State Construction in Reinforcement Learning

by

Banafsheh Rafiee

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In reinforcement learning, the notion of state plays a central role. A reinforcement learning agent requires the state to evaluate its current situation, select actions, and construct a model of the environment. In the classic setting, it is assumed that the environment provides the agent with the state. However, in most cases of interest, the information received from the environment only provides partial information about the state of the environment.

Ideally, the agent would construct the state directly from the data stream of its interaction with the environment. The prevalent approach to state construction is to train a large neural network with backpropagation and representing the state as the hidden state of recurrent neural networks or the last hidden layer of feed-forward networks. Building upon this approach, the existing solution methods have made a lot of progress. However, they remain limited due to several reasons such as the problem of loss of plasticity in neural networks.

The first contribution of this thesis is the proposal of three diagnostic benchmarks inspired by animal learning for studying state construction. The diagnostic benchmarks have a simple setting: there are only a few signals to make predictions about. However, they are complicated because complex computational models are required to solve them. The proposed benchmarks include knobs for controlling the level of difficulty of the problem.

The second contribution of this thesis is empirical. We conduct a comprehensive empirical study of the prominent recurrent learning methods, illu-

minating some of the limitations of existing solution methods. The empirical study suggests that: 1) None of the methods are fully satisfactory. 2) Recurrent neural networks (RNNs) can be expensive in terms of memory and computation. 3) RNNs trained with truncated backpropagation through time are sensitive to the truncation parameter. 4) Augmenting RNNs with stimulating traces of the observation signals can make T-BPTT less sensitive to the truncation parameter.

The third contribution of the thesis is on the topic of auxiliary task discovery. Learning about tasks auxiliary to the main task of maximizing the sum of discounted rewards can assist state construction. It would be appealing if the agent could discover useful auxiliary tasks automatically. In this work, we propose a method for auxiliary task discovery based on the idea of generate-and-test. Our proposed method continually generates auxiliary tasks, evaluates them, and replaces the useless auxiliary tasks with newly generated ones. We show the efficacy of the proposed method empirically.

Preface

Many parts of this thesis were published and co-authored.

Much of what is presented in Chapter 3, 4, and 5 are based on a paper published at the Adaptive Behavior journal (Rafiee et al., 2022). The results on transformer models are, however, new and were not included in the Adaptive Behavior paper. I was involved in all steps of this research including designing and running experiments, implementing algorithmic ideas, and writing the paper. My co-authors were also involved in different parts of the project. Zaheer Abbas, Sina Ghiassian, and Raksha Kumaraswamy took part in running experiments as well as writing the paper. Adam White had a significant role in designing and analyzing the experiments. Rich Sutton and Elliot Ludvig contributed to designing experiments using their expertise in animal learning.

Chapter 6 is based on a published paper (Rafiee et al., 2023). I was involved in all parts of this research including the algorithmic and experimental aspects as well as the paper write-up. My co-authors contributed to different aspects of the paper. Rich Sutton had a major contribution to the design of the proposed algorithm. Adam White significantly contributed to the design and analysis of the experiments.

To my parents

Broch is an inspiration to us not only because of what he accomplished, but also because of all that he aimed at and could not attain.

– Milan Kundera.

Acknowledgements

I would like to express my sincere appreciation for my supervisor, Rich Sutton, whose guidance has been essential throughout my PhD. Rich stands out as more than just a scientist; he is a profound thinker. He has taught me to be ambitious and strive for excellence.

I would like to thank my supervisor, Adam White, for his support and encouragement. From my first collaboration with him as a Master student, he has been actively involved, supporting me in all parts of the projects. He taught me to embrace disappointing results as opportunities for growth. I am grateful for his impactful mentorship.

I am grateful to the members of my examining committee, Aaron Courville, Levi Lelis, and Dale Schuurmans for the insightful discussions and useful feedback during the defense.

I want to express my thanks to my coauthors, Zaheer Abbas, Raksha Kumaraswamy, Elliot Ludvig, Jun Jin, and Jun Luo for their contributions to the works included in this thesis.

I would like to thank all the members of RLAI especially Khurram Javed and Esraa Elelimy for all the useful discussions and all that I learned from them.

Finally, I would like to thank Sina Ghiassian for all his support both in work and in life during my PhD.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A central notion in reinforcement learning is the notion of state. A reinforcement learning agent learns to maximize the sum of future rewards while continually interacting with its environment. At each time step, the environment is in a state, the agent takes an action, the environment in turn emits a reward and transitions to the next state. The agent requires the state for all of its activities: for action selection, for evaluating how good its current situation is, and for constructing a model of its environment.

In the classic setting of reinforcement learning, it is assumed that the agent has access to the state of the environment. In most cases of interest, however, the information received from the environment only provides partial information about the environment state. These cases are known as the case of partial observability. Even in cases where the agent fully observes the environment state, the environment state could be arbitrarily large, and a nontrivial mapping from the environment state to the state representation is required.

It would be appealing if the agent could construct the state directly from the information provided by the environment. Automating the construction of the state is in line with the long standing goal of building agents capable of learning directly from the data stream of experience. Moreover, developing agents that can construct the state directly from the data would significantly enhance the generality and applicability of reinforcement learning.

In order to generalize to the setting where the agent is responsible for constructing the state, we assume that the environment only provides obser-

vations and not the environment state. More specifically, at every time step t, the agent receives observation $\mathbf{o}_t \in \mathbb{R}^d$ which does not necessarily capture the current state of the environment. The agent has to construct its own state, called the *agent state*, from the history of observations and actions where the state is a compact summary of the history that is sufficient for predicting the future.

