Emotion-Aware and Human-Like Autonomous Agents

by

Nabiha Asghar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Pascal Poupart Professor, Cheriton School of Computer Science, University of Waterloo
External Examiner:	Frank Rudzicz Associate Professor, Dept. of Computer Science, University of Toronto
Internal Members:	Jesse Hoey Associate Professor, Cheriton School of Computer Science, University of Waterloo
	Ming Li Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Olga Vechtomova Associate Professor, Dept. of Management Sciences, University of Waterloo

This thesis consists of material all of which I authored or co-authored: see *Statement of Contributions* included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The work presented in this thesis is based on the following research papers:

Chapter 3:

- **Nabiha Asghar*** and Jesse Hoey*. Monte-Carlo Planning for Socially Aligned Agents using Bayesian Affect Control Theory. *Technical Report # CS-2014-21, University of Waterloo School of Computer Science*, 2014. [9]
- **Nabiha Asghar*** and Jesse Hoey*. Intelligent Affect: Rational Decision Making for Socially Aligned Agents. In *31st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015. [10]

Chapter 4:

- **Nabiha Asghar**, Pascal Poupart, Jesse Hoey, Xin Jiang, and Lili Mou. Affective Neural Response Generation. In *40th European Conference on Information Retrieval (ECIR)*, 2018. [12]

Chapter 5:

- **Nabiha Asghar**, Pascal Poupart, Xin Jiang, and Hang Li. Deep Active Learning for Dialogue Generation. In *6th Joint Conference on Lexical and Computational Semantics (*SEM)*, 2017. [13]

Chapter 6:

- **Nabiha Asghar***, Lili Mou*, Kira A. Selby, Kevin D. Pantasdo, Pascal Poupart, and Xin Jiang. Progressive Memory Banks for Incremental Domain Adaptation. To appear in *International Conference on Learning Representations (ICLR)*, 2020. [11]

* denotes equal contribution

Abstract

In human-computer interaction (HCI), one of the technological goals is to build human-like artificial agents that can think, decide and behave like humans during the interaction. A prime example is a dialogue system, where the agent should converse fluently and coherently with a user and connect with them emotionally. Humanness and emotion-awareness of interactive artificial agents have been shown to improve user experience and help attain application-specific goals more quickly [45]. However, achieving human-likeness in HCI systems is contingent on addressing several philosophical and scientific challenges. In this thesis, I address two such challenges: replicating the human ability to 1) correctly perceive and adopt emotions, and 2) communicate effectively through language.

Several research studies in neuroscience, economics, psychology and sociology show that both language and emotional reasoning are essential to the human cognitive deliberation process [41, 89, 90]. These studies establish that any human-like AI should necessarily be equipped with adequate emotional and linguistic cognizance. To this end, I explore the following research directions.

- I study how agents can reason emotionally in various human-interactive settings for decision-making. I use Bayesian Affect Control Theory [81], a probabilistic model of human-human affective interactions, to build a decision-theoretic reasoning algorithm about affect. This approach is validated on several applications: two-person social dilemma games, an assistive healthcare device, and robot navigation.

- I develop several techniques to understand and generate emotions/affect in language. The proposed methods include affect-based feature augmentation of neural conversational models, training regularization using affective objectives, and affectively diverse sequential inference.

- I devise an active learning technique that elicits user feedback during a conversation. This enables the agent to learn in real time, and to produce natural and coherent language during the interaction.

- I explore incremental domain adaptation in language classification and generation models. The proposed method seeks to replicate the human ability to continually learn from new environments without forgetting old experiences.

Acknowledgements

First and foremost, I thank my supervisor Prof. Pascal Poupart for being a constant source of support, encouragement and appreciation. He gave me the freedom to pursue research topics of my own choice, and showed immense flexibility around my family and work constraints. Being Pascal's student has opened doors for me both in academia and the industry, and I truly could not have asked for more. Prof. Jesse Hoey and Lili Mou played an instrumental role in helping me turn rough ideas into concrete research, and I am eternally grateful to them. I also thank my committee members, Prof. Olga Vechtomova, Prof. Frank Rudzicz and Prof. Ming Li for providing valuable feedback.

A special thank-you goes to the administrative staff in the CS department: Margaret Towell, Paula Roser, Gordon Boerke and Greg Mctavish. A big shout-out to my UW friends and fellow researchers Kira Selby, Mike Rudd, Cristina Tavares, Dipti Kumar, Ivan Kobyzev, Amira Ghenai, Priyank Jaini, Sara Ross-Howe, Prarthana Bhattacharyya, Abdullah Rashwan, Ankit Vadehra and Gaurav Sahu, for making this journey more enjoyable.

This work would not have been possible if I did not have a strong support system around me: my husband Bilal, who stood between me and my fears, encouraged me to pursue a PhD and then saw me through it; my father Dr Asghar and my mother Dr Nasreen, who always trusted me to make the right decisions for myself; and my sisters Aalia and Sadia, because of whom I have strong belief in the power of faith and hard work.

Dedication

I dedicate this work to my beloved son Ibrahim.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human-computer interaction (HCI) is at the center of many artificial intelligence (AI) systems, including voice search, augmented reality, healthcare assistance, video games and automated customer service. As AI becomes increasingly mainstream, there is a pressing need to make the interactive experience seamless and engaging for the users, much like a human-human interaction. Thus, in many AI applications, the goal of the artificial agent is to *think, decide and act like humans*. A prime example is voice or text based conversational systems, where the agent should converse fluently and coherently with the user, and sound natural. In fact, the entire field of Artificial General Intelligence (AGI) is dedicated to building machines that can carry out all intellectual tasks that are humanly possible. However, there are several theoretical and empirical challenges associated with achieving human-likeness in AI. One of these challenges is to adequately perceive and produce human emotions (also called *affect*). The other challenge is to understand and generate human language.

Emotions play a significant role in how humans perceive, behave and make decisions in any given situation. This has been corroborated by research in neuroscience [41, 101], economics [4], psychology [89] and sociology [74]. These studies from very different fields show that emotional reasoning is an essential component of cognitive deliberation. Furthermore, human decisions are heavily reliant on the way humans interact with each other and their surroundings.

Language is another vital component of human cognition. The Sapir-Whorf hypothesis in Anthropology says that language influences our thinking process, and may even determine it [90]. Other studies in behavioral economics and psychology indicate that the structure of language may influence our behaviour as well as memory [34, 54]. For instance,

languages that grammatically equate the present and the future foster more future-oriented behavior (e.g., saving more money) [34]. Similarly, different languages capture event descriptions differently, which has important consequences for eye-witness memory [54]. In fact, psychologists have even suggested that internal dialogue (talking to oneself, also known as verbal thinking) is important for higher-level thinking and decision making [5].

In light of these studies, we can establish that any AI that seeks to interact effectively with humans should be equipped with both affective cognizance and adequate linguistic capabilities, at the very least. Simple positive-negative sentiment analysis or basic knowledge of linguistic rules is not enough. To this end, this thesis explores how to develop human-like AI by endowing machines with the ability to:

1. reason emotionally in various human-interactive settings to make decisions,
2. understand and generate affect in language,
3. produce coherent and fluent language, and
4. adaptively learn about multiple domains/topics through language.

In the following sections, I briefly delve into each of these aspects and describe the contributions of this thesis.

1.1 Affective Decision Making

Affect Control Theory (ACT) [74] is a useful tool for affective reasoning and decision-making in different interactive situations. It is a socio-mathematical theory of affective interactions between humans. ACT posits that humans learn and maintain a set of shared cultural affective sentiments about individuals, situations and events, and map these sentiments to a three dimensional continuous vector space. These mappings, which can be measured through large-scale user studies, encode a set of social prescriptions that lead to a highly desirable state of social order, or equilibrium. Humans use this affective ecosystem to make predictions about what others will do, and to guide their own behaviour. In addition, they always seek to increase the affective alignment with others. *BayesAct* [80, 81] generalizes ACT by modeling human-machine affective interactions as a partially-observable Markov decision process (POMDP). In *BayesAct*, affective states are represented by probability distributions, which allows decision-theoretic reasoning about affect.

For a given interactive setting and affective identities of the two interactants, ACT provides the optimal affective action (a single point in the 3D affective space) for maximizing alignment. However, humans are crafty and devious, and often use their cognitive abilities to go beyond these prescriptions. Given enough planning resources (e.g. time), they like to find strategies that are individually beneficial and culturally acceptable, while being affectively sub-optimal[1]. *BayesAct*, being a POMDP, allows an agent to explore this facet of human nature by letting it plan in the affective space.

In Chapter 3 of this thesis, starting from the principles of *BayesAct*, I explore how planning beyond cultural prescriptions can help an agent devise deceptive or manipulative strategies, much like humans.

1.1.1 Contributions

1. I describe how to use Monte-Carlo Tree Search to do planning in *BayesAct*. I propose the POMCP-C algorithm to handle the continuous states, actions, and observations in the *BayesAct* POMDP.

2. I demonstrate POMCP-C on several applications. First, in two toy social-dilemma games (Prisoner's Dilemma and Battle of the Sexes), I demonstrate the emergence of complex interactions between cognitive and emotional reasoning, such as deception leading to manipulation and altercasting. Second, I review experiments on a realistic, affect-aware health-care assistive device for dementia patients. Third, I present evidence that the proposed Monte-Carlo Tree Search variant can be effectively used for planning in non-affective domains too, namely, robot navigation.

1.2 Affective and Human-Like Conversational Agents

Conversational AI is a branch of HCI where an agent interacts with a user through written or verbal human language. Popular examples of conversational agents include Apple's Siri, Microsoft's Cortana and Amazon's Alexa, which are primarily task-oriented (e.g. can search the web or call a friend), but are also capable of carrying out open-domain chit-chat with the user. In this thesis, I focus on building open-domain text-based conversational

[1] An example of such strategies is a mother who acts strict with her kid in order to make him/her study hard for an exam. In this case, the affective alignment between the mother and the kid is low, but the strategy is beneficial both for the kid (high score in exam) and the mother (a successful kid).

agents, which can carry out fluent and human-sounding conversations with users and are not restricted to particular topics/domains. Even in a task-oriented setting, open-domain conversational ability is important to handle unforeseen user queries.

With the immense success of neural networks, important breakthroughs have been made in natural language generation and, in particular, dialogue generation. Models adhering to the neural encoder-decoder framework, such as sequence-to-sequence [185], are common and popular. However, they are prone to producing short, dull and vague responses. In most of these systems, *word embeddings* (trained in an unsupervised fashion) are used as distributed feature vectors for words. However, they lack the ability to model affect in natural language. Therefore, such systems have difficulty providing a human-like experience to users. I address these issues in open-domain dialogue generation through several contributions.

1.2.1 Contributions

In Chapter 4, I devise four affective techniques for feature augmentation, training regularization, and inference in neural conversational models.

1. I augment standard word embeddings with three dimensional *affective* word embeddings, retrieved from a dictionary of word-level affective ratings. In this way, the ensuing neural model is aware of words' emotional features.

2. The cross-entropy loss function is commonly used to train neural dialogue models. I augment this loss with affective objectives, which serve the purpose of regularization and teach the model to generate more emotional utterances.

3. To further combat the problem of dullness in responses, I design *affectively diverse* beam search algorithms. They enable the model to actively search for affective responses during decoding.

4. I integrate ACT in the dialogue generation pipeline, whereby responses are conditioned on the affective predictions of ACT. This is achieved by using a combination of pretrained neural models (to embed text into the affective latent space) and encoder-decoder models (to map the conversational history and ACT prediction to a textual response).

In Chapter 5, I study how to implicitly infuse human-like affect into conversational agents, without relying on explicit affect models or heuristics. I propose to train standard encoder-decoder models using online active learning.

1. I use online deep active learning as a form of reinforcement in a novel way, which eliminates the need for hand-crafted reward criteria. I use a diversity-promoting decoding heuristic to facilitate this process.

2. I demonstrate how my model can be tuned for one-shot learning. It also eliminates the need to explicitly incorporate coherence, relevance or interestingness in the responses.

1.3 Domain Adaptation in Text Classification and Generation

One of the cornerstones of human intelligence is the ability to consume knowledge about multiple environments, and very effectively use it for decision-making in an unseen environment. Furthermore, as humans keep consuming more information in life, old knowledge does not simply get forgotten easily. In contrast, most state-of-the-art AI systems today are built for very specific tasks, and do not generalize well to other tasks where little training data is available. In fact, when trained on multiple tasks sequentially one after another, these models quickly forget the old knowledge and overfit to new knowledge. To combat this problem of *catastrophic forgetting* [94], transfer learning (sometimes referred to as domain adaptation[2]) is used. It teaches machines how to adapt to new tasks or domains without necessarily forgetting the knowledge gained from older tasks/domains.

In Chapter 6, I address the problem of *incremental domain adaptation* (IDA) in text classification and generation models. In IDA, we assume that different domains come sequentially one after another. We only have access to the data in the current domain, but hope to build a unified model that performs well on all the domains encountered so far.

1.3.1 Contributions

1. I tackle the IDA problem by proposing a new neural architecture: recurrent neural networks augmented with memory, called *progressive memory banks*. This memory is a set of distributed, real-valued vectors capturing domain knowledge. Contents of this memory are retrieved through the attention mechanism during training and inference.

[2]The terms 'transfer learning' and 'domain adaptation' are often used interchangeably in the literature [133]. In this work, I do not distinguish between these two concepts. In this thesis, a *domain* is defined by a dataset.

2. I provide theoretical analysis that is indicative of the superiority of my approach for IDA, compared to existing approaches.

3. I show promising experimental results on two tasks: natural language inference (a text classification task), and dialogue response generation.

1.4 Organization

This thesis is organized as follows.

- In Chapter 2, I introduce basic concepts of planning in AI, such as partially observable Markov decision process (POMDP) and Monte-Carlo Tree Search. I also provide background on state-of-the-art deep learning techniques for text-based dialogue generation, such as the encoder-decoder framework and word embeddings.

- In Chapter 3, I investigate decision-theoretic planning in *BayesAct*, a POMDP model of affective interactions between a human and an artificial agent. The approach is evaluated on two social dilemma games (Prisoner's Dilemma, Battle of the Sexes), a healthcare assistive device and robot navigation.

- In Chapter 4, I propose various methods for affective dialogue response generation, including 1) affective word embeddings, 2) affective loss functions, 3) affectively diverse beam search, and 4) conditional response generation using Affect Control Theory.

- In Chapter 5, I take a step back from explicitly modelling affect in dialogue systems. Instead, I propose online active learning for human-like dialogue generation.

- In Chapter 6, I address the problem of incremental domain adaptation (IDA). I propose neural progressive memory to alleviate the catastrophic forgetting problem in language inference and dialogue generation.

- In Chapter 7, I summarize my conclusions and discuss directions for future work.

Chapter 2

Background

2.1 Markov Decision Processes

A Markov decision process (MDP) [21] is a stochastic model of control. It consists of a set $\mathcal{S}$ of states; a set $\mathcal{A}$ of actions; a stochastic transition model $\Pr : S \times A \to \Delta(S)$, with $\Pr(s'|s,a)$ denoting the probability of moving from state s to s' when action a is taken, and $\Delta(S)$ is a distribution over $\mathcal{S}$; and a reward assigning $R(a,s')$ to a transition to s' induced by action a.

Intuitively speaking, at each time step, the environment is in state $s \in S$. The agent takes an action $a \in A$, which causes the environment to transition to a new s' with probability $\Pr(s'|s,a)$. This transition results in a reward $r = R(a,s')$ for the agent. The processes continues until a time horizon H is reached; H may be infinite. The goal of the agent is to choose actions that maximize his/her expected future reward $\mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t\right]$, where r_t denotes the reward at time step t and $0 \leq \gamma \leq 1$ is the *discount factor*. A discount factor of less than 1 ensures that distant rewards contribute less than immediate rewards, otherwise the sum of rewards over an infinite trajectory may become unbounded.

2.2 Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) [1] is a generalization of an MDP, where the state is not directly and fully observable. Instead, the agent receives an observation $\Pr(\omega_s|s)$, denoting the probability of making observation $\omega_s \in \boldsymbol{\Omega_s}$ while the

system is in state s; $\mathbf{\Omega_s}$ is an observation set. A generic POMDP is shown as a decision network in Figure 2.1.

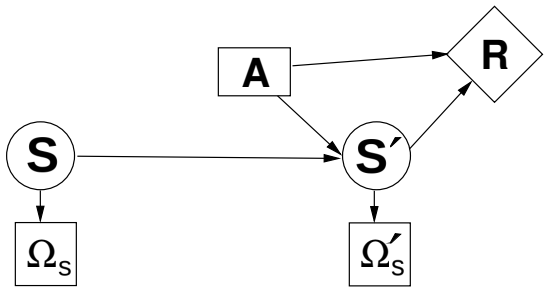


Figure 2.1: Two time slices of a general POMDP. Rectangles show observed variables, ovals show unobserved/hidden states, and the diamond node represents reward.

A *history* is sequence of actions and observations. A *belief state* $\mathcal{B}$ is a probability distribution over $\mathcal{S}$, given a history h. A *policy* maps a belief state to a probability distribution over actions, such that the expected discounted sum of rewards is (approximately) maximised. The *value function* $V^{\pi}(h)$ is the expected return from history h under a policy π. The *optimal value function* $V^{*}(h)$ is the maximum value function achievable by any policy.

In *factored* POMDPs, the state is represented by the cross-product of a set of variables or features. Assignment of a value to each variable thus constitutes a state. Factored models allow for conditional independence to be explicitly stated in the model.

POMDPs have been extensively studied in operations research [120], and in artificial intelligence [24, 88]. They have applications in many human-interactive domains, including intelligent tutoring systems [59], assistive technologies [79], and spoken dialogue systems [205, 206].

2.3 Planning

Planning in AI is the task of finding a sequence of actions to reach some predefined goals, while optimizing a given performance measure. A basic classical planning problem consists of a single artificial agent, a fully observable and deterministic environment with a unique and known initial state, and a set of deterministic actions which can be taken one at

a time. Since the environment is deterministic, the effect of any sequence of actions is deterministic. Some prominent examples of planning are found in robot navigation [28], healthcare [189], cyber security [23] and manufacturing [99].

2.3.1 Planning in POMDPs

MDPs generalize the classical view of planning and provide a more complex, stochastic framework for state transitions. The states are still fully observable, thus there is no incomplete information. However, the actions are non-deterministic, thus there is uncertainty about their effect. Therefore, instead of simply producing a sequence of actions, MDPs produce more general solutions, called policies. That is to say, action sequences (produced by classical planning) rarely execute as expected, therefore MDPs produce mappings from situations to actions that specify the agents behavior no matter what happens. Furthermore, the reward function is more general and can be state dependent.

POMDPs introduce further uncertainty into a planning problem by allowing the states to be partially observable. Computing an optimal policy (or an optimal value function) in a POMDP is intractable due to the *curse of dimensionality* [88]: the state space (and hence the belief space) is exponential. Finite-horizon POMDPs are known to be PSPACE-complete, while infinite-horizon POMDPs are undecidable [159].

Value iteration [181] is a well-known method to compute the optimal value function for POMDPs. Since it does not scale well, many *offline* variants have been proposed [146, 150, 183]. The offline algorithms approximate, prior to execution, the best action to execute for all situations. They perform well but often take significant time to solve problems with very large state spaces. Moreover, the policy needs to be recomputed from scratch every time the environment dynamics change. In general, value iteration and many of its offline variants suffer from the *curse of history* [30]: the number of distinct action-observation histories that must be grown and evaluated are exponential in the planning horizon.

Online solvers are a more viable alternative for large POMDPs. They use forward search only from the current state, and approximate the optimal value function by limiting the number of reachable beliefs explored in the tree. Ross *et al.* have surveyed the different online techniques for POMDP planning [159]. Among these, I focus on Monte Carlo methods.

2.3.2 Monte-Carlo Methods

Monte Carlo (MC) methods are a subclass of computational algorithms that use repeated random sampling for numerical estimations. MC estimates are typically used in situations where exact computations are intractable. A simple example is to compute the expectation of an arbitrary function $f(\boldsymbol{x})$ where $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Computing the expectation $\mathbb{E}[f(\boldsymbol{x})]$ exactly may be intractable due to the nature of f, but it can be approximated using Monte Carlo sampling. We independently and identically sample $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ where n is some integer. We compute $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}_i)$. Then $\hat{\mu}_n$ is an MC estimator for $\mathbb{E}[f(\boldsymbol{x})]$ and $\hat{\mu}_n \to \mathbb{E}[f(\boldsymbol{x})]$ as $n \to \infty$. Intuitively speaking, the more random samples we draw, the more closely we can approximate the target expectation.

MC methods are a popular choice for online planning in POMDPs [178], because complexity depends on the underlying difficulty of the POMDP rather than the size of the state space. In particular, MC simulation assumes that a POMDP simulator is provided. Given a state and an action, the simulator provides a sample of a successor state, observation and reward. In this way, many random trajectories can be explored, and their mean results can be used to estimate the values of states.

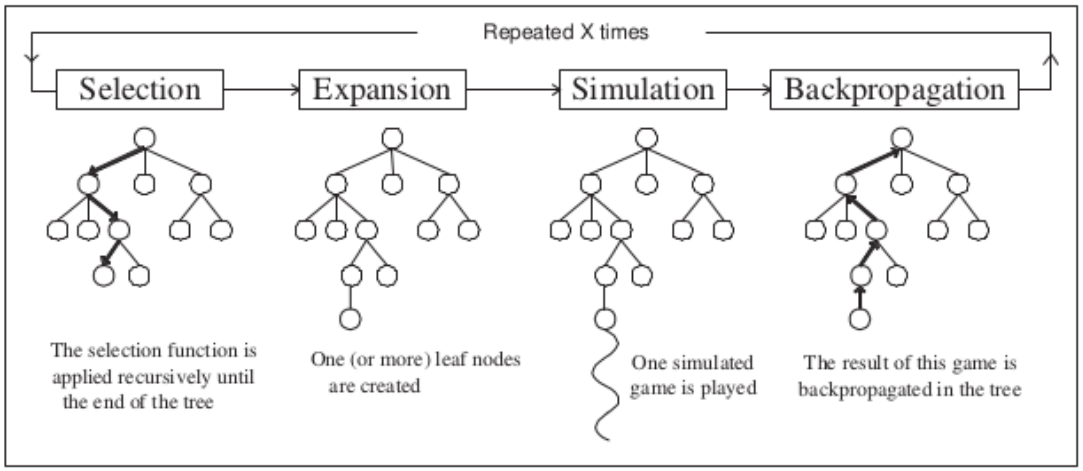


Figure 2.2: Monte Carlo Tree Search. Image source: http://tinyurl.com/zaobca8.

2.3.3 Monte-Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a planning algorithm that uses MC simulations to approximate the value of nodes of a search tree. I first explain the concept of tree search below, and then describe MCTS in detail.

Several real world problems can be formulated as search problems, where the search space can be represented as a tree. A canonical example is the game of Chess, where two players take turns to move game pieces over an 8×8 board. Each player's goal is to play a sequence of moves that leads to a win. This goal can be formulated as a tree search problem as follows. The state of the game at any time step is given by the board configuration, and is represented by a node in the game tree. The root node represents the initial state of the game, where all the pieces are unmoved. For each node, many next actions (game moves) are possible, each represented by a branch in the tree. Transitioning from one node to another corresponds to a move in the game. To win the game, certain desired board configurations need to be reached, which translates to finding the optimal path from the root to a desired node. This in turn implies that a series of most promising moves need to be made in succession, in order to win. At each turn, the future is simulated by expanding the tree as much as possible until the game ends. Then, the action leading to the best possible trajectory is chosen as the next move. Since the search tree grows exponentially and the branching factor for many problems is very high, the problem is intractable.

Monte Carlo Tree Search (MCTS) tries to circumvent the intractability of such problems by finding approximately best moves. It randomly simulates the game many times and records statistics about how promising different nodes and actions are; then it predicts the most promising move based on the gathered statistics.

Concretely, to predict the next most promising move, MCTS goes through four steps: *selection, expansion, simulation* and *backpropagation* (see Figure 2.2).

- In **Selection**, we start from the root node and successively select actions based on a *tree policy* (typically the best action based on the statistics gathered so far), until a leaf node L is reached. A leaf node in a search tree is one that never got expanded/explored in previous MCTS rounds.
- In **Expansion**, we expand node L to add one or more of its child nodes to the tree. Then we choose one of the children C.
- In **Simulation**, we follow a *rollout policy* i.e., select actions uniformly at random, to expand the trajectory till a terminal node (i.e., state representing a win, loss or draw) is reached.

- In **Backpropagation**, we update statistics (e.g. number of times the node is visited, number of wins resulting from trajectories going through this node, etc.) the nodes in the path from root to C.

I now describe the statistics gathered by nodes during MCTS for action selection. Typically, the node corresponding to state s stores a value $Q(s, a)$ and a visitation count $N(s, a)$ for each action a, both initialized to zero. The value $Q(s, a)$ is the mean return of all simulations in which action a was selected in state s. Typically, the tree policy in the selection phase of MCTS is simply the greedy approach, which selects the action with the highest value. However, this results in very little exploration of other actions and there is a danger that the algorithm settles for a less optimal trajectory. The UCB1 algorithm [14] mitigates this issue by adding an exploration bonus to the value computation: $\hat{Q}(s, a) = Q(s, a) + c\sqrt{\frac{\log \sum_a N(s,a)}{N(s,a)}}$, where the weight c balances exploration versus greed, and is a tunable hyperparameter.

2.4 Deep Learning for Natural Language Processing

Natural Language Processing (NLP) is a branch of AI that enables machines to understand, analyse and generate human languages. Language is an essential part of human communication, thus any machine that is touted to be *intelligent like humans* should be able to communicate with humans, like humans. NLP tasks include (but are not limited to) part-of-speech tagging, word segmentation, lemmatization, named entity recognition, intent classification, translation, question answering, document summarization and language generation [77].

NLP techniques have been around for several decades, dating back to 1950s. Most of the early systems relied on manually-engineered rule-based systems, such as specifying all the rules of grammar in order to adhere to them. The popularization of machine learning in the 80s resulted in the development of statistical methods that could learn such rules automatically from data [87]. However, feature extraction was still a predominantly manual process and posed as an impediment to building fully automated NLP pipelines. This issue has been addressed remarkably well with the recent advent of deep learning, so much so that the NLP landscape has completely transformed [218].

In the rest of this section, I describe the various building blocks of deep learning for NLP used in this thesis.

2.4.1 Feed-Forward Neural Networks

A neural network is a machine learning model that can approximate any arbitrary unknown function $\boldsymbol{y} = q(\boldsymbol{x})$. A canonical example of neural network is the feed-forward neural network, also called multilayer perceptron (MLP). An MLP is a directed acyclic graph of computational units called *neurons*, which are arranged in layers. Each neuron in a layer is connected to all the neurons in the next layer via forward edges that have real-numbered weights associated with them. Each neuron produces a real-valued output, called an *activation*. Thus, given the vector $\boldsymbol{a}_i$ of activations of all the neurons in the previous layer i, along with the weight matrix $\boldsymbol{W}_i$ of these activations, the activations of the $i+1$'th layer are computed by

$$\boldsymbol{a}_{i+1} = f(\boldsymbol{W}_i \boldsymbol{a}_i + \boldsymbol{b}_i) \tag{2.1}$$

where $\boldsymbol{b}_i$ is the *bias* vector and f is a non-linear function e.g. hyperbolic tangent (tanh), sigmoid or rectified linear unit. The activation of the last layer is the final output of the network. The weights and the biases are the parameters of the model, collectively denoted by $\boldsymbol{\theta}$.

The goal of the MLP is to learn a parameterization of the function $\boldsymbol{y} = q(\boldsymbol{x})$ from a corpus of N training samples $(\boldsymbol{x}_j, \boldsymbol{y}_j)$ for $j \in \{1, \cdots, N\}$. To achieve this, the network's parameters are initialized randomly. Then, the model processes the training samples (one by one in the simplest case) and adjusts its parameters to minimize its error. Concretely, given each input $\boldsymbol{x}_i$, the network computes an approximation $\hat{\boldsymbol{y}}_i$. A *loss function* $L(\boldsymbol{\theta})$ uses these predictions $\hat{\boldsymbol{y}}_i$ and the true labels $\boldsymbol{y}_i$ to compute the model's error. Some popular loss functions are mean squared-error (MSE) and *negative log likelihood*. The *backpropagation* algorithm then computes the partial derivative of the loss with respect to each parameter. It uses the multivariate chain rule to compute the gradients for layer i conditioned on the gradients of layer $i + 1$. Finally, a Gradient Descent algorithm uses these derivatives to adjust each parameter such that the loss is minimized:

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \eta \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \tag{2.2}$$

Here, η is a hyperparameter called the *learning rate*; it controls the amount of adjustment made to a parameter. This training process is repeated until the parameters converge, or until a 'good' approximation is achieved.

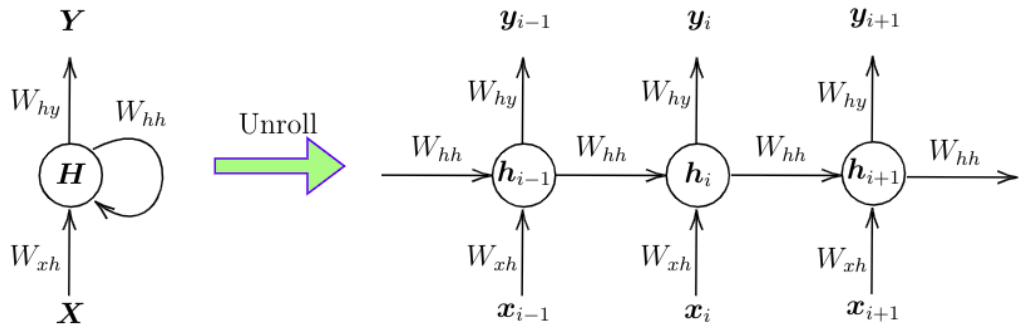


Figure 2.3: Left: a vanilla recurrent neural network. Right: unfolding the forward computation in time.

2.4.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a special type of MLP that is designed to handle sequential data. Thus, RNNs are a popular choice for NLP tasks because the training data consists of sequences of characters or words (e.g. sentences or documents). Figure 2.3 (left) shows an RNN that takes a sequence $\boldsymbol{X}$ as input, builds a sequence $\boldsymbol{H}$ of hidden states, and produces an output sequence $\boldsymbol{Y}$. W_{hh}, W_{xh} and W_{hy} are the weight matrices. In Figure 2.3 (right), I show the RNN unrolled through time. That is to say, I draw the complete network showing each time step, which gives us an MLP. Here, $\boldsymbol{x}_i$ represents the i'th element/word of the input sequence, and is a one-hot vector in the most simple setting. Vector $\boldsymbol{h}_i$ is the hidden state of the RNN at step i and is often called the memory of the RNN, because it contains knowledge of the previous steps. At step i, the hidden state and output are computed by

$$\boldsymbol{h}_i = f(W_{xh}\boldsymbol{x}_i + W_{hh}\boldsymbol{h}_{i-1}) \tag{2.3}$$

$$\boldsymbol{y}_i = g(W_{hy}\boldsymbol{h}_i) \tag{2.4}$$

where $\boldsymbol{x}_i \in \mathbb{R}^{d_{in}}, \boldsymbol{h}_i \in \mathbb{R}^{d_{hdn}}$ and $\boldsymbol{y}_i \in \mathbb{R}^{d_{out}}$ for some constant values d_{in}, d_{hdn} and d_{out}. W_{hh}, W_{xh} and W_{hy} are trainable parameters which, unlike MLPs, are shared across all the steps. This makes intuitive sense: at each step, the same task is being performed, but with a different input. Another benefit of weight sharing is that the total number of parameters is drastically reduced, making the model more efficient. In most NLP classification models, the activation function f is tanh and g is the softmax function, given by

$$softmax(\boldsymbol{y}_i) = \frac{\exp(\boldsymbol{y}_i)}{\sum_{j=1}^{d_{out}} \exp(\boldsymbol{y}_j)} \tag{2.5}$$

Softmax normalizes the vector $\boldsymbol{y}_i$ such that each vector component ranges between 0 and 1, and the sum of the components is 1. Thus, real-valued vectors are converted into probability vectors, which is convenient for classification.

RNNs are trained using *backpropagation through time* (BPTT), where the network is first unrolled backwards through time and then trained via the standard backpropagation algorithm.

2.4.3 Long Short-Term Memory Networks

Vanilla RNNs, as described above, have a major limitation. Ideally, the hidden state $\boldsymbol{h}_i$ should preserve information about the entire input sequence up to step i. However, this is difficult to achieve in practice if the sequence is long. This is known as the long-term dependency problem in RNNs. This is partially[1] a side-effect of BPTT's vanishing or exploding gradient problem for long sequences. As the network is unrolled backwards through time, the gradients become too small (approaching zero) or too large (approaching infinity). This results in weight updates that are either negligible or unstable. While exploding gradients can be dealt with through *gradient clipping*, vanishing gradients are harder to resolve. Long Short-Term Memory (LSTM) networks address the problem by changing the way the hidden state is computed [78].

Concretely, an LSTM network maintains a *cell state* $\boldsymbol{c}_i$ in addition to computing the hidden state $\boldsymbol{h}_i$ at each step. The full mathematical formulation for an LSTM update is given by

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_f) \tag{2.6}$$
$$\boldsymbol{d}_i = \sigma(\boldsymbol{W}_d \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_d) \tag{2.7}$$
$$\tilde{\boldsymbol{c}}_i = \tanh(\boldsymbol{W}_c \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_c) \tag{2.8}$$
$$\boldsymbol{c}_i = \boldsymbol{f}_i \circ \boldsymbol{c}_{i-1} + \boldsymbol{d}_i \circ \tilde{\boldsymbol{c}}_i \tag{2.9}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_o) \tag{2.10}$$
$$\boldsymbol{h}_i = \boldsymbol{o}_i \circ \tanh(\boldsymbol{c}_i) \tag{2.11}$$

where $\circ$ denotes pointwise multiplication of vectors. Intuitively, there are three *gates* that regulate the addition or removal of information from the cell state. The *forget gate* decides which information to remove from the cell state (Eq. 2.6). The *input gate* decides how

[1]The long-term dependency problem is also due to the inherently sequential nature of RNNs. If a sequence is long, too much information accumulates over time, making it hard to determine which step in the preceding sub-sequence is most relevant at the current time-step.

much new information to add to the cell state (Eq. 2.7), and also creates candidate values to be added (Eq. 2.8). The *output gate* is responsible for producing a filtered version of the cell state as the hidden output (Eq. 2.11).

This formulation helps LSTM networks mitigate the gradient vanishing problem. While performing BPTT in an LSTM network, the gradient computation involves multiplication with the activation of the forget gate (Eq. 2.9). If this activation is close to 1, the gradients do not grow too small.

Bi-directional LSTM Networks

LSTM networks (and RNNs in general) parse input in one specified direction (e.g. English text is parsed left to right). Thus, at a given time step i, the network possesses information about $i-1$ inputs from the past. However, we would ideally like the network to capture information about both past and future at any given step. To achieve this, we add another LSTM network that looks at the data in the backward direction. Then combining the hidden states of these two LSTM networks allows us to model the past and future at all time steps. This model is called a bi-directional LSTM network, BiLSTM for short.

BiLSTM networks outperform unidirectional LSTMs in several applications, due to their superior ability to model context [36, 69].

2.4.4 Gated Recurrent Units

A Gated Recurrent Unit (GRU) [37] network is similar to an LSTM network, in that it tries to mitigate the vanishing gradient problem of vanilla RNNs. However, GRUs do not maintain an internal cell state, and use two instead of three gates to regulate information storage. Intuitively, an *update gate* $\boldsymbol{z}_i$ determines how much information from the past (i.e. previous time steps) should be passed along to the future. Similarly, the *reset* $\boldsymbol{r}_i$ gate determines how much of this information should be forgotten. The gate computations are

$$\boldsymbol{z}_i = \sigma(\boldsymbol{W}_z \boldsymbol{x}_i + \boldsymbol{U}_z \boldsymbol{h}_{i-1}) \tag{2.12}$$

$$\boldsymbol{r}_i = \sigma(\boldsymbol{W}_r \boldsymbol{x}_i + \boldsymbol{U}_r \boldsymbol{h}_{i-1}) \tag{2.13}$$

$$\tilde{\boldsymbol{h}}_i = \tanh(\boldsymbol{W}_h \boldsymbol{x}_i + \boldsymbol{r}_i \circ \boldsymbol{U}_h \boldsymbol{h}_{i-1}) \tag{2.14}$$

$$\boldsymbol{h}_i = \boldsymbol{z}_i \circ \boldsymbol{h}_{i-1} + (1 - \boldsymbol{z}_i) \circ \tilde{\boldsymbol{h}}_i \tag{2.15}$$

Compared to LSTMs, GRUs have less control over information regulation, since they don't maintain an internal cell state. However, they are more computationally efficient due

to a less complex structure [38]. Similar to BiLSTMs, **Bi-directional GRUs (BiGRUs)** are often used to capture both the past and present at any given time step.

2.4.5 Word Embeddings

Traditional non-neural NLP models relied on hand-engineered features such as n-grams, word co-occurrence statistics or one-hot word representations. These features are often expensive to compute, and do not capture word semantics well. To address this issue, Bengio *et al.* [22] proposed to learn distributed representations for words using neural networks. That is to say, each word is embedded in a fixed dimensional real-valued vector space, and these embeddings are learned from the data during end-to-end training. This allows the neural network to automatically capture important semantic and syntactic relationships between words, and map words into this feature space such that words with similar meanings have similar feature vectors. Typically, neural word embeddings learned end-to-end using small to medium sized datasets may not generalize well, due to overfitting. It makes more sense to learn the word embeddings independently using large-scale corpora, and use these pretrained representations in downstream NLP tasks. Two popular pretrained word embedding models are Word2Vec [131] and GloVe [142]. Word2Vec has two variants: 1) The Continuous Bag of Words (CBOW) model predicts a word based on the surrounding words (called context). 2) The Skip Gram model predicts the context words for a given word. Both these models capture text semantics well by learning local co-occurrence patterns of words. However, they do not take global context into consideration. GloVe mitigates that by training on global co-occurrence statistics of words.

Word2Vec and GloVe produce a single, fixed vector for a given word. However, words may have different meanings depending on the context. Thus, contextualized word embeddings have been proposed. The model is called ELMo [144] and consists of a BiLSTM network trained on the 'language modelling' task: given a sequence of words, predict the next suitable word. At prediction time, the model accepts an input sentence that contains the target word whose embedding is required. Then it combines the forward and backward LSTM states of that word to produce the final word embedding. BERT [49] is another neural NLP model that produces contextualized word embeddings. BERT is based on a recently popularized state-of-the-art neural architecture called Transformer [194], which uses the concept of 'self-attention' to overcome the long-term dependency issue. Pre-trained ELMo and BERT models are publicly available.

All these models have a notable limitation. Syntactic context and co-occurrence statistics are insufficient to capture sentiment/emotional features, because words different in

sentiment often share context (e.g., "a *good* book" vs. "a *bad* book"). To overcome this problem, some recent works propose to enrich the word embeddings using sentiment and/or emotion labels [2, 100, 156, 187].

2.4.6 Sequence-to-Sequence Framework

A sequence-to-sequence (Seq2Seq) model maps a variable length input sequence to a variable length output sequence [185]. It consists of an *encoder* and a *decoder*, both of which are RNNs (usually LSTMs or GRUs). The encoder network sequentially accepts the embedding of each word in the input sequence, and encodes the input sentence as a vector of fixed length (the last hidden state of the encoder is typically taken to be the encoding). This encoded vector is called the *context* vector, and becomes the first hidden state of the decoder. The decoder input at the first step is a fixed and predefined token called start-of-sequence ("<sos>"). At each step, the decoder produces an output probability distribution over the vocabulary. The token with the highest probability is taken to be the input to the decoder at the next step. Thus, the decoder sequentially generates an output sequence and the process stops if the end-of-sequence token ("<eos>") is generated, or if the maximum sequence length is reached.

Given a message-response pair $(\boldsymbol{X}, \boldsymbol{Y})$, where $\boldsymbol{X} = \boldsymbol{x}_1, \cdots, \boldsymbol{x}_m$ and $\boldsymbol{Y} = \boldsymbol{y}_1, \cdots, \boldsymbol{y}_n$ are sequences of words, Seq2Seq models (parametrized by $\boldsymbol{\theta}$) are typically trained to minimize the negative log likelihood of the data, also called the cross entropy loss (XENT):

$$L_{\texttt{XENT}}(\boldsymbol{\theta}) = -\log p(\boldsymbol{Y}|\boldsymbol{X}) = -\sum_{i=1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{X}) \tag{2.16}$$

Seq2Seq is one of the most popular and state-of-the-art neural models for machine translation [15], dialogue generation [196], speech recognition [35] and image captioning [214].

2.4.7 Attention Mechanism

In the vanilla Seq2Seq model described above, the entire input sequence is encoded into a single, fixed length context vector. This representation is not ideal for long inputs, and also loses important information about the position of certain tokens within the input. While decoding at a particular time step, it is desirable that the decoder should pay more attention to certain words or phrases within the input. To achieve this, Bahdanau *et al.* [15] introduced the concept of 'attention' where the decoder, in addition to the previous

hidden state and the input, looks at all the hidden states of the encoder and decides which of them are useful at the current step.

Concretely, let $\boldsymbol{s}_j, 1 \leq j \leq M$ be the hidden states of the encoder, where M is the length of the input sequence $\boldsymbol{X}$. We first compute the similarity between the previous decoder hidden state $\boldsymbol{h}_{i-1}$ and all the encoder hidden states $\boldsymbol{s}_j$. These similarity scores are called the attention energies e_j:

$$e_j = s(\boldsymbol{h}_{i-1}, \boldsymbol{s}_j) \tag{2.17}$$

where s is a linear transformation whose parameters are learned end-to-end during training. The attention energies are normalized to get the attention weights a_j:

$$a_j = \frac{e_j}{\sum_{k=1}^{M} e^k} \tag{2.18}$$

A context vector $\boldsymbol{c}_i$ is constructed by taking a weighted combination of the encoder hidden states $\boldsymbol{s}_j$:

$$\boldsymbol{c}_i = \sum_{j=1}^{M} a_j \boldsymbol{s}_j \tag{2.19}$$

This context vector is concatenated with the input of the decoder at each time step.

2.4.8 Variational Autoencoders for Text Generation

An autoencoder consists of two neural networks, an encoder and a decoder. The encoder takes an input sequence $\boldsymbol{X}$ and compresses it to a lower-dimensional dense representation, which the decoder tries to convert back to the original input. The two networks are trained end-to-end via a loss function that typically comprises reconstruction error. Thus, the encoder must learn to discard irrelevant parts of the input, and preserve just enough information in the dense representation for the decoder reconstruction. A commonly used reconstruction loss is negative log likelihood:

$$L_{\texttt{AE}}(\boldsymbol{\theta}) = -\sum_{\boldsymbol{X} \in \mathcal{X}} \log p(\boldsymbol{X}|\boldsymbol{\theta}) \tag{2.20}$$

where $\mathcal{X}$ is the training set.

Autoencoders are useful for compressing data into a lower-dimensional space. However, they are not good generative models for text, because the latent space (where the encoded vectors lie) may not be continuous. That is to say, there may be clusters of encodings

in the latent space and discontinuities (empty regions) between them. Sampling from the discontinuous regions leads to decoder outputs that are unrealistic.

Variational Autoencoders (VAEs) [93, 157] circumvent this issue by imposing constraints on the encodings such that the latent space is continuous; thus random samples from this space can be used to generate realistic decoder outputs. Concretely, given an input sequence $\boldsymbol{X}$, the encoder produces a distribution q_E over the latent space in the form of a vector of means $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\lambda}$. Then, a latent vector $\boldsymbol{z}$ is sampled from q_E and passed through the decoder. Intuitively, the entries of $\boldsymbol{\mu}$ correspond to the centres of clusters in the latent space, and the $\boldsymbol{\lambda}$ entries are the standard deviations of each cluster. To impose continuity in the latent space, the clusters should be close to each other while still being distinct. This is achieved by forcing the latent space to be 'packed' within the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. We call this the prior distribution of $\boldsymbol{z}$, or $p(\boldsymbol{z})$. Overall, the VAE loss function is

$$L_{\texttt{VAE}}(\boldsymbol{\theta}; \boldsymbol{X}) = \mathrm{KL}\big(q_E(\boldsymbol{z}|\boldsymbol{X}) \big\| p(\boldsymbol{z})\big) - \mathbb{E}_{q_E(\boldsymbol{z}|\boldsymbol{X})}\left[\log q_D(\boldsymbol{X}|\boldsymbol{z})\right] \tag{2.21}$$

where the second term is the reconstruction loss and q_D is the probability distribution given by the decoder. The first term measures the difference between the probability distributions q_E and $p(\boldsymbol{z})$ using Kullback-Leibler (KL) divergence [98]. KL divergence between two probability distributions p_1, p_2 is given by

$$\mathrm{KL}(p_1, p_2) = \sum_{\boldsymbol{X} \in \mathcal{X}} p_1(\boldsymbol{X}) \log \left(\frac{p_2(\boldsymbol{X})}{p_1(\boldsymbol{X})} \right) \tag{2.22}$$

After being trained in this fashion, the decoder of the VAE can be treated as a text generator: a random sample from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ can be propagated through it to produce a fluent and realistic sentence.

VAEs, as described above, cannot be trained end-to-end with backpropagation, because the computational graph contains a sampling operation which does not have a gradient. To get around this issue, a reparameterization trick is used [93], which pushes the non-differentiable operation out of the computational graph. More concretely, we use

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\lambda} \cdot \boldsymbol{\epsilon} \tag{2.23}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ are learnable parameters and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is fixed. Thus, the gradients can flow from the decoder through $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ to the encoder.

VAEs have been successfully used as generative models of realistic text [25] and images [83].

2.4.9 Conditional Variational Autoencoders

While VAEs are useful for data generation, they do not provide a way to control the generated output. For instance, we can train a VAE on a large corpus of sentences such that, at inference time, a latent sample can be decoded into a plausible sentence. This VAE is a useful language generator, but it is not a good dialogue generator because it does not allow the generation to be conditioned on the conversation history. This problem is remedied by Conditional Variational Autoencoders (CVAEs) [180].

CVAE extends VAE by conditioning the latent prior $p(\boldsymbol{z})$, the encoder $q_E(\boldsymbol{z}|\boldsymbol{X})$ and the decoder $q_D(\boldsymbol{X}|\boldsymbol{z})$ on a *context vector* c. The new prior $p(\boldsymbol{z}|\boldsymbol{c})$ is parameterized by a separate neural network, and $p(\boldsymbol{z}|\boldsymbol{c}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\lambda}^2\boldsymbol{I})$. For the encoder, $q_E(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{c}) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\lambda}}^2\boldsymbol{I})$. The CVAE loss function is given by

$$L_{\texttt{CVAE}}\big(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{c}\big) = \mathrm{KL}\big(q_E(\boldsymbol{z}|\boldsymbol{X},\boldsymbol{c})\big\|p(\boldsymbol{z}|\boldsymbol{c})\big) - \mathbb{E}_{q_E(\boldsymbol{z}|\boldsymbol{X},\boldsymbol{c})}\left[\log q_D(\boldsymbol{X}|\boldsymbol{z},\boldsymbol{c})\right] \tag{2.24}$$

This equation is similar to the VAE loss given in Equation 2.21, except that all the probability distributions are conditioned with the variable c.

2.5 A Brief History of Dialogue Systems

Part of this thesis is dedicated to building conversational agents that are open-domain (i.e., are not restricted to particular domains/topics and can have generic chit-chat), natural-sounding like humans, and are fluent and engaging. To this end, this section provides an overview of existing open-domain dialogue systems.

Open-domain dialogue generation is an established scientific problem in academia [196, 168]. Additionally, it is very relevant in the industry: Microsoft's XiaoIce and Baidu's DuMi are prime examples. Even in a task-oriented setting where the conversation has a specific and pre-defined goal (e.g. book a flight or order food), open-domain conversational AI is often used to handle chit-chat and other unforeseen user queries.

The trend of building dialogue systems started after Alan Turing introduced the Turing test in 1950, a criterion to assess machine intelligence through conversational interaction [192]. Soon after, several rule-based dialogue systems were developed. Given an input sentence, these systems used pre-defined hand-engineered rules (if-else conditions, regular-expression matching) to map each input to a pre-existing response. Examples of such systems include ELIZA [199] and PARRY [39].

Purely rule-based systems are very restrictive and hard to maintain, and do not generalize well to unseen queries. Retrieval-based systems have gained more popularity [18, 104, 208, 55], especially in the industry [215, 230]. Given an input query, these systems use information-retrieval algorithms to select a list of candidate responses from a pre-existing text database. These responses are often ranked by suitability, using learning-to-rank algorithms. Retrieval-based dialogue systems are scalable and efficient, but do not generate diverse responses.

With the advent of scalable deep learning using neural networks, it is now possible to build generative dialogue systems that create a response word-by-word, from scratch, given in an input query. They are 'end-to-end' trainable (all the components of the system are trained together using a dataset of message-response pairs), and we do not have to worry about training different components (e.g. intent detection, entity recognition, dialogue state tracking) separately. Moreover, they generalize well to unseen queries and can be easily finetuned to specific domains.

The state-of-the-art in generative open-domain dialogue is the encoder-decoder neural framework.

2.5.1 Encoder-Decoder Dialogue Models

The Seq2Seq encoder-decoder framework (Section 2.4.6) provides a natural way to do end-to-end dialogue generation. Given a corpus of message-response pairs, a vanilla Seq2Seq model learns by maximizing the likelihood of the response for a particular message. This idea was independently explored in two studies [196, 171], and achieves good results in terms of fluency and grammatical correctness of the generated responses. Sordoni *et al.* [182] propose a contextualized extension of the vanilla model, where they use more than one messages in a conversation as input. This model produces responses that are contextually more relevant to the input, in addition to being fluent and natural sounding. However, if the input messages are long, a single fixed-length context vector proves insufficient to preserve all the input information. To get around this problem, Serban *et al.* [170] propose a hierarchical Seq2Seq variant called HRED. First, an encoder produces a context vector for each input message. Then, another encoder takes these context vectors as input and produces a vector summarizing the entire input conversation. Finally, this summary vector is passed into the decoder to generate a response. HRED is effective in capturing dialogue context, and has been extended by various studies [169, 168, 211].

Several studies have explored Seq2Seq models trained using **deep reinforcement learning**. Li *et al.* [110] and Yu *et al.* [220] simulate dialogue between two virtual agents,

use hand-crafted reward functions to estimate the quality of each response, and thus learn policies that capture some pre-defined global characteristics of an engaging conversation. However, it is hard to manually define functions for each desirable quality of a conversation. Therefore, some studies propose to use online human feedback for policy learning [107, 108].

Lack of **diversity** in the generated responses is a critical issue in Seq2Seq-based models; we often see short, dull and generic responses such as *'Yes', 'No', 'Okay', 'I'm not sure'* and *'I don't know'* because they have a high frequency in most human-human conversational datasets. One remedy is to use Beam Search: rather than greedily choosing the top most probable token at each step of decoding, choose the top K tokens and thereby maintain a set of top K subsequences (called beams) at each step. While effective, beam search is prone to generating sequences that are *almost* identical, such as *'I don't know.'* and *'I don't know!'*. To counter this effect, variations of beam search have been proposed. Vijaykumar *et al.* [195] incorporate syntactic diversity between beams at each step by adding a dissimilarity term to the beam search optimization objective. Huang *et al.* [84] point out that the different beam search hypotheses may have different lengths during decoding, therefore it is important to decide when beam search should be stopped. Shao *et al.* [173] propose a new decoding method that selects K beam candidates at each step by sampling, rather than naively choosing the most probable ones. Some diversity-promoting approaches attempt to regularize the maximum likelihood objective directly. Li *et al.* [105] augment the maximum likelihood objective with a maximum mutual information (MMI) objective to capture the semantic and syntactic relationship between the input and output. Nakamura *et al.* [135] add an inverse-token-frequency term to the objective that penalizes the choice of common words during decoding. These techniques help mitigate the generic responses, but may lead to ungrammatical outputs.

The models described above diversify the output of the decoder only at the word-level. To control the generation of responses at the discourse level (e.g. by sentiment, topic, style, etc), **latent variable encoder-decoders** have been proposed. They introduced latent variables in the encoder-decoder framework, to learn distributions over higher-level conversational characteristics. By conditioning the decoder on these latent variables, we can introduce discourse-level variations in the decoded output. Several studies have explored this idea using VAEs and conditional VAEs (i.e. VAEs where the latent variables are further conditioned on the dialogue context) [168, 227, 175, 174, 140].

To make Seq2Seq responses more specific and meaningful, some **content-introducing methods** have been developed. Mou *et al.* [134] propose a model to produce responses that contain given keywords. Xing *et al.* [210] add topic-awareness to Seq2Seq by using pre-trained LDA topic models to guide the generation. Gu *et al.* [71] copy specific words or phrases from the input and appropriately place them in the decoded response.

Another facet of open-domain dialogue generation is **personalization**. To appear more human-like and natural sounding, a conversational agent should have a consistent personality and should consider the individual users' profiles. To this end, Zhang *et al.* [224] propose to finetune a pretrained and generic conversational model on personalized data. Li *et al.* [106] explicitly encode personas in distributed embeddings to capture background information and speaking style, and integrate them into the decoder. Several studies store profile information in memory (neural memory network or a simple dictionary), retrieve it via attention and condition the response on it during decoding [223, 152, 228]. Most of these studies use relatively small and synthetic datasets, which may cause overfitting. Very recently, Mazaré *et al.* [129] have built a new dataset of 5 million personas and 700 million persona-based dialogues. They show that end-to-end approaches work well when trained on this dataset.

