warp-layers that are universal function approximators parameterised by neural networks into the task-learner without restricting their form or how they interact with  $F$ . In the simplest case, we interleave warp-layers between layers of the task-learner to obtain  $\hat{F} = \omega^{(L)} \circ f^{(L)} \circ \dots \circ \omega^{(1)} \circ f^{(1)}$ , but other forms of interaction can be beneficial (see Appendix 6.A for practical guidelines). Backpropagation automatically induces gradient preconditioning, as in T-Nets, but in our case via the Jacobians of the warp-layers:

$$\frac{\partial \mathcal{L}}{\partial \theta^{(i)}} = \mathbb{E} \left[ \nabla \ell^T \left( \prod_{j=0}^{L-(i+1)} D_x \omega^{(L-j)} D_x f^{(L-j)} \right) D_x \omega^{(i)} D_\theta f^{(i)} \right], \quad (6.6)$$

where  $D_x$  and  $D_\theta$  denote the Jacobian with respect to input and parameters, respectively. In the special case where  $F$  is feed-forward and each  $\omega$  a linear

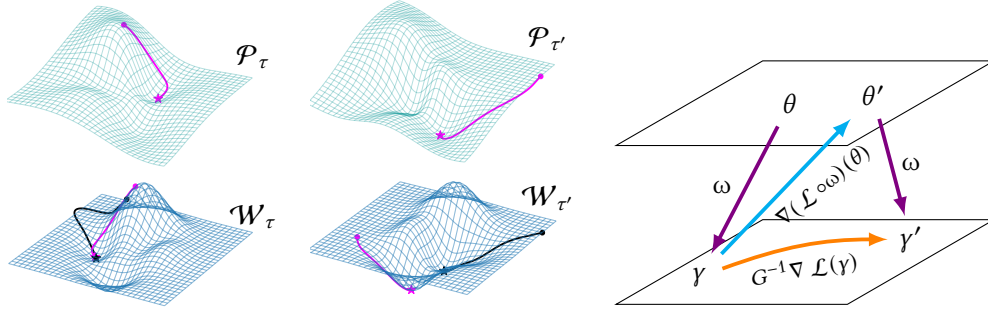


Figure 6.3: *Left*: synthetic experiment illustrating how WarpGrad warps gradients (see Appendix 6.D for full details). Each task  $f \sim p(f)$  defines a distinct loss surface ( $\mathcal{W}$ , bottom row). Gradient descent (black) on these surfaces struggles to find a minimum. WarpGrad meta-learns a *warp*  $\omega$  to produce better update directions (magenta; Section 6.2.4). In doing so, WarpGrad learns a meta-geometry  $\mathcal{P}$  where standard gradient descent is well behaved (top row). *Right*: gradient descent in  $\mathcal{P}$  is equivalent to first-order Riemannian descent in  $\mathcal{W}$  under a meta-learned Riemann metric (Section 6.2.3).

projection, we obtain an instance of WarpGrad that is akin to T-Nets since preconditioning is given by  $D_x \omega = T$ . Conversely, by making warp-layers non-linear, we can induce interdependence between warp-layers, allowing WarpGrad to model preconditioning beyond the block-diagonal structure imposed by prior works. Further, this enables a form of task-conditioning by making Jacobians of warp-layers data dependent. As we have made no assumptions on the form of the task-learner or warp-layers, WarpGrad methods can act on any neural network through any form of warping, including recurrence.

We show that increasing the capacity of the meta-learner by defining warp-layers as Residual Networks (He et al., 2017) improves performance on classification tasks (Section 6.4.1). We also introduce recurrent warp-layers for agents in a gradient-based meta-learner that is the first, to the best of our knowledge, to outperform memory-based meta-learners on a maze navigation task that requires memory (Section 6.4.3).

Warp-layers imbue WarpGrad with three powerful properties. First, due to preconditioned gradients, WarpGrad inherits gradient descent properties, importantly guarantees of convergence. Second, warp-layers form a distributed representation of preconditioning that disentangles the expressiveness of the geometry it encodes from the expressive capacity of the task-learner. Third, warp-layers are meta-learned across tasks and trajectories and can therefore capture properties of the task-distribution beyond local information. Figure 6.3 illustrates these properties in a synthetic scenario, where we construct a family

of tasks  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  (see Appendix 6.D for details) and meta-learn across the task distribution. WarpGrad learns to produce warped loss surfaces (illustrated on two tasks  $\tau$  and  $\tau'$ ) that are smoother and more well-behaved than their respective native loss-surfaces.

### 6.2.3 The Geometry of Warped Gradient Descent

If the matrix  $P$  is invertible, it defines a valid Riemann metric (Amari, 1998) and therefore enjoys similar convergence guarantees to gradient descent. Thus, if warp-layers represent a valid (meta-learned) Riemann metric, WarpGrad is well-behaved. For T-Nets, it is sufficient to require  $T$  to be full rank, since  $T$  defines  $P$  as a block-diagonal matrix with block entries  $TT^T$ . In contrast, non-linearity in warp-layers precludes such an explicit identification.

Instead, we must consider the geometry that warp-layers represent. For this, we need a metric tensor,  $G$ , which is a positive-definite, smoothly varying matrix that measures curvature on a manifold  $\mathcal{W}$ . The metric tensor defines the steepest direction of descent by  $-G^{-1}\nabla \mathcal{L}$  (Lee, 2003), hence our goal is to establish that warp-layers approximate some  $G^{-1}$ . Let  $\Omega$  represent the effect of warp-layers by a reparameterisation  $f^{(i)}(x; \Omega(\theta; \phi))^{(i)} = \omega^{(i)}(f^{(i)}(x; \theta^{(i)}); \phi) \forall x, i$  that maps from a space  $\mathcal{P}$  onto the manifold  $\mathcal{W}$  with  $\gamma = \Omega(\theta; \phi)$ . We induce a metric  $G$  on  $\mathcal{W}$  by push-forward (Figure 6.2):

$$\Delta\theta := \nabla(\mathcal{L} \circ \Omega)(\theta; \phi) = [D_x\Omega(\theta; \phi)]^T \nabla \mathcal{L}(\gamma) \quad \mathcal{P}\text{-space} \quad (6.7)$$

$$\Delta\gamma := D_x\Omega(\theta; \phi) \Delta\theta = G(\gamma; \phi)^{-1} \nabla \mathcal{L}(\gamma) \quad \mathcal{W}\text{-space}, \quad (6.8)$$

where  $G^{-1} := [D_x\Omega][D_x\Omega]^T$ . Provided  $\Omega$  is not degenerate ( $G$  is non-singular),  $G^{-1}$  is positive-definite, hence a valid Riemann metric. While this is the metric induced on  $\mathcal{W}$  by warp-layers, it is not the metric used to precondition gradients since we take gradient steps in  $\mathcal{P}$  which introduces an error term (Figure 6.2). We can bound the error by first-order Taylor series expansion to establish first-order equivalence between the WarpGrad update in  $\mathcal{P}$  (Eq. 6.7) and the ideal update in  $\mathcal{W}$  (Eq. 6.8),

$$\underbrace{(\mathcal{L} \circ \Omega)(\theta - \alpha \Delta\theta)}_{\mathcal{P}\text{-space}} = \underbrace{\mathcal{L}(\gamma - \alpha \Delta\gamma)}_{\mathcal{W}\text{-space}} + \mathcal{O}(\alpha^2). \quad (6.9)$$



Consequently, gradient descent under warp-layers (in  $\mathcal{P}$ -space) is first-order equivalent to warping the native loss surface under a metric  $G$  to facilitate task adaptation. Warp parameters  $\phi$  control the geometry induced by warping, and therefore *what* task-learners converge to. By meta-learning  $\phi$  we can accumulate information that is conducive to task adaptation but that may not be available during that process. This suggests that an ideal geometry (in  $\mathcal{W}$ -space) should yield preconditioning that points in the direction of steepest descent, accounting for global information across tasks,

$$\min_{\phi} \mathbb{E}_{\mathcal{L}, \gamma \sim p(\mathcal{L}, \gamma)} \left[ \mathcal{L} \left( \gamma - \alpha G(\gamma; \phi)^{-1} \nabla \mathcal{L}(\gamma) \right) \right]. \quad (6.10)$$

In contrast to MAML-based approaches (Eq. 6.1), this objective avoids back-propagation through learning processes. Instead, it defines task learning abstractly as optimising the expected gradient update over the joint task parameter distribution. Hence while preconditioning is locally defined, the expectation over tasks and parameters is global. This decoupling allows us to scale beyond few shot-learning and opens up for general-purpose meta-learning.

#### 6.2.4 Meta-Learning Warp Parameters

The canonical objective in Eq. 6.10 describes a meta-objective for learning a geometry on first principles that we can render into a trajectory-agnostic update rule for warp-layers. We define a task  $\tau = (F^\tau, \mathcal{L}_{\text{meta}}^\tau, \mathcal{L}_{\text{task}}^\tau)$  by a task-learner  $\hat{F}$  that is embedded with a shared WarpGrad optimiser, a meta-training objective  $\mathcal{L}_{\text{meta}}^\tau$ , and a task adaptation objective  $\mathcal{L}_{\text{task}}^\tau$ . We use  $\mathcal{L}_{\text{task}}^\tau$  to adapt task parameters  $\theta$  and  $\mathcal{L}_{\text{meta}}^\tau$  to adapt warp parameters  $\phi$ . Note that we allow meta and task objectives to differ in arbitrary ways, but both are expectations over some data, as above. In the simplest case, they differ in terms of validation versus training data, but they may differ in terms of learning paradigm as well, as we demonstrate in continual learning experiment (Section 6.4.3).

To obtain our meta-objective, we recast the canonical objective (Eq. 6.10) in terms of  $\theta$  using first-order equivalence of gradient steps (Eq. 6.9). Next, we factorise  $p(\tau, \theta)$  into  $p(\theta | \tau)p(\tau)$ . Since  $p(\tau)$  is given, it remains to consider a sampling strategy for  $p(\theta | \tau)$ . For meta-learning of warp-layers, we assume this distribution is given. We show how to incorporate meta-learning of a prior  $p(\theta_0 | \tau)$  in Section 6.2.5. While any sampling strategy is valid, in this paper we exploit that task learning under stochastic gradient descent can be seen as sampling from an empirical prior  $p(\theta | \tau)$  (Grant et al., 2018); in particular, each iterate  $\theta_k^\tau$  can be seen as a sample from  $p(\theta_k^\tau | \theta_{k-1}^\tau, \phi)$ . Thus,  $K$ -steps



of gradient descent forms a Monte-Carlo chain  $\theta_0^\tau, \dots, \theta_K^\tau$  and sampling such chains define an empirical distribution  $p(\theta \mid \tau)$  around some prior  $p(\theta_0 \mid \tau)$ , which we will discuss in Section 6.2.5. The joint distribution  $p(\tau, \theta)$  defines a joint search space across tasks. Meta-learning therefore learns a geometry over this space with the steepest expected direction of descent. This direction is however not with respect to the objective that produced the gradient,  $\mathcal{L}_{\text{task}}^\tau$ , but with respect to  $\mathcal{L}_{\text{meta}}^\tau$ ,

$$L(\phi) := \sum_{\tau \sim p(\tau)} \sum_{\theta^\tau \sim p(\theta \mid \tau)} \mathcal{L}_{\text{meta}}^\tau \left( \theta^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta^\tau; \phi); \phi \right). \quad (6.11)$$

Decoupling the task gradient operator  $\nabla \mathcal{L}_{\text{task}}^\tau$  from the geometry learned by  $\mathcal{L}_{\text{meta}}^\tau$  lets us infuse global knowledge in warp-layers, a promising avenue for future research (Metz et al., 2019; Mendonca et al., 2019). For example, in Section 6.4.3, we meta-learn an update-rule that mitigates catastrophic forgetting by defining  $\mathcal{L}_{\text{meta}}^\tau$  over current and previous tasks. In contrast to other gradient-based meta-learners, the WarpGrad meta-objective is an expectation over gradient update steps sampled from the search space induced by task adaptation (for example,  $K$  steps of stochastic gradient descent; Figure 6.2). It is therefore trajectory agnostic and hence compatible with arbitrary task learning processes. Because the meta-gradient is independent of the number of task gradient steps, it avoids vanishing/exploding gradients and the credit assignment problem by design. It does rely on second-order gradients, a requirement we can relax by detaching task parameter gradients ( $\nabla \mathcal{L}_{\text{task}}^\tau$ ),

$$\hat{L}(\phi) := \sum_{\tau \sim p(\tau)} \sum_{\theta^\tau \sim p(\theta \mid \tau)} \mathcal{L}_{\text{meta}}^\tau \left( \text{sg} [\theta^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta^\tau; \phi)]; \phi \right), \quad (6.12)$$

where  $\text{sg}$  is the stop-gradient operator. In contrast to the first-order approximation of MAML (Finn et al., 2017), which ignores the entire trajectory except for the final gradient, this approximation retains all gradient terms and only discards local second-order effects, which are typically dominated by first-order effect in long parameter trajectories (Flennerhag et al., 2019). Empirically, we find that our approximation only incurs a minor loss of performance in an ablation study (Appendix 6.F). Interestingly, this approximation is a form of multitask learning with respect to  $\phi$  (Li & Hoiem, 2016; Bilen & Vedaldi, 2017; Rebuffi et al., 2017) that marginalises over task parameters  $\theta^\tau$ .

### 6.2.5 Integration with Learned Initialisations

WarpGrad is a method for learning warp layer parameters  $\phi$  over a joint search space defined by  $p(\tau, \theta)$ . Because WarpGrad takes this distribution as given, we can integrate WarpGrad with methods that define or learn some form of “prior”  $p(\theta_0 | \tau)$  over  $\theta_0^\tau$ . For instance, (a) *Multi-task solution*: in online learning, we can alternate between updating a multi-task solution and tuning warp parameters. We use this approach in our Reinforcement Learning experiment (Section 6.4.3); (b) *Meta-learned point-estimate*: when task adaptation occurs in batch mode, we can meta-learn a shared initialisation  $\theta_0$ . Our few-shot and supervised learning experiments take this approach (Section 6.4.1); (c) *Meta-learned prior*: WarpGrad can be combined with Bayesian methods that define a full prior (Rusu et al., 2019; Oreshkin et al., 2018; Lacoste et al.,

---

**Algorithm 6.1** WarpGrad: online meta-training

---

**Require:**  $p(\tau)$ : distribution over tasks  
**Require:**  $\alpha, \beta, \lambda$ : hyper-parameters  
1: initialise  $\phi$  and  $p(\theta_0 | \tau)$   
2: **while** not done **do**  
3:   sample mini-batch of tasks  $\mathcal{T}$  from  $p(\tau)$   
4:    $g_\phi, g_{\theta_0} \leftarrow 0$   
5:   **for all**  $\tau \in \mathcal{T}$  **do**  
6:      $\theta_0^\tau \sim p(\theta_0 | \tau)$   
7:     **for all**  $k$  in  $0, \dots, K_\tau - 1$  **do**  
8:        $\theta_{k+1}^\tau \leftarrow \theta_k^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi)$   
9:        $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta_k^\tau)$   
10:        $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta_0; \theta_{0:k}^\tau)$   
11:     **end for**  
12:   **end for**  
13:    $\phi \leftarrow \phi - \beta g_\phi$   
14:    $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$   
15: **end while**

---



---

**Algorithm 6.2** WarpGrad: offline meta-training

---

**Require:**  $p(\tau)$ : distribution over tasks  
**Require:**  $\alpha, \beta, \lambda, \eta$ : hyper-parameters  
1: initialise  $\phi, p(\theta_0 | \tau)$   
2: **while** not done **do**  
3:   initialise  $\mathcal{B} = \{\mathcal{B}_\tau\}_\tau, \mathcal{B}_\tau = \{\}$   
4:   sample mini-batch of tasks  $\mathcal{T}$  from  $p(\tau)$   
5:   **for all**  $\tau \in \mathcal{T}$  **do**  
6:      $\theta_0^\tau \sim p(\theta_0 | \tau)$   
7:      $\mathcal{B}[\tau] = [\theta_0^\tau]$   
8:     **for all**  $k$  in  $0, \dots, K_\tau - 1$  **do**  
9:        $\theta_{k+1}^\tau \leftarrow \theta_k^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi)$   
10:        $\mathcal{B}_\tau = \mathcal{B}_\tau \cup \theta_{k+1}^\tau$   
11:     **end for**  
12:   **end for**  
13:    $i, g_\phi, g_{\theta_0} \leftarrow 0$   
14:   **for all**  $(\tau, k) \in \mathcal{B}$  **do**  
15:      $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta_k^\tau)$   
16:      $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta_0; \theta_{0:k}^\tau)$   
17:      $i \leftarrow i + 1$   
18:     **if**  $i = \eta$  **then**  
19:        $\phi \leftarrow \phi - \beta g_\phi$   
20:        $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$   
21:        $i, g_\phi, g_{\theta_0} \leftarrow 0$   
22:     **end if**  
23:   **end for**  
24: **end while**

---

2018; Kim et al., 2018a). We incorporate such methods by some objective  $C$  (potentially vacuous) over  $\theta_0$  that we optimise jointly with WarpGrad,

$$\mathcal{J}(\phi, \theta_0) := L(\phi) + \lambda C(\theta_0), \quad (6.13)$$

where  $L$  can be substituted for by  $\hat{L}$  and  $\lambda \in [0, \infty)$  is a hyper-parameter. We train the WarpGrad optimiser via stochastic gradient descent and solve Eq. 6.13 by alternating between sampling task parameters from  $p(\tau, \theta)$  given the current parameter values for  $\phi$  and taking meta-gradient steps over these samples to update  $\phi$ . As such, our method can also be seen as a generalised form of gradient descent in the form of Mirror Descent with a meta-learned dual space (Desjardins et al., 2015; Beck & Teboulle, 2003). The details of the sampling procedure may vary depending on the specifics of the tasks (static, sequential), the design of the task-learner (feed-forward, recurrent), and the learning objective (supervised, self-supervised, reinforcement learning). In Algorithm 6.1 we illustrate a simple online algorithm with constant memory and linear complexity in  $K$ , assuming the same holds for  $C$ . A drawback of this approach is that it is relatively data inefficient; in Appendix 6.B we detail a more complex offline training algorithm that stores task parameters in a replay buffer for mini-batched training of  $\phi$ . The gains of the offline variant can be dramatic: in our Omniglot experiment (Section 6.4.1), offline meta-training allows us to update warp parameters 2000 times with each meta-batch, improving final test accuracy from 76.3% to 84.3% (Appendix 6.F).

## 6.3 Related Work

Learning to learn, or meta-learning, has previously been explored in a variety of settings. Early work focused on evolutionary approaches (Schmidhuber, 1987; Bengio et al., 1991; Thrun & Pratt, 1998). Hochreiter et al. (2001) introduced gradient descent methods to meta-learning, specifically for recurrent meta-learning algorithms, extended to RL by Wang et al. (2016a) and Duan et al. (2016). A similar approach was taken by Andrychowicz et al. (2016) and Ravi & Larochelle (2017) to meta-learn a parameterised update rule in the form of a Recurrent Neural Network (RNN).

A related separates parameters into “slow” and “fast” weights, where the former captures meta-information and the latter encapsulates rapid adaptation (Hinton & Plaut, 1987; Schmidhuber, 1992; Ba et al., 2016). This can be implemented by embedding a neural network that dynamically adapts the parameters of a main

architecture (Ha et al., 2017). WarpGrad can be seen as learning slow warp-parameters that precondition adaptation of fast weights. Recent meta-learning focuses almost exclusively on few-shot learning, where tasks are characterised by severe data scarcity. In this setting, tasks must be sufficiently similar that a new task can be learned from a single or handful of examples (Lake et al., 2015; Vinyals et al., 2016; Snell et al., 2017; Ren et al., 2018).

Several meta-learners have been proposed that directly predict the parameters of the task-learner (Bertinetto et al., 2016; Munkhdalai et al., 2018; Gidaris & Komodakis, 2018; Qiao et al., 2018). To scale, such methods typically pretrain a feature extractor and predict a small subset of the parameters. Closely related to our work are gradient-based few-shot learning methods that extend MAML by sharing some subset of parameters between task-learners that is fixed during task training but meta-learner across tasks, which may reduce overfitting (Mishra et al., 2018; Lee & Choi, 2018; Munkhdalai et al., 2018) or induce more robust convergence (Zintgraf et al., 2019). It can also be used to model latent variables for concept or task inference, which implicitly induce gradient modulation (Zhou et al., 2018; Oreshkin et al., 2018; Rusu et al., 2019; Lee et al., 2019a). Our work is also related to gradient-based meta-learning of a shared initialisation that scales beyond few-shot learning (Nichol et al., 2018; Flennerhag et al., 2019).

Meta-learned preconditioning is closely related to parallel work on second-order optimisation methods for high dimensional non-convex loss surfaces (Nocedal & Wright, 2006; Saxe et al., 2013; Kingma & Ba, 2015; Arora et al., 2018). In this setting, second-order optimisers typically struggle to improve upon first-order baselines (Sutskever et al., 2013). As second-order curvature is typically intractable to compute, such methods resort to low-rank approximations (Nocedal & Wright, 2006; Martens, 2010; Martens & Grosse, 2015) and suffer from instability (Byrd et al., 2016). In particular, Natural Gradient Descent (Amari, 1998) is a method that uses the Fisher Information Matrix as curvature metric (Amari & Nagaoka, 2007). Several proposed methods for amortising the cost of estimating the metric (Pascanu & Bengio, 2014; Martens & Grosse, 2015; Desjardins et al., 2015).

As noted by Desjardins et al. (2015), expressing preconditioning through interleaved projections can be seen as a form of Mirror Descent (Beck & Teboulle, 2003). WarpGrad offers a new perspective on gradient preconditioning by introducing a generic form of model-embedded preconditioning that exploits global information beyond the task at hand.

## 6.4 Experiments

We evaluate WarpGrad in a set of experiments designed to answer three questions: (1) do WarpGrad methods retain the inductive bias of MAML-based few-shot learners? (2) Can WarpGrad methods scale to problems beyond the reach of such methods? (3) Can WarpGrad generalise to complex meta-learning problems?

### 6.4.1 Few-Shot Learning

For few-shot learning, we test whether WarpGrad retains the inductive bias of gradient-based meta-learners while avoiding backpropagation through the gradient descent process. To isolate the effect of the WarpGrad objective, we use *linear* warp-layers that we train using online meta-training (Algorithm 6.1) to make WarpGrad as close to T-Nets as possible. For a fair comparison, we meta-learn the initialisation using MAML (Warp-MAML) with  $J(\theta_0, \phi) := L(\phi) + \lambda C^{\text{MAML}}(\theta_0)$ . We evaluate the importance of meta-learning the initialisation in Appendix 6.G and find that WarpGrad achieves similar performance under random task parameter initialisation.

All task-learners use a convolutional architecture that stacks 4 blocks made up of a  $3 \times 3$  convolution, max-pooling, batch-norm, and ReLU activation. We define Warp-MAML by inserting warp-layers in the form of  $3 \times 3$  convolutions after each block in the baseline task-learner. All baselines are tuned with identical and independent hyper-parameter searches (including filter sizes—full experimental settings in Appendix 6.H), and we report best results from our experiments or the literature. Warp-MAML outperforms all baselines (Table 6.1), improving 1- and 5-shot accuracy by 3.6 and 5.5 percentage points on *miniImageNet* (Vinyals et al., 2016; Ravi & Larochelle, 2017) and by 5.2 and 3.8 percentage points on *tieredImageNet* (Ren et al., 2018), which indicates that WarpGrad retains the inductive bias of MAML-based meta-learners.

### 6.4.2 Multi-Shot Learning

Next, we evaluate whether WarpGrad can scale beyond few-shot adaptation on similar supervised problems. We propose a new protocol for *tieredImageNet* that increases the number of adaptation steps to 640 and use 6 convolutional blocks in task-learners, which are otherwise defined as above. Since MAML-based approaches cannot backpropagate through 640 adaptation steps for models of this size, we evaluate WarpGrad against two gradient-based meta-learners that meta-learn an initialisation without such backpropagation, Reptile (Nichol

Table 6.1: Mean test accuracy after task adaptation on held out evaluation tasks.

<sup>†</sup>Multi-headed. <sup>‡</sup>No meta-training; see Appendix 6.E and Appendix 6.H.

<i>miniImageNet</i>		
	5-way 1-shot	5-way 5-shot
Reptile	50.0 $\pm$ 0.3	66.0 $\pm$ 0.6
Meta-SGD	50.5 $\pm$ 1.9	64.0 $\pm$ 0.9
(M)T-Net	51.7 $\pm$ 1.8	–
CAVIA (512)	51.8 $\pm$ 0.7	65.9 $\pm$ 0.6
MAML	48.7 $\pm$ 1.8	63.2 $\pm$ 0.9
<b>Warp-MAML</b>	<b>52.3 <math>\pm</math> 0.8</b>	<b>68.4 <math>\pm</math> 0.6</b>

<i>tieredImageNet</i>		
	5-way 1-shot	5-way 5-shot
MAML	51.7 $\pm$ 1.8	70.3 $\pm$ 1.8
<b>Warp-MAML</b>	<b>57.2 <math>\pm</math> 0.9</b>	<b>74.1 <math>\pm</math> 0.7</b>

<i>Multi-Shot</i>		
	<i>tieredImageNet</i> 10-way 640-shot	Omniglot 20-way 100-shot
SGD <sup>‡</sup>	58.1 $\pm$ 1.5	51.0
KFAC <sup>‡</sup>	–	56.0
Finetuning <sup>†</sup>	–	76.4 $\pm$ 2.2
Reptile	76.52 $\pm$ 2.1	70.8 $\pm$ 1.9
Leap	73.9 $\pm$ 2.2	75.5 $\pm$ 2.6
<b>Warp-Leap</b>	<b>80.4 <math>\pm</math> 1.6</b>	<b>83.6 <math>\pm</math> 1.9</b>

et al., 2018) and Leap (Flennerhag et al., 2019), and we define a Warp-Leap meta-learner by  $J(\theta_0, \phi) := L(\phi) + \lambda C^{\text{Leap}}(\theta_0)$ . Leap is an attractive complement as it minimises the expected gradient descent trajectory length across tasks. Under WarpGrad, this becomes a joint search for a geometry in which task adaptation defines geodesics (shortest paths, see Appendix 6.C for details). While Reptile outperforms Leap by 2.6 percentage points on this benchmark, Warp-Leap surpasses both, with a margin of 3.88 to Reptile (Table 6.1).

We further evaluate Warp-Leap on the multi-shot Omniglot (Lake et al., 2011) protocol proposed by Flennerhag et al. (2019), where each of the 50 alphabets is a 20-way classification task. Task adaptation involves 100 gradient steps on random samples that are preprocessed by random affine transformations. We report results for Warp-Leap under offline meta-training (Algorithm 6.2), which updates warp parameters 2000 times per meta step (see

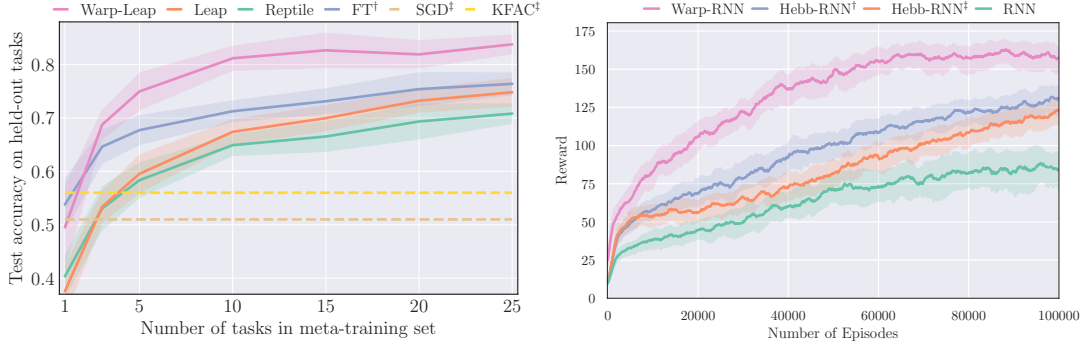


Figure 6.4: *Left*: Omniglot test accuracies on held-out tasks after meta-training on a varying number of tasks. Shading represents standard deviation across 10 independent runs. We compare Warp-Leap, Leap, and Reptile, multi-headed finetuning, as well as SGD and KFAC which used random initialisation but with 4x larger batch size and 10x larger learning rate. *Right*: On a RL maze navigation task, mean cumulative return is shown. Shading represents inter-quartile ranges across 10 independent runs. <sup>†</sup>Simple modulation and <sup>‡</sup>retroactive modulation are used (Miconi et al., 2019).

Appendix 6.E for experimental details). Warp-Leap enjoys similar performance on this task as well, improving over Leap and Reptile by 8.1 and 12.8 points respectively (Table 6.1). We also perform an extensive ablation study varying the number of tasks in the meta-training set. Except for the case of a single task, Warp-Leap substantially outperforms all baselines (Figure 6.4), achieving a higher rate of convergence and reducing the final test error from  $\sim 30\%$  to  $\sim 15\%$ . Non-linear warps, which go beyond block-diagonal preconditioning, reach  $\sim 11\%$  test error (refer to Appendix 6.F and Table 6.2 for the full results). Finally, we find that WarpGrad methods behave distinctly different from Natural Gradient Descent methods in an ablation study (Appendix 6.G). It reduces final test error from  $\sim 42\%$  to  $\sim 19\%$ , controlling for initialisation, while its preconditioning matrices differ from what the literature suggests (Desjardins et al., 2015).

### 6.4.3 Complex Meta-Learning

**(c.1) Reinforcement Learning** To illustrate how WarpGrad may be used both with recurrent neural networks and in meta-reinforcement learning, we evaluate it in a maze navigation task proposed by Miconi et al. (2018). The environment is a fixed maze and a task is defined by randomly choosing a goal location. The agent’s objective is to find the location as many times as possible, being teleported to a random location each time it finds it.

We use advantage actor-critic with a basic recurrent neural network (Wang et al., 2016a) as the task-learner, and we design a Warp-RNN as a HyperNetwork (Ha



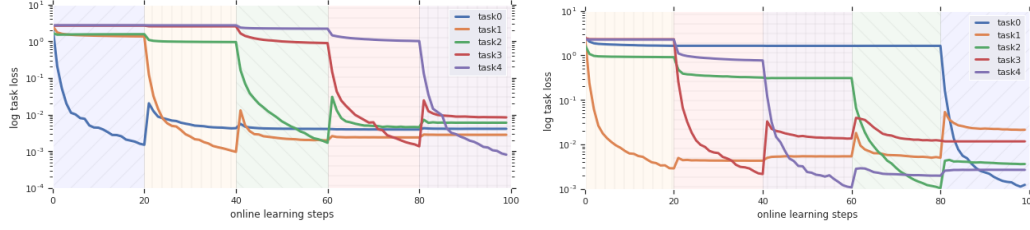


Figure 6.5: Continual learning experiment. Average log-loss over 100 randomly sampled tasks, each comprised of 5 sub-tasks. *Left*: learned sequentially as seen during meta-training. *Right*: learned in random order [sub-task 1, 3, 4, 2, 0].

et al., 2017), specifically, an adaptive LSTM (Flennerhag et al., 2018). We use a meta-LSTM that is fixed during task training to modulate the weights of the RNN that learns on each task (see Appendix 6.I for details), which in turn is trained on mini-batches of 30 episodes for 200 000 steps. We accumulate the gradient of fixed warp-parameters continually (Algorithm 6.3, Appendix 6.B) at each task parameter update. Warp parameters are updated on every 30<sup>th</sup> step on task parameters (we control for meta-LSTM capacity in Appendix 6.I).

We compare against Learning to Reinforcement Learn (Wang et al., 2016a) and Hebbian meta-learning (Miconi et al., 2018, 2019). Notably, linear warps (T-Nets) do worse than the baseline RNN on this task while the Warp-RNN converges to a mean cumulative reward of  $\sim 160$  in 60 000 episodes, compared to baselines that reach at most a mean cumulative reward of  $\sim 125$  after 100 000 episodes (Figure 6.4), reaching  $\sim 150$  after 200 000 episodes (6.I).

**(c.2) Continual Learning** We test if WarpGrad can prevent catastrophic forgetting (French, 1999) in a continual learning scenario. To this end, we design a continual learning version of the sine regression meta-learning experiment in Finn et al. (2017) by splitting the input interval  $[-5, 5] \subset \mathbb{R}$  into 5 consecutive sub-tasks (an alternative protocol was recently proposed independently by Javed & White, 2019). Each sub-task is a regression problem with the target being a mixture of two random sine waves.

We train 4-layer feed-forward task-learner with interleaved warp-layers incrementally on one sub-task at a time (see Appendix 6.J for details). To isolate the behaviour of WarpGrad parameters, we use a fixed random initialisation for each task sequence. Warp parameters are meta-learned to prevent catastrophic forgetting by defining  $\mathcal{L}_{\text{meta}}^{\tau}$  to be the average task loss over current and previous sub-tasks, for each sub-task in a task sequence. This forces warp-parameters to disentangle the adaptation process of current and



previous sub-tasks. We train on each sub-task for 20 steps, for a total of 100 task adaptation steps.

We evaluate WarpGrad on 100 random tasks and find that it learns new sub-tasks well, with mean losses on an order of magnitude  $10^{-3}$ . When switching sub-task, performance immediately deteriorates to  $\sim 10^{-2}$  but is stable for the remainder of training (Figure 6.5). Our results indicate that WarpGrad can be an effective mechanism against catastrophic forgetting, a promising avenue for further research. For detailed results, see Appendix 6.J.

## 6.5 Conclusion

We propose WarpGrad, a novel meta-learner that combines the expressive capacity and flexibility of memory-based meta-learners with the inductive bias of gradient-based meta-learners. WarpGrad meta-learns to precondition gradients during task adaptation without backpropagating through the adaptation process and we find empirically that it retains the inductive bias of MAML-based few-shot learners while being able to scale to complex problems and architectures. Further, by expressing preconditioning through warp-layers that are universal function approximators, WarpGrad can express geometries beyond the block-diagonal structure of prior works.

WarpGrad provides a principled framework for general-purpose meta-learning that integrates learning paradigms, such as continual learning, an exciting avenue for future research. We introduce novel means for preconditioning, for instance with residual and recurrent warp-layers. Understanding how WarpGrad manifolds relate to second-order optimisation methods will further our understanding of gradient-based meta-learning and aid us in designing warp-layers with stronger inductive bias.

In their current form, WarpGrad share some of the limitations of many popular meta-learning approaches. While WarpGrad avoids backpropagating through the task training process, as in *Warp-Leap*, the WarpGrad objective samples from parameter trajectories and has therefore linear computational complexity in the number of adaptation steps, currently an unresolved limitation of gradient-based meta-learning. Algorithm 6.2 hints at exciting possibilities for overcoming this limitation.

## 6

## Appendix

## 6.A WarpGrad Design Principles for Neural Nets

WarpGrad is a model-embedded meta-learned optimiser that allows for several implementation strategies. To embed warp-layers given a task-learner architecture, we may either insert new warp-layers in the given architecture or designate some layers as warp-layers and some as task layers. We found that WarpGrad can both be used in a high-capacity mode, where task-learners are relatively weak to avoid overfitting, as well as in a low-capacity mode where task-learners are powerful and warp-layers are relatively weak. The best approach depends on the problem at hand. We highlight three approaches to designing WarpGrad optimisers, starting from a given architecture:

- (a) **Model partitioning.** Given a desired architecture, designate some operations as task-adaptable and the rest as warp-layers. Task layers do not have to interleave exactly with warp-layers as gradient warping arises both through the forward pass and through backpropagation. This was how we approached the *tieredImageNet* and *miniImageNet* experiments.
- (b) **Model augmentation.** Given a model, designate all layers as task-adaptable and interleave warp-layers. Warp-layers can be relatively weak as backpropagation through non-linear activations ensures expressive gradient warping. This was our approach to the Omniglot experiment; our main architecture interleaves linear warp-layers in a standard architecture.
- (c) **Information compression.** Given a model, designate all layers as warp and interleave weak task layers. In this scenario, task-learners are prone to overfitting. Pushing capacity into the warp allows it to encode general information the task-learner can draw on during task adaptation. This approach is similar to approaches in transfer and meta-learning that restrict the number of free parameters during task training (Rebuffi et al., 2017; Lee & Choi, 2018; Zintgraf et al., 2019).

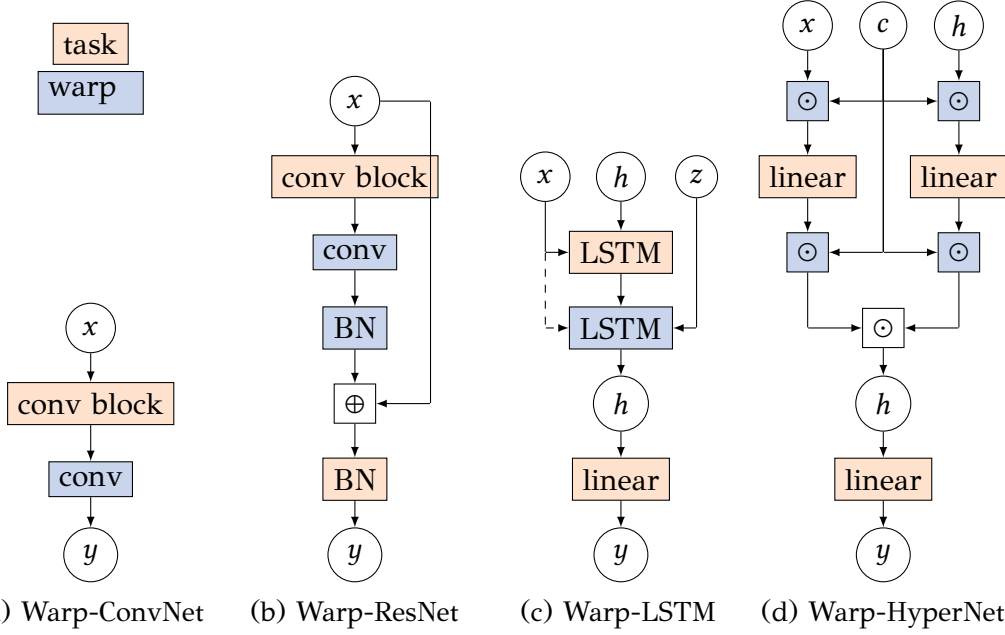


Figure 6.6: Illustration of possible WarpGrad architectures. Orange represents task layers and blue represents warp-layers.  $\oplus$  denotes residual connections and  $\odot$  any form of gating mechanism. We can obtain warped architectures by interleaving task- and warp-layers (a, c) or by designating some layers in standard architectures as task-adaptable and some as warp-layers (b, d).

Note that in either case, once warp-layers have been chosen, standard back-propagation automatically warps gradients for us. Thus, WarpGrad is fully compatible with any architecture, for instance, Residual Neural Networks (He et al., 2016) or LSTMs. For convolutional neural networks, we may use any form of convolution, learned normalization (e.g. Ioffe & Szegedy, 2015), or adaptor module (e.g. Rebuffi et al., 2017; Perez et al., 2018) to design task and warp-layers. For recurrent networks, we can use stacked LSTMs to interleave warped layers, as well as any type of HyperNetwork architecture (e.g. Ha et al., 2017; Suarez, 2017; Flennerhag et al., 2018) or partitioning of fast and slow weights (e.g. Mujika et al., 2017). Figure 6.6 illustrates this process.

## 6.B WarpGrad Meta-Training Algorithms

In this Section (Figure 6.7), we provide the variants of WarpGrad training algorithms used in this paper. Algorithm 6.1 describes a simple online algorithm, which accumulates meta-gradients online during task adaptation. This algorithm has constant memory and scales linearly in the length of task trajectories. While simple, it may not make the most efficient use of collected data.

Figure 6.7: WarpGrad algorithms. In Algorithm 6.1, meta-training occurs online in tandem with task adaptation; Algorithm 6.2 relies on using a replay buffer; Algorithm 6.3 outlines meta-training for meta-continual learning.