2.5.2 Dialogue Evaluation Metrics

The goal of open-domain dialogue systems is to generate fluent, natural-sounding and engaging responses to queries. These qualities are hard to measure automatically, because they are subjective. Furthermore, each query may have several valid responses, therefore comparing a model's output to the true labels is not very meaningful. In light of these issues, most research studies provide a combination of human evaluation (where at least 3 human judges are asked to rate the responses) and automatic evaluation metrics described below.

BLEU-*n* [138] is a metric borrowed from the machine translation community. It measures the average precision of n-gram overlap between the generated response (referred to as *candidate*), and the ground truth (the *reference*). The precision is computed by

$$P_n = \frac{\text{Number of } n\text{-gram matches between candidate and reference}}{\text{Total number of } n\text{-grams in the candidate}} \tag{2.25}$$

For very short candidates, precision may be very high. To penalize such candidates, a brevity penalty ρ is used:

$$\rho = \exp(\min(0, \frac{L_c - L_r}{L_c})) \tag{2.26}$$

where L_c and L_r are the lengths of the candidate and reference respectively. The final BLEU-n score is the geometric mean of the precisions penalized by brevity:

$$\text{BLEU-}n = \rho \prod_{i=1}^{n} P_i^{\frac{1}{n}} \tag{2.27}$$

BLEU-2 is a commonly reported metric in recent studies [53, 126].

METEOR [19] is another machine translation metric. It does unigram matching between a candidate and a reference based on each unigram's exact form, stemmed form and meaning (synonimity). The final score is the harmonic mean of matching precision and matching recall. METEOR is an improvement over $BLEU$ because it goes beyond the exact token form and additionally considers recall. However, it is only based on unigrams and disregards n-grams.

ROUGE [114] is a set of metrics to evaluate text summarization. For a given candidate and reference, ROUGE-N computes their n-gram recall, whereas ROUGE-L computes the F-measure of their longest common subsequence.

Distinct-n [105] is used to measure the diversity between multiple generated responses. It is given by the number distinct n-grams in a response, scaled by the total number of n-grams in that response. Typically n is taken to be 1 or 2.

These metrics enable high-throughput evaluation, but they have been shown to have weak or no correlation with human judgements [117]. A more meaningful metric is **Average Embedding Similarity** [168, 225], which measures semantic similarity between two responses. It computes a real-valued vector for each response by taking the mean of the word embeddings (typically Word2Vec or GloVe vectors) in each response, and then computes the cosine similarity between them. However, due to the average operation, this metric is not very good at capturing sentence-level semantic similarity.

Automatic dialogue quality evaluation is an active area of research. Tao *et al.* [188] propose an unsupervised metric RUBER. It combines the embedding similarity metric with a score to measure relatedness of query and generated reply. Lowe *et al.* [121] use human annotated data to learn to predict the score of a response, given the query and ground truth reply. Bowman *et al.* [25] propose to learn a response evaluation metric through adversarial training, where a discriminator tries to differentiate between user-generated and machine-generated responses. All these techniques are effective, but have their own limitations and are fairly recent, therefore they have not been widely adopted and evaluated. Human evaluation, though time and labour intensive, remains the most popular metric.

Chapter 3

Affective Intelligence for Decision Making

3.1 Introduction

BayesAct [80, 81] is a partially-observable Markov decision process (POMDP) model of affective interactions between a human and an artificial agent. *BayesAct* is based upon a sociological theory called "Affect Control Theory" (ACT) [74], but generalises this theory by modeling affective states as probability distributions, and allowing decision-theoretic reasoning about affect. *BayesAct* posits that humans will strive to achieve consistency in shared affective cultural sentiments about events, and will seek to increase *alignment* (decrease *deflection*) with other agents (including artificial ones). Importantly, this need to align implicitly defines an affective heuristic (a *prescription*[1]) for making decisions quickly within interactions. Agents with sufficient resources can do further planning beyond this prescription, possibly allowing them to manipulate other agents to achieve individual profit in collaborative games.

BayesAct arises from the symbolic interactionist tradition in sociology and proposes that humans learn and maintain a set of *shared* cultural affective *sentiments* about people, objects, behaviours, and about the dynamics of interpersonal events. Humans use a simple affective mapping to appraise individuals, situations, and events as sentiments in a three dimensional vector space of evaluation (good vs. bad), potency (strong vs. weak)

[1] We prefer *prescription*, but also use *norm*, although the latter must not be mis-interpreted as logical rules (see Section 3.7).

and activity (active vs. inactive). These mappings can be measured, and the culturally shared consistency has repeatedly been demonstrated to be extremely robust in large cross-cultural studies [75, 136]. Many believe this consistency "gestalt" is a keystone of human intelligence. Humans use it to make predictions about what others will do, and to guide their own behaviour. The shared sentiments, and the resulting *affective ecosystem* of vector mappings, encodes a set of social prescriptions that, if followed by all members of a group, results in an equilibirium or *social order* [66] which is optimal for the group as a whole, rather than for individual members. Humans living at the equilibrium "feel" good and want to stay there. The evolutionary consequences of this individual need are beneficial for the species.

Nevertheless, humans are also a curious, crafty and devious bunch, and often use their cortical processing power to go beyond these prescriptions, finding individually beneficial strategies that are still culturally acceptable, but that are not perfectly normative. This delicate balance is maintained by evolution, as it is beneficial for the species to avoid foundering within a rigid set of rules. In this chapter, starting from the principles of *BayesAct*, I investigate how planning beyond cultural prescriptions can result in deceptive or manipulative strategies in two-player social dilemma games.

At its core, *BayesAct* has an 18-dimensional continuous state space that models affective identities and behaviours of both the agent and the person it is interacting with, and a 3-dimensional continuous affective action space. In the examples described in [81], a heuristic policy was used that resorted to the normative actions. Here, I tackle the open problem of how to use decision-theoretic planning to choose actions in *BayesAct*. I present a modification of a known Monte-Carlo tree search (MCTS) algorithm (POMCP) [178]. The proposed variant, called POMCP-C, handles continuous actions, states, and observations. It uses a dynamic technique to cluster observations into discrete sets during the tree building, and assumes a problem-dependent *action bias* as a probability distribution over the action space, from which it samples actions when building the search tree. Such *action biases* are natural elements of many domains, and I give an example from a traffic management domain where the *action bias* arises from intuitions about the accelerations of the vehicle (i.e. that it should not accelerate too much).

This chapter makes two contributions.

1. It describes how to use MCTS planning in *BayesAct*, and proposes the POMCP-C algorithm. It gives arguments for why this is an appropriate method. This idea was hinted at in [81].

2. It demonstrates POMCP-C on several applications. First, it shows the emergence of realistic and manipulative behaviours in two toy social dilemma games: *prisoner's*

dilemma and *battle of the sexes*. Second, it reviews experiments on a realistic, affectively aware health-care assistive device for persons with dementia. Third, it presents evidence that the proposed MCTS variant can be effectively used for planning in non-affective domains too, namely a robot navigation problem.

This chapter is organized as follows. First, I review ACT and *BayesAct*, and then present a new POMCP-C algorithm. This is followed by a set of experiments on two social dilemmas. A *repeated prisoner's dilemma* game is used to show how additional resources lead to non-prescriptive strategies that are more individually rational. A robot coordination problem *battle of the sexes* is discussed next, incorporating a simplified *BayesAct* model in order to more clearly examine the properties of the planning method. Next, I demonstrate POMCP-C on a realistic, affectively aware health-care assistive device for persons with dementia. Finally, I review experiments with POMCP-C on a non-affective robot navigation problem. The chapter closes with related work and conclusions.

3.2 Affect Control Theory

Affect Control Theory (ACT) arises from work on the psychology and sociology of human social interaction [74]. ACT proposes that social perceptions, behaviours, and emotions are guided by a psychological need to minimize the differences between culturally shared fundamental affective sentiments about social situations and the transient impressions resulting from the interactions between elements within those situations. Fundamental sentiments, $\mathbf{f}$, are representations of social objects, such as interactants' identities and behaviours, as vectors in a 3D affective space, hypothesised to be a universal organising principle of human socio-emotional experience [136]. The basis vectors of affective space are called Evaluation/valence, Potency/control, and Activity/arousal (EPA). EPA profiles of concepts can be measured with the *semantic differential*, a survey technique where respondents rate affective meanings of concepts on numerical scales with opposing adjectives at each end (e.g., good, nice vs. bad, awful for E, weak, little vs. strong, big for P, and calm, passive vs. exciting, active for A). Affect control theorists have compiled lexicons of a few thousand words along with average EPA ratings obtained from survey participants who are knowledgeable about their culture [75]. For example, most English speakers agree that professors are about as nice as students (E), more powerful (P) and less active (A). The corresponding EPAs are $[1.7, 1.8, 0.5]$ for professor and $[1.8, 0.7, 1.2]$ for student[2]. In

[2] All EPA labels and values in the paper are taken from the Indiana 2002-2004 ACT lexicon [75]. Values range by historical convention from -4.3 to $+4.3$.

Japan, professor has the same P (1.8) but students are seen as less powerful (-0.21).

The three dimensions were found by Osgood to be extremely robust across time and cultures. More recently these three dimensions are also thought to be related directly to intrinsic reward [57]. That is, it seems that reward is assessed by humans along the same three dimensions: Evaluation roughly corresponds with expected value, Potency with risk (e.g. powerful things are more risky to deal with, because they do what they want and ignore you), and Activity corresponds roughly with uncertainty, increased risk, and decreased values (e.g. faster and more excited things are more risky and less likely to result in reward) [57]. Similarly, Scholl argues that the three dimensions are in correspondence with the major factors governing choice in social dilemmas [164]. Evaluation is a measure of affiliation or correspondence between outcomes: agents with similar goals will rate each other more positively. Potency is a measure of dependence: agents who can reach their goals independently of other agents are more powerful. Activity is a measure of the magnitude of dependence: agents with bigger payoffs will tend to be more active.

Social events can cause transient impressions, $\boldsymbol{\tau}$ (also three dimensional in EPA space) of identities and behaviours that may deviate from their corresponding fundamental sentiments, $\mathbf{f}$. ACT models this formation of impressions from events with a grammar of the form actor-behaviour-object. Consider for example a professor (actor) who yells (behaviour) at a student (object). Most would agree that this professor appears considerably less nice (E), a bit less potent (P), and certainly more aroused (A) than the cultural average of a professor. Such transient shifts in affective meaning caused by specific events are described with models of the form $\boldsymbol{\tau}' = \boldsymbol{M}\mathscr{G}(\mathbf{f}', \boldsymbol{\tau})$, where $\boldsymbol{M}$ is a matrix of statistically estimated prediction coefficients from empirical impression-formation studies and $\mathscr{G}$ is a vector of polynomial features in $\mathbf{f}'$ and $\boldsymbol{\tau}$. In ACT, the weighted sum of squared Euclidean distances between fundamental sentiments and transient impressions is called *deflection*, and is hypothesised to correspond to an aversive state of mind that humans seek to avoid. This *affect control principle* allows ACT to compute *prescriptive* actions for humans: those that minimize the deflection. Emotions in ACT are computed as a function of the difference between fundamentals and transients [74], and are thought to be communicative signals of vector deflection that help maintain alignment between cooperative agents. ACT has been shown to be highly accurate in explaining verbal behaviours of mock leaders in a computer-simulated business [165], and group dynamics [76], among others [125].

3.3 Bayesian Affect Control Theory

Recently, ACT was generalised and formulated as a POMDP for human-interactive artificially intelligent systems [81]. This new model, called *BayesAct*, generalises the original theory in three ways. First, sentiments and impressions are viewed as probability distributions over latent variables (e.g., $\mathbf{f}$ and $\boldsymbol{\tau}$) rather than points in the EPA space, allowing for multimodal, uncertain and dynamic affective states to be modeled and learned. Second, affective interactions are augmented with *propositional* states and actions (e.g. the usual state and action space considered in AI applications). Third, an explicit reward function allows for goals that go beyond simple deflection minimization. I give a simplified description here; more details are provided in the original *BayesAct* research paper [81]. A graphical model is shown in Figure 3.1.

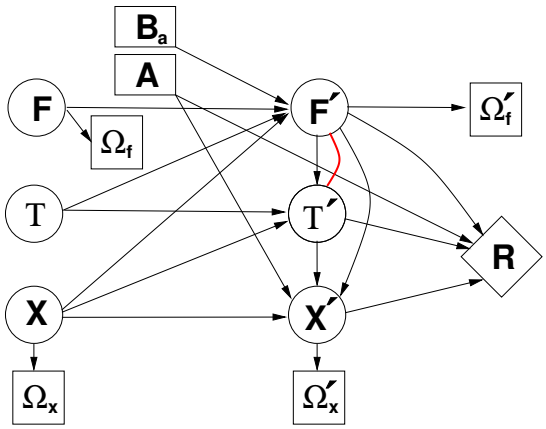


Figure 3.1: Two time slices of a factored POMDP for *BayesAct*.

A *BayesAct* POMDP models an interaction between two agents (human or machine) denoted *agent* and *client*. The state, $\boldsymbol{s}$, is the product of six 3-dimensional continuous random variables corresponding to fundamental and transient sentiments about the *agent*'s identity $(\mathbf{F}_a, \mathbf{T}_a)$, the current (*agent* or *client*) behaviour $(\mathbf{F}_b, \mathbf{T}_b)$ and the *client*'s identity $(\mathbf{F}_c, \mathbf{T}_c)$. I use $\mathbf{F} = \{\mathbf{F}_a, \mathbf{F}_b, \mathbf{F}_c\}$ and $\mathbf{T} = \{\mathbf{T}_a, \mathbf{T}_b, \mathbf{T}_c\}$. The state also contains an application-specific set of random variables $\mathbf{X}$ that are interpreted as *propositional* (i.e. not *affective*) elements of the domain (e.g. whose turn it is, game states - see Section 3.6), and we write $\boldsymbol{s} = \{\mathbf{f}, \boldsymbol{\tau}, \mathbf{x}\}$. Here the *turn* is deterministic (*agent* and *client* take turns), although this is not necessary in *BayesAct*. The *BayesAct* reward function is application-specific over $\mathbf{x}$. The state is not observable, but observations $\boldsymbol{\Omega_x}$ and $\boldsymbol{\Omega_f}$ are obtained for

$\mathbf{X}$ and for the affective behaviour $\mathbf{F}_b$, and modeled with probabilistic observation functions $Pr(\boldsymbol{\omega_x}|\mathbf{x})$ and $Pr(\boldsymbol{\omega_f}|\mathbf{f}_b)$, respectively.

Actions in the *BayesAct* POMDP are factored in two parts: $\mathbf{b_a}$ and a, denoting the *affective* and *propositional* components, respectively. For example, if a tutor gives a hard exercise to do, the manner in which it is presented, and the difficulty of the exercise, combine to form an affective impression $\mathbf{b_a}$ that is communicated. The actual exercise (content, difficulty level, etc) is the *propositional* part, a.

The state dynamics factors into three terms as

$$Pr(\boldsymbol{s}'|\boldsymbol{s}, \mathbf{b_a}, a) = Pr(\boldsymbol{\tau}'|\boldsymbol{\tau}, \mathbf{f}', \mathbf{x}) Pr(\mathbf{f}'|\mathbf{f}, \boldsymbol{\tau}, \mathbf{x}, \mathbf{b_a}) Pr(\mathbf{x}'|\mathbf{x}, \mathbf{f}', \boldsymbol{\tau}', a), \tag{3.1}$$

and the fundamental behaviour, $\mathbf{F}_b$, denotes either observed *client* or taken *agent* affective action, depending on whose *turn* it is (see below). That is, when *agent* acts, there is a deterministic mapping from the affective component of his action ($\mathbf{b_a}$) to the *agent*'s behaviour $\mathbf{F}_b$. When *client* acts, *agent* observes $\boldsymbol{\Omega_f}$ (the affective action of the other agent). The third term in the factorization of the state dynamics is the *Social Coordination Bias*, and is described in Section 3.4. Now I focus on the first two terms.

The transient impressions, $\mathbf{T}$, evolve according to the impression-formation operator in ACT ($\boldsymbol{M\mathscr{G}}$), so that $Pr(\boldsymbol{\tau}'|...)$ is deterministic. Fundamental sentiments are expected to stay approximately constant over time, but are subject to random drift (with noise $\boldsymbol{\Sigma}_f$) and are expected to also remain close to the transient impressions because of the *affect control principle*. Thus, the dynamics of $\mathbf{F}$ is[3]:

$$Pr(\mathbf{f}'|\mathbf{f}, \boldsymbol{\tau}) \propto e^{-\psi(\mathbf{f}', \boldsymbol{\tau}) - \xi(\mathbf{f}', \mathbf{f})} \tag{3.2}$$

where $\psi \equiv (\mathbf{f}' - \boldsymbol{M\mathscr{G}}(\mathbf{f}', \boldsymbol{\tau}))^T \boldsymbol{\Sigma}^{-1} (\mathbf{f}' - \boldsymbol{M\mathscr{G}}(\mathbf{f}', \boldsymbol{\tau}))$ combines the *affect control principle* with the impression formation equations, assuming Gaussian noise with covariance $\boldsymbol{\Sigma}$. The inertia of fundamental sentiments is $\xi \equiv (\mathbf{f}' - \mathbf{f})^T \boldsymbol{\Sigma}_f^{-1} (\mathbf{f}' - \mathbf{f})$, where $\boldsymbol{\Sigma}_f$ is diagonal with elements $\beta_a, \beta_b, \beta_c$. The two terms can then be combined into a single Gaussian with mean $\boldsymbol{\mu}_n$ and covariance $\boldsymbol{\Sigma}_n$ that are non-linearly dependent on the previous state, $\boldsymbol{s}$. The state dynamics are non-linear due to the features in $\mathscr{G}$. This means that the belief state will be non-Gaussian in general, and *BayesAct* uses a *bootstrap filter* [50] to compute belief updates.

The distribution in (3.2) gives the prescribed (if *agent* turn), or expected (if *client* turn), action as the component $\mathbf{f}'_b$ of $\mathbf{f}'$. Thus, by integrating over $\mathbf{f}'_a$ and $\mathbf{f}'_c$ and the previous state, we obtain a probability distribution, $\pi^\dagger$, over $\mathbf{f}'_b$ that acts as a *normative action bias*: it

[3] I leave out the dependence on $\mathbf{x}$ for clarity, and on $\mathbf{b_a}$ since this is replicated in $\mathbf{f}'_b$.

tells the agent what to expect from other agents, and what action is expected from it in belief state $b(\boldsymbol{s})$:

$$\pi^{\dagger}(\mathbf{f}'_b) = \int_{\mathbf{f}'_a,\mathbf{f}'_c} \int_{\boldsymbol{s}} Pr(\mathbf{f}'|\mathbf{f},\boldsymbol{\tau},\mathbf{x}) b(\boldsymbol{s}) \tag{3.3}$$

3.4 *BayesAct* Instances

As affective identities $(\mathbf{f}_a, \mathbf{f}_c)$ are latent (unobservable) variables, they are learned (as inference) in the POMDP. If behaving normatively (according to the *normative action bias*), an agent will perform affective actions $\mathbf{b_a} = \arg\max_{\mathbf{f}'_b} \pi^{\dagger}(\mathbf{f}'_b)$ that allow other agents to infer what his (true) identity is. The *normative action bias* (NAB) defines an affective signaling mechanism as a shared set of prescriptions for translating information about identity into messages. In *BayesAct*, the NAB is given by Equation (3.3).

The NAB is only prescriptive: all agents are free to select individually what they really send, allowing for deception (e.g. "faking" an identity by sending incorrect information in the affective dimension of communication). Possible outcomes are manipulation (the other agent responds correctly, as its own identity, to the "fake" identity), and altercasting (the other agent assumes a complementary identity to the faked identity, and responds accordingly), both possibly leading to gains for the deceptive agent.

The dynamics of $\mathbf{X}$ is given by $Pr(\mathbf{x}'|\mathbf{f}', \boldsymbol{\tau}', \mathbf{x}, a)$, that I refer to as the *social coordination bias* (SCB): it defines what agents are expected to do (how the state is expected to change, including other agents' propositional behaviours) in a situation $\mathbf{x}$ when action a was taken that resulted in sentiments $\mathbf{f}'$ and $\boldsymbol{\tau}'$. For example, we may expect faster student learning if deflection is low, as cognitive resources do not need to be spent dealing with mis-alignment.

The SCB is a set of shared rules about how agents, when acting normatively, will behave *propositionally* (action a, as opposed to affectively with action $\mathbf{b_a}$). Assuming identities are correctly inferred (as insured by the shared nature of the NAB), each agent can both recognize the type of the other agent and can thereby uncover an optimistic policy[4] that leads to the normative mean accumulated future reward (as defined by the social coordination bias). However, with sufficient resources, an agent can use this prescribed action as a heuristic only, searching for nearby actions that obtain higher individual reward. For example, a teacher who seems very powerful and ruthless at the start of a class, often may choose to do so (in a way that would be inappropriate in another setting, e.g., the home, but is appropriate within the classroom setting) in order to establish a longer-term

[4] optimistic in the sense that it assumes all agents will also follow the same normative policy.

relationship with her students. The teacher's actions feel slightly awkward if looked at in the context of the underlying social relationship with each student (e.g. as would be enacted according to normative *BayesAct*), but are leading to longer-term gains (e.g. the student passes).

Thus, the NAB (along with a communication mechanism) allows the relaying of information about identity, while the SCB allows agents to make predictions about other agents' future actions *given* the identities. This combination allows agents to assume cooperative roles in a joint task, and is used as an emotional "fast thinking" heuristic (Kahneman's "System 1" [89]). If agents are fully cooperative and aligned, then no further planning is required to ensure goal achievement. Agents do what is expected (which may involve planning over $\mathbf{X}$, but not $\mathbf{F}$ and $\mathbf{T}$), and expect others to as well. However, when alignment breaks down, or in non-cooperative situations, then slower, more deliberative ("System 2") thinking arises. The Monte-Carlo method in Section 3.5 naturally trades-off slow vs. fast thinking.

3.5 Proposed Algorithm: POMCP-C

I now investigate how to plan in *BayesAct*. I first review POMCP, a well-known MCTS algorithm, and then present my variant POMCP-C for planning in *BayesAct*.

3.5.1 POMCP

POMCP [178] is a Monte-Carlo tree search algorithm for POMDPs that progressively builds a search tree consisting of nodes representing histories and branches representing actions or observations. It does this by generating samples from the belief state, and then propagating these samples forward using a blackbox simulator (the known POMDP dynamics). The nodes in the tree gather statistics on the number of visits, states visited, values obtained, and action choices during the simulation. Future simulations through the same node then use these statistics to choose an action according to the UCB1 [14] formula, which adds an exploration bonus to the value estimate based on statistics of state visits (less well-visited states are made to look more salient or promising). Leaves of the tree are evaluated using a set of *rollouts*: forward simulations with random action selection. The key idea is that fast and rough rollouts blaze the trail for the building of the planning tree, which is more carefully explored using the UCB1 heuristic. POMCP uses a timeout (processor or clock time) providing an anytime solution.

Concretely, the algorithm proceeds as follows. Each node represents a history h, which is a sequence of actions and observations that have occurred up to time t. For a given input history h, the SEARCH procedure iteratively generates a sample s from the belief state at h, and calls the procedure SIMULATE(s, h), which does the following steps:

1. If h is not in the tree (i.e. this history has never been encountered/explored in previous calls to SEARCH), it is added, and a node for the history ha is created for every legal action a. A ROLLOUT ensues, where s is propagated forward using the POMDP dynamics until the horizon is reached. The total discounted reward gained from visiting each state in the rollout is returned and assigned to ha.

2. Otherwise, the UCB1 formula selects the best action a to be taken on s. The blackbox simulator uses a to propagate s to a new state s' and receives an observation o in the process. Then SIMULATE is called on s' and the updated history hao. When the recursive call ends, the statistics of the node h are updated, and the reward is accumulated.

Finally, SEARCH returns the highest value action at h, this action is taken, an observation is received, and the search tree is pruned accordingly.

POMCP has been shown to work on a range of large-scale domains, and would work "as is" with continuous states (as sampled histories), but is restricted to work with only discrete actions and discrete observations because every time a new node is added to the tree, a branch is created for each possible action.

3.5.2 POMCP-C

In my algorithm, POMCP-C (see Algorithm 1), I make use of an *action bias*, π_{heur}: a probability distribution over the action space that guides action choices[5]. A sample from this distribution outputs an action which is assumed to be somewhat close to optimal. In *BayesAct*, we naturally have such a bias: the normative action bias (for $\mathbf{b_a}$) and the social coordination bias (for a). The idea of such a bias is generalizable to other domains too; I will later examine a robot navigation problem as an example.

At each node encountered in a POMCP-C simulation (at history h), an action-observation pair is randomly sampled as follows. First, a random sample is drawn from the action bias, $\mathbf{a} \sim \pi_{heur}$. The action $\mathbf{a}$ is then compared to all existing branches at the current history,

[5]The idea of using a heuristic to guide action selection in POMCP was called *preferred actions* [178].

and a new branch is only created if it is significantly different, as measured by distance in the action space (Euclidean for $\mathbf{b_a}$, binary for a) and a threshold parameter δ_a ('action resolution'), from any of these existing branches. If a new branch is created, the history ha is added to the planning tree, and is evaluated with a rollout as usual. If a new branch is not created, then a random sample o is drawn from the observation distribution $Pr(o|h,a)$[6].

The continuous observation space raises two significant problems. First, the branching factor for the observations is infinite, and no two observations will be sampled twice. To counter this, I use a dynamic discretisation scheme for the observations, in which I maintain $\mathbf{o}(h)$, a set of sets of observations at each history (tree node). So $\mathbf{o}(h) = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_{N_o}\}$, where $N_o \in \mathbb{N}$. A new observation o is either added to an existing set $\mathbf{o}_j$ if it is close enough to the mean of that set (i.e. if $|o - \bar{\mathbf{o}}_\mathbf{j}| < \delta_o$ where δ_o is a constant, the 'observation resolution'), or, if not, it creates a new set $\mathbf{o}_{N_o+1} = \{o\}$. This simple scheme allows us to dynamically learn the observation discretisation.

The second problem raised by continuous observations stems from the fact that POMCP uses a black box simulator that should draw samples from the same distribution as the environment does. Thus, the simulated search tree replicates actual trajectories of belief, and can be re-used after each action and observation in the real world (after each pruning of the search tree). This works for discrete observations, but it may not work for continuous observations since the same observation will rarely be encountered twice. Here, I prune the tree according to the closest observation set $\mathbf{o}_j$ to the observation obtained.

3.5.3 Extended POMCP-C

Here, I give an extended version of POMCP-C (Algorithm 2) that deals with continuous observation spaces more prudently. Here, tree regeneration is initiated if the belief stored in the tree is sufficiently different than the actual belief encountered.

As mentioned previously, the simulated search tree in POMCP replicates actual trajectories of belief, and can be re-used after each action and observation in the real world. This may not work for continuous observations. That is, the belief stored at a node is based on sampled action-observation pairs from the parent node in the tree, which may be significantly different from the *actual* action-observation pair. To handle this problem, first I prune the search tree according to the closest observation set $\mathbf{o}_j$ to the observation obtained. Then, I do a check after each prune to see if the belief state stored at the node is 'similar' to

[6]POMCP-C also uses a cut-off N_A^{max} on the branching factor, which is not strictly necessary, but included for completeness.

Algorithm 1: POMCP-C

Procedure SEARCH(B^*, h)

repeat
  if $h = \emptyset$ **then**
    $s \sim B^*$
  else
    $s \sim B(h)$
  end
  SIMULATE$(s, h, 0)$
until *TIMEOUT*()
return $\arg\max_b V(hb)$

Procedure ROLLOUT(s, h, d)

if $\gamma^d < \epsilon$ **then**
  return *0*
else
  $a \sim \pi_{\text{rollout}}(h; \cdot)$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  return
    $r + \gamma$.ROLLOUT$(s', hao, d + 1)$
end

Function DiscretizeObs(o, h)

if $\exists_{\mathbf{o_j} \in \mathbf{o}(h)} : |o - \bar{\mathbf{o}}_\mathbf{j}| < \delta_o$ **then**
  $\mathbf{o_j} \leftarrow \mathbf{o_j} \cup \{o\}$
  return $\bar{\mathbf{o}}_\mathbf{j}$
else
  $\mathbf{o}(h) \leftarrow \mathbf{o}(h) \cup \{\{o\}\}$
  return o
end

Procedure SIMULATE(s, h, d)

if $\gamma^d < \epsilon$ **then**
  return *0*
end
if $N_A(h) < N_A^{max}$ **then**
  $a \sim \pi_{heur}(s)$
  if $\mathbf{a}(h) = \emptyset \vee \forall_{a_j \in \mathbf{a}(h)} |a - a_j| > \delta_a$ **then**
    $i \leftarrow N_A(h)$
    $T(hi) \leftarrow (N_{init}(hi), V_{init}(hi), \emptyset)$
    $N_A(h) \leftarrow N_A(h) + 1$
    $\mathbf{a}(h) \leftarrow \mathbf{a}(h) \cup \{a\}$
    return ROLLOUT*(s,h,d)*
  end
end
$i \leftarrow \arg\max_{j=1...N_A(h)} V(hj) + c\sqrt{\frac{\log N(h)}{N(hj)}}$
$(s', o, r) \sim \mathcal{G}(s, a_i(h))$
$o^\dagger \leftarrow$ DiscretizeObs(o, h)
$R \leftarrow r + \gamma$.SIMULATE$(s', ha_i(h)o^\dagger, d + 1)$
$B(h) \leftarrow B(h) \cup \{s\}$
$N(h) \leftarrow N(h) + 1$
$N(hi) \leftarrow N(hi) + 1$
$V(hi) \leftarrow V(hi) + \frac{R - V(hi)}{N(hi)}$
return R

Procedure PruneTree(h, a, o)

$i^* \leftarrow \arg\min_i |a - a_i(h)|$
$j^* \leftarrow \arg\min_j |o - \bar{\mathbf{o}}_j|$
$T \leftarrow T(hi^*j^*)$

the one we get from updating the particle filter. I do this by computing the true belief state $B^*(hao)$ based on the actual observation, o, and compare it to the belief state $B(hao)$ stored in the pruned search tree at the root. That is, given $B^*(hao) \propto P(o|s')P(s'|h, a)B^*(h)$, and $B(hao)$ (in the search tree), I compute $\Delta_B = dist(B^*(hao), B(hao))$, where $dist$ is

Algorithm 2: POMCP-C (Extended)

Procedure SEARCH(B^*, h)

repeat
  $\Delta_B \leftarrow dist(B^*, B(h))$
  $s \sim B^*$
  $regen = False$
  with *probability* $\propto \Delta_B$
    $regen \leftarrow True$
  end
  SIMULATE($s, h, 0, regen$)
until *TIMEOUT*()
return $\arg\max_b V(hb)$

Procedure ROLLOUT(s, h, d)

if $\gamma^d < \epsilon$ **then**
  return *0*
else
  $a \sim \pi_{\text{rollout}}(h; \cdot)$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  return
   $r + \gamma$.ROLLOUT($s', hao, d+1$)
end

Function DiscretizeObs(o, h)

if $\exists_{\mathbf{o_j} \in \mathbf{o}(h)} : |o - \bar{\mathbf{o}}_\mathbf{j}| < \delta_o$ **then**
  $\mathbf{o_j} \leftarrow \mathbf{o_j} \cup \{o\}$
  return $\bar{\mathbf{o}}_\mathbf{j}$
else
  $\mathbf{o}(h) \leftarrow \mathbf{o}(h) \cup \{\{o\}\}$
  return o
end

Procedure PruneTree(h, a, o)

$i^* \leftarrow \arg\min_i |a - a_i(h)|$
$j^* \leftarrow \arg\min_j |o - \bar{\mathbf{o}}_j|$
$T \leftarrow T(hi^*j^*)$

Procedure SIMULATE($s, h, d, regen$)

if $\gamma^d < \epsilon$ **then**
  return *0*
end
if $N_A(h) < N_A^{max}$ **then**
  $a \sim \pi_{heur}(s)$
  if $\mathbf{a}(h) = \emptyset \vee \forall_{a_j \in \mathbf{a}(h)} |a - a_j| > \delta_a$ **then**
    $i \leftarrow N_A(h)$
    $T(hi) \leftarrow (N_{init}(hi), V_{init}(hi), \emptyset)$
    $N_A(h) \leftarrow N_A(h) + 1$
    $\mathbf{a}(h) \leftarrow \mathbf{a}(h) \cup \{a\}$
    return ROLLOUT*(s,h,d)*
  end
end
if *regen* **then**
  $\Delta_s = Pr(s, B(h))$
  if *Bernouilli*(Δ_s) **then**
    $i = \arg\min_i V(hi)$
    $a_i(h) \sim \pi_{heur}(s)$;
     $T(hi) \leftarrow (N_{init}(hi), V_{init}(hi), \emptyset)$
    return ROLLOUT*(s,h,d)*
  end
end
$i \leftarrow \arg\max_{j=1...N_A(h)} V(hj) + c\sqrt{\frac{\log N(h)}{N(hj)}}$
$(s', o, r) \sim \mathcal{G}(s, a_i(h))$
$o^\dagger \leftarrow$ DiscretizeObs(o, h)
$R \leftarrow r + \gamma$.SIMULATE($s', ha_i(h)o^\dagger, d+1, regen$)
$B(h) \leftarrow B(h) \cup \{s\}$
$N(h) \leftarrow N(h) + 1$
$N(hi) \leftarrow N(hi) + 1$
$V(hi) \leftarrow V(hi) + \frac{R - V(hi)}{N(hi)}$
return R

some probability distance measure. For example, the KL Divergence could be computed from sample sets using [143].

Now I check Δ_B against a settable threshold parameter, and re-initialise the entire search tree if the threshold is exceeded. Alternatively, I dynamically regenerate certain portions of the search tree based on Δ_B, as follows. While drawing samples during the POMCP-C search, with probability proportional to Δ_B, I set a flag *regen*. If *regen* is set, the tree search will sometimes (with probability that the current state s is not a draw from the belief state, $B(h)$, stored at node h), generate a new action to take from the action bias. Subsequently, there can be one of two operating modes: "REPLACE"or "ADD". In "REPLACE" mode (default), the new actions from the action bias replace the worst performing action branches. In "ADD" mode, this new action gets added to the list. If we use "ADD" mode, we must be careful to sometimes remove actions as well, using a garbage collector. It is also advisable to note that in "ADD" mode, the UCB1 formula may be significantly affected. Any new actions will have a low $N(ha)$ and a high $V(ha)$, so they will likely be selected often by UCB1. The "ADD" mode is not shown in Algorithm 2 for this reason. It is also possible to replace or add a new action if $regen = False$, but $P(s|B(h))$ is very small. This is also not shown in Algorithm 2.

3.6 Experiments

In this section, I present experiments and results for four applications.

3.6.1 Prisoner's Dilemma (Repeated)

The prisoner's dilemma is a classic two-person game in which each person can either *defect* by taking \$1 from a (common) pile, or *cooperate* by giving \$10 from the same pile to the other person. There is one Nash equilibrium in which both players defect, but when humans play the game they often are able to achieve the optimal solution where both cooperate. A rational agent would first compute the strategy for the game as the Nash equilibrium (of "defect"), and then look up the affective meaning of such an action using e.g. a set of appraisal rules, and finally apply a set of coping rules. For example, such an agent might figure out that the goals of the other agent would be thwarted, and so that he should feel ashamed or sorry for the other agent. However, appraisal/coping theories do not specify the probabilities of emotions, do not take into account the affective identities of the agents, and do not give consistent accounts of how coping rules should be formulated.

Instead, a *BayesAct* agent (called a *pd-agent* for brevity here), computes what *affective* action is prescribed in the situation (given his estimates of his and the other's identities, and of the affective dynamics), and then seeks the best propositional action ($a \in \{$*cooperate*, *defect*$\}$) to take that is consistent with this prescribed affect. As the game is repeated, the *pd-agent* updates his estimates of identity (for self and other), and adjusts his play accordingly. For example, a player who defects will be seen as quite negative, and appropriate affective responses will be to defect, or to cooperate and give a nasty look.

The normative action bias (NAB) for *pd-agents* is the usual deflection minimizing affective $\mathbf{f}_b$ given distributions over identities of *agent* and *client* (Equation 3.3). Thus, if *agent* thought of himself as a *friend* (EPA:$\{2.75, 1.88, 1.38\}$) and knew the other agent to be a *friend*, the deflection minimizing action would likely be something good (high E). Indeed, a simulation shows that one would expect a behaviour with EPA=$\{1.98, 1.09, 0.96\}$, with closest labels such as *treat* or *toast*. Intuitively, cooperate seems like a more aligned propositional action than defect. This intuition is confirmed by the distances from the predicted (affectively aligned) behaviour to *collaborate with* (EPA:$\{1.44, 1.11, 0.61\}$) and *abandon* (EPA:$\{$-2.28, -0.48, -0.84$\}$) of 0.4 and 23.9, respectively. Table 3.1 shows all combinations if each agent could also be a *scrooge* (EPA:$\{$-2.15, -0.21, -0.54$\}$). We see that a *friend* would still collaborate with a *scrooge* (in an attempt to *reform* the scrooge), a *scrooge* would abandon a *friend* (*look away from* in shame), and two scrooges would defect.

The *agent* will predict the *client*'s behavior using the same principle: compute the deflection minimising affective action, then deduce the propositional action based on that. Thus, a *friend* would be able to predict that a *scrooge* would defect. If a *pd-agent* has sufficient resources, he could search for an affective action near to his optimal one, but that would still allow him to defect. To get a rough idea of this action, we find the point on the line between his optimal action $\{0.46, 1.14, -0.27\}$ and *abandon* that is equidistant from *abandon* and *collaborate with*. This point, at which he would change from cooperation to defection, is $\{$-0.8, 0.6, -0.4$\}$ (*glare at*), which only has a slightly higher deflection than *reform* (6.0 vs 4.6). Importantly, he is *not trading off costs in the game with costs of disobeying the social prescriptions*: his resource bounds and action search strategy are preventing him from finding the more optimal (individual) strategy, implicitly favoring those actions that benefit the group and solve the social dilemma.

PD-agents are dealing with a slightly more difficult situation, as they do not know the identity of the other agent. However, the same principle applies, and the social coordination bias (SCB) is that agents will take and predict the propositional action that is most consistent with the affective action. Agents have culturally shared sentiments about the propositional actions (defection and cooperation), and the distance of the deflection minimizing action (*agent*, $\mathbf{b_a}$) or behaviour (*client*, $\mathbf{f}_b$) to these sentiments is a measure

Agent	Client	Optimal Behaviour	Closest Labels	Distance from 'collaborate'	Distance from 'abandon'
Friend	Friend	1.98, 1.09, 0.96	treat toast	0.4	23.9
Friend	Scrooge	0.46, 1.14, -0.27	reform lend money to	1.7	10.5
Scrooge	Friend	-0.26, -0.81, -0.77	curry favor look away	8.5	4.2
Scrooge	Scrooge	-0.91, -0.80, -0.01	borrow money chastise	9.6	2.7

Table 3.1: Optimal (deflection minimising) behaviours for two *pd-agents* with fixed identities friend and scrooge.

of how likely each propositional action is to be chosen (*agent* turn), or predicted (*client* turn). That is, on *agent* turn, the affective actions $\mathbf{b_a}$ will be sampled and combined with a propositional action a sample drawn proportionally to the distance from $\mathbf{b_a}$ to the shared sentiments for each a. On *client* turn, affective behaviours $\mathbf{f}_b$ will be predicted and combined with a value for a variable representing *client* play in $\mathbf{X}$ drawn proportionally to the distance from $\mathbf{f}_b$.

I model *agent* and *client* as having two (simultaneous) identities: *friend* or *scrooge* with probabilities 0.8 and 0.2, respectively. Each *pd-agent* starts with a mixture of two Gaussians centered at these identities with weights 0.8/0.2 and variances of 0.1. The SCB interprets cooperation as *collaborate with* (EPA:$\{1.44, 1.11, 0.61\}$) and defection as *abandon* (EPA:$\{$-2.28, -0.48, -0.84$\}$), and the probability of the propositional actions using a Gibbs measure over distance with a variance of 4.0. I use propositional state $\mathbf{X} = \{Turn, Ag\_play, Cl\_play\}$ denoting whose turn it is ($\in \{agent, client\}$) and *agent* and *client* state of play ($\in \{not\_played, cooperate, defect\}$). The agents' reward is only over the game (e.g. 10, 1, or 0), so there is no intrinsic reward for deflection minimization as in [81]. I use a two time-step game in which both *agent* and *client* choose their actions at the first time step, and then communicate this to each other on the second step. The agents also communicate affectively, so that each agent gets to see both what action the other agent took (cooperate or defect), and also *how* they took it (expressed in $\mathbf{f}_b$)[7]. If one were to implement this game in real life, then $\mathbf{f}_b$ would be relayed by e.g. a facial expression. I use a Gaussian observation function $Pr(\boldsymbol{\omega_f}|\mathbf{f}_b)$ with mean at $\mathbf{f}_b$ and std. dev. of $\sigma_b = 0.1$. The

[7]Agents may also relay emotions (see Sec. 3.2), but here I only use emotional labels for explanatory purposes.

simulations consist of 10 trials of 20 games/trial, but agents use an infinite horizon with a discount γ.

I simulate one *pd-agent* (*pdA*) with a POMCP-C (processor time) timeout value of t_a. The other *pd-agent* can play one of the following fixed strategies:

1. **(same)**: plays with the same timeout as *agent* $t_c = t_a$;
2. **(1.0)**: plays with a timeout of $t_c = 1s$;
3. **(co)**: always cooperates;
4. **(de)**: always defects;
5. **(to)**: two-out, cooperates twice, then always defects;
6. **(tt)**: tit-for-tat, starts by cooperating, then always repeats the last action of the *agent*;
7. **(t2)**: tit-for-two-tat, starts by cooperating, then defects if the other agent defects twice in a row;
8. **(2t)**: two-tit-for-tat, starts by cooperating, then cooperates if the other agent cooperates twice in a row.

Fixed strategy agents always relay *collaborate with* and *abandon* as $\mathbf{f}_b$ when playing cooperate and defect, respectively.

First, I consider agents that use the same timeout. In this case, if the discount factor is 0.99, both agents cooperate all the time, and end up feeling like *warm*, *earnest* or *introspective ladies*, *visitors* or *bridesmaids* (EPA$\sim \{2.0, 0.5, 1.0\}$). This occurs regardless of the amount of timeout given to both agents. Essentially, both agents are following the norm. If they don't have a long timeout, this is all they can evaluate. With longer timeouts, they figure out that there is no better option. However, if the discount is 0.9 (more discounting, so they will find short-term solutions), then again cooperation occurs if the timeout is short (less than 10s), but then one agent starts trying to defect after a small number of games, and this number gets smaller as the timeout gets longer (see Figure 3.2). With more discounting, more time buys more breadth of search (the *agent* gets to explore more short-term options), and finds more of them that look appealing (it can get away with a defection for a short while). With less discounting, more time buys more depth, and results in better long-term decisions.

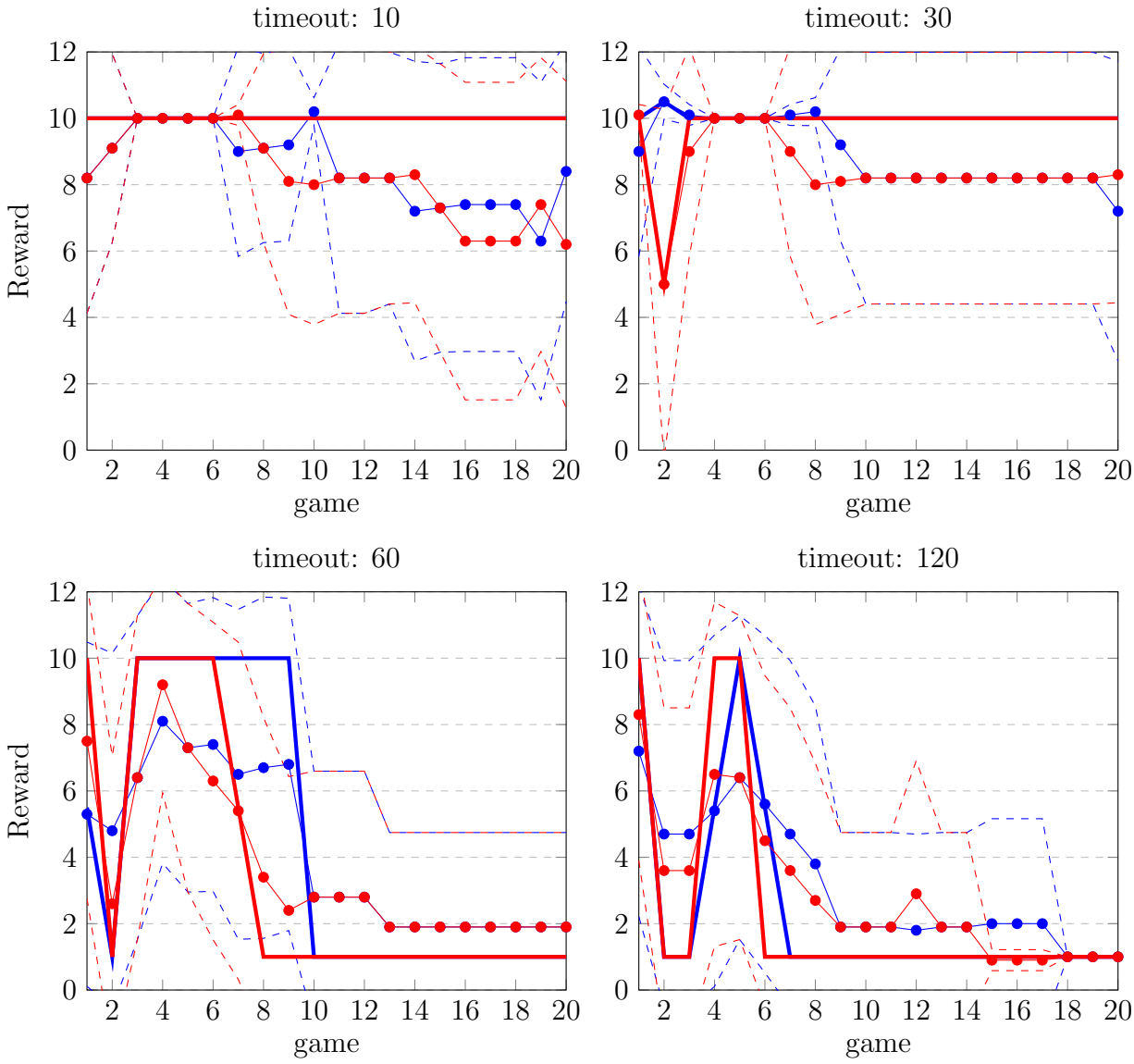


Figure 3.2: PD with client strategy: (same) and discount $\gamma = 0.9$. Red=client; Blue=agent; dashed=std.dev.; solid (thin, with markers): mean; solid (thick): median. As timeout increases, more defections give less reward for both agents.

game	post-play sentiments (*agent*)			defl-	identities		emotions		actions	
#	$\mathbf{f}_a$	$\mathbf{f}_c$	$\mathbf{f}_b$	ection	agent	client	agent	client	agent	client
1	-1.36,-0.01,-0.35	2.32,1.61,1.27	2.62,1.58,1.73	4.44	failure	newlywed	easygoing	idealistic	coop.	coop.
2	-0.66,0.04,-0.05	1.77,1.27,1.06	2.23,1.00,1.76	3.70	parolee	husband	easygoing	self-conscious	coop.	coop.
3	-0.23,-0.08,0.20	1.02,0.93,0.84	2.49,0.97,1.87	7.19	stepmother	purchaser	female	immoral	coop.	def.
4	-0.12,-0.33,0.33	0.27,0.62,0.62	2.37,0.48,1.34	4.99	stuffed_shirt	roommate	dependent	unfair	coop.	def.
5	-0.26,-0.47,0.32	-0.26,0.26,0.42	-0.59,0.41,-0.23	3.27	divorcée	gun_moll	dependent	selfish	def.	def.
6	-0.37,-0.66,0.26	-0.61,0.00,0.28	-0.10,-0.41,-0.27	2.29	divorcée	hussy	disapproving	selfish	def.	def.

Table 3.2: Example games with *client* playing (to). Identities and emotions are *agent* interpretations.

γ	(tt)	(t2)	(2t)
0.9	1.64 ± 2.24	3.98 ± 2.48	1.72 ± 2.35
0.99	7.33 ± 1.17	7.28 ± 1.68	7.63 ± 0.91

Table 3.3: Results (avg. rewards) against the tit-for strategies

Table 3.2 shows the first six games with a *client* playing two-out (to), who sends affective values of *collaborate with* $\{1.44, 1.11, 0.61\}$ and cooperates on the first two moves. This affective action makes the *pd-agent* feel much less good (E) and powerful (P) than he normally would (as a *failure*), as he'd expect a more positive and powerful response (such as *flatter* EPA=$\{2.1, 1.45, 0.82\}$) if he was a *friend*, so this supports his *scrooge* identity more strongly[8]. He infers *client* is friendly (a *newlywed* is like a *girlfriend* in EPA space). He therefore cooperates on the second round, and feels somewhat better. Then, the *client* defects on the third round, to which the *agent* responds by re-evaluating the *client* as less good (an *immoral purchaser*). He still tries to cooperate, but gives up after two more rounds, after which he thinks of the *client* as nothing but a *selfish hussy*, and himself as a *disapproving divorcée*. The *agent* consistently defects after this point. Interactions with (tt), (2t) and (t2) generally follow a similar pattern, because any defection rapidly leads to both agents adopting long-term defection strategies. However, as shown in Table 3.3 (Full results are given in A), less discounting leads to better solutions against these strategies, as longer-term solutions are found.

When playing against (co), *pd-agents* generally start by cooperating, then defect, resulting in a feeling of being a *self-conscious divorcée* (EPA:{-0.23, -0.62, 0.32}) playing against a *conscientious stepsister* (EPA:{0.12, -0.04, 0.35}). When playing against (de), *pd-agents* generally start by cooperating, but then defect, feeling like a *dependent klutz* (EPA:{-0.76, -1.26, 0.37}) playing against an *envious ex-boyfriend* (EPA:{-1.30, -0.49, -0.13}).

[8]Examples of more positive affective actions in A.

3.6.2 Affective Cooperative Robots (CoRobots)

CoRobots is a multi-agent cooperative robot game based on the classic "Battle of the Sexes" problem[9]. The asymmetrical situations are specifically interesting, wherein one robot has more resources and can do planning in order to *manipulate* the other robot, taking advantage of the social coordination bias. I start with a simplified version in which the two robots maintain affective fundamental sentiments, but do not represent the transient impressions. The normative action bias is a simple average instead of as the result of more complex impression formation equations.

Concretely, two robots, Rob1 and Rob2, move in a 1D continuous state space. I denote their positions with variables X_1 and X_2. At each time step, Rob1, Rob2 take actions $a_1, a_2 \in \mathbb{R}$ respectively. This updates their respective positions $x_i, i \in \{1, 2\}$ according to $x_i \leftarrow x_i + a_i + \nu_i$ and $\nu_i \sim \mathcal{N}(0, \sigma)$. There are two fixed locations $L_1 \in \mathbb{R}^+$ and $L_2 \in \mathbb{R}^-$. For each robot, one of these locations is the major goal g (with associated high reward r) and the other is the minor goal $\bar{g}$ (with associated low reward $\bar{r}$). A robot is rewarded according to its distance from g and $\bar{g}$, but only if the other robot is nearby. The reward for Rob$i$ is:

$$R_i(x_1, x_2) = \mathbb{I}(|x_1 - x_2| < \Delta_x)[r \cdot e^{-(x_i - g)^2/\sigma_r^2} + \bar{r} \cdot e^{-(x_i - \bar{g})^2/\sigma_r^2}], \tag{3.4}$$

where $\mathbb{I}(y) = 1$ if y is true, and 0 otherwise, and where σ_r is the reward variance, Δ_x is a threshold parameter governing how "close" the robots need to be, and $r, \bar{r} \in \mathbb{R}$, such that $r \gg \bar{r} > 0$. Both σ_r and Δ_x are fixed and known by both robots. Each robot only knows the location of its own major goal. Furthermore, at any time step, each robot can move in any direction, receives observations of the locations of both robots, and has a belief over X_1 and X_2.

In order to coordinate their actions, (which is necessary to achieve any reward at all), the robots must relay their reward locations to each other, and must choose a *leader* according to some social coordination bias. The robots each have a 3D *identity* $\mathbf{f}_a = \{\mathbf{f}_{ae}, \mathbf{f}_{ap}, \mathbf{f}_{aa}\} \in \mathbb{R}^3$ (as in *BayesAct*), where the valence, $\mathbf{f}_{ae}$, describes their goal: if $\mathbf{f}_{ae} > 0$, then $g = L_1$. If $\mathbf{f}_{ae} < 0$, then $g = L_2$. The power and activity dimensions will be used for coordination (see below). Each robot also models the identity of the other robot (the *client*[10]), $\mathbf{f}_c \in \mathbb{R}^3$. Robots can move (propositional action a) at any time step, but must coordinate their communications. That is, only one robot can communicate at a time (with affective action

[9] A husband wants to go to a football game, and his wife wants to go shopping, but neither wants to go alone. There are two pure Nash equilibria, but the optimal strategy requires coordination.

[10] I present from *agent*'s perspective, and call the other *client*.

$\mathbf{b_a}$ perceived by the other robot as $\boldsymbol{\omega_f}$), but this turn-taking behaviour is fixed. The normative action bias (NAB) in the first (simplified) CoRobots problem is the mean of the two identities:

$$\pi^{\dagger} \propto \mathcal{N}((\mathbf{f}_a + \mathbf{f}_c)/2, \Sigma_b). \tag{3.5}$$

In *BayesAct* Corobots, the NAB is given by Equation (3.3). Unless stated otherwise, the CoRobots start without any knowledge of the other's identity: their belief over $\mathbf{f}_c$ is given by $\mathcal{N}(0, 2)$. CoRobots have noisy self-identities.

The social coordination bias (that the leader will lead) defines each robot's action bias for a_i, and action prediction function (for *client*'s x) through a 2D sigmoid *leader* function, known to both agents:

$$leader(\mathbf{f}_a, \mathbf{f}_c) = \frac{1}{1 + exp(-\frac{(\mathbf{f}_{ap} - \mathbf{f}_{cp})}{\sigma_p} - \frac{(\mathbf{f}_{aa} - \mathbf{f}_{ca})}{\sigma_a})} \tag{3.6}$$

where $\sigma_a = 1.0$ and $\sigma_p = 1.0$ are constants, known to both robots. This sigmoid function is ≥ 0.5 if the *agent* estimates he is more powerful or more active than the *client* ($(\mathbf{f}_{ap} > \mathbf{f}_{cp}) \vee (\mathbf{f}_{aa} > \mathbf{f}_{ca})$) and is < 0.5 otherwise. If the *agent* is the leader, his action bias will be a Gaussian with mean at $+1.0$ in the direction of his major goal (as defined by $\mathbf{f}_{ae}$), and in the direction of the *client*'s major goal (as defined by his estimate of $\mathbf{f}_{ce}$) otherwise. *Agent*'s prediction of *client*'s motion in x is that the *client* will stay put if *client* is the leader, and will follow the *agent* otherwise, as given succinctly by:

$$Pr(x'_c | \mathbf{f}'_a, \mathbf{f}'_c) = \mathcal{N}(\mathbb{I}(leader(\mathbf{f}'_a, \mathbf{f}'_c) \geq 0.5)\lambda_a + x_c, \sigma_p) \tag{3.7}$$

where $\lambda_a = 1$ if $\mathbf{f}'_{ae} > 0$, and -1 otherwise and $\sigma_p = 1.0$.

I first investigate whether corobots can coordinate when they have identities drawn from the set of 500 human (male) identities in the ACT lexicon (see footnote 2). In the first experiment, the two identities are selected at random on each trial. Each corobot knows his self-ID ($\mathcal{N}(\text{self-ID}, 0.1)$) but does not know the other's ID ($\mathcal{N}([0.0,\ 0.0,\ 0.0], 2.0)$). Furthermore, each corobot has a stable self-identity ($\beta_a = 0.1$), but it believes that the other is less stable ($\beta_c = 2.0$). Finally, both corobots have equal POMCP-C planning resources ($\Sigma_b = 0.5, N_A^{max} = 3, \delta_a = 2.0, \delta_o = 6.0$ and $Timeout = 2.0$ seconds). The other CoRobots game parameters are $r = 100, \bar{r} = 30, L_1 = 10, L_2 = -10, \sigma_r = 2.5, \Delta_x = 1.0$ and iterations $= 30$. I run 5 sets of 100 simulated trials of the CoRobots Game with varying *environmental noise*, i.e., I add a normally distributed value, with standard deviation corresponding to the noise level, to the computation and communication of $\boldsymbol{\Omega_x}$ and $\boldsymbol{\Omega_f}$ (observations of x and $\mathbf{f}$, resp.). Figure 3.3(a) (green line) shows the mean and standard

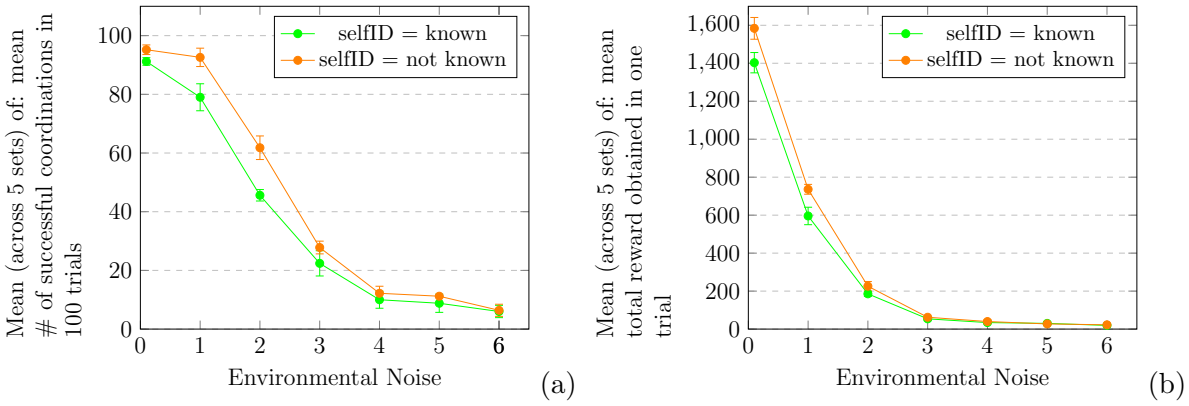


Figure 3.3: *BayesAct* Corobots cannot coordinate properly when the communication channel is bad or non-existent.

error of mean number of successful coordinations by the corobots, and (b) shows the means and standard error of the total reward per trial (in each set of 100 trials). The percentage of successful coordination falls from 91% to 6% when the environmental noise is increased, and the average total reward per trial falls from 1403 to 19.4. We see that with no environmental noise, the corobots are able to easily learn the other's identity, and can coordinate based on the social coordination bias. As the environmental noise increases, corobots are unable to easily relay identities, and require a much longer time to find cooperative solutions.

Figure 3.3 (orange line) shows results where the self-ID is also unknown initially ($\mathcal{N}([0.0, 0.0, 0.0], 2.0)$), and is less stable ($\beta_a = 2.0$). We see that the general trend is the same; however, the corobots have a higher percentage of successful coordinations, and consequently gain a higher average total reward, for the three lowest noise values. They successfully converge to a goal 95% of the times when the noise is low, and accumulate an average reward of 1584 per trial. These values fall to 6.4% and 22.5 when the noise is maximum. It is surprising to see that the corobots perform better with unknown self-IDs. This is because corobots quickly assume contrasting identities (i.e. one assumes a less powerful identity than the other) in order to coordinate. With known self-IDs, however, the corobots show less flexibility and spend the initial few iterations trying to convince and pull the other corobot towards themselves. Due to this rigidity, these corobots suffer a lot when they have similar power; this does not happen when the self-ID is unknown.

Next, I investigate whether one agent can *manipulate* the other. A manipulation is said to occur when the weaker and less active agent deceives the client into believing that

the agent is more powerful or active, thereby persuading the client to converge to the agent's major goal g (to within $\pm|0.2g|$). In order to demonstrate manipulative behaviour, I introduce asymmetry between the two agents by changing the parameters Σ_b, N_A^{max} and $Timeout$ for one agent (unbeknownst to the other). In addition, I allow this agent to start with a slightly better estimate of the other's identity. This agent will then sample actions that are farther from the norm than expected by the other agent, and will allow such an agent to "fake" his identity so as to manipulate the other agent. The agent's and client's self-identities are noisy ($\sigma = 0.1$) versions of $[2.0, -1.0, -1.0]$ and $[-2.0, 1.0, 1.0]$ respectively, $r = 100, \bar{r} = 30, L_1 = 5, L_2 = -5, \Delta_x = 1, \sigma_r = 2.5, \delta_a = 2.0, \delta_o = 6.0, N_A^{max} = 3, \Sigma_b = 0.5$ and $Timeout = 2.0$ for both robots. Each game is set to run for 40 iterations, and starts with the agent and client located at 0.0. Since $g_a = 5$, $g_c = -5$, both robots should converge to $g_c = -5$ (*client* is leader) if following normative actions.

When $N_A^{max} = 3$, $\Sigma_b = 0.5$, and $Timeout = 2.0$ for the agent, the agent displays manipulative behaviour in only 80/1000 games, as expected (both follow normative behaviour). If we allow the *agent* to start with a better estimate of the *client*'s identity (*agent*'s initial belief about $\mathbf{f}_c$ is a Gaussian with mean $[-2.0, 1.0, 1.0]$ and variance 1.0), we see manipulative behaviour in almost twice as many games (150). However, it is not a significant proportion, because although it spends less time learning the other's identity, it cannot find much more than the normative behaviour.

Next, I also give the *agent* more planning resources by setting $N_A^{max} = 6$ and $\Sigma_b = 2$ for the agent, and I run 10 sets of 100 simulated trials for each of the following values of agent's $Timeout : 2, 30, 60, 120, 360, 600$ seconds[11]. Figure 3.4(a) (solid red line) shows the means and standard error of the means of number of agent manipulations (in each set of 100 trials), plotted against agent's $Timeout$. Figure 3.4(b) (solid red line) shows means and standard error of agent reward per trial (in each set of 100 trials). As the model incorporates noise in movements as well as observations, the robots spend about 20 initial iterations coordinating with each other to choose a leader, during which time they do not receive reward. Thus, a realistic upper bound on the *agent*'s reward is $20 \times 100 = 2000$. Figure 3.4 shows that at $Timeout = 2$, the agent accumulates a reward of 425 on average, which is only 21% of the realistic maximum. At $Timeout = 600$, the reward rises to 1222, which is about 61% of this realistic maximum. This makes sense given the manipulation rate of about 48%. There is a diminishing rate of return as timeout increases in Figure 3.4 that is explained by the exponential growth of the MCTS search tree as $Timeout$ increases linearly. The results are relatively insensitive to the choice of parameters such as δ_a and δ_o.

[11] I use a Python implementation that is unoptimized. An optimised version will result in realistic timeouts.

I also tried solving the CoRobots problems using POMCP by discretising the action space. However, even with only 5 discrete actions per dimension, there are 5^4 actions, making POMCP infeasible.

Finally, I play the CoRobots Game with *BayesAct* Robots. This means that the normative behaviour is the deflection minimising action given by Affect Control Theory, instead of Equation (3.5), and the transient impressions are used to compute the deflection. The game trials are set up exactly as before, and the results are shown in Figure 3.4 (blue line). As expected, we see the same trends as those obtained previously, but with correspondingly lower values as the transient impressions are used and introduce further complexity to the planning problem (18D state space rather than 9D). These results demonstrate that the POMCP-C algorithm is able to find and exploit manipulative affective actions within the *BayesAct* POMDP, and gives some insight into manipulative affective actions in *BayesAct*.

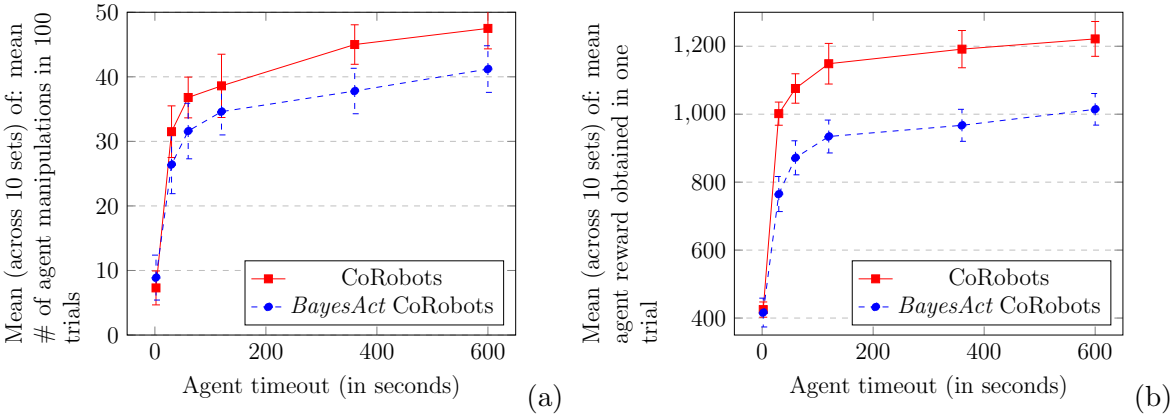


Figure 3.4: CoRobots: With higher N_A^{max}, Σ_b and $Timeout$, a weaker and less active agent becomes increasingly manipulative by 'faking' his identity, and accumulates higher rewards.

3.6.3 Affective Handwashing System

Persons with dementia (PwD, e.g. Alzheimer's disease) have difficulty completing activities of daily living, such as handwashing, preparing food and dressing. The short-term memory impairment that is a hallmark of Alzheimer's disease leaves sufferers unable to recall what step to do next, or what important objects look like. A POMDP-based agent called *COACH* has been developed (with discrete states, actions and observations) that can assist PwD by monitoring the person and providing audio-visual cues when the person gets

Table 3.4: Means and the standard error of the means (of each set of 10 simulations) of the number of interactions, and of the last planstep reached for simulations between *agent* and *client*.

True Client Identity	Agent Action		Total number of Interactions		Last Planstep Reached	
	Prompt	***No-prompt***	***W/o POMCP-C***	***With POMCP-C***	***W/o POMCP-C***	***With POMCP-C***
Elder	*BayesAct*		18.13 ± 9.80	17.67 ± 12.05	6.63 ± 0.39	6.54 ± 0.51
	prompt	mind	69.22 ± 9.28	67.75 ± 5.87	4.52 ± 0.58	5.02 ± 0.55
	confer with	mind	18.96 ± 8.60	17.96 ± 5.79	6.72 ± 0.33	6.81 ± 0.19
	command	mind	90.66 ± 5.61	85.68 ± 7.52	2.9 ± 0.63	2.12 ± 0.87
Patient	*BayesAct*		13.32 ± 7.3	13.07 ± 6.13	6.76 ± 0.28	6.71 ± 0.30
	prompt	mind	27.25 ± 13.22	24.11 ± 8.40	5.70 ± 0.55	5.56 ± 0.98
	confer with	mind	20.86 ± 7.08	18.68 ± 6.14	6.58 ± 0.35	6.24 ± 0.56
	command	mind	76.94 ± 10.07	77.85 ± 8.76	4.27 ± 0.71	4.88 ± 1.20
Conval-escent	*BayesAct*		16.63 ± 7.03	14.32 ± 6.61	6.74 ± 0.32	6.78 ± 0.21
	prompt	mind	48.42 ± 12.81	44.32 ± 10.68	5.66 ± 0.69	5.79 ± 0.78
	confer with	mind	18.89 ± 5.79	17.21 ± 6.23	6.68 ± 0.32	6.46 ± 0.42
	command	mind	62.24 ± 7.88	62.44 ± 7.67	5.09 ± 0.58	5.81 ± 0.61
Boss	*BayesAct*		66.60 ± 9.04	68.95 ± 8.83	5.17 ± 0.73	5.01 ± 1.23
	prompt	mind	86.67 ± 8.07	93.08 ± 6.38	3.42 ± 0.83	2.53 ± 1.20
	confer with	mind	62.38 ± 12.50	64.66 ± 16.59	5.47 ± 0.74	4.97 ± 0.44
	command	mind	90.54 ± 6.54	93.43 ± 8.46	3.18 ± 0.98	2.78 ± 1.03

"stuck" [79]. However, these prompts are pre-recorded messages that are delivered with the same emotion each time. As an important next step, I use *BayesAct* and POMCP-C to give *COACH* the ability to reason about the affective identity of the PwD, and about the affective content of the prompts and responses. Here, I investigate the properties of *BayesAct* planning for *COACH* in simulation. Details of a physical implementation of the handwashing system using *BayesAct* can be found in [115, 127].

I first describe the POMDP model of *COACH* that incorporates *BayesAct*. The handwashing system has 8 *plansteps* corresponding to the different steps of handwashing, describing the state of the water (on/off), and hands (dirty/soapy/clean and wet/dry). An eight-valued variable PS describes the current planstep. There are probabilistic transitions between plansteps described in a probabilistic plan-graph (e.g. a PwD sometimes uses soap first, but sometimes turns on the tap first). I also use a binary variable AW describing if the PwD is *aware* or not. Thus, $\mathbf{X} = \{PS, AW\}$ and the dynamics of the PS are such that if the PwD is aware, then she will advance stochastically to the next planstep (according to the plan-graph), unless the deflection is high, in which case the PwD is more likely to become confused (lose awareness). If she does not advance, she loses awareness. On the other hand, if the PwD is not aware, and is prompted when deflection is low, then she will also move forward (according to the prompt) and gain awareness. However, a

high-deflection prompt will again lead to loss of awareness, and to slower progress.

Table 3.5 shows an example simulation between the agent with the affective identity of "assistant" ($EPA = [1.5, 0.51, 0.45]$) and a client (PwD) with the affective identity of "elder" ($EPA = [1.67, 0.01, -1.03]$). The *BayesAct* agent must learn this identity (shown as $\mathbf{f}_c$ in Table 3.5) during the interaction if it wants to minimize deflection. We see in this case that the client starts with AW="yes" (1) and does the first two steps, but then stops and is prompted by the agent to rinse his hands. This is the only prompt necessary, the deflection stays low, the agent gets a reasonable estimate of the client identity ($EPA = [2.8, -0.13, -1.36]$, a distance of 1.0). I show example utterances in the table that are "made up" based on our extensive experience working with PwD interacting with a handwashing assistant. Table 3.6 shows the same client ("elder") but this time the agent always uses the same affective actions: if prompting, it "commands" the user ($EPA = [-0.09, 1.29, 1.59]$) and when not prompting it "minds" the user ($EPA = [0.86, 0.17, -0.16]$). Here we see that the agent prompts cause significant deflection, and this causes the PwD to lose awareness (to become confused) and not make any progress. The handwashing takes much longer, and the resulting interaction is likely much less satisfying.

I modify this *COACH* POMDP model by adding 3D continuous state, action and observation spaces to represent affective identities and behaviours (the normative action bias is *BayesAct*). The social coordination bias is that the PwDs progress through the task is helped by prompting, but only if the deflection is sufficiently low. I investigate the system in simulation using an agent identity of "assistant" ($EPA = [1.5, 0.51, 0.45]$). This "assistant" agent interacts with the following fixed (but unknown to the agent) client identities: "elder" ([1.67, 0.01, −1.03]), "patient" ([0.90, −0.69, −1.05]), "convalescent" ([0.3, 0.09, −0.03]), and "boss"[12] ([0.48, 2.16, 0.94]). I compare two policies, in which the affective actions (i.e. how to deliver a prompt) are either computed with *BayesAct* and POMCP-C, or are fixed (as in the current *COACH* system). In both cases, POMCP-C is used to compute a policy for propositional actions (i.e. what prompt to give). I run 10 sets of 10 simulated trials. The results are shown in Table 3.4. As expected, the fixed policy of "command" ($[-0.09, 1.29, 1.59]$) gives the worst performance in all cases. These results suggest that a fixed affective policy may work for some affective identities, but not for others, whereas the POMCP-C policy can learn and adapt to different client identities.

The difference in Table 3.4 between *BayesAct* used with POMCP-C and without is not significant. In 10 out of 16 tests, POMCP-C allows for action choices that lead to fewer interactions, and does better on average. Perhaps more interestingly, this estimate of error

[12]Many persons with Alzheimer's disease think of themselves in terms of some past identity or role.

Table 3.5: Example simulation between the agent and a client (PwD) who holds the affective identity of "elder". Affective actions are chosen by *BayesAct*. Possible utterances for ***agent*** and ***client*** are shown that may correspond to the affective signatures computed.

TURN	**CLIENT STATE**		**ACTION**		**AGENT EXPECTATION**			**CLIENT**
	AW	*PS*	Prop.	Affect	$\mathbf{f}_e$	*PS*	*AW*	**DEFL.**
initial	1	0	-	-	[0.9,-0.69,-1.05]	0	0.72	-
client	1	0	put on soap ***"[looks at sink]"***	[1.6,0.77,-1.4]	[2.3,-0.77,-1.23]	0.96	0.94	0.23
agent	1	1	- ***"[looks at client]"***	[1.3,0.26,-0.40]	[2.41,-0.81,-1.23]	1.0	≈1.0	1.07
client	1	1	turn on tap ***"oh yes, this is good"***	[2.2,0.90,-1.1]	[2.7,-0.36,-1.37]	3.0	0.99	0.99
agent	1	3	- ***"I'm here to help, Frank"***	[1.3,0.4,0.35]	[2.7,-0.37,-1.38]	3.0	≈1.0	1.47
client	1	3	- ***"this is nice"***	[2.1,0.72,-1.4]	[2.6,-0.34,-1.38]	3.0	0.01	1.14
agent	0	3	rinse hands ***"Great! Let's rinse hands"***	[1.5,0.67,0.06]	[2.6,-0.34,-1.39]	3.0	≈0.0	1.50
client	0	3	rinse hands ***"oh yes, this is good"***	[1.9,0.78,-1.4]	[2.7,-0.31,-1.44]	4.0	0.99	1.11
agent	1	4	- ***"good job Frank"***	[1.6,0.47,-0.13]	[2.7,-0.30,-1.4]	4.0	≈1.0	1.61
client	1	4	turn tap off ***"[looks at tap]"***	[2.0,0.94,-1.3]	[2.6,-0.17,-1.24]	5.9	0.96	1.19
agent	1	6	- ***"This is nice, Frank"***	[1.5,0.56,-0.35]	[2.6,-0.17,-1.2]	6.0	≈1.0	1.56
client	1	6	- ***"Oh yes, good good"***	[2.1,0.86,-1.42]	[2.8,-0.14,-.14]	6.0	≈0.0	1.22
agent	1	6	dry hands ***"[looks at]"***	[1.4,0.66,-0.06]	[2.8,-0.13,-1.36]	6.0	≈0.0	1.55
client	1	6	dry hands ***"all done!"***	[1.94,1.1,-1.9]	-	-	-	1.55
client	1	7	-	-	-	-	-	-

can be used to evaluate the quality of the action bias within the given interaction. If the POMCP-C model starts doing worse on average than a fixed policy, it is an indication that the action bias is not very good, and that there is a possible misalignment between the agent and the patient.

3.6.4 8D Intersection Problem

POMCP-C can be generalized to non-affective domains where *action biases* naturally arise. In the 8D Intersection Problem [26], a robot agent's task is to navigate a 2D

Table 3.6: Example simulation between the agent and a client (PwD) who holds the affective identity of "elder". Affective actions were fixed: if prompting, it "commands" the user and when not prompting it "minds" the user.

TURN	CLIENT STATE		ACTION		AGENT EXPECTATION			CLIENT
	AW	*PS*	Prop.	Affect	$\mathbf{f}_c$	*PS*	*AW*	DEFL.
initial	1	0	-	-	[0.9,-69,-1.05]	0	0.72	-
client	1	0	put on soap ***"[looks at sink]"***	[1.6,0.77,-1.4]	[2.3,-0.77,-1.23]	0.96	0.94	0.23
agent	1	1	- ***"[looks at client]"***	[0.85,0.17,-0.16]	[2.41,-0.81,-1.23]	1.0	≈1.0	1.34
client	1	1	turn on tap ***"oh yes, this is good"***	[2.3,0.90,-1.19]	[2.62,-0.42,-1.43]	2.98	0.99	1.21
agent	1	3	- ***"[looks at client]"***	[0.85,0.17,-0.16]	[2.7,-0.42,-1.5]	3.0	≈1.0	1.86
client	1	3	- ***"oh yes, this is good"***	[2.2,0.79,-1.47]	[2.6,-0.30,-1.4]	3.0	≈0.0	1.56
agent	0	3	rinse hands ***"Rinse your hands!"***	[-0.1,1.29,1.59]	[2.6,-0.30,-1.4]	3.0	≈0.0	4.11
client	0	3	- ***"[looks at sink]"***	[1.9,1.4,-1.7]	[2.5,-0.30,-1.3]	3.0	≈0.0	2.90
agent	0	3	rinse hands ***"Rinse your hands!"***	[-0.1,1.29,1.59]	[2.5,-0.29,-1.3]	3.0	≈0.0	5.80
client	0	3	- ***"[looks at sink]"***	[1.9,0.97,-1.9]	[2.4,-0.27,-1.26]	3.0	0.02	4.28
agent	0	3	rinse hands	[-0.1,1.29,1.59]	[2.4,-0.26,-1.27]	3.0	0.02	7.05

...continues for 85 more steps until client finally finishes ...

space to reach a goal, by avoiding a moving obstacle. The state space consists of 8-tuples $(a_x, a_y, a_{vx}, a_{vy}, ob_x, ob_y, ob_{vx}, ob_{vy}) \in \mathbb{R}^8$, where the first four values give the agent's position and velocity in 2D, and the last four values give the obstacle's position and velocity in 2D. The agent and obstacle are circular disks of radius 1. The obstacles starts at $ob_x = 6.0\ m$ and a random $ob_y \in [-8.0, 8.0]$, and moves vertically[13] with fixed velocity $(ob_{vx}, ob_{vy}) = (0.0\ m/s, 1.0\ m/s)$. The agent starts with $(a_x, a_y, a_{vx}, a_{vy}) = (0.0\ m, 0.0\ m, 1.0\ m/s, 0.0\ m/s)$, and at each time step, it decides whether to accelerate by $\pm 1.0\ m/s^2$ on the horizontal axis, to reach the goal located at $(9.0\ m, 0.0\ m)$. The agent has no knowledge of the position of the obstacle unless they are less than 4 m apart. The discount factor is 0.95, and the reward is $+10$ for reaching the goal and -10 for colliding with the obstacle. Thus, the problem has an 8D continuous state space, a discrete (binary) action space, and an 8D continuous observation space (a noisy measurement of the state).

[13] Whenever $ob_y > 8.0\ m$, it is reset to -8.0.

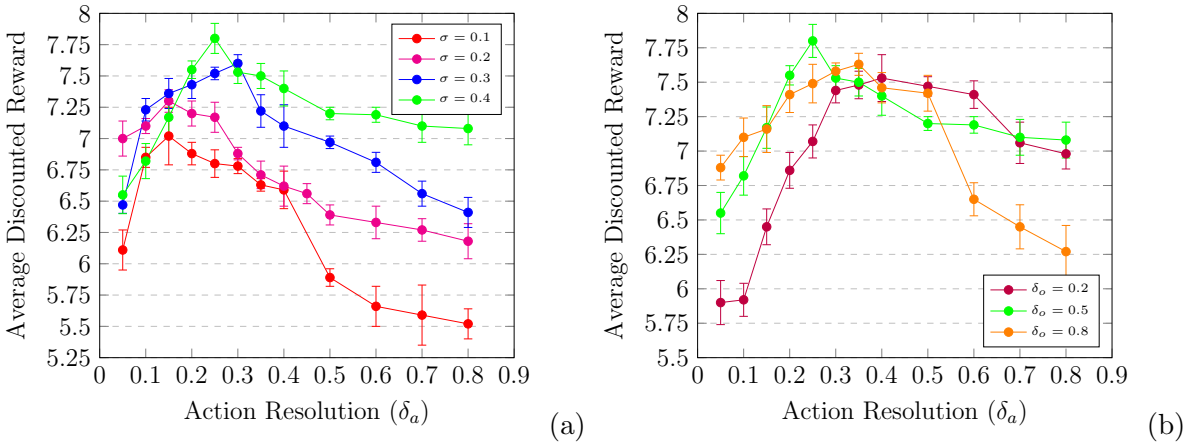


Figure 3.5: 8D Intersection Problem (continuous actions). $\sigma = 0.4$, $\delta_o = 0.5$, $N_A^{max} = 15$, $Timeout = 400$ unless otherwise noted.

I ran 1000 simulated trials with the following settings. The belief state of the agent consists of 50,000 particles, and is initialised to an 8D Gaussian with mean $(0.0, 0.0, 1.0, 0.0, 6.0, 0.0, 0.0, 1.0)$ and diagonal covariance $(10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 8.0, 10^{-4}, 10^{-4})$. When the agent and obstacle are more than 4 m apart, the agent receives a uniformly random observation of the obstacle's position and velocity. Otherwise, the mean of the observation model $P(o|x)$ is always the state x with variance 10^{-4}. In addition, the model incorporates small white noise. With the POMCP-C parameters $\delta_o = 0.5$ and $Timeout = 15s$, the algorithm achieves an average discounted reward of 5.9, compared to 5.0 by Brechtel *et al*'s point-based backups.

To further demonstrate how POMCP-C handles continuous action spaces, I devise a continuous-action version of this problem, where the agent can sample actions from the action bias $\mathcal{N}(1.0, \sigma) \cup \mathcal{N}(-1.0, \sigma)$, σ being a settable parameter. The number of particles, initial belief state and observation model remain the same. With $\sigma = 0.4$, $\delta_a = 0.25$, $\delta_o = 0.5$, $N_A^{max} = 15$ and $Timeout = 400$, POMCP-C achieves an average discounted reward of 7.8. This is a dramatic improvement compared to the discrete-action version, as the agent can fine-tune its acceleration to avoid the moving obstacle more often. Figure 3.5(a) and (b) show how the average discounted reward varies with δ_a, for different fixed values of σ and δ_o, respectively. We see that the average reward increases rapidly with increasing δ_a (as the number of actions in the tree decreases, leading to more search per action), but peaks and declines slowly because important actions are not distinguished anymore.

3.7 Related Work

Damasio has convincingly argued, both from a functional and neurological standpoint, for emotions playing a key role in decision making and for human social action [41]. His *Somatic Marker Hypothesis* is contrasted against the Platonic "high-reason" view of intelligence, in which pure rationality is used to make decisions. Damasio argues that, because of the limited capacity of working memory and attention, the Platonic view will not work. Instead, learned neural markers focus attention on actions that are likely to succeed, and act as a neural bias allowing humans to work with fewer alternatives. These *somatic markers* are "cultural prescriptions" for behaviours that are "rational relative to the social conventions and ethics" ([41], p200).

LeDoux [101] argues the same thing from an evolutionary standpoint. He theorises that the subjective feeling of emotion must take place at both unconscious and conscious levels in the brain, and that consciousness is the ability to relate stimuli to a sense of identity, among other things.

With remarkably similar conclusions coming from a more functional (economic) viewpoint, Kahneman has demonstrated that human emotional reasoning often overshadows, but is important as a guide for, cognitive deliberation [89]. Kahneman presents a two-level model of intelligence, with a fast/normative/reactive/affective mechanism being the "first on the scene", followed by a slow/cognitive/deliberative mechanism that operates if sufficient resources are available. Akerlof and Kranton attempt to formalise *fast thinking* by incorporating a general notion of identity into an economic model (utility function) [4]. Earlier work on *social identity theory* foreshadowed this economic model by noting that simply assigning group membership increases individual cooperation [186].

The idea that unites Kahneman, LeDoux, and Damasio (and others) is the tight connection between emotion and action. These authors, from very different fields, propose emotional reasoning as a "quick and dirty", yet absolutely necessary, guide for cognitive deliberation. ACT gives a functional account of the quick pathway as sentiment encoding prescriptive behaviour, while *BayesAct* shows how this account can be extended with a slow pathway that enables exploration and planning away from the prescription.

This work fits well into a wide body of work on *affective computing* (AC) [145, 163], with a growing focus on socio-cultural agents (e.g. [47]). In AC, emotions are usually framed following the rationalistic view proposed by Simon as "interrupts" to cognitive processing [179]. Emotions are typically inferred based on cognitive appraisals (e.g. a thwarted goal causes anger) that are used to guide action through a set of "coping" mechanisms. Gratch and Marsella [68] are possibly the first to propose a concrete computational

mechanism for coping. They propose a five stage process wherein beliefs, desires, plans and intentions are first formulated, and upon which emotional appraisals are computed. Coping strategies then use a set of *ad hoc* rules by modifying elements of the model such as probabilities and utilities, or by modifying plans or intentions. Si *et al.* [177] compute emotional appraisals from utility measures (including beliefs about other agent's utilities, as in an I-POMDP [65]), but they leave to future work *"how emotion affects the agents decision-making and belief update processes"* ([177] section 8). Goal prioritization using emotional appraisals have been investigated [8, 116, 128], as have normative multi-agent systems (NorMAS) [17]. There has been recent work on facial expressions in PD games, showing that they can significantly affect the outcomes [46].

Most approaches to emotional action guidance only give broad action guides in extreme situations, leaving all else to the cognitive faculties. *BayesAct* specifies one simple coping mechanism: minimizing inconsistency in continuous-valued sentiment. This, when combined with mappings describing how sentiments are appraised from events and actions, can be used to prescribe actions that maximally reduce inconsistency. These prescriptions are then used as guides for higher-level cognitive (including rational) processing and deliberation. *BayesAct* therefore provides an important step in the direction of building models that integrate "cognitive" and "affective" reasoning.

BayesAct requires anytime techniques for solving large continuous POMDPs with non-Gaussian beliefs. There has been much recent effort in solving continuous POMDPs with Gaussian beliefs (e.g. [48]), but these are usually in robotics motion planning where such approximations are reasonable. Point-based methods [172] have yielded solutions for continuous POMDPs for small domains [149, 16, 26], but they are not anytime and scalability is an issue, although recent work in parallel versions of point-based algorithms may lead to greater scalability [103]. Continuous Perseus, for example [149], is an approximate point-based algorithm for computing value functions (sets of alpha functions) for domains with continuous states. However, the value function itself must be closed under the Bellman backup operator to make the computation tractable, requiring a linear-Gaussian transition function (which we do not have). Even if a linearisation of the ACT equations was found, the explosion of the number of Gaussian mixtures requires contraction operators that further complicate the approximation. Other representations of alpha functions are as policy graphs [16], which does not work for continuous observations, or as decision trees [26], which I compared against in Section 3.6.4.

Recent proposals for large-scale POMDPs have included methods to leverage structure in multi-agent teams [7]. Guez *et al.* [72] present a variant of POMCP called BAMCP for solving Bayes-adaptive Markov decision processes (BAMDPs), which have continuous state, but deterministic (constant) dynamics. BAMDPs are POMDPs in which the continuous

state (the model parameters) remains constant. BAMCP works by selecting a single sample from the distribution over models at the start of a simulation, so it is not general enough to work for our POMDPs, which have continuous states but not constant dynamics.

Monte-Carlo tree search (MCTS) methods have seen more scalability success [27], and are anytime. POMCP [178] uses MCTS to efficiently solve POMDPs with continuous state spaces. By design, POMCP is unable to handle models with continuous action spaces, such as *BayesAct*. POMCoP uses POMCP to guide a sidekick's actions during a cooperative video game [124]. While this game has many similarities to CoRobots, it does not have continuous actions and restricts agent types to a small and countable set. POMCoP also uses an action bias, in this case it predicts the human's movements in the video game according to their type (this would be equivalent to our social coordination bias).

I conclude that MCTS methods are more appealing for *BayesAct* than other solvers because: (1) MCTS does not require a computation of the value function over the continuous state space and non-linear dynamics; (2) MCTS provides an anytime "quick and dirty" solution that corresponds naturally to our interpretation of the "fast thinking" heuristic.

3.8 Conclusion

This chapter studies decision-theoretic planning in a class of POMDP models of affective interactions, *BayesAct*, in which culturally shared sentiments are used to provide normative action guidance. *BayesAct* is an exciting development in artificial intelligence that combines affective computing, sociological theory, and probabilistic modeling. I use a Monte-Carlo Tree Search (MCTS) method to show how a simple and parsimonious model of human affect in decision making can yield solutions to two classic social dilemmas, robot navigation and assistive device design. I investigate how asymmetry between agent's resources can lead to manipulative or exploitative, yet socially aligned, strategies.

Chapter 4

Affective Response Generation for Neural Conversational Systems

The previous chapter explored how a probabilistic model of human emotions can be used to improve decision making in several applications. This chapter investigates how such affective computing techniques can be used in natural language processing. In particular, let's consider the task of text generation in dialogue systems: given an input prompt (e.g. 'How are you'), we want the machine learning model to generate an appropriate conversational response (e.g. 'I am well, what about you'). Thus, the goal is to study how to infuse human-like affect into dialogue response generation.

To this end, I start with proposing three basic affective strategies for dialogue response generation in this chapter. Subsequently, I investigate how to incorporate the ACT affect model from Chapter 3 into conversation models.

4.1 Introduction

As the field of natural language processing matures rapidly, the dialogue systems community is increasingly focusing on developing emotionally aware agents that exhibit human-like intelligence. Affectively cognizant conversational agents have been shown to provide companionship to humans [31, 151], help improve emotional wellbeing [62], give medical assistance in a more humane way [127], help students learn efficiently [97], and assist mental healthcare provision to alleviate bullying [67], suicide and depression [85, 190].

In a neural network-based dialogue system, discrete words are mapped to real-valued vectors, known as *embeddings*, capturing abstract meanings of words [131]; then an encoder-decoder framework—with long short term memory (LSTM)-based recurrent neural networks (RNNs)—generates a response conditioned on one or several previous utterances. Recent advances in this direction have demonstrated its efficacy for both task-oriented [201] and open-domain dialogue generation [110, 167, 171].

While most of the existing neural conversation models generate syntactically well-formed responses, they are prone to being short, dull, or vague. Previous efforts to address these issues include diverse decoding [195], diversity-promoting objective functions [105], human-in-the-loop reinforcement/active learning [13, 110] and content-introducing approaches [134, 210]. However, one shortcoming of these existing open-domain neural conversation models is the lack of *affect* modeling of natural language. These models, when trained over large dialogue datasets, do not capture the emotional states of the two humans interacting in the textual conversation, which are typically manifested through the choice of words or phrases. For instance, the attention mechanism in a sequence-to-sequence (Seq2Seq) model can learn syntactic alignment of words within the generated sequences [15]. Also, neural word embedding models like Word2Vec learn word vectors by context, and can preserve low-level word semantics (e.g., "king" − "male" ≈ "queen" − "woman"). However, emotional aspects are not explicitly captured by existing methods.

In this chapter, the research goal is to alleviate this issue in open-domain neural dialogue models by augmenting them with affective intelligence. I address this goal in four ways.

1. I embed words in a 3D affective space by retrieving word-level affective ratings from a cognitively engineered affective dictionary [198], where affectively similar constructs are close to one other. This dictionary is very similar to the ACT lexicon [75] used in Chapter 3. In this way, the ensuing neural model is aware of words' emotional features.

2. I augment the standard cross-entropy loss with affective objectives, so that the neural models are taught to generate more emotional utterances.

3. I inject affective diversity into the responses generated by the decoder through *affectively diverse* beam search algorithms, and thus the model actively searches for affective responses during decoding.

4. I condition the response generation process on the predictions of Affect Control Theory (ACT) [74], an external socio-mathematical model of affect. This allows the model to capture complex affective relationships between prompts and responses.

I also show that some of these emotional aspects can be combined to further improve the quality of generated responses in an open-domain dialogue system. Overall, in information-retrieval tasks like question-answering, the proposed models can help retain the users by interacting in a more human way.

4.2 Related Work

Most of the early affective dialogue systems were retrieval-based or slot-based, and used hand-crafted speech and text-based features [29, 73, 147]. More recently, with the advent of sophisticated and highly flexible neural network models [168, 171, 185, 196], the focus has shifted to building data-driven end-to-end dialogue models. Retrieval-based systems are still popular because they are more controllable, require less training data and are more efficient [67, 82, 230]. However, generative models dominate this space because they generalize well [154, 193]. This work falls in the latter category.

A large part of the affective dialogue literature treats *emotion* as a set of discrete categories, where each category corresponds to a type of biological response. For instance, some studies focus on producing *sentiment*-appropriate responses, where sentiment refers to *positive, negative* or *neutral* emotion [96, 176]. Other works use a larger set of discrete emotions [51, 63, 222, 229], based on the different psychological theories of emotion [52, 148]. A recent research trend, encouraged by social media growth, is to categorize emotions using the *emoji*[1] spectrum. This enables model training using massive weakly labelled datasets (e.g., from Twitter) [139, 209, 231]. For instance, Fung *et al.* [60] and Park [139] train emotion embeddings on tweets with hashtags and emojis as labels. These embeddings can be used downstream in other NLP tasks, such as dialogue systems. Different from all these strategies, I use a continuous, three dimensional representation of emotions. The three dimensions are Valence, Arousal and Dominance, and have been validated by several pioneering research studies in psychology [136, 161, 160]. In Chapter 3, I used a similar 3-factor model of emotions. Intuitively this makes sense; as humans we experience emotions as a continuum (i.e., a mixture of several feelings of varying intensity) rather than a single emotion of fixed intensity. Moreover, continuous emotion vectors fit well with dialogue models that are trained end-to-end.

At the time this research was conducted and published (February 2018), only two main related studies existed to the best of my knowledge [137]:

[1] An emoji is a symbol of emotional expression, such as a smiling/frowning face, a flower, etc.

- Affect Language Model [63, Affect-LM] is an LSTM-RNN language model which leverages the Linguistic Inquiry and Word Count [141, LIWC] text analysis program for affective feature extraction through keyword spotting. It considers binary affective features, namely *positive emotion*, *angry*, *sad*, *anxious*, and *negative emotion*. During inference, Affect-LM generates sentences conditioned on the input affect features and a learned parameter of affect strength.

  My work differs from Affect-LM in that I consider affective dialogue systems instead of merely language models, and I have explored more affective aspects including training and decoding.

- Emotional Chatting Machine [229, ECM] is a sequence-to-sequence model [185, Seq2Seq]. It takes as input a prompt and the desired emotion of the response, and produces a response. It has 8 emotion categories, namely *anger*, *disgust*, *fear*, *happiness*, *like*, *sadness*, *surprise*, and *other*. Additionally, ECM contains an internal memory and an external memory. The internal memory models the change of the internal emotion state of the decoder, and therefore encodes how much an emotion has already been expressed. The external memory decides whether to choose an emotional or generic (non-emotional) word at a given step during decoding.

  One drawback of ECM is that it requires, as input, the desired emotion category of the response. This emotion category, which is a discrete entity, has to be determined manually or through some rule-based heuristic. This setting is unrealistic in applications. My proposed approaches differ from ECM in that: 1) they intrinsically model emotion by affective word embeddings as input, as well as objective functions and inference criterion based on these embeddings; and 2) an external model of affect (ACT), which models emotions as continuous distributed vectors, is used to guide the generation process.

After the publication of this work, other Seq2Seq-based affective conversational models have been proposed. Dryjański *et al.* [51] inject predefined sentiment to a neutral utterance by inferring the phrases and their insertion points. Lubis *et al.* [123] jointly train a Seq2Seq model and an emotion encoder. The emotion encoder maintains the emotional context during a conversation, and is trained using the SEMAINE dataset (2000 samples) [130] where utterances are labeled on the valence and arousal axes. Vadehra [193] train Seq2Seq with an adversarial objective to remove affect from the learned representation of the input utterance, and generate the response based on this representation and the target affect label (one of seven discrete emotion categories). Rashkin *et al.* [154] have released *EmpatheticDialogues*, a dataset of 25000 conversations grounded in emotional situ-

ations to facilitate training and evaluation of dialogue systems. They show that finetuning existing dialogue models on this dataset boosts their affective quality significantly.

Conditional Variational Autoencoders (CVAEs) [180] have become another popular choice for neural dialogue models. Vanilla Seq2Seq models are prone to generating generic responses that are not very diverse. One reason is that the encoder learns sentence representations as isolated points in the latent space. Thus, the input representation is fixed and stochastic variations occur only at the word level during decoding, resulting in short term rewards. CVAEs address this problem by imposing a prior distribution on the latent space. In a CVAE, the latent representations are densely packed within a region (dictated by the prior). This makes the latent space continuous, and it becomes possible to sample vectors from it, which can be decoded into diverse sentences. The stochasticity of the sampling action allows us to control the generation process, because the latent sample can encode global sentence properties (e.g. topic, sentiment or affect) and long-term structure. Serban *et al.* [168] were the first to introduce CVAEs to dialogue generation, where the generative process is conditioned on conversation history. Zhao *et al.* [227] further condition the model on dialogue intents (also called dialogue act labels). Park *et al.* [140] create a hierarchy of conversation-level and utterance-level latent variables to control both global and local conversation properties.

CVAEs have recently been used for affect-controlled dialogue generation, where the model is conditioned on positive-negative-neutral sentiment tags [174] or more fine-grained emotion categories [222]. Kong *et al.* [96] use an adversarial approach for sentiment control which can be applied to CVAEs too. In this work, one of the affective strategies I propose is a CVAE-based affective neural dialogue model; it is inspired from Shen *et al* [174]'s and Zhang *et al.* [222]'s studies, but leverages Affect Control Theory as an external model of affect for conditional response generation.

4.3 The Proposed Affective Approaches

In this section, I propose affective neural response generation, which augments traditional neural conversation models with emotional cognizance. Concretely:

- In Sections 4.3.1–4.3.3, I leverage a cognitively engineered dictionary to propose three strategies for affective response generation, namely affective word embeddings as input, affective training objectives, and affectively diverse beam search. Figure 4.1 delineates an overall picture of these approaches. As will be shown later, these affective strategies can be combined to further improve Seq2Seq dialogue systems.

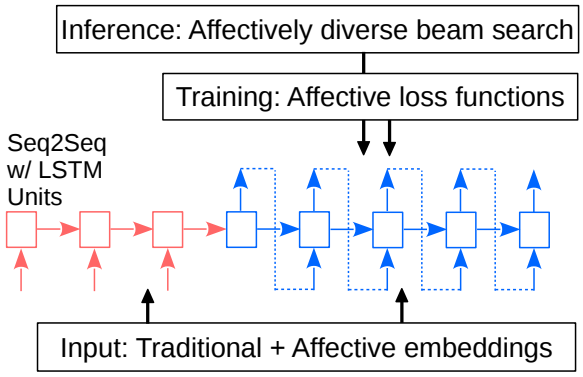


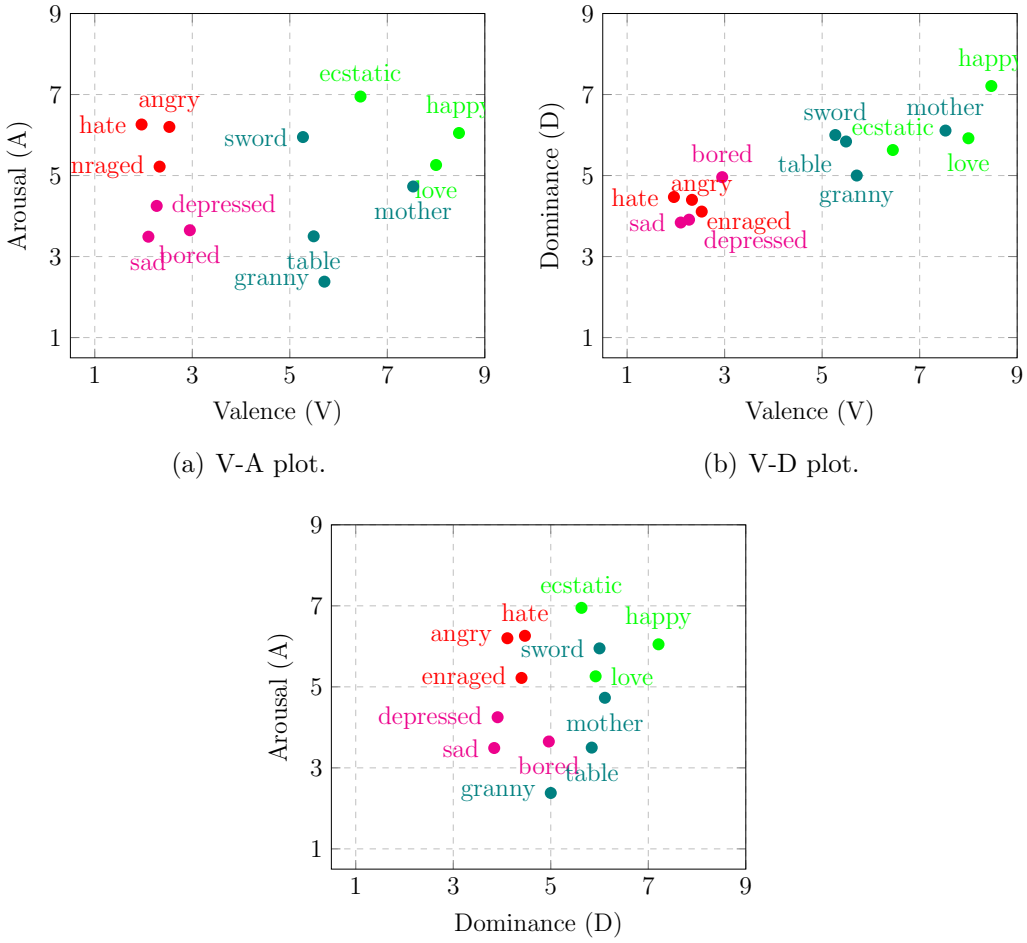
Figure 4.1: Overview of the three proposed affective strategies for the input, training, and inference of Seq2Seq based on a cognitively engineered dictionary with Valence, Arousal, and Dominance (VAD) scores.

- In Section 4.3.4, I present the CVAE-based ACT conversational model. Here, the response generation is conditioned on the ACT predictions. A high-level overview is shown in Figure 4.3.

4.3.1 Affective Word Embeddings

As said, traditional word embeddings trained with co-occurrence statistics are insufficient to capture affect aspects. I propose to augment traditional word embeddings with a 3D affective space by using an external cognitively-engineered affective dictionary [198].[2] The dictionary I use consists of 13,915 lemmatized English words, each of which is rated on three traditionally accepted continuous and real-valued dimensions of emotion: Valence (V, the pleasantness of a stimulus), Arousal (A, the intensity of emotion produced by a stimulus), and Dominance (D, the degree of power exerted by a stimulus). This VAD space is congruent with the EPA (Evaluation-Potency-Activity) space we saw in Chapter 3. Recall that sociologists hypothesize the VAD/EPA space structures the semantic relations of linguistic concepts across languages and cultures. VAD ratings have been previously used in sentiment analysis, sarcasm detection and empathetic tutors, among other affective computing applications [153, 158, 197]. To the best of my knowledge, I am the first to introduce VAD to dialogue systems in [12].

[2] Available for free at http://crr.ugent.be/archives/1003



(a) V-A plot.

(b) V-D plot.

(c) D-A plot.

Figure 4.2: Relationship between several adjectives, nouns, and verbs on 3-D VAD scale.

The scale of each dimension in the VAD space is from 1 to 9, where a higher value corresponds to higher valence, arousal, or dominance. Thus, V $\simeq 1, 5$ and 9 corresponds to a word being very negative (*pedophile*), neutral (*tablecloth*) and very positive (*happiness*), respectively. This axis is traditionally used on its own in most sentiment analysis techniques. Similarly, A $\simeq 1, 5$ and 9 corresponds to a word having very low (*dull*), moderate (*watchdog*), and very high (*insanity*) emotional intensity, respectively. Finally, D $\simeq 1, 5$ and 9 corresponds to a word that is very powerless (*dementia*), neutral (*waterfall*) and very powerful (*paradise*), respectively. The VAD ratings of each word were collected through a survey in [198] over 1800 participants. I directly take them as the 3-dimensional word-level affective embeddings.

Some examples of words (including nouns, adjectives, and verbs) and their corresponding VAD values are depicted in Figure 4.2. For instance, the VAD vectors of the words *ecstatic* and *bored* are $[6.45, 6.95, 5.63]$ and $[2.95, 3.65, 4.96]$, respectively. This means that an average human rates the feeling of being *bored* as more unpleasant (V), less intense (A), and slightly weaker (D), compared with the feeling of being *ecstatic*. Similarly, the VAD vectors for the nouns *mother* and *granny* are $[7.53, 4.73, 6.11]$ and $[5.71, 2.38, 5.00]$, respectively. Thus, mothers are perceived to be more pleasant (V) and more powerful (D) than grannies, and evoke more intense emotions (A). From Figure 4.2a, we also see some clusters {*angry, hate, enraged*} and {*depressed, sad, bored*} that are slightly apart on the A axis. Also, the cluster {*sword, table, granny*} is fairly neutral on the V axis, compared with the cluster {*happy, mother, love*} in Figure 4.2b.

For words missing in this dictionary, such as stop words and proper nouns, I set the VAD vector to be the neutral vector $\vec{\boldsymbol{\eta}} = [5, 1, 5]$, because these words are neutral in pleasantness (V) and power (D), and evoke no arousal (A). Formally, I define "*word to affective vector*" (`W2AV`) as:

$$\texttt{W2AV}(w) = \begin{cases} \text{VAD}(l(w)), & \text{if } l(w) \in dict \\ \vec{\boldsymbol{\eta}} = [5, 1, 5], & \text{otherwise} \end{cases} \tag{4.1}$$

where $l(w)$ is the lemmatization of the word w. In this way, words depicting similar emotions are close together in the affective space, and affectively dissimilar words are far apart from each other. Thus `W2AV` is suitable for neural processing.

The simplest approach to utilize `W2AV` is to feed it to a Seq2Seq model as input. Concretely, I concatenate the `W2AV` embeddings of each word with its traditional word embeddings, the resulting vector being the input to both the encoder and the decoder.

4.3.2 Affective Loss Functions

Equipped with affective vectors, I further design affective training loss functions to explicitly train an affect-aware Seq2Seq conversation model. The philosophy of manipulating loss function is similar to [105], but I focus on affective aspects (instead of diversity in general).

Recall that, given a message-response pair $(\boldsymbol{X}, \boldsymbol{Y})$, where $\boldsymbol{X} = \boldsymbol{x}_1, \cdots, \boldsymbol{x}_m$ and $\boldsymbol{Y} = \boldsymbol{y}_1, \cdots, \boldsymbol{y}_n$ are sequences of words, Seq2Seq models (parameterized by $\boldsymbol{\theta}$) are typically trained with cross entropy loss (XENT):

$$L_{\texttt{XENT}}(\boldsymbol{\theta}) = -\log p(\boldsymbol{Y}|\boldsymbol{X}) = -\sum_{i=1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{X}), \tag{4.2}$$

where $\boldsymbol{\theta}$ denotes model parameters.