---

**Algorithm 6.1** Online meta-training
 

---

**Require:**  $p(\tau)$ : distribution over tasks

**Require:**  $\alpha, \beta, \lambda$ : hyper-parameters

```

1: initialise  $\phi$  and  $\theta_0$ 
2: while not done do
3:   Sample mini-batch of tasks  $\mathcal{B}$ 
   from  $p(\tau)$ 
4:    $g_\phi, g_{\theta_0} \leftarrow 0$ 
5:   for all  $\tau \in \mathcal{B}$  do
6:      $\theta_0^\tau \leftarrow \theta_0$ 
7:     for all  $k$  in  $0, \dots, K_\tau - 1$  do
8:        $\theta_{k+1}^\tau \leftarrow \theta_k^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi)$ 
9:        $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta_k^\tau)$ 
10:       $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta_0^\tau; \theta_{0:k}^\tau)$ 
11:    end for
12:  end for
13:   $\phi \leftarrow \phi - \beta g_\phi$ 
14:   $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$ 
15: end while
```

---



---

**Algorithm 6.3** Continual meta-training
 

---

**Require:**  $p(\tau)$ : distribution over tasks

**Require:**  $\alpha, \beta, \lambda, \eta$ : hyper-parameters

```

1: initialise  $\phi$  and  $\theta$ 
2:  $i, g_\phi, g_\theta \leftarrow 0$ 
3: while not done do
4:   Sample mini-batch of tasks  $\mathcal{B}$ 
   from  $p(\tau)$ 
5:   for all  $\tau \in \mathcal{B}$  do
6:      $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta)$ 
7:      $g_\theta \leftarrow g_\theta + \nabla C(\theta; \phi)$ 
8:   end for
9:    $\theta \leftarrow \theta - \lambda \beta g_\theta$ 
10:   $g_\theta, i \leftarrow 0, i + 1$ 
11:  if  $i = \eta$  then
12:     $\phi \leftarrow \phi - \beta g_\phi$ 
13:     $i, g_\theta \leftarrow 0$ 
14:  end if
15: end while
```

---



---

**Algorithm 6.2** Offline meta-training
 

---

**Require:**  $p(\tau)$ : distribution over tasks

**Require:**  $\alpha, \beta, \lambda, \eta$ : hyper-parameters

```

1: initialise  $\phi$  and  $\theta_0$ 
2: while not done do
3:   Sample mini-batch of tasks  $\mathcal{B}$ 
   from  $p(\tau)$ 
4:    $\mathcal{T} \leftarrow \{\mathcal{T}_\tau\}_{\tau \in \mathcal{B}}, \mathcal{T}_\tau \leftarrow \{\theta_0\}$ 
5:   for all  $\tau \in \mathcal{B}$  do
6:      $\theta_0^\tau \leftarrow \theta_0$ 
7:     for all  $k$  in  $0, \dots, K_\tau - 1$  do
8:        $\theta_{k+1}^\tau \leftarrow \theta_k^\tau - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi)$ 
9:        $\mathcal{T}_\tau \leftarrow \mathcal{T}_\tau \cup \theta_{k+1}^\tau$ 
10:    end for
11:  end for
12:   $i, g_\phi, g_{\theta_0} \leftarrow 0$ 
13:  while  $\mathcal{T}$  not empty do
14:    sample  $\tau, k$  without replacement
15:     $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta_k^\tau)$ 
16:     $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta_0^\tau; \theta_{0:k}^\tau)$ 
17:     $i \leftarrow i + 1$ 
18:    if  $i = \eta$  then
19:       $\phi \leftarrow \phi - \beta g_\phi$ 
20:       $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$ 
21:       $i, g_\phi, g_{\theta_0} \leftarrow 0$ 
22:    end if
23:  end while
24: end while
```

---

In Algorithm 6.2, we describe an offline meta-training algorithm. This algorithm is similar to Algorithm 6.1 in many respects, but differs in that we do not compute meta-gradients online during task adaptation. Instead, we accumulate them into a replay buffer of sampled task parameterisations.

This buffer is a Monte-Carlo sample of the expectation in the meta objective (Eq. 6.13) that can be thought of as a dataset in its own right. Hence, we can apply standard mini-batching with respect to the buffer and perform mini-batch gradient descent on warp parameters. This allows us to update warp parameters several times for a given sample of task parameter trajectories, which can greatly improve data efficiency. In our Omniglot experiment, we found offline meta-training to converge faster: in fact, a mini-batch size of 1 (i.e.  $\eta = 1$  in Algorithm 6.2) converges rapidly without any instability.

Finally, in Algorithm 6.3, we present a continual meta-training process where meta-training occurs throughout a stream of learning experiences. Here,  $C$  represents a multi-task objective, such as the average task loss,  $C^{\text{multi}} = \sum_{\tau \sim p(\tau)} \mathcal{L}_{\text{task}}^{\tau}$ . Meta-learning arises by collecting experiences continuously (across different tasks) and using these to accumulate the meta-gradient online. Warp parameters are updated intermittently with the accumulated meta-gradient. We use this algorithm in our maze navigation experiment, where task adaptation is internalised within the RNN task-learner.

## 6.C WarpGrad Optimisers

In this Section, we detail WarpGrad methods used in our experiments.

**Warp-MAML** We use this model for few-shot learning (Section 6.4.1), under with the full warp-objective in Eq. 6.11 and the MAML objective (Eq. 6.1),

$$\mathcal{J}^{\text{Warp-MAML}} := L(\phi) + \lambda C^{\text{MAML}}(\theta_0), \quad (6.14)$$

where  $C^{\text{MAML}}$  is the original MAML (Eq. 6.2). In our experiments, we trained Warp-MAML using the online training algorithm (Algorithm 6.1).

**Warp-Leap** We use this model for multi-shot meta-learning. It is defined by applying Leap (Flennerhag et al., 2019) to  $\theta_0$  (Eq. 6.16),

$$\mathcal{J}^{\text{Warp-Leap}} := L(\phi) + \lambda C^{\text{Leap}}(\theta_0), \quad (6.15)$$

where the Leap objective is defined by minimising the expected cumulative chordal distance,

$$C^{\text{Leap}}(\theta_0) := \sum_{\tau \sim p(\tau)} \sum_{k=1}^{K_\tau} \|\text{sg}[\vartheta_k^\tau] - \vartheta_{k-1}^\tau\|_2, \quad \vartheta_k^\tau := [\theta_k^\tau; \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi)]. \quad (6.16)$$

Note that the Leap meta-gradient makes a first-order approximation to avoid backpropagating through the adaptation process. It is given by

$$\nabla C^{\text{Leap}}(\theta_0) \approx - \sum_{\tau \sim p(\tau)} \sum_{k=1}^{K_\tau} \frac{\Delta \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi) \nabla \mathcal{L}_{\text{task}}^\tau(\theta_{k-1}^\tau; \phi) + \Delta \theta_k^\tau}{\|\vartheta_k^\tau - \vartheta_{k-1}^\tau\|_2}, \quad (6.17)$$

where  $\Delta \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi) := \mathcal{L}_{\text{task}}^\tau(\theta_k^\tau; \phi) - \mathcal{L}_{\text{task}}^\tau(\theta_{k-1}^\tau; \phi)$  and  $\Delta \theta_k^\tau := \theta_k^\tau - \theta_{k-1}^\tau$ . In our experiments, we train Warp-Leap using Algorithm 6.1 in the multi-shot *tieredImageNet* experiment and Algorithm 6.2 in the Omniglot experiment. We perform an ablation study for training algorithms, comparing exact (Eq. 6.11) versus approximate (Eq. 6.12) meta-objectives, and several implementations of the warp-layers on Omniglot in Appendix 6.F.

**Warp-RNN** For our Reinforcement Learning experiment, we define a Warp-Grad optimiser by meta-learning an LSTM that modulates the weights of the task-learner (see Appendix 6.I for details). For this algorithm, we face a continuous stream of experiences (episodes) that we meta-learn using our continual meta-training algorithm (Algorithm 6.3). In our experiment, both  $\mathcal{L}_{\text{task}}^\tau$  and  $\mathcal{L}_{\text{meta}}^\tau$  are the advantage actor-critic objective (Wang et al., 2016a);  $C$  is computed on one batch of 30 episodes, whereas  $L$  is accumulated over  $\eta = 30$  such batches, for a total of 900 episodes. As each episode involves 300 steps in the environment, we cannot apply the exact meta objective, but use the approximate meta objective (Eq. 6.12). Specifically, let  $E^\tau = \{s_0, a_1, r_1, s_1, \dots, s_T, a_T, r_T, s_{T+1}\}$  denote an episode on task  $\tau$ , where  $s$  denotes state,  $a$  action, and  $r$  instantaneous reward. Denote a mini-batch of randomly sampled task episodes by  $E = \{E^\tau\}_{\tau \sim p(\tau)}$  and an ordered set of  $k$  consecutive mini-batches by  $\mathcal{E}^k = \{E_{k-i}^\tau\}_{i=0}^{k-1}$ . Then

$$\hat{L}(\phi; \mathcal{E}^k) = \frac{1}{n} \sum_{E_i \in \mathcal{E}^k} \sum_{E_{i,j}^\tau \in E_i} \mathcal{L}_{\text{meta}}^\tau(\phi; \theta, E_{i,j}^\tau) \quad (6.18)$$

and

$$C^{\text{multi}}(\theta; E_k) = \frac{1}{n'} \sum_{E_{k,j}^\tau \in E_k} \mathcal{L}_{\text{task}}^\tau(\theta; \phi, E_{k,j}^\tau), \quad (6.19)$$

where  $n, n'$  are normalising constants. We define the Warp-RNN objective

$$\mathcal{J}^{\text{Warp-RNN}} := \begin{cases} \hat{L}(\phi; \mathcal{E}^k) + \lambda C^{\text{multi}}(\theta; E_k) & \text{if } k = \eta \\ \lambda C^{\text{multi}}(\theta; E_k) & \text{otherwise.} \end{cases} \quad (6.20)$$

**WarpGrad for Continual Learning** For this experiment, we focus on meta-learning warp-parameters. Hence, the initialisation for each task sequence is a fixed random initialisation, (i.e.  $\lambda C(\theta^0) = 0$ ). For the warp meta-objective, we take expectations over  $N$  task sequences, where each task sequence is a sequence of  $T = 5$  sub-tasks that the task-learner observes one at a time; thus while the task loss is defined over the current sub-task, the meta-loss averages of the current and all prior sub-tasks, for each sub-task in the sequence. See Appendix 6.J for detailed definitions.

Importantly, because WarpGrad defines task adaptation abstractly by a probability distribution, we can readily implement a continual learning objective by modifying the joint task parameter distribution  $p(\tau, \theta)$  that we use in the meta-objective (Eq. 6.11). A task defines a sequence of sub-tasks over which we generate parameter trajectories  $\theta^\tau$ . Thus, the only difference from multi-task meta-learning is that parameter trajectories are not generated under a fixed task, but arise as a function of the continual learning algorithm used for adaptation. We define the conditional distribution  $p(\theta | \tau)$  as before by sampling sub-task parameters  $\theta^{\tau_t}$  from a mini-batch of such task trajectories, keeping track of which sub-task  $t$  it belongs to and which sub-tasks came before it in the given task sequence  $\tau$ . The meta-objective is constructed, for any sub-task parameterisation  $\theta^{\tau_t}$ , as  $\mathcal{L}_{\text{meta}}^\tau(\theta^{\tau_t}) = \frac{1}{t} \sum_{s=1}^t \mathcal{L}_{\text{task}}^\tau(\theta^{\tau_s}, \mathcal{D}_s; \phi)$ , where  $\mathcal{D}_s$  is data from sub-task  $s$  (Appendix 6.J). The meta-objective is an expectation over task parameterisations,

$$L^{\text{CL}}(\phi) := \sum_{\tau \sim p(\tau)} \sum_{t=1}^T \sum_{\theta^{\tau_t} \sim p(\theta | \tau_t)} \mathcal{L}_{\text{meta}}^\tau(\theta^{\tau_t}; \phi). \quad (6.21)$$

## 6.D Synthetic Experiment

To build intuition for what it means to warp space, we construct a simple 2-D problem over loss surfaces. A learner is faced with the task of minimising an objective function of the form

$$f^\tau(x_1, x_2) = g_1^\tau(x_1) \exp(g_2^\tau(x_2)) - g_3^\tau(x_1) \exp(g_4^\tau(x_1, x_2)) - g_5^\tau \exp(g_6^\tau(x_1)),$$

where each function  $f^\tau$  is defined by scale and rotation functions  $g^\tau$  that are randomly sampled from a predefined distribution. Specifically, each task is defined by the objective function

$$\begin{aligned} f^\tau(x_1, x_2) = & b_1^\tau (a_1^\tau - x_1)^2 \exp(-x_1^2 - (x_2 + a_2^\tau)^2) \\ & - b_2^\tau (x_1/s^\tau - x_1^3 - x_2^5) \exp(-x_1^2 - x_2^2) \\ & - b_3^\tau \exp(-(x_1 + a_3^\tau)^2 - x_1^2), \end{aligned} \quad (6.22)$$

where each  $a, b$  and  $s$  are randomly sampled parameters from

$$\begin{aligned} s^\tau &\sim \text{Cat}(1, 2, \dots, 9, 10) \\ a_i^\tau &\sim \text{Cat}(-1, 0, 1) \\ b_i^\tau &\sim \text{Cat}(-5, -4, \dots, 4, 5). \end{aligned} \quad (6.23)$$

The task is to minimise the given objective from a randomly sampled initialisation,  $x_{\{i=1,2\}} \sim U(-3, 3)$ . During meta-training, we train on a task for 100 steps using a learning rate of 0.1. Each task has a unique loss-surface that the learner traverses from the randomly sampled initialisation. While each loss-surface is unique, they share an underlying structure. Thus, by meta-learning to warp loss surfaces across the task distribution, we expect WarpGrad to learn a warped surface that is close to invariant to spurious descent directions. In particular, WarpGrad should produce a smooth warped space that is quasi-convex for any given task to ensure that the task-learner finds a minimum as fast as possible regardless of initialisation.

To visualise the geometry, we use an explicit warp  $\Omega$  defined by a 2-layer feed-forward network with a hidden-state size of 30 and tanh non-linearities. We train warp parameters for 100 meta-training steps; in each meta-step we sample a new task surface and a mini-batch of 10 random initialisations that we train separately. We train to convergence and accumulate the warp meta-gradient online (Algorithm 6.1). We evaluate against gradient descent on randomly sampled loss surfaces (Figure 6.8). Both optimisers start from the same initialisation, chosen such that standard gradient descent struggles;



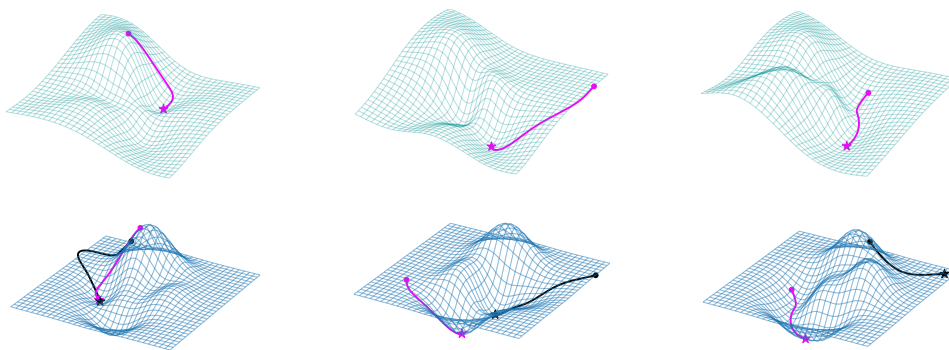


Figure 6.8: Example trajectories on three task loss surfaces. We start Gradient Descent (black) and WarpGrad (magenta) from the same initialisation; while SGD struggles with the curvature, the WarpGrad optimiser has learned a warp such that gradient descent in the representation space (top) leads to rapid convergence in model parameter space (bottom).

we expect the WarpGrad optimisers to learn a geometry that is robust to the initialisation (top row). This is indeed what we find; the geometry learned by WarpGrad smoothly warps the native loss surface into a well-behaved space where gradient descent converges to a local minimum.

## 6.E Omniglot

We follow the protocol of [Flennerhag et al. \(2019\)](#), including the choice of hyper-parameters unless otherwise noted. In this setup, each of the 50 alphabets that comprise the dataset constitutes a distinct task. Each task is treated as a 20-way classification problem. Four alphabets have fewer than 20 characters in the alphabet and are discarded, leaving us with 46 alphabets in total. 10 alphabets are held-out for final meta-testing; which alphabets are held out depend on the seed to account for variations across alphabets; we train and evaluate all baselines on 10 seeds. For each character in an alphabet, there are 20 raw samples. Of these, 5 are held out for final evaluation on the task while the remainder is used to construct a training set. Raw samples are pre-processed by random affine transformations in the form of (a) scaling between  $[0.8, 1.2]$ , (b) rotation  $[0, 360]$ , and (c) cropping height and width by a factor of  $[-0.2, 0.2]$  in each dimension. This ensures tasks are too hard for few-shot learning. During task adaptation, mini-batches are sampled at random without ensuring class-balance (in contrast to few-shot classification protocols ([Vinyals et al., 2016](#))). Note that benchmarks under this protocol are not compatible with few-shot learning benchmarks.

We use the same convolutional neural network architecture and hyper-parameters

Table 6.2: Mean test error after 100 training steps on held out evaluation tasks. <sup>†</sup>Multi-headed. <sup>‡</sup>No meta-training, but 10x larger learning rates. # refers to number of meta-training tasks.

#	Warp-Leap	Leap	Reptile	FT <sup>†</sup>	MAML	KFAC <sup>‡</sup>	SGD <sup>‡</sup>
1	49.5 ± 7.8	37.6 ± 4.8	40.4 ± 4.0	53.8 ± 5.0	40.0 ± 2.6	<b>56.0</b>	51.0
3	<b>68.8</b> ± 2.8	53.4 ± 3.1	53.1 ± 4.2	64.6 ± 3.3	48.6 ± 2.5	56.0	51.0
5	<b>75.0</b> ± 3.6	59.5 ± 3.7	58.3 ± 3.3	67.7 ± 2.8	51.6 ± 3.8	56.0	51.0
10	<b>81.2</b> ± 2.4	67.4 ± 2.4	65.0 ± 2.1	71.3 ± 2.0	54.1 ± 2.8	56.0	51.0
15	<b>82.7</b> ± 3.3	70.0 ± 2.4	66.6 ± 2.9	73.5 ± 2.4	54.8 ± 3.4	56.0	51.0
20	<b>82.0</b> ± 2.6	73.3 ± 2.3	69.4 ± 3.4	75.4 ± 3.2	56.6 ± 2.0	56.0	51.0
25	<b>83.8</b> ± 1.9	74.8 ± 2.7	70.8 ± 1.9	76.4 ± 2.2	56.7 ± 2.1	56.0	51.0

as in [Flennerhag et al. \(2019\)](#). This learner stacks a convolutional block comprised of a  $3 \times 3$  convolution with 64 filters, followed by  $2 \times 2$  max-pooling, batch-normalisation, and ReLU activation, four times. All images are down-sampled to  $28 \times 28$ , resulting in a  $1 \times 1 \times 64$  feature map that is passed on to a final linear layer. We create a Warp Leap meta-learner that inserts warp-layers between each convolutional block,  $W \circ \omega^{(4)} \circ f^{(4)} \circ \dots \circ \omega^{(1)} \circ f^{(1)}$ , where each  $f^{(i)}$  is defined as above. In our main experiment, each  $\omega^{(i)}$  is simply a  $3 \times 3$  convolutional layer with zero padding; in [Appendix 6.F](#) we consider both simpler and more sophisticated versions. We find that relatively simple warp-layers do quite well. However, adding capacity does improve generalisation performance. We meta-learn the initialisation of task parameters using the Leap objective (Eq. 6.16), detailed in [Appendix 6.C](#).

Both  $\mathcal{L}_{\text{meta}}^{\tau}$  and  $\mathcal{L}_{\text{task}}^{\tau}$  are defined as the negative log-likelihood loss; importantly, we evaluate them on *different* batches of task data to ensure warp-layers encourage generalisation. We found no additional benefit in this experiment from using held-out data to evaluate  $\mathcal{L}_{\text{meta}}^{\tau}$ . We use the offline meta-training algorithm ([Appendix 6.B](#), [Algorithm 6.2](#)); during meta-training, we sample mini-batches of 20 tasks and train task-learners for 100 steps to collect 2000 task parameterisations into a replay buffer. Task-learners share a common initialisation and warp parameters that are held fixed during task adaptation.

Once collected, we iterate over the buffer by randomly sampling mini-batches of task parameterisations without replacement. Unless otherwise noted, we used a batch size of  $\eta = 1$ . For each mini-batch, we update  $\phi$  by applying gradient descent under the canonical meta-objective (Eq. 6.11), where we evaluate  $\mathcal{L}_{\text{meta}}^{\tau}$  on a randomly sampled mini-batch of data from the corresponding task. Consequently, for each meta-batch, we take (up to) 2000 meta-gradient steps

on warp parameters  $\phi$ . This form of mini-batching causes the meta-training loop to converge much faster and induces no discernible instability.

We compare Warp-Leap against no meta-learning with SGD or KFAC (Martens & Grosse, 2015). We also benchmark against baselines provided in Flennerhag et al. (2019); Leap, Reptile (Nichol et al., 2018), MAML, and multi-headed fine-tuning. All learners benefit substantially from large batch sizes as this enables higher learning rates. To render no-pretraining a competitive option within a fair computational budget, we allow SGD and KFAC to use 4x larger batch sizes, enabling 10x larger learning rates.

Table 6.3: Ablation study on WarpGrad hyper-parameters. Mean test error after 100 training steps on held out evaluation tasks. Mean and standard deviation over 4 independent runs. *Offline* refers to offline meta-training (Appendix 6.B), *online* to online meta-training Algorithm 6.1; *full* denotes Eq. 6.11 and *approx* denotes Eq. 6.12; *full*,  $\alpha$  denotes full with a meta-learned  $\alpha$ ; <sup>†</sup>Batch Normalization (Ioffe & Szegedy, 2015); <sup>‡</sup>equivalent to FiLM layers (Perez et al., 2018); <sup>§</sup>Residual connection (He et al., 2016), when combined with BN, similar to the Residual Adaptor architecture (Rebuffi et al., 2017); <sup>¶</sup>FiLM task embeddings.

Architecture	Algorithm	Objective	Accuracy
None (Leap)	Online	None	74.8 $\pm$ 2.7
3 $\times$ 3 conv (default)	Offline	full	84.4 $\pm$ 1.7
3 $\times$ 3 conv	Offline	approx	83.1 $\pm$ 2.7
3 $\times$ 3 conv	Online	full	76.3 $\pm$ 2.1
3 $\times$ 3 conv	Offline	full, $\alpha$	83.1 $\pm$ 3.3
Scaling <sup>‡</sup>	Offline	full	77.5 $\pm$ 1.8
1 $\times$ 1 conv	Offline	full	79.4 $\pm$ 2.2
3 $\times$ 3 conv + ReLU	Offline	full	83.4 $\pm$ 1.6
3 $\times$ 3 conv + BN <sup>†</sup>	Offline	full	84.7 $\pm$ 1.7
3 $\times$ 3 conv + BN <sup>†</sup> + ReLU	Offline	full	85.0 $\pm$ 0.9
3 $\times$ 3 conv + BN <sup>†</sup> + Res <sup>§</sup> + ReLU	Offline	full	86.3 $\pm$ 1.1
2-layer 3 $\times$ 3 conv + BN <sup>†</sup> + Res <sup>§</sup>	Offline	full	<b>88.0</b> $\pm$ 1.0
2-layer 3 $\times$ 3 conv + BN <sup>†</sup> + Res <sup>§</sup> + TA <sup>¶</sup>	Offline	full	<b>88.1</b> $\pm$ 1.0

## 6.F Ablation Study: Layers, Objectives, Algorithms

WarpGrad provides a principled approach for model-informed meta-learning and offers several degrees of freedom. To evaluate these design choices, we conduct an ablation study on Warp-Leap where we vary the design of warp-layers as well as meta-training approaches (Table 6.3). For the ablation study, we fixed the number of pretraining tasks to 25 and report final test accuracy

over 4 independent runs. All ablations use the same hyper-parameters, except for online meta-training which uses a learning rate of 0.001.

First, we vary meta-training by (a) using the approximate objective (Eq. 6.12), (b) using online meta-training (Algorithm 6.1), and (c) whether meta-learning the learning rate used for task adaptation is beneficial in this experiment. We meta-learn a single scalar learning rate (as warp parameters can learn layer-wise scaling). Meta-gradients for the learning rate are clipped at 0.001 and we use a learning rate of 0.001. Note that when using offline meta-training, we store both task parameterisations and the momentum buffer in that phase and use them in the update rule when computing the canonical objective (Eq. 6.11).

Further, we vary the architecture used for warp-layers. We study simpler versions that use channel-wise scaling and more complex versions that use non-linearities and residual connections. We also evaluate a version where each warp-layer has two stacked convolutions, where the first warp convolution outputs 128 filters and the second warp convolution outputs 64 filters. Finally, in the two-layer warp-architecture, we evaluate a version that inserts a FiLM layer between the two warp convolutions. These are adapted during task training from a 0 initialisation; they amount to task embeddings that condition gradient warping on task statistics. Full results are reported in Table 6.3.

## 6.G Ablation study: Warped and Natural Gradients

Here, we perform ablation studies to compare the geometry that a WarpGrad optimiser learns to the geometry that Natural Gradient Descent (NGD) methods represent (approximately). For consistency, we run the ablation on Omniglot. As computing the true Fisher Information Matrix is intractable, we can compare WarpGrad against two common block-diagonal approximations, KFAC (Martens & Grosse, 2015) and Natural Neural Nets (Desjardins et al., 2015).

First, we isolate the effect of warping task loss surfaces by fixing a random initialisation and only meta-learning warp parameters. That is, in this experiment, we set  $\lambda C(\theta^0) = 0$ . We compare against two baselines, stochastic gradient descent (SGD) and KFAC, both trained from a random initialisation. We use task mini-batch sizes of 200 and task learning rates of 1.0, otherwise we use the same hyper-parameters as in the main experiment. For WarpGrad, we meta-train with these hyper-parameters as well.

We evaluate two WarpGrad architectures, in one, we use linear warp-layers, which gives a block-diagonal preconditioning, as in KFAC. In the other, we

Table 6.4: Ablation study: mean test error after 100 training steps on held out evaluation tasks from a random initialisation. Mean and standard deviation over 4 seeds.<sup>†</sup>Preconditioning acts only on gradients in the backward pass.<sup>‡</sup>Preconditioning acts both in the forward and backward-pass.

Method	Initialisation	Preconditioning	Action	Accuracy
SGD	random	none	$\mathbf{b}^\dagger$	$40.1 \pm 6.1$
KFAC (NGD)	random	linear (block-diagonal)	$\mathbf{b}^\dagger$	$58.2 \pm 3.2$
WarpGrad	random	linear (block-diagonal)	$\mathbf{f}^\ddagger/\mathbf{b}^\dagger$	$68.0 \pm 4.4$
WarpGrad	random	non-linear (full)	$\mathbf{f}^\ddagger/\mathbf{b}^\dagger$	$81.3 \pm 4.0$

use our most expressive warp configuration from the ablation experiment in Appendix 6.F, where warp-layers are two-layer convolutional block with residual connections, batch normalisation, and ReLU activation. We find that warped geometries facilitate task adaptation on held-out tasks to a greater degree than either SGD or KFAC by a significant margin (Table 6.4). We further find that going beyond block-diagonal preconditioning yields a significant improvement in performance.

Second, we explore whether the geometry that we meta-learn under in the full Warp-Leap algorithm is approximately Fisher. In this experiment, we use the main Warp-Leap architecture. We use a meta-learner trained on 25 tasks and that we evaluate on 10 held-out tasks. Because warp-layers are linear in this configuration, if the learned geometry is approximately Fisher, post-warp activations should be zero-centred and the layer-wise covariance matrix should satisfy  $\text{Cov}(\omega^{(i)}(f^{(i)}(x)), \omega^{(i)}(f^{(i)}(x))) = I$ , where  $I$  is the identity matrix (Desjardins et al., 2015). If true, Warp-Leap would learn a block-diagonal approximation to the Inverse Fisher Matrix, as Natural Neural Nets.

To test this, during task adaptation on held-out tasks, we compute the mean activation in each convolutional layer pre- and post-warping. We also compute the Shatten-1 norm of the difference between layer activation covariance and the identity matrix pre- and post-warping, as described above. We average statistics over task and adaptation step (we found no significant variation in these dimensions).

Figure 6.10 summarise our results. We find that, in general, WarpGrad-Leap has zero-centered post-warp activations. That pre-warp activations are positive is an artefact of the ReLU activation function. However, we find that the correlation structure is significantly different from what we would expect if Warp-Leap were to represent the Fisher matrix; post-warp covariances are significantly dissimilar from the identity matrix and varies across layers.

These results indicate that WarpGrad methods behave distinctly different from Natural Gradient Descent methods. One possibility is that WarpGrad methods do approximate the Fisher Information Matrix, but with higher accuracy than other methods. A more likely explanation is that WarpGrad methods encode a different geometry since they can learn to leverage global information beyond the task at hand, which enables them to express geometries that standard Natural Gradient Descent cannot.

## 6.H *miniImageNet* and *tieredImageNet*

***miniImageNet*** This dataset is a subset of 100 classes sampled randomly from the 1000 base classes in the ILSVRC-12 training set, with 600 images for each class. Following (Ravi & Larochelle, 2017), classes are split into non-overlapping meta-training, meta-validation and meta-tests sets with 64, 16, and 20 classes in each respectively.

***tieredImageNet*** As described in (Ren et al., 2018), this dataset is a subset of ILSVRC-12 that stratifies 608 classes into 34 higher-level categories in the ImageNet human-curated hierarchy (Deng et al., 2009). In order to increase the separation between meta-train and meta-evaluation splits, 20 of these categories are used for meta-training, while 6 and 8 are used for meta-validation and meta-testing respectively. Slicing the class hierarchy closer to the root creates more similarity within each split, and correspondingly more diversity between splits, rendering the meta-learning problem more challenging. High-level categories are further divided into 351 classes used for meta-training, 97 for meta-validation and 160 for meta-testing, for a total of 608 base categories. All the training images in ILSVRC-12 for these base classes are used to generate problem instances for *tieredImageNet*, of which there are a minimum of 732 and a maximum of 1300 images per class.

For all experiments,  $N$ -way  $K$ -shot classification problem instances are sampled following the standard image classification protocol for meta-learning (Vinyals et al., 2016). A subset of  $N$  classes was sampled at random from the corresponding split. For each class,  $K$  arbitrary images were chosen without replacement to form the training dataset of that problem instance. As usual, a disjoint set of  $L$  images per class were selected for the validation set.

**Few-shot classification** We use established experimental protocols for evaluation in meta-validation and meta-testing: 600 task instances were selected, all using  $N = 5$ ,  $K = 1$  or  $K = 5$ , as specified, and  $L = 15$ . During meta-training



we used  $N = 5$ ,  $K = 5$  or  $K = 15$  respectively, and  $L = 15$ .

Task-learners used 4 convolutional blocks defined by with 128 filters (or less, chosen by hyper-parameter tuning),  $3 \times 3$  kernels and strides set to 1, followed by batch normalisation with learned scales and offsets, a ReLU non-linearity and  $2 \times 2$  max-pooling. The output of the convolutional stack ( $5 \times 5 \times 128$ ) was flattened and mapped, using a linear layer, to the 5 output units. The last 3 convolutional layers were followed by warp-layers with 128 filters each. Only the final 3 task-layer parameters and their corresponding scale and offset batch-norm parameters were adapted during task-training, with the corresponding warp-layers and the initial convolutional layer kept fixed and meta-learned using the WarpGrad objective. Note that, with the exception of CAVIA, other baselines do worse with 128 filters as they overfit; MAML and T-Nets achieve 46% and 49 % 5-way-1-shot test accuracy with 128 filters, compared to their best reported results (48.7% and 51.7%, respectively).

Hyper-parameters were tuned independently for each condition using random grid search for highest test accuracy on meta-validation left-out tasks. Grid sizes were 50 for all experiments. We choose the optimal hyper-parameters (using early stopping at the meta-level) in terms of meta-validation test set accuracy for each condition and we report test accuracy on the meta-test set of tasks. 60000 meta-training steps were performed using meta-gradients over a single randomly selected task instances and their entire trajectories of 5 adaptation steps. We use SGD without momentum for task adaptation and Adam (Kingma & Ba, 2015) for updates to meta-parameters.

**Multi-shot classification** We use  $N = 10$ ,  $K = 640$  and  $L = 50$ . Task-learners are defined similarly as above, but stack 6 convolutional blocks defined by  $3 \times 3$  kernels and strides set to 1, followed by batch normalisation with learned scales and offsets, a ReLU non-linearity and  $2 \times 2$  max-pooling (first 5 layers). The sizes of convolutional layers were chosen by hyper-parameter tuning to  $\{64, 64, 160, 160, 256, 256\}$ . The output of the convolutional stack ( $2 \times 2 \times 256$ ) was flattened and mapped, using a linear layer, to the 10 output units.

Hyper-parameters were tuned independently for each algorithm, version, and baseline using random grid search for highest test accuracy on meta-validation left-out tasks. Grid sizes were 200 for all multi-shot experiments. We choose the optimal hyper-parameters in terms of mean meta-validation test set accuracy AUC (using early stopping at the meta-level) for each condition and we report test accuracy on the meta-test set of tasks. 2000 meta-training steps were performed using averaged meta-gradients over 5 random task instances and

their entire trajectories of 100 adaptation steps with batch size 64, or inner-loops. Task-specific adaptation was done using stochastic gradient descent with momentum (0.9). Meta-gradients were passed to Adam in the outer loop.

We test WarpGrad against Leap, Reptile, and training from scratch with large batches and tuned momentum. We tune all meta-learners for optimal performance on the validation set. WarpGrad outperforms all baselines both in terms of rate of convergence and final test performance (Figure 6.11).

## 6.1 Maze Navigation

To illustrate both how WarpGrad may be used with RNNs in an online meta-learning setting, as well as in an RL environment, we evaluate it in a maze navigation task proposed by [Miconi et al. \(2018\)](#). The environment is a fixed maze and a task is defined by randomly choosing a goal location in the maze. During a task episode of length 200, the goal location is fixed but the agent gets teleported once it finds it. Thus, during an episode the agent must first locate the goal, then return to it as many times as possible, each time being randomly teleported to a new starting location. We use an identical setup as [Miconi et al. \(2019\)](#), except our grid is of size  $11 \times 11$  as opposed to  $9 \times 9$ . We compare our Warp-RNN to Learning to Reinforcement Learn ([Wang et al., 2016a](#)) and Hebbian meta-learners ([Miconi et al., 2018, 2019](#)).

The task-learner in all cases is an advantage actor-critic ([Wang et al., 2016a](#)), where the actor and critic share an underlying basic RNN, whose hidden state is projected into a policy and value function by two separate linear layers. The RNN has a hidden state size of 100 and tanh non-linearities. Following ([Miconi et al., 2019](#)), for all benchmarks, we train the task-learner using Adam with a learning rate of  $1e-3$  for 200 000 steps using batches of 30 episodes, each of length 200. Meta-learning arises in this setting as each episode encodes a different task, as the goal location moves, and by learning across episodes the RNN is encoding meta-information in its parameters that it can leverage during task adaptation (via its hidden state ([Hochreiter & Schmidhuber, 1997](#); [Wang et al., 2016a](#))). See [Miconi et al. \(2019\)](#) for further details.

We design a Warp-RNN by introducing a warp-layer in the form of an LSTM that is frozen for most of the training process (c.f. Figure 6.6, Appendix 6.A). Following [Flennerhag et al. \(2018\)](#), we use this meta-LSTM to modulate the



task RNN. Given an input vector  $x_t$ , the task RNN is defined by

$$h_t = \tanh \left( U_{h,t}^{(2)} V U_{h,t}^{(1)} h_{t-1} + U_{x,t}^{(2)} W U_{x,t}^{(1)} x_t + U_t^{(b)} b \right), \quad (6.24)$$

where  $W$ ,  $V$ ,  $b$  are task-adaptable parameters; each  $U_{j,t}^{(i)}$  is a diagonal warp matrix produced by projecting from the hidden state of the meta-LSTM,  $U_{j,t}^{(i)} = \text{diag}(\tanh(P_j^{(i)} z_t))$ , where  $z$  is the hidden-state of the meta-LSTM. For details, see [Flennerhag et al. \(2018\)](#). Because the meta-LSTM is frozen for most of the training process, task adaptable parameters correspond to those of the baseline RNN.

To control for the capacity of the meta-LSTM, we train a HyperRNN where the LSTM is updated with every task adaptation; we find this model does worse than the WarpGrad-RNN. We also compare the non-linear preconditioning that we obtain in our Warp-RNN to linear forms of preconditioning defined in prior works. We implement a T-Nets-RNN meta-learner by embedding linear projections  $T_h$ ,  $T_x$  and  $T_b$  that are meta-learned in the task RNN,  $h_t = \tanh(T_h V h_t + T_x W x_t + b)$ . We cannot backpropagate to these meta-parameters as per the T-Nets (MAML) framework; instead, we train  $T_h, T_x, T_b$  with the meta-objective and meta-training algorithm we use for the Warp-RNN. The T-Nets-RNN does worse than the baseline RNN and generally fails to learn.

We meta-train the Warp-RNN using the continual meta-training algorithm (Algorithm 6.3, see Appendix 6.B for details), which accumulates meta-gradients continuously during training. Because task training is a continuous stream of batches of episodes, we accumulating the meta-gradient using the approximate objective (Eq. 6.12, where  $\mathcal{L}_{\text{task}}^\tau$  and  $\mathcal{L}_{\text{meta}}^\tau$  are both the same advantage actor-critic objective) and update warp-parameters on every 30th task parameter update. We detail the meta-objective in Appendix 6.C (see Eq. 6.20). Our implementation of a Warp-RNN can be seen as meta-learning “slow” weights to facilitate learning of “fast” weights ([Schmidhuber, 1992](#); [Mujika et al., 2017](#)). Implementing Warp-RNN requires four lines of code on top of the standard training script. The task-learner is the same in all experiments with the same number of learnable parameters and hidden state size. Compared to all baselines, we find that the Warp-RNN converges faster and achieves a higher cumulative reward (Figure 6.4 and Figure 6.12).

## 6.J Meta-Learning for Continual Learning

Online SGD and related optimisation methods tend to adapt neural network models to the data distribution encountered last during training, usually leading to what has been termed “catastrophic forgetting” (French, 1999). We investigate whether WarpGrad optimisers can meta-learn to avoid this problem altogether and directly minimise the joint objective over all tasks with every update in the fully online learning setting where no past data is retained.

**Continual Sine Regression** We propose a continual learning version of the sine regression meta-learning experiment in Finn et al. (2017). We split the input interval  $[-5, 5] \subset \mathbb{R}$  evenly into 5 consecutive sub-intervals, corresponding to 5 regression tasks. These are presented one at a time to a task-learner, which adapts to each sub-task using 20 gradient steps on data from the given sub-task only. Batch sizes were set to 5 samples. Sub-tasks thus differ in their input domain. A task sequence is defined by a target function composed of two randomly mixed sine functions of the form  $f_{a_i, b_i}(x) = a_i \sin(x - b_i)$  each with randomly sampled amplitudes  $a_i \in [0.1, 5]$  and phases  $b_i \in [0, \pi]$ . A task  $\tau = (a_1, b_1, a_2, b_2, o)$  is therefore defined by sampling the parameters that specify this mixture; a task specifies a target function  $g^\tau$  by

$$g^\tau(x) = \varphi_o(x)g_{a_1, b_1}(x) + (1 - \varphi_o(x))g_{a_2, b_2}(x), \quad (6.25)$$

where  $\varphi_o(x) = \sigma(x + o)$  for a randomly sampled offset  $o \in [-5, 5]$ , with  $\sigma$  being the sigmoid activation function.

**Model** We define a task-learner as 4-layer feed-forward networks with hidden layer size 200 and ReLU non-linearities to learn the mapping between inputs and regression targets,  $F(\cdot, \theta, \phi)$ . For each task sequence  $\tau$ , a task-learner is initialised from a fixed random initialisation  $\theta_0$  (not meta-learned). Each non-linearity is followed by a residual warping block consisting of 2-layer feed-forward networks with 100 hidden units and tanh non-linearities, with meta-learned parameters  $\phi$  which are fixed during the task adaptation process.

**Continual learning as task adaptation** The task target function  $g^\tau$  is partitioned into 5 sets of sub-tasks. The task-learner sees one partition at a time and is given  $n = 20$  gradient steps to adapt, for a total of  $K = 100$  steps of online gradient descent updates for the full task sequence; recall that every such sequence starts from a fixed random initialisation  $\theta_0$ . The adaptation is completely online since at step  $k = 1, \dots, K$  we sample a new mini-batch  $D_{\text{task}}^k$  of

5 samples from a single sub-task (sub-interval). The data distribution changes after each  $n = 20$  steps with inputs  $x$  coming from the next sub-interval and targets form the same function  $g^\tau(x)$ . During meta-training we always present tasks in the same order, presenting intervals from left to right. The online (sub-)task loss is defined on the current mini-batch  $D_{\text{task}}^k$  at step  $k$ :

$$\mathcal{L}_{\text{task}}^\tau(\theta_k^\tau, D_{\text{task}}^k; \phi) = \frac{1}{2|D_{\text{task}}^k|} \sum_{x \in D_{\text{task}}^k} (F(x, \theta_k^\tau; \phi) - g^\tau(x))^2. \quad (6.26)$$

Adaptation to each sub-task uses sub-task data only to form task parameter updates  $\theta \leftarrow \theta - \alpha \nabla \mathcal{L}_{\text{task}}^\tau(\theta, D_{\text{task}}^k; \phi)$ . We used a constant learning rate  $\alpha = 0.001$ . Warp parameters  $\phi$  are fixed across the full task sequence during adaptation and are meta-learned across random samples of task sequences.

**Meta-learning an optimiser for continual learning** To investigate the ability of WarpGrad to learn an optimiser for continual learning that mitigates catastrophic forgetting, we fix a random initialisation prior to meta-training that is not meta-learned; every task-learner is initialised with these parameters. To meta-learn an optimiser for continual learning, we need a meta-objective that encourages such behaviour. Here, we take a first step towards a framework for meta-learned continual learning.