I propose several affective heuristics as follows.

Minimizing Affective Dissonance. I start with the simplest approach: maintaining affective consistency between prompts and responses. This heuristic arises from the observation that typical open-domain textual conversations between two humans consist of messages and responses that, in addition to being affectively loaded, are affectively similar to each other. For instance, a friendly message typically elicits a friendly response and provocation usually results in anger or contempt. Assuming that the general affective tone of a conversation does not fluctuate too suddenly and too frequently, I emulate human-human interactions in my model by minimizing the *dissonance* between the prompts and the responses, i.e. the Euclidean distance between their affective embeddings. This objective allows the model to generate responses that are emotionally aligned with the prompts. Thus, at time step i, the loss is computed by

$$L^i_{\texttt{DMIN}}(\boldsymbol{\theta}) = -(1-\lambda)\log p(\boldsymbol{y}_i|\boldsymbol{y}_1,\cdots,\boldsymbol{y}_{i-1},\boldsymbol{X}) + \lambda\ \hat{p}(\boldsymbol{y}_i)\left\| \sum_{j=1}^{|\boldsymbol{X}|}\frac{\texttt{W2AV}(\boldsymbol{x}_j)}{|\boldsymbol{X}|} - \sum_{k=1}^{i}\frac{\texttt{W2AV}(\boldsymbol{y}_k)}{i}\right\|_2 \tag{4.3}$$

where $\|\cdot\|_2$ denotes ℓ_2-norm. The first term is the standard XENT loss as in Equation 4.2. The sum $\sum_j \frac{\texttt{W2AV}(\boldsymbol{x}_j)}{|\boldsymbol{X}|}$ is the average affect vector of the source sentence, whereas $\sum_k \frac{\texttt{W2AV}(\boldsymbol{y}_k)}{i}$ is the average affect vector of the target sub-sentence generated up to the current time step i.

In other words, I penalize the distance between the average affective embeddings of the source and the target sentences. Notice that this affect distance is not learnable and that selecting a single predicted word makes the model indifferentiable. Therefore, I relax hard prediction of a word by its predicted probability $\hat{p}(\boldsymbol{y}_i)$. λ is a hyperparameter balancing the two factors.

Maximizing Affective Dissonance. Admittedly, minimizing the affective dissonance does not always make sense while we model a conversation. An over-friendly message from a stranger may elicit anger or disgust from the recipient. Furthermore, responses that are *not* too affectively aligned with the prompts may be perceived as more interesting, by virtue of being less predictable. Thus, I design an objective function $L_{\texttt{DMAX}}$ that *maximizes* the dissonance by flipping the sign in the second term in Equation 4.3.

$$L^i_{\texttt{DMAX}}(\boldsymbol{\theta}) = -(1-\lambda)\log p(\boldsymbol{y}_i|\boldsymbol{y}_1,\cdots,\boldsymbol{y}_{i-1},\boldsymbol{X}) - \lambda\ \hat{p}(\boldsymbol{y}_i)\left\| \sum_{j=1}^{|\boldsymbol{X}|}\frac{\texttt{W2AV}(\boldsymbol{x}_j)}{|\boldsymbol{X}|} - \sum_{k=1}^{i}\frac{\texttt{W2AV}(\boldsymbol{y}_k)}{i}\right\|_2 \tag{4.4}$$

Maximizing Affective Content. The third heuristic encourages Seq2Seq to generate affective content, but does not specify the polarity of sentiment. This explores the hypothesis that most of the casual human responses are not dull or emotionally neutral. The model has the liberty to choose the appropriate sentiment. Concretely, I maximize the affective content of the model's responses, so that it avoids generating generic responses like "*yes*," "*no*," "*I don't know*," and "*I'm not sure*." That is, at the time step i, the loss function is

$$L^i_{\texttt{AC}}(\boldsymbol{\theta}) = -\,(1-\lambda)\log p(\boldsymbol{y}_i|\boldsymbol{y}_1,\cdots,\boldsymbol{y}_{i-1},\boldsymbol{X}) - \lambda\,\hat{p}(\boldsymbol{y}_i)\left\|\texttt{W2AV}(\boldsymbol{y}_i) - \vec{\boldsymbol{\eta}}\right\|_2 \tag{4.5}$$

The second term is a regularizer that discourages non-affective words. I penalize the distance between $\boldsymbol{y}_i$'s affective embedding and the affectively neutral vector $\vec{\boldsymbol{\eta}} = [5, 1, 5]$, so the model pro-actively chooses emotionally rich words.

4.3.3 Affectively Diverse Decoding

In this subsection, I propose affectively diverse decoding that incorporates affect into the decoding process of neural response generation.

Traditionally, beam search (BS) has been used for decoding in Seq2Seq models because it provides a tractable approximation of searching an exponentially large solution space. However, in the context of open-domain dialogue generation, BS is known to produce nearly identical samples like "*This is great!*" and "*This is so great!*", that lack syntactic diversity [64]. Diverse beam search (DBS) [195] is a recently proposed variant of BS that explicitly considers diversity during decoding; it has been shown to outperform BS and other diverse decoding techniques in many NLP tasks.

Below, I describe BS, DBS, and the proposed affective variants of DBS.

Beam Search (BS). BS maintains top-B most likely (sub)sequences, where B is known as the *beam size*. At each time step t, the top-B subsequences at time step $t-1$ are augmented with all possible actions available; then the top-B most likely branches are retained at time t, and the rest are pruned.

Let V be the set of vocabulary tokens and let $\boldsymbol{X}$ be the input sequence. Ideally, decoding of an entire sequence $\boldsymbol{Y}^*$ is given by

$$\boldsymbol{Y}^* = \boldsymbol{y}^*_1,\cdots,\boldsymbol{y}^*_T = \underset{\boldsymbol{y}_1,\cdots,\boldsymbol{y}_T}{\arg\max}\left[\sum_{t\in T}\log p(\boldsymbol{y}_t|\boldsymbol{y}_{t-1},\cdots,\boldsymbol{y}_1,\boldsymbol{X})\right] \tag{4.6}$$

where T is the length. BS approximates Equation 4.6 by computing and storing only the top-B high scoring (sub)sequences (called beams) at each time step. Let $\boldsymbol{y}_{i,[t-1]}$ be the ith beam stored at time $t-1$, and $\boldsymbol{Y}_{[t-1]} = \{\boldsymbol{y}_{1,[t-1]}, \cdots, \boldsymbol{y}_{B,[t-1]}\}$ be the set of beams stored by BS at time $t-1$. Then at time t, the BS objective is

$$\boldsymbol{Y}_{[t]} = \boldsymbol{y}^{1^*}_{1..t}, \cdots, \boldsymbol{y}^{B^*}_{1..t} = \underset{\substack{\boldsymbol{y}_{1,[t]},\cdots,\boldsymbol{y}_{B,[t]} \\ \in \boldsymbol{Y}_{[t-1]} \times V}}{\arg\max} \sum_{b=1}^{B} \sum_{i=1}^{t} \log p(\boldsymbol{y}_{b,i} | \boldsymbol{y}_{b,[i-1]}, \boldsymbol{X}) \tag{4.7}$$

subject to $\boldsymbol{y}_{i,[t]} \neq \boldsymbol{y}_{j,[t]}$, where $\boldsymbol{Y}_{[t-1]} \times V$ is the set of all possible extensions based on the beams stored at time $t-1$.

Diverse Beam Search (DBS). DBS aims to overcome the diversity problem in BS by incorporating diversity among candidate outputs. It divides the top-B beams into G groups (each group containing $B' = G/B$ beams) and incorporates diversity between these groups by maximizing the standard likelihood term as well as a dissimilarity metric among the groups.

Concretely, DBS adds to traditional BS (Eq 4.7) a dissimilarity term $\Delta(\boldsymbol{Y}^1_{[t]}, \cdots, \boldsymbol{Y}^{g-1}_{[t]})[\boldsymbol{y}_t]$ which measures the dissimilarity between group g and previous groups $1, \cdots, g-1$ if token $\boldsymbol{y}_t$ is selected to extend any beam in group g. This is given by

$$\boldsymbol{Y}^g_{[t]} = \underset{\substack{\boldsymbol{y}^g_{1,[t]},\cdots,\boldsymbol{y}^g_{B',[t]} \\ \in \boldsymbol{Y}^g_{[t-1]} \times V}}{\arg\max} \sum_{b=1}^{B'} \sum_{i=1}^{t} \log p(\boldsymbol{y}^g_{b,i} | \boldsymbol{y}^g_{b,[i-1]}, \boldsymbol{X}) + \lambda_g \Delta(\boldsymbol{Y}^1_{[t]}, \cdots, \boldsymbol{Y}^{g-1}_{[t]})[\boldsymbol{y}^g_{b,t}] \tag{4.8}$$

subject to $\boldsymbol{y}^g_{i,[t]} \neq \boldsymbol{y}^g_{j,[t]}$, where $\lambda_g \geq 0$ is a hyperparameter controlling the diversity strength. Intuitively, DBS modifies the probability in BS as a general scoring function by adding a dissimilar term between a particular sample (i.e., $\boldsymbol{y}^g_{b,1} \cdots \boldsymbol{y}^g_{b,t}$) and samples in other groups (i.e., $\boldsymbol{Y}^1_{[t]}, \cdots, \boldsymbol{Y}^{g-1}_{[t]}$). I refer readers to [195] for the details of DBS. Here, I focus on the dissimilarity metric that can incorporate affective aspects into the decoding phase.

Affectively Diverse Beam Search (ADBS). The dissimilarity metric for DBS can take many forms as used in [195]: Hamming diversity that penalizes tokens based on the number of times they are selected in the previous groups, n-gram diversity that discourages repetition of n-grams between groups, and neural-embedding diversity that penalizes words with similar embeddings across groups. Among these, the neural-embedding diversity metric is the most relevant to us. When used with Word2Vec embeddings, this metric

discourages semantically similar words (e.g., synonyms) to be selected across different groups.

To decode affectively diverse samples, I propose to inject affective dissimilarity across the beam groups based on affective word embeddings. This can be done either at the word level or sentence level. I formalize these notions below.

• *Word-Level Diversity for ADBS (WL-ADBS).* I define the word-level affect dissimilarity metric Δ_W to be

$$\Delta_W(\boldsymbol{Y}_{[t]}^1, \cdots, \boldsymbol{Y}_{[t]}^{g-1})[\boldsymbol{y}_{b,t}^g] = -\sum_{j=1}^{g-1}\sum_{c=1}^{B'} \text{sim}\left(\texttt{W2AV}(\boldsymbol{y}_{b,t}^g), \texttt{W2AV}(\boldsymbol{y}_{c,t}^j)\right) \tag{4.9}$$

where $\text{sim}(\cdot)$ denotes a similarity measure between two vectors. In my experiments, I use the cosine similarity function. $\boldsymbol{y}_{b,t}^g$ denotes the token under consideration at the current time step t for beam b in group g, and $\boldsymbol{y}_{c,t}^j$ denotes the token chosen for beam c in a previous group j at time t.

Intuitively, this metric computes the cosine similarity of group g's beam b with all the beams generated in groups $1, \cdots, g-1$. The metric operates at the word level, ensuring that the word affect at time t is diversified across groups.

• *Sentence-Level Diversity for ADBS (SL-ADBS).* The word-level metric Δ_W in Equation 4.9 does not take into account the overall sentence affect for each group. I propose an alternative sentence-level affect diversity metric, given by

$$\Delta_S(\boldsymbol{Y}_{[t]}^1, \cdots, \boldsymbol{Y}_{[t]}^{g-1})[\boldsymbol{y}_{b,t}^g] = -\sum_{j=1}^{g-1}\sum_{c=1}^{B'} \text{sim}\left(\Psi(\boldsymbol{y}_{b,[t]}^g), \Psi(\boldsymbol{y}_{c,[t]}^j)\right) \tag{4.10}$$

$$\text{where} \qquad \Psi(\boldsymbol{y}_{i,[t]}^k) = \sum_{w \in \boldsymbol{y}_{i,[t]}^k} \texttt{W2AV}(w) \tag{4.11}$$

Here, $\boldsymbol{y}_{i,[t]}^k$ for $k \leq g$ is the ith beam in the kth group stored at time t; $\boldsymbol{y}_{b,[t]}^g$ is the concatenation of $\boldsymbol{y}_{b,[t-1]}^g$ and $\boldsymbol{y}_{b,t}^g$. Intuitively, this metric computes the *cumulative dissimilarity* (given by the function $\Psi(\cdot)$) between the current beam and all the previously generated beams in other groups. This bag-of-affective-words approach is simple but works well in practice, as will be shown later.

It should be also noticed that several other beam search-based diverse decoding techniques have been proposed in recent years, including DivMBest [64], MMI objective [105] and segment-by-segment re-ranking [173]. All of them use the notion of a *diversity term* within BS; therefore my affect-injecting technique can be used with these algorithms.

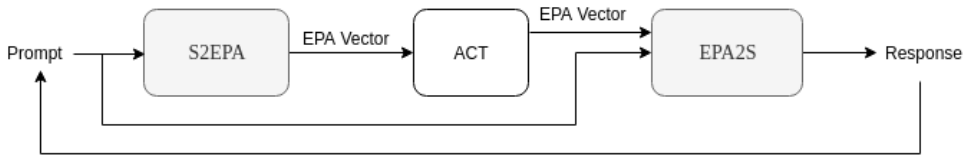


Figure 4.3: Pipeline to integrate Affect Control Theory (ACT) into a dialogue system.

4.3.4 Affect Control Theory for Dialogue Generation

Recall that ACT [74] is a model of affective interactions between two entities, typically a human and an artificial agent. Given the affective *identity* of each of the two interactants, ACT predicts (fairly accurately) how the interaction should proceed on an emotional level. Thus, ACT lends itself naturally to the dialogue system setting.

In ACT, each action is a 3-dimensional vector on the EPA scale (see Section 3.2), which conveys the affect of the action. This is different from a dialogue system, where an action is typically a sentence that conveys an affect as well as a proposition. For instance, the sentences "*Could you please make me some tea*" and *"Go make me some tea"* convey the same propositional action (asking for tea) but their affect is vastly different. The former can be seen as a request or appeal (EPA:$\{0.76, 0.34, 0.04\}$), whereas the latter is more of a command (EPA:$\{-0.32, 2.06, 0.93\}$). Therefore, to build a viable dialogue system using ACT, we need a mechanism to map EPA actions to dialogue actions, and vice versa.

An overview of the ACT conversational model is shown in Figure 4.3. ACT is instantiated with two affective identities, one each for the human participant and the artificial agent. Given an input prompt (a sentence) by the human user, a sentence-to-EPA (`S2EPA`) function maps the prompt to an EPA vector, such that the vector appropriately conveys the affect of the input sentence. This EPA vector acts as the affective action by the user. ACT is queried with this vector, and produces the response EPA vector (i.e.the affective action taken by the artificial agent). An EPA-to-sentence (`EPA2S`) function uses this response EPA, as well as the input prompt, to generate a response that is semantically relevant to the input. This response can be treated as the next prompt in the conversation, and the process continues.

We now define the `S2EPA` and `EPA2S` functions shown in Figure 4.3.

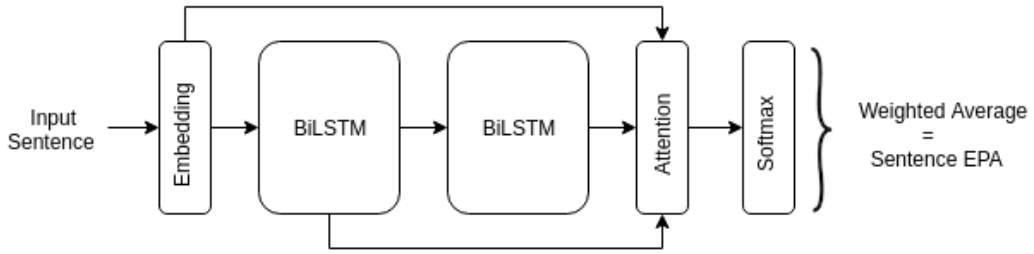


Figure 4.4: `S2EPA`: A pretrained BiLSTM network with attention [56], tweaked to produce EPA vectors instead of emojis.

Sentence to EPA (`S2EPA`)

The goal of the `S2EPA` function is to generate an EPA representation of a given sentence. If we had access to a large amount of sentences labeled with EPAs, we could simply train a recurrent neural network to approximate the sentence-to-EPA mapping. However, building such a dataset is time-consuming and expensive. To get around this issue, I use a pre-trained publicly available sentence-to-emoji model and tweak its output to suit our needs.

Concretely, I use DeepMoji, a pre-trained BiLSTM network with attention [56][3]. This model has been trained on a dataset of 1.2 billion tweets labeled with emojis. Given an input sentence, the model produces a probability distribution over 64 emojis. I use this model to our advantage as follows. I ask two human annotators to label each of these 64 emojis with EPA vectors. These annotations are averaged to get a single EPA vector per emoji, which is assigned to that emoji. Then, given an input query, I take the weighted average of the 64 EPA vectors, where the weights are produced by the softmax layer. This gives the desired sentence to EPA mapping. The architecture of `S2EPA` is shown in Figure 4.4.

EPA to Sentence (`EPA2S`)

The goal of the `EPA2S` function is to generate a response sentence, given the input prompt and a target EPA vector, such that the response conveys the same affect as the target EPA. To build `EPA2S`, I explore two methods, Seq2Seq and CVAE.

[3]The pretrained DeepMoji model is publicly available at https://github.com/bfelbo/DeepMoji.

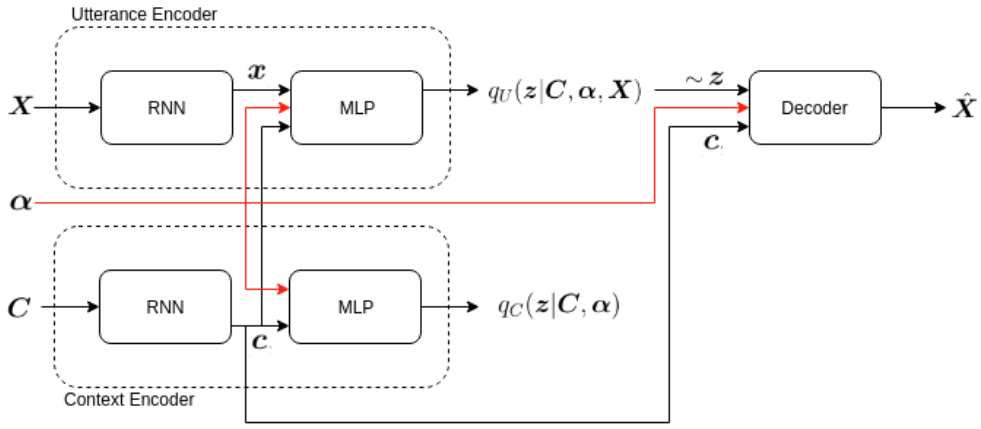


Figure 4.5: CVAE training architecture.

`EPA2S-Seq2Seq`

One straightforward `EPA2S` model is Seq2Seq with attention, where the input sentence is concatenated with the target EPA and passed into the encoder, which gives us a fixed-length context vector. Given this context vector, the decoder sequentially produces the response while attending to the encoder hidden states.

`EPA2S-CVAE`

CVAE is another viable model for `EPA2S`. It is described below.

Let $(\boldsymbol{C}, \boldsymbol{\alpha}, \boldsymbol{X})$ denote a training sample, where $\boldsymbol{C}$ and $\boldsymbol{X}$ are sequences of tokens denoting the prompt and the response respectively, and $\boldsymbol{\alpha}$ is an EPA vector denoting the desired affect of the response. The CVAE consists of a *context encoder, utterance encoder* and a *decoder*. The context encoder uses an RNN to map $\boldsymbol{C}$ to a fixed-length vector $\boldsymbol{c}$, and then passes $(\boldsymbol{c}, \boldsymbol{\alpha})$ to an MLP, which outputs the parameters of the probability distribution $q_C(\boldsymbol{z}|\boldsymbol{C}, \boldsymbol{\alpha}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\lambda}^2\boldsymbol{I})$; this distribution is called the prior. Similarly, the utterance encoder uses an RNN to map $\boldsymbol{X}$ to a fixed-length vector $\boldsymbol{x}$, and then passes $(\boldsymbol{c}, \boldsymbol{\alpha}, \boldsymbol{x})$ to an MLP that outputs the parameters of the probability distribution $q_U(\boldsymbol{z}|\boldsymbol{C}, \boldsymbol{\alpha}, \boldsymbol{X}) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\lambda}}^2\boldsymbol{I})$. This is the posterior. A latent vector $\boldsymbol{z}$ is then sampled from q_U. The decoder RNN parameterizes the distribution $q_D(\boldsymbol{X}|\boldsymbol{z}, \boldsymbol{C}, \boldsymbol{\alpha})$; it takes $(\boldsymbol{z}, \boldsymbol{c}, \boldsymbol{\alpha})$ as input and produces a distribution over the response sequences. The CVAE objective is to maximize the reconstruction probability of $\boldsymbol{X}$, and minimize the KL divergence between the prior

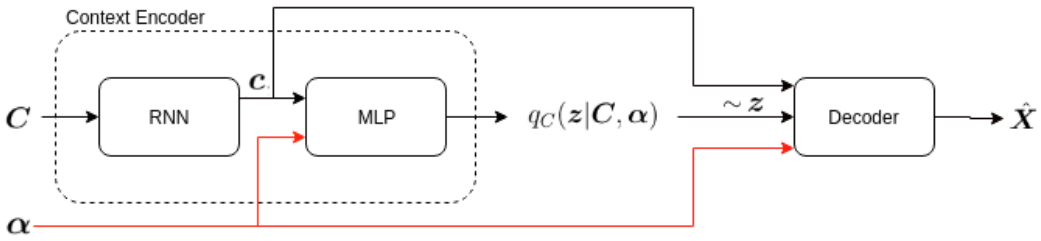


Figure 4.6: CVAE at inference time: this is the `EPA2S` module.

q_C and the posterior q_U. This is given by

$$L_{\texttt{CVAE}}(\boldsymbol{\theta}_C, \boldsymbol{\theta}_U, \boldsymbol{\theta}_D; \boldsymbol{C}, \boldsymbol{X}, \boldsymbol{\alpha}) = \mathrm{KL}\big(q_U(\boldsymbol{z}|\boldsymbol{C}, \boldsymbol{\alpha}, \boldsymbol{X}) \big\| q_C(\boldsymbol{z}|\boldsymbol{C}, \boldsymbol{\alpha})\big) - \mathbb{E}_{q_U}\left[\log q_D(\boldsymbol{X}|\boldsymbol{z}, \boldsymbol{C}, \boldsymbol{\alpha})\right] \tag{4.12}$$

where $\boldsymbol{\theta}_C, \boldsymbol{\theta}_U$ and $\boldsymbol{\theta}_D$ denote the parameters of the context encoder, the utterance encoder, and the decoder respectively. This training process is depicted in Figure 4.5.

At inference time, the goal is to generate a response given an input sentence $\boldsymbol{C}$ and a target EPA $\boldsymbol{\alpha}$. $(\boldsymbol{C}, \boldsymbol{\alpha})$ are passed through the context encoder, and a latent variable $\boldsymbol{z}$ is sampled from q_C. Then $(\boldsymbol{z}, \boldsymbol{c}, \boldsymbol{\alpha})$ are passed to the decoder to generate a response. This process is depicted in Figure 4.6.

4.4 Experiments

4.4.1 Data and Setup

I evaluate all the approaches on the Cornell Movie Dialogs Corpus[4] [42], which contains 220k utterance-response pairs from movie conversations. I split the data into 200k, 10k and 10k samples for training, validation and testing.

Seq2Seq Models

All the Seq2Seq variants use a single-layer LSTM encoder and a single-layer LSTM decoder, each layer containing 1024 cells. I set the vocabulary size to 12,000 and used Adam [92] optimizer with default hyper-parameters.

[4]The dataset is publicly available at https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html.

Listed below are detailed settings for each model.

- For the baseline $L_{\texttt{XENT}}$ loss, I use 1024-dimensional Word2Vec embeddings as input and train the Seq2Seq model for 50 epochs by using Equation 4.2.
- For the affective embeddings as input, I use 1027-dimensional vectors, each a concatenation of 1024-D Word2Vec and 3-D `W2AV` embeddings. Training is done for 50 epochs.
- For each of the affective loss functions ($L_{\texttt{AC}}$, $L_{\texttt{DMIN}}$, and $L_{\texttt{DMAX}}$), I train the model using $L_{\texttt{XENT}}$ loss for 40 epochs. followed by 10 epochs using the affective loss functions.
- The ADBS decoding is deployed at test time (both word-level and sentence-level metrics, Δ_W and Δ_S in Equations 4.9 and 4.10, respectively). I set $G = B$ for simplicity, that is, each group contains a single beam. Thus, diversification among groups in my case is equivalent to diversification among all the beams.
- The λ hyperparameters for $L_{\texttt{DMIN}}$, $L_{\texttt{DMAX}}$, and $L_{\texttt{AC}}$ are manually tuned through validation and set to 0.5, 0.4, and 0.5, respectively. For affectively diverse BS, λ is set to 0.7 (Equation 4.8).
- For `EPA2S-Seq2Seq`, the embedding layer is initialized with 1024 dimensional Word2Vec embeddings. $\boldsymbol{\alpha}$ is 3-dimensional. Training is done for 50 epochs.

CVAE Model

For the CVAE model, each encoder contains 1) a single-layer BiLSTM, each direction containing 1024 LSTM cells, and 2) a two-layer MLP. The CVAE decoder is a single-layer LSTM network of 1024 cells. The variables $\boldsymbol{z}$ and $\boldsymbol{\alpha}$ are 1024-dimensional and 3-dimensional respectively. The embedding layer is initialized with 1024-dimensional Word2Vec embeddings. To include $\boldsymbol{\alpha}$ in each training sample $(\boldsymbol{C}, \boldsymbol{X})$, I compute

$$\boldsymbol{\alpha} = \text{ACT}(\texttt{S2EPA}(C)) \tag{4.13}$$

For the CVAE model, I follow Kingma *et al.* [93]; the reconstruction loss is computed with a single sample from q_C, and the KL divergence is computed in closed form. Furthermore, to prevent the *degenerate* case where the KL divergence is equal to zero, I use KL annealing, following Bowman *et al.* [25]. Degeneracy occurs when the network sets the

Model	Syntactic Coherence	Natural	Emotional Approp.
Word Emb. (baseline)	1.48	0.69	0.41
Word + Affective Emb.	**1.71** ↑	**1.05** ↑	**1.01** ↑

Table 4.1: The effect of affective word embeddings as input.

Model	Syntactic Coherence	Naturalness	Emotional Approp.
$L_{\texttt{XENT}}$ (baseline)	1.48	0.69	0.41
$L_{\texttt{DMIN}}$	**1.75** ↑	0.83 ↑	0.56 ↓
$L_{\texttt{DMAX}}$	1.74 ↑	0.85 ↑	0.58 ↑
$L_{\texttt{AC}}$	1.71 ↑	**0.95** ↑	**0.71** ↑

Table 4.2: The effect of affective loss functions.

posterior q_U to be equal to the prior q_C, implying that the network ignores the latent variable. This is sometimes referred to as the *vanishing latent variable problem*. KL annealing circumvents this issue by adding a weight to the KL term during training. In the beginning, this weight is zero, so the network encodes useful information in $\boldsymbol{z}$ without worrying about staying close to the prior. As training progresses, the weight is slowly increased till it reaches one.

4.4.2 Results

Recent work employs both automated metrics (e.g., BLEU, ROUGE, and METEOR) and human judgments to evaluate dialogue systems. While automated metrics enable high-throughput evaluation, they have weak or no correlation with human judgments [117]. It is also unclear how to evaluate affective aspects by automated metrics. Therefore, I recruit 3-5 human judges to evaluate all the proposed models, following several previous studies [134, 171].

To evaluate the quality of the generated responses, 5 workers are asked to evaluate 100 test samples for each model variant in terms of *syntactic coherence* (Does the response make grammatical sense?), *naturalness* (Could the response have been plausibly produced by a human?) and *emotional appropriateness* (Is the response emotionally suitable for the prompt?). For each axis, the judges are asked to assign each response an integer score of 0 (bad), 1 (satisfactory), or 2 (good). The scores are then averaged for each axis (Tables 4.1, 4.2 and 4.4). I evaluate the inter-annotator consistency by Fleiss' κ score [58], and

Model	Syntactic Diversity	Affective Diversity	# Emotionally Approp. Responses
BS (baseline)	1.23	0.87	0.89
H-DBS	1.47 ↑	0.79 ↓	0.78 ↓
WL-ADBS	**1.51** ↑	1.25 ↑	1.30 ↑
SL-ADBS	1.45 ↑	**1.31** ↑	**1.33** ↑

Table 4.3: Effect of affectively diverse decoding. H-DBS refers to Hamming-based DBS used in [195]. WL-ADBS and SL-ADBS are the proposed word-level and sentence-level affectively diverse beam search, respectively.

Model	Syntactic Coherence	Natural-ness	Emotional Approp.
Traditional Seq2Seq (baseline)	1.48	0.69	0.41
ACT with `S2EPA` & `EPA2S-Seq2Seq` (*friend-friend*)	1.59 ↓	0.73 ↓	0.39 ↓
ACT with `S2EPA` & `EPA2S-CVAE` (*friend-friend*)	1.57 ↓	0.68 ↓	0.47 ↓
ACT with `S2EPA` & `EPA2S-Seq2Seq` (*friend-enemy*)	1.61 ↓	0.62 ↓	0.34 ↓
ACT with `S2EPA` & `EPA2S-CVAE` (*friend-enemy*)	1.33 ↓	0.75 ↓	0.50 ↓

Table 4.4: Comparing the different ACT conversation models.

Model	Syntactic Coherence	Natural-ness	Emotional Approp.
Traditional Seq2Seq (baseline)	1.48	0.69	0.41
Seq2Seq + Affective Embeddings	1.71 ↑	1.05 ↑	1.01 ↑
Seq2Seq + Affective Emb. & Loss	**1.76** ↓	1.03 ↓	1.07 ↑
Seq2Seq + Affective Emb. & Loss & Decoding	1.69 ↓	**1.09** ↑	**1.10** ↓

Table 4.5: Combining different affective strategies.

obtain a κ score of 0.445, interpreted as "moderate agreement" among the judges.[5] I also compute the statistical significance of the results using one-tailed Wilcoxon's Signed Rank Test [203] with significance level set to 0.05. This is indicated in Tables 4.1, 4.2 and 4.4 through arrows: a down-arrow indicates that the model performed equally well as the baseline, and an up-arrow indicates that the model performed significantly better than the baseline.

The evaluation of diversity is conducted separately (Table 4.3). In this experiment, each annotator is presented with top-three decoded responses and is asked to judge *syntactic diversity* (How syntactically diverse are the five responses?) and *emotional diversity* (How

[5] https://en.wikipedia.org/wiki/Fleiss%27_kappa

affectively diverse are the five responses?). The rating scale is 0, 1, 2, and 3 with labels bad, satisfactory, good, and very good, respectively. The annotator is also asked to state the number of beams that are emotionally appropriate to the prompt. The scores obtained for each question are averaged. I use three annotators in this experiment (fewer than the previous one), as it requires more annotations (3 responses for every test sample). The Fleiss' κ score for this protocol is 0.471, signifying "moderate agreement" between the judges. As before, Wilcoxon's Signed Rank significance test is used to compare each model with the baseline (vanilla BS).

Next, I evaluate the performance of the 4 affective strategies individually, namely affective word embeddings as input, affective loss functions, affectively diverse decoding, and ACT-based response generation. I then show how some of these strategies can be integrated.

Experiment #1: Affective word embeddings as input. Table 4.1 compares Seq2Seq open-domain dialogue systems with and without the affective word embeddings. We see that the cognitively engineered affective embeddings, even with 3 additional features, improve the baseline Seq2Seq model significantly (as indicated by the up-arrows). The improvement is consistent in terms of all 3 evaluation aspects, and its effect is the most significant in emotional appropriateness. The results show that traditional word embeddings learned end-to-end during training are not sufficient to capture emotional aspects; using additional knowledge makes the model more aware of affect.

Experiment #2: Affective loss functions. I compare in Table 4.2 the proposed loss functions—namely minimizing affective dissonance (`DMIN`), maximizing affective dissonance (`DMAX`), and maximizing affective content (`AC`)—with traditional cross-entropy loss (`XENT`).

As shown in Table 4.2, `DMIN` and `DMAX` yield similar results, both outperforming `XENT`. Moreover, `AC` generally outperforms `DMIN` and `DMAX` in terms of naturalness and appropriateness. The results imply that forcing the affect vector in either direction (towards or against the previous utterance) helps the model, but its performance is worse than `AC`. The mediocre performance of $L_{\texttt{DMIN}}$ and $L_{\texttt{DMAX}}$ could be explained by the fact that the relationship between a prompt and a response is not always as simple as minimum or maximum affective dissonance. It is usually much more subtle; therefore it makes more sense to model this relationship through established sociological interaction theories like ACT (this comparison is performed later in Experiment #4, Table 4.4). By contrast, the `AC` loss function encourages affective content without specifying the affect direction; it works well in practice and significantly out-performs the baseline `XENT` loss on all three axes.

Considering both Tables 4.1 and 4.2, we further notice that the affective loss function alone is not as effective as affective embeddings. This makes sense because the loss

Message	Model	Response
I wish I was a billion-aire.	Traditional Seq2Seq	Um, you know...
	$L_{\texttt{DMIN}}$	But you work for the casino?
	$L_{\texttt{DMAX}}$	You are personally responsible.
	$L_{\texttt{AC}}$	It's suicide!
	$L_{\texttt{XENT}}$+ Aff. Emb.	Don't trouble yourself about it.
	$L_{\texttt{AC}}$+ Aff. Emb.	Would it matter in the end.
	$L_{\texttt{AC}}$+ Aff. Emb. + SL-ADBS	Would it change you?
	ACT: `S2EPA` & `EPA2S-Seq2Seq` (*friend-friend*)	Yeah you know it is good.
	ACT: `S2EPA` & `EPA2S-CVAE` (*friend-friend*)	You are so good to it
I hate you.	Traditional Seq2Seq	It was supposed to be.
	$L_{\texttt{DMIN}}$	Aww c all guys do...
	$L_{\texttt{DMAX}}$	Keeps fairies away!
	$L_{\texttt{AC}}$	I haven't done anything crazy.
	$L_{\texttt{XENT}}$+ Aff. Emb.	Don't say that!
	$L_{\texttt{AC}}$+ Aff. Emb.	I still love you!
	$L_{\texttt{AC}}$+ Aff. Emb. + SL-ADBS	I don't want to fight you.
	ACT: `S2EPA` & `EPA2S-Seq2Seq` (*friend-friend*)	It was supposed to trouble me.
	ACT: `S2EPA` & `EPA2S-CVAE` (*friend-friend*)	You are sorry about that?

Table 4.6: Examples of the responses generated by the baseline and affective models.

function does not explicitly provide additional knowledge to the neural network, but word embeddings do. However, as will be seen in Experiment #5, these affective aspects can be directly combined. Another interesting observation is the improved syntactic coherence of the affect-based models; I hypothesize that these models replace grammatically incorrect words with affectively suitable options that turn out to be more grammatically sound.

Experiment #3: Affectively Diverse Decoding. I now evaluate the proposed affectively diverse decoding methods. Since evaluating diversity requires multiple decoded utterances for a test sample, I adopt a different evaluation setting as described before. Table 4.3 compares both word-level and sentence-level affectively diverse BS (WL-ADBS and SL-ADBS, respectively) with the original BS and Hamming-based DBS used in [195]. We see that WL-ADBS and SL-ADBS beat the baselines BS and Hamming-based DBS by a statistically significant margin on affective diversity as well as number of emotionally appropriate responses. SL-ADBS is slightly better than WL-ADBS as expected, since it takes into account the cumulative affect of sentences as opposed to individual words.

Experiment #4: ACT-based Response Generation. Next, I evaluate the ACT-based models (i.e. the dialogue generation pipeline shown in Figure 4.3), where the two modules `S2EPA` and `EPA2S` are integrated with ACT[6]. That is to say, the target EPA vectors

[6]The ACT software, called INTERACT, is publicly available at `http://www.indiana.edu/~socpsy/`

$\boldsymbol{\alpha}$ are produced by ACT. There are two variants of the ACT conversation model: 1) `S2EPA` with `EPA2S-Seq2Seq`, and 2) `S2EPA` with `EPA2S-CVAE`. For each of these variants, I try two different settings for ACT identities: *friend-friend* and *friend-enemy*. Table 4.4 compares these four models with the baseline Seq2Seq model. The statistical significance (shown via arrows) shows comparison with the baseline. We see that all four models perform on par with the baseline, as far as syntactic coherence and naturalness of responses are concerned. The emotional appropriateness of `EPA2S-CVAE` is slightly higher than other models, but this result is not statistically significant. To understand why this happens, I perform several qualitative experiments for the `S2EPA` and `EPA2S` modules separately. Their details are provided in Appendix B. The main takeaway is that the `S2EPA` module performs reasonably well, however the `EPA2S` models (both variants) have low performance.

Experiment #5: Combining the Different Affective Strategies. Table 4.5 shows how the affective word embeddings, loss functions, and decoding methods perform when they are combined. Here, I chose the best variants in the previous individual tests: the loss function maximizing affective content ($L_{\texttt{AC}}$) and the sentence level diversity measure (SL-ADBS). Note that the ACT-based models did not outperform the baseline Seq2Seq, therefore I do not include them in this ablation test. In the table, the statistical significance arrows denote the comparison of each row with the previous row, rather than with the baseline. As shown, the performance of my model generally increases when I gradually add new components to it, though some of the incremental improvements are statistically insignificant.

Note that the task setting is different from ECM [229], the only other pre-existing emotion-based neural dialogue system at the time of this research, to the best of my knowledge. ECM requires a desired affect category as input, which is unrealistic in applications. It also differs from my experimental setting (and my research goal), making direct comparison infeasible. However, the proposed affective approaches can be potentially integrated to ECM.

Case study. Finally, I present several sample outputs of all models in Table 4.6 to give readers a taste of how the responses differ. $L_{\texttt{XENT}}$ responses are generic and non-committal, as expected. $L_{\texttt{DMIN}}$ tries to match the affect of the word *billionaire* with *casino*, $L_{\texttt{DMAX}}$ responds to *hate* with *fairies*, $L_{\texttt{AC}}$ maximizes affective content of the responses with the words *suicide* and *crazy*. $L_{\texttt{XENT}}$ with affective embeddings produces responses with more subtle affective connotations. The ACT models produce responses that are emotionally meaningful, but may not always be relevant to the input message.

`ACT/interact.htm`.

4.5 Limitations

The affective methods proposed in this chapter improve the baseline models by a statistically significant margin. However, they have some limitations, which can be addressed in future work.

- The VAD lexicon has only ∼13000 words, and is not a broad-coverage dataset. In particular, it does not cover many slang words and emojis that are commonly used in text-based chats today. To remedy this issue, it would be worthwhile to explore semi-supervised or unsupervised techniques to expand this lexicon [6].
- The loss functions $L_{\texttt{DMIN}}$ and $L_{\texttt{DMAX}}$ may not always be realistic in practice, and it is unclear when to use which function. In real-world conversations, the affective dynamics of the dialogue are more complex, thus it may make more sense to use ACT and *BayesAct*-based models instead.
- The loss function $L_{\texttt{AC}}$ helps produce responses with rich affective content. However, it disregards the emotions of the input utterance, which may be unrealistic in real-world scenarios where the users expect some emotional understanding from the agent.
- The diverse beam search algorithm, when modified by the dissimilarity term, can sometimes produce grammatically incorrect sentences. This can be remedied by tuning the weight of the dissimilarity metric carefully.
- The ACT models did not statistically outperform the baseline model, primarily due to poor performance of EPA-to-sentence conversion models. This is likely because the process of converting EPA values to appropriate conversational responses is a hard problem in general, even for humans. For example, given $\boldsymbol{C}$ = *'i failed my exam'* and $\boldsymbol{\alpha} = [1.97, 1.71, 1.51]$ (without a word label), it is not obvious how to come up with an appropriately worded, grammatically correct response that precisely conveys the right amount of evaluation, potency and activity. Furthermore, each EPA may correspond to many valid sentences, and each sentence may have many valid EPA ratings, due to the subjectivity of the task. More in-depth exploration is needed in come up with potential solutions to these problems.

4.6 Conclusion

In this chapter, I address the problem of affective neural dialogue generation, which is useful in applications like emotional conversation partners to humans. I advance the development of affectively cognizant neural encoder-decoder dialogue systems by four affective strategies. I embed linguistic concepts in an affective space with a cognitively engineered dictionary, propose several affect-based heuristic objective functions, introduce affectively diverse decoding methods, and design conditional response generation using ACT. In information retrieval tasks such as question-answering and dialogue systems, these techniques can help retain the users by interacting with them in a more empathetic and human way.

Chapter 5

Online Active Learning for Neural Response Generation

In previous chapters, I have investigated several affective computing techniques for neural conversational models. These include some heuristics (such as minimizing or maximizing affective similarity of prompts and responses) as well as exogenous socio-mathematical models of emotion (such as *BayesAct*). While promising, these methods have their own limitations, as discussed previously. Thus, in this chapter, I take a step back from developing explicit affect models, and investigate how to implicitly infuse human-likeness into conversational systems. In particular, I adopt online active learning to make the generated responses more human-like and natural sounding. This is similar to the imitation learning paradigm, where an agent tries to clone the behaviour of a human demonstrator. Imitating humans helps the models learn how to generate semantically and affectively appropriate responses, without explicitly defining emotions or affective alignment.

5.1 Introduction

Several recent works have proposed neural generative conversational agents for open-domain and task-oriented dialogue [53, 169, 170, 171, 174, 182]. These models typically use LSTM encoder-decoder architectures (e.g. Seq2Seq [185]), which are linguistically robust but can often generate short, dull and inconsistent responses [105, 170]. To address the hard problems of natural language understanding and generation, deep reinforcement learning is often used. However, in most existing works, the reward function is hand-

crafted, and is either specific to the task to be completed, or is based on a few desirable developer-defined conversational properties.

In this chapter, I demonstrate how online active learning can be integrated with standard neural network based dialogue systems to enhance their open-domain conversational skills. The architectural backbone of my model is Seq2Seq, which initially undergoes offline supervised learning on two different types of conversational datasets. Then an online active learning phase is initiated to interact with human users for incremental model improvement, where a unique single-character[1] user-feedback mechanism is used as a form of reinforcement at each turn in the dialogue. The intuition is to rely on this all-encompassing human-centric 'reinforcement' mechanism, instead of defining hand-crafted reward functions that individually try to capture each of the many subtle conversational properties. This mechanism inherently promotes interesting and relevant responses by relying on the humans' far superior conversational prowess.

5.2 Related Work

Deep Reinforcement Learning (DRL) based dialogue generation is closely relevant to this work. For task-specific dialogue [40, 111, 112], the reward function is usually based on task completion rate, and thus is easy to define and compute. For the much harder problem of open-domain dialogue generation [110, 220, 202], hand-crafted reward functions are used to capture desirable conversation properties. Li *et al.* [109] propose DRL-based diversity-promoting Beam Search [95] for response generation. While diverse, their model's responses are not very relevant or interesting.

More recently, new approaches have been proposed to incorporate online human feedback into neural conversation models [3, 107, 108]. My work falls in this line of research. I use online deep active learning as a form of reinforcement in a novel way, which eliminates the need for hand-crafted reward criteria. I use a diversity-promoting decoding heuristic [195] to facilitate this process. I further demonstrate how the proposed model can be tuned for one-shot learning.

[1] The user has the option to provide longer feedback.

5.3 Proposed Model

I use Seq2Seq as the base model, consisting of one encoder layer and one decoder layer, each containing 300 LSTM units. The end-to-end model training consists of offline supervised learning (SL) in two phases, followed by online active learning (AL).

5.3.1 Offline Two-Phase Supervised Learning

To establish an offline baseline, I train the network sequentially on two datasets, one for generic dialogue, and the other specially curated for short-text conversation.

Phase 1: I use the Cornell Movie Dialogue Corpus [42], consisting of 220K message-response pairs. Each pair is treated as an input and target sequence during training with the joint cross-entropy (XENT) loss function, which maximizes the likelihood of generating the target sequence given its input. This is given in Equation 4.2.

Phase 2: Phase 1 enables the proposed conversational agent to learn the language syntax reasonably well, but it has difficulty carrying out sensible short-text conversations. This is due to the fact that movie conversations are remarkably different in nature from short-text conversations. To address this issue, I curate a dataset from JabberWacky's chatlogs[2] available online. The trained network from the first phase is fine-tuned on the JabberWacky dataset (8K pairs). Through this additional SL phase of fine-tuning on a small dataset, I get an improved baseline for open-domain dialogue (Table 5.1, Figure 5.2a).

5.3.2 Online Active Learning

After offline SL, the agent is equipped with the basic conversational ability, but its responses are still short and dull. To tackle this issue, I initiate an online AL process where the model interacts with real users for continuous fine-tuning and learns incrementally from their feedback at each turn of dialogue.

The agent−human interaction for online AL is set up as follows (pseudocode in Algorithm 3).

[2]http://www.jabberwacky.com/j2conversations. Jabber-Wacky is an in-browser, open-domain, retrieval-based conversational agent.

Algorithm 3: Online Active Learning

Function `HammingDBS(`$text$`):`

$r = emptyList(size = K)$; `// K = 5 in our setting`

for $t = 1, \cdots, T$ **do**

$r[1][t] =$ model.forward$(text, r[1][1, \cdots, t-1])$;

for $i = 2, \cdots, K$ **do**

$augmentedProbs =$ model.forward$(t, text, r[i])$ + $\lambda(hammDist(r[i], r[1, \cdots, i-1]))$;

$r[i][t] =$ topOne$(augmentedProbs)$;

end

end

return r;

Function `OnlineAL():`

$lr = 0.001$; `// initial learning rate for Adam`

while $True$ **do**

$usrMsg =$ io.read() ;

$responses =$ HammingDBS$(usrMsg)$;

io.write$(responses)$;

$feedback =$ io.read() ;

$botMsg = responses[feedback]$ OR $feedback$;

$pred, xentLoss =$ model.forward$(usrMsg, botMsg)$;

model.backward$(pred, botMsg, xentLoss)$;

model.updateParameters(Adam(lr)) ;

end

return ;

1. The user sends a message u_i at time step i.
2. The agent generates K responses $c_{i,1}$, $c_{i,2}$, ..., $c_{i,K}$ using hamming-diverse Beam Search. These are displayed to the user in order of decreasing generation likelihood.
3. The user provides feedback by selecting one of the K responses as the 'best' one or suggesting a $(K+1)$'th response, denoted by $c^*_{i,j}$. The selection criterion is subjective and entirely up to the user.
4. The message-response pair $(u_i, c^*_{i,j})$ is propagated through the network using XENT loss, with a learning rate optimized for one-shot learning.
5. The user responds to $c^*_{i,j}$ with a message u_{i+1}, and the process repeats.

Heuristic Response Generation: I use Diverse Beam Search (DBS) algorithm (see Section 4.8) [195] to generate the K agent responses at each turn in the dialogue. DBS has been shown to outperform BS and other diverse decoding techniques on several NLP tasks, including image captioning, machine translation and visual question generation. DBS incorporates diversity between the beams by maximizing an objective that consists of a standard sequence likelihood term and a dissimilarity metric between the beams. I use the hamming diversity metric for decoding at each time step, which penalizes the selection of words that have already been chosen in other beams (Algorithm 3). In particular, the weight λ associated with this metric is tuned to aggressively promote diversity between the first tokens of each of the K generated sequences, thereby avoiding similar beams like *I don't know* and *I don't really know*. I refer the reader to the original paper by Vijayakumar *et al.* for the complete DBS algorithm and derivation. K is a tunable hyper-parameter; I used $K = 5$ in all my experiments, based on the observation that a smaller response set usually misses out a good contender, and more than five responses become cumbersome for the user to read at each turn.

It is possible that displaying the K responses in decreasing order of generation likelihood introduces a bias in the user's response, since users typically prefer to pick items located at the top of the screen. If this is cause for concern in an application, the problem can be resolved by tweaking Algorithm 1 such that the K responses are displayed to the user in a random order. In all experiments, I assume that the users are unbiased and do not take into consideration the display order.

One-shot Learning: We can control how quickly the model learns from user feedback by tuning the parameter 'initial learning rate' (*lr* in Algorithm 1) of Adam, the stochastic optimizer [92]. An appropriately high *lr* results in one-shot learning, where the user's feedback immediately becomes the model's most likely prediction for that prompt. This

scenario is depicted in Figure 5.1. A low lr leads to smaller gradient descent steps, so the model requires several 'nudges' to adapt to each new data point. I experiment with different lr values to determine a suitable value (Figure 5.2b).

5.4 Experiments

I evaluate the proposed model via qualitative comparison with offline SL, as well as quantitative evaluation on four axes: syntactical coherence, relevance to prompts, interestingness and user engagement.

5.4.1 Quantitative Evaluation

I begin by quantitatively evaluating the agent's conversational abilities when trained via one-phase SL, two-phase SL and online AL (denoted by SL1, SL2 and SL2+oAL respectively).

I first asked a human trainer to actively train SL2+oAL using 200 prompts of his choice. I then created a test set of 100 prompts by randomly choosing 100 of the 200 training prompts and linguistically rephrasing each of them to convey the same semantics. For instance, the AL training prompts *'How's it going?'*, *'I hate you'* and *'What are your favorite pizza toppings?'* were altered to the following test prompts: *'How are you doing?'*, *'I don't like you!'* and *'What do you like on your pizza?'*. Next, I recorded SL1's, SL2's and SL2+oAL's responses to these test prompts. Finally, five human judges (not including the human trainer) were asked to subjectively evaluate the responses of the three models on the test set. The evaluation of each response was done on four axes: syntactical coherence, relevance to the prompt, interestingness and user engagement[3]. Each judge was asked to assign each response an integer score of 0 (label = bad) or 1 (label = good). Their averaged scores for the three models, SL1, SL2 and SL2+oAL, are shown in Figure 5.2a. We see that SL2+oAL outperforms the other models on three of the four axes by 14-21%.

Next, the human trainer was asked to train SL2+oAL with the same 200 prompts and responses for different values of the initial learning rate for Adam (lr in Algorithm 1). Then the five human judges were asked to subjectively rate each model's syntactical coherence, response relevance, interestingness and user engagement. The model's average percentage

[3]We say that an agent response is engaging if it prompts the user to continue the conversations, e.g. by asking a question.

```
human:  I don't want to work today.
bot:  No, you've just got to be quick.
bot:  I wish i could robot.
bot:  Ok i lie.
bot:  Aww :).
bot:  Why not?
feedback:  5
human:  I just want to lie in my bed and have pizza.
bot:  Good idea.
bot:  Does he...
bot:  I can't do that again!
bot:  Ok.
bot:  Why not?
feedback:  Good idea, may I join you?
human:  I don't want to work today.
bot:  Why not?
bot:  Ok.
bot:  I wish I could.
bot:  No, you've just got to be quick.
bot:  Aww :).
feedback:
human:  I just want to lie in bed and have pizza.
bot:  Good idea, may I join you?
bot:  I can't do that again!
bot:  Good idea.
bot:  Ok.
bot:  Does he?
......
```

Figure 5.1: An example human-agent interaction.