We define the meta-objective  $\mathcal{L}_{\text{meta}}^\tau$  as an incremental multitask objective that, for each sub-task  $\tau_t$  in a given task sequence  $\tau$ , averages the validation sub-task losses (Eq. 6.26) for the current and every preceding loss in the task sequence. The task meta-objective is defined by summing over all sub-tasks in the task sequence. For some sub-task parameterisation  $\theta^{\tau_t}$ , we have

$$\mathcal{L}_{\text{meta}}^\tau(\theta^{\tau_t}; \phi) = \sum_{i=1}^t \frac{1}{n(T-t+1)} \mathcal{L}_{\text{task}}^\tau(\theta^{\tau_i}, D_{\text{val}}^i; \phi). \quad (6.27)$$

As before, the full meta-objective is an expectation over the joint task parameter distribution (Eq. 6.11); for further details on the meta-objective, see Appendix 6.C, Eq. 6.21. This meta-objective gives equal weight to all the tasks in the sequence by averaging the regression step loss over all sub-tasks where a prior sub-task should be learned or remembered. For example, losses from the first sub-task, defined using the interval  $[-5, -3]$ , will appear  $nT$  times in the meta-objective. Conversely, the last sub-task in a sequence, defined on the interval  $[3, 5]$ , is learned only in the last  $n = 20$  steps of task adaptation, and

hence appears  $n$  times in the meta-objective. Normalising on number of appearances corrects for this bias. We trained warp-parameters using Adam and a meta-learning rate of 0.001, sampling 5 random tasks to form a meta-batch and repeating the process for 20 000 steps of meta-training.

**Results** Figure 6.13 shows a breakdown of the validation loss across the 5 sequentially learned tasks over the 100 steps of online learning during task adaptation. Results are averaged over 100 random regression problem instances. The meta-learned WarpGrad optimiser reduces the loss of the task currently being learned in each interval while also largely retaining performance on previous tasks. There is an immediate relatively minor loss of performance, after which performance on previous tasks is retained. We hypothesise that this is because the meta-objectives averages over the full learning curve, as opposed to only the performance once a task has been adapted to. As such, the WarpGrad optimiser may allow for some degree of performance loss. Intriguingly, in all cases, after an initial spike in previous sub-task losses when switching to a new task, the spike starts to revert back some way towards optimal performance, suggesting that the WarpGrad optimiser facilitates positive backward transfer, without this being explicitly enforced in the meta-objective. Deriving a principled meta-objective for continual learning is an exciting area for future research.

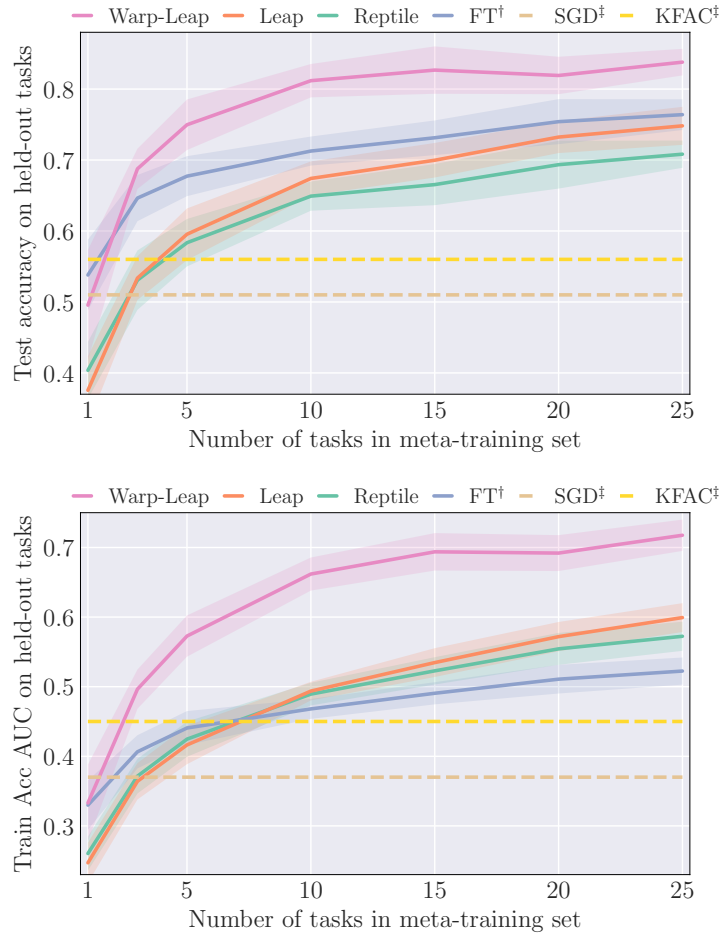


Figure 6.9: Omniglot results. *Top*: test accuracies on held-out tasks after meta-training on a varying number of tasks. *Bottom*: AUC under accuracy curve on held-out tasks after meta-training on a varying number of tasks. Shading represents standard deviation across 10 independent runs. We compare between Warp-Leap, Leap, and Reptile, multi-headed finetuning, as well as SGD and KFAC which used random initialisation but with a 10x larger learning rate.

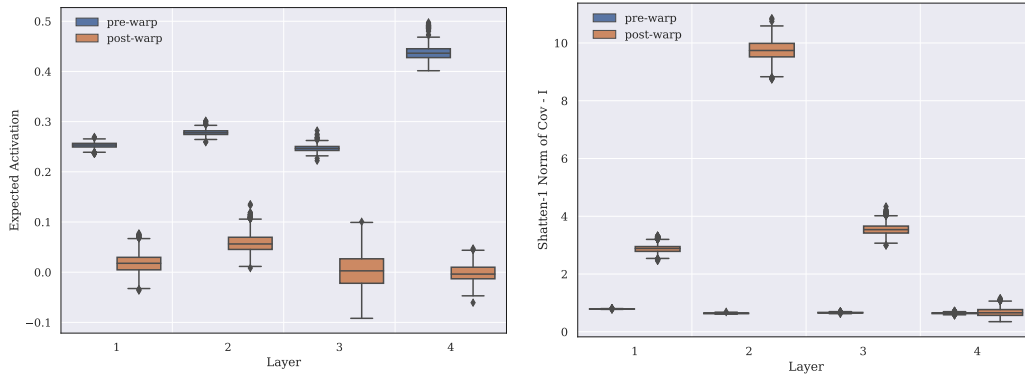


Figure 6.10: Ablation study. *Left*: Comparison of mean activation value  $\mathbb{E}[f(x)]$  across layers, pre- and post-warping. *Right*: Shatten-1 norm of  $\text{Cov}(f(x), f(x)) - I$ , pre- and post-norm. Statistics are gathered on held-out test set and averaged over tasks and adaptation steps.

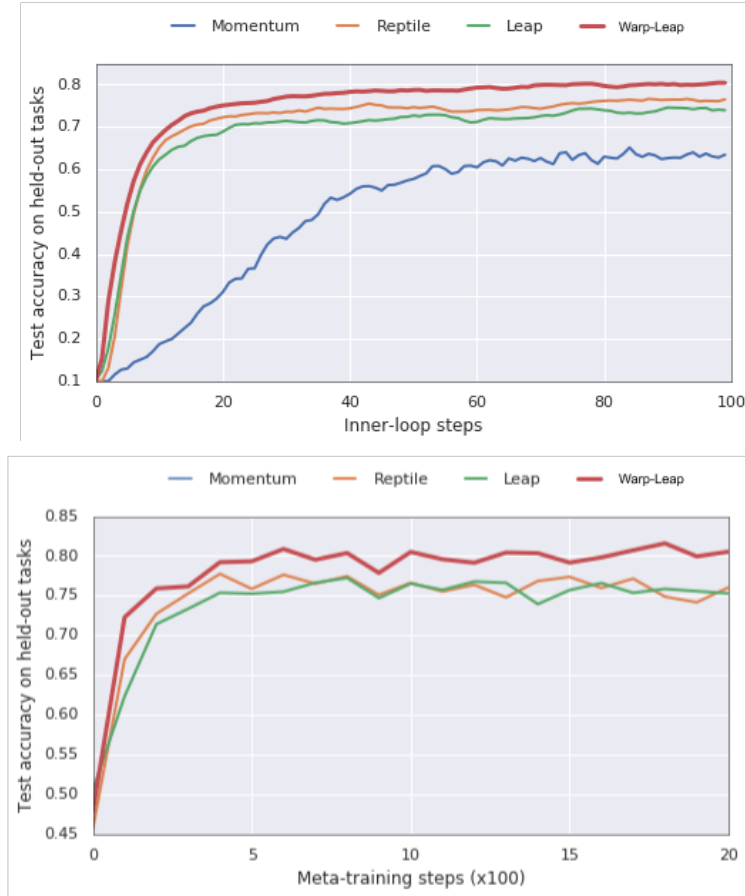


Figure 6.11: Multi-shot *tieredImageNet* results. *Top*: mean learning curves (test classification accuracy) on held-out meta-test tasks. *Bottom*: mean test classification performance on held-out meta-test tasks during meta-training. Training from scratch omitted as it is not meta-trained.

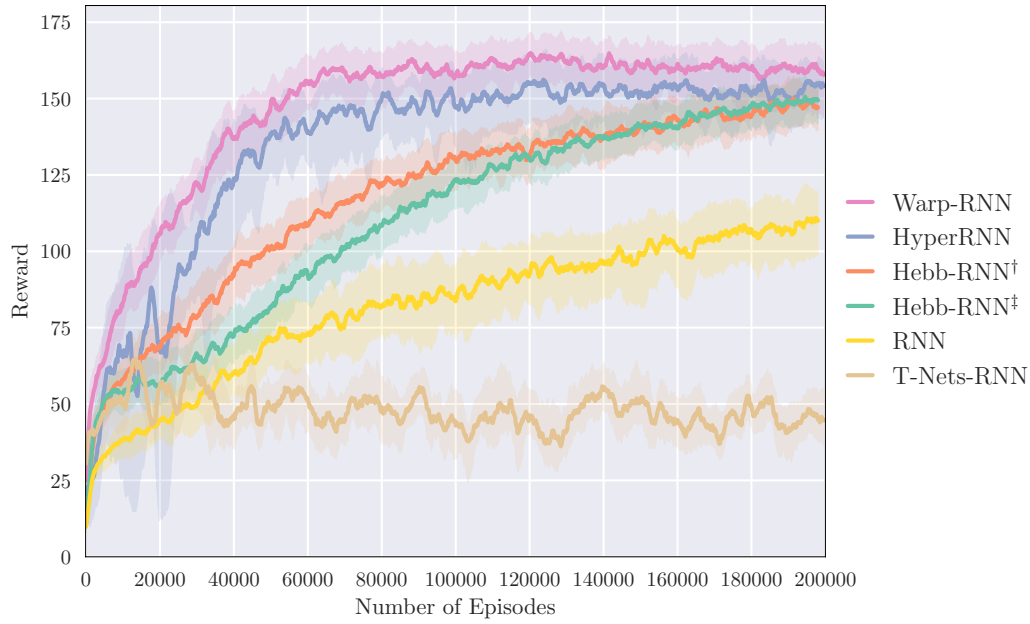


Figure 6.12: Mean cumulative return for maze navigation task, for 200 000 training steps. Shading represents inter-quartile ranges across 10 independent runs. <sup>†</sup>Simple modulation; <sup>‡</sup>retroactive modulation (Miconi et al., 2019).

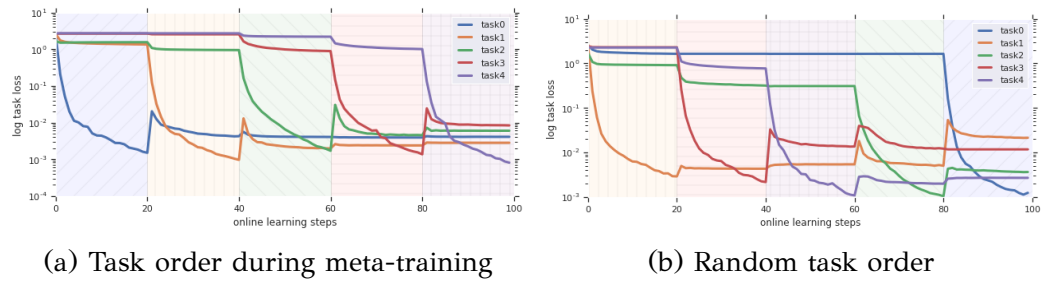


Figure 6.13: Continual learning experiment: average log-loss over 100 randomly sampled tasks. Each task contains 5 sub-tasks learned (a) sequentially as seen during meta-training or (b) in random order [sub-task 1, 3, 4, 2, 0]. We train on each sub-task for 20 steps, for a total of  $K = 100$  task adaptation steps.

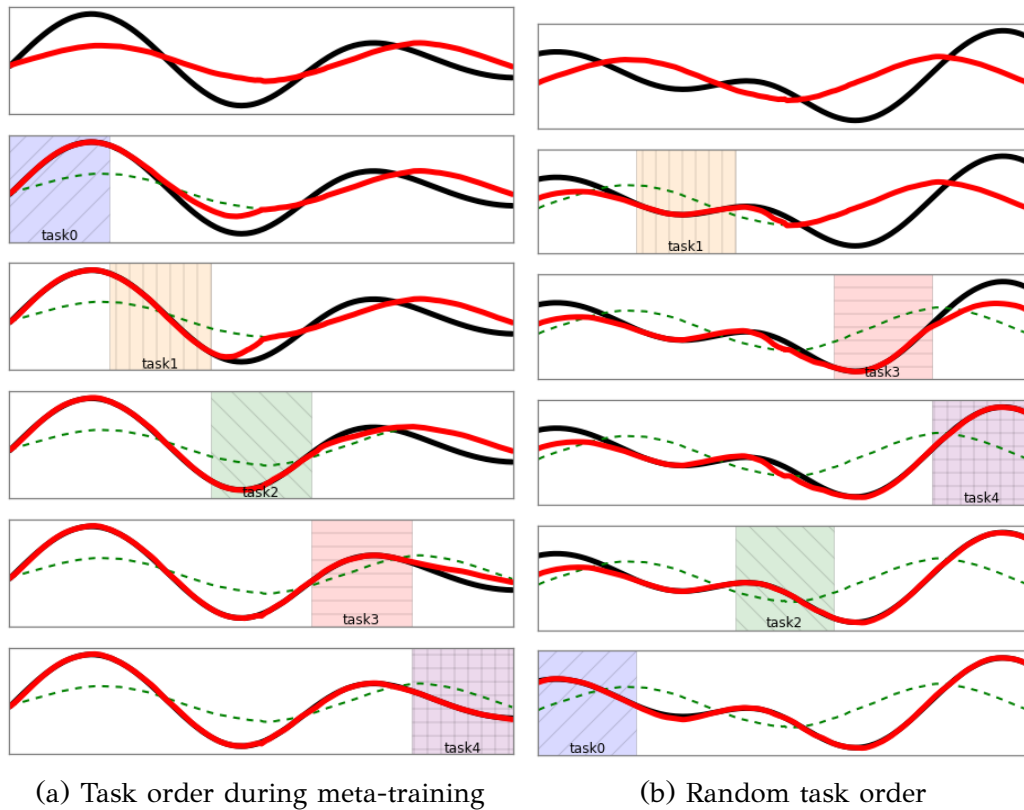


Figure 6.14: Continual learning: evaluation after partial task adaptation. Ground truth (black), task-learner prediction before adaptation (dashed green), and task-learner prediction after adaptation (red). Each row illustrates how task-learner predictions evolve (red) after training on sub-tasks up to and including that sub-task (current task illustrate in plot). (a) sub-tasks are presented in the same order as seen during meta-training; (b) sub-tasks are presented in random order at meta-test time in sub-task order [1, 3, 4, 2 and 0].



---

### III

---

## *Towards Never-Ending Learning*

# 7 Lifelong Learning in Autonomous Agents

---

This chapter tackles the assumption of an exogenous task distribution. One might argue that the best meta-learner we know—i.e. humans—are able to integrate information from a continuous stream of data without being told what constitutes a “task”. In fact, the notion of a task is intrinsic to each of us, we construct tasks to provide a meaningful demarcation of a learning problem. Ideally, an AI should be able to do learn to the same from its own experiences.

In order for an AI to do so, it must have some mechanism by which it can define some abstract notion of a task. This work proposes one such approach that relies on defining two types of behaviours in a reinforcement learning agent, formalised into two distinct policies. One policy is rewarded for collecting experience that the other policy is highly uncertain about; the other is rewarded for reducing its uncertainty on the experience it sees. In this way, each policy defines a task for the other, and their continual evolution gives rise to a form of task distribution or automatic curriculum.

While the paper focuses on the statistical properties of the proposed method as a means of reducing bias in uncertainty estimation, the connection to meta-learning is discussed from the point of view of a multi-agent game. This discussion hints at a more general point, that the task distribution in meta-learning can be generated from a multi-agent game between two players, one that proposes tasks and one that tries to solve them. There is some evidence suggesting this view might have the potential to fully overcome the need for exogenous tasks ([Sukhbaatar et al., 2018](#); [Wang et al., 2019](#); [Zheng et al., 2020](#)). Prior works tend to focus on adversarial approaches ([Sukhbaatar et al., 2018](#); [Florensa et al., 2018](#); [Kachalsky et al., 2019](#)). In this setting tasks are proposed in an adversarial way to make them as hard as possible for the learner. This can fail to meet a central criteria of meta-learning, that tasks should be useful

be optimised for *learning*. Our approach reverses the direction: our goal is to construct tasks that are efficient for learning—progressively harder tasks arises as a consequence of the agent learning. Chapter 8 discusses this point of view in the context of the contributions of this thesis.

## Temporal Difference Uncertainties as a Signal for Exploration

Under review for publication; notational modifications.

*Under Review.*

Authors: **Sebastian Flennerhag**, University of Manchester,  
Jane X. Wang, DeepMind,  
Pablo Sprechmann, DeepMind,  
Francesco Visin, DeepMind,  
Alexandre Galashov, DeepMind,  
Steven Kapturowski, DeepMind,  
Diana Borsa, DeepMind,  
Nicolas Heess, DeepMind,  
André Baretto, DeepMind,  
Razvan Pascanu, DeepMind.

*Abstract. An effective approach to exploration in reinforcement learning is to rely on an agent’s uncertainty over the optimal policy, which can yield near-optimal exploration strategies in tabular settings. However, in non-tabular settings that involve function approximators, obtaining accurate uncertainty estimates is almost as challenging as the exploration problem itself. In this paper, we highlight that value estimates are easily biased and temporally inconsistent. In light of this, we propose a novel method for estimating uncertainty over the value function that relies on inducing a distribution over temporal difference errors. This exploration signal controls for state-action transitions so as to isolate uncertainty in value that is due to uncertainty over the agent’s parameters. Because our measure of uncertainty conditions on state-action transitions, we cannot act on this measure directly. Instead, we incorporate it as an intrinsic reward and treat exploration as a separate learning problem, induced by the agent’s temporal difference uncertainties. We introduce a distinct exploration policy that learns to collect data with high estimated uncertainty, which gives rise to a “curriculum” that smoothly changes throughout learning and vanishes in the limit of perfect value estimates. We evaluate our method on hard-exploration tasks, including Deep Sea and Atari 2600 environments and find that our proposed form of exploration facilitates efficient exploration.*

---

## 7.1 Introduction

Striking the right balance between exploration and exploitation is fundamental to the reinforcement learning problem. A common approach is to derive exploration from the policy being learned. Dithering strategies, such as  $\epsilon$ -greedy exploration, render a reward-maximising policy stochastic around its reward maximising behaviour (Williams & Peng, 1991). Other methods encourage higher entropy in the policy (Ziebart et al., 2008), introduce an intrinsic reward (Singh et al., 2005), or drive exploration by sampling from the agent’s belief over the MDP (Strens, 2000).

While greedy or entropy-maximising policies cannot facilitate temporally extended exploration (Osband et al., 2013, 2016a), the efficacy of intrinsic rewards depends crucially on how they relate to the extrinsic reward that comes from the environment (Burda et al., 2018a). Typically, intrinsic rewards for exploration provide a bonus for visiting novel states (e.g Bellemare et al., 2016) or visiting states where the agent cannot predict future transitions (e.g Pathak et al., 2017; Burda et al., 2018a). Such approaches can facilitate learning an optimal policy, but they can also fail entirely in large environments as they prioritise novelty over rewards (Burda et al., 2018b).

Methods based on the agent’s uncertainty over the optimal policy explicitly trade off exploration and exploitation (Kearns & Singh, 2002). Posterior Sampling for Reinforcement Learning (PSRL; Strens, 2000; Osband et al., 2013) is one such approach, which models a distribution over Markov Decision Processes (MDPs). While PSRL is near-optimal in tabular settings (Osband et al., 2013, 2016b), it cannot be easily scaled to complex problems that require function approximators. Prior work has attempted to overcome this by instead directly estimating the agent’s uncertainty over the policy’s value function (Osband et al., 2016a; Moerland et al., 2017; Osband et al., 2019; O’Donoghue et al., 2018; Janz et al., 2019). While these approaches can scale posterior sampling to complex problems and nonlinear function approximators, estimating uncertainty over value functions introduces issues that can cause a bias in the posterior distribution (Janz et al., 2019).

In response to these challenges, we introduce *Temporal Difference Uncertainties* (TDU), which derives an intrinsic reward from the agent’s uncertainty over the value function. Concretely, TDU relies on the Bootstrapped DQN (Osband et al., 2016a) and separates exploration and reward-maximising behaviour into two separate policies that bootstrap from a shared replay buffer. This separation allows us to derive an exploration signal for the exploratory policy from esti-

mates of uncertainty of the reward-maximising policy. Thus, TDU encourages exploration to collect data with high model uncertainty over reward-maximising behaviour, which is made possible by treating exploration as a separate learning problem. In contrast to prior works that directly estimate value function uncertainty, we estimate uncertainty over *temporal difference (TD) errors*. By conditioning on observed state-action transitions, TDU controls for environment uncertainty and provides an exploration signal only insofar as there is model uncertainty. We demonstrate that TDU can facilitate efficient exploration in challenging exploration problems such as Deep Sea and Montezuma’s Revenge.

## 7.2 Estimating Value Function Uncertainty

We begin by highlighting that estimating uncertainty over the value function can suffer from bias that is very hard to overcome with typical approaches (see also Janz et al., 2019). Our analysis shows that biased estimates arise because uncertainty estimates require an integration over unknown future state visitations. This requires tremendous model capacity and is in general infeasible. Our results show that we cannot escape a bias in general, but we can take steps to mitigate it by *conditioning* on an observed trajectory. Doing so removes some uncertainty over future state-visitations and we show in Section 7.3 that it can result in a substantially smaller bias.

We consider a Markov Decision Process  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$  for some given state space  $(\mathcal{S})$ , action space  $(\mathcal{A})$ , transition dynamics  $(\mathcal{P})$ , reward function  $(\mathcal{R})$  and discount factor  $(\gamma)$ . For a given (deterministic) policy  $\pi : \mathcal{S} \mapsto \mathcal{A}$ , the action value function is defined as the expected cumulative reward under the policy starting from state  $s$  with action  $a$ :

$$Q_\pi(s, a) := \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| s_0 = s, a_0 = a \right] = \mathbb{E}_{\substack{r \sim \mathcal{R}(s,a) \\ s' \sim \mathcal{P}(s,a)}} [r + \gamma Q_\pi(s', \pi(s'))], \quad (7.1)$$

where  $t$  index time and the expectation  $\mathbb{E}_\pi$  is with respect to realised rewards  $r$  sampled under the policy  $\pi$ ; the right-hand side characterises  $Q$  recursively under the Bellman Equation. The action-value function  $Q_\pi$  is estimated under a function approximator  $Q_\theta$  parameterised by  $\theta$ . Uncertainty over  $Q_\pi$  is expressed by placing a distribution over the parameters of the function approximator,  $p(\theta)$ . We overload notation slightly and write  $p(\theta)$  to denote the probability density function  $p_\theta$  over a random variable  $\theta$ . Further, we denote by  $\theta \sim p(\theta)$  a random sample  $\theta$  from the distribution defined by  $p_\theta$ . Methods that rely on posterior sampling under function approximators assume that the

induced distribution,  $p(Q_\theta)$ , is an accurate estimate of the agent’s uncertainty over its value function,  $p(Q_\pi)$ , so that sampling  $Q_\theta \sim p(Q_\theta)$  is approximately equivalent to sampling from  $Q_\pi \sim p(Q_\pi)$ .

For this to hold, the moments of  $p(Q_\theta)$  at each state-action pair  $(s, a)$  must correspond to the expected moments in future states. In particular, moments of  $p(Q_\pi)$  must satisfy a Bellman Equation akin to Eq. 7.1 (O’Donoghue et al., 2018). We focus on the mean ( $\mathbb{E}$ ) and variance ( $\mathbb{V}$ ):

$$\mathbb{E}_\theta [Q_\theta(s, a)] = \mathbb{E}_\theta [\mathbb{E}_{r,s'} [r + \gamma Q_\theta(s', \pi(s'))]], \quad (7.2)$$

$$\mathbb{V}_\theta [Q_\theta(s, a)] = \mathbb{V}_\theta [\mathbb{E}_{r,s'} [r + \gamma Q_\theta(s', \pi(s'))]]. \quad (7.3)$$

If  $\mathbb{E}_\theta [Q_\theta]$  and  $\mathbb{V}_\theta [Q_\theta]$  fail to satisfy these conditions, the estimates of  $\mathbb{E}[Q_\pi]$  and  $\mathbb{V}[Q_\pi]$  are biased, causing a bias in exploration under posterior sampling from  $p(Q_\theta)$ . Formally, the agent’s uncertainty over  $p(Q)$  implies uncertainty over the MDP (Strens, 2000). Given a belief over the MDP, i.e., a distribution  $p(M)$ , we can associate each  $M \sim p(M)$  with a distinct value function  $Q_\pi^M$ . Lemma 7.1 below shows that, for  $p(\theta)$  to be interpreted as representing some  $p(M)$  by push-forward to  $p(Q_\theta)$ , the induced moments must match under the Bellman Equation.

**Lemma 7.1** (Bellman uncertainty bias). *If  $\mathbb{E}_\theta [Q_\theta]$  and  $\mathbb{V}_\theta [Q_\theta]$  fail to satisfy Eqs. 7.2 and 7.3, respectively, they are biased estimators of  $\mathbb{E}_M [Q_\pi^M]$  and  $\mathbb{V}_M [Q_\pi^M]$  for any choice of  $p(M)$ .*

All proofs are deferred to Appendix 7.B. Lemma 7.1 highlights why estimating uncertainty over value functions is so challenging; while the left-hand sides of Eqs. 7.2 and 7.3 are stochastic in  $\theta$  only, the right-hand sides depend on marginalising over the MDP. This requires the function approximator to generalise to unseen future trajectories. Lemma 7.1 is therefore a statement about scale; the harder it is to generalise, the more likely we are to observe a bias—even in deterministic environments.

This requirement of “strong generalisation” poses a particular problem for neural networks that tend to interpolate over the training data (e.g. Li et al., 2020; Liu et al., 2020; Belkin et al., 2019), but the issue is more general. In particular, we show that factorising the posterior  $p(\theta)$  will typically cause estimation bias for all but tabular MDPs. This is problematic because it is often computationally infeasible to maintain a full posterior; previous work either maintains a full posterior over the final layer of the function approximator

(Osband et al., 2016a; O’Donoghue et al., 2018; Janz et al., 2019) or maintains a diagonal posterior over all parameters (Fortunato et al., 2018; Plappert et al., 2018) of the neural network. Either method limits how expressive the function approximator can be with respect to future states, thereby causing an estimation bias. To establish this formally, let  $Q_\theta := w \circ \phi_\theta$ , where  $\theta = (w_1, \dots, w_n, \vartheta_1, \dots, \vartheta_v)$ , with  $w \in \mathbb{R}^n$  a linear projection and  $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^n$  a function approximator with parameters  $\vartheta \in \mathbb{R}^v$ .

**Theorem 7.1** (Function approximation bias). *If the number of state-action pairs with unique predictions  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$  and non-zero Temporal Difference error  $Q_\theta(s, a) \neq \mathbb{E}_{r, s'}[r + \gamma Q_\theta(s', \pi(s'))]$  is greater than  $n + 1$ , where  $w \in \mathbb{R}^n$ , then  $\mathbb{E}_\theta[Q_\theta]$  and  $\mathbb{V}_\theta[Q_\theta]$  are biased estimators of  $\mathbb{E}_M[Q_\pi^M]$  and  $\mathbb{V}_M[Q_\pi^M]$  for any choice of  $p(M)$ .*

This result is a consequence of the function approximator  $\phi$  mapping into a co-domain that is larger than the space spanned by  $w$ ; the bias results from function approximation errors  $\phi_\theta(s, a) - \mathbb{E}_{s', r}[r + \gamma \phi_\theta(s', \pi(s'))]$  in more state-action pairs than there are degrees of freedom in  $w$ . The implication is that function approximators under factorised posteriors cannot generalise uncertainty estimates across states (a similar observation in tabular settings was made by Janz et al., 2019)—they can only produce temporally consistent uncertainty estimates if they have the capacity to memorise point-wise uncertainty estimates for each  $(s, a)$ , which defeats the purpose of a function approximator. This is a statement about the structure of  $p(\theta)$  and holds for any estimation method. Thus, common approaches to uncertainty estimation with neural networks generally fail to provide unbiased uncertainty estimates over the value function in non-trivial MDPs. Theorem 7.1 shows that to accurately capture value function uncertainty, we need a full posterior over parameters, which is often infeasible. It also underscores that the main issue is the dependence on future state visitation. This motivates Temporal Difference Uncertainties as an estimate of uncertainty *conditioned* on observed state-action transitions.

### 7.3 Temporal Difference Uncertainties

While Theorem 7.1 states that we cannot remove this bias unless we are willing to maintain a full posterior  $p(\theta)$ , we can construct uncertainty estimates that control for uncertainty over future state-action transition. In this paper, we propose to estimate uncertainty over a full transition  $\tau := (s, a, r, s')$  to isolate uncertainty due to  $p(\theta)$ . Fixing a transition, we induce a conditional distribution  $p(\delta \mid \tau)$  over Temporal Difference (TD) errors,  $\delta(\theta, \tau) := \gamma Q_\theta(s', \pi(s')) + r -$



$Q_\theta(s, a)$ , that we characterise by its mean and variance:

$$\mathbb{E}_\delta[\delta \mid \tau] = \mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau] \quad \text{and} \quad \mathbb{V}_\delta[\delta \mid \tau] = \mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]. \quad (7.4)$$

Estimators over TD-errors is akin to first-difference estimators of uncertainty over the action-value. They can therefore exhibit smaller bias if that bias is temporally consistent. To illustrate, for simplicity assume that  $\mathbb{E}_\theta[Q_\theta]$  consistently over/under-estimates  $\mathbb{E}_M[Q_\pi^M]$  by an amount  $b \in \mathbb{R}$ . The corresponding bias in  $\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]$  is given by  $\text{Bias}(\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]) = \text{Bias}(\gamma \mathbb{E}_\theta[Q_\theta(s', \pi(s'))] + r - \mathbb{E}_\theta[Q_\theta(s, a)]) = (\gamma - 1)b$ . This bias is close to 0 for typical values of  $\gamma$ —notably for  $\gamma = 1$ ,  $\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]$  is unbiased. More generally, unless the bias is constant over time as in the above example, we cannot fully remove the bias when constructing an estimator over a quantity that relies on  $Q_\theta$ . However, as the above example shows, by conditioning on a state-action transition, we can make it significantly smaller. We formalise this logic in the following result.

**Theorem 7.2** (Temporal difference uncertainty estimation). *For any  $\tau := (s, a, r, s')$  and any  $p(M)$ , given  $p(\theta)$ , define the following ratios:*

$$\begin{aligned} \rho &= \frac{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s', \pi(s'))])}{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)])} & \phi &= \frac{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s', \pi(s'))^2])}{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)^2])} \\ \kappa &= \frac{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s', \pi(s'))Q_\theta(s, a)])}{\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)^2])} & \alpha &= \frac{\mathbb{E}_M[Q_\pi^M(s', \pi(s'))]}{\mathbb{E}_M[Q_\pi^M(s, a)]}. \end{aligned}$$

*If  $\rho \in (0, 2/\gamma)$ , then  $\mathbb{E}_\delta[\delta \mid \tau]$  has lower bias than  $\mathbb{E}_\theta[Q_\theta(s, a)]$ . Moreover, if  $\rho = 1/\gamma$ , then  $\mathbb{E}_\delta[\delta \mid \tau]$  is unbiased. If  $\rho \in (0, 2/\gamma)$ ,  $\alpha \in (0, 2/\gamma)$ ,  $\kappa \in (0, 2/\gamma)$  and  $\phi \in (1 - 2\gamma\kappa, (2/\gamma)^2)$ , then  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  have less bias than  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . In particular, if  $\rho = \phi = \kappa = \alpha = 1$ , then*

$$\begin{aligned} |\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| &= |(\gamma - 1)^2 \text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])| \\ &< |\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])|. \end{aligned} \quad (7.5)$$

*Further,  $\rho = 1/\gamma$ ,  $\kappa = 1/\gamma$ ,  $\phi = 1/\gamma^2$ , then  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  is unbiased for any  $\alpha$ .*

The first part of Theorem 7.2 generalises the example above to cases where the bias  $b$  varies across action-state transitions. It is worth noting that the required “smoothness” on the bias is not very stringent: the bias of  $\mathbb{E}_\theta[Q_\theta](s', \pi(s'))$  can be twice as large as that of  $\mathbb{E}_\theta[Q_\theta](s, a)$  and  $\mathbb{E}_\delta[\delta \mid \tau]$  can still produce a less

biased estimate. Importantly, it must have the same sign, and so Theorem 7.2 requires temporal consistency.

To establish a similar claim for  $\mathbb{V}_\delta[\delta \mid \tau]$ , we need a bit more structure to capture the second-order nature of the estimator. The ratios  $\rho$ ,  $\phi$ , and  $\kappa$  describe the temporal structure of the bias in second-order estimators. Analogous to the mean, given sufficient temporal consistency in these biases,  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  will have less bias. Again, the temporal consistency is not overly stringent; the bias of estimates at consecutive states must agree on sign but their relative magnitudes can vary significantly. For most transitions, it is reasonable to assume that these conditions hold true. In some MDPs, large changes in the reward can cause these requirements to break. Because Theorem 7.2 only establishes sufficiency, violating this requirement does not necessarily mean that  $\mathbb{V}_\delta[\delta \mid \tau]$  has greater bias than  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . Finally, it is worth noting that these are statements about a given transition  $\tau$ . In most state-action transitions, the requirements in Theorem 7.2 will hold, in which case  $\mathbb{E}_\delta[\delta \mid \tau]$  and  $\mathbb{V}_\delta[\delta \mid \tau]$  exhibit less overall bias. We provide direct empirical support that Theorem 7.2 holds in practice through careful ceteris paribus comparisons in Section 7.5.1.

To obtain a concrete signal for exploration, we follow O’Donoghue et al. (2018) and derive an exploration signal from the variance  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$ . Because  $p(\delta \mid \tau)$  is defined per transition, it cannot be used as-is for posterior sampling. Therefore, we incorporate TDU as a signal for exploration via an intrinsic reward. To obtain an exploration signal that is on approximately the same scale as the extrinsic reward, we use the standard deviation  $\sigma(\tau) := \sqrt{\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]}$  to define an augmented reward function

$$\tilde{\mathcal{R}}(\tau) := \mathcal{R}((s, a) \in \tau) + \beta \sigma(\tau), \quad (7.6)$$

where  $\beta \in [0, \infty)$  is a hyper-parameter that determines the emphasis on exploration. Another appealing property of  $\sigma$  is that it naturally decays as the agent converges on a solution (as model uncertainty diminishes); TDU defines a distinct MDP  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \tilde{\mathcal{R}}, \gamma)$  under Eq. 7.6 that converges on the true MDP in the limit of no model uncertainty. For a given policy  $\pi$  and distribution  $p(Q_\theta)$ , there exists an exploration policy  $\mu$  that collects transitions over which  $p(Q_\theta)$  exhibits maximal uncertainty, as measured by  $\sigma$ . In hard exploration problems, the exploration policy  $\mu$  can behave fundamentally differently from  $\pi$ . To capture such distinct exploration behaviour, we treat  $\mu$  as a separate exploration policy that we train to maximise the augmented reward  $\tilde{\mathcal{R}}$ , along-side training a policy  $\pi$  that maximises the extrinsic reward  $\mathcal{R}$ .

This gives rise to a natural separation of exploitation and exploration in the form of a cooperative multi-agent game, where the exploration policy is tasked with finding experiences where the agent is uncertain of its value estimate for the greedy policy  $\pi$ . As  $\pi$  is trained on this data, we expect uncertainty to vanish (up to noise). As this happens, the exploration policy  $\mu$  is incentivised to find new experiences with higher estimated uncertainty. This induces a particular pattern where exploration will reinforce experiences until the agent’s uncertainty vanishes, at which point the exploration policy expands its state visitation further. This process can allow TDU to overcome estimation bias in the posterior—since it is in effect exploiting it—in contrast to previous methods that do not maintain a distinct exploration policy. We demonstrate this empirically both on Montezuma’s Revenge and on Deep Sea (Osband et al., 2020).

## 7.4 Implementing TDU with Bootstrapping

The distribution over TD-errors that underlies TDU can be estimated using standard techniques for probability density estimation. In this paper, we leverage the statistical bootstrap as it is both easy to implement and provides a robust approximation without requiring distributional assumptions. TDU is easy to implement under the statistical bootstrap—it requires only a few lines of extra code. It can be implemented with value-based as well as actor-critic algorithms (we provide generic pseudo code in Appendix 7.A); in this paper, we focus on  $Q$ -learning.  $Q$ -learning alternates between policy evaluation (Eq. 7.1) and policy improvement under a greedy policy  $\pi_\theta(s) = \arg \max_a Q_\theta(s, a)$ . Deep  $Q$ -learning (Mnih et al., 2015) learns  $Q_\theta$  by minimising its TD-error by stochastic gradient descent on transitions sampled from a replay buffer. Unless otherwise stated, in practice we adopt a common approach of evaluating the action taken by the learned network through a target network with separate parameters that are updated periodically (Van Hasselt et al., 2016).

Our implementation starts from the bootstrapped DQN (Osband et al., 2016a), which maintains a set of  $K$  function approximators  $\mathcal{Q} = \{Q_{\theta^k}\}_{k=1}^K$ , each parameterised by  $\theta^k$  and regressed towards a unique target function using bootstrapped sampling of data from a shared replay memory. The Bootstrapped DQN derives a policy  $\pi_\theta$  by sampling  $\theta$  uniformly from  $\mathcal{Q}$  at the start of each episode. We provide an overview of the Bootstrapped DQN in Algorithm 7.1 for reference. To implement TDU in this setting, we make a change to the loss function (Algorithm 7.2, changes highlighted in green). First, we estimate the TDU signal  $\sigma$  using bootstrapped value estimation. We estimate  $\sigma$  through observed

TD-errors  $\{\delta_k\}_{k=1}^K$  incurred by the ensemble  $\mathcal{Q}$  on a given transition:

$$\sigma(\tau) \approx \sqrt{\frac{1}{K-1} \sum_{k=1}^K (\delta(\theta_k, \tau) - \bar{\delta}(\tau))^2}, \quad (7.7)$$

where  $\bar{\delta} = \gamma \bar{Q}' + r - \bar{Q}$ , with  $\bar{x} := \frac{1}{K} \sum_{i=1}^K x_i$  and  $Q' := Q(s', \pi(s'))$ . An important assumption underpinning the bootstrapped estimation is that of stochastic optimism (Osband et al., 2016b), which requires the distribution over  $\mathcal{Q}$  to be approximately as wide as the true distribution over value estimates. If not, uncertainty over  $\mathcal{Q}$  can collapse, which would cause  $\sigma$  to also collapse. To prevent this,  $\mathcal{Q}$  can be endowed with a prior (Osband et al., 2018) that maintains diversity in the ensemble by defining each value function as  $Q_{\theta^k} + \lambda P_k$ ,  $\lambda \in [0, \infty)$ , where  $P_k$  is a random prior function.

Rather than feeding this exploration signal back into the value functions in  $\mathcal{Q}$ , which creates a positive feedback loop (uncertainty begets higher reward, which begets higher uncertainty ad-infinitum), we introduce a separate ensemble of exploration value functions  $\tilde{\mathcal{Q}} = \{Q_{\tilde{\theta}^k}\}_{k=1}^N$  that we train over the augmented reward (Eq. 7.6). We derive an exploration policy  $\mu_{\tilde{\theta}}$  by sampling exploration parameters  $\tilde{\theta}$  uniformly from  $\tilde{\mathcal{Q}}$ , as in the standard bootstrapped DQN.