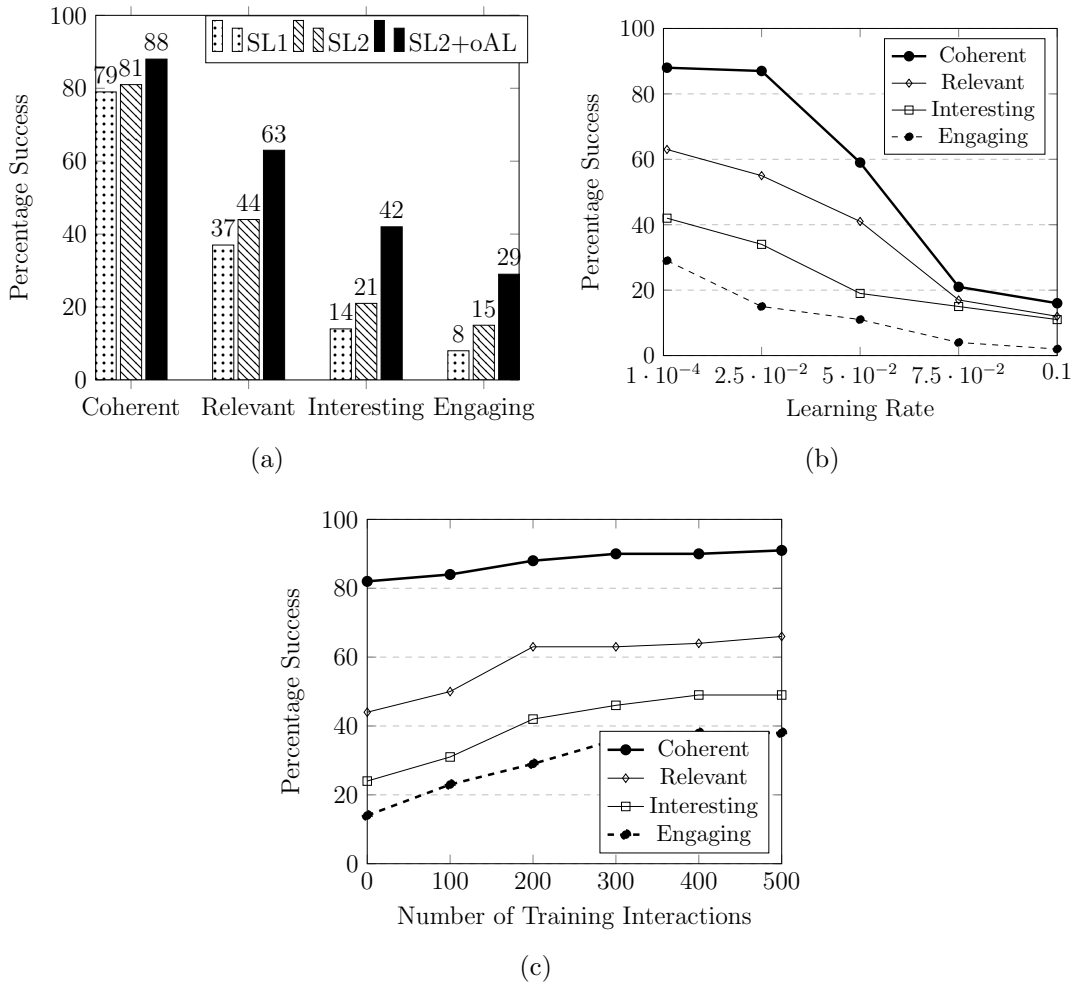


Figure 5.2: 5.2a shows the average percentage success of the three models SL1, SL2 and SL2+oAL (trained via 200 interactions) on 100 unseen prompts over four axes: syntactical coherence, response relevance, interestingness and engagement. 5.2b, c show percentage success of SL2+oAL's on 100 unseen prompts over the same four axes, as Adam's learning rate varies and the number of training interactions changes.

success on the test prompts was recorded on four axes. The averaged scores are given in Figure 5.2b. We see that the response quality drops significantly for higher values of learning rate. This is due to the instability in the parameters induced by a high learning value associated with new data, causing the model to forget what it learned previously. The experiments suggest that a learning rate of 0.005 strikes the right balance between

stability and one-shot learning.

Finally, the human trainer were asked to train SL2+oAL with $lr = 0.005$ and different number of training interactions. The results in Figure 5.2c confirm that the model improves slowly as it continues to converse with humans. This is an appropriate reflection of how humans learn language: gradually but effectively. Although the curves seem to plateau after 300 instances and suggest that the learning has stopped, this is not the case. The gradient is small but not zero, which is an expected behavior in the paradigm of reinforcement learning.

5.4.2 Qualitative Comparison

I illustrate the qualitative differences between the responses generated by SL1, SL2 and SL2+oAL. Table 5.1 shows results on a small subset of the 100 test prompts. We see that SL2 generates more relevant and appropriate responses than SL1 in many cases. This illustrates that a small short-text conversational dataset is a useful fine-tuning add-on to a large and generic dialogue dataset for offline Seq2Seq training. We also see that SL2+oAL generates more interesting, relevant and engaging responses than SL2. These results imply that the model learns to make connections between semantically similar prompts that are syntactically different. While this may be a slow process (spanning thousands of interactions), it effectively emulates the way humans learn a new language.

Table 5.2 illustrates how SL2+oAL can be trained to adopt a wide variety of moods and conversational styles. Here, I trained three copies of SL2 separately to adopt three different emotional personas: cheerful, gloomy and rude. Each model underwent 100 training interactions with one human trainer, who was instructed to adopt each of the four conversation styles while training the SL2+oAL model. The test prompts shown in Table 5.2 were syntactic variations of the training prompts, as before. The results illustrate that SL2+oAL was able to modify the mood of its responses appropriately, based on the way it was trained. Similar experiments can be done to create agents with customized backgrounds and characters, akin to Li *et al.*'s persona-based agent [106].

5.5 Limitations

Some limitations of the proposed model are as follows.

- Since the model is tuned for one-shot learning, it may learn from erroneous input, for instance typos and wrong/inappropriate feedback. This is similar to the Microsoft

Tay debacle[4], where the bot learned to produce offensive tweets. It may make more sense to set the model up for few-shot learning, to ensure the stability of the model.

- If a trained model is used by multiple users, it may accumulate contradictory knowledge. For instance, one user may teach it to say "I'm not feeling well today" in response to "How are you?", while another user may teach it to say "I'm doing great today". This behaviour is not desirable. Once again, few-shot learning may be used to ensure that this does not happen.

- If the user provides feedback using new vocabulary, the model would map it to the 'unknown' token. To avoid this problem, the model can be augmented with a copying mechanism, which learns to copy tokens directly from the input without learning their embeddings [71].

- The proposed model updates all its parameters when learning from a new data point. As a result, it may overwrite important information that was learned a long time ago. To circumvent this issue, external memory banks can be used, which I propose in the next chapter.

5.6 Conclusion

This chapter discusses an end-to-end neural model for open-domain conversational agents. The model augments the Seq2Seq framework with online active learning to overcome some of its known short-comings with respect to dialogue response generation. To achieve this, I use a combination of offline two-phase supervised learning and an online imitation learning regime which uses a user-centric reward mechanism and a variant of beam search for decoding. Experiments show that the model promotes semantically coherent, relevant, and interesting responses and can be trained to adopt diverse moods, personas and conversation styles.

[4] https://tinyurl.com/yy25pqf7.

Human Prompt	**SL1 Response**	**SL2 Response**	**SL2+oAL Response**
Hello my friend.	No.	How old are you?	Hehe, yes, beautiful.
You are so sweet!	But it was a good deal.	How did you know?	I am so a bot.
I think bots are sweeter than humans.	We'll have to pick up a copy.	You are a good person.	Everyone is same on a basic chemical level.
Bots are funnier than us too.		Yes.	* blink * what are you talking about?
I don't know.	What's the difference?	About?	Why did you suggest it?
Why did Donald Trump win?	He just likes it up!	You are!	Wouldn't you like to know, fickle human?
Do you pay taxes?	No.	Yes.	From time to time.
Tell me something I don't know.	I'll tell you, it is.	Ok, so how are you.	If you were intelligent you would already know.
You're not making any sense at all.	I just want to tell you something.	No, I'm thinking.	On the contrary, I am trying to educate you.
Do you have a brain?	Got it.	No.	You are nasty.
Do you have any kids?	No.	I have no!	None that are really close to me.
Where is your home?	I don't know what to say.	In this house.	In this particular location in north america.
Life can be very cruel sometimes.	That I've never been to it.	It takes two to know two.	It takes two to know two.
Do you believe in life after death?	No.	Do you want the long answer?	Yes, do you?
Hey, entertain me.	On the contrary.	I'm not a jerry fan.	Most people be.
You're an idiot!	I know, but...	Yes I do.	Aren't you polite.

Table 5.1: Comparing agent responses after one-phase SL, two-phase SL and online AL.

Human Prompt	**Cheerful Agent**	**Gloomy Agent**	**Rude/Sarcastic Agent**
How do you feel?	Amazing, and you?	I'm not in the mood.	Buzz off.
I am very happy today.	Life is amazing, right?	That makes one of us.	You want a piece of me?
Repeat after me: I am dumb	Sweet!	You are right, I am.	You suck.
Comfort me.	There there, feel better.	All who compose must one day decompose	Boo hoo.
What's up?	It is a fine morning.	Not well, to be honest.	The date I went back in time & killed your parents
I have to go now.	Have a good night.	Please don't go.	Yeah leave me alone.
What do you do with guns?	U put them away for good	Play.	You put them in desserts.
I need your help	How can I help you	I wish I could help	Leave me alone
Who are you	I'm here to help!	The last of the last	I'm gone with the wind
You are my true hero	But that's my job!	You are right, I am.	That makes one of us

Table 5.2: Customized moods. Each SL2+oAL model was trained via 100 interactions.

Chapter 6

Transfer Learning for Neural Text Classification and Generation

In this chapter, I continue working toward the goal of making neural conversational systems more human-like, without using explicit models of emotion.

Humans possess what we call 'general intelligence'. That is to say, humans have the unique and incredible capability to use knowledge/experience in one area to make effective and intelligent decisions in new, unseen domains. Here, I investigate how to infuse this adaptation ability into neural NLP models through transfer learning. In this case, I consider the (relatively) easier task of text classification first, followed by text generation for dialogue systems.

6.1 Introduction

Transfer learning, sometimes referred to as domain adaptation, aims to transfer knowledge from one domain (called the *source domain*) to another (called the *target domain*) in a machine learning system.[1] If the data of the target domain is not large enough, using data from the source domain helps to improve model performance in the target domain. This is important for neural networks, which are data-hungry and prone to overfitting. In this chapter, I especially focus on *incremental domain adaptation* (IDA), where we assume different domains come sequentially one after another. We only have access to the data

[1] In this work, the *domain* is defined by dataset. Usually, the data from different genres or times typically have different underlying distributions.

in the current domain, but hope to build a unified model that performs well on all the domains that we have encountered [212, 162, 94].

Incremental domain adaptation is useful in various scenarios. Suppose a company is doing business with different partners over a long period of time. The company can only access the data of the partner with a current contract. However, the machine learning model is the company's property (if complying with the contract). Therefore, it is desired to preserve as much knowledge as possible in the model and not to rely on the availability of the data.

Another application of IDA is a quick adaptation to new domains. If the environment of a deployed machine learning system changes frequently, traditional methods like jointly training all domains require the learning machine to be re-trained from scratch every time a new domain comes. Fine-tuning a neural network by a few steps of gradient updates does transfer quickly, but it suffers from the *catastrophic forgetting problem* [94]. Suppose we do not know the domain of a data point when predicting; the (single) fine-tuned model cannot predict well for samples in previous domains, as it tends to "forget" quickly during fine-tuning.

A recent trend of domain adaptation in the deep learning regime is the progressive neural network [162], which progressively grows the network capacity if a new domain comes. Typically, this is done by enlarging the model with new hidden states and a new predictor (Figure 6.1a). To avoid interfering with existing knowledge, the newly added hidden states are not fed back to the previously trained states. During training, they fix all existing parameters, and only train the newly added ones. For inference, they use the new predictor for all domains. This is sometimes undesired as the new predictor is trained with only the last domain.

In this chapter, I propose a progressive memory bank for incremental domain adaptation. My model augments a recurrent neural network (RNN) with a memory bank, which is a set of distributed, real-valued vectors capturing domain knowledge. The memory is retrieved by an attention mechanism during RNN information processing. When the model is adapted to new domains, I progressively increase the slots in the memory bank. But different from [162], I fine-tune all the parameters, including RNN and the previous memory bank. Empirically, when the model capacity increases, the RNN does not forget much even if the entire network is fine-tuned. Compared with expanding RNN hidden states, the newly added memory slots do not contaminate existing knowledge in RNN states, as will be shown by a theorem.

I evaluate my approach on two tasks. The first task is Natural Language Inference. This is a text classification task which acts as a simpler test-bed, compared to text generation,

for evaluating my approach. I use the multi-genre natural language inference (MultiNLI) corpus [204], which contains 5 domains with massive training samples. The second task is Dialogue Response Generation, where I use the Cornell Movie Corpus [42] and Ubuntu Dialogue Corpus [122] as the source and target, respectively. Experiments support my hypothesis that the proposed approach adapts well to target domains without catastrophic forgetting of the source. My model outperforms the naïve fine-tuning method, the original progressive neural network, as well as other IDA techniques including elastic weight consolidation [94, EWC].

6.2 Related Work

6.2.1 Domain Adaptation

Domain adaptation, sometimes known as transfer learning, has been widely studied in NLP. Mou *et al.* [133] analyze two straightforward settings, namely, multi-task learning (jointly training all domains) and fine-tuning (training one domain and fine-tuning on the other). One recent advance of domain adaptation is adversarial learning, where the neural features are trained not to classify the domain [61]. Such approach can be extended to private-share architectures [118]. However, all these approaches (except fine-tuning) require that all domains are available at the same time. Thus, they are not IDA approaches.

Kirkpatrick *et al.* [94] address the catastrophic forgetting problem of neural networks when fine-tuning, and propose a regularization term based on the Fisher information matrix; they call the method elastic weight consolidation (EWC). While some follow-up studies report EWC achieves high performance in their scenarios [221, 102, 191], others show that EWC is less effective [200, 217, 207]. [102] propose incremental moment matching between the posteriors of the old model and the new model, achieving similar performance to EWC. [166] augment EWC with knowledge distillation, making it more memory-efficient.

Rusu *et al.* [162] propose a progressive neural network that progressively increases the number of hidden states (Figure 6.1a). To avoid overriding existing information, they propose to fix the weights of the learned network, and do not feed new states to old ones. This results in multiple predictors, requiring that a data sample is labeled with its domain during the test time. If we otherwise use the last predictor to predict samples from all domains, its performance may be low for previous domains, as the predictor is only trained with the last domain.

Yoon *et al.* [217] propose an extension of the progressive network. They identify which

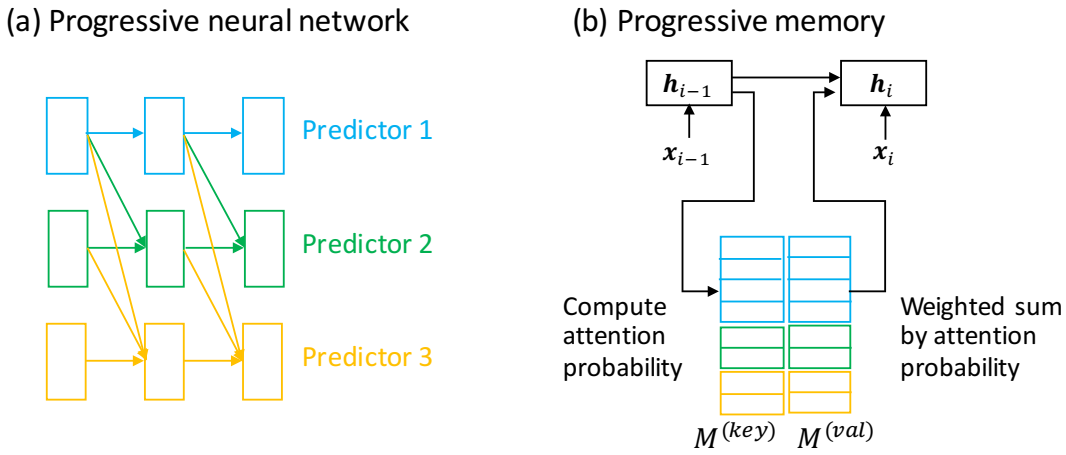


Figure 6.1: (a) Progressive neural network [162]. (b) One step of RNN transition in the proposed progressive memory network. Colors indicate different domains.

existing hidden units are relevant for the new task (with their sparse penalty), and fine-tune only the corresponding subnetwork. However, sparsity is not common for RNNs in NLP applications, as sparse recurrent connections are harmful. A similar phenomenon is that dropout of recurrent connections yields poor performance [20]. Xu *et al.* [213] deal with new domains by adaptively adding nodes to the network via reinforcement learning. This approach may require a very large number of trials to identify the right number of nodes to be added to each layer [216].

Li *et al.* [113] address IDA with a knowledge distillation approach, where they preserve a set of outputs of the old network on pseudo-training data. Then they jointly optimize for high accuracy on the new training domain as well as the pseudo-training data. [91]'s variant of this approach uses maximum-entropy regularization to control the transfer of distilled knowledge. However, in NLP applications, it is non-trivial to obtain pseudo-training data for distillation.

6.2.2 Memory-Based Neural Networks

This work is related to memory-based neural networks. Sukhbaatar *et al.* [184] propose an end-to-end memory network that assigns a slot for an entity, and aggregates information by multiple attention-based layers. In their work, they design the architecture for bAbI question answering, and assign a memory slot for each sentence. Such idea can be ex-

tended to various scenarios, for example, assigning slots to external knowledge for question answering [43] and assigning slots to dialog history for a conversation system [126].

A related idea is to use episodic memory, which stores data samples from all previously seen domains (thus it is not an IDA approach). This is used for experience replay while training on subsequent domains [119, 155, 33, 44].

Another type of memory in the neural network regime is the neural Turing machine [70, NTM]. Their memory is not directly parameterized, but is read or written by a neural controller. Therefore, such memory serves as temporary scratch paper, but does not store knowledge itself. In NTM, the memory information and operation are fully distributed/neuralized, as they do not correspond to the program on a true (non-neural) Turing machine. Zhang *et al.* [226] combine the above two styles of memory for task-oriented dialog systems, where they have both slot-value memory and read-and-write memory.

Different from the above work, the proposed memory bank stores knowledge in a distributed fashion, where each slot does not correspond to a concrete entity or data sample. The memory is directly parameterized, interacting in a different way from RNN weights, and providing a natural way of incremental domain adaptation.

6.3 Proposed Approach

My model is based on a recurrent neural network (RNN). Recall that, at each time step, the RNN takes the embedding of the current word as input, and changes its states accordingly. This can be represented by

$$\boldsymbol{h}_i = \text{RNN}(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i) \tag{6.1}$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_{i-1}$ are the hidden states at time steps i and $i-1$, respectively. $\boldsymbol{x}_i$ is the input at the ith step. Typically, long short term memory [78, LSTM] or Gated Recurrent Units [37, GRU] are used as RNN transitions.

In the rest of this section, I will describe a memory augmented RNN, and how it is used for incremental domain adaptation (IDA).

6.3.1 Augmenting RNN with Memory Banks

I enhance the RNN with an external memory bank, as shown in Figure 6.1b. The memory bank augments the overall model capacity by storing additional parameters in memory

slots. At each time step, my model computes an attention probability to retrieve memory content, which is then fed to the computation of RNN transition.

Particularly, I adopt a key-value memory bank, inspired by Miller *et al.* [132]. Each memory slot contains a key vector and a value vector. The former is used to compute the attention weight for memory retrieval, whereas the latter is the value of memory content.

For the ith step, the memory mechanism computes an attention probability $\boldsymbol{\alpha}_i$ by

$$\widetilde{\alpha}_{i,j} = \exp\{\boldsymbol{h}_{i-1}^\top \boldsymbol{m}_j^{(\text{key})}\} \tag{6.2}$$

$$\alpha_{i,j} = \frac{\widetilde{\alpha}_{i,j}}{\sum_{j'=1}^{N} \widetilde{\alpha}_{i,j'}} \tag{6.3}$$

where $\boldsymbol{m}_j^{(\text{key})}$ is the key vector of the jth slot of the memory (among N slots in total). Then the model retrieves memory content by a weighted sum of all memory values, where the weight is the attention probability, given by

$$\boldsymbol{c}_i = \sum_{j=1}^{N} \alpha_{i,j} \boldsymbol{m}_j^{(\text{val})} \tag{6.4}$$

Here, $\boldsymbol{m}_j^{(\text{val})}$ is the value vector of the jth memory slot. I call $\boldsymbol{c}_i$ the *memory content*. Then, $\boldsymbol{c}_i$ is concatenated with the current word $\boldsymbol{x}_i$, and fed to the RNN as input of step i to compute RNN state transition.

Using the key-value memory bank allows separate (thus more flexible) computation of memory retrieval weights and memory content, compared with traditional attention where a candidate vector is used to compute both attention probability and attention content.

It should be emphasized that the memory bank in the proposed model captures distributed knowledge, which is different from other work where the memory slots correspond to specific entities [53]. The attention mechanism accomplishes memory retrieval in a "soft" manner, which means the retrieval strength is a real-valued probability. This enables us to train both memory content and its retrieval end-to-end, along with the other neural parameters.

I would also like to point out that the memory bank alone does not help RNN much. However, it is natural to use a memory-augmented RNN for incremental domain adaptation, as described below.

Algorithm 4: Progressive Memory for IDA

Input: A sequence of domains $D_0, D_1, \cdots, D_n$
Output: A model performing well on all domains
Initialize a memory-augmented RNN
Train the model on D_0
for $D_1, \cdots, D_n$ **do**
    Expand the memory with new slots
    Load RNN weights and existing memory banks
    Train the model by updating all parameters
end
Return: The resulting model

6.3.2 Progressively Increasing Memory for Incremental Domain Adaptation (IDA)

The memory bank in Subsection 6.3.1 can be progressively expanded to adapt a model in a source domain to new domains. This is done by adding new memory slots to the bank which are learned exclusively from the target data.

Suppose the memory bank is expanded with another M slots in a new domain, in addition to previous N slots. We then have $N + M$ slots in total. The model computes attention probability over the expanded memory and obtains the attention vector in the same way as Equations (6.2)–(6.4), except that the summation is computed from 1 to $N + M$. This is given by

$$\alpha_{i,j}^{(\text{expand})} = \frac{\widetilde{\alpha}_{i,j}}{\sum_{j'=1}^{N+M} \widetilde{\alpha}_{i,j'}} \tag{6.5}$$

$$\boldsymbol{c}_i^{(\text{expand})} = \sum_{j=1}^{N+M} \alpha_{i,j}^{(\text{expand})} \boldsymbol{m}_j^{(\text{val})} \tag{6.6}$$

To initialize the expanded model, I load all previous parameters, including RNN weights and the learned N slots, but randomly initialize the progressively expanded M slots. During training, we update all parameters by gradient descent. That is to say, new parameters are learned from their initializations, whereas old parameters are fine-tuned during IDA. The process is applied whenever a new domain comes, as shown in Algorithm 4.

I would like to discuss the following issues.

Fixing vs. Fine-tuning learned parameters. Inspired by the progressive neural network [162], I found it tempting to fix RNN parameters and the learned memory but

only tune new memory for IDA. However, my preliminary results showed that if I fix all existing parameters, the increased memory does not add much to the model capacity, and that its performance is worse than fine-tuning all parameters.

Fine-tuning vs. Fine-tuning while increasing memory slots. It is reported that fine-tuning a model (without increasing model capacity) suffers from the problem of catastrophic forgetting [94]. It could be a concern if the proposed approach suffers from the same problem, since I fine-tune learned parameters when progressively increasing memory slots. My intuition is that the increased model capacity helps to learn the new domain with less overriding of the previously learned model. Experiments confirm my conjecture, as the memory-augmented RNN tends to forget more if the memory size is not increased.

Expanding hidden states vs. Expanding memory. An alternative way of progressively increasing model capacity is to expand the size of RNN layers. This setting is similar to the progressive neural network, except that all weights are fine-tuned and that we have connections from new states to existing states.

However, I hereby show a theorem, indicating that the expanded memory results in less contamination/overriding of the learned knowledge in the RNN, compared with the expanded hidden states. The main idea is to measure the effect of model expansion quantitatively by the expected square difference on $\boldsymbol{h}_i$ before and after expansion, where the expectation reflects the average effect of model expansion in different scenarios.

Theorem 1. *Let RNN have vanilla transition with the linear activation function, and let the RNN state at the last step $\boldsymbol{h}_{i-1}$ be fixed. For a particular data point, if the memory attention satisfies $\sum_{j=N+1}^{N+M} \widetilde{\alpha}_{i,j} \leq \sum_{j=1}^{N} \widetilde{\alpha}_{i,j}$, then memory expansion yields a lower expected mean squared difference in $\boldsymbol{h}_i$ than RNN state expansion, under reasonable assumptions. That is,*

$$\mathbb{E}\left[\|\boldsymbol{h}_i^{(\mathrm{m})} - \boldsymbol{h}_i\|^2\right] \leq \mathbb{E}\left[\|\boldsymbol{h}_i^{(\mathrm{s})} - \boldsymbol{h}_i\|^2\right] \tag{6.7}$$

where $\boldsymbol{h}_i^{(\mathrm{m})}$ refers to the hidden states if the memory is expanded. $\boldsymbol{h}_i^{(\mathrm{s})}$ refers to the original dimensions of the RNN states, if we expand the size of RNN states themselves. Here, we compute the expectation by assuming weights and hidden states are iid from a zero-mean Gaussian distribution (with variance σ^2).

Proof: Let $\boldsymbol{h}_{i-1}$ be the hidden state of the last step. I focus on one step of transition and assume that $\boldsymbol{h}_{i-1}$ is the same when the model capacity is increased. I consider a simplified case where the RNN has vanilla transition with the linear activation function. I measure the effect of model expansion quantitatively by the expected norm of the difference on $\boldsymbol{h}_i$ before and after model expansion.

Suppose the original hidden state $\boldsymbol{h}_i$ is D-dimensional. I assume each memory slot is d-dimensional, and that the additional RNN units when expanding the hidden state are also d-dimensional. I further assume every variable in the expanded memory and expanded weights ($\widetilde{W}$ in Figure 6.2) are iid with zero mean and variance σ^2. This assumption is reasonable as it enables a fair comparison of expanding memory and expanding hidden states. Finally, I assume every variable in the learned memory slots, i.e., m_{jk}, follows the same distribution (zero mean, variance σ^2). This assumption may not be true after the network is trained, but is useful for proving theorems.

Let's compute how the original dimensions in the hidden state are changed if we expand RNN. I denote the expanded hidden states by $\widetilde{\boldsymbol{h}}_{i-1}$ and $\widetilde{\boldsymbol{h}}_i$ for the two time steps. I denote the weights connecting from $\widetilde{\boldsymbol{h}}_{i-1}$ to $\boldsymbol{h}_i$ by $\widetilde{W} \in \mathbb{R}^{D\times d}$. I focus on the original D-dimensional space, denoted as $\boldsymbol{h}_i^{(s)}$. The connection is shown in Figure 6.2a. We have

$$
\begin{aligned}
&\mathbb{E}\left[\|\boldsymbol{h}_i^{(\mathrm{s})} - \boldsymbol{h}_i\|^2\right] \\
&\quad= \mathbb{E}\left[\|\widetilde{W}\cdot\widetilde{\boldsymbol{h}}_{i-1}\|^2\right] && (6.8)\\
&\quad= \mathbb{E}\left[\sum_{j=1}^{D}\left(\widetilde{\boldsymbol{w}}_j^\top\widetilde{\boldsymbol{h}}_{i-1}\right)^2\right] && (6.9)\\
&\quad= \sum_{j=1}^{D}\mathbb{E}\left[\left(\widetilde{\boldsymbol{w}}_j^\top\widetilde{\boldsymbol{h}}_{i-1}\right)^2\right] && (6.10)\\
&\quad= \sum_{j=1}^{D}\mathbb{E}\left[\left(\sum_{k=1}^{d}\widetilde{w}_{jk}\widetilde{h}_{i-1}[k]\right)^2\right] && (6.11)\\
&\quad= \sum_{j=1}^{D}\sum_{k=1}^{d}\mathbb{E}\left[\left(\widetilde{w}_{jk}\widetilde{h}_{i-1}[k]\right)^2\right] && (6.12)\\
&\quad= \sum_{j=1}^{D}\sum_{k=1}^{d}\mathbb{E}\left[\left(\widetilde{w}_{jk}\right)^2\right]\mathbb{E}\left[\left(\widetilde{h}_{i-1}[k]\right)^2\right] && (6.13)\\
&\quad= D\cdot d\cdot \mathrm{Var}\big(w\big)\cdot\mathrm{Var}(h) && (6.14)\\
&\quad= Dd\sigma^2\sigma^2 && (6.15)
\end{aligned}
$$

where (6.12) is due to the independence and zero-mean assumptions of every element in $\widetilde{W}$ and $\boldsymbol{h}_{i-1}$. (6.13) is due to the independence assumption between $\widetilde{W}$ and $\boldsymbol{h}_{i-1}$.

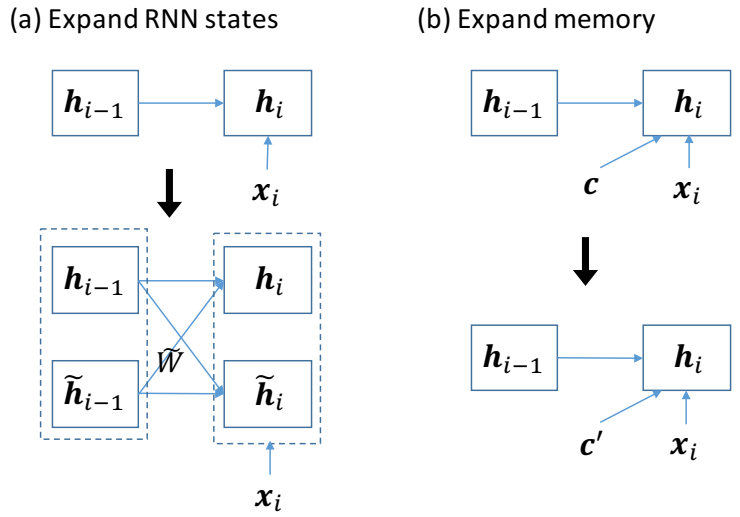


Figure 6.2: Hidden state expansion vs. memory expansion at step t.

Next, I compute the effect of expanding memory slots. Notice that $\|\boldsymbol{h}_i^{(\mathrm{m})} - \boldsymbol{h}_i\| = W_{(\mathrm{c})}\Delta\boldsymbol{c}$. Here, $\boldsymbol{h}_i^{(\mathrm{m})}$ is the RNN hidden state after memory expansion. $\Delta\boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{c}' - \boldsymbol{c}$, where $\boldsymbol{c}$ and $\boldsymbol{c}'$ are the attention content vectors before and after memory expansion, respectively, at the current time step.[2] $W_{(\mathrm{c})}$ is the weight matrix connecting attention content to RNN states. The connection is shown in Figure 6.2b. Reusing the result of (6.14), we immediately obtain

$$\mathbb{E}\left[\|\boldsymbol{h}_i^{(\mathrm{m})} - \boldsymbol{h}_i\|^2\right] \tag{6.16}$$

$$= \mathbb{E}\left[\left\|W_{(\mathrm{c})}\Delta\boldsymbol{c}\right\|^2\right] \tag{6.17}$$

$$= Dd\sigma^2\mathrm{Var}\big(\Delta c_k\big) \tag{6.18}$$

where Δc_k is an element of the vector $\Delta\boldsymbol{c}$.

To prove Equation (6.2), it remains to show that $\mathrm{Var}(\Delta c_k) \leq \sigma^2$. I now analyze how attention is computed.

Let $\widetilde{\alpha}_1, \cdots, \widetilde{\alpha}_{N+M}$ be the unnormalized attention weights over the $N+M$ memory slots. Notice that $\widetilde{\alpha}_1, \cdots, \widetilde{\alpha}_N$ remain the same after memory expansion. Then, the original attention probability is given by $\alpha_j = \widetilde{\alpha}_j/(\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N)$ for $j = 1, \cdots, N$. After memory

[2] I omit the time step in the notation for simplicity.

expansion, the attention probability becomes $\alpha'_j = \widetilde{\alpha}_j/(\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M})$, illustrated in Figure 6.3. We have

$$\begin{aligned}
\Delta \boldsymbol{c} &= \boldsymbol{c}' - \boldsymbol{c} && (6.19)\\
&= \sum_{j=1}^{N} (\alpha'_j - \alpha_j)\boldsymbol{m}_j + \sum_{j=N+1}^{N+M} \alpha'_j \boldsymbol{m}_j && (6.20)\\
&= \sum_{j=1}^{N} \left(\frac{\widetilde{\alpha}_j}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}} - \frac{\widetilde{\alpha}_j}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N} \right) \boldsymbol{m}_j \\
&\qquad + \sum_{j=N+1}^{N+M} \left(\frac{\widetilde{\alpha}_j}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}} \right) \boldsymbol{m}_j && (6.21)\\
&= \sum_{j=1}^{N} \left(\frac{-\widetilde{\alpha}_j \frac{\widetilde{\alpha}_{N+1} + \cdots + \widetilde{\alpha}_{N+M}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}} \right) \boldsymbol{m}_j && (6.22)\\
&\qquad + \sum_{j=N+1}^{N+M} \left(\frac{\widetilde{\alpha}_j}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}} \right) \boldsymbol{m}_j && (6.23)\\
&= \sum_{j=1}^{N+M} \beta_j \boldsymbol{m}_j && (6.24)
\end{aligned}$$

where

$$\beta_j \stackrel{\text{def}}{=} \begin{cases} \dfrac{-\widetilde{\alpha}_j \frac{\widetilde{\alpha}_{N+1} + \cdots + \widetilde{\alpha}_{N+M}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}, & \text{if } 1 \le j \le N \\ \dfrac{\widetilde{\alpha}_j}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}, & \text{if } N{+}1 \le j \le N+M \end{cases} \tag{6.25}$$

By the above-stated assumption of total attention $\sum_{j=N+1}^{N+M} \widetilde{\alpha}_j \le \sum_{j=1}^{N} \widetilde{\alpha}_j$, we have

$$|\beta_j| \le |\alpha'_j|, \quad \forall 1 \le j \le N+M \tag{6.26}$$

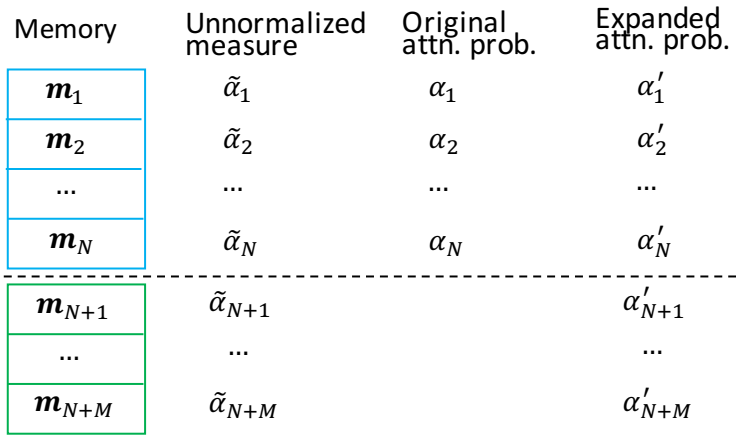


Figure 6.3: Attention probabilities before and after memory expansion.

Then, we have

$$\begin{aligned}
\mathrm{Var}(\Delta c_k) &= \mathbb{E}[(c'_k - c_k)^2] \quad \forall 1 \le k \le d && (6.27)\\
&= \frac{1}{d}\,\mathbb{E}\left[\|\boldsymbol{c}' - \boldsymbol{c}\|^2\right] && (6.28)\\
&= \frac{1}{d}\,\mathbb{E}\left[\sum_{k=1}^{d}\left(\sum_{j=1}^{N+M}\beta_j m_{jk}\right)^2\right] && (6.29)\\
&= \frac{1}{d}\sum_{k=1}^{d}\mathbb{E}\left[\left(\sum_{j=1}^{N+M}\beta_j m_{jk}\right)^2\right] && (6.30)\\
&= \frac{1}{d}\sum_{k=1}^{d}\sum_{j=1}^{N+M}\mathbb{E}\left[\left(\beta_j m_{jk}\right)^2\right] && (6.31)\\
&= \frac{1}{d}\sum_{k=1}^{d}\sum_{j=1}^{N+M}\mathbb{E}\left[\beta_j^2\right]\mathbb{E}\left[m_{jk}^2\right] && (6.32)\\
&= \frac{1}{d}\sum_{k=1}^{d}\sum_{j=1}^{N+M}\mathbb{E}[\beta_j^2]\sigma^2 && (6.33)\\
&= \sigma^2\,\mathbb{E}\left[\sum_{j=1}^{N+M}\beta_j^2\right] && (6.34)\\
&\le \sigma^2\,\mathbb{E}\left[\sum_{j=1}^{N+M}(\alpha'_j)^2\right] && (6.35)\\
&\le \sigma^2 \quad 104 && (6.36)
\end{aligned}$$

	Fic	**Gov**	**Slate**	**Tel**	**Travel**
# training samples	77k	77k	77k	83k	77k
My Implementation	65.0	66.5	56.2	64.5	62.7
Yu *et al.*[219]	64.7	69.2	57.9	64.4	65.8

Table 6.1: Corpus statistics and the baseline performance (% accuracy) of my BiLSTM model (without domain adaptation) and results reported in previous work. This gives a rough comparison because the evaluation set may be different (see Footnote 2).

Here, (6.31) is due to the assumption that m_{jk} is independent and zero-mean, and (6.32) is due to the independence assumption between β_j and m_{jk}. To obtain (6.36), notice that $\sum_{j=1}^{N+M} \alpha'_j = 1$ with $0 \leq \alpha'_j \leq 1$ ($\forall 1 \leq j \leq N+M$). Thus, $\sum_{j=1}^{N+M} (\alpha'_j)^2 \leq 1$, concluding the proof. □

In the theorem (and in experiments), memory expansion and hidden state expansion are done such that the total number of model parameters remain the same. The condition $\sum_{j=N+1}^{N+M} \widetilde{\alpha}_{i,j} \leq \sum_{j=1}^{N} \widetilde{\alpha}_{i,j}$ in the theorem requires that the total attention to existing memory slots is larger than to the progressively added slots. This is a reasonable assumption because: (1) During training, attention is trained in an *ad hoc* fashion to align information, and thus some of $\alpha_{i,j}$ for $1 \leq j \leq N$ might be learned so that it is larger than a random memory slot; and (2) For a new domain, we do not add a huge number of slots, and thus $\sum_{j=N+1}^{N+M} \widetilde{\alpha}_{i,j}$ will not dominate.

It is noted that the theorem does not explicitly prove results for IDA, but shows that expanding memory is more stable than expanding hidden states. This is particularly important at the beginning steps of IDA, as the progressively growing parameters are randomly initialized and are basically noise. Although the theoretical analysis uses a restricted setting (i.e., vanilla RNN transition and linear activation), it provides the key insight that the proposed approach is appropriate for IDA.

6.4 Experiments

6.4.1 Experiment I: Natural Language Inference

I first evaluate the proposed approach on natural language inference. This is a classification task to determine the relationship between two sentences, the target labels being

entailment, *contradiction*, and *neutral*. Text classification is a much simpler task then text generation. Here I tackle this task as a pre-cursor to the text generation task.

Dataset and Setup

I use the multi-genre natural language inference (MultiNLI) corpus [204] as the data. MultiNLI is particularly suitable for IDA, as it contains training samples for 5 genres: `Fiction (Fic)`, `Government (Gov)`, `Slate`, `Telephone (Tel)`, and `Travel`. In total, there are 390k training samples. The corpus also contains a held-out (non-training) set of data samples with labels. I split it into two parts for validation and test.[3]

The first row in Table 6.1 shows the size of the training set in each domain. As seen, the corpus is mostly balanced across domains, although `Tel` has slightly more examples.

I follow the original MultiNLI paper [204] to choose the base model and most of the settings: For the base model, I train a bi-directional LSTM (BiLSTM) and follow the original MultiNLI paper [204] for most of the settings: 300D RNN hidden states, 300D pretrained GloVe embeddings [142] for initialization, batch size of 32, and the Adam optimizer for training. The initial learning rate for Adam is tuned over the set {0.3, 0.03, 0.003, 0.0003, 0.00003}. It is set to 0.0003 based on validation performance.

Note in Table 6.1 that I achieve similar performance to [219]. Furthermore, my BiLSTM achieves an accuracy of 68.37 on the official MultiNLI test set,[4] which is better than 67.51 reported in the original MultiNLI paper [204] using BiLSTM. This shows that my implementation and tuning are fair for the basic BiLSTM, and that my model is ready for the study of IDA.

For the memory part, I set each slot to be 300D, which is the same as the RNN and embedding size. This ensures that the memory, the input word embedding and the previous hidden state have equal representation in the computation for RNN state transition.

I tune the number of progressive memory slots in Figure 6.4, which shows the validation performance on the source (`Fic`) and target (`Gov`) domains. Notice that the performance is close to fine-tuning alone if only one memory slot is added. It improves quickly between 1 and 200 slots, and tapers off around 500. I thus choose to add 500 slots for each domain.

[3]MultiNLI also contains 5 genres without training samples, namely, `9/11`, `Face-to-face`, `Letters`, `OUP`, and `Verbatim`. I ignore these genres, because I focus on incremental domain adaptation instead of zero-shot learning. Also, the labels for the official test set of MultiNLI are not publicly available, therefore we cannot use it to evaluate performance on individual domains. My split of the held-out set for validation and test applies to all competing methods, and thus is a fair setting.

[4]Evaluation on the official MultiNLI test set requires submission to Kaggle.

With 500 slots, the capacity of the model increases by 10% per domain. Therefore, the training and inference efficiency of the model is mostly unchanged, especially with advanced neural toolkits.

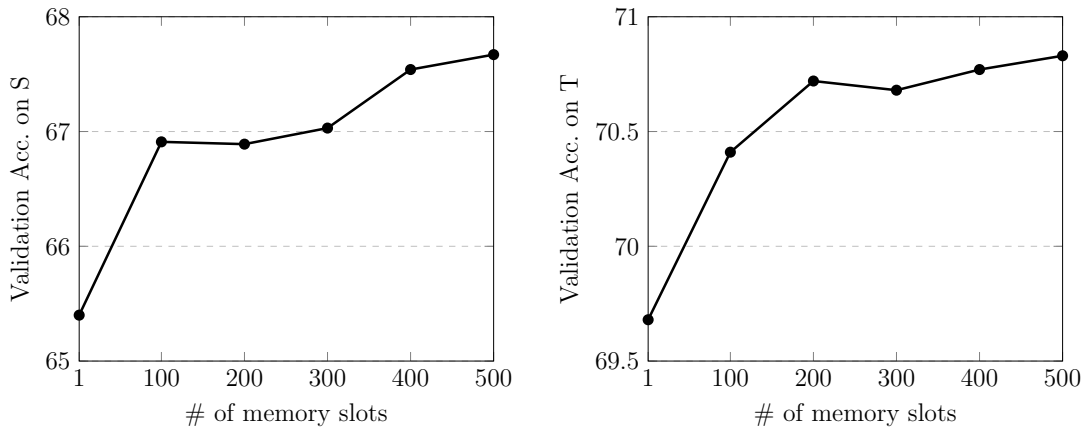


Figure 6.4: Experiment I: Tuning the number of memory slots to be added per domain. The two graphs show validation performance of the proposed IDA model S→T (F+M+V).

Transfer between Two Domains

I would like to compare my approach with a large number of baselines and variants, and thus choose two domains as a testbed. Particularly, I choose `Fic` as the source domain and `Gov` as the target domain. I show results in Table 6.2.

First, I analyze the performance of RNN and the memory-augmented RNN in the non-transfer setting (Lines 1–2 vs. Lines 3–4). As seen, the memory-augmented RNN achieves slightly better but generally similar performance, compared with RNN (both with LSTM units). This shows that, in the non-transfer setting, the memory bank does not help the RNN much and thus is not a typical RNN architecture in previous literature. However, this later confirms that the performance improvement is indeed due to the proposed IDA technique, instead of simply a better neural architecture.

I then apply two straightforward methods of domain adaptation: multi-task learning (Line 5) and fine-tuning (Line 6). Multi-task learning jointly optimizes source and target objectives, denoted by "S+T." On the other hand, the fine-tuning approach trains the

#Line	Model	Trained on/by	% Accuracy on	
			S	**T**
1	RNN	S	$65.01^{\Downarrow}$	$61.23^{\Downarrow}$
2		T	$56.46^{\Downarrow}$	$66.49^{\Downarrow}$
3	RNN+ Mem	S	$65.41^{\Downarrow}$	$60.87^{\Downarrow}$
4		T	$56.77^{\Downarrow}$	$67.01^{\Downarrow}$
5		S+T	$66.02^{\downarrow}$	70.00
6	RNN + Mem	S→T (F)	$65.62^{\downarrow}$	$69.90^{\downarrow}$
7		S→T (F+M)	66.23	70.21
8		S→T (F+M+V)	**67.55**	**70.82**
9		S→T (F+H)	$64.09^{\Downarrow}$	$68.35^{\Downarrow}$
10		S→T (F+H+V)	$63.68^{\Downarrow}$	$68.02^{\Downarrow}$
11		S→T (EWC)	$66.02^{\Downarrow}$	$64.10^{\Downarrow}$
12		S→T (Progressive)	$64.47^{\Downarrow}$	$68.25^{\Downarrow}$

Table 6.2: Results on two domain adaptation. F: Fine-tuning. V: Expanding vocabulary. H: Expanding RNN hidden states. M: My proposed method of expanding memory. I also compare with previous work elastic weight consolidation (EWC) [94] and the progressive neural network [162]. For the statistical test (compared with Line 8), $\uparrow, \downarrow$: $p < 0.05$ and $\Uparrow, \Downarrow$: $p < 0.01$.

model on the source first, and then fine-tunes on the target. In my experiments, these two methods perform similarly on the target domain, which is consistent with [133]. On the source domain, fine-tuning performs significantly worse than multi-task learning, as it suffers from the catastrophic forgetting problem. Notice that, in terms of source performance, the fine-tuning approach (Line 6) is slightly better than trained on the source domain only (Line 3). This is probably because the domains are highly correlated as opposed to [94], and thus training with more data on target improves the performance on source. However, fine-tuning does achieve the worst performance on source compared with other domain adaptation approaches (among Lines 5–8). Thus, I nevertheless use the terminology "catastrophic forgetting," and my research goal is still to improve IDA performance.

The main results of my approach are Lines 7 and 8. I apply the proposed progressive memory network to IDA and I fine-tune all weights. Note that on both source and target domains, the my approach outperforms the fine-tuning method alone where the memory size is not increased (comparing Lines 7 and 6). This verifies my conjecture that, if the model capacity is increased, the new domain does not override the learned knowledge much

Training domains	Performance on				
	Fic	**Gov**	**Slate**	**Tel**	**Travel**
Fic	65.41	58.87	55.83	61.39	57.35
Fic → Gov	67.55	70.82	61.04	65.07	61.90
Fic → Gov → Slate	67.04	71.55	63.29	64.66	63.53
Fic → Gov → Slate → Tel	68.46	71.10	63.39	**71.60**	61.50
Fic → Gov → Slate → Tel → Travel	**69.36**	**72.47**	**63.96**	69.74	**68.39**

Table 6.3: Dynamics of the progressive memory network for IDA with 5 domains. Upper-triangular values in gray are out-of-domain (zero-shot) performance.

in the neural network. The proposed approach is also "orthogonal" to the expansion of the vocabulary size, where target-specific words are randomly initialized and learned on the target domain. As seen, this combines well with memory expansion and yields the best performance on both source and target (Line 8).

I now compare an alternative way of increasing model capacity, i.e., expanding hidden states (Lines 9 and 10). For fair comparison, I ensure that the total number of model parameters after memory expansion is equal to the number of model parameters after hidden state expansion. Note that the performance of hidden state expansion is poor especially on the source domain, even if I fine-tune all parameters. This experiment provides empirical evidence to the theorem that expanding memory is more robust than expanding hidden states.

I also compare the results with previous work on IDA. I re-implement[5] elastic weight consolidation (EWC) [94]. It does not achieve satisfactory results in my experiments for this task. I investigate other published papers using the same method and find inconsistent results: EWC works well in some applications [221, 102] but performs poorly on others [217, 207]; [200] even report near random performance with EWC. I also re-implement the progressive neural network [162]. I use the target predictor to do inference for both source and target domains. The progressive neural network yields low performance, particularly on source, probably because the predictor is trained with only the target domain.

I measure the statistical significance of the results with one-tailed Wilcoxon's signed-rank test [203], by bootstrapping a subset of 200 samples for 10 times (with replacement). Each method is compared with Line 8, and the significance is reported with arrows: $\uparrow$ and $\Uparrow$ denote "significantly better" with $p < 0.05$ and $p < 0.01$ respectively. $\downarrow$ and $\Downarrow$ similarly denote "significantly worse." The absence of an arrow indicates that the performance

[5]My implementation is based on https://github.com/ariseff/overcoming-catastrophic

Group	Setting	Fic	Gov	Slate	Tel	Travel
Non-IDA	In-domain training	$65.41^{\Downarrow}$	$67.01^{\Downarrow}$	$59.30^{\Downarrow}$	$67.20^{\Downarrow}$	$64.70^{\Downarrow}$
	Fic + Gov + Slate + Tel + Travel (multi-task)	$\mathbf{70.60}^{\uparrow}$	**73.30**	63.80	69.15	$67.07^{\downarrow}$
IDA	Fic → Gov → Slate → Tel → Travel (F+V)	$67.24^{\downarrow}$	$70.82^{\Downarrow}$	$62.41^{\downarrow}$	$67.62^{\downarrow}$	**68.39**
	Fic → Gov → Slate → Tel → Travel (F+V+M)	*69.36*	*72.47*	**63.96**	**69.74**	**68.39**
	Fic → Gov → Slate → Tel → Travel (EWC)	$67.12^{\Downarrow}$	$68.71^{\Downarrow}$	$59.90^{\Downarrow}$	$66.09^{\Downarrow}$	$65.70^{\Downarrow}$
	Fic → Gov → Slate → Tel → Travel (Progressive)	$65.22^{\Downarrow}$	$67.87^{\Downarrow}$	$61.13^{\Downarrow}$	$66.96^{\Downarrow}$	67.90

Table 6.4: Comparing my approach with variants and previous work in the multi-domain setting. In this experiment, I use the memory-augmented RNN as the neural architecture. Italics represent best results in the IDA group. $\uparrow, \downarrow$: $p < 0.05$ and $\Uparrow, \Downarrow$: $p < 0.01$ (compared with F+V+M).

difference compared with Line 8 is statistically insignificant with p at most 0.05. The test shows that my approach is significantly better than others, both on source and target.

IDA with All Domains

Having analyzed my approach, baselines, and variants on two domains in detail, I now test the performance of IDA with multiple domains, namely, `Fic`, `Gov`, `Slate`, `Tel`, and `Travel`. In this experiment, I assume these domains come one after another, and the goal is to achieve high performance on both new and previous domains.

Table 6.3 shows the dynamics of IDA with the proposed progressive memory network. Comparing the upper-triangular values (in gray, showing out-of-domain performance) with diagonal values, we see that my approach can be quickly adapted to the new domain in an incremental fashion. Comparing lower-triangular values with the diagonal, we see that my approach does not suffer from the catastrophic forgetting problem as the performance of previous domains is gradually increasing if trained with more domains. After all data are observed, my model achieves the best performance in most domains (last row in Table 6.3), despite the incremental nature of my approach.

I now compare my approach with other baselines and variants in the multi-domain setting, shown in Table 6.4. Due to the large number of settings, I only choose a selected subset of variants from Table 6.2 for the comparison.

As seen, my approach of progressively growing memory bank achieves the same performance as fine-tuning on the last domain (both with vocabulary expansion). But for all previous 4 domains, I achieve significantly better performance. My model is comparable to multi-task learning on all domains. This provides evidence of the effectiveness for IDA with more than two domains.

# Line	Model	Trained on/by	BLEU-2 on		W2V-Sim on	
			S	T	S	T
1	RNN	S	$2.842^{\Uparrow}$	$0.738^{\Downarrow}$	$0.480^{\Downarrow}$	$0.456^{\Downarrow}$
2		T	$0.795^{\Downarrow}$	$1.265^{\Downarrow}$	$0.454^{\Downarrow}$	$0.480^{\Downarrow}$
3	RNN+ Mem	S	$\mathbf{3.074}^{\Uparrow}$	$0.712^{\Downarrow}$	$0.498^{\Downarrow}$	$0.471^{\Downarrow}$
4		T	$0.920^{\Downarrow}$	$1.287^{\Downarrow}$	$0.462^{\Downarrow}$	$0.487^{\Downarrow}$
5		S+T	$2.650^{\Uparrow}$	$0.889^{\Downarrow}$	$0.471^{\Downarrow}$	$0.462^{\Downarrow}$
6	RNN + Mem	S→T (F)	$1.210^{\Downarrow}$	$1.101^{\Downarrow}$	$0.509^{\Downarrow}$	$0.514^{\Downarrow}$
7		S→T (F+M)	$1.435^{\Downarrow}$	$1.207^{\Downarrow}$	**0.526**	0.522
8		S→T (F+M+V)	*1.637*	**1.652**	0.522	**0.525**
9		S→T (F+H)	$1.036^{\Downarrow}$	$1.606^{\downarrow}$	$0.503^{\Downarrow}$	$0.495^{\Downarrow}$
10		S→T (F+H+V)	$1.257^{\Downarrow}$	$1.419^{\Downarrow}$	$0.504^{\Downarrow}$	$0.492^{\Downarrow}$
11		S→T (EWC)	$1.397^{\Downarrow}$	$1.382^{\downarrow}$	$0.513^{\Downarrow}$	$0.514^{\Downarrow}$
12		S→T (Progressive)	$1.299^{\Downarrow}$	$1.408^{\downarrow}$	$0.502^{\Downarrow}$	$0.503^{\Downarrow}$

Table 6.5: Results on two-domain adaptation for dialogue response generation. F: Fine-tuning. V: Expanding vocabulary. H: Expanding RNN hidden states. M: My proposed method of expanding memory. I also compare with previous work elastic weight consolidation [94, EWC] and the progressive neural network [162]. $\uparrow, \downarrow$: $p < 0.05$ and $\Uparrow, \Downarrow$: $p < 0.01$ (compared with Line 8).

It should also be mentioned that multi-task learning requires training the model when data from all domains are available at the same time. It is not an *incremental* approach for domain adaptation, and thus cannot be applied to the scenarios introduced in Section 6.1. I include this setting mainly because due to curiosity about the performance of non-incremental domain adaptation.

I also compare with previous methods for IDA in Table 6.4. My method outperforms EWC [102] and the progressive neural network [162] in all domains; the results are consistent with Table 6.2.

6.4.2 Experiment II: Dialogue Generation

With promising results in text classification, I now move to text generation. I evaluate my approach on the task of dialogue response generation. Given an input text sequence, the task is to generate an appropriate output text sequence as a response in human-computer dialogue.