In summary, our implementation of TDU maintains  $K + N$  value functions. The first  $K$  defines a standard Bootstrapped DQN. From these, we derive an exploration signal  $\sigma$ , which we use to train the last  $N$  value functions. At the start of each episode, we proceed as in the standard Bootstrapped DQN and randomly sample a parameterisation  $\theta$  from  $\mathcal{Q} \cup \tilde{\mathcal{Q}}$  that we act under for the duration of the episode. All value functions are trained by bootstrapping from a single shared replay memory (Algorithm 7.1); see Appendix 7.A for a complete JAX (Bradbury et al., 2018) implementation. Consequently, we execute the (extrinsic) reward-maximising policy  $\pi_{\theta \sim \mathcal{Q}}$  with probability  $K/(K+N)$  and the exploration policy  $\mu_{\tilde{\theta} \sim \tilde{\mathcal{Q}}}$  with probability  $N/(K+N)$ . While  $\pi$  visits states around current reward-maximising behaviour,  $\mu$  searches for data with high model uncertainty. While each population  $\mathcal{Q}$  and  $\tilde{\mathcal{Q}}$  can be seen as performing Bayesian inference, it is not immediately clear that the full agent admits a Bayesian interpretation. We leave this question for future work.

There are several equally valid implementations of TDU (see Appendix 7.A for generic implementations for value-based learning and policy-gradient methods). In our case, it would be equally valid to define only a single exploration policy (i.e.  $N = 1$ ) and specify the probability of sampling this

**Algorithm 7.1** Bootstrapped DQN with TDU**Require:**  $M, \mathcal{L}$ : MDP to solve, TDU loss**Require:**  $\beta, K, N, \rho$ : hyper-parameters

```

1: Initialise  $\mathcal{B}$ : replay buffer
2: Initialise  $K + N$  value functions,  $Q \cup \tilde{Q}$ 
3: while not done do
4:   Observe  $s$  and choose value function for acting:  $Q_k \sim Q \cup \tilde{Q}$ 
5:   while episode not done do
6:     Take greedy action under  $Q_k$ :  $a = \arg \max_{\hat{a}} Q_k(s, \hat{a})$ 
7:     Sample bootstrapping mask  $m = (m_0, \dots, m_{K+N})$  with
        $m_i \sim \text{Binomial}(n=1, p=\rho) \quad \forall i \in 1, \dots, K+N$ 
8:     Enqueue transition  $(s, a, r, s', m)$  to  $\mathcal{B}$ 
9:     Optimise  $\mathcal{L}(\{\theta^k\}_1^K, \{\tilde{\theta}^k\}_1^N, \gamma, \beta, \mathcal{D} \sim \mathcal{B})$ 
10:   end while
11: end while

```

**Algorithm 7.2** Bootstrapped TD-loss with TDU.**Require:**  $\{\theta^k\}_1^K, \{\tilde{\theta}^k\}_1^N$ : parameters for  $Q$  and  $\tilde{Q}$ , respectively**Require:**  $\gamma, \beta, \mathcal{D}$ : hyper-parameters, mini-batch of transitions

```

1: Initialise mini-batch loss  $\ell \leftarrow 0$ 
2: for  $s, a, r, s', m \in \mathcal{D}$  do
3:    $\tau \leftarrow (s, a, r, s', \gamma)$ 
4:   Compute TD-errors for  $Q$ :  $\{\delta_i\}_{i=1}^K = \{\delta(\theta^i, \tau)\}_{i=1}^K$ 
5:   Compute  $\sigma$  from  $\{\delta_k\}_{k=1}^K$  (Eq. 7.7)
6:   Update  $\tau$  by  $r \leftarrow r + \beta \text{sg}[\sigma]$ , where  $\text{sg}$  stops gradients
7:   Compute TD-errors for  $\tilde{Q}$ :  $\{\tilde{\delta}_j\}_{j=K+1}^N = \{\delta(\tilde{\theta}^j, \tau)\}_{j=K+1}^N$ 
8:   Increment bootstrapped loss:  $\ell \leftarrow \ell + \sum_{i=1}^K m_i \delta_i^2 + \sum_{j=1}^N m_{K+j} \tilde{\delta}_j^2$ 
9: end for
10: return:  $\ell / (2(N+K) |\mathcal{D}|)$ 

```

policy. While this can result in faster learning, a potential drawback is that it restricts the exploratory behaviour that  $\mu$  can exhibit at any given time. Using a full bootstrapped ensemble for the exploration policy leverages the behavioural diversity of bootstrapping.

## 7.5 Empirical Evaluation

### 7.5.1 Behaviour Suite

Bsuite (Osband et al., 2020) was introduced as a benchmark for characterising core capabilities of RL agents. We focus on a Deep Sea, which is explicitly designed to test for deep exploration. This is a simple instance of problems

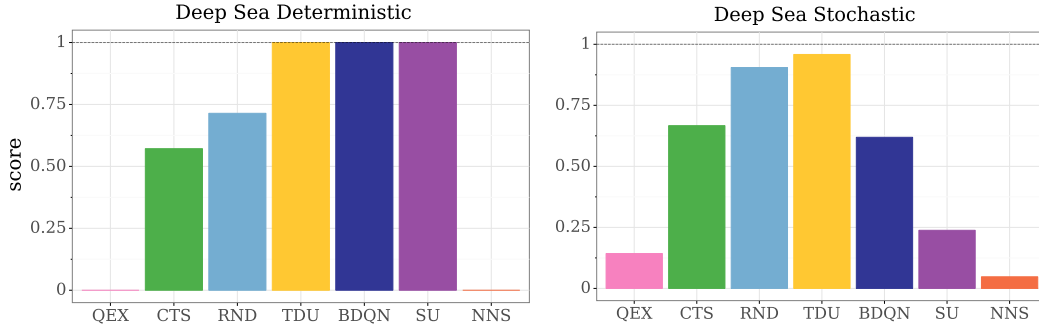


Figure 7.1: Deep Sea Benchmark. QEX, CTS, and RND use intrinsic rewards; BDQN, SU, and NNS use posterior sampling (Section 7.5.1). Posterior sampling does well on the deterministic version, but struggles on the stochastic version, suggesting an estimation bias (Section 7.2). TDU performs (near-)optimally on both the deterministic and the stochastic version of Deep Sea.

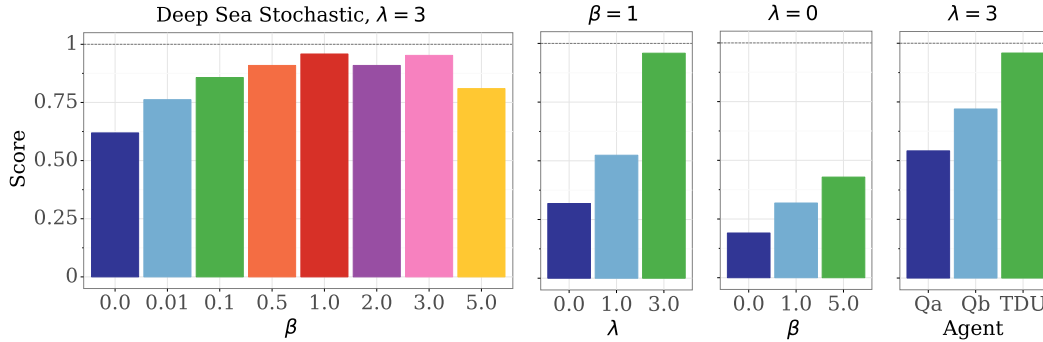


Figure 7.2: Deep Sea results. All models solve the deterministic version for prior scale  $\lambda = 3$  (dashed line). TDU also solves it for  $\lambda = 1$ . *Left*: introducing stochasticity substantially deteriorates baseline performance; including TDU ( $\beta > 0$ ) recovers close to full performance. *Center left*: effect of varying  $\lambda$ , TDU benefits from diversity in  $Q$  estimates. *Center right*: effect of removing prior ( $\lambda = 0$ ). Increasing  $\beta$  improves exploration, but does not reach full performance. *Right*: Qa replaces  $\sigma(\delta)$  with  $\sigma(Q)$ , Qb acts by  $\arg\max_a(Q + \sigma(Q))(s, a)$ . Estimating uncertainty over  $Q$  fails to match TDU.

that require persistent and directed exploration in the absence of positive reward signals, which can happen for sparse reward problems like strategic games, theorem proving, etc.. In Deep Sea, only one out of  $2^N$  policies yields any positive reward. Performance is compared over environment grid sizes  $N \in \{10, 12, \dots, 50\}$ , with an overall “score” that is the percentage of  $N$  for which average regret goes to below 0.9 faster than  $2^N$ . The stochastic version generates a ‘bad’ transition with probability  $1/N$ . This is a relatively high degree of uncertainty since the agent cannot recover from a bad transition in an episode.

For all experiments, we use a standard MLP with  $Q$ -learning, off-policy replay and a separate target network. See Appendix 7.D for details and TDU results on the full suite. We compare TDU on Deep Sea to a battery of exploration methods, broadly divided into methods that facilitate exploration by (a) sampling from a posterior (Bootstrapped DQN, Noisy Nets (Fortunato et al., 2018), Successor Uncertainties (Janz et al., 2019)) or (b) use an intrinsic reward (Random Network Distillation (RND; Burda et al., 2018b), CTS (Bellemare et al., 2016), and Q-Explore (QEX; Simmons-Edler et al., 2019)). We report best scores obtained from a hyper-parameter sweep for each method. Overall, performance varies substantially between methods; only TDU performs (near-)optimally on both the deterministic and stochastic version. Methods that rely on posterior sampling do well on the deterministic version, but suffer a substantial drop in performance on the stochastic version. As the stochastic version serves to increase the complexity of modelling future state visitation, this is clear evidence that these methods suffer from the estimation bias identified in Section 7.2. We could not make Q-explore and NoisyNets perform well in the default Bsuite setup, while Successor Uncertainties suffers a catastrophic loss of performance on the stochastic version of DeepSea.

Examining TDU, we find that it facilitates exploration while retaining overall performance except on Mountain Car where  $\beta > 0$  hurts performance (Appendix 7.D). For Deep Sea (Figure 7.2), prior functions are instrumental, even for large exploration bonuses ( $\beta \gg 0$ ). However, for a given prior strength, TDU does better than the BDQN ( $\beta = 0$ ). In the stochastic version of Deep Sea, BDQN suffers a significant loss of performance (Figure 7.2). As this is a ceteris paribus comparison, this performance difference can be directly attributed to an estimation bias in the BDQN that TDU circumvents through its intrinsic reward. That TDU is able to facilitate efficient exploration despite environment stochasticity demonstrates that it can correct for such estimation errors.

Finally, we verify Theorem 7.2 experimentally. We compare TDU to versions that estimate uncertainty directly over  $Q$  (full analysis in Section 7.D). We compare TDU to (a) a version where  $\sigma$  is defined as standard deviation over  $Q$  and (b) where  $\sigma(Q)$  is used as an upper confidence bound in the policy instead of as an intrinsic reward (Figure 7.2). Neither matches TDU’s performance across Bsuite and in particular on Deep Sea. Being ceteris paribus comparisons, this demonstrates that estimating uncertainty over TD-errors provides a stronger signal for exploration, as per Theorem 7.2.



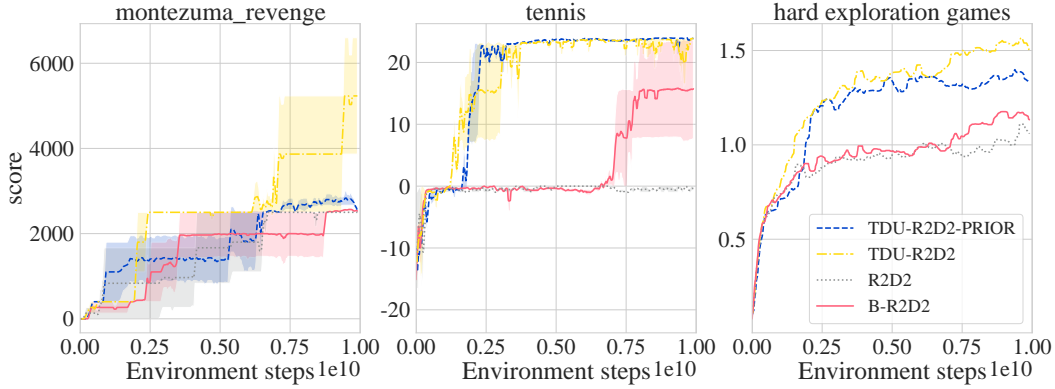


Figure 7.3: Atari results with distributed training. We compare TDU with and without additive prior functions to R2D2 and Bootstrapped R2D2 (B-R2D2). *Left*: Results for `montezuma_revenge`. *Center*: Results for `tennis`. *Right*: Mean HNS for the hard exploration games in the Atari2600 suite (including `tennis`). TDU achieves a higher performance than baselines.

### 7.5.2 Atari

Theorem 7.1 shows that estimation bias is particularly likely in complex environments that require neural networks to generalise across states. In recent years, such domains have seen significant improvements from running on distributed training platforms that can process large amounts of experience obtained through agent parallelism. It is thus important to develop exploration algorithms that scale gracefully and can leverage the benefits of distributed training. Therefore, we evaluate whether TDU can have a positive impact when combined with the Recurrent Replay Distributed DQN (R2D2) (Kapturowski et al., 2018), which achieves state-of-the-art results on the Atari2600 suite by carefully combining a set of key components: a recurrent state, experience replay, off-policy value learning and distributed training.

As a baseline we implemented a distributed version of the bootstrapped DQN with additive prior functions. We present full implementation details, hyperparameter choices, and results on all games in Section 7.E. For our main results, we run each agent on 8 seeds for 20 billion steps. We focus on games that are well-known to pose challenging exploration problems (Machado et al., 2018): `montezuma_revenge`, `pitfall`, `private_eye`, `solaris`, `venture`, `gravitar`, and `tennis`. Following standard practice, Figure 7.3 reports Human Normalized Score (HNS),

$$\text{HNS} = \frac{\text{Agent}_{\text{score}} - \text{Random}_{\text{score}}}{\text{Human}_{\text{score}} - \text{Random}_{\text{score}}},$$



as an aggregate result across exploration games as well as results on `tennis` and `montezuma_revenge`, which are both known to be particularly hard exploration games (Machado et al., 2018).

Generally, we find that TDU facilitates exploration substantially, improving the mean HNS score across exploration games by 30% compared to baselines (right panel, Figure 7.3). An ANOVA analysis yields a statistically significant difference between TDU and non-TDU methods, controlling for game ( $F = 8.17, p = 0.0045$ ). Notably, TDU achieves significantly higher returns on `montezuma_revenge` and is the only agent that consistently achieves the maximal return on `tennis`. We report all per-game results in Section 7.E.4. We observe no significant gains from including prior functions with TDU and find that bootstrapping alone produces relatively marginal gains. Beyond exploration games, TDU can match or improve upon the baseline, but exhibits sensitivity to TDU hyper-parameters ( $\beta$ , number of explorers ( $N$ ); see Section 7.E.3 for details). This finding is in line with observations made by (Puigdomènech Badia et al., 2020); combining TDU with online hyper-parameter adaptation (Schaul et al., 2019; Xu et al., 2018a; Zahavy et al., 2020) are exciting avenues for future research. See Section 7.E for further comparisons.

In Table 7.1, we compare TDU to recently proposed state-of-the-art exploration methods. While comparisons must be made with care due to different training regimes, computational budgets, and architectures, we note a general trend that no method is uniformly superior. Methods that are good on extremely sparse exploration games (`montezuma_revenge` and `pitfall!`) tend to do poorly on games with dense rewards and vice versa. TDU is generally among the top 2 algorithms in all cases except on `montezuma_revenge` and `pitfall!`, state-based exploration is needed to achieve sufficient coverage of the MDP. TDU generally outperforms both Pixel-CNN (Ostrovski et al., 2017), CTS, and RND. TDU is the only algorithm to achieve super-human performance on `solaris` and achieves the highest score of all baselines considered on `venture`.

## 7.6 Related Work

Bayesian approaches to exploration typically use uncertainty as the mechanism for balancing exploitation and exploration (Strens, 2000). A popular instance of this form of exploration is the PILCO algorithm (Deisenroth & Rasmussen, 2011). While we rely on the bootstrapped DQN (Osband et al., 2016a) in this paper, several other uncertainty estimation techniques have been proposed, such as by placing a parameterised distribution over model parameters (Fortunato et al.,

Table 7.1: Atari benchmark on exploration games.<sup>†</sup>Ostrovski et al. (2017),  
<sup>‡</sup>Bellemare et al. (2016), <sup>◊</sup>Burda et al. (2018b), <sup>\*</sup>Choi et al. (2018),  
<sup>§</sup>Puigdomènech Badia et al. (2020), <sup>+</sup>With prior functions.

Algorithm	Gravitar	Montez. Revenge	Pitfall!	Private Eye	Solaris	Venture
Avg. Human	3,351	4,753	6,464	69,571	12,327	1,188
R2D2	15,680	2,061	0	5,322	3,787	1,971
DQN-PixelCNN <sup>†</sup>	859	2,514	0	15,807	5,502	1,356
DQN-CTS <sup>‡</sup>	498	3,706	0	8,359	82	–
RND <sup>◊</sup>	3,906	10,070	-3	8,666	3,282	1,859
CoEx <sup>*</sup>	–	11,618	–	11,000	–	1,916
NGU <sup>§</sup>	14,100	10,400	8,400	100,000	4,900	1,700
TDU-R2D2	13,000	5,233	0	40,544	14,712	2,000
TDU-R2D2 <sup>+</sup>	10,916	2,833	0	61,168	15,230	1,977

2018; Plappert et al., 2018) or by modeling a distribution over both the value and the returns (Moerland et al., 2017), using Bayesian linear regression on the value function (Azizzadenesheli et al., 2018; Janz et al., 2019), or by modelling the variance over value estimates as a Bellman operation (O’Donoghue et al., 2018). The underlying exploration mechanism in these works is posterior sampling from the agent’s current beliefs (Thompson, 1933; Dearden et al., 1998); our work suggests that estimating this posterior is significantly more challenging than previously thought.

An alternative to posterior sampling is to facilitate exploration via learning by introducing an intrinsic reward function. Previous works typically formulate intrinsic rewards in terms of state visitation (Lopes et al., 2012; Bellemare et al., 2016; Puigdomènech Badia et al., 2020), state novelty (Schmidhuber, 1991; Oudeyer & Kaplan, 2009; Pathak et al., 2017), or state predictability (Florensa et al., 2017; Burda et al., 2018b; Gregor et al., 2016; Hausman et al., 2018). Most of these works rely on properties of the state space to drive exploration while ignoring rewards. While this can be effective in sparse reward settings (e.g. Burda et al., 2018b; Puigdomènech Badia et al., 2020), it can also lead to arbitrarily bad exploration (see analysis in Osband et al., 2019).

A smaller body of work uses statistics derived from observed rewards (Nachum et al., 2016) or TD-errors to design intrinsic reward functions; our work is particularly related to the latter. Tokic (2010) proposes an extension of  $\epsilon$ -greedy exploration, where the TD-error modulates  $\epsilon$  to be higher in states with higher TD-error. Gehring & Precup (2013) use the mean absolute TD-error,

accumulated over time, to measure controllability of a state and reward the agent for visiting states with low mean absolute TD-error. In contrast to our work, this method integrates the TD-error over time to obtain a measure of irreducibility. [Simmons-Edler et al. \(2019\)](#) propose to use two  $Q$ -networks, where one is trained on data collected under both networks and the other obtains an intrinsic reward equal to the absolute TD-error of the first network on a given transition. In contrast to our work, this method does not have a probabilistic interpretation and thus does not control for uncertainty over the environment. TD-errors have also been used in [White \(2015\)](#), where surprise is defined in terms of the moving average of the TD-error over the full variance of the TD-error. [Kumaraswamy et al. \(2018\)](#) rely on least-squares TD-errors to derive a context-dependent upper-confidence bound for directed exploration. Finally, using the TD-error as an exploration signal is related to the notion of “learnability” or curiosity as a signal for exploration, which is often modelled in terms of the prediction error in a dynamics model (e.g. [Schmidhuber, 1991](#); [Oudeyer et al., 2007](#); [Gordon & Ahissar, 2011](#); [Pathak et al., 2017](#)).

## 7.7 Conclusion

We present Temporal Difference Uncertainties (TDU), a method for estimating uncertainty over an agent’s value function. Obtaining well-calibrated uncertainty estimates under function approximation is non-trivial and we show that popular approaches, while in principle valid, can fail to accurately represent uncertainty over the value function because they must represent an unknown future.

This motivates TDU as an estimate of uncertainty conditioned on observed state-action transitions, so that the only source of uncertainty for a given transition is due to uncertainty over the agent’s parameters. This gives rise to an intrinsic reward that encodes the agent’s model uncertainty, and we capitalise on this signal by introducing a distinct exploration policy. This policy is incentivised to collect data over which the agent has high model uncertainty and we highlight how this separation gives rise to a form of cooperative multi-agent game. We demonstrate empirically that TDU can facilitate efficient exploration in hard exploration games such as Deep Sea and Montezuma’s Revenge.

## 7

## Appendix

## 7.A Implementation and Code

In this Section, we provide code for implementing TDU in a general policy-agnostic setting and in the specific case of bootstrapped  $Q$ -learning. Algorithm 7.3 presents TDU in a policy-agnostic framework. TDU can be implemented as a pre-processing step (Algorithm 7.3, Line 9) that augments the reward with the exploration signal before computing the policy loss. If Algorithm 7.3 is used to learn a single policy, it benefits from the TDU exploration signal but cannot learn distinct exploration policies for it. In particular, on-policy learning does not admit such a separation. To learn a distinct exploration policy, we can use Algorithm 7.3 to train the *exploration* policy, while the another policy is trained to maximise extrinsic rewards only using both its own data and data from the exploration policy. In case of multiple policies, we need a mechanism for sampling behavioural policies. In our experiments we settled on uniform sampling; more sophisticated methods can potentially yield better performance.

In the case of value-based learning, TDU takes a special form that can be implemented efficiently as a staggered computation of TD-errors (Algorithm 7.4). Concretely, we compute an estimate of the distribution of TD-errors from some given distribution over the value function parameters (Algorithm 7.4, Line 3). These TD-errors are used to compute the TDU signal  $\sigma$ , which then modulates the reward used to train a  $Q$  function (Algorithm 7.4, Line 7). Because the only quantities being computed are TD-errors, this can be combined into a single error signal (Algorithm 7.4, Line 11). When implemented under bootstrapping,  $Q_{\text{params}}$  denotes the ensemble  $Q$  and  $Q_{\text{tilde\_distribution\_params}}$  denotes the ensemble  $\tilde{Q}$ ; we compute the loss as in Algorithm 7.2.

Finally, Algorithm 7.5 presents a complete JAX (Bradbury et al., 2018) implementation that can be used along with the Bsuite (Osband et al.,

2020) codebase.<sup>10</sup> We present the corresponding TDU agent class (Algorithm 7.4), which is a modified version of the BootstrappedDqn class in `bsuite/baselines/jax/boot_dqn/agent.py` and can be used by direct swap-in.

---

**Algorithm 7.3** Pseudo-code for generic TDU loss
 

---

```

1 def loss(transitions, pi_params, Qtilde_distribution_params, beta):
2     # Estimate TD-error distribution.
3     td = array([td_error(p, transitions) for p in sample(
4         Qtilde_distribution_params)])
5     # Compute critic loss.
6     td_loss = mean(0.5 * (td ** 2))
7
8     # Compute exploration bonus.
9     transitions.r_t += beta * stop_gradient(std(td, axis=1))
10
11    # Compute policy loss on transition with augmented reward.
12    pi_loss = pi_loss_fn(pi_params, transitions)
13
14    return pi_loss, td_loss

```

---



---

**Algorithm 7.4** Pseudo-code for Q-learning TDU loss
 

---

```

1 def loss(transitions, Q_params, Qtilde_distribution_params, beta):
2     # Estimate TD-error distribution.
3     td_K = array([td_error(p, transitions) for p in sample(
4         Qtilde_distribution_params)])
5     # Compute exploration bonus and Q-function reward.
6     transitions.reward_t += beta * stop_gradient(std(td_K, axis=1))
7
8     td_N = td_error(Q_params, transitions)
9
10    # Combine for overall TD-loss.
11    td_errors = concatenate((td_ex, td_in), axis=1)
12    td_loss = mean(0.5 * (td_errors ** 2))
13
14    return td_loss

```

---



---

<sup>10</sup>Available at: <https://github.com/deepmind/bsuite>.

**Algorithm 7.5** JAX implementation of TDU agent under Bootstrapped DQN

---

```

1 # Copyright 2020 the Temporal Difference Uncertainties as a Signal for Exploration authors. Licensed under
2 # the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with
3 # the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0.
4 # Unless required by applicable law or agreed to in writing, software distributed under the License is
5 # distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
6 # See the License for the specific language governing permissions and limitations under the License.
7
8 class TDU(bsuite.baselines.jax.boot_dqn.BootstrappedDqn):
9
10     def __init__(self, K: int, beta: float, **kwargs: Any):
11         """TDU under Bootstrapped DQN with randomized prior functions."""
12         super(TDU, self).__init__(**kwargs)
13         network, optimizer, N = kwargs['network'], kwargs['optimizer'], kwargs['num_ensemble']
14         noise_scale, discount = kwargs['noise_scale'], kwargs['discount']
15
16     def td(params: hk.Params, target_params: hk.Params,
17           transitions: Sequence[jnp.ndarray]) -> jnp.ndarray:
18         """TD-error with added reward noise + half-in bootstrap."""
19         o_tm1, a_tm1, r_t, d_t, o_t, z_t = transitions
20         q_tm1 = network.apply(params, o_tm1)
21         q_t = network.apply(target_params, o_t)
22         r_t += noise_scale * z_t
23         return jax.vmap(rlax.q_learning)(q_tm1, a_tm1, r_t, discount * d_t, q_t)
24
25     def loss(params: Sequence[hk.Params], target_params: Sequence[hk.Params],
26             transitions: Sequence[jnp.ndarray]) -> jnp.ndarray:
27         """Q-learning loss with TDU."""
28         # Compute TD-errors for first K members.
29         o_tm1, a_tm1, r_t, d_t, o_t, m_t, z_t = transitions
30         td_K = [td(params[k], target_params[k],
31                   [o_tm1, a_tm1, r_t, d_t, o_t, z_t[:, k]]) for k in range(K)]
32
33         # TDU signal on first K TD-errors.
34         r_t += beta * jax.lax.stop_gradient(jnp.std(jnp.stack(td_K, axis=0), axis=0))
35
36         # Compute TD-errors on augmented reward for last K members.
37         td_N = [td(params[k], target_params[k],
38                   [o_tm1, a_tm1, r_t, d_t, o_t, z_t[:, k]]) for k in range(K, N)]
39
40         return jnp.mean(m_t.T * jnp.stack(td_K + td_N) ** 2)
41
42     def update(state: TrainingState, gradient: Sequence[jnp.ndarray]) ->
43     TrainingState:
44         """Gradient update on ensemble member."""
45         updates, new_opt_state = optimizer.update(gradient, state.opt_state)
46         new_params = optix.apply_updates(state.params, updates)
47         return TrainingState(params=new_params, target_params=state.target_params,
48                             opt_state=new_opt_state, step=state.step + 1)
49
50     @jax.jit
51     def sgd_step(states: Sequence[TrainingState],
52                 transitions: Sequence[jnp.ndarray]) -> Sequence[TrainingState]:
53         """Does a step of SGD for the whole ensemble over 'transitions'."""
54         params, target_params = zip(*[(state.params, state.target_params) for state in
55                                       states])
56         gradients = jax.grad(loss)(params, target_params, transitions)
57         return [update(state, gradient) for state, gradient in zip(states,
58                           gradients)]
59
60     self._sgd_step = sgd_step # patch BootDQN sgd_step with TDU sgd_step.
61
62     def update(self, timestep: dm_env.TimeStep, action: base.Action,
63               new_timestep: dm_env.TimeStep):
64         """Update the agent: add transition to replay and periodically do SGD."""
65         if new_timestep.last():
66             self._active_head = self._ensemble[np.random.randint(0, self._num_ensemble)]
67         ]
68
69         mask = np.random.binomial(1, self._mask_prob, self._num_ensemble)
70         noise = np.random.randn(self._num_ensemble)
71         transition=[timestep.observation, action, np.float32(new_timestep.reward),
72                   np.float32(new_timestep.discount), new_timestep.observation, mask,
73                   noise]
74         self._replay.add(transition)
75         if self._replay.size < self._min_replay_size:
76             return
77
78         if self._total_steps % self._sgd_period == 0:
79             transitions = self._replay.sample(self._batch_size)
80             self._ensemble = self._sgd_step(self._ensemble, transitions)
81
82         for k, state in enumerate(self._ensemble):
83             if state.step % self._target_update_period == 0:
84                 self._ensemble[k] = state._replace(target_params=state.params)

```

---

## 7.B Proofs

We begin with the proof of Lemma 7.1. First, we show that if Eq. 7.2 and Eq. 7.3 fail,  $p(\theta)$  induce a distribution  $p(Q_\theta)$  whose first two moments are biased estimators of the moments of the distribution of interest  $p(Q_\pi)$ , for *any* choice of belief over the MDP,  $p(M)$ . We restate it here for convenience.

**Lemma 7.1.** *If  $\mathbb{E}_\theta[Q_\theta]$  and  $\mathbb{V}_\theta[Q_\theta]$  fail to satisfy Eqs. 7.2 and 7.3, respectively, they are biased estimators of  $\mathbb{E}_M[Q_\pi^M]$  and  $\mathbb{V}_M[Q_\pi^M]$  for any choice of  $p(M)$ .*

*Proof.* Assume the contrary, that  $\mathbb{E}_M[Q_\pi^M(s, \pi(s))] = \mathbb{E}_\theta[Q_\theta(s, \pi(s))]$  for all  $(s, a) \in \mathcal{S} \times \mathcal{A}$ . If Eqs. 7.2 and 7.3 do not hold, then for any  $\mathbb{M} \in \{\mathbb{E}, \mathbb{V}\}$ ,

$$\mathbb{M}_M[Q_\pi^M(s, \pi(s))] = \mathbb{M}_\theta[Q_\theta(s, \pi(s))] \quad (7.8)$$

$$\neq \mathbb{M}_\theta \left[ \mathbb{E}_{\substack{s' \sim \mathcal{P}(s, \pi(s)) \\ r \sim \mathcal{R}(s, \pi(s))}} [r + \gamma Q_\theta(s', \pi(s'))] \right] \quad (7.9)$$

$$= \mathbb{E}_{\substack{s' \sim \mathcal{P}(s, \pi(s)) \\ r \sim \mathcal{R}(s, \pi(s))}} [r + \gamma \mathbb{M}_\theta[Q_\theta(s', \pi(s'))]] \quad (7.10)$$

$$= \mathbb{E}_{\substack{s' \sim \mathcal{P}(s, \pi(s)) \\ r \sim \mathcal{R}(s, \pi(s))}} [r + \gamma \mathbb{M}_M[Q_\pi^M(s', \pi(s'))]] \quad (7.11)$$

$$= \mathbb{M}_M \left[ \mathbb{E}_{\substack{s' \sim \mathcal{P}(s, \pi(s)) \\ r \sim \mathcal{R}(s, \pi(s))}} [r + \gamma Q_\pi^M(s', \pi(s'))] \right] \quad (7.12)$$

$$= \mathbb{M}_M[Q_\pi^M(s, \pi(s))], \quad (7.13)$$

a contradiction; conclude that  $\mathbb{M}_M[Q_\pi^M(s, \pi(s))] \neq \mathbb{M}_\theta[Q_\theta(s, \pi(s))]$ . Eqs. 7.9 and 7.13 use Eqs. 7.2 and 7.3; Eqs. 7.8, 7.9 and 7.11 follow by assumption; Eqs. 7.10 and 7.12 use linearity of the expectation operator  $\mathbb{E}_{r, s'}$  by virtue of  $\mathbb{M}$  being defined over  $\theta$ . As  $(s, a, r, s')$  and  $p(M)$  are arbitrary, the conclusion follows.  $\blacksquare$

Methods that take inspiration from by PSRL but rely on neural networks typically approximate  $p(M)$  by a parameter distribution  $p(\theta)$  over the value function. Lemma 7.1 establishes that the induced distribution  $p(Q_\theta)$  under push-forward of  $p(\theta)$  must propagate the moments of the distribution  $p(Q_\theta)$  consistently over the state-space to be unbiased estimate of  $p(Q_\pi^M)$ , for any  $p(M)$ .

With this in mind, we now turn to neural networks and their ability to estimate value function uncertainty in MDPs. To prove our main result, we establish two intermediate results. Recall that we define a function approximator  $Q_\theta = w \circ \phi_\theta$ , where  $\theta = (w_1, \dots, w_n, \vartheta_1, \dots, \vartheta_v)$ ;  $w \in \mathbb{R}^n$  is a linear layer and

$\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^n$  is a function approximator with parameters  $\vartheta \in \mathbb{R}^v$ . We denote by  $e(s, a) = \phi_\vartheta(s, a) - \mathbb{E}_{s', r} [r + \gamma \phi_\vartheta(s', \pi(s'))]$  its function approximation error, i.e. the Temporal Difference error in terms of  $\phi$ . Note that if  $Q_\theta(s, a) \neq \mathbb{E}_{r, s'} [r + \gamma Q_\theta(s', \pi(s'))]$ , then  $e(s, a)$  is non-zero by linearity in  $w$ .

As before, let  $M$  be an MDP  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$  with discrete state and action spaces. We denote by  $N$  the number of states and actions with  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$  and  $Q_\theta(s, a) \neq \mathbb{E}_{r, s'} [r + \gamma Q_\theta(s', \pi(s'))]$ , with  $\mathcal{N} \subset \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A}$  the set of all such pairs  $(s, a, s', a')$ . This set can be thought of as a minimal MDP—the set of states within a larger MDP where the function approximator generates unique predictions. It arises in an MDP through dense rewards, stochastic rewards, or irrevocable decisions, such as in Deep Sea. Our first result is concerned with a very common approach, where  $\vartheta$  is taken to be a point estimate so that  $p(\theta) = p(w)$ . This approach is often used for large neural networks, where placing a posterior over the full network would be too costly (Osband et al., 2016a; O'Donoghue et al., 2018; Azizzadenesheli et al., 2018; Janz et al., 2019).

**Lemma 7.2** (Bellman uncertainty bias under diagonal prior). *Let  $p(\theta) = p(w)$ . If  $N - 1 > n$ , with  $w \in \mathbb{R}^n$ , then  $\mathbb{E}_\theta[Q_\theta]$  fail to satisfy the first moment Bellman Equation (Eq. 7.2). Further, if  $N > n^2$ , then  $\mathbb{V}_\theta[Q_\theta]$  fail to satisfy the second moment Bellman Equation (Eq. 7.3).*

*Proof.* Write the first condition of Eq. 7.2 as

$$\mathbb{E}_\theta[w^T \phi_\vartheta(s, a)] = \mathbb{E}_\theta[\mathbb{E}_{r, s'} [r + \gamma w^T \phi_\vartheta(s', \pi(s'))]] . \quad (7.14)$$

Using linearity of the expectation operator along with  $p(\theta) = p(w)$ , we have

$$\mathbb{E}_w[w]^T \phi_\vartheta(s, a) = \mu(s, a) + \gamma \mathbb{E}_w[w]^T \mathbb{E}_{s'} [\phi_\vartheta(s', \pi(s'))] , \quad (7.15)$$

where  $\mu(s, a) = \mathbb{E}_{r \sim \mathcal{R}(s, a)} [r]$ . Rearrange to get

$$\mu(s, a) = \mathbb{E}_w[w]^T \left( \phi_\vartheta(s, a) - \gamma \mathbb{E}_{s'} [\phi_\vartheta(s', \pi(s'))] \right) . \quad (7.16)$$

By assumption  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$ , which implies  $\phi_\vartheta(s, a) \neq \phi_\vartheta(s', \pi(s'))$  by linearity in  $w$ . Hence  $\phi_\vartheta(s, a) - \gamma \mathbb{E}_{s'} [\phi_\vartheta(s', \pi(s'))]$  is non-zero and unique for each  $(s, a)$ . By definition of  $e$ ,  $\phi_\vartheta(s, a) = \mu(s, a) \frac{1}{1 + \gamma \mathbb{E}_{s'} [\phi_\vartheta(s', \pi(s'))]} + e_\phi(s, a)$ ,



where  $\mathbf{1} \in \mathbb{R}^n$  is a vector of ones. By assumption,  $e(s, a) \in \mathbb{R}^n$  is a random vector. Thus,  $\phi_{\vartheta}(s, a) - \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', \pi(s'))] = \mu(s, a) \mathbf{1} + e(s, a)$ . Repeating this for all  $(s, a)$  forms a system of linear equations over  $\mathcal{S} \times \mathcal{A}$ , which can be reduced to a full-rank system over  $\mathcal{N}$ :  $\mu = \Phi \mathbb{E}_w[w]$ , where  $\mu \in \mathbb{R}^N$  stacks expected reward  $\mu(s, a)$  and  $\Phi = \mu \mathbf{1}^T + E \in \mathbb{R}^{N \times n}$  stacks  $\phi_{\vartheta}(s, a) - \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', \pi(s'))]$  row-wise, with  $E$  a random matrix, hence full rank. Because  $E$  is full rank,  $\Phi$  is at least of rank  $N - 1$ . If  $N - 1 > n$ , this system has no solution. The conclusion follows for  $\mathbb{E}_{\theta}[Q_{\theta}]$ . If the estimator of the mean is used to estimate the variance, then the estimator of the variance is biased. For an unbiased mean, using linearity in  $w$ , write the condition of Eq. 7.3 as

$$\mathbb{E}_{\theta} \left[ \left[ (w - \mathbb{E}_w[w])^T \phi_{\vartheta}(s, a) \right]^2 \right] = \mathbb{E}_{\theta} \left[ \left[ \gamma (w - \mathbb{E}_w[w])^T \mathbb{E}_{s'}[\phi_{\vartheta}(s', \pi(s'))] \right]^2 \right]. \quad (7.17)$$

Let  $\tilde{w} = (w^T - \mathbb{E}_w[w])$ ,  $x = \tilde{w}^T \phi_{\vartheta}(s, a)$ ,  $y = \gamma \tilde{w}^T \mathbb{E}_{s'}[\phi_{\vartheta}(s', a')]$ . Rearrange to get

$$\mathbb{E}_{\theta} [x^2 - y^2] = \mathbb{E}_w [(x - y)(x + y)] = 0. \quad (7.18)$$

Expanding terms, we find

$$0 = \mathbb{E}_{\theta} \left[ \left( \tilde{w}^T [\phi_{\vartheta}(s, a) - \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', a')]] \right) \left( \tilde{w}^T [\phi_{\vartheta}(s, a) + \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', a')]] \right) \right] \quad (7.19)$$

$$= \sum_{i=1}^n \sum_{j=1}^n \mathbb{E}_w [\tilde{w}_i \tilde{w}_j] d_i^- d_j^+ = \sum_{i=1}^n \sum_{j=1}^n \text{Cov}(w_i, w_j) d_i^- d_j^+. \quad (7.20)$$

where we define  $d^- = \phi_{\vartheta}(s, a) - \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', a')]$  and  $d^+ = \phi_{\vartheta}(s, a) + \gamma \mathbb{E}_{s'}[\phi_{\vartheta}(s', a')]$ . As before,  $d^-$  and  $d^+$  are non-zero by assumption of unique  $Q$ -values. Perform a change of variables  $\omega_{\alpha(i,j)} = \text{Cov}(w_i, w_j)$ ,  $\lambda_{\alpha(i,j)} = d_i^- d_j^+$  to write Eq. 7.20 as  $0 = \lambda^T \omega$ . Repeating the above process for every state and action we have a system  $0 = \Lambda \omega$ , where  $0 \in \mathbb{R}^N$  and  $\Lambda \in \mathbb{R}^{N \times n^2}$  are defined by stacking vectors  $\lambda$  row-wise. This is a system of linear equations and if  $N > n^2$  no solution exists; thus, the conclusion follows for  $\mathbb{V}_{\theta}[Q_{\theta}]$ , concluding the proof. ■

Note that if  $\mathbb{E}_{\theta}[Q_{\theta}]$  is biased and used to construct the estimator  $\mathbb{E}_{\theta}[Q_{\theta}]$ , then this estimator is also biased; hence if  $N - 1 > n$ ,  $p(\theta)$  induce biased estimators  $\mathbb{E}_{\theta}[Q_{\theta}]$  and  $\mathbb{V}_{\theta}[Q_{\theta}]$  of  $\mathbb{E}_M[Q_{\pi}^M]$  and  $\mathbb{V}_M[Q_{\pi}^M]$ , respectively.

Lemma 7.2 can be seen as a statement about linear uncertainty. While the result is not too surprising from this point of view, it is nonetheless a frequently used approach to uncertainty estimation. We may hope then that by placing uncertainty over the feature extractor as well, we can benefit from

its nonlinearity to obtain greater representational capacity with respect to uncertainty propagation. Such posteriors come at a price. Placing a full posterior over a neural network is often computationally infeasible, instead a common approach is to use a diagonal posterior, i.e.  $\text{Cov}(\theta_i, \theta_j) = 0$  (Fortunato et al., 2018; Plappert et al., 2018). Our next result shows that any posterior of this form suffers from the same limitations as placing a posterior only over the final layer. We establish something stronger: any posterior of the form  $p(\theta) = p(w)p(\vartheta)$  suffers from the limitations described in Lemma 7.2.

**Lemma 7.3** (Bellman uncertainty bias under factorised prior). *Let  $p(\theta) = p(w)p(\vartheta)$ ; if  $N - 1 > n$ , with  $w \in \mathbb{R}^n$ , then  $\mathbb{E}_\theta[Q_\theta]$  fail to satisfy the first moment Bellman Equation (Eq. 7.2). Further, if  $N > n^2$ , then  $\mathbb{V}_\theta[Q_\theta]$  fail to satisfy the second moment Bellman Equation (Eq. 7.3).*

*Proof.* The proof largely proceeds as in the proof of Lemma 7.2. Re-write Eq. 7.15 as

$$\mathbb{E}_w[w]^T \mathbb{E}_\vartheta[\phi_\vartheta(s, a)] = \mu(s, a) + \gamma \mathbb{E}_w[w]^T \mathbb{E}_{s'}[\mathbb{E}_\vartheta[\phi_\vartheta(s', \pi(s'))]]. \quad (7.24)$$

Perform a change of variables  $\tilde{\phi} = \mathbb{E}_\vartheta[\phi_\vartheta]$  to obtain

$$\mu(s, a) = \mathbb{E}_w[w]^T \left( \tilde{\phi}(s, a) - \gamma \mathbb{E}_{s'}[\tilde{\phi}(s', \pi(s'))] \right). \quad (7.22)$$

Because  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$ , by linearity in  $w$  we have that  $\tilde{\phi}(s, a) - \tilde{\phi}(s', a')$  is non-zero for any  $(s', a')$ , and similarly by assumption  $e(s, a)$  is non-zero, hence Eq. 7.22 has no trivial solutions. Proceeding as in the proof of Lemma 7.2 obtains  $\mu = \tilde{\Phi} \mathbb{E}_w[w]$ , where  $\tilde{\Phi}$  is analogously defined. Note that if  $N - 1 > n$  there is no solution  $\mathbb{E}_w[w]$  for *any* admissible choice of  $\tilde{\Phi}$  (with rank at least  $N - 1$ ), and hence the conclusion follows for the first part. For the second part, using that  $\mathbb{E}_\theta = \mathbb{E}_w \mathbb{E}_\vartheta$  in Eq. 7.20 yields

$$0 = \sum_{i=1}^n \sum_{j=1}^n \mathbb{E}_w[\tilde{w}_i \tilde{w}_j] \mathbb{E}_\vartheta[d_i^- d_j^+] = \sum_{i=1}^n \sum_{j=1}^n \text{Cov}(w_i, w_j) \mathbb{E}_\vartheta[d_i^- d_j^+]. \quad (7.23)$$

Perform a change of variables  $\tilde{\lambda}_{\alpha(i,j)} = \mathbb{E}_\vartheta[d_i^- d_j^+]$ . Again, by  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$  we have that  $\tilde{\lambda}$  is non-zero; proceed as before to complete the proof. ■

We are now ready to prove our main result. We restate it here for convenience:

**Theorem 7.1.** *If the number of state-action pairs with unique predictions  $\mathbb{E}_\theta[Q_\theta(s, a)] \neq \mathbb{E}_\theta[Q_\theta(s', a')]$  and non-zero TD error  $Q_\theta(s, a) \neq \mathbb{E}_{r, s'}[r + \gamma Q_\theta(s', \pi(s'))]$  is greater than  $n+1$ , where  $w \in \mathbb{R}^n$ , then  $\mathbb{E}_\theta[Q_\theta]$  and  $\mathbb{V}_\theta[Q_\theta]$  are biased estimators of  $\mathbb{E}_M[Q_\pi^M]$  and  $\mathbb{V}_M[Q_\pi^M]$  for any choice of  $p(M)$ .*

*Proof.* Let  $p(\theta)$  be of the form  $p(\theta) = p(w)$  or  $p(\theta) = p(w)p(\vartheta)$ . By Lemmas 7.2 and 7.3,  $p(\theta)$  fail to satisfy Eq. 7.2. By Lemma 7.4, this causes  $\mathbb{E}_\theta[Q_\theta]$  to be a biased estimator of  $\mathbb{E}_M[Q_\pi^M]$ . This in turn implies that  $\mathbb{V}_\theta[Q_\theta]$  is a biased estimator of  $\mathbb{V}_M[Q_\pi^M]$ . Further, if  $N > n^2$ ,  $\mathbb{V}_\theta[Q_\theta]$  is biased independently of  $\mathbb{E}_\theta[Q_\theta]$ . ■

We now turn to analysing the bias of our proposed estimators. As before, we will build up to Theorem 7.2 through a series of lemmas. For the purpose of these results, let  $B : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  denote the bias of  $\mathbb{E}_\theta[Q_\theta]$  in any tuple  $(s, a) \in \mathcal{S} \times \mathcal{A}$ , so that  $\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)]) = B(s, a)$ .

**Lemma 7.4** (Bias of mean TD error). *Given a transition  $\tau := (s, a, r, s')$ , for any  $p(M)$ , given  $p(\theta)$ , if*

$$\frac{B(s', \pi(s'))}{B(s, a)} \in (0, 2/\gamma) \quad (7.24)$$

*then  $\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]$  has less bias than  $\mathbb{E}_\theta[Q_\theta(s, a)]$ .*

*Proof.* From direct manipulation of  $\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]$ , we have

$$\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau] = \mathbb{E}_\theta[\gamma Q_\theta(s', \pi(s')) + r - Q_\theta(s, a)] \quad (7.25)$$

$$= \gamma \mathbb{E}_\theta[Q_\theta(s', \pi(s'))] + r - \mathbb{E}_\theta[Q_\theta(s, a)] \quad (7.26)$$

$$= \gamma \mathbb{E}_M[Q_\pi^M(s', \pi(s'))] + r - \mathbb{E}_M[Q_\pi^M(s, a)] \quad (7.27)$$

$$+ \gamma B(s', \pi(s')) - B(s, a) \quad (7.28)$$

$$= \mathbb{E}_M[\delta_\pi^M(\tau)] + \gamma B(s', \pi(s')) - B(s, a). \quad (7.29)$$

Consequently,  $\text{Bias}(\mathbb{E}_\theta[\delta(\theta, \tau) \mid \tau]) = \gamma B(s', \pi(s')) - B(s, a)$  and for this bias to be less than  $\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)]) = B(s, a)$ , we require  $|\gamma B(s', \pi(s')) - B(s, a)| < |B(s, a)|$ . Let  $\rho = B(s', \pi(s'))/B(s, a)$  and write  $|(\gamma\rho - 1)B(s, a)| < |B(s, a)|$  from which it follows that for this to hold true, we must have  $\rho \in (0, 2/\gamma)$ , as to be proved. ■

We now turn to characterising the conditions under which  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  enjoys a smaller bias than  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . Because the variance term involves squaring

the TD-error, we must place some restrictions on the expected behaviour of the  $Q$ -function to bound the bias. First, as with  $B$ , let  $C : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  denote the bias of  $\mathbb{E}_\theta[Q_\theta^2]$  for any tuple  $(s, a) \in \mathcal{S} \times \mathcal{A}$ , so that  $\text{Bias}(\mathbb{E}_\theta[Q_\theta(s, a)^2]) = C(s, a)$ . Similarly, let  $D : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  denote the bias of  $\mathbb{E}_\theta[Q_\theta(s', \pi(s'))Q_\theta(s, a)]$  for any transition  $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ .

**Lemma 7.5** (Bias of TD error variance). *For any  $\tau$  and any  $p(M)$ , given  $p(\theta)$ , define relative bias ratios*

$$\begin{aligned} \rho &= \frac{B(s', \pi(s'))}{B(s, a)}, \quad \phi = \frac{C(s', \pi(s'))}{C(s, a)}, \\ \kappa &= \frac{D(s, a, s')}{C(s, a)}, \quad \alpha = \frac{\mathbb{E}_M[Q_\pi^M(s', \pi(s'))]}{\mathbb{E}_M[Q_\pi^M(s, a)]}. \end{aligned} \quad (7.30)$$

There exists  $\rho \approx 1$ ,  $\phi \approx 1$ ,  $\kappa \approx 1$ ,  $\alpha \approx 1$  such that  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  have less bias than  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . In particular, if  $\rho = \phi = \kappa = \alpha = 1$ , then

$$\begin{aligned} |\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| &= |(\gamma - 1)^2 \text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])| \\ &< |\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])|. \end{aligned} \quad (7.31)$$

Further, if  $\rho = 1/\gamma$ ,  $\kappa = 1/\gamma$ ,  $\phi = 1/\gamma^2$ , then  $|\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| = 0$  for any  $\alpha$ .

*Proof.* We begin by characterising the bias of  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . Write

$$\mathbb{V}_\theta[Q_\theta(s, a)] = \mathbb{E}_\theta[Q(s, a)^2] - \mathbb{E}_\theta[Q(s, a)]^2 \quad (7.32)$$

$$= \mathbb{E}_M[Q_\pi^M(s, a)^2] + C(s, a) - \left( \mathbb{E}_M[Q_\pi^M(s, a)] + B(s, a) \right)^2. \quad (7.33)$$

The squared term expands as

$$\begin{aligned} \left( \mathbb{E}_M[Q_\pi^M(s, a)] + B(s, a) \right)^2 &= \\ \mathbb{E}_M[Q_\pi^M(s, a)]^2 + 2\mathbb{E}_M[Q_\pi^M(s, a)] B(s, a) + B(s, a)^2. \end{aligned} \quad (7.34)$$

Let  $A(s, a) = \mathbb{E}_M[Q_\pi^M(s, a)] B(s, a)$  and write the bias of  $\mathbb{V}_\theta[Q_\theta(s, a)]$  as

$$\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)]) = C(s, a) + 2A(s, a) + B(s, a)^2. \quad (7.35)$$

We now turn to  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$ . First note that the reward cancels in this expression:

$$\begin{aligned} \delta(\theta, \tau) - \mathbb{E}_\theta[\delta(\theta, \tau)] &= \\ \gamma Q_\theta(s', \pi(s')) - Q_\theta(s, a) - (\gamma \mathbb{E}_\theta[Q_\theta(s', \pi(s'))] - \mathbb{E}_\theta[Q_\theta(s, a)]) . \end{aligned} \quad (7.36)$$

Denote by  $x_\theta = \gamma Q_\theta(s', \pi(s')) - Q_\theta(s, a)$  with  $\mathbb{E}_\theta[x_\theta] = \gamma \mathbb{E}_\theta[Q_\theta(s', \pi(s'))] - \mathbb{E}_\theta[Q_\theta(s, a)]$ . Write

$$\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau] = \mathbb{E}_\theta \left[ (\delta(\theta, \tau) - \mathbb{E}_\theta[\delta(\theta, \tau)])^2 \right] \quad (7.37)$$

$$= \mathbb{E}_\theta \left[ (x_\theta - \mathbb{E}_\theta[x_\theta])^2 \right] \quad (7.38)$$

$$= \mathbb{E}_\theta [x_\theta^2] - \mathbb{E}_\theta[x_\theta]^2 \quad (7.39)$$

$$= \mathbb{E}_\theta \left[ (\gamma Q_\theta(s', \pi(s')) - Q_\theta(s, a))^2 \right] \quad (7.40)$$

$$- (\gamma \mathbb{E}_\theta[Q_\theta(s', \pi(s'))] - \mathbb{E}_\theta[Q_\theta(s, a)])^2. \quad (7.41)$$

Eq. 7.38 uses Eq. 7.36 and Eq. 7.40 substitutes back for  $x_\theta$ . We consider each term in the last expression in turn. For the first term,  $\mathbb{E}_\theta \left[ (\gamma Q_\theta(s', \pi(s')) - Q_\theta(s, a))^2 \right]$ , expanding the square yields

$$\gamma^2 \mathbb{E}_\theta [Q_\theta(s', \pi(s'))^2] - 2\gamma \mathbb{E}_\theta [Q_\theta(s', \pi(s')) Q_\theta(s, a)] + \mathbb{E}_\theta [Q_\theta(s, a)^2]. \quad (7.42)$$

From this, we obtain the bias as

$$\begin{aligned} \text{Bias} \left( \mathbb{E}_\theta \left[ (\gamma Q_\theta(s', \pi(s')) - Q_\theta(s, a))^2 \right] \right) &= \\ &= \gamma^2 C(s', \pi(s')) - 2\gamma D(s, a, s') + C(s, a) \end{aligned} \quad (7.43)$$

$$= (\gamma^2 \phi - 2\gamma \kappa + 1) C(s, a). \quad (7.44)$$

We can compare this term to  $C(s, a)$  in the bias of  $\mathbb{V}_\theta[Q_\theta(s, a)]$  (Eq. 7.35). For the bias term in Eq. 7.44 to be smaller, we require  $|(\gamma^2 \phi - 2\gamma \kappa + 1) C(s, a)| < |C(s, a)|$  from which it follows that  $(\gamma^2 \phi - 2\gamma \kappa + 1) \in (-1, 1)$ . In terms of  $\phi$ , this means

$$\phi \in \left( \frac{2k\gamma - 2}{\gamma^2}, \frac{2k}{\gamma} \right). \quad (7.45)$$

If the bias term  $D$  is close to  $C$  ( $\kappa \approx 1$ ), this is approximately the same condition as for  $\rho$  in Lemma 7.4. Generally, as  $\kappa$  grows large,  $\phi$  must grow small and vice-versa. The gist of this requirement is that the biases should be relatively balanced  $\kappa \approx \phi \approx 1$ .

For the second term in Eq. 7.40, recall that  $\mathbb{E}_\theta[Q_\theta(s', \pi(s'))] = \mathbb{E}_M[Q_\pi^M(s', \pi(s'))] + B(s', \pi(s'))$  and  $\mathbb{E}_\theta[Q_\theta(s, a)] = \mathbb{E}_M[Q_\pi^M(s, a)] + B(s, a)$ . We have

$$\begin{aligned} (\mathbb{E}_\theta[Q_\theta(s', \pi(s'))] - \mathbb{E}_\theta[Q_\theta(s, a)])^2 = \\ \left( (\gamma\alpha - 1)\mathbb{E}_M[Q_\pi^M(s, a)] + (\gamma\rho - 1)B(s, a) \right)^2, \end{aligned} \quad (7.46)$$

where  $\alpha = \mathbb{E}_M[Q_\pi^M(s', \pi(s'))] / \mathbb{E}_M[Q_\pi^M(s, a)]$ . Expanding the square,

$$\begin{aligned} (\mathbb{E}_\theta[Q_\theta(s', \pi(s'))] - \mathbb{E}_\theta[Q_\theta(s, a)])^2 = \\ (\gamma\alpha - 1)^2 \mathbb{E}_M[Q_\pi^M(s, a)]^2 \\ + 2(\gamma\alpha - 1)(\gamma\rho - 1) \mathbb{E}_M[Q_\pi^M(s, a)] B(s, a) \\ + (\gamma\rho - 1)^2 B(s, a)^2. \end{aligned} \quad (7.47)$$

Then from Eq. 7.31;  $(\mathbb{E}_M[Q_\pi^M(s', \pi(s'))] - \mathbb{E}_M[Q_\pi^M(s, a)])^2 = (\gamma\alpha - 1)^2 \mathbb{E}_M[Q_\pi^M(s, a)]^2$  and so the bias of  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  can be written as

$$\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]) = w_1(\phi, \kappa)C(s, a) + w_2(\alpha, \rho)2A(s, a) + w_3(\rho)B(s, a)^2 \quad (7.48)$$

where  $w_1(\phi, \kappa) = (\gamma^2\phi - 2\gamma\kappa + 1)$ ,  $w_2(\alpha, \rho) = (\gamma\alpha - 1)(\gamma\rho - 1)$ ,  $w_3(\rho) = (\gamma\rho - 1)^2$ . Note that the bias in Eq. 7.48 involves the same terms as the bias of  $\mathbb{V}_\theta[Q_\theta(s, a)]$  (Eq. 7.35) but are weighted. Hence, there always exist a set of weights such that  $|\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| < |\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])|$ . In particular, if  $\rho = 1/\gamma$ ,  $\kappa = 1/\gamma$ ,  $\phi = 1/\gamma^2$ , then  $\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]) = 0$  for any  $\alpha$ . Further, if  $\rho = \alpha = \kappa = \phi = 1$ , then we have that  $w_1(\phi, \kappa) = w_2(\alpha, \rho) = w_3(\rho) = (\gamma - 1)^2$  and so

$$\begin{aligned} |\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| &= |(\gamma - 1)^2 \text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])| \\ &< |\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])|, \end{aligned} \quad (7.49)$$

as desired. ■

**Theorem 7.2.** *For any  $\tau$  and any  $p(M)$ , given  $p(\theta)$ , if  $\rho \in (0, 2/\gamma)$ , then  $\mathbb{E}_\delta[\delta \mid \tau]$  has lower bias than  $\mathbb{E}_\theta[Q_\theta(s, a)]$ . If  $\rho \in (0, 2/\gamma)$ , then  $\mathbb{E}_\delta[\delta \mid \tau]$  has lower bias than  $\mathbb{E}_\theta[Q_\theta(s, a)]$ . Moreover, if  $\rho = 1/\gamma$ , then  $\mathbb{E}_\delta[\delta \mid \tau]$  is unbiased. If  $\rho \in (0, 2/\gamma)$ ,  $\alpha \in (0, 2/\gamma)$ ,  $\kappa \in (0, 2/\gamma)$  and  $\phi \in (1 - 2\gamma\kappa, (2/\gamma)^2)$ , then  $\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau]$  have less bias than  $\mathbb{V}_\theta[Q_\theta(s, a)]$ . In particular, if  $\rho = \phi = \kappa = \alpha = 1$ , then  $|\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| = |(\gamma - 1)^2 \text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])| < |\text{Bias}(\mathbb{V}_\theta[Q_\theta(s, a)])|$ . Further, if  $\rho = 1/\gamma$ ,  $\kappa = 1/\gamma$ ,  $\phi = 1/\gamma^2$ , then  $|\text{Bias}(\mathbb{V}_\theta[\delta(\theta, \tau) \mid \tau])| = 0$  for any  $\alpha$ .*

*Proof.* The first part follows from Lemma 7.4, the second from Lemma 7.5. ■

## 7.C Binary Tree MDP

In this Section, we make a direct comparison between the Bootstrapped DQN and TDU on the Binary Tree MDP introduced by Janz et al. (2019). In this MDP, the agent has two actions in every state. One action terminates the episode with 0 reward while the other moves the agent one step further up the tree. At the final branch, one leaf yields a reward of 1. Which action terminates the episode and which moves the agent to the next branch is randomly chosen per branch, so that the agent must learn an action map for each branch separately. This is a similar environment to Deep Sea, but simpler in that an episode terminates upon taking a wrong action and the agent does not receive a small negative reward for taking the correct action. We include the Binary Tree MDP experiment to compare the scaling property of TDU as compared to TDU on a well-known benchmark.

We use the default Bsuite implementation<sup>11</sup> of the bootstrapped DQN, with the default architecture and hyper-parameters from the published baseline, reported in Table 7.2. The agent is composed of a two-layer MLP with RELU activations that approximate  $Q(s, a)$  and is trained using experience replay. In the case of the bootstrapped DQN, all ensemble members learn from a shared replay buffer with bootstrapped data sampling, where each member  $Q_{\theta^k}$  is a separate MLP (no parameter sharing) that is regressed towards separate target networks. We use Adam (Kingma & Ba, 2015) and update target networks periodically (Table 7.2).

We run 5 seeds per tree-depth, for depths  $L \in \{10, 20, \dots, 250\}$  and report mean performance in Figure 7.4. Our results are in line with those of Janz et al. (2019), differences are due to how many gradient steps are taken per episode (our results are between the reported scores for the 1× and 25× versions of the bootstrapped DQN). We observe a clear beneficial effect of including TDU, even for small values of  $\beta$ . Further, we note that performance is largely monotonically increasing in  $\beta$ , further demonstrating that the TDU signal is well-behaved and robust to hyper-parameter values.

We study the properties of TDU in Figure 7.5, which reports performance without prior functions ( $\lambda = 0$ ). We vary  $\beta$  and the number of exploration value functions  $N$ . The total number of value functions is fixed at 20, and so varying  $N$  is equivalent to varying the degree of exploration. We note that  $N$  has a similar effect to  $\beta$ , but has a slightly larger tendency to induce over-exploration for large values of  $N$ .

<sup>11</sup><https://github.com/deepmind/bsuite/tree/master/bsuite/baselines/jax/bootdqn>.

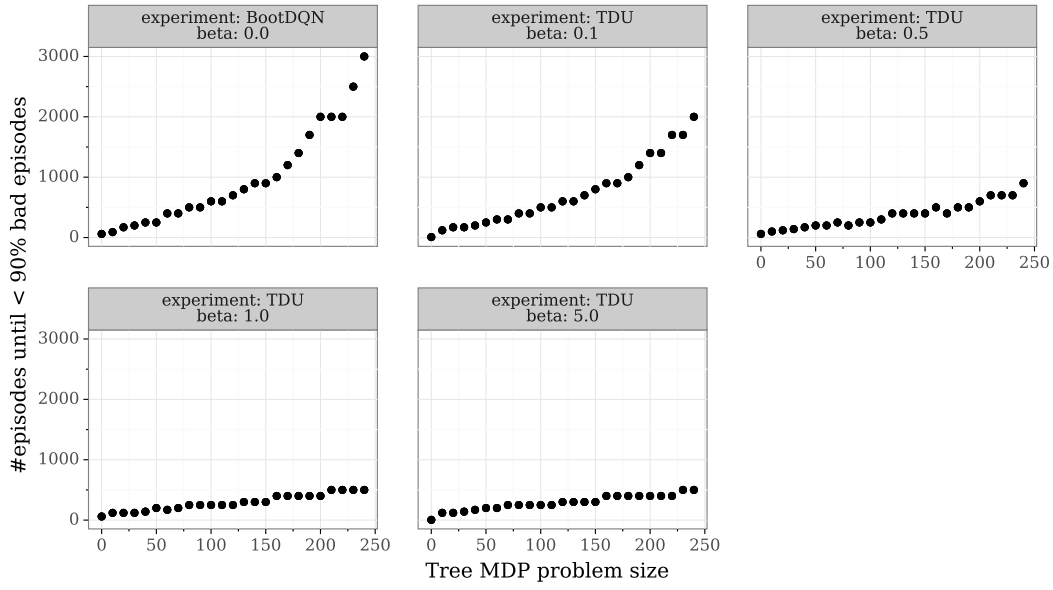


Figure 7.4: Performance on Binary Tree MDP. *Top left*: BootDQN ( $\beta = 0$ ). *Others*: TDU with varying strengths of intrinsic reward ( $\beta > 0$ ). Results with prior strength  $\lambda = 3$ . Mean performance over 5 seeds for each tree depth.

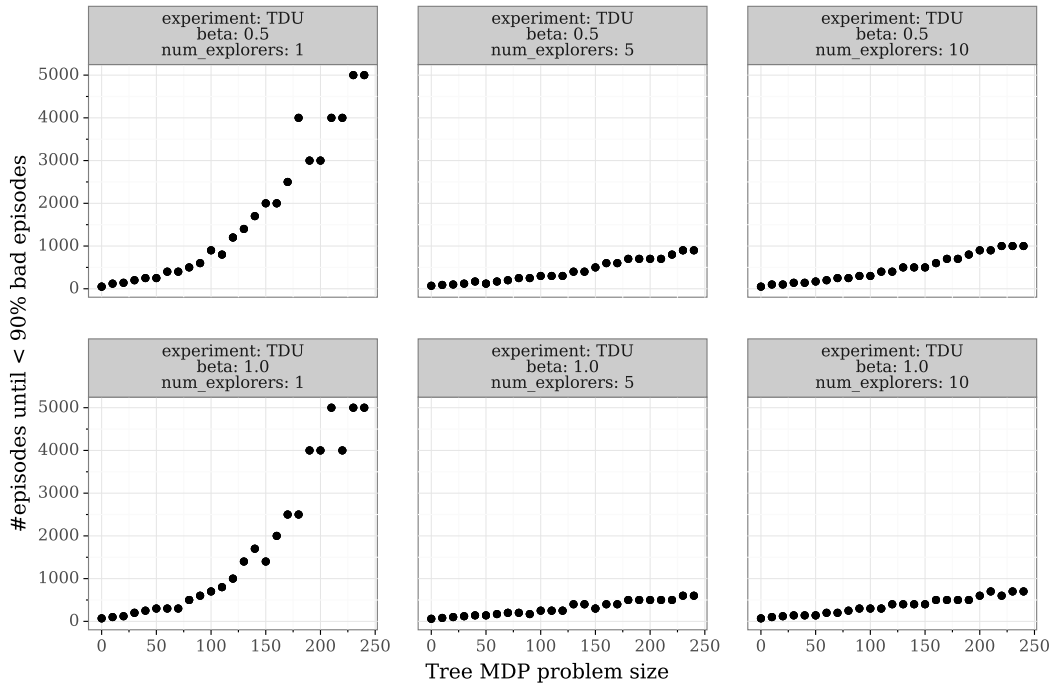


Figure 7.5: Hyper-parameter sensitivity analysis on Binary Tree MDP. *Top*: Sweep over number of exploration policies ( $N$ ) for  $\beta = 0.5$ . *Top*: Sweep over number of exploration value functions ( $N$ ) for  $\beta = 1$ . All results without prior functions ( $\lambda = 0$ ). Mean performance over 5 seeds for each tree depth.



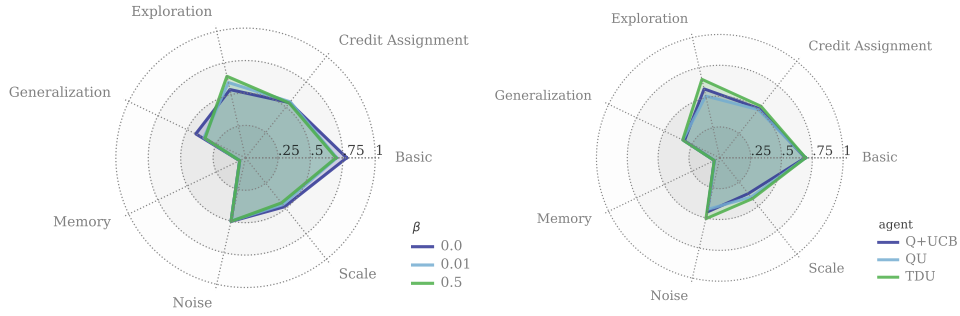


Figure 7.6: Overall performance scores on Bsuite. *Left*: Effect of varying  $\beta$ . *Right*: comparison of TDU to exploration under  $\sigma = \sigma(Q)$  as intrinsic reward (QU) or as an immediate bonus (Q+UCB).

## 7.D Behaviour Suite

From [Osband et al. \(2020\)](#): “The Behaviour Suite for Reinforcement Learning (Bsuite) is a collection of carefully-designed experiments that investigate core capabilities of a reinforcement learning agent . The aim of the Bsuite project is to collect clear, informative and scalable problems that capture key issues in the design of efficient and general learning algorithms and study agent behaviour through their performance on these shared benchmarks.”

### 7.D.1 Agents and Hyper-Parameters

All baselines use the default Bsuite DQN implementation<sup>12</sup>. We use the default architecture and hyper-parameters from the published baseline, reported in Table 7.2, and sweep over algorithm-specific hyper-parameters, reported in Table 7.3. The agent is composed of a two-layer MLP with RELU activations that approximate  $Q(s, a)$  and is trained using experience replay. In the case of the bootstrapped DQN, all ensemble members learn from a shared replay buffer with bootstrapped data sampling, where each member  $Q_{\theta^k}$  is a separate MLP (no parameter sharing) that is regressed towards separate target networks. We use Adam ([Kingma & Ba, 2015](#)) and update target networks periodically (Table 7.2).

**QEX** Uses two networks  $Q_\theta$  and  $Q_\phi$ , where  $Q_\theta$  is trained to maximise the extrinsic reward, while  $Q_\phi$  is trained to maximise the absolute TD-error of  $Q_\theta$  ([Simmons-Edler et al., 2019](#)). In contrast to TDU, the intrinsic reward is given as a point estimate of the TD-error for a given transition, and thus cannot be interpreted as measuring uncertainty as such.

<sup>12</sup><https://github.com/deepmind/bsuite/tree/master/bsuite/baselines/jax/dqn>.

Table 7.2: Hyper-parameters for Bsuite.

discount factor ( $\gamma$ )	0.99
batch size	32
num hidden layers	2
hidden layer sizes	[64, 64]
ensemble size	20
learning rate	0.001
mask prob	1.0
replay size	10000
env steps per gradient step	1
env steps per target update	4

**CTS** Implements a count-based reward defined by  $i(s, a, \mathcal{H}) = (N(s, a, \mathcal{H}) + 0.01)^{-1/2}$ , where  $\mathcal{H}$  is the history and  $N(s, a, \mathcal{H}) = \sum_{\tau \in \mathcal{H}} \mathbf{1}_{(s,a) \in \tau}$  is the number of times  $(s, a)$  has appeared in a transition  $\tau := (s, a, r, s')$ . This intrinsic reward is added to the extrinsic reward to form an augmented reward  $\tilde{r} = r + \beta i$  used to train a DQN agent (Bellemare et al., 2016).

**RND** Uses two auxiliary networks  $f_{\vartheta}$  and  $f_{\tilde{\vartheta}}$  that map a state into vectors  $x = f_{\vartheta}(s)$  and  $\tilde{x} = f_{\tilde{\vartheta}}(s)$ ,  $x, \tilde{x} \in \mathbb{R}^m$ . While  $\tilde{\vartheta}$  is a random parameter vector that is fixed throughout,  $\vartheta$  is trained to minimise the mean squared error  $i(s) = \|x - \tilde{x}\|^2$ . This error is simultaneously used as an intrinsic reward in the augmented reward function  $\tilde{r}(s, a) = r(s, a) + \beta i(s)$  and is used to train a DQN agent. Following Burda et al. (2018b), we normalise intrinsic rewards by an exponential moving average of the mean and the standard deviation that are being updated with batch statistics (with decay  $\alpha$ ).

**BDQN** Trains an ensemble  $\mathcal{Q} = \{Q_{\theta^k}\}_{k=1}^K$  of DQNs (Osband et al., 2016a). At the start of each episode, one DQN is randomly chosen from which a greedy policy is derived. Data collected is placed in a shared replay memory, and all ensemble members have some probability  $\rho$  of training on any transition in the replay. Each ensemble member has its own target network. In addition, each DQN is augmented with a random prior function  $f_{\tilde{\vartheta}}$ , where  $\tilde{\vartheta}$  is a fixed parameter vector that is randomly sampled at the start of training. Each DQN is defined by  $Q_{\theta^k} + \lambda f_{\tilde{\vartheta}^k}$ , where  $\lambda$  is a hyper-parameter regulating the scale of the prior. Note that the target network uses a distinct prior function.

**SU** Decomposes the DQN as  $Q_{\theta}(s, a) = w^T \psi_{\vartheta}(s, a)$ . The parameters  $\vartheta$  are trained to satisfy the Success Feature identity while  $w$  is learned using Bayesian linear regression; at the start of each episode, a new  $w$  is sampled from the

Table 7.3: Hyper-parameter grid searches for Bsuite. Best values in bold.

Algorithm	Hyper-parameter	Sweep set
Q-Explore	Intrinsic reward scale ( $\beta$ )	$\{10^{-3}, 0.01, 0.1, 1, 5, \mathbf{10}, 10^2, 10^3\}$
CTS	Intrinsic reward scale ( $\beta$ )	$\{10^{-3}, 0.01, \mathbf{0.1}, 1, 5, 10, 10^2, 10^3\}$
RND	Intrinsic reward scale ( $\beta$ )	$\{0.01, 0.1, 0.5, 1, 5, \mathbf{10}, 100\}$
	$x$ -dim ( $m$ )	$\{\mathbf{10}, 64, 128\}$
	Moving average decay ( $\alpha$ )	$\{0.9, 0.99, \mathbf{0.999}\}$
	Normalise intrinsic reward	$\{\mathbf{True}, \text{False}\}$
BDQN	Prior scale ( $\lambda$ )	$\{0, 1, \mathbf{3}, 5, 10, 50, 100\}$
SU	Hidden size	$\{20, \mathbf{64}\}$
	Likelihood variance ( $\beta$ )	$\{0.01, 0.1, 1, \mathbf{10}, 10^2\}$
	Prior variance ( $\theta$ )	$\{\mathbf{0.001}, 0.01, 0.1, 1, 10, 10^3\}$
NNS	Noise scale ( $\beta$ )	$\{0.01, 0.1, 1, \mathbf{10}, 100\}$
TDU	Prior scale ( $\lambda$ )	$\{0, 1, \mathbf{3}\}$
	Intrinsic reward scale ( $\beta$ )	$\{10^{-3}, 0.01, 0.1, \mathbf{1}, 5, 10\}$

posterior  $p(w \mid \text{history})$  (Janz et al., 2019).<sup>13</sup>

**NNS** NoisyNets replace feed-forward layers  $Wx + b$  by a noisy equivalent  $(W + \Sigma \odot \epsilon^W)x + (b + \sigma \odot \epsilon^b)$ , where  $\odot$  is element-wise multiplication;  $\epsilon_{ij}^W \sim \mathcal{N}(0, \beta)$  and  $\epsilon_i^b \sim \mathcal{N}(0, \beta)$  are white noise of the same size as  $W$  and  $b$ , respectively. The set  $(W, \Sigma, b, \sigma)$  are learnable parameters that are trained on the normal TD-error, but with the noise vector re-sampled after every optimisation step. Following Fortunato et al. (2018), sample noise separately for the target and the online network.

**TDU** We fix the number of explorers to 10 (half of the number of value functions in the ensemble), which roughly corresponds to randomly sampling between a reward-maximising policy and an exploration policy. Our experiments can be replicated by running the TDU agent implemented in Algorithm 7.5 in the Bsuite GitHub repository.<sup>14</sup>

## 7.D.2 TDU Experiments

**Effect of TDU** Our main experiment sweeps over  $\beta$  to study the effect of increasing the TDU exploration bonus, with  $\beta \in \{0, 0.01, 0.1, 0.5, 1, 2, 3, 5\}$ ;

<sup>13</sup>See [https://github.com/DavidJanz/successor\\_uncertainties\\_tabular](https://github.com/DavidJanz/successor_uncertainties_tabular).

<sup>14</sup><https://github.com/deepmind/bsuite/blob/master/bsuite/baselines/jax>.

$\beta = 0$  corresponds to default bootstrapped DQN. We find that  $\beta$  reflects the exploitation-exploration trade-off: increasing  $\beta$  leads to better performance on exploration tasks (see main paper) but typically leads to worse performance on tasks that do not require further exploration beyond  $\epsilon$ -greedy (Figure 7.6). In particular, we find that  $\beta > 0$  prevents the agent from learning on Mountain Car, but otherwise retains performance on non-exploration tasks. Figure 7.7 provides an in-depth comparison per game.

Because  $\sigma$  is a principled measure of concentration in the distribution  $p(\delta \mid s, a, r, s')$ ,  $\beta$  can be interpreted as specifying how much of the tail of the distribution the agent should care about. The higher we set  $\beta$ , the greater the agent’s sensitivity to the tail-end of its uncertainty estimate. Thus, there is no reason in general to believe that a single  $\beta$  should fit all environments, and recent advances in multi-policy learning (Schaul et al., 2019; Zahavy et al., 2020; Puigdomènech Badia et al., 2020) suggests that a promising avenue for further research is to incorporate mechanisms that allow either  $\beta$  to dynamically adapt or the sampling probability over policies. To provide concrete evidence to that effect, we conduct an ablation study that uses bandit policy sampling below.

**Effect of prior functions** We study the inter-relationship between additive prior functions (Osband et al., 2019) and TDU. We sweep over  $\lambda \in [0, 1, 3]$ , where prior functions define value function estimates by  $Q^k = Q_{\theta^k} + \lambda P^k$  for some random network  $P^k$ . Thus,  $\lambda = 0$  implies no prior function. We find a general synergistic relationship; increasing  $\lambda$  improves performance (both with and without TDU), and for a given level of  $\lambda$ , performance on exploration tasks improve for any  $\beta > 0$ . It should be noted that these effects do not materialise as clearly in our Atari settings, where we find no conclusive evidence to support  $\lambda > 0$  under TDU.

**Ablation: exploration under non-TD signals** To empirically support theoretical underpinnings of TDU, we conduct an ablation study where  $\sigma$  is re-defined as the standard deviation over value estimates:

$$\sigma(Q) := \sqrt{\frac{1}{K-1} \sum_{k=1}^K Q^k - \bar{Q}}. \quad (7.50)$$

In contrast to TDU, this signal does not condition on the future and consequently is a biased estimate of epistemic uncertainty. We apply this signal both as in intrinsic reward (QU), as in TDU, and as an UCB-style exploration bonus (Q+UCB), where  $\sigma$  is instead applied while acting by defining a policy

by  $\pi(\cdot) = \arg \max_a Q(\cdot, a) + \beta \sigma(Q; \cdot, a)$ . Note that TDU cannot be applied in this way because the TDU exploration signal depends on  $r$  and  $s'$ . We tune each baseline over the same set of  $\beta$  values as above (incidentally, these coincide to  $\beta = 1$ ) and report best results in Figure 7.6. We find that either alternative is strictly worse than TDU. They suffer a significant drop in performance on exploration tasks, but are also less able to handle noise and reward scaling. These findings are in line with theory, as noise makes bias more likely in uncertainty estimates over  $Q$ . Moreover, biased uncertainty estimates will negatively impact the agent’s ability to explore.

**Ablation: bandit policy sampling** Our main results indicate, unsurprisingly, that different environments require different emphasis on exploration. To test this more concretely, in this experiment we replace uniform policy sampling with the UCB1 bandit algorithm. However, in contrast to that example, where UCB1 is used to take actions, here it is used to select a policy for the next episode. We treat each  $N + K$  value function as an “arm” and estimate its mean reward  $V^k \approx \mathbb{E}_{\pi^k}[r]$ , where the expectation is with respect to rewards  $r$  collected under policy  $\pi^k(\cdot) = \arg \max_a Q^k(\cdot, a)$ . The mean reward is estimated as the running average

$$V^k(n) = \frac{1}{n(k)} \sum_{i=1}^{n(k)} r_i, \quad (7.51)$$

where  $n(k)$  is the number of environment steps for which policy  $\pi^k$  has been used and  $r_i$  are the observed rewards under policy  $\pi^k$ . Prior to an episode, we choose a policy to act under according to:  $\arg \max_{k=1, \dots, N+K} V^k(n) + \eta \sqrt{\log n / n(k)}$ , where  $n$  is the total number of environment steps taken so far and  $\eta$  is a hyper-parameter that we tune. As in the bandit example, this sampling strategy biases selection towards policies that currently collect higher reward, but balances sampling by a count-based exploration bonus that encourages the agent to eventually try all policies. This bandit mechanism is very simple as our purpose is to test whether some form of adaptive sampling can provide benefits; more sophisticated methods (e.g. Schaul et al., 2019) can yield further gains.

We report full results in Figure 7.7; we use  $\beta = 1$  and tune  $\eta \in \{0.1, 1, 2, 4, 6, 8\}$ . We report results for the hyper-parameter that performed best overall,  $\eta = 8$ , though differences with  $\eta > 4$  are marginal. While TDU does not impact performance negatively in general, in the one case where it does—Mountain Car—introducing a bandit to adapt exploration can largely recover performance. The bandit yields further gains in dense reward settings, such as in Cartpole

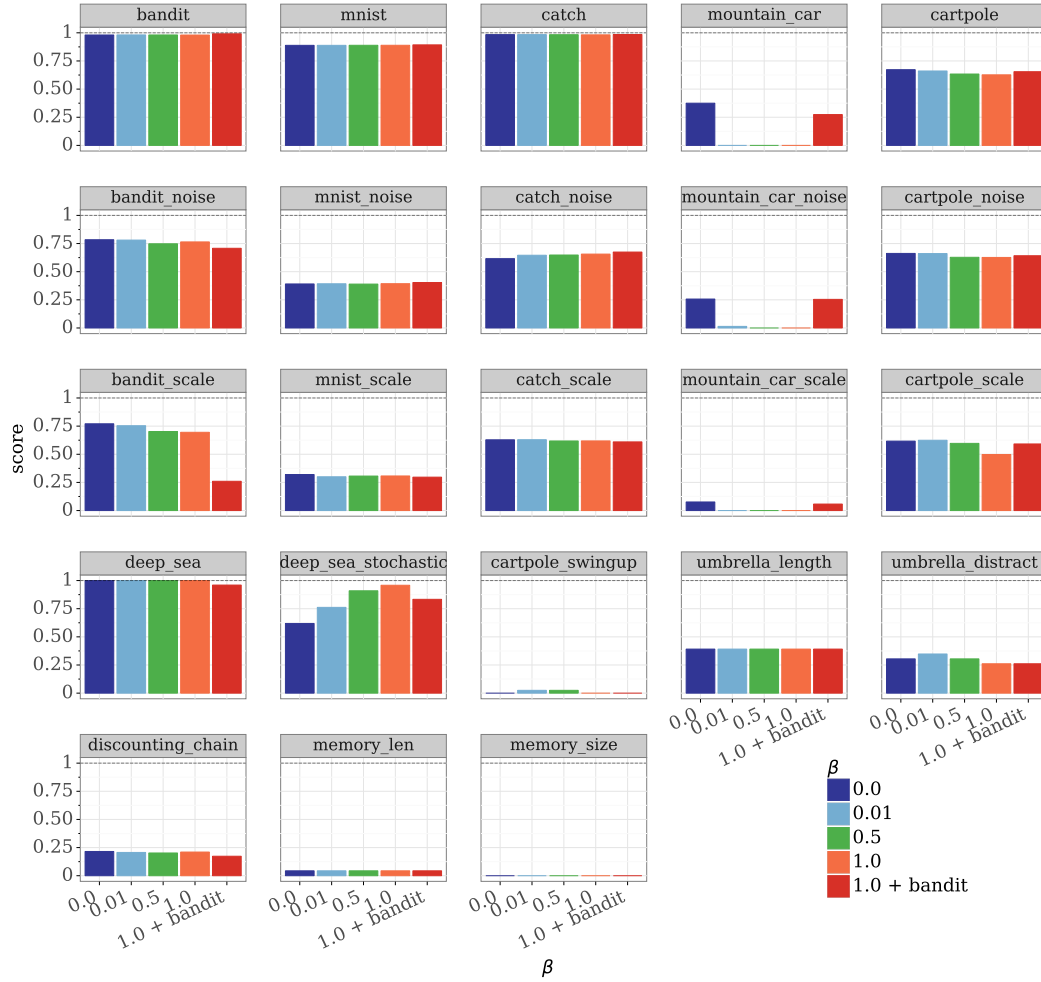


Figure 7.7: Bsuite per-task results. Results reported for different values of  $\beta$  with prior  $\lambda = 3$ . We also report results under UCB1 policy sampling (“bandit”) for  $\beta = 1, \lambda = 3, \eta = 8$ .

and Catch, with an outlying exception in the bandit setting with scaled rewards.