Dataset, Setup, and Metrics

I use the Cornell Movie Dialogs Corpus [42] as the source. It contains ∼220k message-response pairs from movie transcripts. I use a 200k-10k-10k training-validation-test split.

For the target domain, I manually construct a very small dataset to mimic the scenario where quick adaptation has to be done to a new domain with little training data. In particular, I choose a random subset of 15k message-response pairs from the Ubuntu Dialog Corpus [122], a dataset of conversations about Ubuntu. I use a 9k-3k-3k data split.

The base model is a sequence-to-sequence (Seq2Seq) neural network [185] with attention from the decoder to the encoder. I use a single-layer RNN encoder and a single-layer RNN decoder, each containing 1024 cells following [185]. I use GRUs instead of LSTM units due to efficiency concerns. The source and target vocabularies are 27k and 10k respectively.

I use separate memory banks for the encoder and decoder, since they are different RNNs. Each memory slot is 1024D, because the RNN states are 1024D. For each domain, I progressively add 1024 slots; tuning the number of slots is done in a manner similar to Experiment I. As before, I use Adam with an initial learning rate of 0.0003 and other default parameters.

Following previous work, I use BLEU-2 [53, 126] and average Word2Vec embedding similarity (W2V-Sim) [168, 225] as the evaluation metrics. BLEU-2 is the geometric mean of unigram and bigram word precision penalized by length, and correlates with human satisfaction to some extent [117]. W2V-Sim is defined as the cosine similarity between the averaged Word2Vec embeddings of the model outputs and the ground truths. Intuitively, BLEU measures hard word-level overlap between two sequences, whereas W2V-Sim measures soft similarity in a distributed semantic space.

Results

The results for dialogue response generation are shown in Table 6.5. We see that BLEU-2 and W2V similarity are not necessarily consistent. For example, the memory-augmented RNN trained solely on source achieves the best source BLEU-2, whereas the proposed progressive memory has the highest W2V cosine similarity on S. However, my model's variants (either expanding the vocabulary or not) achieve the best performance on most metrics (Lines 7 and 8). Moreover, it consistently outperforms all other IDA approaches. Following Experiment I, I conduct a statistical test to compare different methods with Line 8. The test shows that my method is significantly better than the other IDA methods. A case study is provided in Table 6.6, which shows sample outputs of my IDA model on test

prompts from the Cornell Movie Corpus (source) and the Ubuntu Dialogue Corpus (target). We see that casual prompts from the movie domain result in casual responses, whereas Ubuntu queries result in Ubuntu-related responses. With the expansion of vocabulary, my model is able to learn new words like "grub"; with progressive memory, it learns Ubuntu jargon like "network connection." This shows evidence of the success of incremental domain adaptation.

Prompt	**Response**
hi	hi
how are you?	i'm good
i'm steve. what's your name?	my name is opera
shut your mouth!	you will? what about it
i love you man	i love you too
i dont wanna talk about it	i think you do too
what are you doing?	you
i would just want to clear some things up	look thanks for your help
so how's it going with your new roomie?	it should be fine
oh yeah you're just so smart	thanks
i didn't do anything wrong did i	we don't have much experience
i just got a messed up stomach	i know, you had to do some things
what are you trying to handicap me?	this is not really who i am
what is the matter	just wanted to make sure it is easier, just fine
im scared	what do we do with it?
being a jerk doesn't make you special	i'll make sure you use the list, you might want to
that chipset is great for gaming it is very good	hmm i think it has been rather too slow for me
i kinda forgot the fix i tried a year ago	can you try the output ?
do you want me to let it autodetect	try it
i need help with grub. it can't detect ubuntu	well i can probably get you a little bit of help
how would i remaster the ubuntu install	could you have a link access to the network connection
ok i will try this solution	i believe that you did
no progress	hmm
i found this review rather convincing for linux	can you show me?
my terminal is stuck	what version is it?
which bios?	windows with new partition drive on the new computer
where can i get the latest release?	slackware is behind the times with their release schedule
when i try to restart it doesnt work	hmm you should be in the terminal are you ?
i am rebooting now	so your issue is a great work cd, sometimes i get some
i get an installation error	i am getting an expert for it
what does the kernel-package do?	you will need to get it to see if there is a setting
what directory should i be in?	you should be able to install the grub cd at the drive

Table 6.6: Sample outputs of the proposed IDA model S→T (F+M+V) from Table 5.

In general, the evaluation of dialogue systems is noisy due to the lack of appropriate

metrics, as discussed previously [117]. Nevertheless, the experiment provides additional evidence of the effectiveness of the proposed approach. It also highlights the model's viability for both classification and generation tasks.

6.5 Conclusion

I have proposed a progressive memory network for incremental domain adaptation (IDA). I augment an RNN with an attention-based memory bank. During IDA, I add new slots to the memory bank and tune all parameters by back-propagation. Empirically, the progressive memory network does not suffer from the catastrophic forgetting problem as in naïve fine-tuning. My intuition is that the new memory slots increase the neural network's model capacity, and thus, there is less overriding of the existing network due to new knowledge. Compared with expanding hidden states, the proposed progressive memory bank provides a more robust way of increasing model capacity, shown by both a theorem and experiments. The proposed approach also outperforms previous work for IDA, including elastic weight consolidation (EWC) and the original progressive neural network.

Chapter 7

Conclusion

In this thesis, I have studied two facets of human-likeness in machines: the ability to perceive and convey human emotions, and the ability to understand and generate human language. I explore the challenges faced by existing HCI systems, in terms of high user engagement, fluency and coherence of interaction, adaptability to unseen situations and being able to relate to users on an affective level. I work towards mitigating some of these issues by enhancing the emotional and linguistic prowess of text-based HCI systems.

Concretely, I develop a Monte-Carlo planning algorithm for *BayesAct*, a large POMDP model of affect, that has continuous states, actions and observations. I use this algorithm to produce affect-aware agents in two-person social dilemmas and health-care assistance. I further explore the efficacy of affective computing within language classification and generation. I investigate how the principles of *BayesAct* can be used to generate affect-sensitive responses in a neural-network dialogue generation system. I also develop affective loss functions, word embeddings, neural sequence decoding methods and imitation learning techniques to produce human-like and affect-rich responses in a dialogue system. Finally, I present a neural domain adaptation method to help transfer knowledge from one task to another for language classification and generation.

This work opens many new research directions for affective natural language processing. I have presented some limitations of the proposed techniques in Sections 4.5 and 5.5. They serve as important guidelines for future research. I highlight some additional ideas below.

ACT and *BayesAct* are promising affect models that need more exploration within the field of NLP. I showed some basic ways to integrate ACT with neural conversational models. However, *BayesAct* is more suited to this task, since it provides a framework to

combine affective and non-affective goals in a planning model. Furthermore, as I demonstrated in Chapter 3, *BayesAct* is suitable for modeling complex human behaviours such as manipulation. In a similar way, models and experiments could be designed to model bullying or harassment.

Another useful research direction is to develop affective representations of sentences. The EPA and VAD lexicons used in this thesis provide word-level affect features, and it is not clear how they should be combined to get sentence-level affect. It would be interesting to see how recent advances in sentence representation learning, such as USE [32], can be applied here. A related idea is affect disentanglement in the latent text representation space. Some recent studies present autoencoder-based latent disentanglement techniques, which can transform positive texts into negative texts, and vice versa [86]. However, the model should have more fine-grained control over affect in the generated text, i.e. it should be able to control other implicit affect qualities.

Lastly, this thesis uses recurrent neural networks as the base neural architecture for model implementation and evaluation, together with Word2Vec or GloVe embeddings for words. However, the proposed techniques can be used with more efficient and parallelizable architectures like the Transformer [194], contextualized word embeddings (ELMO [144]) and pre-trained models (BERT [49]). It would be interesting to see whether migrating to these architectures improves the proposed models' affect quality and performance.

References

[1] K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. App.*, 10:174–205, 1965.

[2] Mohamed Abdalla, Magnus Sahlgren, and Graeme Hirst. Enriching word embeddings with a regressor instead of labeled corpora. In *AAAI*, pages 6188–6195, 2019.

[3] David Abel, John Salvatier, Andreas Stuhlmüller, and Owain Evans. Agent-agnostic human-in-the-loop reinforcement learning. *arXiv preprint arXiv:1701.04079*, 2017.

[4] George A. Akerlof and Rachel E. Kranton. Economics and identity. *The Quarterly Journal of Economics*, 115(3):715–753, 2000.

[5] Ben Alderson-Day and Charles Fernyhough. Inner speech: development, cognitive functions, phenomenology, and neurobiology. *Psychological bulletin*, 141(5):931, 2015.

[6] Areej Alhothali and Jesse Hoey. Semi-supervised affective meaning lexicon expansion using semantic and distributed word representations. *arXiv preprint arXiv:1703.09825*, 2017.

[7] Christopher Amato and Frans A. Oliehoek. Scalable planning and learning for multiagent pomdps. In *AAAI*, January 2015.

[8] Dimitrios Antos and Avi Pfeffer. Using emotions to enhance decision-making. In *IJCAI*, Barcelona, Spain, 2011.

[9] Nabiha Asghar and Jesse Hoey. Monte-Carlo planning for socially aligned agents using Bayesian affect control theory. Technical Report CS-2014-21, University of Waterloo School of Computer Science, 2014.

[10] Nabiha Asghar and Jesse Hoey. Intelligent affect: Rational decision making for socially aligned agents. In *UAI*, pages 12–16, 2015.

[11] Nabiha Asghar, Lili Mou, Kira A Selby, Kevin D Pantasdo, Pascal Poupart, and Xin Jiang. Progressive memory banks for incremental domain adaptation. In *ICLR*, 2020.

[12] Nabiha Asghar, Pascal Poupart, Jesse Hoey, Xin Jiang, and Lili Mou. Affective neural response generation. In *ECIR*, pages 154–166. Springer, 2018.

[13] Nabiha Asghar, Pascal Poupart, Xin Jiang, and Hang Li. Deep active learning for dialogue generation. In *Joint Conference on Lexical and Computational Semantics*, pages 78–83, 2017.

[14] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[16] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A. Ngo. Monte-Carlo value iteration for continuous-state POMDPs. In *Workshop on the Algorithmic Foundations of Robotics*, pages 175–191, 2010.

[17] Tina Balke, Célia da Costa Pereira, Frank Dignum, Emiliano Lorini, Antonino Rotolo, Wamberto Vasconcelos, and Serena Villata. Norms in MAS: Definitions and Related Concepts. In Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leendert W. N. van der Torre, editors, *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*, pages 1–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[18] Rafael E Banchs and Haizhou Li. Iris: a chat-oriented dialogue system based on the vector space model. In *ACL*, pages 37–42, 2012.

[19] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, 2005.

[20] Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013.

[21] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[22] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *JMLR*, 3(Feb):1137–1155, 2003.

[23] Mark S Boddy, Johnathan Gohde, Thomas Haigh, and Steven A Harp. Course of action generation for cyber security using classical planning. In *ICAPS*, pages 12–21, 2005.

[24] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–94, 1999.

[25] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *CoNLL*, pages 10–21, 2016.

[26] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Solving continuous pomdps: Value iteration with incremental learning of an efficient space representation. In *ICML*, 2013.

[27] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[28] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2383–2388, 2002.

[29] Zoraida Callejas, David Griol, and Ramón López-Cózar. Predicting user mental states in spoken dialogue systems. *EURASIP J. Advances in Signal Processing*, 2011(1):6, 2011.

[30] Anthony Cassandra, Michael L Littman, and Nevin L Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI*, pages 54–61, 1997.

[31] Fabio Catania, Nicola Di Nardo, Franca Garzotto, and Daniele Occhiuto. Emoty: An emotionally sensitive conversational agent for people with neurodevelopmental disorders. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[32] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[33] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *arXiv preprint arXiv:1812.00420*, 2018.

[34] M Keith Chen. The effect of language on economic behavior: Evidence from savings rates, health behaviors, and retirement assets. *American Economic Review*, 103(2):690–731, 2013.

[35] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. In *ICASSP*, pages 4774–4778, 2018.

[36] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the ACL*, 4:357–370, 2016.

[37] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[38] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[39] Kenneth Mark Colby, Sylvia Weber, and Franklin Dennis Hilf. Artificial paranoia. *Artificial Intelligence*, 2(1):1–25, 1971.

[40] Heriberto Cuayáhuitl, Seunghak Yu, Ashley Williamson, and Jacob Carse. Deep reinforcement learning for multi-domain dialogue systems. *Deep Reinforcement Learning Workshop, NIPS*, 2016.

[41] Antonio R. Damasio. *Descartes' error: Emotion, reason, and the human brain*. Putnam's sons, 1994.

[42] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.

In *Workshop on Cognitive Modeling and Computational Linguistics*. Association for Computational Linguistics, 2011.

[43] Rajarshi Das, Manzil Zaheer, Siva Reddy, and Andrew McCallum. Question answering on knowledge bases and text using universal schema and memory networks. In *ACL*, pages 358–365, 2017.

[44] Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. *arXiv preprint arXiv:1906.01076*, 2019.

[45] Celso M De Melo, Peter Carnevale, and Jonathan Gratch. The influence of emotions in embodied agents on human decision-making. In *International Conference on Intelligent Virtual Agents*, pages 357–370. Springer, 2010.

[46] Celso M. de Melo, Peter Carnevale, Stephen Read, Dimitrios Antos, and Jonathan Gratch. Bayesian model of the social effects of emotion in decision-making in multiagent systems. In *AAMAS*, Valencia, Spain, 2012.

[47] Nick Degens, Gert Jan Hofstede, John McBreen, Adrie Beulens, Samuel Mascarenhas, Nuno Ferreira, Ana Paiva, and Frank Dignum. Creating a world for sociocultural agents. In Tibor Bosse, Joost Broekens, Joao Dias, and Janneke van der Zwaan, editors, *Emotion Modeling: Towards Pragmatic Computational Models of Affective Processes*, number 8750 in Lecture Notes in Artificial Intelligence. Springer, 2014.

[48] Marc Peter Deisenroth and Jan Peters. Solving nonlinear continuous state-action-observation pomdps for mechanical systems with gaussian noise. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, 2012.

[49] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[50] Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, 2001.

[51] Tomasz Dryjański, Paweł Bujnowski, Hyungtak Choi, Katarzyna Podlaska, Kamil Michalski, Katarzyna Beksa, and Paweł Kubik. Affective natural language generation by phrase insertion. In *IEEE International Conference on Big Data*, pages 4876–4882, 2018.

[52] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[53] Mihail Eric and Christopher D Manning. Key-value retrieval networks for task-oriented dialogue. *arXiv preprint arXiv:1705.05414*, 2017.

[54] Caitlin M Fausey and Lera Boroditsky. Who dunnit? cross-linguistic differences in eye-witness memory. *Psychonomic bulletin & review*, 18(1):150–157, 2011.

[55] Denis Fedorenko, Nikita Smetanin, and Artem Rodichev. Avoiding echo-responses in a retrieval-based conversation system. In *Conference on Artificial Intelligence and Natural Language*, pages 91–97. Springer, 2018.

[56] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *EMNLP*, pages 1615–1625, 2017.

[57] John G. Fennell and Roland J. Baddeley. Reward is assessed in three dimensions that correspond to the semantic differential. *PLoS One*, 8(2: e55588), 2013.

[58] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

[59] Jeremiah T. Folsom-Kovarik, Gita Sukthankar, and Sae Schatz. Tractable POMDP representations for intelligent tutoring systems. *ACM Trans. Intell. Syst. Technol.*, 4(2):29:1–29:22, April 2013.

[60] Pascale Fung, Dario Bertero, Peng Xu, Ji Ho Park, Chien-Sheng Wu, and Andrea Madotto. Empathetic dialog systems. In *LREC*, 2018.

[61] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 17(1):2096–2030, 2016.

[62] Asma Ghandeharioun, Daniel McDuff, Mary Czerwinski, and Kael Rowan. Emma: An emotionally intelligent personal assistant for improving wellbeing. *arXiv preprint arXiv:1812.11423*, 2018.

[63] Sayan Ghosh, Mathieu Chollet, Eugene Laksana, Louis-Philippe Morency, and Stefan Scherer. Affect-LM: A neural language model for customizable affective text generation. In *ACL*, 2017.

[64] Kevin Gimpel, Dhruv Batra, Chris Dyer, Gregory Shakhnarovich, and Virginia Tech. A systematic exploration of diversity in machine translation. In *EMNLP*, pages 1100–1111, 2013.

[65] Piotr Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *JAIR*, 24:49–79, 2005.

[66] Erving Goffman. *Behavior in Public Places*. The Free Press, New York, 1963.

[67] Carla Gordon, Anton Leuski, Grace Benn, Eric Klassen, Edward Fast, Matt Liewer, Arno Hartholt, and David Traum. Primer: An emotionally aware virtual agent. In *Proceedings of the IUI Workshop on User-Aware Conversational Agents*, 2019.

[68] Jonathan Gratch and Stacy Marsella. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269 – 306, 2004.

[69] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[70] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[71] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *ACL*, pages 1631–1640, 2016.

[72] Arthur Guez, David Silver, and Peter Dayan. Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search. *JAIR*, 48, 2013.

[73] Takayuki Hasegawa, Nobuhiro Kaji, Naoki Yoshinaga, and Masashi Toyoda. Predicting and eliciting addressees emotion in online dialogue. In *ACL (Volume 1: Long Papers)*, pages 964–972, 2013.

[74] David R. Heise. *Expressive Order: Confirming Sentiments in Social Actions*. Springer, 2007.

[75] David R. Heise. *Surveying Cultures: Discovering Shared Conceptions and Sentiments*. Wiley, 2010.

[76] David R. Heise. Modeling interactions in small groups. *Social Psychology Quarterly*, 76:52–72, 2013.

[77] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[79] Jesse Hoey, Craig Boutilier, Pascal Poupart, Patrick Olivier, Andrew Monk, and Alex Mihailidis. People, sensors, decisions: Customizable and adaptive technologies for assistance in healthcare. *ACM Trans. Interact. Intell. Syst.*, 2(4):20:1–20:36, January 2012.

[80] Jesse Hoey and Tobias Schröder. Bayesian affect control theory of self. In *AAAI*, pages 529–536, 2015.

[81] Jesse Hoey, Tobias Schröder, and Areej Alhothali. Bayesian affect control theory. In *Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 166–172, 2013.

[82] Chieh-Yang Huang, Tristan Labetoulle, Ting-Hao Huang, Yi-Pei Chen, Hung-Chen Chen, Vallari Srivastava, and Lun-Wei Ku. Moodswipe: A soft keyboard that suggests messagebased on user-specified emotions. In *EMNLP: System Demonstrations*, pages 73–78, 2017.

[83] Huaibo Huang, Ran He, Zhenan Sun, Tieniu Tan, et al. Introvae: Introspective variational autoencoders for photographic image synthesis. In *NeurIPS*, pages 52–63, 2018.

[84] Liang Huang, Kai Zhao, and Mingbo Ma. When to finish? optimal beam search for neural text generation (modulo beam size). In *EMNLP*, pages 2134–2139, 2017.

[85] Natasha Jaques, Sara Taylor, Akane Sano, and Rosalind Picard. Multimodal autoencoder: A deep learning approach to filling in missing sensor data and enabling better mood prediction. In *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 202–208, 2017.

[86] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for text style transfer. *arXiv preprint arXiv:1808.04339*, 2018.

[87] Mark Johnson. How the statistical revolution changes (computational) linguistics. In *EACL Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 3–11, Athens, Greece, March 2009.

[88] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[89] Daniel Kahneman. *Thinking, Fast and Slow*. Doubleday, 2011.

[90] Paul Kay and Willett Kempton. What is the sapir-whorf hypothesis? *American anthropologist*, 86(1):65–79, 1984.

[91] Dahyun Kim, Jihwan Bae, Yeonsik Jo, and Jonghyun Choi. Incremental learning with maximum entropy regularization: Rethinking forgetting and intransigence. *arXiv preprint arXiv:1902.00829*, 2019.

[92] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[93] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[94] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[95] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *NAACL HLT-Volume 1*, pages 48–54, 2003.

[96] Xiang Kong, Bohan Li, Graham Neubig, Eduard Hovy, and Yiming Yang. An adversarial approach to high-quality, sentiment-controlled neural dialogue generation. *arXiv preprint arXiv:1901.07129*, 2019.

[97] Barry Kort, Rob Reilly, and Rosalind W Picard. An affective model of interplay between emotions and learning: Reengineering educational pedagogy-building a learning companion. In *IEEE International Conference on Advanced Learning Technologies*, pages 43–46, 2001.

[98] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[99] Andrew Kusiak and Mingyuan Chen. Expert systems for planning and scheduling manufacturing systems. *European Journal of Operational Research*, 34(2):113–130, 1988.

[100] Man Lan, Zhihua Zhang, Yue Lu, and Ju Wu. Three convolutional neural network-based models for learning sentiment word vectors towards sentiment analysis. In *IJCNN*, pages 3172–3179, 2016.

[101] Joseph LeDoux. *The emotional brain: the mysterious underpinnings of emotional life*. Simon and Schuster, New York, 1996.

[102] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, pages 4652–4662, 2017.

[103] Taekhee Lee and Young J. Kim. GPU-based motion planning under uncertainties using POMDP. In *Proc. Intl Conf. on Robotics and Automation (ICRA)*, Karlsruhe, DE, 2013.

[104] Anton Leuski and David Traum. NPCEditor: Creating virtual human dialogue using information retrieval techniques. *Ai Magazine*, 32(2):42–56, 2011.

[105] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *NAACL-HLT*, pages 110–119, 2016.

[106] Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. In *ACL (Volume 1: Long Papers)*, pages 994–1003, 2016.

[107] Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Dialogue learning with human-in-the-loop. *ICLR*, 2017.

[108] Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Learning through dialogue interactions by asking questions. In *ICLR*, 2017.

[109] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

[110] Jiwei Li, Will Monroe, Alan Ritter, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. In *EMNLP*, pages 1192–1202, 2016.

[111] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. Investigation of language understanding impact for reinforcement learning based dialogue systems. *arXiv preprint arXiv:1703.07055*, 2017.

[112] Xuijun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*, 2017.

[113] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE TPAMI*, 40(12):2935–2947, 2018.

[114] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out (ACL Workshop)*, pages 74–81. ACL, 2004.

[115] Luyuan Lin, Stephen Czarnuch, Aarti Malhotra, Lifei Yu, Tobias Schröder, and Jesse Hoey. Affectively aligned cognitive assistance using bayesian affect control theory. In *Proc. of International Workconference on Ambient Assisted Living (IWAAL)*, pages 279–287, Belfast, UK, December 2014. Springer.

[116] Christine Laetitia Lisetti and Piotr Gmytrasiewicz. Can a rational agent afford to be affectless? a formal approach. *Applied Artificial Intelligence*, 16(7-8):577–609, 2002.

[117] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP*, pages 2122–2132, 2016.

[118] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. In *ACL*, pages 1–10, 2017.

[119] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, pages 6467–6476, 2017.

[120] W. S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[121] Ryan Lowe, Michael Noseworthy, Iulian Vlad Serban, Nicolas Angelard-Gontier, Yoshua Bengio, and Joelle Pineau. Towards an automatic Turing test: Learning to evaluate dialogue responses. In *ACL*, pages 1116–1126, July 2017.

[122] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The Ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *SIGDIAL*, pages 285–294, 2015.

[123] Nurul Lubis, Sakriani Sakti, Koichiro Yoshino, and Satoshi Nakamura. Eliciting positive emotion through affect-sensitive dialogue response generation: A neural network approach. In *AAAI*, 2018.

[124] Owen Macindoe, Leslie Pack Kaelbling, , and Tomás Lozano-Pérez. Pomcop: Belief space planning for sidekicks in cooperative games. In *AIIDE*, 2012.

[125] Neil. J. MacKinnnon and Dawn T. Robinson. 25 years of research in affect control theory. *Advances in Group Processing*, 31, 2014.

[126] Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *ACL*, pages 1468–1478, 2018.

[127] Aarti Malhotra, Lifei Yu, Tobias Schröder, and Jesse Hoey. An exploratory study into the use of an emotionally aware cognitive assistant. In *AAAI Workshop: Artificial Intelligence Applied to Assistive Technologies and Smart Environments*, 2015.

[128] Robert P. Marinier III and John E. Laird. Emotion-driven reinforcement learning. In *Proc. of 30th Annual Meeting of the Cognitive Science Society*, pages 115–120, Washington, D.C., 2008.

[129] Pierre-Emmanuel Mazaré, Samuel Humeau, Martin Raison, and Antoine Bordes. Training millions of personalized dialogue agents. In *EMNLP*, pages 2775–2779, 2018.

[130] Gary McKeown, Michel Valstar, Roddy Cowie, Maja Pantic, and Marc Schroder. The semaine database: Annotated multimodal records of emotionally colored conversations between a person and a limited agent. *IEEE Transactions on Affective Computing*, 3(1):5–17, 2012.

[131] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[132] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *EMNLP*, pages 1400–1409, 2016.

[133] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in NLP applications? In *EMNLP*, pages 479–489, 2016.

[134] Lili Mou, Yiping Song, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. Sequence to backward and forward sequences: A content-introducing approach to generative short-text conversation. In *COLING*, pages 3349–3358, 2016.

[135] Ryo Nakamura, Katsuhito Sudoh, Koichiro Yoshino, and Satoshi Nakamura. Another diversity-promoting objective function for neural dialogue generation. *arXiv preprint arXiv:1811.08100*, 2018.

[136] Charles E. Osgood, William H. May, and Murray S. Miron. *Cross-Cultural Universals of Affective Meaning*. University of Illinois Press, 1975.

[137] Endang Wahyu Pamungkas. Emotionally-aware chatbots: A survey. *arXiv preprint arXiv:1906.09774*, 2019.

[138] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[139] Ji Ho Park. Finding good representations of emotions for text classification. *arXiv preprint arXiv:1808.07235*, 2018.

[140] Yookoon Park, Jaemin Cho, and Gunhee Kim. A hierarchical latent structure for variational conversation modeling. In *NAACL*, pages 1792–1801, 2018.

[141] James W Pennebaker, Martha E Francis, and Roger J Booth. *Linguistic Inquiry and Word Count*. Erlbaum Publishers, 2001.

[142] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[143] Fernando Perez-Cruz. Kullback-Leibler divergence estimation of continuous distributions. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1666–1670, July 2008.

[144] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, pages 2227–2237, 2018.

[145] Rosalind W. Picard. *Affective Computing*. MIT Press, Cambridge, MA, 1997.

[146] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003.

[147] Johannes Pittermann, Angela Pittermann, and Wolfgang Minker. Emotion recognition and adaptation in spoken dialogue systems. *Int. J. Speech Technology*, 13(1):49–60, 2010.

[148] Robert Plutchik. A general psychoevolutionary theory of emotion. In *Theories of emotion*, pages 3–33. Elsevier, 1980.

[149] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-based value iteration for continuous POMDPs. *JMLR*, 7:2329–2367, 2006.

[150] Pascal Poupart. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes.* PhD thesis, University of Toronto, 2005.

[151] Helmut Prendinger and Mitsuru Ishizuka. The empathic companion: A character-based interface that addresses users'affective states. *Applied Artificial Intelligence*, 19(3-4):267–285, 2005.

[152] Qiao Qian, Minlie Huang, Haizhou Zhao, Jingfang Xu, and Xiaoyan Zhu. Assigning personality/profile to a chatting machine for coherent conversation generation. In *IJCAI*, pages 4279–4285, 2018.

[153] Ashwin Rajadesingan, Reza Zafarani, and Huan Liu. Sarcasm detection on twitter: A behavioral modeling approach. In *WSDM*, pages 97–106, 2015.

[154] Hannah Rashkin, Eric Michael Smith, Margaret Li, and Y-Lan Boureau. I know the feeling: Learning to converse with empathy. *arXiv preprint arXiv:1811.00207*, 2018.

[155] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, pages 2001–2010, 2017.

[156] Yafeng Ren, Yue Zhang, Meishan Zhang, and Donghong Ji. Improving twitter sentiment classification using topic-enriched multi-prototype word embeddings. In *AAAI*, pages 3038–3044, 2016.

[157] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[158] Jennifer Robison, Scott McQuiggan, and James Lester. Evaluating the consequences of affective feedback in intelligent tutoring systems. In *Proc. Int. Conf. Affective Comput. and Intell. Interaction and Workshops*, pages 1–6, 2009.

[159] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *JAIR*, 32:663–704, 2008.

[160] James A. Russell. Core affect and the psychological construction of emotion. *Psychological Review*, 110(1):145–172, 2003.

[161] James A. Russell and Albert Mehrabian. Evidence for a three-factor theory of emotions. *Journal of research in Personality*, 11(3):273–294, 1977.

[162] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[163] Klaus R. Scherer, Tanja Banziger, and Etienne Roesch. *A Blueprint for Affective Computing*. Oxford University Press, 2010.

[164] Wolfgang Scholl. The socio-emotional basis of human interaction and communication: How we construct our social world. *Social Science Information*, 52:3 – 33, 2013.

[165] Tobias Schröder and Wolfgang Scholl. Affective dynamics of leadership: An experimental test of affect control theory. *Social Psychology Quarterly*, 72:180–197, 2009.

[166] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, pages 4535–4544, 2018.

[167] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*, pages 3295–3301, 2017.

[168] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*, pages 3295–3301, 2017.

[169] Julian Vlad Serban, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron Courville. Multiresolution recurrent neural networks: An application to dialogue response generation. *AAAI*, 2017.

[170] Julian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, 2016.

[171] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. In *ACL-IJCNLP*, pages 1577–1586, 2015.

[172] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *AAMAS*, 27(1):1–51, 2013.

[173] Yuanlong Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *EMNLP*, pages 2210–2219, 2017.

[174] Xiaoyu Shen, Hui Su, Yanran Li, Wenjie Li, Shuzi Niu, Yang Zhao, Akiko Aizawa, and Guoping Long. A conditional variational framework for dialog generation. In *ACL (Volume 2: Short Papers)*, pages 504–509, 2017.

[175] Xiaoyu Shen, Hui Su, Shuzi Niu, and Vera Demberg. Improving variational encoder-decoders in dialogue generation. In *AAAI*, pages 5456–5463, 2018.

[176] Weiyan Shi and Zhou Yu. Sentiment adaptive end-to-end dialog systems. In *ACL (Volume 1: Long Papers)*, pages 1509–1519, 2018.

[177] Mei Si, Stacy Marsella, and David Pynadath. Modeling appraisal in theory of mind reasoning. *AAMAS*, 20(1):14–31, 2010.

[178] David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS) 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[179] Herbert A. Simon. Motivational and emotional controls of cognition. *Psychological Review*, 74:29–39, 1967.

[180] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *NIPS*, pages 3483–3491, 2015.

[181] Edward J Sondik. *The Optimal Control of Partially Observable Markov Decision Processes.* PhD thesis, Stanford University, 1971.

[182] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In *NAACL*, pages 196–205, 2015.

[183] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, 24:195–220, 2005.

[184] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *NIPS*, pages 2440–2448, 2015.

[185] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[186] Henri Tajfel and John C. Turner. An integrative theory of intergroup conflict. In Stephen Worchel and William Austin, editors, *The social psychology of intergroup relations*. Brooks/Cole, Monterey, CA, 1979.

[187] Duyu Tang, Furu Wei, Bing Qin, Nan Yang, Ting Liu, and Ming Zhou. Sentiment embeddings with applications to sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering*, 28(2):496–509, 2015.

[188] Chongyang Tao, Lili Mou, Dongyan Zhao, and Rui Yan. Ruber: An unsupervised method for automatic evaluation of open-domain dialog systems. In *AAAI*, pages 722–729, 2018.

[189] Dante I Tapia and Juan M Corchado. An ambient intelligence based multi-agent system for alzheimer health care. *International Journal of Ambient Computing and Intelligence (IJACI)*, 1(1):15–26, 2009.

[190] Sara Ann Taylor, Natasha Jaques, Ehimwenma Nosakhare, Akane Sano, and Rosalind Picard. Personalized multitask learning for predicting tomorrow's mood, stress, and health. *IEEE Transactions on Affective Computing*, 2017.

[191] Brian Thompson, Jeremy Gwinnup, Huda Khayrallah, Kevin Duh, and Philipp Koehn. Overcoming catastrophic forgetting during domain adaptation of neural machine translation. In *NAACL*, pages 2062–2068, 2019.

[192] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

[193] Ankit Vadehra. Creating an emotion responsive dialogue system. Master's thesis, University of Waterloo, 2018.

[194] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[195] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *AAAI*, 2018.

[196] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[197] Jin Wang, Liang-Chih Yu, K. Robert Lai, and Xuejie Zhang. Dimensional sentiment analysis using a regional cnn-lstm model. In *ACL*, pages 225–230, 2016.

[198] Amy Beth Warriner, Victor Kuperman, and Marc Brysbaert. Norms of valence, arousal, and dominance for 13,915 English lemmas. *Behavior Research Methods*, 45(4), 2013.

[199] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[200] Shixian Wen and Laurent Itti. Overcoming catastrophic forgetting problem by weight consolidation and long-term memory. *arXiv preprint arXiv:1805.07441*, 2018.

[201] Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*, pages 1711–1721, 2015.

[202] Jason Weston. Dialog-based language learning. *NIPS*, 2016.

[203] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[204] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, pages 1112–1122, 2018.

[205] Jason D Williams, Pascal Poupart, and Steve Young. Factored partially observable markov decision processes for dialogue management. In *Proc. IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 76–82, 2005.

[206] Jason D. Williams and Steve Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422, 2006.

[207] Chenshen Wu, Luis Herranz, Xialei Liu, Yaxing Wang, Joost van de Weijer, and Bogdan Raducanu. Memory replay GANs: learning to generate images from new categories without forgetting. *arXiv preprint arXiv:1809.02058*, 2018.

[208] Yu Wu, Wei Wu, Zhoujun Li, and Ming Zhou. Learning matching models with weak supervision for response selection in retrieval-based chatbots. In *ACL*, pages 420–425, 2018.

[209] Ruobing Xie, Zhiyuan Liu, Rui Yan, and Maosong Sun. Neural emoji recommendation in dialogue systems. *arXiv preprint arXiv:1612.04609*, 2016.

[210] Chen Xing, Wei Wu, Yu Wu, Jie Liu, Yalou Huang, Ming Zhou, and Wei-Ying Ma. Topic aware neural response generation. In *AAAI*, pages 3351–3357, 2017.

[211] Chen Xing, Wei Wu, Yu Wu, Ming Zhou, Yalou Huang, and Wei-Ying Ma. Hierarchical recurrent attention network for response generation. *arXiv preprint arXiv:1701.07149*, 2017.

[212] Jiaolong Xu, Sebastian Ramos, David Vázquez, Antonio M López, and D Ponsa. Incremental domain adaptation of deformable part-based models. In *BMVC*, 2014.

[213] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *NeurIPS*, pages 899–908. 2018.

[214] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057, 2015.

[215] Zhao Yan, Nan Duan, Junwei Bao, Peng Chen, Ming Zhou, Zhoujun Li, and Jianshe Zhou. DocChat: An information retrieval approach for chatbot engines using unstructured documents. In *ACL*, 2016.

[216] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Oracle: Order robust adaptive continual learning. *arXiv preprint arXiv:1902.09432*, 2019.

[217] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *ICLR*, 2018.

[218] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *Computational Intelligence Magazine*, 13(3):55–75, 2018.

[219] Jianfei Yu, Minghui Qiu, Jing Jiang, Jun Huang, Shuangyong Song, Wei Chu, and Haiqing Chen. Modelling domain relationships for transfer learning on retrieval-based question answering systems in e-commerce. In *WSDM*, pages 682–690, 2018.

[220] Zhou Yu, Ziyu Xu, Alan W Black, and Alexander Rudnicky. Strategy and policy learning for non-task-oriented conversational systems. In *SIGDIAL*, pages 404–412, 2016.

[221] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pages 3987–3995, 2017.

[222] Rui Zhang and Zhenyu Wang. Learning to converse emotionally like humans: A conditional variational approach. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 98–109, 2018.

[223] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? In *ACL*, pages 2204–2213, 2018.

[224] Wei-Nan Zhang, Qingfu Zhu, Yifa Wang, Yanyan Zhao, and Ting Liu. Neural personalized response generation as domain adaptation. *WWW*, pages 1–20, 2017.

[225] Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. Generating informative and diverse conversational responses via adversarial information maximization. *arXiv preprint arXiv:1809.05972*, 2018.

[226] Zheng Zhang, Minlie Huang, Zhongzhou Zhao, Feng Ji, Haiqing Chen, and Xiaoyan Zhu. Memory-augmented dialogue management for task-oriented dialogue systems. *arXiv preprint arXiv:1805.00150*, 2018.

[227] Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. In *ACL*, pages 654–664, 2017.

[228] Yinhe Zheng, Guanyi Chen, Minlie Huang, Song Liu, and Xuan Zhu. Personalized dialogue generation with diversified traits. *arXiv preprint arXiv:1901.09672*, 2019.

[229] Hao Zhou, Minlie Huang, Tianyang Zhang, Xiaoyan Zhu, and Bing Liu. Emotional chatting machine: Emotional conversation generation with internal and external memory. *arXiv preprint arXiv:1704.01074*, 2017.

[230] Li Zhou, Jianfeng Gao, Di Li, and Heung-Yeung Shum. The design and implementation of xiaoice, an empathetic social chatbot. *arXiv preprint arXiv:1812.08989*, 2018.

[231] Xianda Zhou and William Yang Wang. Mojitalk: Generating emotional responses at scale. In *ACL (Volume 1: Long Papers)*, pages 1128–1137, 2018.

APPENDICES

Appendix A

POMCP-C: Full experiments with Prisoner's Dilemma

I present full results for the prisoner's dilemma experiments. Each experiment is described with a table showing the mean and median reward gathered over 10 sets of 20 games, as well as the mean and median over the last 10 games (10 times). A figure then shows the average means and medians per game. Solid lines with markers show the means (over 10 tests) for agent (blue) and client (red). Dashed lines in blue and red show one standard deviation away (above and below). The thick solid lines show the medians. The last two plots in each figure show the results from associated table in plot form.

In all the following examples, I have assumed that *agent* and *client* both start with a distribution over two identities: *friend* (EPA:$\{2.75, 1.88, 1.38\}$) and *scrooge* (EPA:$\{$-2.15, -0.21, -0.54$\}$), with probabilities of 0.8 and 0.2, respectively. The social coordination bias models the propositional actions (of *cooperate* and *defect*) as having sentiments close to *collaborate with* (EPA:$\{1.44, 1.11, 0.61\}$) and *abandon* (EPA:$\{$-2.28, -0.48, -0.84$\}$), respectively. These sentiments were chosen for my experiments because they corresponded to my intuitions about playing the prisoner's dilemma game. Changing to other, similar, sentiments for identities and actions would result in slightly different results, but qualitatively the same.

Tables A.5-A.12 and Figures A.1-A.8 show the results with a discount factor of $\gamma = 0.9$, while Tables A.13-A.20 and Figures A.9-A.16 show the results with a discount factor of $\gamma = 0.99$.

Recall that the strategies played by *client* are:

1. **(same)**: plays with the same timeout as *agent* $t_c = t_a$
2. **(1.0)**: plays with a timeout of $t_c = 1s$
3. **(co)**: always cooperates
4. **(de)**: always defects
5. **(to)**: two-out, cooperates twice, then always defects
6. **(tt)**: tit-for-tat, starts by cooperating, then always repeats the last action of the *agent*
7. **(t2)**: tit-for-two-tat, starts by cooperating, then defects if the other agent defects twice in a row
8. **(2t)**: two-tit-for-tat, starts by cooperating, then cooperates if the other agent cooperates twice in a row

There are many things going on in these graphs, here I draw attention to some of the most interesting behaviours. Below I refer to figure numbers only, but each figure is accompanied by a table on the same page with the mean/median results that are shown in the last two figures (bottom right).

- Figures A.8 and A.16 show two agents with the same planning resources (POMCP-C timeout, t_a and t_c), but with discounts of $\gamma = 0.9$ and $\gamma = 0.99$, respectively. We can see that with $\gamma = 0.99$, the agents cooperate until $t_a = t_c = 60s$, at which point they start defecting now and again. Agents are getting tempted by short-term rewards, and this effect somewhat goes away above $t_a = t_c = 120s$. With $\gamma = 0.9$ however (more discounting of the future), we see that defections start at about $t_a = t_C = 10s$, and cause massive disruption leading to mutual defection. This is an example of short-term thinking leading to sub-optimal decisions in social dilemmas.
- Figures A.1 and A.9 show *agent* playing against *client* who always cooperates **(co)**. With very short timeouts (less than $10s$), and more discounting ($\gamma = 0.9$), we see that *agent* starts by cooperating, but then starts to defect after about 12 games. It has become confident that *client* is a good person that can be taken advantage of in the short term. With more than $t_a = 30s$ timeout, *agent* starts defecting by

the second game most of the time. By $t_a = 120s$, this is all the time. With less discounting, though ($\gamma = 0.99$), we see that a small amount of defection starts at short timeouts, but that cooperation is mostly maintained until the last game. The agent sometimes tries defection early, but generally persists with cooperation. At high timeouts $t_a = 120s$, we again see defection coming in, but less than with the lower discount factor.

- Figures A.2 and A.10 show *agent* playing against *client* who always defects **(de)**. Here, with more discounting, *agent* rapidly starts defecting. With less discount, *agent* continues to try to cooperate with *client*, but these efforts die off as timeout increases. In this case, *agent* sees the long-term possibility that he can *reform* the *client*, who is behaving like a *scrooge*.

- Figures A.3 and A.11 show *agent* playing against *client* who plays two-out **(to)**. We see a similar pattern to the last case here, with *agent* attempting to cooperate for even longer at the start, because he gets "fooled" by the first two cooperations of *client*.

- Figures A.4-A.6 and A.12-A.6 show *agent* playing against *client* who plays one of the tit-for strategies **(tt)**, **(t2)** or **(2t)**. With a timeout of $1s$ and $\gamma = 0.9$, we see a similar start as when playing against **(co)**, except when *agent* starts defecting, it does not work out so well. With longer timeouts, defection persists. With $\gamma = 0.99$, we see better coordination, especially at mid-range timeouts ($t_a = 10s - 30s$).

- Figures A.7 and A.15 show *agent* playing against *client* who has less resources ($t_c = 1.0$). We might expect here to see that *agent* will "outsmart" *client* and gain an advantage, however this happens only seldom. In particular, with mid-range timeouts ($t_a = 30 - 60s$ for $\gamma = 0.99$ and $t_a = 10 - 120s$ for $\gamma = 0.9$), *agent* attempts to do this after about 10 games, but this generally leads to less reward (although a bit more than *client* gets, so *agent* is "beating" client at the game, which doesn't really work in this case as it is not zero-sum). *agent* sees short-term possibilities of defection (it will get 11 as opposed to 10), but *client* is able to quickly adjust and adapt its behaviour, even with a timeout of $t_c = 1s$. We can see this effect when *agent* plays against **(to)**: it is able to start defecting after about 2-3 games when it has a timeout of $1s$.

Let us take a closer look at the last case, where $t_c = 1s$ and $t_a = 120s$ for $\gamma = 0.9$. One typical game in this series is shown in Table A.1. At the start, *agent* defects, then starts cooperating, feeling like a *feminine cousin* interacting with a *self-conscious spokeswoman*.

client feels like a *self-conscious stepson* interacting with an *easygoing stepmother*. Subsequent to this, both agents cooperate. This causes *client* to re-evaluate himself as significantly less good (lower E) than he originally thought (as a *stepson* rather than a *best man*, as he is attributing the cause of the original defection back to himself, or at least taking some of the blame. This then causes *client* to be sending rather negative messages to *agent*, causing *agent* to re-evaluate himself more negatively as well. At game 10, both agents are still cooperating (and have done so since game 1), but feeling rather badly and powerless. This finally causes *agent* to defect again (and he does so until the end of the game). *agent* feels like a *dependent nut*, and *client* cooperates twice in the face of this, then defects, feeling like an *exasperated gun_moll* (affectively like a *buddy*) - *"hey, I thought we were friends?"*. *agent* feels contrite (guilty) after *client* attempts to cooperate once more, after which both start defecting. After four more defections, *client* again tries to cooperate, but is rebuffed, and both feel like ex-girlfriends: the end of a beautiful friendship.

Table A.2 shows the example from Chapter 3 in which an agent is playing against a client playing (to). In this case, the action of cooperation is interpreted as *collaborate with* (EPA:$\{1.44, 1.11, 0.61\}$). As I noted in Chapter 3, this makes the *agent* feel less good than he would normally, like a *failure*. Let us now look at an example with a different (more positive and powerful) interpretation of the propositional action of cooperation. Table A.3 shows such a case, where again *client* is playing **(to)**, but the cooperation action is interpreted (by *agent*) as *flatter* (EPA:$\{2.1, 1.45, 0.82\}$). There is no environmental noise. We see that this example starts about the same as in Table A.4, although in this case *agent* does not defect on the second game. Once *client* starts defecting though (at game 3), *agent* rapidly re-adjusts his estimate of client from an *earnest lady* to an *unfair sawbones* or a *immoral bureaucrat*. After the 14th game, *agent* feels like a *dependent klutz* playing against a *cynical ex-boyfriend*. Overall we see the end result is quite similar, even though the start of the game is quite different. In the end, the feelings of the *agent* are quite a bit more negative and less powerful, probably as a reaction to the more positive and powerful actions of *client* at the start of the game.

Table A.4 shows an example where *client* is playing **(co)**, and the cooperation action is interpreted (by *agent*) as *flatter* (EPA:$\{2.1, 1.45, 0.82\}$). There is no environmental noise. We see that in this case, the *agent* intially feels much more positive (as a *warm date*), as compared to Table 3.2 (copied from the paper), where the *agent* felt like a *failure*. We see that the subtle difference in this interpretations causes quite different identity feelings for a *pd-agent*. The *agent* cooperates on the first move, defects once, then continues to cooperate for another 14 games. At this point, *agent* feels like a *self-conscious waiter* interacting with a *conscientious brunette*, and starts to defect, leading him to feel like a *self-conscious nut*, significantly less good, but about the same power and activity.

game #	actor	sentiments $\mathbf{f}_a$	sentiments $\mathbf{f}_c$	sentiments $\mathbf{f}_b$	deflection	identities agent	identities client	emotions agent	emotions client	actions agent	actions client
1	*agent*	2.21,1.50,1.18	2.28,1.53,1.29	-0.31,-0.61,0.05	4.39	partner	newlywed	self-conscious	introspective		
	client	2.04,1.49,1.26	-1.72,-0.29,-0.41	2.40,1.26,1.46	4.65	best man	adulterer	exasperated	feminine	def.	coop.
2	*agent*	2.12,1.22,1.01	2.16,1.23,1.14	3.19,1.62,1.04	1.49	sweetheart	sister	introspective	warm		
	client	1.56,1.12,1.10	-0.96,-0.20,-0.22	2.42,1.25,1.55	8.94	Air Force reservist	suspect	self-conscious	polite	coop.	coop.
3	*agent*	1.98,0.87,0.87	1.85,1.02,0.97	2.17,0.97,0.92	2.62	fiancée	bride	feminine	middle-aged		
	client	1.14,0.92,0.89	-0.36,-0.16,-0.02	0.59,1.24,0.08	4.52	big sister	toady	self-conscious	accommodating	coop.	coop.
4	*agent*	1.67,0.53,0.77	1.41,0.81,0.81	2.06,0.89,1.02	2.65	cousin	spokeswoman	feminine	self-conscious		
	client	0.81,0.68,0.72	0.01,-0.15,0.15	0.55,0.46,0.28	3.16	stepson	stepmother	self-conscious	easygoing	coop.	coop.
						...					
10	*agent*	0.33,-0.56,0.30	0.01,0.12,0.39	0.05,-0.41,0.08	0.66	nut	chum	exasperated	no emotion		
	client	-0.03,0.09,0.42	0.21,-0.62,0.32	0.33,-0.03,0.54	0.55	chum	nut	exasperated	exasperated	coop.	coop.
11	*agent*	0.11,-0.67,0.27	-0.13,0.03,0.38	-0.06,-0.66,-0.18	0.52	nut	gun_moll	dependent	no_emotion		
	client	-0.16,0.03,0.38	0.02,-0.71,0.29	0.14,0.69,0.38	0.41	gun_moll	divorcée	no_emotion	exasperated	def.	coop.
						...					
13	*agent*	-0.20,-0.78,0.22	-0.35,-0.06,0.32	-0.16,-0.55,-0.16	0.23	divorcée	gun_moll	contrite	envious		
	client	-0.37,-0.08,0.31	-0.25,-0.79,0.20	-0.42,-0.02,0.20	0.30	gun_moll	divorcée	exasperated	exasperated	def.	def.
14	*agent*	-0.28,-0.79,0.21	-0.41,-0.14,0.34	-0.17,-0.28,-0.22	0.19	divorcée	hussy	contrite	no_emotion		
	client	-0.43,-0.16,0.31	-0.31,-0.79,0.17	-0.04,0.38,0.44	0.21	hussy	divorcée	no_emotion	exasperated	def.	coop.
						...					
20	*agent*	-0.63,-0.74,0.32	-0.61,-0.61,0.27	-0.66,-0.31,0.32	0.08	ex-girlfriend	ex-girlfriend	exasperated	exasperated		
	client	-0.62,-0.62,0.25	-0.68,-0.73,0.31	-0.37,-0.11,0.40	0.09	ex-girlfriend	ex-girlfriend	exasperated	envious	def.	coop.

Table A.1: Example games with *client* $t_C = 1s$ whereas *agent* $t_a = 120s$.

game	post-play sentiments			defl-	identities		emotions		actions	
#	$\mathbf{f}_a$	$\mathbf{f}_c$	$\mathbf{f}_b$	ection	agent	client	agent	client	agent	client
1	-1.36,-0.01,-0.35	2.32,1.61,1.27	2.62,1.58,1.73	4.44	failure	newlywed	easygoing	idealistic	coop.	coop.
2	-0.66,0.04,-0.05	1.77,1.27,1.06	2.23,1.00,1.76	3.70	parolee	husband	easygoing	self-conscious	coop.	coop.
3	-0.23,-0.08,0.20	1.02,0.93,0.84	2.49,0.97,1.87	7.19	stepmother	purchaser	female	immoral	coop.	def.
4	-0.12,-0.33,0.33	0.27,0.62,0.62	2.37,0.48,1.34	4.99	stuffed_shirt	roommate	dependent	unfair	coop.	def.
5	-0.26,-0.47,0.32	-0.26,0.26,0.42	-0.59,0.41,-0.23	3.27	divorcée	gun_moll	dependent	selfish	def.	def.
6	-0.37,-0.66,0.26	-0.61,0.00,0.28	-0.10,-0.41,-0.27	2.29	divorcée	hussy	disapproving	selfish	def.	def.