## 7.E Atari with R2D2

### 7.E.1 Bootstrapped R2D2

We augment the R2D2 agent with an ensemble of dueling action-value heads  $Q_i$ . The behavior policy followed by the actors is an  $\epsilon$ -greedy policy as before, but where the greedy action is determined according to a single  $Q_i$  for a fixed length of time (100 actor steps in all of our experiments), before sampling a new  $Q_i$  uniformly at random. The evaluation policy is also  $\epsilon$ -greedy with  $\epsilon = 0.001$ , where the Q-values are averaged only over the exploiter heads.

Table 7.4: R2D2 hyper-parameters.

Ensemble size	10
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate	0.0002
Adam epsilon	0.001
Adam beta1	0.9
Adam beta2	0.999
Adam global clip norm	40
Discount	0.997
Batch size	64
Trace length	80
Replay period	40
Burn in length	20
$\lambda$ for RL loss	0.97
R2D2 reward transformation	$\text{sign}(x) \cdot (\sqrt{ x  + 1} - 1) + 0.001 \cdot x$
Replay capacity (num of sequences)	$1e5$
Replay priority exponent	0.9
Importance sampling exponent	0.6
Minimum sequences to start replay	5000
Actor update period	100
Target Q-network update period	400
Evaluation $\epsilon$	0.001

Each trajectory inserted into the replay buffer is associated with a binary mask indicating which  $Q_i$  will be trained from this data, ensuring that the same mask is used every time the trajectory is sampled. Priorities are computed as in R2D2, except that TD-errors are now averaged over all heads.

Instead of using reward clipping, R2D2 estimates a transformed version of the state-action value function to make it easier to approximate for a neural network. One can define a transformed Bellman operator given any squashing function  $h : \mathbb{R} \rightarrow \mathbb{R}$  that is monotonically increasing and invertible. We use the function  $h : \mathbb{R} \mapsto \mathbb{R}$  defined by

$$h(z) = \text{sign}(z)(\sqrt{|z| + 1} - 1) + \epsilon z, \quad (7.52)$$

$$h^{-1}(z) = \text{sign}(z) \left( \left( \frac{\sqrt{1 + 4\epsilon(|z| + 1 + \epsilon)} - 1}{2\epsilon} \right) - 1 \right), \quad (7.53)$$

for  $\epsilon$  small. In order to compute the TD errors accurately we need to account for the transformation,

$$\delta(\theta, s, a, r, s') := \gamma h^{-1}(Q_\theta(s', \pi(s'))) + r - h^{-1}(Q_\theta(s, a)). \quad (7.54)$$

Similarly, at evaluation time we need to apply  $h^{-1}$  to the output of each head before averaging.

When making use of a prior we use the form  $Q^k = Q_\theta^k + \lambda P^k$ , where  $P^k$  is of the same architecture as the  $Q_\theta^k$  network, but with the widths of all layers cut to reduce computational cost. Finally, instead of n-step returns we utilise  $Q(\lambda)$  (Peng & Williams, 1994) as was done in (Guez et al., 2020). In all variants we used the hyper-parameters listed in Table 7.4.

### 7.E.2 Pre-processing

We used the standard pre-process of the frames received from the Arcade Learning Environment.<sup>15</sup> See Table 7.5 for details.

### 7.E.3 Hyper-Parameter Selection

In the distributed setting we have three TDU-specific hyper-parameters to tune namely:  $\beta$ ,  $N$  and the prior weight  $\lambda$ . For our main results, we run each agent across 8 seeds for 20 billions steps. For ablations and hyper-parameter tuning, we ran agents across 3 seeds for 5 billion environment steps on a subset of 8 games: frostbite, gravitar,

Table 7.5: Atari pre-processing hyperparameters.

Max episode length	30 min
Num. action repeats	4
Num. stacked frames	4
Zero discount on life loss	<i>false</i>
Random noops range	30
Sticky actions	<i>false</i>
Frames max pooled	3 and 4
Grayscaled/RGB	Grayscaled
Action set	Full

hero, montezuma\_revenge, ms\_pacman, seaquest, space\_invaders, venture. This subset presents quite a bit of diversity including dense-reward games as well as three hard exploration games: gravitar, montezuma\_revenge and venture. To minimise the computational cost, we started by setting  $\lambda$  and  $N$  while maintaining  $\beta = 1$ . We employed a coarse grid of  $\lambda \in \{0., 0.05, 0.1\}$  and  $N \in \{2, 3, 5\}$ . Figure 7.8 summarises the results in terms of the mean Human Normalised Scores (HNS) across the set. We see that the performance depends on the type of games being evaluated. Specifically, hard exploration games achieve a significantly lower score. Performance does not significantly change with the number of explorers. The largest differences are observed for the exploration games

<sup>15</sup>Publicly available at <https://github.com/mgbellemare/Arcade-Learning-Environment>.



when  $N = 5$ . We select best performing sets of hyper parameters for TDU with and without additive priors: ( $N = 2, \lambda = 0.1$ ) and ( $N = 5, \lambda = 0$ ), respectively.

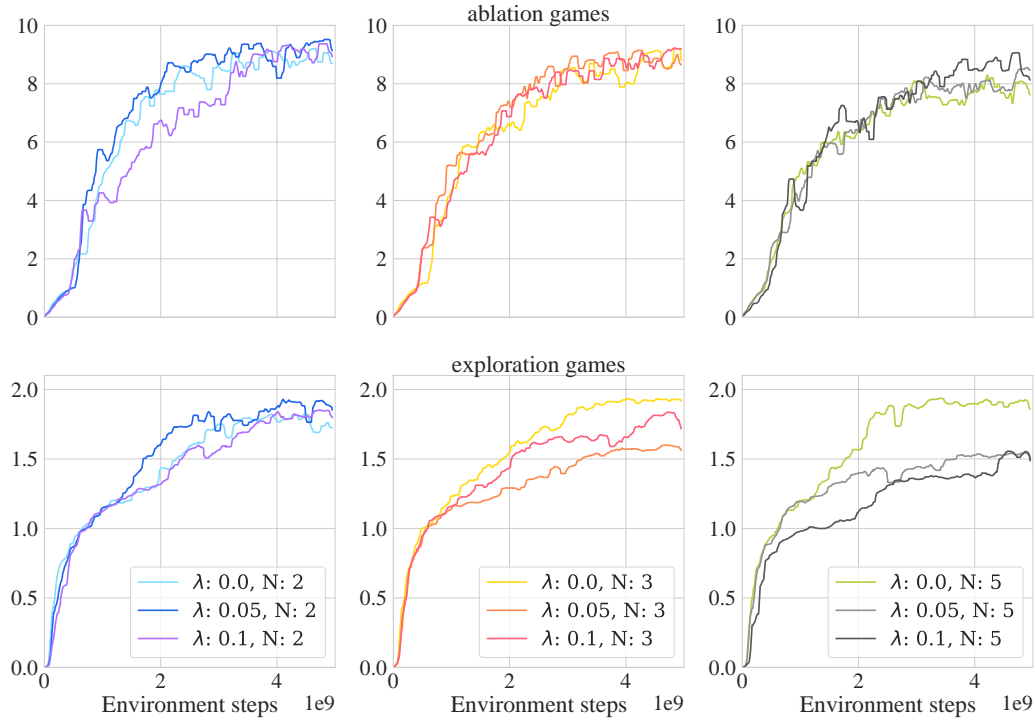


Figure 7.8: Ablation for prior scale,  $\lambda$  and the number of explorers,  $N$ , on the distributed setting. We fix  $\beta = 1$ . Refer to the text for details on the ablation and exploration set of games.

We evaluate the influence of the exploration bonus strength by fixing ( $N = 5, \lambda = 0$ ) and choosing  $\beta \in \{0.1, 1., 2.\}$ . Figure 7.9 summarises the results. The set of dense rewards is composed of the games in the ablation set that are not considered hard exploration games. We observe that larger values of  $\beta$  help on exploration but affect performance on dense reward games. We plot jointly the performance in mean HNS acting when averaging the Q-values for both, the exploiter heads (solid lines) and the explorer heads (dotted lines). We can see that higher strengths for the exploration bonus (higher  $\beta$ ) renders the explorers “uninterested” in the extrinsic rewards, preventing them to converge to exploitative behaviours. This effect is less strong for the hard exploration games. Figure 7.10 we show how this effect manifests itself on the performance on three games: `gravitar`, `space_invaders`, and `hero`. This finding also applies to the evaluation performed on our evaluation using all 57 games in the Atari suite, as shown below. We conjecture that controlling for the strength of the exploration bonus on a per game manner would significantly improve the results. This finding is in line with observations made by (Puigdomènech Badia et al., 2020); combining TDU with adaptive policy sampling (Schaul et al.,

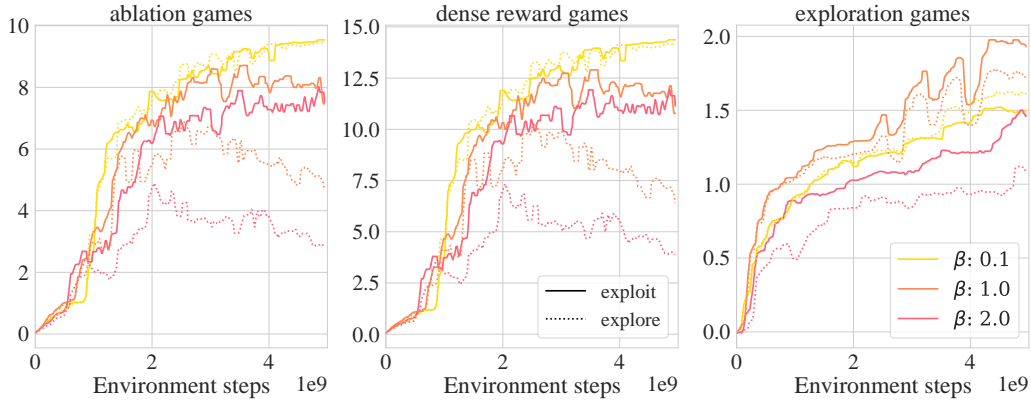


Figure 7.9: Ablation for the exploration bonus strength,  $\beta$ , on the distributed setting. We fix ( $N = 5, \lambda = 0$ ). We report the mean HNS for the ensemble of exploiter (solid lines) and the ensemble of explorers (dotted lines). All runs are average over three seeds per game. Refer to the text for details on ablation and exploration set of games.

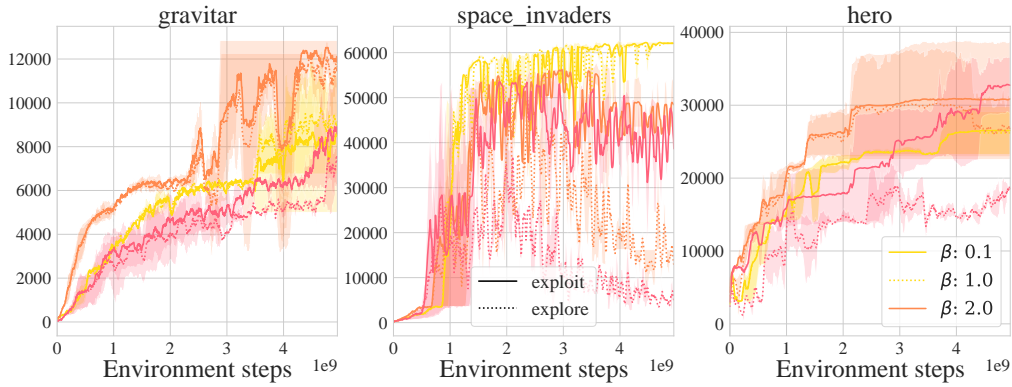


Figure 7.10: Ablation for the exploration bonus strength,  $\beta$ , on the distributed setting. We fix ( $N = 5, \lambda = 0$ ). We report the score on three different games for the ensemble of exploiter (solid lines) and the ensemble of explorers (dotted lines). All runs are average over three seeds per game.

2019) or online hyper-parameter tuning (Xu et al., 2018a; Zahavy et al., 2020) are exciting avenues for future research.

#### 7.E.4 Detailed Results: Main Experiment

In this Section we provide more detailed results from our main experiment in Section 7.5.2. We concentrated our attention on the subset of games that are well-known to pose challenging exploration problems (Machado et al., 2018): montezuma\_revenge, pitfall, private\_eye, solaris, venture, gravitar, and tennis. We also add a varied set of dense reward games.

Figure 7.11 shows the performance for each game. We can see that TDU

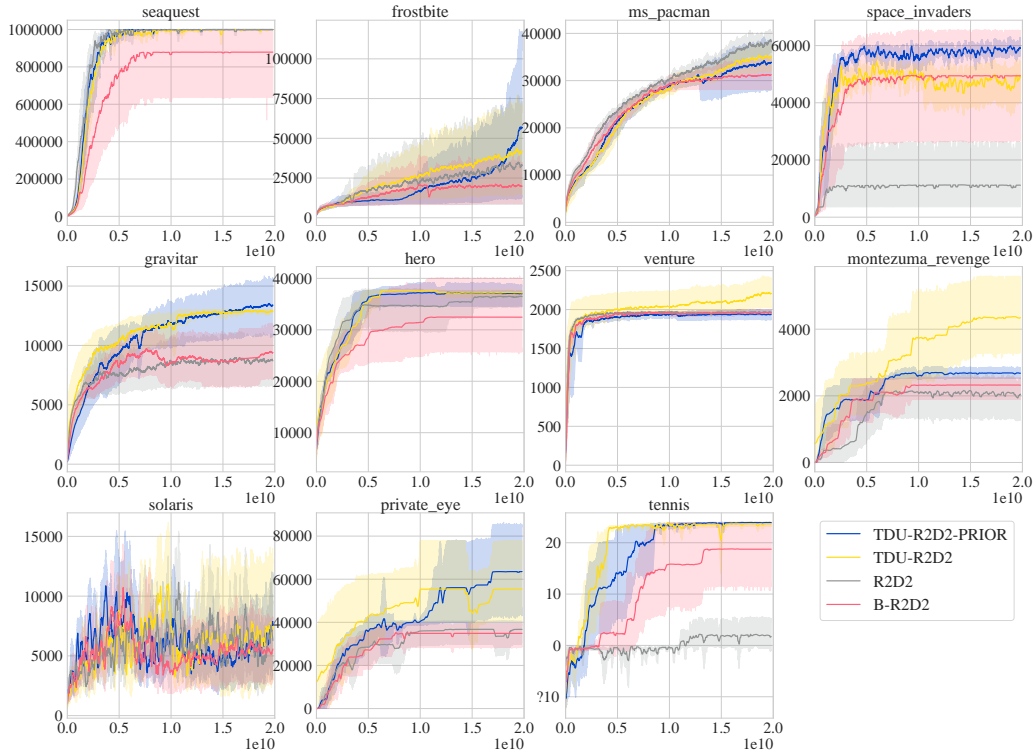


Figure 7.11: Performance on each game in the main experiment in Section 7.5.2. Shading depicts standard deviation over 8 seeds.

always performs on par or better than each of the baselines, leading to significant improvements in data efficiency and final score in games such as `montezuma_revenge`, `private_eye`, `venture`, `gravitar`, and `tennis`. Gains in exploration games can be substantial, and in `montezuma_revenge`, `private_eye`, `venture`, and `gravitar`, TDU without prior functions achieves statistically significant improvements. TDU with prior functions achieve statistically significant improvements on `montezuma_revenge`, `private_eye`, and `gravitar`. Beyond this, both methods improve the rate of convergence on `seaquest` and `tennis`, and achieve higher final mean score. Overall, TDU yields benefits across both dense reward and exploration games, as summarised in Figure 7.12. Note that R2D2’s performance on dense reward games is deflated due to particularly low scores on `space_invaders`. Our results are in line with the original publication, where R2D2 does not show substantial improvements until after 35 Bn steps.

### 7.E.5 Full Atari suite

In this Section we report the performance on all 57 games of the Atari suite. In addition to the two configurations used to obtain the results presented in the main text (reported in Section 5.2), in this Section we included a variant of

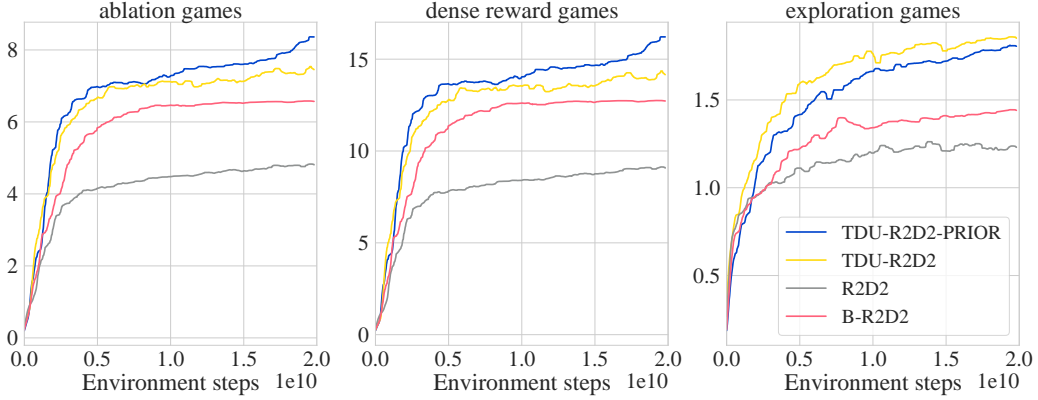


Figure 7.12: Performance across all games in the main experiment in Section 7.5.2. We report mean HNS over the full set of games used in the main experiment, dense reward games, and exploration games. Shading depicts standard deviation over 8 seeds.

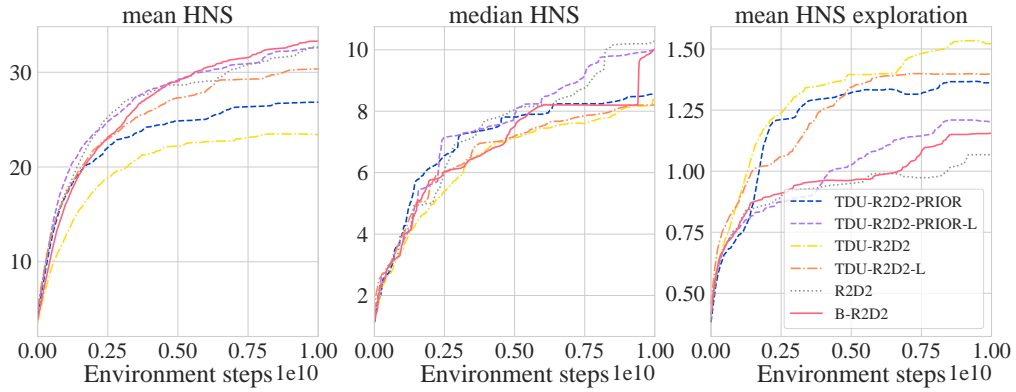


Figure 7.13: Performance over the 57 atari games. We report mean and median HNS over the full suite, and mean HNS over the exploration games.

each of them with lower exploration bonus strength of  $\lambda = 0.1$ . In all figures we refer to these variants by adding an L (for lower  $\lambda$ ) at the end of the name, e.g. TDU-R2D2-L. In Figure 7.13 we report a summary of the results in terms of mean HNS and median HNS for the suite as well as mean HNS restricted to the hard exploration games only. We show the performance on each game in Figure 7.14. Reducing the value of  $\beta$  significantly improves the mean HNS without strongly degrading the performance on the games that are challenging from an exploration standpoint. The difference in performance in terms of mean HNS can be explained by looking at a few high scoring games, for instance: assault, asterix, demon\_attack and gopher (see Figure 7.14). We can see that incorporating priors to TDU is not crucial for achieving high performance in the distributed setting.

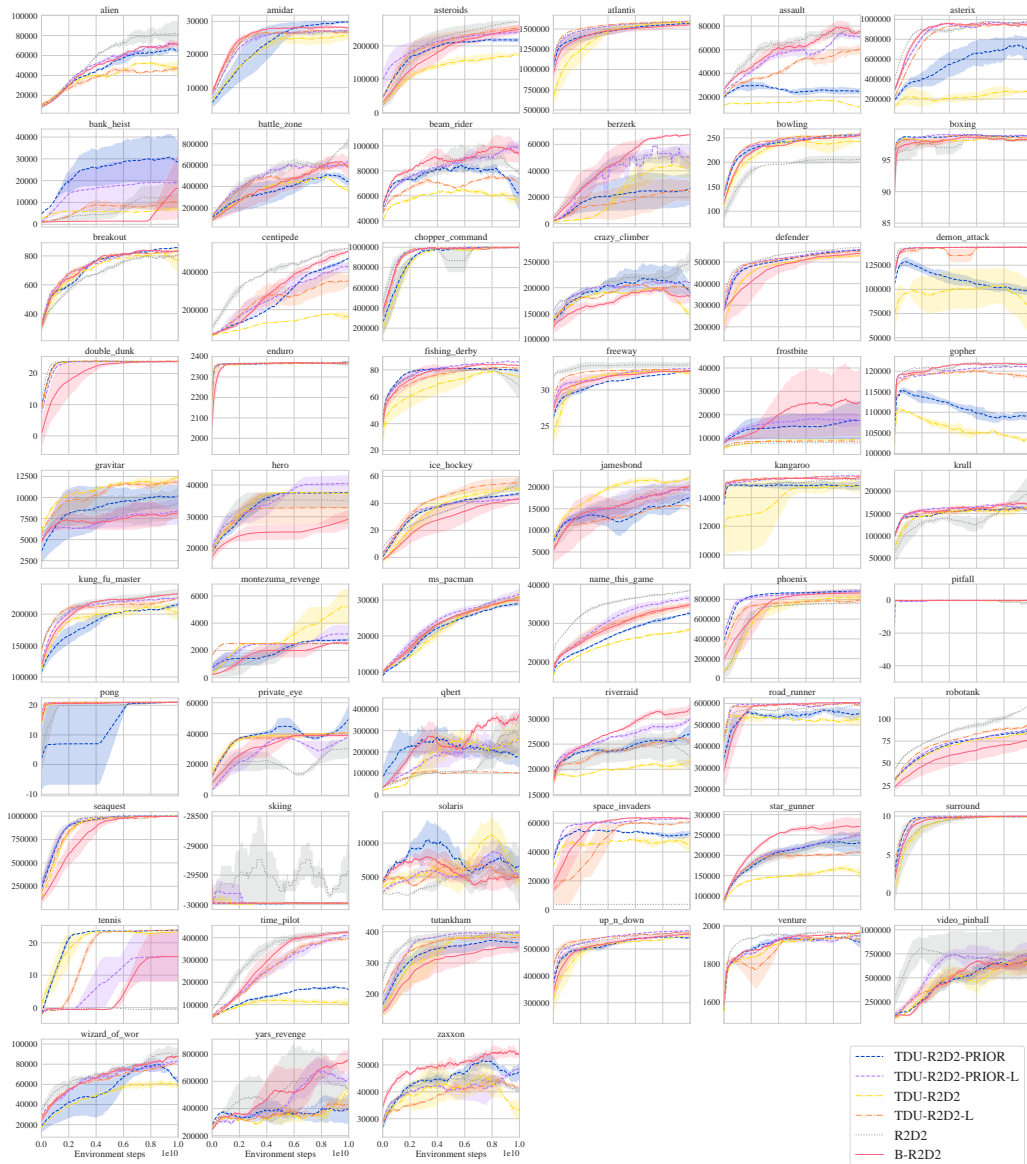


Figure 7.14: Results for each individual game. Shading depicts standard deviation over 3 seeds.

## 8

## Conclusion

---

This thesis has considered ways of producing learning systems that can learn to learn at scale. The contributions in the thesis are well situated within the burgeoning literature on neural network-based meta-learning. This final chapter begins by summarising the point of view taken in the thesis and the analyses and methods that have developed from that. It concludes by outlining contributions of this thesis before turning to limitations of the proposed methods. Finally, it looks ahead to future research needed to overcome these limitations.

## 8.1 Thesis Summary

This thesis has been based on the premise that meta-learning is a critical component for reaching human-level intelligence in an AI (Chapter 1). By synthesising a body of work that aims to define artificial intelligence, a key component was found to be the ability to perform future, unknown problems in a cumulative, never-ending process (Section 1.1). This in turn implies learning must incorporate the adaptive nature of intelligence. A natural way to do so is through the perspective of meta-learning (Chapter 3).

Historically, meta-learning was thought of as a general machine that would take a description of the current model and current experience and produce a new model (Section 3.3). From this perspective, learning and meta-learning are two continuous, intertwined processes. Contemporary meta-learning has instead favoured a sharp distinction between learning a model and meta-learning how to learn that model (Section 3.5). This formalism provides a rigorous framework for applying conventional machine learning protocols (Chapter 2) to meta-learning and has generated substantial momentum for meta-learning research (Chapter 3). Within this space, this thesis is motivated by three key limitations of current meta-learning (Section 1.2): scalability, generalisation, and the need of a task distribution. The contributions of the thesis are steps towards scalable solutions to these limitations.

**Generalisation** Chapter 4 studies the expressive capacity of neural networks and identifies a bottleneck stemming from typical activation functions. As these are predominantly linear over their domains, they reduce the adaptive capacity of neural networks over the data domain. Based on this insight, this work develops a method to adaptively parameterise neural networks and proved that very simple forms of adaptive parameterisation can carry universal representational capacity. Empirically, adaptive parameterisation leads to greater efficiency in a variety of domains; as evidenced by achieving state-of-the-art performance on two common natural language processing datasets and by achieving state-of-the-art performance on a challenging meta-reinforcement learning benchmark (Chapter 6).

**Scalability** Chapter 5 studies the typical meta-learning protocol (Section 3.5) where the goal is to meta-learn over a distribution of tasks. The meta-learner has access to a set of tasks for meta-training and is evaluated on held-out tasks once meta-training is complete. Motivated by the goal to scale beyond few-shot learning, this work proposes an alternative perspective of gradient-based meta-learning derived from geometrical properties of gradient descent. It derives a novel algorithm whose meta-gradient can be approximated with low error without backpropagating through the task adaptation process. This allows the proposed algorithm to scale far beyond few-shot learning, opening up for meta-learning at scale.

Chapter 6 builds on the insights of Chapters 4 and 5. Based on Riemannian geometry, this work proposes a method for meta-learning a gradient based learning rule that is embedded in a task-adaptive model. This method is first-order equivalent to meta-learning a metric tensor across a distribution of tasks; in other words, it is equivalent to learning a warped gradient field (or more generally, vector field) for learning across a distribution of tasks. The method is the first gradient-based meta-learner whose meta-objective does not rely on evaluating a full learning trajectory, and thus avoids back-propagating through the adaptation process without the need for approximations. This work introduces a single algorithm which—to the best of our knowledge is the first of its kind—that can learn-to-learn across several learning problems, including few-shot, supervised, and reinforcement learning. Finally, this work introduces a novel form of meta-learning, meta-learning to continually learn, and demonstrates that the proposed method can meta-learn how to avoid catastrophic forgetting.



**Exogenous Tasks** Chapter 7 removes task distributions and instead focuses on an agent that must construct its own abstract tasks. Focusing on exploration as a means of inducing tasks, this work analyses challenges that arise when estimating uncertainty over value functions. Based on insights from this analysis, it instead proposes a method for estimating uncertainty over temporal difference errors. By conditioning on an observed transition, the estimator can leverage the temporal structure in the value estimate to reduce the bias in its estimate of uncertainty.

The proposed signal requires some algorithmic advances, as it cannot be used for posterior sampling (Strens, 2000), nor as an intrinsic reward (Singh et al., 2005). Instead, this work introduces a special form of a multi-agent game, where an exploration policy is tasked with finding data where an exploitation policy has maximal uncertainty. This form of exploration can be highly efficient in hard exploration games, in particular under environment stochasticity, and generally outperforms other state-of-the-art exploration methods on the environments considered. A striking implication is that this “exploration game” implicitly gives rise to a task distribution, or rather, an automatic curriculum. The implication of this is discussed momentarily in Section 8.3.

## 8.2 Contributions

### 8.2.1 Scalability

Issues of scalability are arguably the most pressing concern for the field today. These issues have arisen because most meta-learning with neural networks is designed for the few-shot learning scenario. The scalability problem is characterised by tasks that exhibit extreme data-scarcity. Because of such scarcity, learning is by necessity lightweight in that it must rely on non-parametric approaches, a few steps of adaptation, or in some other way rely heavily on a (typically meta-learned) prior. Because the task learning process is assumed to be lightweight, meta-learning can be computationally expensive. MAML, for instance, requires backpropagating through the entire learning process for each task in a mini-batch of tasks to compute a single meta-gradient.

Unfortunately, many real-world applications are not few-shot in nature, particularly so when they involve reinforcement learning. For instance, personalised technological services can be seen as a meta-learning problem, but in general it will take more than a few seconds of human interaction for the model to adapt to its user. At present, there appears to be no meta-learning system that can scale to such problems, although this thesis has taken a first initial step.



Indeed one goal of this thesis is to develop methods for meta-learning that are scalable and can be applied to large-scale problems. *Leap*, presented in Chapter 5, is a first step in this direction. Inspired by MAML, *Leap* is primarily designed to meta-learn an initialisation that is useful across a distribution of tasks. While MAML is defined within the few-shot paradigm, *Leap* is moulded from an altogether different philosophy. Instead of meta-learning the initialisation for optimal performance within a  $K$ -step adaptation budget, *Leap* is derived in terms of the characteristics of an arbitrary learning process. *Leap* learns about gradient fields over a distribution of manifolds, with the goal of finding an initialisation from which learning does not have to travel “far” on any one task from the given task distribution, irrespective of how many steps are actually taken on any one task.

Because it is derived based on first principles of geometrical quantities that arise naturally during task adaptation, the resulting algorithm can circumvent the need to backpropagate through the adaptation process. Thus, *Leap* can be scaled to learning problems whose complexity goes far beyond those that previous works have considered. While *Leap* provides us with means of scaling gradient-based meta-learning of an initialisation beyond few-shot learning, a key limitation of this class of algorithms is that they compress all meta-knowledge into a single point that interacts with task learning only once—at initialisation. This limits meta-learning both in that it imposes a potentially severe bottleneck—by compressing full trajectories into a single point—and because it renders the inductive bias provided by meta-learning passive: after the initialisation, the meta-learner has no means by which it can steer the learning process.

In response to these limitations, Chapter 6 develops a stronger form of meta-learning explicitly designed for meta-learning at scale. The WarpGrad framework presents a principled approach to blending black-box meta-learning with gradient-based meta-learning. By fusing the meta-learner with the task learner, WarpGrad is always interacting with—and therefore has far greater influence over—the learning process.

Because warp-layers are generic, they can be represented by universal function approximators and can therefore, in principle, represent any gradient-based learning rule. Moreover, WarpGrad explicitly avoids backpropagating through the learning process by leveraging its interaction on a meta-learned Riemannian manifold.

A consequence of this is that WarpGrad can readily be applied to problems for which an adaptation from a shared initialisation is not a well-defined concept, such as in continual learning. WarpGrad is directly applicable to any form of learning, including supervised, unsupervised, reinforcement, online, and continual learning. The work in this thesis demonstrates developments in several of these learning paradigms, paving the way for a comprehensive meta-learner that can be made relevant to real-world applications.

### 8.2.2 Generalisation

One outcome from WarpGrad is that more expressive forms of meta-learning yield better results (Section 6.F). The ablation studies showed that non-linear interaction between the learned update rule and the task learner’s parameters greatly improved performance, and indeed the meta-learned update rule could empirically outperform efficient algorithms such as Natural Gradient Descent when starting from the same initialisation.

Moreover, in the meta-reinforcement learning setting, it demonstrated that it is possible to achieve strong gains over the learning to reinforcement learn algorithm (Wang et al., 2016a; Duan et al., 2016). This algorithm parameterises the meta-learner as an RNN, and while Turing Complete, limits the interaction between the algorithm and the learner to the hidden state of the RNN. WarpGrad demonstrates that—by utilising the architecture proposed in Chapter 4—more complex interactions can vastly improve performance. What makes the proposed architecture powerful in the meta-learning setting is that it provides a means of adapting not only the learner but the meta-learner. In particular, since the parameters of the RNN defines the algorithm, dynamically adapting those parameters amounts to dynamically adapting the algorithm itself.

In general, the question of what architectures provide stronger meta-learners has received far less attention and is still a largely unanswered question (Chapter 3). Chapter 4 provides some answers in this direction by noting that neural networks are typically relatively inefficient: each layer behaves as a linear operator over large parts of their domain. By factorising neural networks into a composition of linear maps, this work suggests that the expressive capacity of a layer can be increased by adaptively parameterising the layer. An immediate consequence of the proposed mechanism is to provide a natural avenue for slow and fast weights in neural networks that embed a meta-learner.

### 8.2.3 Exogenous Tasks

Above contributions rely on an exogenously given task or task distribution. A potential issue for modern meta-learning is the reliance on a predefined task distribution. This is particularly challenging in the reinforcement learning domain, where the number of tasks in current task distributions are in the singular or double digits range (Finn et al., 2017; Packer et al., 2018; Cobbe et al., 2019; Yu et al., 2020), far too few to enable meta-generalisation.<sup>16</sup> One possible solution to this problem is to place the definition of a task within the meta-learner itself. In particular, in reinforcement learning, we can provide an agent with a single environment that the agent can use to induce its own distribution of tasks (Silver et al., 2018; Vinyals et al., 2019; Al-Shedivat et al., 2018; Sukhbaatar et al., 2018).

The final contribution of this thesis (Chapter 7) takes steps in this direction by developing a reinforcement learning system that can be seen as simultaneously proposing and solving abstract “tasks” induced by the set of value functions currently in the distribution. The proposed algorithm, TDU, defines two jointly learnable policies,  $\pi$  and  $\mu$ , where  $\pi$  is learned to maximise rewards on its own data and data collected by the exploration policy  $\mu$ .

The exploration policy,  $\mu$ , is tasked with collecting data where the greedy policy,  $\pi$ , has high uncertainty over the value of its policy, as measured by a learned distribution  $p(Q_\pi)$ . This distribution induces a task for  $\mu$  by defining a specific intrinsic reward function. For any given parameterisation of  $p(Q_\pi)$ , there will be regions in the state-space where uncertainty is maximised: the task for  $\mu$  is to collect data from these regions. This data is then used to evaluate uncertainty over  $\pi$ ; the goal of  $\pi$  (or rather, of  $p(Q_\pi)$ ), is to minimise this uncertainty. Thus as the estimate  $p(Q_\pi)$  fit to the data collected by  $\mu$ , its uncertainty will decrease and  $\mu$  must seek out other regions with higher uncertainty, thus inducing a new task for  $\mu$ . In this way, TDU proposes a never-ending learning system where  $\pi$  and  $\mu$  define task distributions for one another via a form of multi-player game.

Because  $\pi$  implicitly defines the task facing  $\mu$ , the notion of task in this system is well defined and meaningful, solving a task means collecting data where the agent is maximally uncertain over the value of its reward-maximising policy.<sup>17</sup> Notably, these policies are defined in terms of parameter space. Therefore, these

<sup>16</sup>A notable concurrent exception is (Metz et al., 2020), which presents a meta-training set of 1000s of tasks. They demonstrate strong benefits to meta-generalisation from increasing the number of meta-training tasks. Collecting thousands of reinforcement learning domains seems very unlikely.

<sup>17</sup>Conversely, for  $\pi$  to solve a task means maximally reducing uncertainty over the value of  $\pi$  on the data collected by itself and by  $\mu$ .

tasks implicitly define a distribution  $p(\theta)$  that can be used for meta-learning. The benefit of inducing tasks in the manner of TDU is that we may benefit from constructing tasks in a more structured—and potentially smaller—space than an abstract high-dimensional parameter space.

While each of the contributions in this thesis target basic components for an AI that achieves the properties identified in Chapter 1, they each have certain limitations that must be addressed before they can give rise to an AI powered by a scalable gradient-based meta-learning system.

### 8.3 Limitations and Future Work

This thesis relies on the notion of optimisation as machine learning, and thus inherits limitations that arise by formulating learning as an optimisation problem (Eq. 2.4). In particular, taking an optimisation-view on meta-learning, it is a conventional machine learning problem but over an unconventional quantity; a distribution of machine learning problems (as opposed to a distribution of data). A central limitation of this perspective is that of generalisation, indeed all frameworks considered in this thesis assume access to an i.i.d. sampled of data, an assumption that becomes burdensome in meta-learning.

A meta-learner samples tasks. Manually building i.i.d. task distributions can be just as challenging as engineering machine learning solutions. As an example of the monumental effort that goes into defining task (or task distributions), in order to train autonomous vehicles, car manufacturers have built a 32-acre uninhabited metropolis just for autonomous vehicles’ research.<sup>18</sup> Currently, scaling meta-learning to complex tasks is hampered by a lack of suitable task distributions (Hospedales et al., 2020). One possible solution is to simply create ever-larger task distributions, but not only does this seem infeasible, it also appears to run counter to the notion of a continually adapting AI.

While the TDU algorithm (Chapter 7) proposes one way to internalise an abstract task distribution, it does not leverage the structure of the world to generate meaningful tasks. In particular, tasks are only interesting insofar that they force the agent to try new behaviours, but there are no guarantees that the tasks that arise have this property. One possible way forward would be to explicitly meta-learn the task distribution under a model of the data-generating process. In supervised learning, this would mean learning a distribution over data-generative processed (Such et al., 2020); for reinforcement learning, it would mean learning a distribution over models (Ferreira et al., 2021).

<sup>18</sup>Home page: <https://mcity.umich.edu/our-work/mcity-test-facility/>.

There are two critical challenges with this approach. The first is the computational complexity of meta-learning a data-generating process; it is difficult to imagine how a task parameterised as a neural network would be a small, simple architecture. More likely, such neural tasks would require very large networks and enormous amounts of computation. The second challenge is to define a coherent meta-learning objective that promotes learning of a well-behaved task distribution. The approach taken in previous works is to learn a data-generative process with respect to some reference machine learning problem. This approach however falls into the trap of requiring a task distribution, this time to evaluate the meta-learned task distribution.

Perhaps a more promising way forward would be to, instead of defining tasks, define a consistent environment within which tasks naturally arise. The problem is then to train an AI that lives in this consistent environment indefinitely while facing a continuous stream of experience as it interacts with the world. At its most extreme, we can imagine an AI that lives in the real world, say through a digital interface with access to the internet. Researchers have already started to make inroads into this problem space ([Mahdavinejad et al., 2018](#); [Yao et al., 2018b](#)). One immediate challenge for AIs that learn from humans is that such interaction might not provide useful learning signals and can instead derail learning—a famous example of such failures being Microsoft’s chat bot ([Wolf et al., 2017](#)).

The benefit of introducing a large, consistent environment for learning is that it implicitly defines a class of tasks without requiring us to explicitly define the tasks themselves. In many cases, a single learning signal would be sufficient to engender a task distribution. For instance, suppose Amazon embedded an AI assistant on their website, whose goal it is to provide helpful assistance to customers. Each customer interaction would essentially give rise to a task, as the assistance needed would differ from case to case. Some might need help navigating a website, some might need help claiming a refund, or some might want a dictionary-style look-up. The learning signal in this case could be the reported level of satisfaction at the end of the interaction.

In this example, by simply defining a large environment—the Amazon internet domain—a single learning signal is sufficient to generate distinct, well-defined tasks that number in the millions. Such examples abound; for instance, Spotify, YouTube, Facebook, or government digital domains are just a few examples of potential applications. While research in this direction requires simpler environments, we can—and should—introduce meta-learning domains that do not explicitly provide the agent with tasks.