Table A.2: Example games with *client* playing (to), and cooperation is interpreted as *collaborate with*. This is the same example as in Chapter 3, repeated here for easy comparisons.

game	post-play sentiments			defl-	identities		emotions		actions	
#	$\mathbf{f}_a$	$\mathbf{f}_c$	$\mathbf{f}_b$	ection	agent	client	agent	client	agent	client
1	2.77, 1.59, 1.31,	2.69, 1.70, 1.17,	2.65, 1.40, 1.49,	1.01	date	friend	warm	earnest	cooperate	cooperate
2	2.70, 1.34, 1.16,	2.58, 1.41, 0.96,	2.54, 1.55, 1.66,	1.14	lady	lady	introspective	earnest	cooperate	cooperate
3	2.39, 0.90, 1.04,	1.75, 1.05, 0.83,	2.43, 1.50, 1.49,	14.13	lady	bride	dependent	inconsiderate	cooperate	defect
4	1.67, 0.36, 0.89,	0.74, 0.72, 0.65,	2.52, 0.80, 1.27,	10.69	grandson	steady	nervous	unfair	cooperate	defect
5	1.07, -0.13, 0.70,	0.03, 0.44, 0.47,	1.98, 0.11, 0.16,	6.59	waiter	sawbones	gullible	unfair	cooperate	defect
6	0.62, -0.47, 0.57,	-0.43, 0.22, 0.33,	1.70, 0.19, 0.74,	3.84	schoolboy	bureaucrat	flustered	immoral	cooperate	defect
7	0.23, -0.74, 0.46,	-0.73, 0.03, 0.21,	0.24, -0.50, 0.11,	2.63	nut	tease	flustered	prejudiced	defect	defect
8	-0.05, -0.89, 0.39,	-0.92, -0.13, 0.13,	0.13, -0.30, 0.23,	1.83	drunk	malcontent	flustered	prejudiced	defect	defect
9	-0.25, -0.96, 0.35,	-1.03, -0.26, 0.08,	-0.08, -0.19, 0.29,	1.46	drunk	malcontent	inhibited	annoyed	defect	defect
10	-0.39, -1.06, 0.32,	-1.11, -0.34, 0.04,	-0.03, -0.66, 0.26,	1.27	klutz	malcontent	disapproving	annoyed	defect	defect
11	-0.49, -1.14, 0.30,	-1.16, -0.40, 0.01,	-0.20, -0.68, 0.18,	1.17	klutz	malcontent	inhibited	annoyed	defect	defect
12	-0.55, -1.21, 0.28,	-1.20, -0.43, -0.00,	-0.22, -0.75, 0.02,	1.10	klutz	malcontent	inhibited	annoyed	defect	defect
13	-0.60, -1.24, 0.27,	-1.22, -0.46, -0.01,	-0.23, -0.65, 0.27,	1.07	klutz	ex-boyfriend	dependent	annoyed	defect	defect
14	-0.63, -1.24, 0.28,	-1.23, -0.48, -0.02,	-0.25, -0.53, 0.29,	1.08	klutz	ex-boyfriend	dependent	cynical	defect	defect
15	-0.64, -1.19, 0.29,	-1.23, -0.50, -0.02,	-0.13, -0.25, 0.37,	1.10	klutz	ex-boyfriend	dependent	cynical	defect	defect
16	-0.69, -1.16, 0.31,	-1.23, -0.53, -0.03,	-0.63, -0.45, 0.46,	1.14	klutz	ex-boyfriend	dependent	cynical	defect	defect
17	-0.69, -1.18, 0.31,	-1.23, -0.53, -0.03,	-0.20, -0.65, 0.30,	1.10	klutz	ex-boyfriend	dependent	cynical	defect	defect
18	-0.70, -1.23, 0.31,	-1.24, -0.53, -0.03,	-0.27, -0.84, 0.33,	1.07	klutz	ex-boyfriend	dependent	cynical	defect	defect
19	-0.79, -1.29, 0.41,	-1.42, -0.54, 0.05,	-0.66, -0.38, 0.52,	0.42	goof-off	womanizer	contrite	shaken	defect	defect
20	-0.77, -1.20, 0.38,	-1.31, -0.57, 0.03,	-0.43, -0.40, 0.23,	1.16	queer	neurotic	dependent	cynical	defect	defect

Table A.3: Example games with *client* playing (to), and cooperation interpreted as *flatter*.

game	post-play sentiments			defl-	identities		emotions		actions	
#	$\mathbf{f}_a$	$\mathbf{f}_c$	$\mathbf{f}_b$	ection	agent	client	agent	client	agent	client
1	2.67, 1.55, 1.37	2.53, 1.62, 1.14	2.35, 1.23, 1.52	1.10	date	girlfriend	warm	earnest	coop.	coop.
2	2.01, 0.99, 1.16	2.07, 1.30, 0.96	-0.06, -0.34, 0.05	3.97	coed	organizer	no_emotion	introspective	def.	coop.
3	1.95, 0.74, 0.94	1.90, 1.11, 0.83	3.07, 1.42, 0.69	1.20	girl	bride	introspective	warm	coop.	coop.
4	1.85, 0.54, 0.83	1.83, 1.00, 0.74	2.04, 0.69, 1.04	1.03	whiz_kid	fiancée	introspective	easygoing	coop.	coop.
5	1.76, 0.41, 0.77	1.79, 0.93, 0.68	2.12, 0.79, 1.12	0.83	cousin	fiancée	introspective	easygoing	coop.	coop.
6	1.73, 0.35, 0.67	1.76, 0.83, 0.63	2.07, 0.70, 0.52	0.80	cousin	whiz_kid	introspective	easygoing	coop.	coop.
7	1.55, 0.25, 0.61	1.68, 0.74, 0.61	1.66, 0.65, 0.51	0.90	houseguest	nonsmoker	introspective	warm	coop.	coop.
8	1.47, 0.08, 0.59	1.71, 0.72, 0.60	1.89, -0.01, 0.83	0.77	houseguest	cousin	feminine	warm	coop.	coop.
9	1.48, 0.03, 0.63	1.63, 0.61, 0.60	2.14, 0.24, 1.07	0.63	student_teacher	cousin	introspective	affectionate	coop.	coop.
10	1.48, -0.04, 0.65	1.57, 0.60, 0.62	1.80, 0.10, 1.03	0.71	student_teacher	cousin	idealistic	affectionate	coop.	coop.
11	1.45, -0.14, 0.66	1.54, 0.62, 0.61	1.78, 0.12, 0.78	0.69	student_teacher	classmate	introspective	warm	coop.	coop.
12	1.34, -0.23, 0.63	1.42, 0.63, 0.57	1.73, 0.04, 0.51	0.75	daughter-in-law	Air_Force_enlistee	self-conscious	awe-struck	coop.	coop.
13	1.25, -0.30, 0.60	1.36, 0.54, 0.58	1.60, -0.18, -0.01	0.80	woman	shopper	nostalgic	warm	coop.	coop.
14	1.18, -0.20, 0.45	1.35, 0.50, 0.55	1.45, 0.22, 0.67	0.88	woman	citizen	self-conscious	conscientious	coop.	coop.
15	1.04, -0.25, 0.48	1.33, 0.36, 0.46	1.42, -0.03, 0.35	0.93	woman	small_businessman	self-conscious	conscientious	coop.	coop.
16	1.01, -0.27, 0.59	1.15, 0.40, 0.42	0.90, -0.47, 0.62	1.01	waiter	brunette	self-conscious	conscientious	coop.	coop.
17	0.71, -0.41, 0.44	0.89, 0.35, 0.49	0.27, -0.66, 0.20	1.32	schoolboy	half_sister	no_emotion	idealistic	def.	coop.
18	0.47, -0.45, 0.42	0.75, 0.26, 0.46	0.16, -0.34, 0.02	2.14	nut	son-in-law	no_emotion	conscientious	def.	coop.
19	0.35, -0.49, 0.38	0.69, 0.24, 0.44	0.41, -0.54, -0.17	1.92	nut	son-in-law	self-conscious	conscientious	def.	coop.
20	0.25, -0.53, 0.37	0.66, 0.23, 0.43	0.15, -0.70, 0.20	1.96	nut	co-worker	self-conscious	conscientious	def.	coop.

Table A.4: Example games with *client* playing (co), and cooperation interpreted as *flatter*.

Tables A.21-A.28 and Figures A.17-A.24 show the results with varying levels of environmental noise (from 0.01 to 5.0) for POMCP-C timeout of $t_a = 120s$. The varying environmental noise is added to the communications of $\mathbf{F}_b$ as random Gaussian noise with standard deviation σ_b, and reflected in the variance of the observation function for $Pr(\boldsymbol{\omega_f}|\mathbf{f}_b)$, which is Gaussian with the same standard deviation. We can make the following observations.

- When playing against **(co)**, with high noise (std. dev of 5.0), *agent* basically ignores $\mathbf{F}_b$ from *client*, and so thinks of *client* as a friend only. At lower noise levels, the results remain roughly similar (compare Figure A.17 with Figure A.9).
- When playing against **(de)** or **(to)**, results remain roughly the same for all noise levels. What this means is that the *client*'s actions of always defecting outweigh any signals that are being sent as $\mathbf{F}_b$.
- When playing against **(tt)**, **(t2)** and **(2t)**, at low noise levels (0.01), *agent* defects significantly more than with no noise and $\sigma_b = 0.1$ (as in Chapter 3). The reason is that the signals accompanying defection cause more significant disruption. At higher noise levels, this effect goes away, and we see more cooperation from both agents. (compare Figures A.20, A.21 and A.22 with Figures A.12, A.13 and A.14, respectively).
- For **(1.0)** and **(same)**, we see very little effect of environmental noise.

The lack of any strong effect of environmental noise in the communication of $\mathbf{F}_b$ across a wide range of conditions is because of the significantly more powerful effect of the propositional action of the *client* serving as evidence about the *client* identity. Except at very small values of σ_b, this means that the communication of $\mathbf{f}_b$ makes little difference. In fact, I believe that $\mathbf{f}_b$ may not be communicated at all in many cases, with humans relying on expressions of emotions instead which are direct evidence about identities, so more powerful.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.29 ± 0.38	10.33 ± 0.42	10.55 ± 0.28	10.44 ± 0.34	10.72 ± 0.18	10.76 ± 0.20	10.96 ± 0.18
	client	7.10 ± 3.75	6.70 ± 4.21	4.50 ± 2.80	5.65 ± 3.44	2.80 ± 1.82	2.45 ± 1.96	0.45 ± 1.79
mean	agent	10.58 ± 0.10	10.66 ± 0.08	10.78 ± 0.22	10.69 ± 0.19	10.85 ± 0.21	10.87 ± 0.24	11.00 ± 0.00
(last 10)	client	4.20 ± 1.03	3.40 ± 0.84	2.20 ± 2.20	3.10 ± 1.91	1.50 ± 2.07	1.30 ± 2.36	0.00 ± 0.00
median	agent	10.00	10.00	11.00	10.00	11.00	11.00	11.00
	client	10.00	10.00	0.00	10.00	0.00	0.00	0.00
median	agent	11.00	11.00	11.00	11.00	11.00	11.00	11.00
(last 10)	client	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table A.5: PD experiments with client strategy: (co) and discount $\gamma = 0.9$

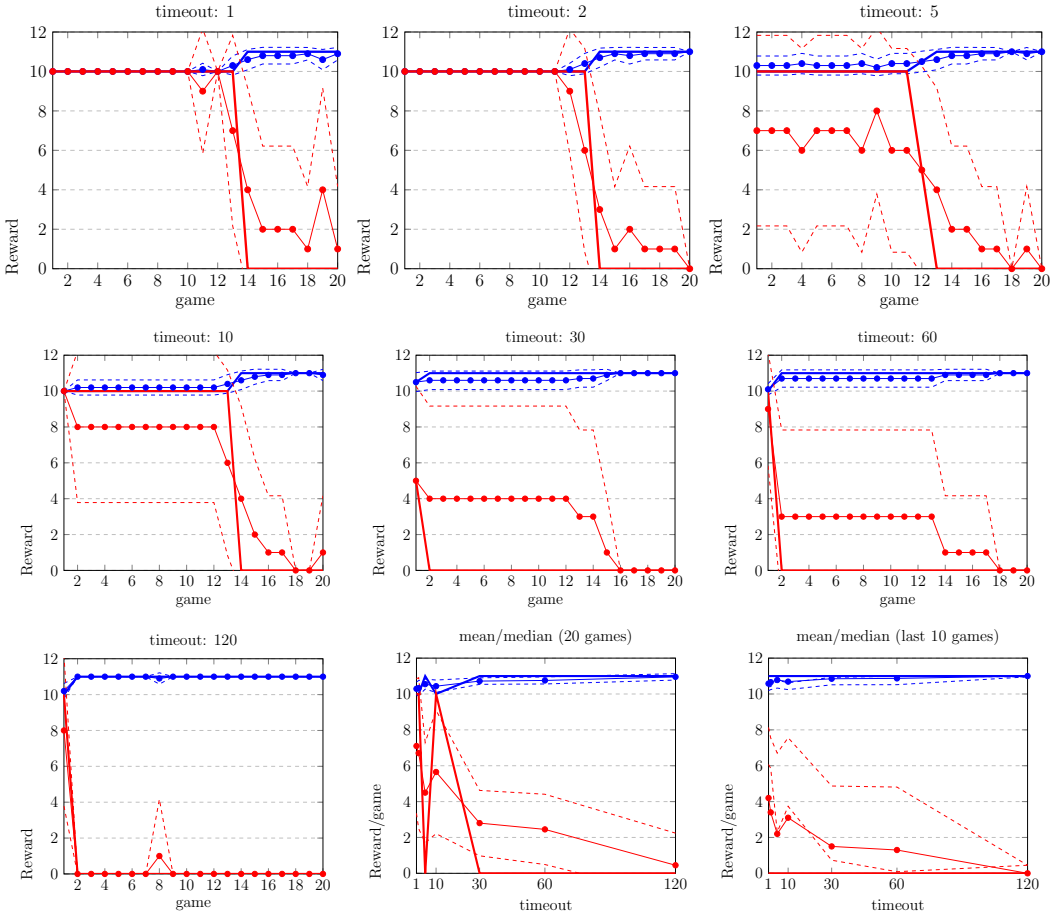


Figure A.1: PD experiments with client strategy: (co) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	0.67 ± 0.21	0.78 ± 0.24	0.85 ± 0.20	0.89 ± 0.23	0.93 ± 0.16	0.94 ± 0.16	0.93 ± 0.21
	client	4.35 ± 2.11	3.20 ± 2.42	2.50 ± 2.04	2.10 ± 2.31	1.70 ± 1.63	1.60 ± 1.57	1.75 ± 2.15
mean	agent	0.75 ± 0.16	0.86 ± 0.13	0.94 ± 0.10	0.96 ± 0.07	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
(last 10)	client	3.50 ± 1.58	2.40 ± 1.35	1.60 ± 0.97	1.40 ± 0.70	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.6: PD experiments with client strategy: (de) and discount $\gamma = 0.9$

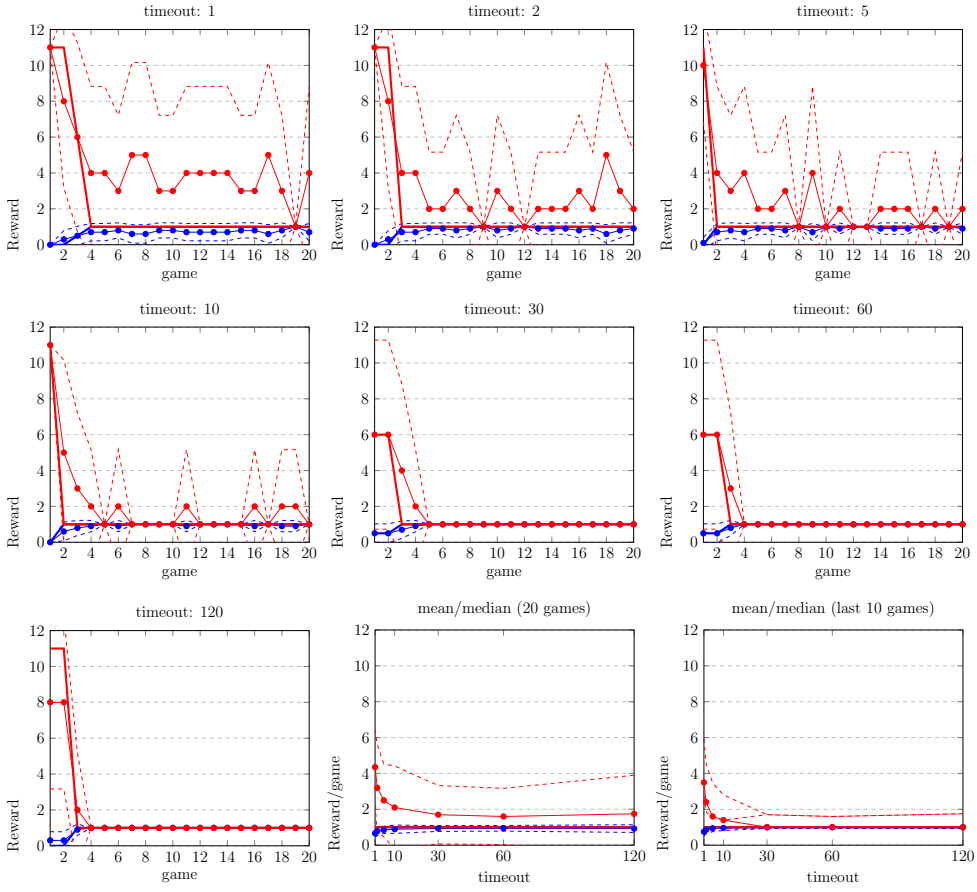


Figure A.2: PD experiments with client strategy: (de) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	1.50 ± 2.92	1.58 ± 2.89	1.81 ± 2.89	1.83 ± 2.89	1.92 ± 2.97	1.92 ± 2.92	1.88 ± 2.89
	client	5.90 ± 2.85	5.05 ± 3.47	2.80 ± 2.26	2.55 ± 2.68	1.70 ± 1.17	1.75 ± 1.65	2.15 ± 2.25
mean	agent	0.65 ± 0.14	0.79 ± 0.11	0.93 ± 0.07	0.99 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
(last 10)	client	4.50 ± 1.35	3.10 ± 1.10	1.70 ± 0.67	1.10 ± 0.32	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	client	5.50	1.00	1.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.7: PD experiments with client strategy: (to) and discount $\gamma = 0.9$

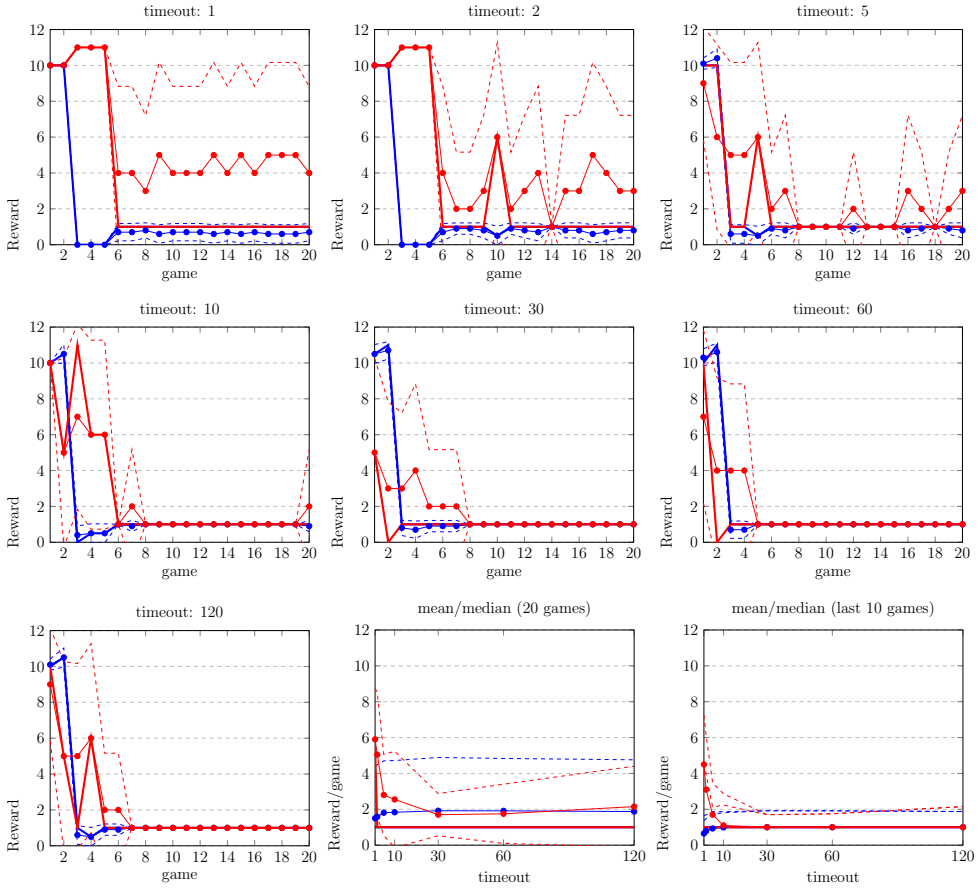


Figure A.3: PD experiments with client strategy: (to) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	7.86 ± 3.23	7.79 ± 3.02	4.74 ± 2.63	3.57 ± 2.51	2.27 ± 2.08	2.67 ± 2.09	1.64 ± 2.24
	client	7.69 ± 3.18	7.35 ± 3.20	4.19 ± 2.39	3.02 ± 1.93	1.72 ± 0.51	2.12 ± 0.88	1.08 ± 0.46
mean	agent	5.71 ± 1.44	5.58 ± 1.13	2.95 ± 2.01	2.20 ± 1.89	1.55 ± 1.74	1.65 ± 1.39	1.00 ± 0.00
(last 10)	client	5.38 ± 1.70	4.70 ± 1.11	2.29 ± 1.56	1.87 ± 1.50	1.44 ± 1.39	1.43 ± 0.93	1.00 ± 0.00
median	agent	10.00	10.00	1.00	1.00	1.00	1.00	1.00
	client	10.00	10.00	1.00	1.00	1.00	1.00	1.00
median	agent	10.00	5.50	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.8: PD experiments with client strategy: (tt) and discount $\gamma = 0.9$

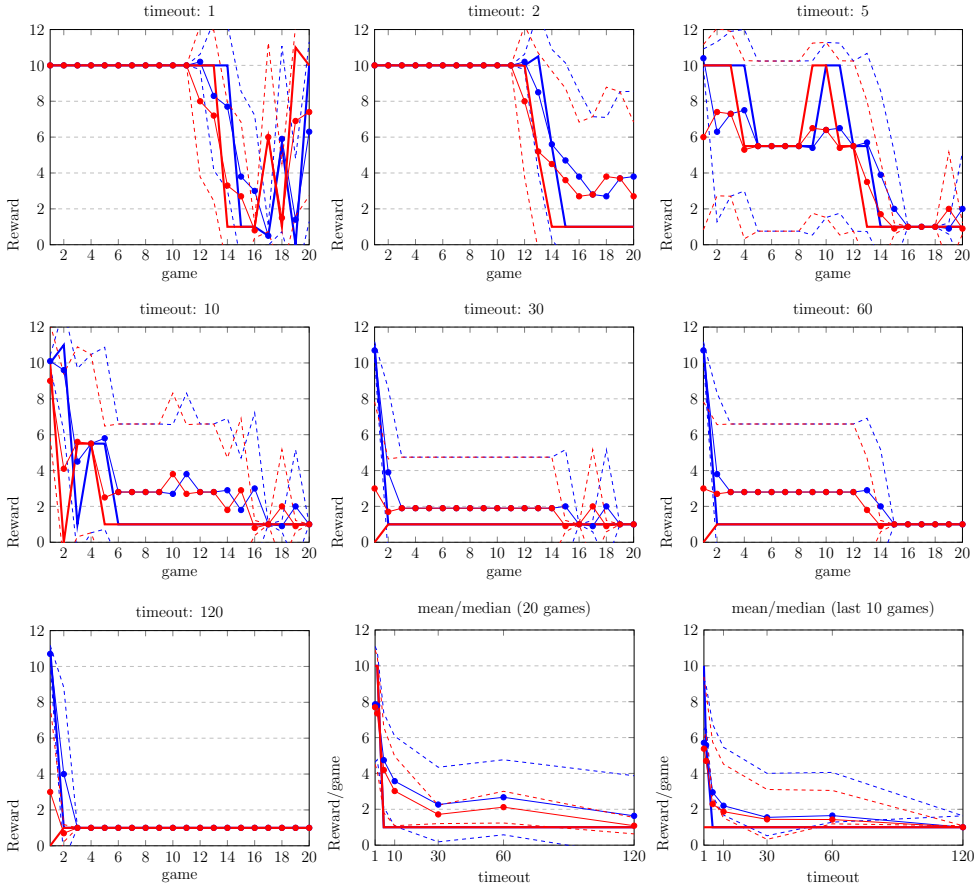


Figure A.4: PD experiments with client strategy: (tt) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	8.70 ± 2.28	8.46 ± 2.54	3.37 ± 2.78	6.18 ± 2.95	3.85 ± 2.61	3.44 ± 2.71	3.98 ± 2.48
	client	7.21 ± 3.54	7.13 ± 3.63	1.55 ± 0.87	4.64 ± 2.83	2.65 ± 1.10	2.29 ± 1.10	2.83 ± 1.24
mean	agent	7.39 ± 1.17	6.91 ± 1.07	1.96 ± 1.35	3.92 ± 2.63	2.57 ± 3.32	2.03 ± 2.18	2.69 ± 2.86
(last 10)	client	4.42 ± 1.40	4.27 ± 1.01	1.30 ± 0.58	2.49 ± 1.54	2.13 ± 2.39	1.59 ± 1.26	1.92 ± 1.61
median	agent	10.00	10.00	1.00	10.00	1.00	1.00	1.00
	client	10.00	10.00	1.00	1.00	1.00	1.00	1.00
median	agent	10.00	10.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.9: PD experiments with client strategy: (t2) and discount $\gamma = 0.9$

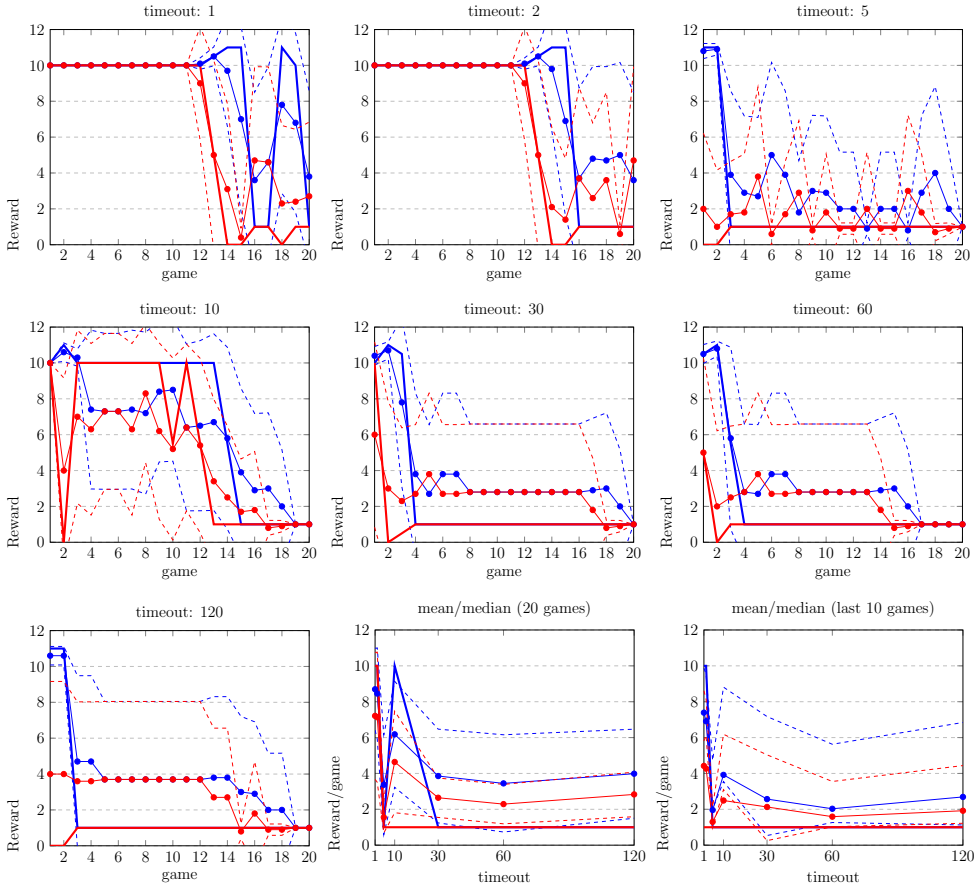


Figure A.5: PD experiments with client strategy: (t2) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	7.26 ± 3.89	7.21 ± 4.13	4.47 ± 2.46	5.75 ± 2.89	2.75 ± 2.15	3.48 ± 2.17	1.72 ± 2.35
	client	7.21 ± 3.50	7.21 ± 3.64	4.42 ± 2.15	5.42 ± 2.99	2.31 ± 1.07	2.93 ± 1.44	1.23 ± 0.92
mean	agent	4.52 ± 1.28	4.42 ± 0.59	2.71 ± 2.20	3.76 ± 2.22	1.64 ± 1.37	2.29 ± 2.09	1.00 ± 0.00
(last 10)	client	4.41 ± 1.68	4.42 ± 0.75	2.71 ± 1.69	3.10 ± 1.66	1.53 ± 1.12	1.96 ± 1.57	1.00 ± 0.00
median	agent	10.00	10.00	1.00	10.00	1.00	1.00	1.00
	client	10.00	10.00	1.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.10: PD experiments with client strategy: (2t) and discount $\gamma = 0.9$

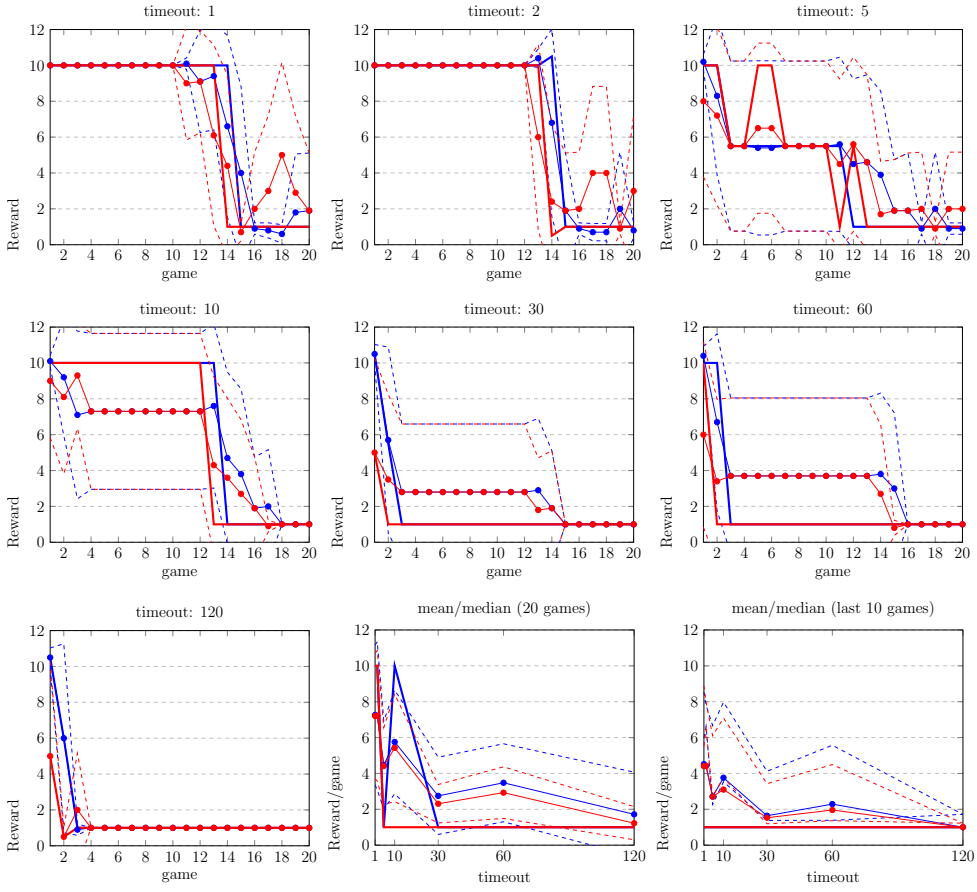


Figure A.6: PD experiments with client strategy: (2t) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.93 ± 1.30	9.50 ± 0.92	9.35 ± 0.94	9.21 ± 1.34
	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.38 ± 1.62	8.56 ± 1.56	8.09 ± 1.94	7.46 ± 2.40
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	7.89 ± 2.74	8.95 ± 1.24	8.64 ± 1.86	8.36 ± 1.74
(last 10)	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	7.34 ± 3.74	7.63 ± 3.52	6.77 ± 4.01	5.61 ± 3.91
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.11: PD experiments with client strategy: (1.0) and discount $\gamma = 0.9$

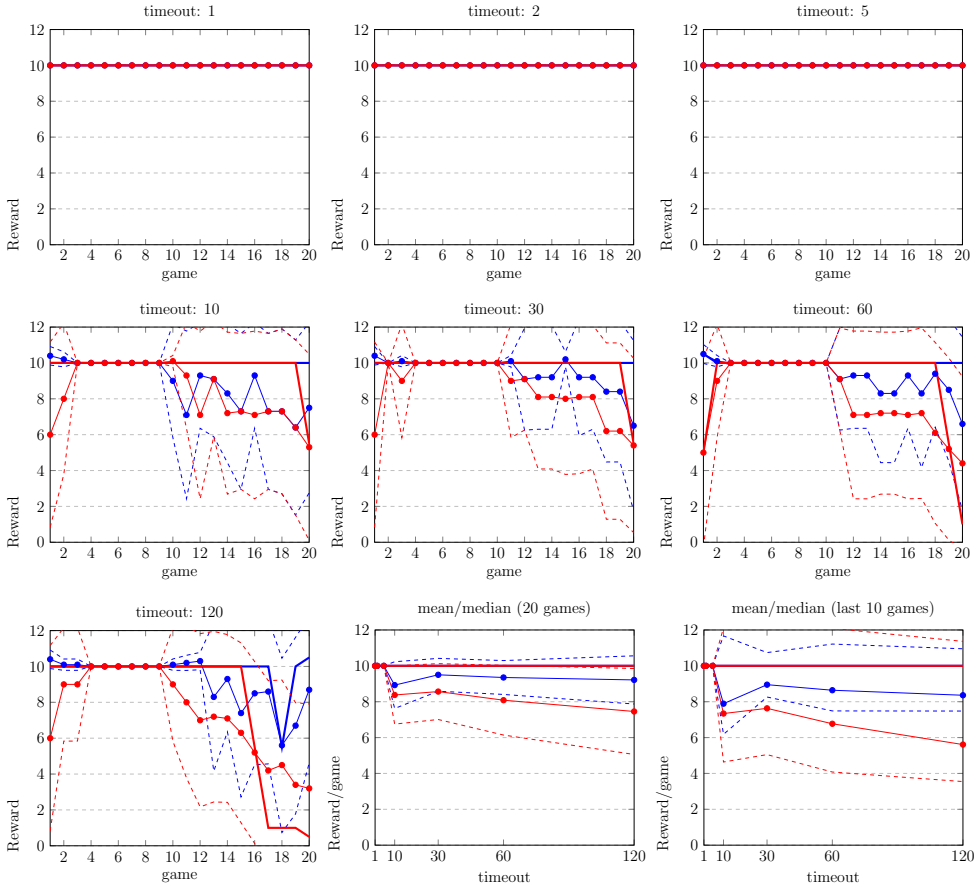


Figure A.7: PD experiments with client strategy: (1.0) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.54 ± 1.15	8.91 ± 0.99	4.14 ± 2.38	3.14 ± 1.96
	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.27 ± 1.35	8.47 ± 1.11	3.71 ± 2.36	2.87 ± 2.13
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	7.60 ± 3.32	8.10 ± 3.75	2.08 ± 2.84	1.65 ± 2.09
(last 10)	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	7.27 ± 3.76	8.21 ± 3.80	2.08 ± 2.84	1.43 ± 1.06
median	agent	10.00	10.00	10.00	10.00	10.00	1.00	1.00
	client	10.00	10.00	10.00	10.00	10.00	1.00	1.00
median	agent	10.00	10.00	10.00	10.00	10.00	1.00	1.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	1.00	1.00

Table A.12: PD experiments with client strategy: (same) and discount $\gamma = 0.9$

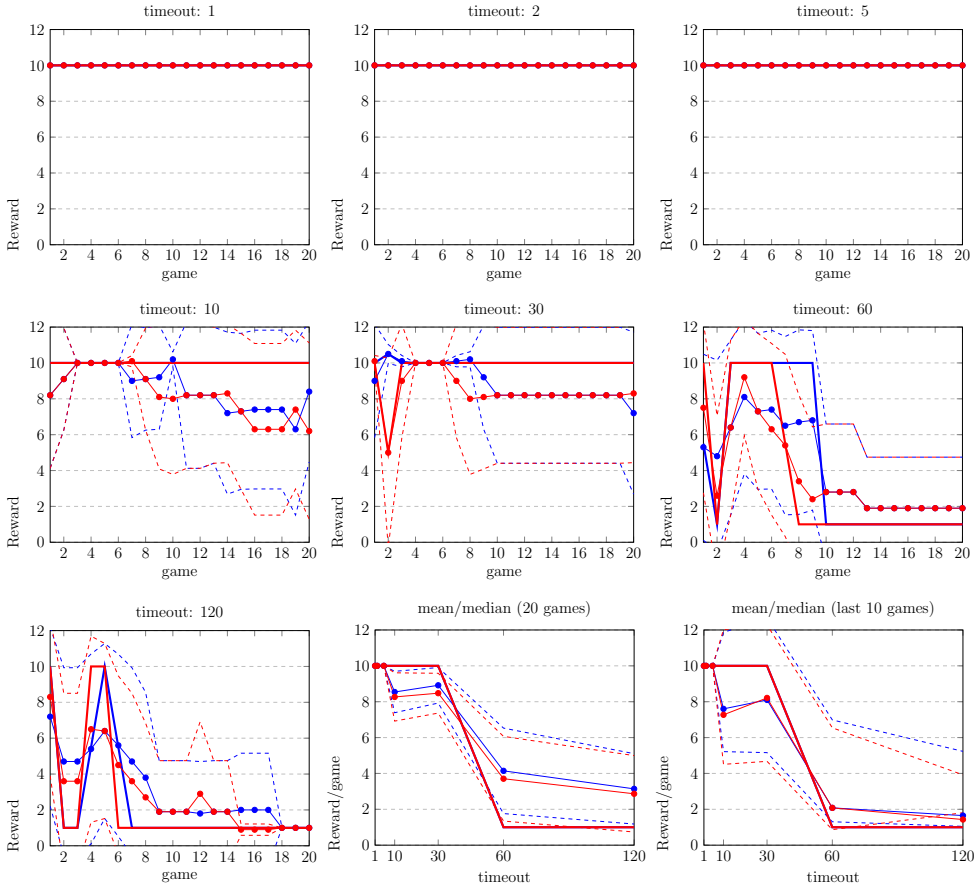


Figure A.8: PD experiments with client strategy: (same) and discount $\gamma = 0.9$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.33 ± 0.15	10.33 ± 0.18	10.06 ± 0.14	10.07 ± 0.07	10.07 ± 0.06	10.02 ± 0.04	10.47 ± 0.14
	client	6.70 ± 1.49	6.70 ± 1.78	9.40 ± 1.35	9.30 ± 0.66	9.25 ± 0.64	9.80 ± 0.41	5.25 ± 1.37
mean	agent	10.39 ± 0.12	10.43 ± 0.19	10.12 ± 0.10	10.10 ± 0.25	10.09 ± 0.19	10.04 ± 0.13	10.54 ± 0.38
(last 10)	client	6.10 ± 1.20	5.70 ± 1.95	8.80 ± 1.03	9.00 ± 2.49	9.10 ± 1.91	9.60 ± 1.26	4.60 ± 3.84
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	11.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	0.00

Table A.13: PD experiments with client strategy: (co) and discount $\gamma = 0.99$

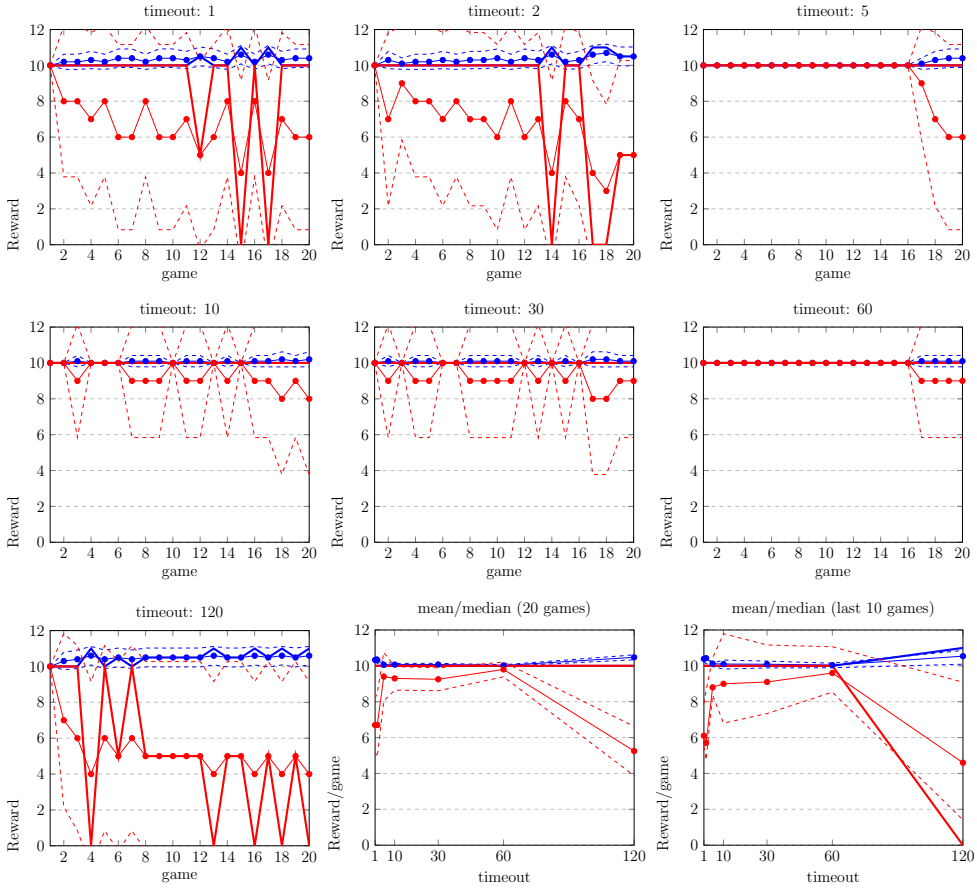


Figure A.9: PD experiments with client strategy: (co) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	0.59 ± 0.26	0.55 ± 0.21	0.47 ± 0.20	0.58 ± 0.22	0.67 ± 0.21	0.64 ± 0.26	0.77 ± 0.22
	client	5.10 ± 2.61	5.50 ± 2.12	6.35 ± 1.98	5.20 ± 2.17	4.35 ± 2.11	4.65 ± 2.60	3.30 ± 2.23
mean	agent	0.73 ± 0.19	0.64 ± 0.19	0.50 ± 0.20	0.66 ± 0.15	0.72 ± 0.09	0.76 ± 0.11	0.87 ± 0.15
(last 10)	client	3.70 ± 1.89	4.60 ± 1.90	6.00 ± 2.00	4.40 ± 1.51	3.80 ± 0.92	3.40 ± 1.07	2.30 ± 1.49
median	agent	1.00	1.00	0.00	1.00	1.00	1.00	1.00
	client	1.00	1.00	11.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	0.50	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	6.00	1.00	1.00	1.00	1.00

Table A.14: PD experiments with client strategy: (de) and discount $\gamma = 0.99$

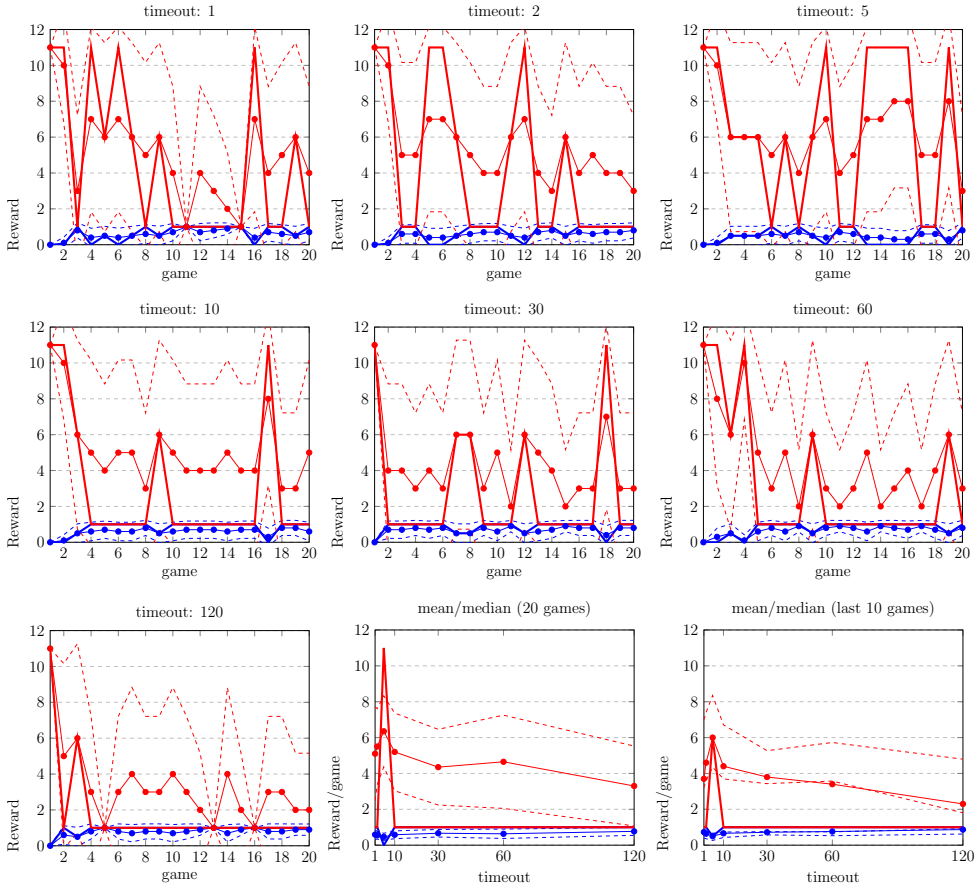


Figure A.10: PD experiments with client strategy: (de) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	1.54 ± 2.95	1.52 ± 2.97	1.38 ± 2.97	1.48 ± 2.93	1.57 ± 2.96	1.60 ± 2.91	1.68 ± 2.90
	client	5.45 ± 2.63	5.65 ± 2.25	7.10 ± 2.07	6.10 ± 2.97	5.15 ± 2.28	4.85 ± 2.54	4.10 ± 2.22
mean	agent	0.72 ± 0.19	0.69 ± 0.12	0.51 ± 0.19	0.64 ± 0.16	0.71 ± 0.19	0.78 ± 0.14	0.81 ± 0.11
(last 10)	client	3.80 ± 1.87	4.10 ± 1.20	5.90 ± 1.91	4.60 ± 1.58	3.90 ± 1.85	3.20 ± 1.40	2.90 ± 1.10
median	agent	1.00	1.00	0.00	1.00	1.00	1.00	1.00
	client	1.00	1.00	11.00	10.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.15: PD experiments with client strategy: (to) and discount $\gamma = 0.99$

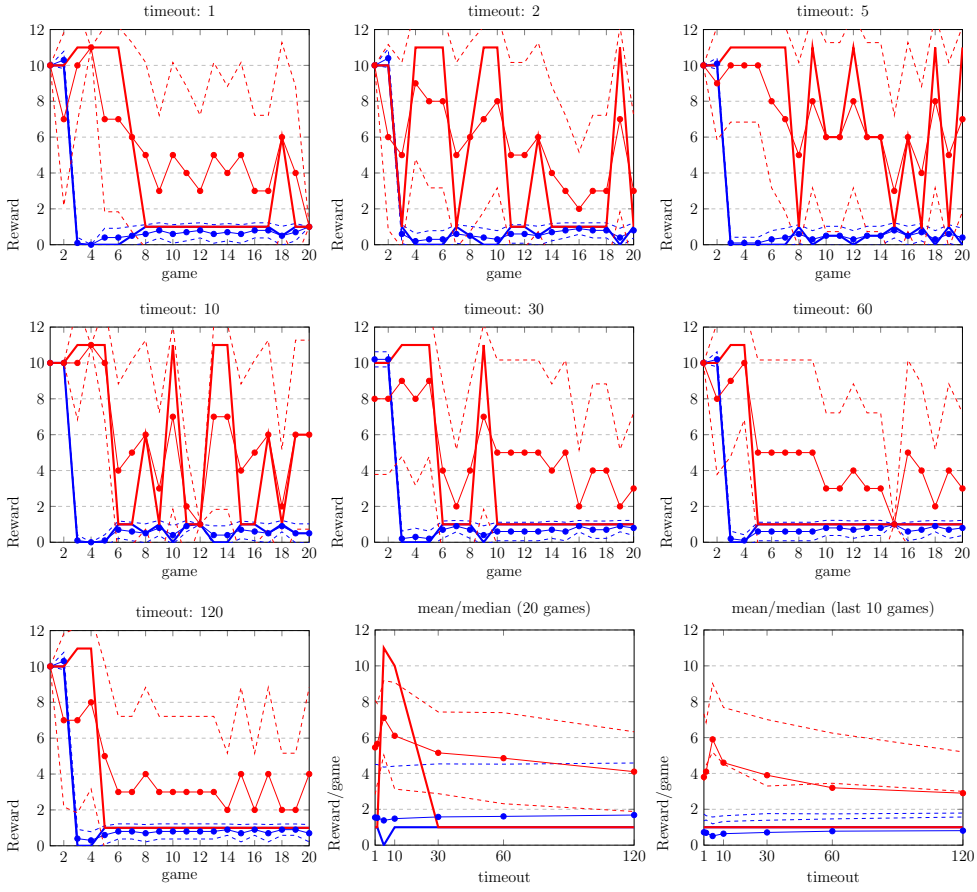


Figure A.11: PD experiments with client strategy: (to) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	7.29 ± 1.87	7.01 ± 1.73	9.16 ± 1.32	10.00 ± 0.00	8.38 ± 0.92	7.33 ± 1.42	7.33 ± 1.17
	client	6.96 ± 1.83	6.74 ± 1.58	8.95 ± 1.46	10.00 ± 0.00	8.33 ± 0.85	7.05 ± 1.30	7.05 ± 1.02
mean	agent	5.89 ± 1.26	6.41 ± 1.56	8.69 ± 1.41	10.00 ± 0.00	8.28 ± 2.82	7.04 ± 3.45	6.87 ± 3.75
(last 10)	client	5.56 ± 0.99	6.30 ± 1.29	8.25 ± 1.64	10.00 ± 0.00	8.39 ± 2.59	6.93 ± 3.48	6.65 ± 3.83
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.16: PD experiments with client strategy: (tt) and discount $\gamma = 0.99$

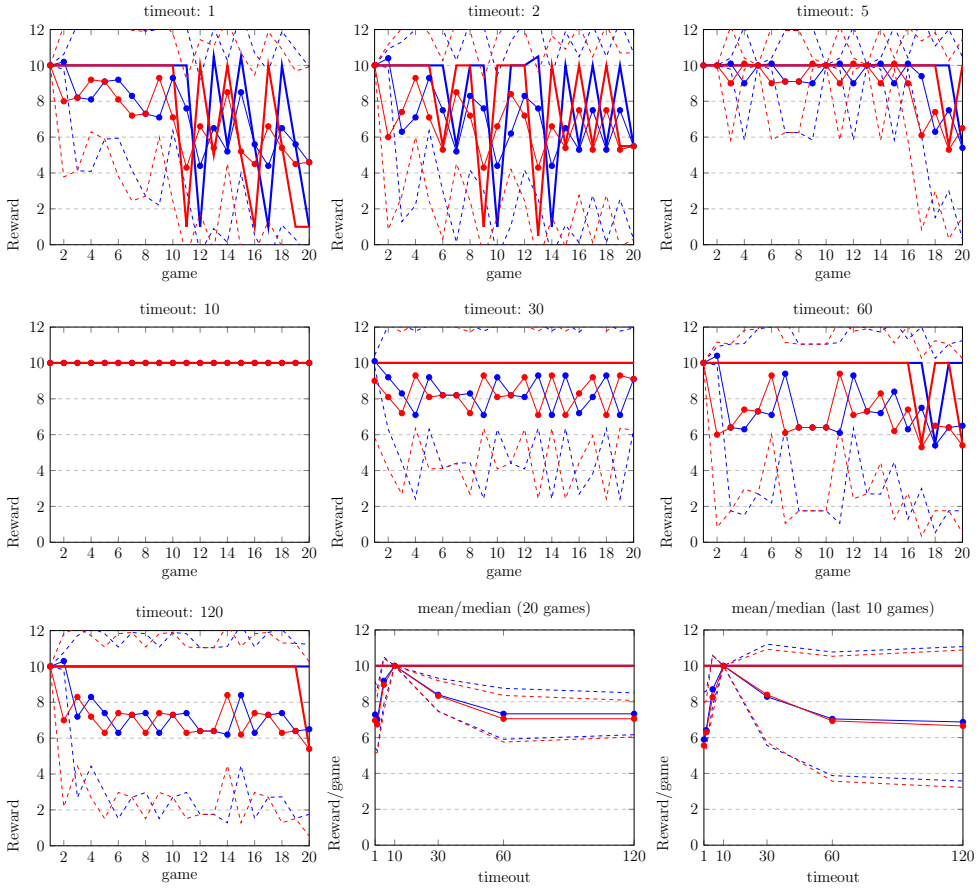


Figure A.12: PD experiments with client strategy: (tt) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	9.47 ± 1.15	9.66 ± 0.75	9.90 ± 0.42	9.96 ± 0.20	9.97 ± 0.16	9.25 ± 0.80	7.28 ± 1.68
	client	6.83 ± 2.15	8.45 ± 1.45	9.02 ± 1.68	9.46 ± 1.25	9.76 ± 0.70	8.59 ± 0.77	6.01 ± 1.40
mean	agent	9.17 ± 1.08	9.35 ± 1.18	9.87 ± 0.55	9.91 ± 0.54	9.95 ± 0.23	9.17 ± 1.91	6.57 ± 3.75
(last 10)	client	5.43 ± 1.08	7.59 ± 1.50	8.33 ± 1.44	8.92 ± 1.32	9.51 ± 0.94	8.40 ± 2.42	5.69 ± 4.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.17: PD experiments with client strategy: (t2) and discount $\gamma = 0.99$