This also removes a related limitation that has arisen from the few-shot learning setup. Here, we have a notion of a meta-test time (Vinyals et al., 2016). The original intention is sensible: test an AI on tasks it has not yet seen before. The issue we face in this paradigm is that this introduces an unrealistic two-phase paradigm. First, we meta-train on a given set of tasks, then we deploy. This means that we need the full task distribution immediately at the start of meta-training, and once meta-training is complete no further meta-learning is allowed. This is both inefficient and rather unrealistic. Recall the definition of AI from Wang (2019):

“intelligence is the capacity of an information-processing system to adapt to its environment while operating with insufficient knowledge and resources.”

The key word is *adaptation*, to continuously change and improve as the environment evolves. If meta-learning is to deliver on its potential to produce AIs with human-level intelligence, we need a more general meta-learning paradigm that allows both task adaptation and meta-learning to happen in parallel through online interactions with the world.

Recent work has started to make progress in this direction (Denevi et al., 2019; Finn et al., 2019) by considering a meta-learning problem where tasks are sequentially sampled from a task distribution. Harrison et al. (2020) go one step further and consider time-series where tasks are not explicitly given to the meta-learner and propose a changepoint detection algorithm for constructing a task distribution.

These works, and virtually all prior research in meta-learning—including the contributions of this thesis—assume that tasks are discrete. However, in terms of meta-optimisation, there is no technical reason for why this must be the case. In a typical meta-objective, the task distribution serves to define the expectation for the outer objective. A highly promising avenue for future work comes from our analysis in Chapter 5, where we establish that tasks serve to induce a distribution over learning process trajectories; that is, a distribution over gradient vector fields (Chapter 6). It therefore follows that we can replace the notion of a “task” with the more abstract notion of a distribution over learning trajectories, or vector fields. These objects commute: a task induces a learning trajectory, and a learning trajectory identifies some task. Hence, we are free to choose the representation that benefits us the most.

One possible avenue to realise this abstraction is through the WarpGrad framework, whose objective can be defined in an entirely task-agnostic fashion. Since WarpGrad is compatible with any probability distribution  $p(\theta)$  defined

over  $\Theta_t$ , a straightforward approach is to define  $\Theta_t = (\theta_0, \dots, \theta_t)$  as the set of all observed parameters through a learner’s life and to define tasks as sub-sequences in  $\Theta_t$ . For instance, choose  $\theta_s$  at random from  $\Theta_t$  and define a “task” as a sub-sequence  $(\theta_s, \dots, \theta_{s+k}) \subset \Theta_t$  for some  $k \sim p(k)$ . This sequence can be seen as a learner starting learning in  $\theta_s$  on some task and progressing through to its final model  $\theta_{s+k}$ . Assume the meta-learner has a way of sampling relevant data for a given sequence and let  $\tau := (\theta_s, k)$ ; one possible WarpGrad objective in the absence of an exogenous task distribution is

$$\mathcal{J}_t(\phi) := \mathbb{E}_{(\theta_s, k) \sim p(\theta_s, k)} \left[ \sum_{i=0}^{k-1} \mathcal{L} \left( \theta_{s+k} - \nabla_{\theta} \mathcal{L}(\theta_{s+k}; \phi, \mathcal{D}_{s+k}); \phi, \mathcal{D}_{s+k+1} \right) \right].$$

where  $\mathcal{J}_t$  denotes the meta-objective at time  $t$  and  $\mathcal{L}$  denotes some given objective. This is just one possibility and most likely further research will reveal more efficient formulations. Regardless, this form of meta-learning is closer to how meta-learning was originally perceived, wherein meta-learning is a continuous process that co-evolves with the model itself. While some research in meta-reinforcement learning uses this type of meta-learning for automatic tuning of hyper-parameters (Xu et al., 2018b; Zahavy et al., 2020), few contemporary research papers have fully embraced a task-agnostic meta-learning framework (a notable exception being recent work by Xu et al., 2020).

Yet it seems more closely aligned with how we perceive human intelligence and with contemporary definitions of intelligence (Chapter 1). At a high level, we typically imagine artificial intelligence as an entity that continuously interacts with the world. It performs well when it knows what to do, and rapidly learns when it does not. When we define an AI in terms of a parameterised model, learning involves alterations to the AI’s parameters: adaptation that does not involve changes to parameters amount to inference where the AI maps current experience onto past experience to infer desirable behaviour.

Alterations to parameters can come in many ways. It seems likely—at present—that this will, to some extent, involve gradient-based learning. Gradient-based meta-learning seems therefore poised to play an important role in the advance of artificial intelligence. To deliver on this potential, current methods need to improve their scalability, be more widely applicable, and increase their expressive capacity. This thesis has taken some initial steps in this direction.

*Fin.*



# Bibliography

- Abbati, Gabriele, Tosi, Alessandra, Osborne, Michael, and Flaxman, Seth. AdaGeo: Adaptive Geometric Learning for Optimization and Sampling. In *International Conference on Artificial Intelligence and Statistics*, 2018. See p. 97.
- Achille, Alessandro and Soatto, Stefano. Emergence of Invariance and Disentanglement in Deep Representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018. See p. 27.
- Achille, Alessandro, Eccles, Tom, Matthey, Loic, Burgess, Christopher P., Watters, Nick, Lerchner, Alexander, and Higgins, Irina. Life-long disentangled representation learning with cross-domain latent homologies. In *Advances in Neural Information Processing Systems*, 2018. See pp. 52, 86, and 95.
- Agarwal, Alekh, Jiang, Nan, Kakade, Sham M., and Sun, Wen. Reinforcement Learning: Theory and Algorithms. Technical report, Microsoft Research, 2020. See pp. 43, 45, and 47.
- Ahlberg, J. Harold, Nilson, Edwin Norman, and Walsh, Joseph Leonard. *The Theory of Splines and Their Applications*. Academic Press, 1967. p. 51. See p. 90.
- Al-Shedivat, Maruan, Bansal, Trapit, Burda, Yuri, Sutskever, Ilya, Mordatch, Igor, and Abbeel, Pieter. Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments. In *International Conference on Learning Representations*, 2018. See pp. 75, 96, and 207.
- Alet, Ferran, Schneider, Martin F., Lozano-Perez, Tomas, and Kaelbling, Leslie Pack. Meta-Learning Curiosity Algorithms. In *International Conference on Learning Representations*, 2020. See pp. 23, 47, and 51.
- Allen-Zhu, Zeyuan, Li, Yuanzhi, and Song, Zhao. A Convergence Theory for Deep Learning via Over-Parameterization. *arXiv preprint arXiv:1811.03962*, 2018. See p. 40.
- Allis, Louis Victor. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Rijksuniversiteit Limburg, 1994. See p. 18.
- Allmendinger, Richard, Handl, Julia, and Knowles, Joshua. Multiobjective Optimization: When Objectives Exhibit Non-Uniform Latencies. *European Journal of Operational Research*, 243(2):497–513, 2015. See p. 52.
- Amari, Shun-Ichi. Natural Gradient Works Efficiently in Learning. *Neural computation*, 10(2):251–276, 1998. See pp. 90, 123, and 128.

- Amari, Shun-ichi and Nagaoka, Hiroshi. *Methods of Information Geometry*, volume 191. American Mathematical Society, 2007. See pp. 90, 97, 118, and 128.
- Andrychowicz, Marcin, Denil, Misha, Gómez, Sergio, Hoffman, Matthew W., Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to Learn by Gradient Descent by Gradient Descent. In *Advances in Neural Information Processing Systems*, 2016. See pp. 50, 51, 56, 61, 75, 87, 96, 117, and 127.
- Andrychowicz, Marcin, Wolski, Filip, Ray, Alex, Schneider, Jonas, Fong, Rachel, Welinder, Peter, McGrew, Bob, Tobin, Josh, Abbeel, Pieter, and Zaremba, Wojciech. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, 2017. See p. 28.
- Antoniou, Antreas, Edwards, Harrison, and Storkey, Amos J. How to Train Your MAML. In *International Conference on Learning Representations*, 2019. See pp. 62 and 120.
- Arora, Sanjeev, Cohen, Nadav, and Hazan, Elad. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *International Conference on Machine Learning*, 2018. See pp. 40 and 128.
- Arora, Sanjeev, Du, Simon S., Hu, Wei, Li, Zhiyuan, and Wang, Ruosong. Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. *arXiv preprint arXiv:1901.08584*, 2019. See p. 40.
- Arulkumaran, Kai, Cully, Antoine, and Togelius, Julian. Alphastar: An Evolutionary Computation Perspective. In *The Genetic and Evolutionary Computation Conference*, 2019. See p. 57.
- Arvanitidis, Georgios, Hansen, Lars Kai, and Hauberg, Søren. Latent Space Oddity: on the Curvature of Deep Generative Models. In *International Conference on Learning Representations*, 2018. See p. 97.
- Azizzadenesheli, Kamyar, Brunskill, Emma, and Anandkumar, Animashree. Efficient Exploration through Bayesian Deep Q-Networks. *arXiv preprint arXiv:1802.04412*, 2018. See pp. 174 and 180.
- Ba, Jimmy, Hinton, Geoffrey E., Mnih, Volodymyr, Leibo, Joel Z., and Ionescu, Catalin. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, 2016. See pp. 56, 61, and 127.
- Badue, Claudine, Guidolini, Rânik, Carneiro, Raphael Vivacqua, Azevedo, Pedro, Cardoso, Vinicius Brito, Forechi, Avelino, Jesus, Luan Ferreira Reis, Berriel, Rodrigo Ferreira, Paixão, Thiago Meireles, Mutz, Filipe Wall, Oliveira-Santos, Thiago, and de Souza, Alberto Ferreira. Self-Driving Cars: A Survey. *arXiv preprint arXiv:1901.04407*, 2019. See p. 19.
- Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, 2015. See p. 80.
- Balaji, Yogesh, Sankaranarayanan, Swami, and Chellappa, Rama. Metareg: Towards Domain Generalization Using Meta-Regularization. In *Advances in Neural Information Processing Systems*, 2018. See p. 53.
- Balduzzi, David and Ghifary, Muhammad. Strongly-Typed Recurrent Neural Networks. In *International Conference on Machine Learning*, 2016. See p. 24.

- Beck, Amir and Teboulle, Marc. Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. *Operations Research Letters*, 31:167–175, 2003. See pp. 127 and 128.
- Belkin, Mikhail, Hsu, Daniel, and Xu, Ji. Two Models of Double Descent for Weak Features. *arXiv preprint arXiv:1903.07571*, 2019. See p. 163.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. See pp. 87 and 100.
- Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, 2016. See pp. 161, 171, 174, and 190.
- Bengio, Samy, Bengio, Yoshua, Cloutier, Jocelyn, and Gecsei, Jan. On the Optimization of a Synaptic Learning Rule. In *Optimality in Biological and Artificial Networks*, pp. 6–8, 1995. See p. 75.
- Bengio, Yoshua. Gradient-Based Optimization of Hyperparameters. *Neural computation*, 12(8):1889–1900, 2000. See p. 52.
- Bengio, Yoshua, Bengio, Samy, and Cloutier, Jocelyn. *Learning a Synaptic Learning Rule*. Université de Montréal, Département d’informatique et de recherche opérationnelle, 1991. See pp. 21, 22, 56, 75, 96, and 127.
- Bengio, Yoshua, Deleu, Tristan, Rahaman, Nasim, Ke, Nan Rosemary, Lachapelle, Sébastien, Bilaniuk, Olexa, Goyal, Anirudh, and Pal, Christopher J. A Meta-Transfer Objective for Learning to Disentangle Causal Mechanisms. In *International Conference on Learning Representations*, 2019. See p. 51.
- Bennett, Kristin P. and Parrado-Hernández, Emilio. The Interplay of Optimization and Machine Learning Research. *Journal of Machine Learning Research*, 7(Jul):1265–1281, 2006. See p. 39.
- Berger, James O. and Wolpert, Robert L. The Likelihood Principle. In *Lecture Notes—Monograph Series*, volume 6. Institute of Mathematical Statistics, 1988. See pp. 34, 36, and 37.
- Bergstra, James, Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, 2011. See pp. 39 and 52.
- Berner, Christopher, Brockman, Greg, Chan, Brooke, Cheung, Vicki, Dębiak, Przemysław, Dennison, Christy, Farhi, David, Fischer, Quirin, Hashme, Shariq, Hesse, Chris, Józefowicz, Rafal, Gray, Scott, Olsson, Catherine, Pachocki, Jakub, Petrov, Michael, de Oliveira Pinto, Henrique Pondé, Raiman, Jonathan, Salimans, Tim, Schlatter, Jeremy, Schneider, Jonas, Sidor, Szymon, Sutskever, Ilya, Tang, Jie, Wolski, Filip, and Zhang, Susan. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arxiv:1912.06680*, 2019. See p. 18.
- Bertinetto, Luca, Henriques, João F., Valmadre, Jack, Torr, Philip, and Vedaldi, Andrea. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*, 2016. See pp. 60, 75, and 128.
- Bertsekas, Dimitri P. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4 edition, 1995. See p. 42.

- Bilen, Hakan and Vedaldi, Andrea. Universal Representations: the Missing Link Between Faces, Text, Planktons and Cat Breeds. *arXiv preprint arXiv:1701.07275*, 2017. See pp. 52 and 125.
- Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006. See pp. 32 and 35.
- Bottou, Léon. On-line Learning and Stochastic Approximations. In *On-line Learning in Neural Networks*, pp. 9–42. Cambridge University Press, 1998. See p. 40.
- Botvinick, Matthew, Wang, Jane X., Dabney, Will, Miller, Kevin J., and Kurth-Nelson, Zeb. Deep Reinforcement Learning and its Neuroscientific Implications. *Neuron*, 2020. See p. 41.
- Bradbury, James, Frostig, Roy, Hawkins, Peter, Johnson, Matthew James, Leary, Chris, Maclaurin, Dougal, and Wanderman-Milne, Skye. JAX: Composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>. See pp. 168 and 176.
- Braun, Daniel A., Aertsen, Ad, Wolpert, Daniel M., and Mehring, Carsten. Motor Task Variation Induces Structural Learning. *Current Biology*, 19(4):352–357, 2009. See p. 21.
- Britannica, Encyclopaedia. Go. <https://www.britannica.com/topic/go-game>, 2021. Accessed: 2021-01-12. See p. 18.
- Brock, Andrew, Donahue, Jeff, and Simonyan, Karen. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *International Conference on Machine Learning*, 2018a. See p. 19.
- Brock, Andrew, Lim, Theo, Ritchie, J.M., and Weston, Nick. SMASH: One-Shot Model Architecture Search through HyperNetworks. In *International Conference on Learning Representations*, 2018b. See pp. 75 and 76.
- Brown, Tom B., Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, Agarwal, Sandhini, Herbert-Voss, Ariel, Krueger, Gretchen, Henighan, Tom, Child, Rewon, Ramesh, Aditya, Ziegler, Daniel M., Wu, Jeffrey, Winter, Clemens, Hesse, Christopher, Chen, Mark, Sigler, Eric, Litwin, Mateusz, Gray, Scott, Chess, Benjamin, Clark, Jack, Berner, Christopher, McCandlish, Sam, Radford, Alec, Sutskever, Ilya, and Amodei, Dario. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*, 2020. See pp. 19, 21, 40, 52, and 63.
- Burda, Yuri, Edwards, Harrison, Pathak, Deepak, Storkey, Amos J., Darrell, Trevor, and Efros, Alexei A. Large-Scale Study of Curiosity-Driven Learning. *arXiv preprint arXiv:1808.04355*, 2018a. See p. 161.
- Burda, Yuri, Edwards, Harrison, Storkey, Amos J., and Klimov, Oleg. Exploration by Random Network Distillation. *arXiv preprint arXiv:1810.12894*, 2018b. See pp. 161, 171, 174, and 190.
- Byrd, R., Hansen, S., Nocedal, J., and Singer, Y. A Stochastic Quasi-Newton Method for Large-Scale Optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016. See p. 128.
- Bzdok, Danilo, Altman, Naomi, and Krzywinski, Martin. Points of Significance: Statistics versus Machine Learning. *Nature Methods*, 15:233–234, 2018. See p. 35.

- Canziani, Alfredo, Paszke, Adam, and Culurciello, Eugenio. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv preprint arXiv:1605.07678*, 2016. See pp. 27 and 68.
- Caruana, Rich. Multitask Learning. *Machine learning*, 28(1):41–75, 1997. See p. 52.
- Chen, Nutan, Klushyn, Alexej, Kurle, Richard, Jiang, Xueyan, Bayer, Justin, and van der Smagt, Patrick. Metrics for Deep Generative Models. In *International Conference on Artificial Intelligence and Statistics*, 2018. See p. 97.
- Chen, Yutian, Hoffman, Matthew W., Colmenarejo, Sergio Gomez, Denil, Misha, Lillicrap, Timothy P., and de Freitas, Nando. Learning to learn for Global Optimization of Black Box Functions. In *Advances in Neural Information Processing Systems*, 2016. See p. 56.
- Chen, Yutian, Hoffman, Matthew W., Colmenarejo, Sergio Gómez, Denil, Misha, Lillicrap, Timothy P., Botvinick, Matt, and de Freitas, Nando. Learning to Learn Without Gradient Descent by Gradient Descent. In *International Conference on Machine Learning*, 2017. See pp. 28, 39, and 117.
- Chesney, Bobby and Citron, Danielle. Deep Fakes: a Looming Challenge for Privacy, Democracy, and National Security. *California Law Review*, 107:1753, 2019. See p. 19.
- Cho, Kyunghyun, van Merriënboer, Bart, Gülçehre, Çağlar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2014. See p. 76.
- Choi, Jongwook, Guo, Yijie, Moczulski, Marcin, Oh, Junhyuk, Wu, Neal, Norouzi, Mohammad, and Lee, Honglak. Contingency-Aware Exploration in Reinforcement Learning. *arXiv preprint arXiv:1811.01483*, 2018. See p. 174.
- Chollet, François. On the Measure of Intelligence. *arxiv Preprint arXiv:1911.01547*, 2019. See pp. 19, 20, 21, and 22.
- Chung, Junyoung, Gülçehre, Çağlar, Cho, Kyunghyun, and Bengio, Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint, arXiv:1412.3555*, 2014. See p. 76.
- Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and Accurate Deep Network Learning by Exponential Linear Units (Elus). In *International Conference on Learning Representations*, 2015. See p. 68.
- Clune, Jeff. AI-GAs: AI-Generating Algorithms, An Alternate Paradigm for Producing General Artificial Intelligence. *arXiv preprint arXiv:1905.10985*, 2019. See p. 57.
- Cobbe, Karl, Klimov, Oleg, Hesse, Chris, Kim, Taehoon, and Schulman, John. Quantifying Generalization in Reinforcement Learning. In *International Conference on Machine Learning*, 2019. See p. 207.
- Cooijmans, Tim, Ballas, Nicolas, Laurent, César, and Courville, Aaron. Recurrent Batch Normalization. In *International Conference on Learning Representations*, 2016. See p. 76.
- Cortes, Corinna and Vapnik, Vladimir. Support-Vector Networks. *Machine learning*, 20(3):273–297, 1995. See p. 39.

- Cybenko, George. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. See pp. 61, 68, and 118.
- Czarnecki, Wojciech Marian, Osindero, Simon, Pascanu, Razvan, and Jaderberg, Max. A Deep Neural Network’s Loss Surface Contains Every Low-dimensional Pattern. *arXiv preprint arXiv:1912.07559*, 2019. See p. 40.
- Dauphin, Yann N., Fan, Angela, Auli, Michael, and Grangier, David. Language Modeling with Gated Convolutional Networks. In *International Conference on Machine Learning*, 2017. See pp. 68 and 77.
- Dearden, Richard, Friedman, Nir, and Russell, Stuart. Bayesian Q-learning. In *Association for the Advancement of Artificial Intelligence*, 1998. See p. 174.
- Deisenroth, Marc and Rasmussen, Carl E. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning*, 2011. See p. 173.
- Denevi, Giulia, Stamos, Dimitris, Ciliberto, Carlo, and Pontil, Massimiliano. Online-Within-Online Meta-Learning. In *Advances in Neural Information Processing Systems*, 2019. See p. 210.
- Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition*, 2009. See pp. 52, 60, and 146.
- Denil, Misha, Shakibi, Babak, Dinh, Laurent, Ranzato, Marc’Aurelio, and De Freitas, Nando. Predicting Parameters in Deep Learning. In *Advances in Neural Information Processing Systems*, 2013. See pp. 73 and 75.
- Desjardins, Guillaume, Simonyan, Karen, Pascanu, Razvan, and Kavukcuoglu, Koray. Natural Neural Networks. In *Advances in Neural Information Processing Systems*, 2015. See pp. 118, 127, 128, 131, 144, and 145.
- Doersch, Carl, Gupta, Ankush, and Zisserman, Andrew. CrossTransformers: Spatially-Aware Few-Shot Transfer. *arXiv preprint arXiv:2007.11498*, 2020. See p. 25.
- Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *International Conference on Learning Representations*, 2014. See p. 52.
- Drake, John H., Kheiri, Ahmed, Özcan, Ender, and Burke, Edmund K. Recent Advances in Selection Hyper-Heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020. See pp. 39 and 52.
- Du, Simon S., Zhai, Xiyu, Póczos, Barnabás, and Singh, Aarti. Gradient Descent Provably Optimizes Over-parameterized Neural Networks. *arXiv preprint arXiv:1810.02054*, 2018. See p. 40.
- Duan, Yan, Schulman, John, Chen, Xi, Bartlett, Peter L., Sutskever, Ilya, and Abbeel, Pieter. RL2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016. See pp. 55, 61, 127, and 206.
- Edwards, Harrison and Storkey, Amos. Towards a Neural Statistician. In *International Conference on Learning Representations*, 2017. See p. 53.



- Falkner, Stefan, Klein, Aaron, and Hutter, Frank. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*, 2018. See pp. 39 and 52.
- Fawcett, Tom. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8): 861–874, 2006. See p. 34.
- Fei-Fei, Li, Fergus, Rob, and Perona, Pietro. A Bayesian Approach to Unsupervised One-Shot Learning of Object Categories. In *International Conference on Computer Vision*, 2003. See pp. 53 and 59.
- Fernando, Chrisantha, Banarse, Dylan, Reynolds, Malcolm, Besse, Frederic, Pfau, David, Jaderberg, Max, Lanctot, Marc, and Wierstra, Daan. Convolution by Evolution - Differentiable Pattern Producing Networks. *The Genetic and Evolutionary Computation Conference*, 2016. See p. 75.
- Ferreira, Fabio, Nierhoff, Thomas, and Hutter, Frank. Learning Synthetic Environments for Reinforcement Learning with Evolution Strategies. *arXiv preprint arXiv:2101.09721*, 2021. See p. 208.
- Feurer, Matthias, Klein, Aaron, Eggenberger, Katharina, Springenberg, Jost Tobias, Blum, Manuel, and Hutter, Frank. Auto-Sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning*, pp. 113–134. Springer, Cham, 2019. See p. 52.
- Finn, Chelsea and Levine, Sergey. Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm. In *iclr*, 2018. See pp. 24 and 61.
- Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, 2017. See pp. 25, 39, 41, 50, 51, 60, 61, 75, 87, 91, 94, 96, 97, 106, 109, 118, 119, 125, 132, 150, and 207.
- Finn, Chelsea, Xu, Kelvin, and Levine, Sergey. Probabilistic Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems*, 2018. See p. 62.
- Finn, Chelsea, Rajeswaran, Aravind, Kakade, Sham, and Levine, Sergey. Online Meta-Learning. In *International Conference on Machine Learning*, 2019. See pp. 53, 62, and 210.
- Flennerhag, Sebastian, Yin, Hujun, Keane, John, and Elliot, Mark. Breaking the Activation Function Bottleneck Through Adaptive Parameterization. In *Advances in Neural Information Processing Systems*, 2018. See pp. 27, 29, 68, 132, 135, 148, and 149.
- Flennerhag, Sebastian, Moreno, Pablo G., Lawrence, Neil D., and Damianou, Andreas. Transferring Knowledge across Learning Processes. In *International Conference on Learning Representations*, 2019. See pp. 25, 29, 51, 61, 86, 118, 119, 125, 128, 130, 137, 141, 142, and 143.
- Flennerhag, Sebastian, Rusu, Andrei A., Pascanu, Razvan, Visin, Francesco, Yin, Hujun, and Hadsell, Raia. Meta-Learning with Warped Gradient Descent. In *International Conference on Learning Representations*, 2020a. See pp. 23, 25, 26, 27, 29, 54, 58, and 117.
- Flennerhag, Sebastian, Wang, Jane, Visin, Francesco, Galashov, Alexandre, Sprechmann, Pablo, Heess, Nicolas, Borsa, Diana, Baretto, André, and Razvan Pas-

- canu. Temporal Difference Uncertainties as a Signal for Exploration. *arXiv preprint arXiv:2010.02255*, 2020b. See pp. 28 and 30.
- Florensa, Carlos, Duan, Yan, and Abbeel, Pieter. Stochastic Neural Networks for Hierarchical Reinforcement Learning. In *International Conference on Learning Representations*, 2017. See p. 174.
- Florensa, Carlos, Held, David, Geng, Xinyang, and Abbeel, Pieter. Automatic Goal Generation for Reinforcement Learning Agents. In *International Conference on Machine Learning*, 2018. See pp. 29 and 158.
- Fortunato, Meire, Azar, Mohammad Gheshlaghi, Piot, Bilal, Menick, Jacob, Osband, Ian, Graves, Alex, Mnih, Vlad, Munos, Rémi, Hassabis, Demis, Pietquin, Olivier, Blundell, Charles, and Legg, Shane. Noisy Networks for Exploration. In *International Conference on Learning Representations*, 2018. See pp. 164, 171, 173, 182, and 191.
- Frankle, Jonathan and Carbin, Michael. The Lottery Ticket Hypothesis: Training Pruned Neural Networks. *International Conference on Learning Representations*, 2019. See p. 68.
- Freeman, C. Daniel and Bruna, Joan. Topology and Geometry of Half-Rectified Network Optimization. In *International Conference on Learning Representations*, 2017. See pp. 39 and 61.
- French, Robert M. Catastrophic Forgetting in Connectionist Networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. See pp. 30, 53, 132, and 150.
- Gal, Yarin and Ghahramani, Zoubin. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, 2016. See p. 83.
- Garipov, Timur, Izmailov, Pavel, Podoprikin, Dmitrii, Vetrov, Dmitry P., and Wilson, Andrew G. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. In *Advances in Neural Information Processing Systems*, 2018. See pp. 39 and 61.
- Garnelo, Marta, Rosenbaum, Dan, Maddison, Christopher, Ramalho, Tiago, Saxton, David, Shanahan, Murray, Teh, Yee Whye, Rezende, Danilo, and Eslami, S.M. Ali. Conditional Neural Processes. In *International Conference on Machine Learning*, 2018. See p. 53.
- Gehring, Clement and Precup, Doina. Smart Exploration in Reinforcement Learning using Absolute Temporal Difference Errors. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, 2013. See p. 174.
- Gers, Felix A., Schmidhuber, Jürgen, and Cummins, Fred. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. See pp. 69, 73, and 76.
- Ghahramani, Zoubin. Unsupervised learning. In *Summer School on Machine Learning*, pp. 72–112. Springer, 2003. See p. 34.
- Gidaris, Spyros and Komodakis, Nikos. Dynamic Few-Shot Visual Learning Without Forgetting. In *Computer Vision and Pattern Recognition*, 2018. See pp. 60 and 128.
- Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Computer Vision and Pattern Recognition*, 2014. See p. 86.



- Gomez, Faustino and Schmidhuber, Jürgen. Evolving Modular Fast-Weight Networks for Control. In *International Conference on Artificial Neural Networks*, 2005. See pp. 56 and 75.
- Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. See pp. 32, 39, and 40.
- Goodfellow, Ian J., Mirza, Mehdi, Xiao, Da, Courville, Aaron, and Bengio, Yoshua. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013. See p. 95.
- Gordon, Goren and Ahissar, Ehud. Reinforcement Active Learning Hierarchical Loops. In *International Joint Conference on Neural Networks*, 2011. See p. 175.
- Grant, Erin, Finn, Chelsea, Levine, Sergey, Darrell, Trevor, and Griffiths, Thomas L. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations*, 2018. See pp. 53, 62, and 124.
- Grave, Edouard, Joulin, Armand, and Usunier, Nicolas. Improving Neural Language Models with a Continuous Cache. In *International Conference on Learning Representations*, 2017. See pp. 78 and 81.
- Graves, Alex. Generating Sequences With Recurrent Neural Networks. *arXiv preprint, arXiv:1308.0850*, 2013. See pp. 75 and 78.
- Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014. See p. 61.
- Gregor, Karol, Rezende, Danilo Jimenez, and Wierstra, Daan. Variational Intrinsic Control. *arXiv preprint arXiv:1611.07507*, 2016. See p. 174.
- Griffiths, Thomas L., Callaway, Frederick, Chang, Michael B., Grant, Erin, Krueger, Paul M., and Lieder, Falk. Doing More With Less: Meta-Reasoning and Meta-Learning in Humans and Machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019. See p. 59.
- Guez, Arthur, Viola, Fabio, Weber, Théophane, Buesing, Lars, Kapturowski, Steven, Precup, Doina, Silver, David, and Heess, Nicolas. Value-Driven Hindsight Modelling. In *Advances in Neural Information Processing Systems*, 2020. See p. 196.
- Ha, David and Eck, Douglas. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations*, 2018. See pp. 75 and 76.
- Ha, David, Dai, Andrew, and Le, Quoc V. HyperNetworks. In *International Conference on Learning Representations*, 2017. See pp. 56, 60, 61, 74, 75, 76, 81, 128, 131, and 135.
- Harrison, James, Sharma, Apoorva, Finn, Chelsea, and Pavone, Marco. Continuous Meta-Learning without Tasks. In *Advances in Neural Information Processing Systems*, 2020. See p. 210.
- Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009. See p. 39.
- Hausman, Karol, Springenberg, Jost Tobias, Wang, Ziyu, Heess, Nicolas, and Riedmiller, Martin. Learning an Embedding Space for Transferable Robot Skills. In *International Conference on Learning Representations*, 2018. See p. 174.

- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving Deep into Rectifiers - Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Computer Vision*, 2015. See p. 68.
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep Residual Learning for Image Recognition. In *Computer Vision and Pattern Recognition*, 2016. See pp. 135 and 143.
- He, Kaiming, Gkioxari, Georgia, Dollár, Piotr, and Girshick, Ross. Mask R-CNN. In *International Conference on Computer Vision*, 2017. See pp. 86 and 122.
- Hernández-Orallo, José. Evaluation in Artificial Intelligence: From Task-Oriented to Ability-Oriented Measurement. *Artificial Intelligence Review*, 48(3):397–447, 2017. See pp. 19 and 20.
- Higgins, Irina, Pal, Arka, Rusu, Andrei A., Matthey, Loic, Burgess, Christopher P., Pritzel, Alexander, Botvinick, Matthew, Blundell, Charles, and Lerchner, Alexander. Darla: Improving Zero-Shot Transfer in Reinforcement Learning. In *International Conference on Machine Learning*, 2017. See pp. 52, 86, and 95.
- Hinton, Geoffrey E. and Plaut, David C. Using Fast Weights to Deblur Old Memories. In *Cognitive Science Society*, 1987. See pp. 21, 22, 55, and 127.
- Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. *Neural Computation*, 9:1735–1780, 1997. See pp. 66, 69, 73, 76, and 148.
- Hochreiter, Sepp, Younger, A. Steven, and Conwell, Peter R. Learning To Learn Using Gradient Descent. In *International Conference on Artificial Neural Networks*, 2001. See pp. 23, 24, 54, 61, 96, and 127.
- Hornik, Kurt. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257, 1991. See pp. 61, 68, and 118.
- Hospedales, Timothy, Antoniou, Antreas, Micaelli, Paul, and Storkey, Amos. Meta-Learning in Neural Networks: A Survey. *arXiv preprint arXiv:2004.05439*, 2020. See pp. 25, 26, 52, 56, 59, 63, and 208.
- Hou, Ruibing, Chang, Hong, Bingpeng, M.A., Shan, Shiguang, and Chen, Xilin. Cross Attention Network for Few-Shot Classification. In *Advances in Neural Information Processing Systems*, 2019. See p. 61.
- Howard, Jeremy and Ruder, Sebastian. Universal Language Model Fine-tuning for Text Classification. In *Computational Linguistics*, 2018. See p. 52.
- Huang, Kexin and Zitnik, Marinka. Graph Meta Learning via Local Subgraphs. In *Advances in Neural Information Processing Systems*, 2020. See p. 62.
- Inan, Hakan, Khosravi, Khashayar, and Socher, Richard. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. In *International Conference on Learning Representations*, 2017. See pp. 81 and 83.
- Ioffe, Sergey and Szegedy, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015. See pp. 135 and 143.
- Izmailov, Pavel, Podoprikin, Dmitrii, Garipov, Timur, Vetrov, Dmitry, and Wilson, Andrew Gordon. Averaging Weights Leads to Wider Optima and Better Generalization. In *Conference on Uncertainty in Artificial Intelligence*, 2018. See p. 40.

- Jacot, Arthur, Gabriel, Franck, and Hongler, Clément. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In *Advances in Neural Information Processing Systems*, 2018. See p. 40.
- Jaderberg, Max, Czarnecki, Wojciech Marian, Osindero, Simon, Vinyals, Oriol, Graves, Alex, and Kavukcuoglu, Koray. Decoupled Neural Interfaces using Synthetic Gradients. In *International Conference on Machine Learning*, 2017a. See p. 75.
- Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M., Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, Fernando, Chrisantha, and Kavukcuoglu, Koray. Population Based Training of Neural Networks. *arXiv preprint arXiv:1711.09846*, 2017b. See pp. 39 and 52.
- Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z., Silver, David, and Kavukcuoglu, Koray. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations*, 2017c. See p. 28.
- Jankowski, Norbert, Duch, Włodzisław, and Grąbczewski, Krzysztof. *Meta-Learning in Computational Intelligence*, volume 358. Springer, 2011. See pp. 22 and 57.
- Janz, David, Hron, Jiri, Hernández-Lobato, José Miguel, Hofmann, Katja, and Tschitschek, Sebastian. Successor Uncertainties: Exploration and Uncertainty in Temporal Difference Learning. In *Advances in Neural Information Processing Systems*, 2019. See pp. 161, 162, 164, 171, 174, 180, 187, and 191.
- Javed, Khurram and White, Martha. Meta-Learning Representations for Continual Learning. *Advances in Neural Information Processing Systems*, 2019. See pp. 23, 25, 30, 54, 62, and 132.
- Jo, Jason and Bengio, Yoshua. Measuring the Tendency of CNNs to Learn Surface Statistical Regularities. *arXiv preprint arXiv:1711.11561*, 2017. See p. 61.
- Johnson-Laird, Philip N. Mental Models in Cognitive Science. *Cognitive science*, 4(1): 71–115, 1980. See p. 18.
- Justesen, Niels, Rodriguez Torrado, Ruben, Bontrager, Philip, Khalifa, Ahmed, Togelius, Julian, and Risi, Sebastian. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2018. See p. 61.
- Kachalsky, Ilya, Zabashta, Alexey, Filchenkov, Andrey, and Korneev, Georgiy. Generating Datasets for Classification Task and Predicting Best Classifiers with Conditional Generative Adversarial Networks. In *International Conference on Advances in Artificial Intelligence*, 2019. See pp. 29 and 158.
- Kapturowski, Steven, Ostrovski, Georg, Quan, John, Munos, Remi, and Dabney, Will. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*, 2018. See p. 172.
- Kawaguchi, Kenji. Deep Learning Without Poor Local Minima. In *Advances in Neural Information Processing Systems*, 2016. See p. 40.
- Kearns, Michael and Singh, Satinder. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine learning*, 49(2-3):209–232, 2002. See p. 161.
- Kim, Taesup, Yoon, Jaesik, Dia, Ousmane, Kim, Sungwoong, Bengio, Yoshua, and Ahn,

- Sungjin. Bayesian Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems*, 2018a. See pp. 62 and 127.
- Kim, Young-Bum. The Scalable Neural Architecture behind Alexa's Ability to Select Skills, 2018. URL <https://www.amazon.science/blog/the-scalable-neural-architecture-behind-alexa-s-ability-to-select-skills>. See p. 19.
- Kim, Young-Bum, Kim, Dongchan, Kumar, Anjishnu, and Sarikaya, Ruhi. Efficient Large-Scale Neural Domain Classification with Personalized Attention. In *Computational Linguistics*, 2018b. See p. 19.
- Kingma, Diederik P. and Ba, Jimmy. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015. See pp. 76, 111, 128, 147, 187, 189, and 195.
- Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil, Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A., Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, Hassabis, Demis, Clopath, Claudia, Kumaran, Dharshan, and Hadsell, Raia. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, 2017. See pp. 53 and 95.
- Kirsch, Louis, van Steenkiste, Sjoerd, and Schmidhuber, Jürgen. Improving Generalization in Meta Reinforcement Learning using Learned Objectives. In *International Conference on Learning Representations*, 2019. See p. 50.
- Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-Normalizing Neural Networks. In *Advances in Neural Information Processing Systems*, 2017. See p. 68.
- Klein, Aaron, Falkner, Stefan, Bartels, Simon, Hennig, Philipp, and Hutter, Frank. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Artificial Intelligence and Statistics*, 2017. See p. 52.
- Koch, Gregory. *Siamese Neural Networks for One-Shot Image Recognition*. PhD thesis, University of Toronto, 2015. See pp. 57 and 60.
- Konda, Vijay R. and Tsitsiklis, John N. Actor-Critic Algorithms. In *Advances in Neural Information Processing Systems*, 2000. See p. 47.
- Kotthoff, Lars, Thornton, Chris, Hoos, Holger H., Hutter, Frank, and Leyton-Brown, Kevin. Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. *The Journal of Machine Learning Research*, 18(1):826–830, 2017. See p. 54.
- Krause, Ben, Lu, Liang, Murray, Iain, and Renals, Steve. Multiplicative LSTM for Sequence Modelling. *arXiv preprint, arXiv:1609:07959*, 2016. See p. 76.
- Krause, Ben, Kahembwe, Emmanuel, Murray, Iain, and Renals, Steve. Dynamic Evaluation of Neural Sequence Models. *arXiv preprint, arXiv:1709:07432*, 2017. See pp. 75, 76, and 78.
- Kumar, Abhishek, Sattigeri, Prasanna, and Fletcher, P. Thomas. Improved Semi-supervised Learning with GANs using Manifold Invariances. In *Advances in Neural Information Processing Systems*, 2017. See p. 97.
- Kumaraswamy, Raksha, Schlegel, Matthew, White, Adam, and White, Martha. Context-

- Dependent Upper-Confidence Bounds for Directed Exploration. In *Advances in Neural Information Processing Systems*, pp. 4779–4789, 2018. See p. 175.
- Lacoste, Alexandre, Oreshkin, Boris, Chung, Wonchang, Boquet, Thomas, Rostamzadeh, Negar, and Krueger, David. Uncertainty in Multitask Transfer Learning. In *Advances in Neural Information Processing Systems*, 2018. See p. 126.
- Lake, Brenden, Salakhutdinov, Ruslan, Gross, Jason, and Tenenbaum, Joshua. One Shot Learning of Simple Visual Concepts. In *Annual Meeting of the Cognitive Science Society*, 2011. See pp. 22, 23, 57, 59, 60, 87, and 130.
- Lake, Brenden M., Salakhutdinov, Ruslan, and Tenenbaum, Joshua B. Human-Level Concept Learning through Probabilistic Program Induction. *Science*, 350(6266): 1332–1338, 2015. See pp. 21, 23, 96, 97, and 128.
- Lake, Brenden M., Ullman, Tomer D., Tenenbaum, Joshua B., and Gershman, Samuel J. Building Machines that Learn and Think Like People. *Behavioral and brain sciences*, 40, 2017. See pp. 20 and 61.
- Larochelle, Hugo, Erhan, Dumitru, and Bengio, Yoshua. Zero-Data Learning of New Tasks. In *AAAI*, 2008. See p. 59.
- Lawrence, Neil D. and Platt, John C. Learning to Learn with the Informative Vector Machine. In *International Conference on Machine Learning*, 2004. See pp. 53 and 59.
- LeCun, Yann, Bengio, Yoshua, et al. Convolutional Networks for Images, Speech, and Time Series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. See p. 80.
- LeCun, Yann, Bottou, Léon, Orr, Genevieve B., and Müller, Klaus-Robert. Efficient Backprop. In *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer, 1998. See p. 76.
- Lee, John M. *Introduction to Smooth Manifolds*. Springer, 2003. See p. 123.
- Lee, Kwonjoon, Maji, Subhransu, Ravichandran, Avinash, and Soatto, Stefano. Meta-Learning with Differentiable Convex Optimization. In *Computer Vision and Pattern Recognition*, 2019a. See pp. 62 and 128.
- Lee, Kwonjoon, Maji, Subhransu, Ravichandran, Avinash, and Soatto, Stefano. Meta-Learning with Differentiable Convex Optimization. In *Computer Vision and Pattern Recognition*, 2019b. See p. 60.
- Lee, Sang-Woo, Kim, Jin-Hwa, Ha, JungWoo, and Zhang, Byoung-Tak. Overcoming Catastrophic Forgetting by Incremental Moment Matching. In *Advances in Neural Information Processing Systems*, 2017. See p. 95.
- Lee, Yoonho and Choi, Seungjin. Meta-Learning with Adaptive Layerwise Metric and Subspace. In *International Conference on Machine Learning*, 2018. See pp. 26, 50, 62, 75, 96, 118, 120, 128, and 134.
- Legg, Shane and Hutter, Marcus. Universal Intelligence: A Definition of Machine Intelligence. *arXiv preprint arXiv:0712.3329*, 2007. See p. 20.
- Leite, Rui, Brazdil, Pavel, and Vanschoren, Joaquin. Selecting classification Algorithms with Active Testing. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2012. See p. 50.