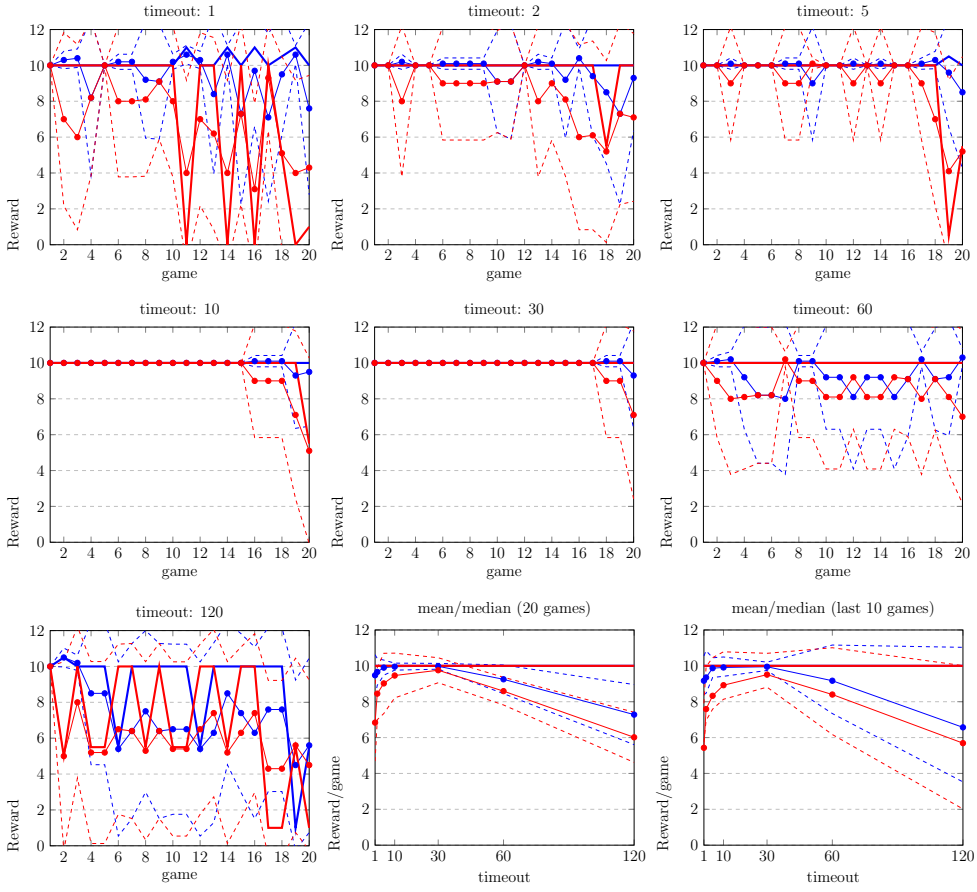


Figure A.13: PD experiments with client strategy: (t2) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	5.25 ± 2.75	7.42 ± 2.35	8.72 ± 1.41	9.90 ± 0.31	8.32 ± 0.90	8.45 ± 0.87	7.63 ± 0.91
	client	7.01 ± 1.80	8.08 ± 1.96	8.89 ± 1.27	9.96 ± 0.23	8.93 ± 0.91	8.67 ± 1.20	7.91 ± 0.84
mean	agent	2.90 ± 2.20	5.84 ± 2.83	8.19 ± 2.62	9.81 ± 0.60	7.86 ± 3.76	8.00 ± 3.84	7.33 ± 4.31
(last 10)	client	5.76 ± 1.04	7.05 ± 1.51	8.30 ± 1.53	9.92 ± 0.25	8.63 ± 2.22	8.22 ± 2.61	7.99 ± 3.25
median	agent	1.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	1.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	5.50	10.00	10.00	10.00	10.00	10.00	10.00

Table A.18: PD experiments with client strategy: (2t) and discount $\gamma = 0.99$

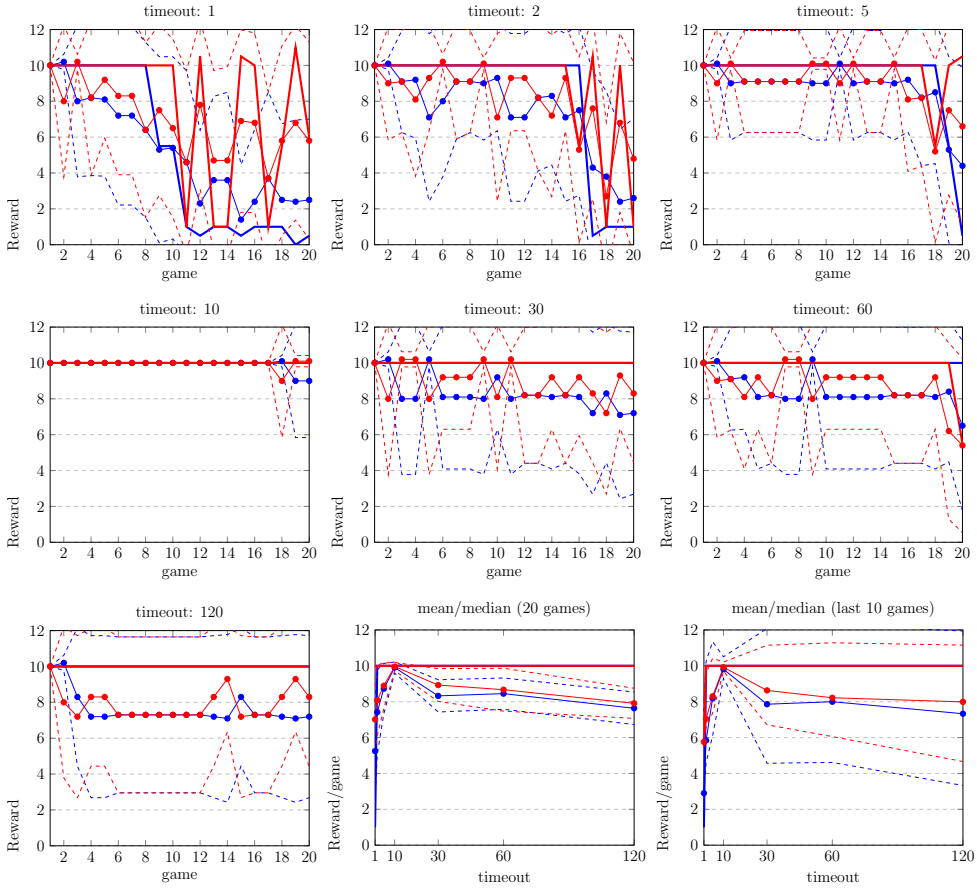


Figure A.14: PD experiments with client strategy: (2t) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	9.41 ± 1.17	9.79 ± 0.43	10.00 ± 0.00
	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.97 ± 1.25	9.68 ± 0.48	10.00 ± 0.00
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.79 ± 1.60	9.67 ± 1.04	10.00 ± 0.00
(last 10)	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.24 ± 2.29	9.34 ± 2.09	10.00 ± 0.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.19: PD experiments with client strategy: (1.0) and discount $\gamma = 0.99$

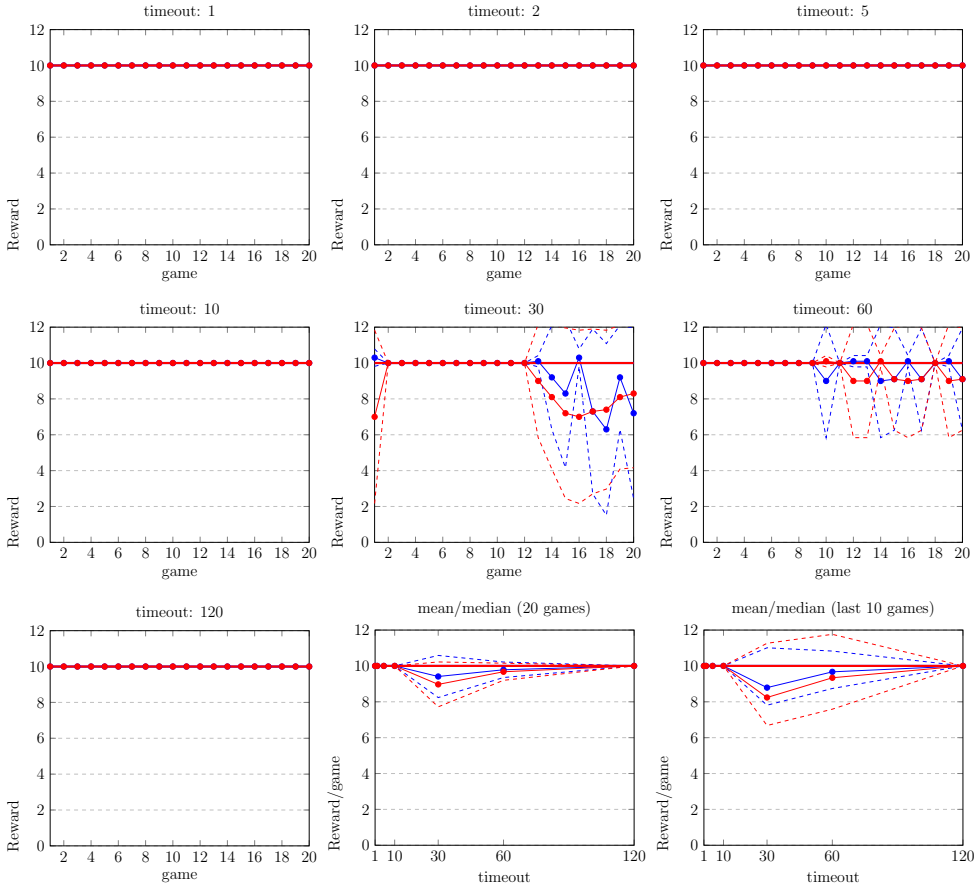


Figure A.15: PD experiments with client strategy: (1.0) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		timeout						
		1	2	5	10	30	60	120
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.15 ± 1.55	9.40 ± 0.46
	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	8.21 ± 1.50	9.52 ± 0.48
mean	agent	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	6.91 ± 3.38	9.26 ± 2.34
(last 10)	client	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	7.24 ± 3.08	9.48 ± 1.64
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.20: PD experiments with client strategy: (same) and discount $\gamma = 0.99$

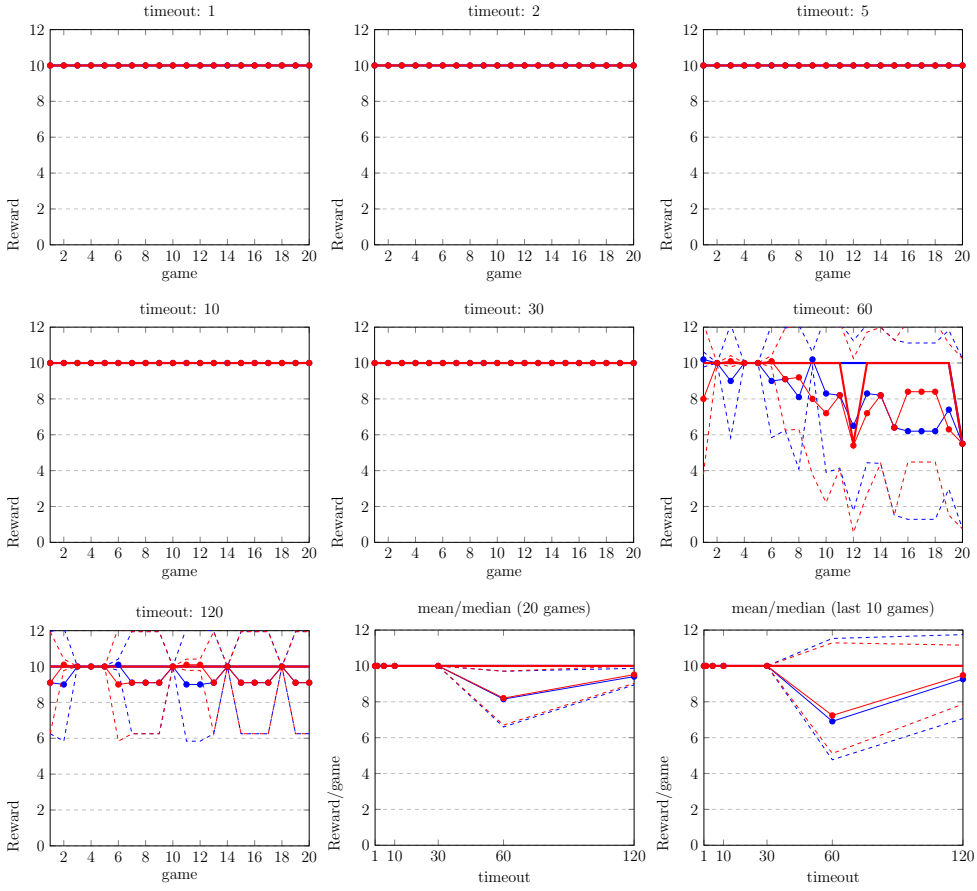


Figure A.16: PD experiments with client strategy: (same) and discount $\gamma = 0.99$ Red=client, Blue=agent, dashed=std.dev. solid (thin, with markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	10.64 ± 0.27	10.77 ± 0.22	10.23 ± 0.10	10.02 ± 0.05	10.04 ± 0.08	10.00 ± 0.00	10.00 ± 0.00
	client	3.60 ± 2.74	2.35 ± 2.18	7.70 ± 1.03	9.85 ± 0.49	9.60 ± 0.75	10.00 ± 0.00	10.00 ± 0.00
mean	agent	10.75 ± 0.13	10.80 ± 0.11	10.28 ± 0.39	10.03 ± 0.07	10.08 ± 0.16	10.00 ± 0.00	10.00 ± 0.00
(last 10)	client	2.50 ± 1.27	2.00 ± 1.05	7.20 ± 3.88	9.70 ± 0.67	9.20 ± 1.62	10.00 ± 0.00	10.00 ± 0.00
median	agent	11.00	11.00	10.00	10.00	10.00	10.00	10.00
	client	0.00	0.00	10.00	10.00	10.00	10.00	10.00
median	agent	11.00	11.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	0.00	0.00	10.00	10.00	10.00	10.00	10.00

Table A.21: PD experiments with client strategy (co), timeout=120.0, discount $\gamma = 0.99$.

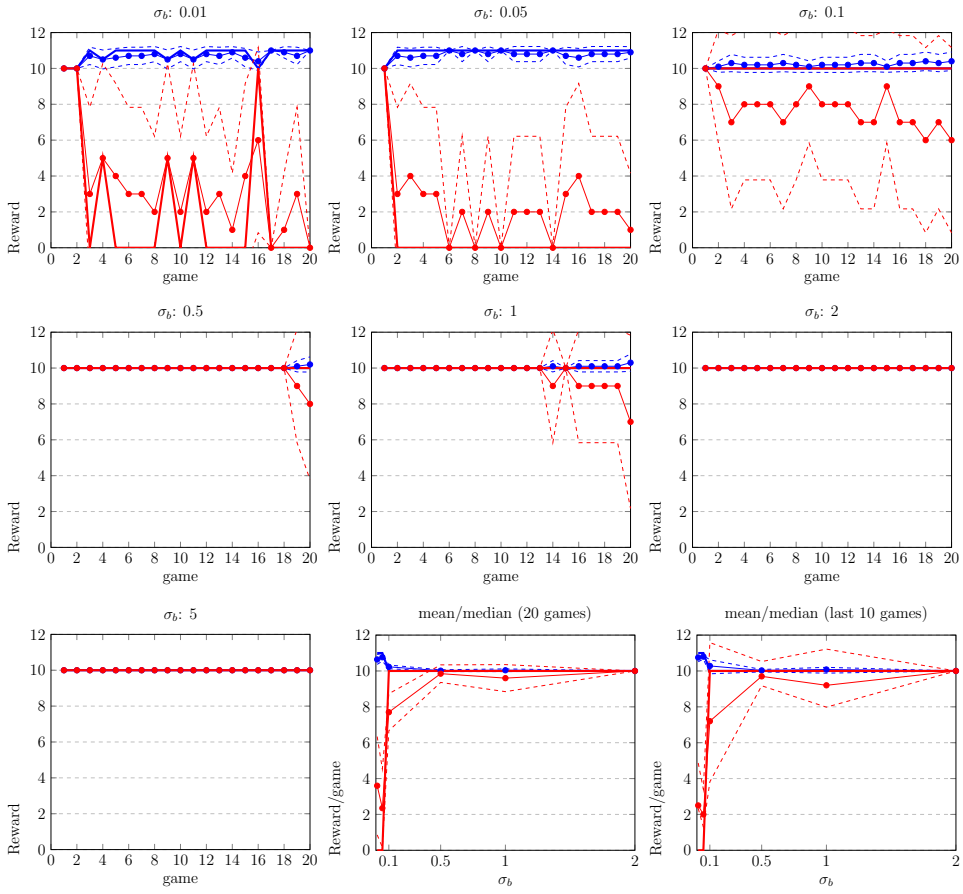


Figure A.17: PD experiments with client strategy (co), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	0.71 ± 0.23	0.74 ± 0.31	0.78 ± 0.26	0.78 ± 0.24	0.73 ± 0.25	0.76 ± 0.27	0.73 ± 0.24
	client	3.85 ± 2.32	3.60 ± 3.10	3.25 ± 2.55	3.20 ± 2.38	3.65 ± 2.48	3.40 ± 2.74	3.70 ± 2.36
mean	agent	0.81 ± 0.11	0.88 ± 0.06	0.87 ± 0.13	0.86 ± 0.13	0.83 ± 0.14	0.89 ± 0.12	0.79 ± 0.12
(last 10)	client	2.90 ± 1.10	2.20 ± 0.63	2.30 ± 1.25	2.40 ± 1.35	2.70 ± 1.42	2.10 ± 1.20	3.10 ± 1.20
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.22: PD experiments with client strategy: (de), timeout=120.0, discount $\gamma = 0.99$.

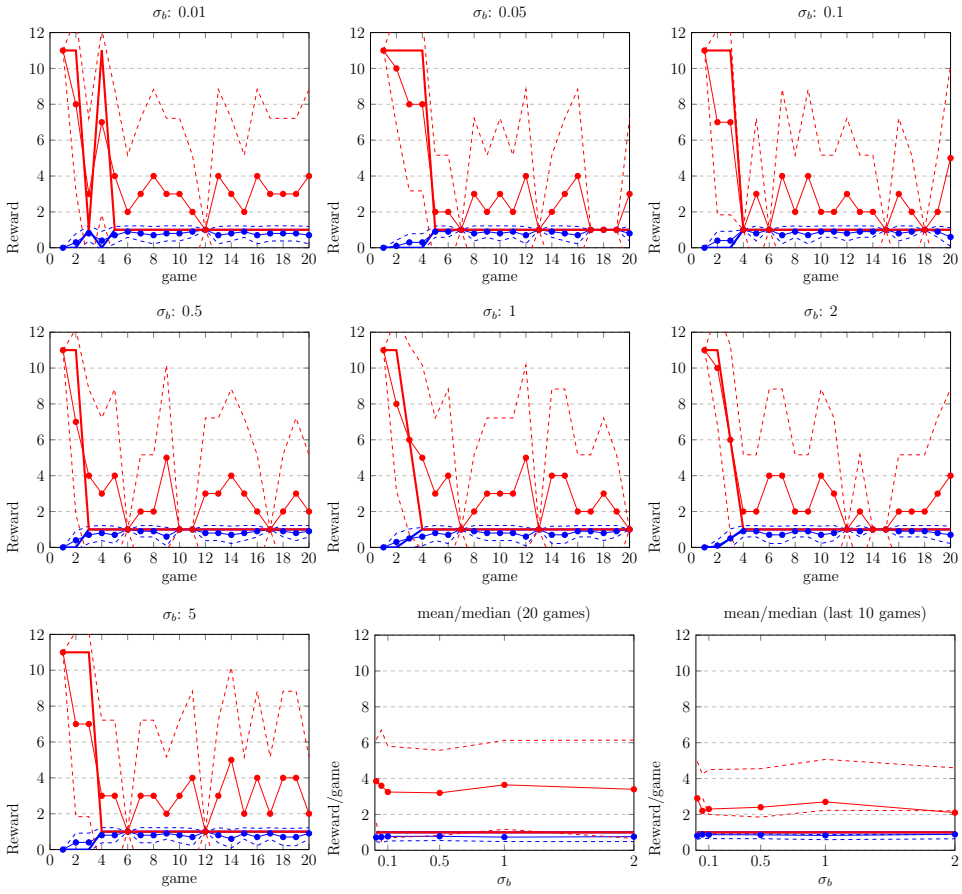


Figure A.18: PD experiments with client strategy: (de), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	1.69 ± 2.85	1.76 ± 2.93	1.70 ± 2.89	1.62 ± 2.88	1.52 ± 2.91	1.60 ± 2.89	1.60 ± 2.89
	client	4.05 ± 2.31	3.30 ± 2.32	3.95 ± 3.10	4.70 ± 3.74	5.65 ± 3.08	4.85 ± 3.38	4.85 ± 3.42
mean	agent	0.77 ± 0.13	0.85 ± 0.12	0.88 ± 0.09	0.88 ± 0.11	0.72 ± 0.18	0.82 ± 0.16	0.82 ± 0.14
(last 10)	client	3.30 ± 1.34	2.50 ± 1.18	2.20 ± 0.92	2.20 ± 1.14	3.80 ± 1.81	2.80 ± 1.62	2.80 ± 1.40
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00
median	agent	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(last 10)	client	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table A.23: PD experiments with client strategy (to), timeout=120.0, discount $\gamma = 0.99$.

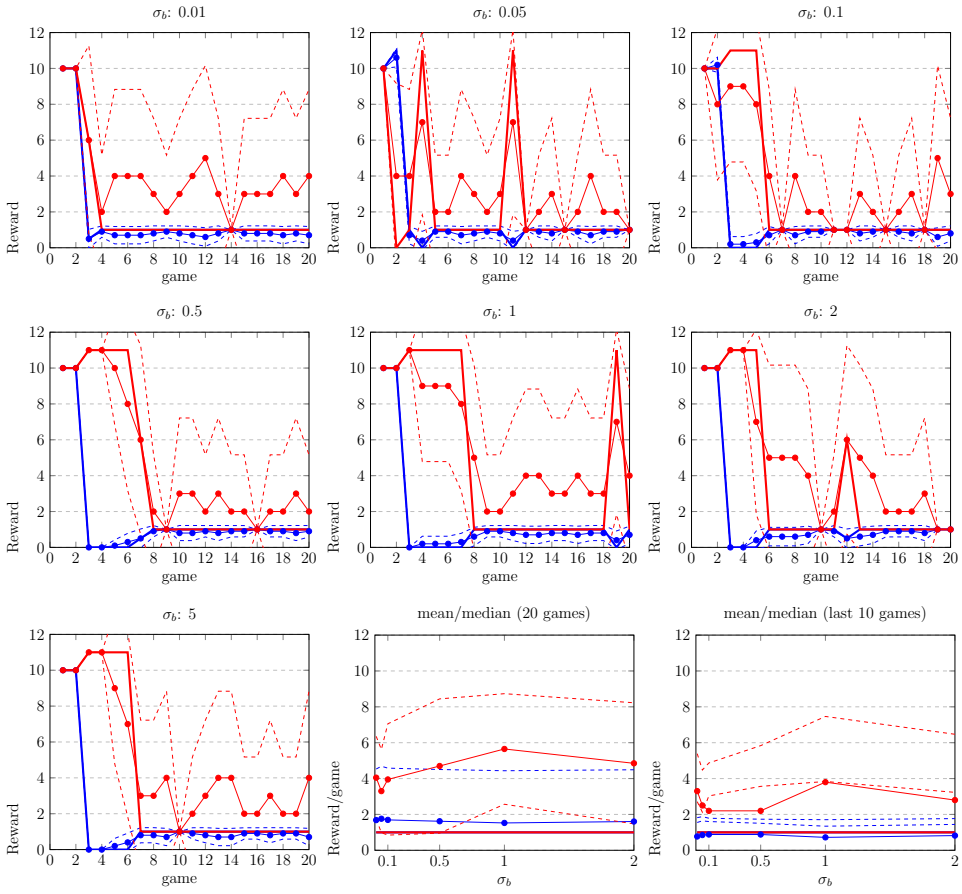


Figure A.19: PD experiments with client strategy (to), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	4.67 ± 2.76	3.42 ± 2.83	5.93 ± 1.84	9.34 ± 0.50	9.93 ± 0.25	9.82 ± 0.40	9.78 ± 0.44
	client	4.34 ± 2.39	2.98 ± 2.25	5.66 ± 1.49	9.18 ± 0.68	9.76 ± 0.69	9.77 ± 0.43	9.72 ± 0.47
mean	agent	3.87 ± 1.04	1.80 ± 0.69	4.96 ± 3.89	9.03 ± 2.83	9.85 ± 0.34	9.65 ± 1.11	9.65 ± 1.11
(last 10)	client	3.98 ± 1.16	1.91 ± 0.62	4.96 ± 3.92	8.81 ± 2.82	9.52 ± 0.81	9.54 ± 1.45	9.54 ± 1.45
median	agent	1.00	1.00	10.00	10.00	10.00	10.00	10.00
	client	1.00	1.00	10.00	10.00	10.00	10.00	10.00
median	agent	1.00	1.00	1.00	10.00	10.00	10.00	10.00
(last 10)	client	1.00	1.00	1.00	10.00	10.00	10.00	10.00

Table A.24: PD experiments with client strategy (tt), timeout=120.0, discount $\gamma = 0.99$.

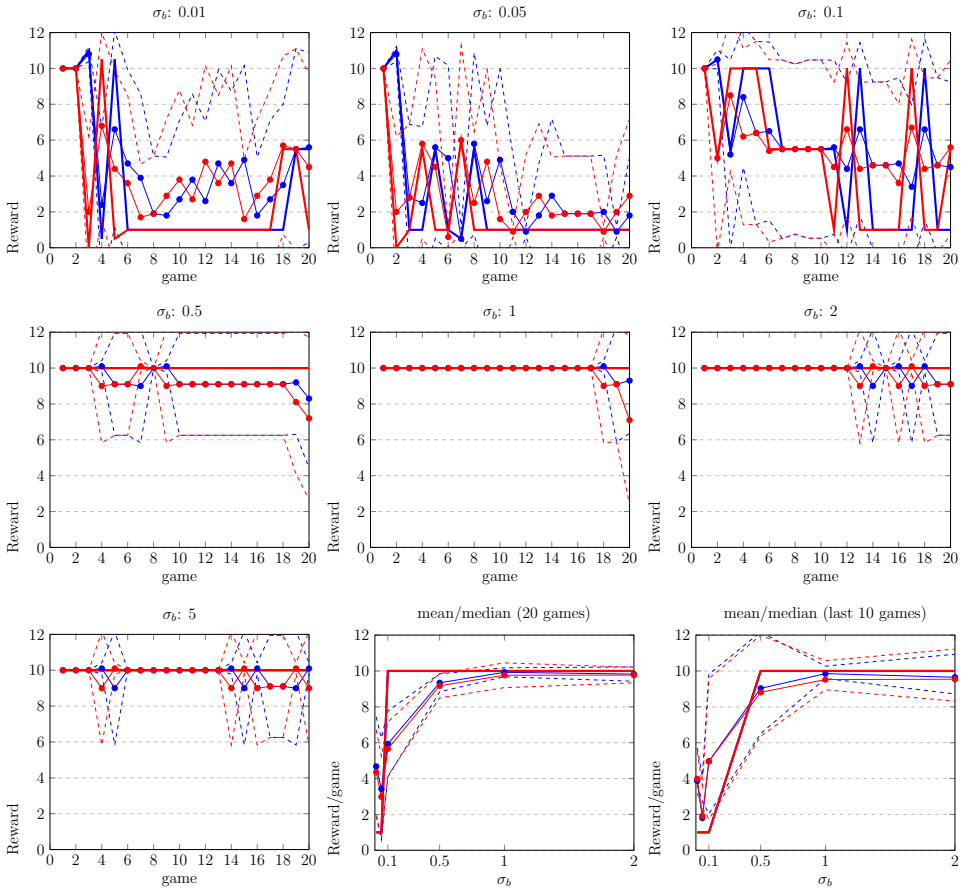


Figure A.20: PD experiments with client strategy: (tt), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	7.46 ± 2.33	4.83 ± 2.80	8.11 ± 1.18	9.90 ± 0.43	10.02 ± 0.05	10.02 ± 0.04	10.00 ± 0.00
	client	5.16 ± 2.58	2.75 ± 2.12	7.17 ± 1.03	9.46 ± 1.12	9.85 ± 0.49	9.85 ± 0.37	10.00 ± 0.00
mean	agent	5.59 ± 2.47	3.73 ± 2.67	7.43 ± 3.42	9.81 ± 0.33	10.03 ± 0.05	10.03 ± 0.05	10.00 ± 0.00
(last 10)	client	3.61 ± 1.67	2.41 ± 1.32	6.99 ± 3.92	8.93 ± 1.55	9.70 ± 0.48	9.70 ± 0.48	10.00 ± 0.00
median	agent	10.00	1.00	10.00	10.00	10.00	10.00	10.00
	client	1.00	1.00	10.00	10.00	10.00	10.00	10.00
median	agent	1.00	1.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	1.00	1.00	10.00	10.00	10.00	10.00	10.00

Table A.25: PD experiments with client strategy (t2), timeout=120.0, discount $\gamma = 0.99$.

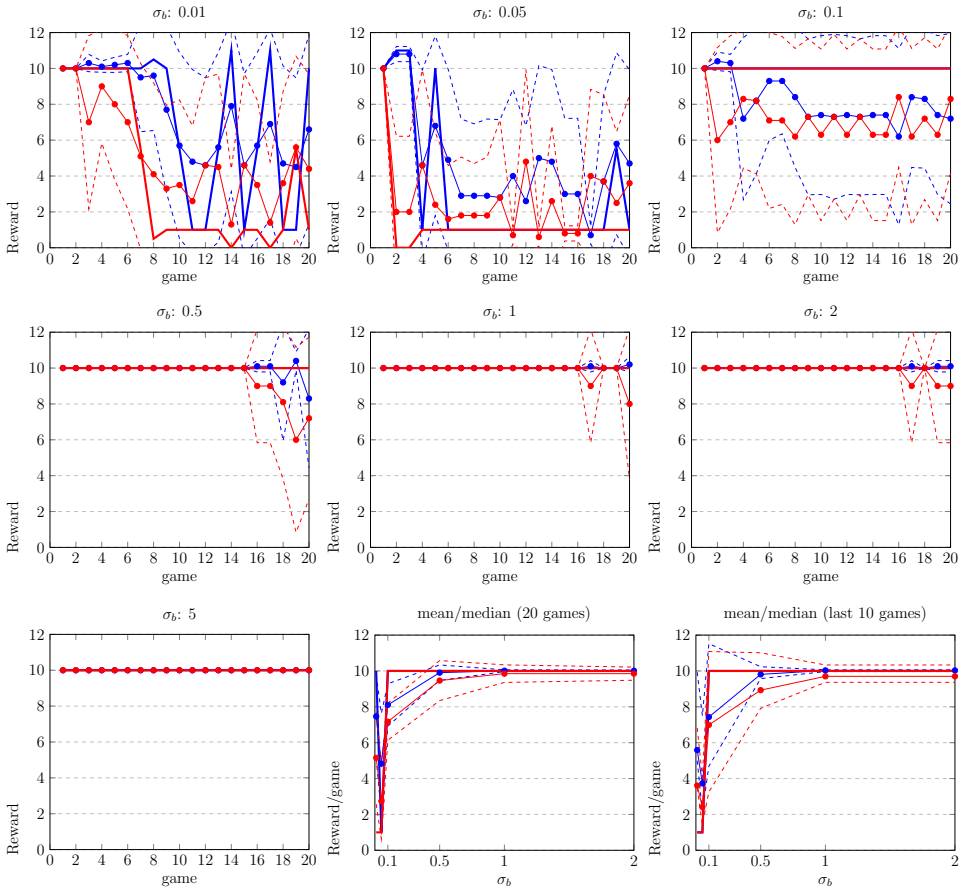


Figure A.21: PD experiments with client strategy (t2), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	2.63 ± 3.39	2.69 ± 3.01	8.41 ± 0.83	9.76 ± 0.99	9.82 ± 0.40	9.78 ± 0.58	9.90 ± 0.31
	client	3.96 ± 2.56	3.85 ± 2.08	8.57 ± 1.21	9.48 ± 1.47	9.82 ± 0.40	9.72 ± 0.60	9.96 ± 0.23
mean	agent	1.52 ± 0.57	1.12 ± 0.59	8.01 ± 3.62	9.51 ± 0.63	9.64 ± 1.14	9.56 ± 0.96	9.81 ± 0.60
(last 10)	client	2.73 ± 1.17	2.77 ± 1.35	8.12 ± 2.66	8.96 ± 0.82	9.64 ± 1.14	9.45 ± 1.16	9.92 ± 0.25
median	agent	1.00	1.00	10.00	10.00	10.00	10.00	10.00
	client	1.00	1.00	10.00	10.00	10.00	10.00	10.00
median	agent	1.00	1.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	1.00	1.00	10.00	10.00	10.00	10.00	10.00

Table A.26: PD experiments with client strategy (2t), timeout=120.0, discount $\gamma = 0.99$.

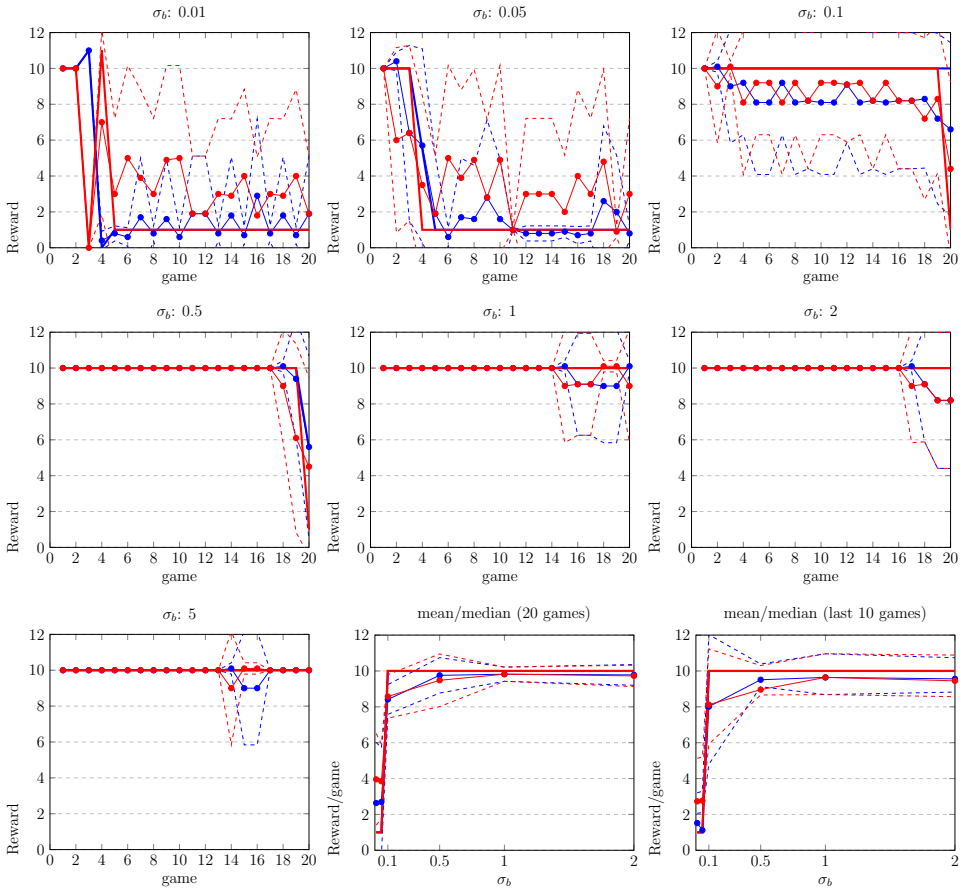


Figure A.22: PD experiments with client strategy (2t), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	9.71 ± 0.51	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00
	client	9.38 ± 0.51	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00
mean	agent	9.36 ± 2.02	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00
(last 10)	client	9.47 ± 1.68	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.27: PD experiments with client strategy (1.0), timeout=120.0, discount $\gamma = 0.99$.

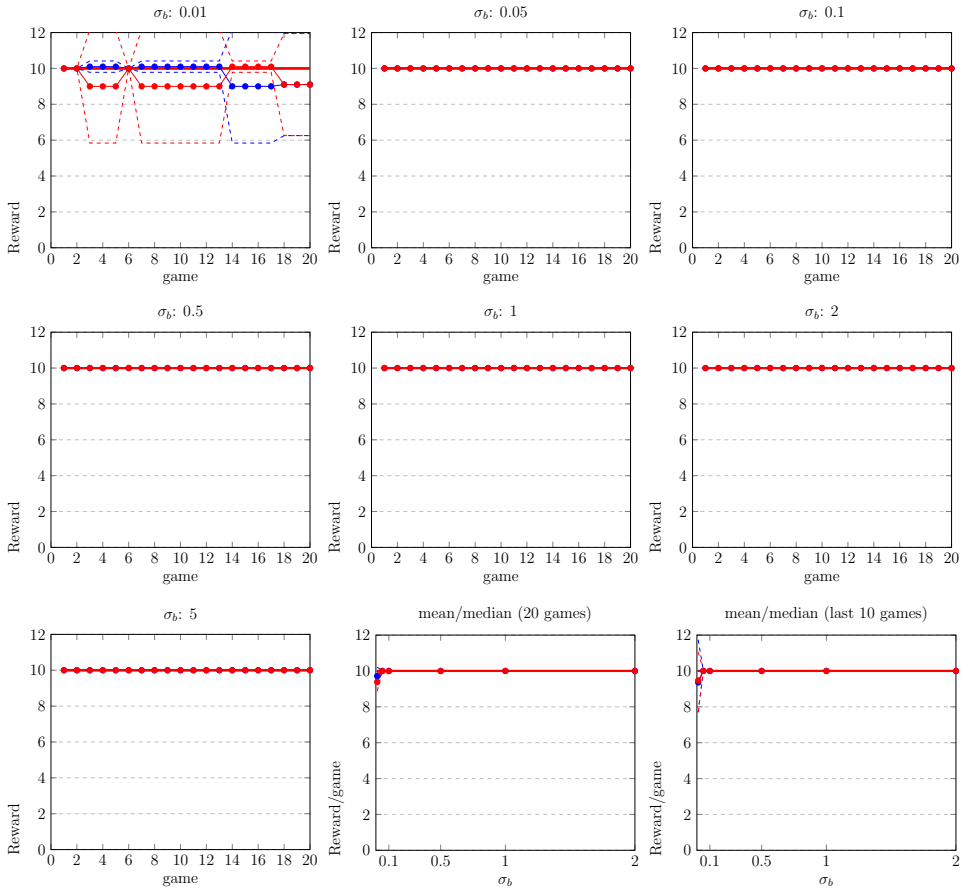


Figure A.23: PD experiments with client strategy (1.0), timeout=120.0, discount $\gamma = 0.99$. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

		σ_b						
		0.01	0.05	0.10	0.50	1.00	2.00	5.00
mean	agent	10.00 ± 0.00	9.11 ± 0.80	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.01 ± 0.02	9.99 ± 0.22
	client	10.00 ± 0.00	9.00 ± 0.79	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	9.95 ± 0.22	9.61 ± 0.59
mean	agent	10.00 ± 0.00	8.68 ± 2.79	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	9.96 ± 0.16
(last 10)	client	10.00 ± 0.00	8.35 ± 3.51	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	10.00 ± 0.00	9.41 ± 1.55
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00
median	agent	10.00	10.00	10.00	10.00	10.00	10.00	10.00
(last 10)	client	10.00	10.00	10.00	10.00	10.00	10.00	10.00

Table A.28: PD experiments with client strategy (same), timeout=120, discount γ=0.99.

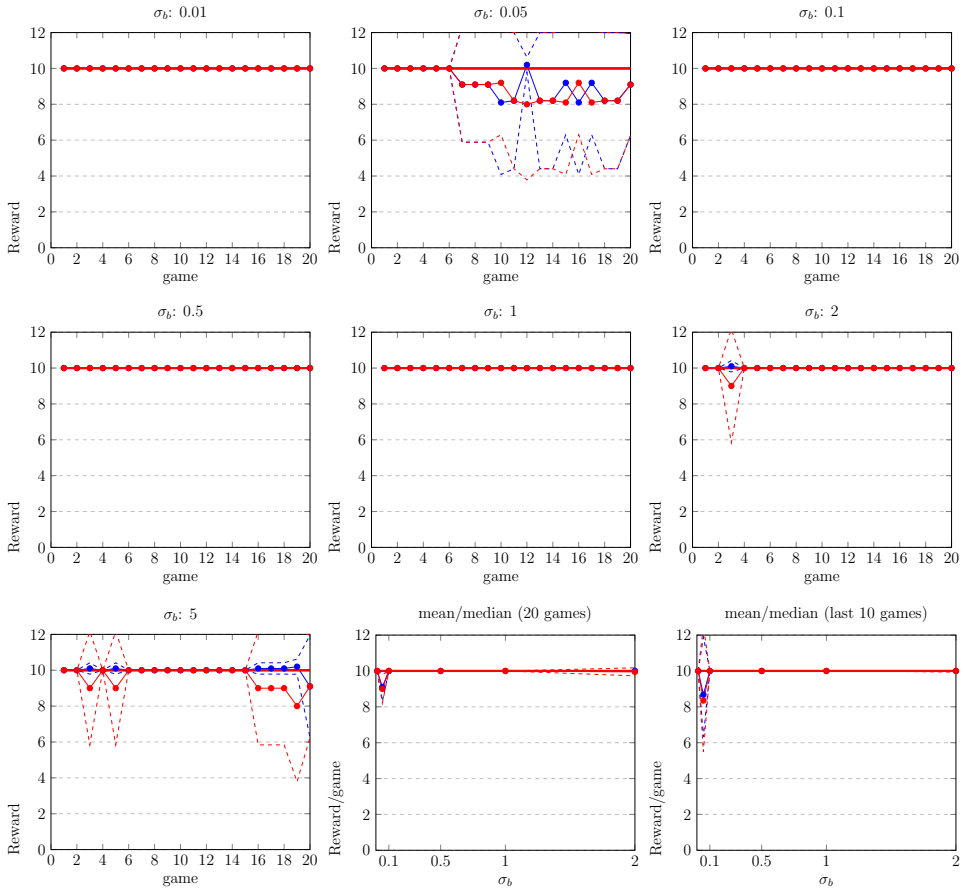


Figure A.24: PD experiments with client strategy (same), timeout=120.0, discount γ = 0.99. Red=client, Blue=agent, dashed=std.dev. solid (thin, markers): mean, solid (thick): median.

Appendix B

ACT-based Dialogue Response Generation: Additional Qualitative Experiments

This appendix consists of additional experiments to evaluate the ACT-based neural response generation models (namely, `S2EPA`, `EPA2S-Seq2Seq` and `EPA2S-CVAE`) presented in Chapter 4.

B.1 Assessing `S2EPA`

First, I assess the quality of EPA vectors produced by the `S2EPA` model. Some example sentences from the Cornell test set are shown in Table B.1, along with their EPA predictions produced by `S2EPA`. I also include the closest word labels for each EPA from the ACT lexicon of behaviours.

We note that the model's EPA predictions are generally appropriate, and in many cases they are in alignment with the ACT behaviour labels. For instance, *'i think i am in love'* is fairly positive due to the presence of the word *love*; it is moderately potent and slightly active because of the phrase *i think*. The closest labels in the ACT lexicon are *caution* and *collaborate with*. Among these, *caution* seems to describe the input well. A similar phenomenon is seen for the input *'i hate you'*, whose EPA prediction closely matches the ACT labels *malign, injure*. An interesting case is *'i have no fear of failure'*: it has two

Sentence	EPA	Closest ACT Labels
i think i am in love	[1.60, 0.95, 0.55]	caution, collaborate with
i hate you	[-1.63, 0.85, 0.49]	malign, injure
i have no fear of failure	[0.64, 1.27, 0.80]	train, confront
what the hell are you doing?	[-1.64, 0.41, 1.39]	badger, club
he's determined, unstoppable	[0.66, 1.87, 1.45]	apprehend, challenge
what do i do for fun?	[-0.35, -0.21, -0.04]	poke, gawk at
will you have dinner with me?	[0.91, 0.45, 0.79]	concur with, jest with
please don't talk with food in your mouth	[-0.82, 0.10, -0.64]	defer to, monitor
i insist on being told exactly what you have in mind	[0.06, 0.03, 0.13]	joggle, beckon to
you go ahead and relax, i'll cook	[0.95, 0.32, 0.47]	pay for, concur with
i've been thinking about you	[1.59, 1.12, 0.66]	caution, collaborate with
you are despicable	[-1.74, 0.86, 0.94]	kick, club
i quit.	[-0.1, 0.89, 0.17]	search, smirk at
how about a drink?	[0.60, 0.42, 1.06]	query, jest with
there is nothing for me here anymore	[-0.56, 0.30, 0.14]	flee, sound out

Table B.1: Examples of EPA vectors (and their closest word labels in ACT) produced for input sentences by `S2EPA`.

negative and strong words *fear* and *failure*. Yet, the model correctly predicts that the overall sentiment is positive and powerful, and is described well by the label *confront*.

We also see some negative examples. The E value of *'i quit'* is -0.1, but it should be much more negative. The closest ACT labels *search* and *smirk at* don't make sense either. Similarly, the input *'i've been thinking about you'* is composed of fairly neutral individual words; however the model correctly predicts that overall the sentence is positive, moderately potent and slightly active. On the other hand, its ACT labels *caution* and *collaborate with* are not appropriate.

To quantify the success rate of alignment between sentences and `S2EPA`'s predictions, I take a test set of 100 sentences (from the Cornell corpus, as before) and ask 3 human judges to classify the EPA predictions as 'good' or 'bad'. Averaging these ratings yields a score of 0.739, indicating that 73.9% of the EPA predictions (on the test set) are in alignment with the input sentence affect. Thus, overall the `S2EPA` module works reasonably well.

B.2 Assessing `EPA2S`

Next, I would like to analyse the affective quality of responses produced by the two variants of the `EPA2S` model: Seq2Seq and CVAE.

First, I present the outputs of the traditional Seq2Seq model (without the $\boldsymbol{\alpha}$ vectors as input). Table B.2 shows its predictions on a small test set of inputs $\boldsymbol{C}$ chosen from the Cornell test set. We observe that the responses are grammatically well-formed but generally short and vague; this is a known shortcoming of naive Seq2Seq models.

$\boldsymbol{C}$	**Traditional Seq2Seq (baseline)**
i hate you	that is okay
i think i'm in love	who is it
i have no fear of failure	i have it in me
how about a drink?	sure why not
i've been thinking about you	that makes it okay
there is nothing for me here anymore	so what do you want

Table B.2: The outputs of traditional Seq2Seq with attention, without $\boldsymbol{\alpha}$ labels.

Next, I add an EPA vector $\boldsymbol{\alpha}$ to each training sample; in this experiment the $\boldsymbol{\alpha}$ is chosen manually from the ACT lexicon. Table B.3 shows the input prompts, target EPA vectors and the corresponding text generated by `EPA2S-Seq2Seq` and `EPA2S-CVAE`.

Line #	$\boldsymbol{C}$	$\boldsymbol{\alpha}$ **(Manually Chosen)**	`EPA2S-Seq2Seq`	`EPA2S-CVAE`
1	i hate you	[1.71,1.39,-0.90] (calm)	you know me	what do you want
2		[-0.50,0.72,0.81] (criticize)	okay	man can you scream
3		[-0.83,-0.93,0.44] (hide from)	you write a proper part for me	i feel so tired
4	i think i'm in love	[0.98,0.38,0.02] (agree with)	who is it	i don't really know you
5		[-1.39, -0.47, 2.15] (laugh at)	wait up please	yeah but don't make any noise
6		[-1.53,-0.20,-0.19] (ignore)	i don't think so	we should find a leader to fight
7	i have no fear of failure	[2.14,1.21,-0.17] (appreciate)	yes	i believe it when you say
8		[-1.61,0.66,1.25] (antagonize)	i don't know	i need to leave early tomorrow
9		[1.90,0.82,-0.11] (smile at)	what do you say	i know you, <unk>
10	how about a drink?	[0.98,0.38,0.02] (agree with)	sure that's nice	let me see what i can do about you
11		[-1.05,-0.69,0.33] (avoid)	i'm sorry i can't	there is something on the clouds
12		[1.18, 1.47, 0.20] (charm)	how long have you been awake	i'm going with you baby
13	there is nothing for me here anymore	[2.12, 1.12, -0.81] (comfort)	yeah you know me	it is better this way
14		[1.64, 1.17, 0.47] (encourage)	no it is	it's not too late to try
15		[1.27, 1.14, 1.44] (entertain)	not now	you need to calm down

Table B.3: Example outputs generated by `EPA2S` for a given input sentence and EPA vector.

Similar to the Seq2Seq baseline, we see short and non-committal responses by `EPA2S-Seq2Seq`. As far as their quality and relevance is concerned, we see some positive examples (Lines 1, 4, 6, 7, 10, 11, 14) where the output sentences are well-aligned with the inputs $\boldsymbol{C}$ and $\boldsymbol{\alpha}$; the rest of the examples show output that is syntactically coherent but does not align well with either $\boldsymbol{C}$ or $\boldsymbol{\alpha}$ or both. For instance, in Line 2, *'okay'* is a valid response to *'i hate you'*, but it does not correspond to criticizing. Similarly, in Line 5, the response *'wait*

up please' is not relevant to the input *'i think i'm in love'* or the target affect of *laugh at*. Overall, the results are pretty evenly divided between positive and negative examples.

We see similar results for `EPA2S-CVAE`. There are some positive examples (Lines 2, 3, 7, 12, 13, 14). On the other hand we see several outputs that are contextually relevant but affectively misaligned (Lines 1, 4, 5, 15). The responses are generally longer and less vague than baseline Seq2Seq and `EPA2S-Seq2Seq`.

To quantify the performance of the two `EPA2S` variants, I set up an experiment as follows. Given a test set of 100 sentences and the desired $\boldsymbol{\alpha}$ vector, I ask 3 human judges to specify whether the predicted response aligns with $\boldsymbol{C}$, $\boldsymbol{\alpha}$, both or none. The results are presented in Table B.4. Overall, the results are evenly distributed across the four classes. Strictly speaking, the success rate (alignment with both $\boldsymbol{C}$ and $\boldsymbol{\alpha}$) is 23.1% and 27.6% respectively for `EPA2S-Seq2Seq` and `EPA2S-CVAE`.

	`EPA2S-Seq2Seq`	`EPA2S-CVAE`
% Alignment with $\boldsymbol{C}$ and $\boldsymbol{\alpha}$	23.1	27.6
% Alignment with $\boldsymbol{C}$ only	25.5	22.0
% Alignment with $\boldsymbol{\alpha}$ only	22.6	20.7
% Alignment with neither $\boldsymbol{C}$ nor $\boldsymbol{\alpha}$	28.8	29.7

Table B.4: Evaluating the two `EPA2S` variants.

B.3 Assessing the Full ACT Dialogue Pipeline

I now test the full model (the dialogue pipeline shown in Figure 4.3), where the two modules `S2EPA` and `EPA2S` are integrated with ACT[1]. That is to say, the target EPA vectors $\boldsymbol{\alpha}$ are produced by ACT. I use two ACT settings for identities: *friend-friend* and *friend-enemy*. The quantitative results are presented in the main chapter (Table 4.4). Here, I present the qualitative results.

I first examine the setting where the ACT identity of both interactants is *friend*. The results are shown in Table B.5. We see that ACT produces target actions that are very friendly and nice (e.g. *care for, thank, kiss, embrace*). This is consistent with the responder's identity of *friend*. As far as the response quality is concerned, we see mixed results as before. Both Seq2Seq and CVAE produces responses that are generally well-formed and

[1]The ACT software, called INTERACT, is publicly available at http://www.indiana.edu/~socpsy/ACT/interact.htm.

relevant to the input prompt $\boldsymbol{C}$, but they often seem to ignore $\boldsymbol{\alpha}$. Though the affective interpretation of the responses is very subjective, we observe that Seq2Seq produces emotionally aligned responses in Lines 2 and 4, while CVAE produces affectively appropriate results on Lines 1, 5 and 6. I also include the ACT deflection values in the table for the sake of completeness.

In the second setting, I set the ACT identities of prompter and responder to *friend* and *enemy* respectively. The results are presented in Table B.6. We observe that the actions predicted by ACT are not as friendly anymore (*giggle at, disagree with, bellow at, be sarcastic with*); these behaviours are consistent with the responder's identity of *enemy*. Once again, we see that the responses don't align with $\boldsymbol{\alpha}$ in many cases. The positive examples for Seq2Seq are Lines 5 and 6; those for CVAE are Lines 1, 2 and 6.

Line	$\boldsymbol{C}$	$\boldsymbol{\alpha}$ **(ACT) & Closest ACT Labels**	**Defl.**	`EPA2S-Seq2Seq`	`EPA2S-CVAE`
1	i hate you	[2.52, 2.52, -0.41] (care for, caress)	17.09	that's not the point	you must be tired now
2	i think i'm in love	[3.13, 1.70, 1.39] (thank, kiss)	1.84	i'm glad you like it	i wouldn't do you if i were you
3	i have no fear of failure	[3.72, 1.90, 1.3] (thank, propose marriage to)	4.36	well that's me	i will ride with you love
4	how about a drink?	[3.37, 1.68, 0.92] (reward, thank)	4.06	sure that's nice	i have money
5	i've been thinking about you	[3.12, 1.96, 1.31] (thank, kiss)	1.87	okay	i like you
6	there is nothing for me here anymore	[3.55, 1.99, 0.45] (embrace, propose marriage to)	9.05	i don't think so	it is better this way

Table B.5: The full ACT conversational model with ACT identities *friend-friend*.

Line	$\boldsymbol{C}$	$\boldsymbol{\alpha}$ **(ACT) and Closest ACT Labels**	**Defl.**	`EPA2S-Seq2Seq`	`EPA2S-CVAE`
1	i hate you	[-0.14, 0.43, 0.73] (rib, giggle at)	7.31	that is okay	man you can scream
2	i think i'm in love	[-0.03, 0.52, 0.62] (giggle at, disagree with)	6.33	I don't doubt it	I have second thoughts
3	i have no fear of failure	[-0.25, 0.28, 0.77] (rib, bellow at)	4.78	you are right	take care of you
4	how about a drink?	[-0.38, 0.47, 0.75] (rib, bellow at)	4.39	where?	let me see what i can do
5	i've been thinking about you	[-0.13, 0.17, 0.47] (laud, josh)	6.27	that is great	i believe in it
6	there is nothing for me here anymore	[-0.20, 0.32, 1.07] (be sarcastic toward, banter with)	5.48	wait for me	it is better to calm down

Table B.6: The full ACT conversational model with ACT identities *friend-enemy*.

Overall, it can be concluded that the performance of ACT response generation is not significantly better than the baseline. This can be attributed to the underwhelming performance of `EPA2S` models.