- Li, Da, Yang, Yongxin, Song, Yi-Zhe, and Hospedales, Timothy M. Learning to Generalize: Meta-learning for Domain Generalization. In *Association for the Advancement of Artificial Intelligence*, 2018. See p. 53.
- Li, Ke and Malik, Jitendra. Learning to Optimize. In *International Conference on Machine Learning*, 2016. See pp. 24, 61, 117, and 120.
- Li, Yiyang, Yang, Yongxin, Zhou, Wei, and Hospedales, Timothy. Feature-Critic Networks for Heterogeneous Domain Generalization. In *International Conference on Machine Learning*, 2019. See p. 53.
- Li, Yujia, Gimeno, Felix, Kohli, Pushmeet, and Vinyals, Oriol. Strong Generalization and Efficiency in Neural Programs. *arXiv preprint arXiv:2007.03629*, 2020. See p. 163.
- Li, Zhenguo, Zhou, Fengwei, Chen, Fei, and Li, Hang. Meta-SGD: Learning to Learn Quickly for Few-Shot Learning. *arXiv preprint arXiv:1707.09835*, 2017. See pp. 41, 51, 62, 118, and 120.
- Li, Zhizhong and Hoiem, Derek. Learning Without Forgetting. In *European Conference on Computer Vision*, 2016. See pp. 39, 52, 61, and 125.
- Liang, Tengyuan, Poggio, Tomaso, Rakhlin, Alexander, and Stokes, James. Fisher-Rao Metric, Geometry, and Complexity of Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, 2019. See p. 27.
- Liu, Chaoyue, Zhu, Libin, and Belkin, Mikhail. Toward a Theory of Optimization for Over-Parameterized Systems of Non-Linear Equations: the Lessons of Deep Learning. *arXiv preprint arXiv:2003.00307*, 2020. See p. 163.
- Liu, Hao, Socher, Richard, and Xiong, Caiming. Taming MAML: Efficient Unbiased Meta-Reinforcement Learning. In *International Conference on Machine Learning*, 2019. See pp. 62 and 120.
- Lopes, Manuel, Lang, Tobias, Toussaint, Marc, and Oudeyer, Pierre-Yves. Exploration in Model-Based Reinforcement Learning by Empirically Estimating Learning Progress. In *Advances in Neural Information Processing Systems*, 2012. See p. 174.
- Loshchilov, Ilya and Hutter, Frank. SGDR: Stochastic Gradient Descent with Restarts. In *International Conference on Learning Representations*, 2017. See p. 111.
- Luk, Kevin and Grosse, Roger. A Coordinate-Free Construction of Scalable Natural Gradient. *arXiv preprint arXiv:1808.10340*, 2018. See pp. 90 and 97.
- Machado, Marlos C., Bellemare, Marc G., Talvitie, Erik, Veness, Joel, Hausknecht, Matthew, and Bowling, Michael. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. See pp. 172, 173, and 198.
- MacKay, David J.C. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 4<sup>th</sup> edition, 2003. See pp. 32 and 35.
- Maclaurin, Dougal, Duvenaud, David, and Adams, Ryan. Gradient-Based Hyperparameter Optimization Through Reversible Learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015. See p. 52.
- Mahajan, Dhruv, Girshick, Ross B., Ramanathan, Vignesh, He, Kaiming, Paluri, Manohar, Li, Yixuan, Bharambe, Ashwin, and van der Maaten, Laurens. Ex-



- ploring the Limits of Weakly Supervised Pretraining. In *European Conference on Computer Vision*, 2018. See p. 86.
- Mahdaveinejad, Mohammad Saeid, Rezvan, Mohammadreza, Barekatain, Mohammadamin, Adibi, Peyman, Barnaghi, Payam, and Sheth, Amit P. Machine Learning for Internet of Things Data Analysis: A Survey. *Digital Communications and Networks*, 4(3):161–175, 2018. See p. 209.
- Mangasarian, Olvi L. and Solodov, Mikhail V. Backpropagation Convergence via Deterministic Nonmonotone Perturbed Minimization. In *Advances in Neural Information Processing Systems*, 1994. See p. 39.
- Marcus, Mitchell P., Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. See p. 77.
- Martens, James. Deep Learning via Hessian-Free Optimization. In *International Conference on Machine Learning*, 2010. See pp. 39, 90, and 128.
- Martens, James and Grosse, Roger. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning*, 2015. See pp. 128, 143, and 144.
- McCloskey, Michael and Cohen, Neal J. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Elsevier, 1989. See p. 95.
- Melis, Gábor, Dyer, Chris, and Blunsom, Phil. On the State of the Art of Evaluation in Neural Language Models. In *International Conference on Learning Representations*, 2018. See pp. 78, 79, 80, and 81.
- Mendonca, Russell, Gupta, Abhishek, Kravev, Rosen, Abbeel, Pieter, Levine, Sergey, and Finn, Chelsea. Guided Meta-Policy Search. In *Advances in Neural Information Processing Systems*, 2019. See p. 125.
- Merity, Stephen, Xiong, Caiming, Bradbury, James, and Socher, Richard. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017. See p. 79.
- Merity, Stephen, Keskar, Nitish Shirish, and Socher, Richard. Regularizing and Optimizing LSTM Language Models. In *International Conference on Learning Representations*, 2018. See pp. 78, 79, 80, 81, and 83.
- Metz, Luke, Maheswaranathan, Niru, Cheung, Brian, and Sohl-Dickstein, Jascha. Meta-Learning Update Rules for Unsupervised Representation Learning. In *International Conference on Learning Representations*, 2019. See pp. 23 and 125.
- Metz, Luke, Maheswaranathan, Niru, Sun, Ruoxi, Freeman, C. Daniel, Poole, Ben, and Sohl-Dickstein, Jascha. Using a Thousand Optimization Tasks to Learn Hyperparameter Search Strategies. *arXiv preprint arXiv:2002.11887*, 2020. See p. 207.
- Miconi, Thomas, Clune, Jeff, and Stanley, Kenneth O. Differentiable Plasticity: Training Plastic Neural Networks with Backpropagation. *International Conference on Machine Learning*, 2018. See pp. 95, 131, 132, and 148.
- Miconi, Thomas, Clune, Jeff, and Stanley, Kenneth O. Backpropamine: Training Self-Modifying Neural Networks with Differentiable Neuromodulated Plasticity. In

- International Conference on Learning Representations*, 2019. See pp. 119, 131, 132, 148, and 155.
- Mikolov, Tomas, Karafiat, Martin, Burget, Lukas, Cernocky, Jan, and Khudanpur, Sanjeev. Recurrent Neural Network Based Language Model. *Interspeech*, 2:3, 2010. See p. 77.
- Mikolov, Tomáš. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012. See pp. 75 and 78.
- Mikolov, Tomáš, Sutskever, Ilya, Deoras, Anoop, Le, Hai-Son, Kombrink, Stefan, and Cernocky, Jan. Subword Language Modeling with Neural Networks. *Preprint*, 2012. See p. 76.
- Miller, John and Hardt, Moritz. Stable Recurrent Models. In *International Conference on Learning Representations*, 2019. See p. 24.
- Minsky, Marvin. *The Society of Mind*. Simon and Schuster, 1985. See p. 20.
- Mishra, Nikhil, Rohaninejad, Mostafa, Chen, Xi, and Abbeel, Pieter. A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*, 2018. See pp. 24, 60, 61, 96, and 128.
- Mitchell, Tom M. *Machine Learning*. McGraw-Hill, New York, 1997. See p. 32.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013. See pp. 45 and 113.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540): 529–533, 2015. See pp. 46 and 167.
- Moerland, Thomas M., Broekens, Joost, and Jonker, Catholijn M. Efficient Exploration with Double Uncertain Value Networks. In *Advances in Neural Information Processing Systems*, 2017. See pp. 161 and 174.
- Monett, Dagmar, Lewis, Colin, and Thórisson, Kristinn. On Defining Artificial Intelligence—Commentaries and Author’s Response. *Journal of Artificial General Intelligence*, 11:1–100, 02 2020. See p. 21.
- Muandet, Krikamol, Balduzzi, David, and Schölkopf, Bernhard. Domain Generalization via Invariant Feature Representation. In *International Conference on Learning Representations*, 2013. See p. 53.
- Mujika, Asier, Meier, Florian, and Steger, Angelika. Fast-Slow Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, 2017. See pp. 135 and 149.
- Munkhdalai, Tsendsuren and Yu, Hong. Meta Networks. In *International Conference on Learning Representations*, 2017. See pp. 56 and 61.
- Munkhdalai, Tsendsuren, Yuan, Xingdi, Mehri, Soroush, Wang, Tong, and Trischler, Adam. Learning Rapid-Temporal Adaptations. In *International Conference on Machine Learning*, 2018. See pp. 60 and 128.



- Murphy, Kevin P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. See pp. 34, 35, 37, and 39.
- Nachum, Ofir, Norouzi, Mohammad, and Schuurmans, Dale. Improving Policy Gradient by Exploring Under-Appreciated Rewards. In *International Conference on Learning Representations*, 2016. See p. 174.
- Neyshabur, Behnam, Bhojanapalli, Srinadh, McAllester, David, and Srebro, Nati. Exploring Generalization in Deep Learning. In *Advances in Neural Information Processing Systems*, 2017. See p. 27.
- Nguyen, Cuong V., Li, Yingzhen, Bui, Thang D., and Turner, Richard E. Variational Continual Learning. In *International Conference on Learning Representations*, 2018. See p. 54.
- Nichol, Alex, Achiam, Joshua, and Schulman, John. On First-Order Meta-Learning Algorithms. *arXiv preprint ArXiv:1803.02999*, 2018. See pp. 61, 94, 96, 106, 118, 128, 129, and 143.
- Nocedal, Jorge and Wright, Stephen J. *Numerical Optimization*. Springer, 2006. See pp. 35, 118, and 128.
- Norouzi, Mohammad, Mikolov, Tomas, Bengio, Samy, Singer, Yoram, Shlens, Jonathon, Frome, Andrea, Corrado, Greg S., and Dean, Jeffrey. Zero-Shot Learning by Convex Combination of Semantic Embeddings. In *International Conference on Learning Representations*, 2014. See p. 59.
- Novak, Roman, Bahri, Yasaman, Abolafia, Daniel A., Pennington, Jeffrey, and Sohl-Dickstein, Jascha. Sensitivity and Generalization in Neural Networks: an Empirical Study. In *International Conference on Learning Representations*, 2018. See pp. 19, 21, 27, and 68.
- Oh, Junhyuk, Hessel, Matteo, Czarnecki, Wojciech M., Xu, Zhongwen, van Hasselt, Hado, Singh, Satinder, and Silver, David. Discovering Reinforcement Learning Algorithms. *arXiv preprint arXiv:2007.08794*, 2020. See p. 50.
- Oreshkin, Boris N., Lacoste, Alexandre, and Rodriguez, Paul. TADAM: Task Dependent Adaptive Metric for Improved Few-Shot Learning. In *Advances in Neural Information Processing Systems*, 2018. See pp. 62, 126, and 128.
- Osband, Ian, Russo, Daniel, and Van Roy, Benjamin. (More) Efficient Reinforcement Learning via Posterior Sampling. In *Advances in Neural Information Processing Systems*, 2013. See p. 161.
- Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems*, 2016a. See pp. 161, 164, 167, 173, 180, and 190.
- Osband, Ian, Van Roy, Benjamin, and Wen, Zheng. Generalization and Exploration via Randomized Value Functions. In *International Conference on Machine Learning*, 2016b. See pp. 161 and 168.
- Osband, Ian, Aslanides, John, and Cassirer, Albin. Randomized Prior Functions for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2018. See p. 168.
- Osband, Ian, Van Roy, Benjamin, Russo, Daniel J., and Wen, Zheng. Deep Exploration

- via Randomized Value Functions. *Journal of Machine Learning Research*, 20:1–62, 2019. See pp. 161, 174, and 192.
- Osband, Ian, Doron, Yotam, Hessel, Matteo, Aslanides, John, Sezener, Eren, Saraiva, Andre, McKinney, Katrina, Lattimore, Tor, Szepezvari, Csaba, Singh, Satinder, Roy, Benjamin Van, Sutton, Richard, Silver, David, and Hasselt, Hado Van. Behaviour Suite for Reinforcement Learning. In *International Conference on Learning Representations*, 2020. See pp. 167, 169, 176, and 189.
- Ostrovski, Georg, Bellemare, Marc G., van den Oord, Aäron, and Munos, Rémi. Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning*, 2017. See pp. 173 and 174.
- O’Sullivan, Joseph, Langford, John, Caruana, Rich, and Blum, Avrim. FeatureBoost: A Meta-Learning Algorithm that Improves Model Robustness. In *International Conference on Machine Learning*, 2000. See p. 51.
- Oudeyer, Pierre-Yves and Kaplan, Frederic. What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in Neurorobotics*, 1:6, 2009. See p. 174.
- Oudeyer, Pierre-Yves, Kaplan, Frdric, and Hafner, Verena V. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007. See pp. 28 and 175.
- O’Donoghue, Brendan, Osband, Ian, Munos, Remi, and Mnih, Volodymyr. The Uncertainty Bellman Equation and Exploration. In *International Conference on Machine Learning*, 2018. See pp. 161, 163, 164, 166, 174, and 180.
- Packer, Charles, Gao, Katelyn, Kos, Jernej, Krähenbühl, Philipp, Koltun, Vladlen, and Song, Dawn. Assessing Generalization in Deep Reinforcement Learning. *arXiv preprint arXiv:1810.12282*, 2018. See p. 207.
- Pan, Sinno Jialin and Yang, Qiang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge & Data Engineering*, 12(10):1345–1359, 2009. See pp. 53, 86, and 95.
- Parisi, German I., Kemker, Ronald, Part, Jose L., Kanan, Christopher, and Wermter, Stefan. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 113:54–71, 2019. See p. 53.
- Park, Eunbyung and Oliva, Junier B. Meta-curvature. In *Advances in Neural Information Processing Systems*, 2019. See pp. 26, 62, 118, and 120.
- Pascanu, Razvan and Bengio, Yoshua. Revisiting Natural Gradient for Deep Networks. In *International Conference on Learning Representations*, 2014. See pp. 90 and 128.
- Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. Understanding the Exploding Gradient Problem. *arXiv preprint arXiv:1211.5063*, 2013. See p. 24.
- Pathak, Deepak, Agrawal, Pulkit, Efros, Alexei A., and Darrell, Trevor. Curiosity-Driven Exploration by Self-Supervised Prediction. In *International Conference on Machine Learning*, 2017. See pp. 161, 174, and 175.
- Peng, Jing and Williams, Ronald J. Incremental Multi-Step Q-Learning. In *Machine Learning Proceedings 1994*, pp. 226–232. Elsevier, 1994. See p. 196.
- Perez, Ethan, Strub, Florian, De Vries, Harm, Dumoulin, Vincent, and Courville, Aaron. Film: Visual Reasoning With a General Conditioning Layer. In *Association for the Advancement of Artificial Intelligence*, 2018. See pp. 135 and 143.

- Plappert, Matthias, Houthooft, Rein, Dhariwal, Prafulla, Sidor, Szymon, Chen, Richard Y., Chen, Xi, Asfour, Tamim, Abbeel, Pieter, and Andrychowicz, Marcin. Parameter Space Noise for Exploration. In *International Conference on Learning Representations*, 2018. See pp. 164, 174, and 182.
- Pratt, Lorien Y. Discriminability-Based Transfer Between Neural Networks. In *Advances in Neural Information Processing Systems*, 1993. See p. 52.
- Press, Ofir and Wolf, Lior. Using the Output Embedding to Improve Language Models. In *Proceedings of the European Chapter of the Association for Computational Linguistics*, volume 2, pp. 157–163, 2017. See pp. 80 and 83.
- Puigdomènech Badia, Adrià, Sprechmann, Pablo, Vitvitskyi, Alex, Guo, Daniel, Piot, Bilal, Kapturowski, Steven, Tieleman, Olivier, Arjovsky, Martin, Pritzel, Alexander, Bolt, Andrew, and Blundell, Charles. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*, 2020. See pp. 173, 174, 192, and 197.
- Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014. See p. 42.
- Qiao, Siyuan, Liu, Chenxi, Shen, Wei, and Yuille, Alan L. Few-Shot Image Recognition by Predicting Parameters from Activations. In *Computer Vision and Pattern Recognition*, 2018. See pp. 60 and 128.
- Racaniere, Sébastien, Lampinen, Andrew K., Santoro, Adam, Reichert, David P., Firoiu, Vlad, and Lillicrap, Timothy P. Automated Curricula Through Setter-Solver Interactions. In *International Conference on Learning Representations*, 2020. See p. 57.
- Radford, Alec, Jozefowicz, Rafal, and Sutskever, Ilya. Learning to Generate Reviews and Discovering Sentiment. *arXiv preprint, arXiv:1704.01444*, 2017. See p. 76.
- Radford, Alec, Wu, Jeffrey, Child, Rewon, Luan, David, Amodei, Dario, and Sutskever, Ilya. Language Models are Unsupervised Multitask Learners. OpenAI Blog, 2019. URL <https://openai.com/blog/better-language-models/>. See p. 19.
- Raghu, Aniruddh, Raghu, Maithra, Bengio, Samy, and Vinyals, Oriol. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *International Conference on Learning Representations*, 2020. See p. 27.
- Ravi, Sachin and Beatson, Alex. Amortized Bayesian Meta-Learning. In *International Conference on Learning Representations*, 2018. See pp. 53 and 62.
- Ravi, Sachin and Larochelle, Hugo. Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*, 2017. See pp. 24, 50, 60, 61, 75, 87, 96, 117, 118, 127, 129, and 146.
- Real, Esteban, Liang, Chen, So, David R., and Le, Quoc V. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *International Conference on Machine Learning*, 2020. See pp. 23, 51, and 54.
- Rebuffi, Sylvestre-Alvise, Bilen, Hakan, and Vedaldi, Andrea. Learning Multiple Visual Domains with Residual Adapters. In *Advances in Neural Information Processing Systems*, 2017. See pp. 52, 125, 134, 135, and 143.
- Ren, Mengye, Triantafillou, Eleni, Ravi, Sachin, Snell, Jake, Swersky, Kevin, Tenenbaum, Joshua B., Larochelle, Hugo, and Zemel, Richard S. Meta-Learning for

- Semi-Supervised Few-Shot Classification. In *International Conference on Learning Representations*, 2018. See pp. 119, 128, 129, and 146.
- Ren, Mengye, Liao, Renjie, Fetaya, Ethan, and Zemel, Richard. Incremental Few-Shot Learning with Attention Attractor Networks. In *Advances in Neural Information Processing Systems*, 2019. See p. 60.
- Riedmiller, Martin A., Hafner, Roland, Lampe, Thomas, Neunert, Michael, Degraeve, Jonas, de Wiele, Tom Van, Mnih, Volodymyr, Heess, Nicolas, and Springenberg, Jost Tobias. Learning by Playing - Solving Sparse Reward Tasks from Scratch. *arXiv preprint arXiv:1802.10567*, 2018. See p. 28.
- Riemer, Matthew, Cases, Ignacio, Ajemian, Robert, Liu, Miao, Rish, Irina, Tu, Yuhai, and Tesauro, Gerald. Learning to Learn without Forgetting by Maximizing Transfer and Minimizing Interference. In *International Conference on Learning Representations*, 2018. See p. 53.
- Risi, Sebastian and Togelius, Julian. Increasing Generality in Machine Learning Through Procedural Content Generation. *Nature Machine Intelligence*, pp. 1–9, 2020. See p. 57.
- Ritter, Hippolyt, Botev, Aleksandar, and Barber, David. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. In *Advances in Neural Information Processing Systems*, 2018. See p. 54.
- Rodríguez, Pau, Laradji, Issam, Drouin, Alexandre, and Lacoste, Alexandre. Embedding Propagation: Smoother Manifold for Few-Shot Classification. *arXiv preprint arXiv:2003.04151*, 2020. See p. 60.
- Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. Learning Representations by Back-Propagating Errors. *nature*, 323(6088):533–536, 1986. See p. 39.
- Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. Imagenet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. See p. 52.
- Rusu, Andrei A., Rabinowitz, Neil C., Desjardins, Guillaume, Soyer, Hubert, Kirkpatrick, James, Kavukcuoglu, Koray, Pascanu, Razvan, and Hadsell, Raia. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016. See pp. 53, 96, 98, and 111.
- Rusu, Andrei A., Rao, Dushyant, Sygnowski, Jakub, Vinyals, Oriol, Pascanu, Razvan, Osindero, Simon, and Hadsell, Raia. Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations*, 2019. See pp. 27, 62, 126, and 128.
- Santoro, Adam, Bartunov, Sergey, Botvinick, Matthew, Wierstra, Daan, and Lillicrap, Timothy. Meta-Learning with Memory-Augmented Neural Networks. In *International Conference on Machine Learning*, 2016. See pp. 22, 61, 87, and 96.
- Saxe, Andrew M., McClelland, James L., and Ganguli, Surya. Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks. *arXiv preprint arXiv:1312.6120*, 2013. See pp. 40, 73, and 128.
- Schäfer, Anton Maximilian and Zimmermann, Hans-Georg. Recurrent Neural Networks are Universal Approximators. *International Journal of Neural Systems*, 17(04):253–263, 2007. See pp. 61 and 118.

- Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952*, 2015. See p. 46.
- Schaul, Tom, Borsa, Diana, Ding, David, Szepesvari, David, Ostrovski, Georg, Dabney, Will, and Osindero, Simon. Adapting Behaviour for Learning Progress. *arXiv preprint arXiv:1912.06910*, 2019. See pp. 173, 192, 193, and 197.
- Schmidhuber, Juergen, Zhao, Jieyu, and Wiering, M.A. Simple Principles of Metalearning. *Technical report IDSIA*, 69:1–23, 1996. See p. 54.
- Schmidhuber, Jürgen. *Evolutionary Principles in Self-Referential Learning*. PhD thesis, Technische Universität München, 1987. See pp. 21, 22, 23, 24, 54, 57, 66, 67, 96, and 127.
- Schmidhuber, Jürgen. Curious Model-Building Control Systems. In *Proceedings of the International Joint Conference on Neural Networks*, 1991. See pp. 28, 174, and 175.
- Schmidhuber, Jürgen. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, 1992. See pp. 56, 75, 127, and 149.
- Schmidhuber, Jürgen. Powerplay: Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem. *Frontiers in psychology*, 4:313, 2013. See p. 57.
- Schwarz, Jonathan, Luketina, Jelena an Czarnecki, Wojciech M., Grabska-Barwinska, Agnieszka, Teh, Yee Whye, Pascanu, Razvan, and Hadsell, Raia. Progress & Compress: A Scalable Framework for Continual Learning. In *International Conference on Machine Learning*, 2018. See pp. 53, 96, and 109.
- Seo, Sanghyun, Jeon, Yongjin, and Kim, Juntae. Meta Learning for Imbalanced Big Data Analysis by Using Generative Adversarial Networks. In *International Conference on Big Data and Computing*, 2018. See p. 50.
- Serrá, Joan, Surís, Dídac, Miron, Marius, and Karatzoglou, Alexandros. Overcoming Catastrophic Forgetting with Hard Attention to the Task. In *International Conference on Machine Learning*, 2018. See pp. 53, 95, 98, and 111.
- Shaban, Amirreza, Bansal, Shray, Liu, Zhen, Essa, Irfan, and Boots, Byron. One-Shot Learning for Semantic Segmentation. In *British Machine Vision Conference*, 2017. See p. 60.
- Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan, and de Freitas, Nando. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 2016. See p. 54.
- Shao, Hang, Kumar, Abhishek, and Fletcher, P. Thomas. The Riemannian Geometry of Deep Generative Models. In *Computer Vision and Pattern Recognition*, 2018. See p. 97.
- Sharif Razavian, Ali, Azizpour, Hossein, Sullivan, Josephine, and Carlsson, Stefan. CNN Features Off-The-Shelf: an Astounding Baseline for Recognition. In *Computer Vision and Pattern Recognition*, 2014. See p. 52.
- Shin, Hanul, Lee, Jung Kwon, Kim, Jaehong, and Kim, Jiwon. Continual Learning with Deep Generative Replay. In *Advances in Neural Information Processing Systems*, 2017. See p. 53.

- Siegelmann, Hava T. and Sontag, Eduardo D. On the Computational Power of Neural Nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995. See pp. 24 and 61.
- Silver, David, Hubert, Thomas, Schrittwieser, Julian, Antonoglou, Ioannis, Lai, Matthew, Guez, Arthur, Lanctot, Marc, Sifre, Laurent, Kumaran, Dharshan, Graepel, Thore, Lillicrap, Timothy, Simonyan, Karen, and Hassabis, Demis. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play. *Science*, 362(6419):1140–1144, 2018. See pp. 18 and 207.
- Silvetti, Massimo and Verguts, Tom. Reinforcement Learning, High-Level Cognition, and The Human Brain. *Neuroimaging–Cognitive and Clinical Neuroscience*, pp. 283–96, 2012. See p. 41.
- Simmons-Edler, Riley, Eisner, Ben, Mitchell, Eric, Seung, H. Sebastian, and Lee, Daniel D. QXplore: Q-learning Exploration by Maximizing Temporal Difference Error. *arXiv preprint arXiv:1906.08189*, 2019. See pp. 171, 175, and 189.
- Singh, Satinder P., Barto, Andrew G., and Chentanez, Nuttapon. Intrinsically Motivated Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2005. See pp. 161 and 204.
- Sinz, Fabian H., Pitkow, Xaq, Reimer, Jacob, Bethge, Matthias, and Tolias, Andreas S. Engineering a less artificial intelligence. *Neuron*, 103(6):967–979, 2019. See pp. 20 and 21.
- Snell, Jake, Swersky, Kevin, and Zemel, Richard S. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*, 2017. See pp. 50, 57, 60, 96, and 128.
- Spelke, Elizabeth S. and Kinzler, Katherine D. Core Knowledge. *Developmental science*, 10(1):89–96, 2007. See p. 21.
- Stanley, Kenneth O., D’Ambrosio, David B., and Gauci, Jason. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, 2009. See p. 75.
- Strens, Malcolm. A Bayesian Framework for Reinforcement Learning. In *International Conference on Machine Learning*, 2000. See pp. 161, 163, 173, and 204.
- Suarez, Joseph. Character-Level Language Modeling with Recurrent Highway Hypernetworks. In *Advances in Neural Information Processing Systems*, 2017. See pp. 75, 76, and 135.
- Such, Felipe Petroski, Rawal, Aditya, Lehman, Joel, Stanley, Kenneth, and Clune, Jeffrey. Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data. In *International Conference on Machine Learning*, 2020. See p. 208.
- Sukhbaatar, Sainbayar, Kostrikov, Ilya, Szlam, Arthur, and Fergus, Rob. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *International Conference on Learning Representations*, 2018. See pp. 28, 29, 57, 158, and 207.
- Sung, Flood, Yang, Yongxin, Zhang, Li, Xiang, Tao, Torr, Philip H. S., and Hospedales, Timothy M. Learning to Compare: Relation Network for Few-Shot Learning. In *Computer Vision and Pattern Recognition*, 2018. See pp. 50 and 60.



- Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. Generating Text with Recurrent Neural Networks. In *International Conference on Machine Learning*, 2011. See p. 76.
- Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. On the Importance of Initialization and Momentum in Deep Learning. In *International Conference on Machine Learning*, 2013. See pp. 39 and 128.
- Sutton, Richard S. and Barto, Andrew G. *Introduction to Reinforcement Learning*. MIT Press, 1998. See pp. 41, 42, 43, 44, 45, 47, and 100.
- Sutton, Richard S., McAllester, David A., Singh, Satinder, and Mansour, Yishay. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, 1999. See p. 47.
- Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian, and Fergus, Rob. Intriguing Properties of Neural Networks. In *International Conference on Learning Representations*, 2014. See pp. 19, 21, and 40.
- Szepesvári, Csaba. Algorithms for Reinforcement Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010. See p. 45.
- Thompson, William R. On the Likelihood that one Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294, 1933. See p. 174.
- Thórisson, Kristinn and Helgasson, Helgi. Cognitive Architectures and Autonomy: A Comparative Review. *Journal of Artificial General Intelligence*, 3(2):1–30, 2012. See pp. 20 and 22.
- Thórisson, Kristinn R., Bieger, Jordi, Li, Xiang, and Wang, Pei. Cumulative Learning. In *Artificial General Intelligence*, 2019. See p. 22.
- Thrun, Sebastian and Pratt, Lorien. Learning To Learn: Introduction and Overview. In *In Learning To Learn*. Springer, 1998. See pp. 23, 54, 55, and 127.
- Tian, Yonglong, Wang, Yue, Krishnan, Dilip, Tenenbaum, Joshua B., and Isola, Phillip. Rethinking Few-Shot Image Classification: a Good Embedding Is All You Need? In *European Conference on Computer Vision*, 2020. See p. 25.
- Tokic, Michel. Adaptive  $\epsilon$ -Greedy Exploration in Reinforcement Learning Based on Value Differences. In *Annual Conference on Artificial Intelligence*, pp. 203–210, 2010. See p. 174.
- Tosi, Alessandra, Hauberg, Søren, Vellido, Alfredo, and Lawrence, Neil D. Metrics for Probabilistic Geometries. In *Conference on Uncertainty in Artificial Intelligence*, 2014. See p. 97.
- Triantafillou, Eleni, Zhu, Tyler, Dumoulin, Vincent, Lamblin, Pascal, Xu, Kelvin, Goroshin, Ross, Gelada, Carles, Swersky, Kevin, Manzagol, Pierre-Antoine, and Larochelle, Hugo. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. In *International Conference on Learning Representations*, 2020. See pp. 25 and 55.
- Tseng, Hung-Yu, Lee, Hsin-Ying, Huang, Jia-Bin, and Yang, Ming-Hsuan. Cross-Domain Few-Shot Classification via Learned Feature-Wise Transformation. In *International Conference on Learning Representations*, 2019. See pp. 27 and 53.

- Turing, Alan M. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 10 1950. See pp. 21 and 22.
- Turovsky, Barak. Ten Years of Google Translate. Google Blog, 2016. URL <https://www.blog.google/products/translate/ten-years-of-google-translate/>. See p. 19.
- van Erven, Tim and Koolen, Wouter M. MetaGrad: Multiple Learning Rates in Online Learning. In *Advances in Neural Information Processing Systems*, 2016. See p. 52.
- Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep Reinforcement Learning with Double Q-Learning. In *Association for the Advancement of Artificial Intelligence*, 2016. See pp. 46 and 167.
- van Rijn, Jan N., Abdulrahman, Salisu Mamman, Brazdil, Pavel, and Vanschoren, Joaquin. Fast Algorithm Selection Using Learning Curves. In *International Symposium on Intelligent Data Analysis*, 2015. See p. 50.
- Vanschoren, Joaquin. Meta-Learning: A Survey. *arXiv preprint arXiv:1810.03548*, 2018. See pp. 23 and 54.
- Vapnik, Vladimir N. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1999. See pp. 37 and 38.
- Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia. Attention is All you Need. In *Advances in Neural Information Processing Systems*, 2017. See p. 80.
- Vilalta, Ricardo and Drissi, Youssef. A Perspective View and Survey of Meta-Learning. *Artificial intelligence review*, 18(2):77–95, 2002. See pp. 23 and 54.
- Vilalta, Ricardo, Giraud-Carrier, Christophe G., Brazdil, Pavel, and Soares, Carlos. Using Meta-Learning to Support Data Mining. *Using Meta-Learning to Support Data Mining*, 1(1):31–45, 2004. See pp. 50 and 54.
- Vinyals, Oriol, Blundell, Charles, Lillicrap, Timothy, Kavukcuoglu, Koray, and Wierstra, Daan. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems*, 2016. See pp. 22, 23, 24, 25, 50, 51, 57, 59, 60, 87, 96, 109, 118, 128, 129, 141, 146, and 210.
- Vinyals, Oriol, Babuschkin, Igor, Czarnecki, Wojciech M., Mathieu, Michaël, Dudzik, Andrew, Chung, Junyoung, Choi, David H., Powell, Richard, Ewalds, Timo, Georgiev, Petko, Oh, Junhyuk, Horgan, Dan, Kroiss, Manuel, Danihelka, Ivo, Huang, Aja, Sifre, Laurent, Cai, Trevor, Agapiou, John P., Jaderberg, Max, Vezhnevets, Alexander S., Leblond, Rémi, Pohlen, Tobias, Dalibard, Valentin, Budden, David, Sulsky, Yury, Molloy, James, Paine, Tom L., Gulcehre, Caglar, Wang, Ziyu, Pfaff, Tobias, Wu, Yuhuai, Ring, Roman, Yogatama, Dani, McKinney, Katrina, Smith, Oliver, Schaul, Tom, Lillicrap, Timothy, Kavukcuoglu, Koray, Hassabis, Demis, Apps, Chris, and Silver, David. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 575(7782):350–354, 2019. See pp. 18 and 207.
- Wald, Abraham. Note on the Consistency of the Maximum Likelihood Estimate. *Annals of Mathematical Statistics*, 20(4):595–601, 12 1949. See p. 36.
- Wang, Jane X., Kurth-Nelson, Zeb, Tirumala, Dhruva, Soyer, Hubert, Leibo, Joel Z., Munos, Rémi, Blundell, Charles, Kumaran, Dharshan, and Botvinick, Matthew. Learning to Reinforcement Learn. In *Annual Meeting of the Cognitive Science Society*, 2016a. See pp. 55, 127, 131, 132, 138, 148, and 206.



- Wang, Pei. On Defining Artificial Intelligence. *Journal of Artificial General Intelligence*, 10(2):1–37, 2019. See pp. 20, 21, 22, and 210.
- Wang, Rui, Lehman, Joel, Clune, Jeff, and Stanley, Kenneth O. Paired Ppen-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv preprint arXiv:1901.01753*, 2019. See pp. 57 and 158.
- Wang, Ziyu, Schaul, Tom, Hessel, Matteo, Hasselt, Hado, Lanctot, Marc, and Freitas, Nando. Dueling Network Architectures for Deep Reinforcement Learning. In *International Conference on Learning Representations*, pp. 1995–2003, 2016b. See pp. 46 and 61.
- Wasserman, Larry. *All of Statistics: a Concise Course in Statistical Inference*. Springer Science & Business Media, 2013. See pp. 36 and 38.
- Watkins, Christopher and Dayan, Peter. Q-Learning. *Machine learning*, 8(3-4):279–292, 1992. See p. 45.
- White, Adam. *Developing a Predictive Approach to Knowledge*. PhD thesis, University of Alberta, 2015. See p. 175.
- Williams, Ronald J. and Peng, Jing. Function Optimization using Connectionist Reinforcement Learning Algorithms. *Connection Science*, 3(3):241–268, 1991. See pp. 46 and 161.
- Wolf, Marty J., Miller, Keith W, and Grodzinsky, Frances S. Why We Should Have Seen That Coming: Comments on Microsoft’s Tay “Experiment,” and Wider Implications. *The ORBIT Journal*, 1(2):1–12, 2017. See p. 209.
- Wolpert, David H. and Macready, William G. No Free Lunch Theorems For Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. See p. 49.
- Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Lukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa, Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff, and Dean, Jeffrey. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016a. See p. 19.
- Wu, Yuhuai, Zhang, Saizheng, Zhang, Ying, Bengio, Yoshua, and Salakhutdinov, Ruslan. On Multiplicative Integration with Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, 2016b. See p. 76.
- Wu, Yuhuai, Ren, Mengye, Liao, Renjie, and Grosse, Roger B. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. In *International Conference on Learning Representations*, 2018. See pp. 62, 84, 87, 97, and 120.
- Xu, Tianbing, Liu, Qiang, Zhao, Liang, and Peng, Jian. Learning to Explore via Meta-Policy Gradient. In *International Conference on Machine Learning*, 2018a. See pp. 56, 173, and 198.
- Xu, Zhongwen, van Hasselt, Hado P., and Silver, David. Meta-Gradient Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2018b. See pp. 47, 51, 52, and 211.

- Xu, Zhongwen, van Hasselt, Hado, Hessel, Matteo, Oh, Junhyuk, Singh, Satinder, and Silver, David. Meta-Gradient Reinforcement Learning with an Objective Discovered Online. In *Advances in Neural Information Processing Systems*, 2020. See p. 211.
- Yang, Zhilin, Dai, Zihang, Salakhutdinov, Ruslan, and Cohen, William W. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. In *International Conference on Learning Representations*, 2018. See pp. 75 and 78.
- Yang, Zhilin, Dai, Zihang, Yang, Yiming, Carbonell, Jaime G., Salakhutdinov, Ruslan, and Le, Quoc V. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems*, 2019. See p. 52.
- Yao, Quanming, Wang, Mengshuo, Escalante, Hugo Jair, Guyon, Isabelle, Hu, Yi-Qi, Li, Yu-Feng, Tu, Wei-Wei, Yang, Qiang, and Yu, Yang. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. *arXiv preprint arXiv:1810.13306*, 2018a. See pp. 23, 25, and 54.
- Yao, Shuochao, Zhao, Yiran, Zhang, Aston, Hu, Shaohan, Shao, Huajie, Zhang, Chao, Su, Lu, and Abdelzaher, Tarek. Deep Learning for The Internet of Things. *Computer*, 51(5):32–41, 2018b. See p. 209.
- Yin, Chengxiang, Tang, Jian, Xu, Zhiyuan, and Wang, Yanzhi. Adversarial Meta-Learning. *arXiv preprint arXiv:1806.03316*, 2018. See p. 23.
- Yu, Tianhe, Quillen, Deirdre, He, Zhanpeng, Julian, Ryan, Hausman, Karol, Finn, Chelsea, and Levine, Sergey. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning*, 2020. See pp. 25 and 207.
- Zahavy, Tom, Xu, Zhongwen, Veeriah, Vivek, Hessel, Matteo, Oh, Junhyuk, van Hasselt, Hado, Silver, David, and Singh, Satinder. Self-Tuning Deep Reinforcement Learning. *arXiv preprint arXiv:2002.12928*, 2020. See pp. 52, 173, 192, 198, and 211.
- Zamir, Amir R., Sax, Alexander, Shen, William B., Guibas, Leonidas J., Malik, Jitendra, and Savarese, Silvio. Taskonomy: Disentangling Task Transfer Learning. In *Computer Vision and Pattern Recognition*, 2018. See p. 25.
- Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent Neural Network Regularization. In *International Conference on Learning Representations*, 2015. See pp. 78 and 80.
- Zenke, Friedemann, Poole, Ben, and Ganguli, Surya. Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning*, 2017. See pp. 53 and 95.
- Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding Deep Learning Requires Rethinking Generalization. In *International Conference on Learning Representations*, 2017. See pp. 19, 21, 27, 40, and 66.
- Zhang, Ruixiang, Che, Tong, Ghahramani, Zoubin, Bengio, Yoshua, and Song, Yangqiu. MetaGAN: An Adversarial Approach to Few-Shot Learning. In *Advances in Neural Information Processing Systems*, 2018. See pp. 50 and 51.
- Zheng, Zeyu, Oh, Junhyuk, Hessel, Matteo, Xu, Zhongwen, Kroiss, Manuel, Van Hasselt, Hado, Silver, David, and Singh, Satinder. What Can Learned Intrinsic Rewards Capture? In *International Conference on Machine Learning*, 2020. See p. 158.

- Zhou, Fan, Cao, Chengtai, Zhang, Kunpeng, Trajcevski, Goce, Zhong, Ting, and Geng, Ji. Meta-GNN: On Few-shot Node Classification in Graph Meta-learning. In *International Conference on Information and Knowledge Management*, 2019. See p. 62.
- Zhou, Fengwei, Wu, Bin, and Li, Zhenguo. Deep Meta-Learning: Learning to Learn in the Concept Space. *arXiv preprint arXiv:1802.03596*, 2018. See pp. 62 and 128.
- Ziebart, Brian D., Maas, Andrew, Bagnell, J. Andrew, and Dey, Anind K. Maximum Entropy Inverse Reinforcement Learning. In *Association for the Advancement of Artificial Intelligence*, 2008. See p. 161.
- Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutnik, Jan, and Schmidhuber, Jürgen. Recurrent Highway Networks. In *International Conference on Machine Learning*, 2017. See pp. 75 and 80.
- Zintgraf, Luisa M., Shiarlis, Kyriacos, Kurin, Vitaly, Hofmann, Katja, and Whiteson, Shimon. Fast Context Adaptation via Meta-Learning. *International Conference on Machine Learning*, 2019. See pp. 128 and 134.
- Zoph, Barret and Le, Quoc V. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, 2017. See p. 80.