1.1 The state update function

A computationally appealing approach to constructing the agent state is to construct it recursively from the previous agent state:

$$S_{t+1} \doteq u(S_t, A_t, O_{t+1})$$

where u is called the *state update* function. The state update function covers the case of full observability as well. It is assumed in that case that the information in O_{t+1} is sufficient for forming the agent state and S_t and A_t are no longer required.

There is a large body of work on state construction each with its own development of the state update function. Some theoretical approaches include the works on partially observable Markov decision processes (POMDP), predictive state representations, and Observable Operator Models (Monahan,1982; Littman, Sutton, and Singh, 2001; Thon, 2017). The more common approach to state construction is to train a sufficiently large neural network with back-propagation. In this case, the state is represented either as the hidden state of a recurrent neural network (RNN) or as the last hidden layer of a feed-forward network. While the existing solution methods have hints to the final solution, they are not fully satisfactory due to several reasons such as the problem of loss of plasticity in neural networks (Dohare et al., 2023). We will discuss some of the existing methods in more detail in Chapter 4.

1.2 Objective and approach

This thesis aims to answer the following question:

How well do current state construction solution methods perform and how can they be improved?

To tackle this question, we take an empirical approach. Conducting an empirical study of the existing solution methods requires suitable benchmarks. We believe that there is no satisfactory benchmark for studying the problem of state construction in isolation without the need for addressing other confounding factors such as exploration in reinforcement learning. As a first step, we design diagnostic benchmarks for studying the solution methods for state construction. Diagnostic benchmarks are simple issue-oriented benchmarks that illuminate the fundamental limitations of the existing methods. We design the benchmarks such that they include knobs to control the level of difficulty of the problem. The proposed benchmarks can facilitate research on state construction enabling researchers to quickly evaluate new ideas.

Note that our proposed benchmarks are different from time series forecasting benchmarks (e.g. the M-Competitions by Makridakis et al. (1982)) in that our focus is on the problem of multi-step prediction. Multi-step predictions are different from time series forecasting problems in that they are predictions about the discounted sum of future observations as opposed to the next observation itself. Multi-step predictions enable a class of solutions called temporal-difference learning that are efficient because they have a recursive form and can be answered independent of span.

To pursue our goal of understanding how well the current state construction methods perform, we study them empirically on the proposed benchmarks. For the empirical study, we are interested in gaining a more granular understanding of how the methods perform. We will explore questions such as to what extent they fail or succeed, and how their performance changes as a function of their key hyper-parameters.

Another topic that we explore is the effect of auxiliary tasks on state construction. State construction can be assisted by learning about prediction and

control tasks auxiliary to the main task of maximizing the sum of discounted rewards while sharing the state representation. These tasks, called *auxiliary tasks*, exert pressure on the lower layers of the neural network during training, yielding agents that can learn faster (Mirowski et al., 2016; Shelhamer et al., 2016), produce better final performance (Jaderberg et al., 2016), and at times transfer to other related problems (Wang et al., 2022). A common view is that the positive influence of auxiliary tasks is related to the emergence of good internal representations that are shared for learning the main task and the auxiliary tasks. This positive influence is called the *auxiliary effect*.

To improve the *auxiliary effect* on state construction, we explore the topic of auxiliary task discovery. It would be appealing if the agent could automatically discover useful auxiliary tasks for itself over time. Relying on human experts for designing auxiliary tasks is not ideal because it is challenging to know what auxiliary tasks will be useful in advance. Moreover, the usefulness of auxiliary tasks might change over the course of learning.

Despite significant interest, the problem of auxiliary task discovery is rather unexplored with only a few existing solution methods that discover auxiliary tasks systematically. This dissertation presents a new method for auxiliary task discovery based on the idea of generate-and-test.

1.3 Contributions

This thesis includes three main contributions:

Diagnostic benchmarks for state construction (Chapter 3)
We designed three diagnostic benchmarks inspired by animal learning experiments. The first benchmark, *trace conditioning*, requires an agent to predict a distal stimulus from a previously observed cue, just as a rabbit predicts an air puff based on a tone. The challenge here is representational: how does the agent bridge the gap between the tone and the air puff in a way that is not specific to the particular arrangement or timing of the stimuli? (Ludvig et al., 2012; Sutton and Barto, 2018) The second benchmark, *noisy patterning*,

is inspired by biconditional patterning experiments (Mackintosh, 1974; Harris et al., 2008). This benchmark tests the agent's ability to determine which configuration of the observation signals to pay attention to, in the presence of noise and distracting stimuli. Finally, the third benchmark, *trace patterning*, combines trace conditioning and noisy patterning and requires the agent to simultaneously discover the relevant observation signals and build their temporal representations. The trace patterning benchmark is more complicated than the sum of trace conditioning and noisy patterning as it requires temporal representations of the configurations of the observation signals. These benchmarks are useful diagnostic tools for assessing methods for state construction.

Empirical Study of prominent recurrent learning methods on the diagnostic benchmarks (Chapter 5)

We used the proposed diagnostic benchmarks to conduct a comprehensive empirical study of prominent recurrent learning methods, including Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014). We combined these methods with training algorithms of Truncated Back-prop Through Time (T-BPTT) (Williams and Peng, 1990) and Real Time Recurrent Learning (RTRL) (Williams and Zisper, 1989). We systematically investigated each method's performance as we varied the key problem parameters. We also introduced a simple input augmentation scheme based on memory traces, improving both T-BPTT and RTRL based methods. In total, the results showed that the proposed diagnostic benchmarks can effectively isolate the limitations of the current training methods and help stimulate research in online representation learning. The insights gained from the experiments include: 1) Recurrent learning algorithms can simultaneously learn the temporal associations and handle nonlinearities; however, they can be expensive in computational and memory requirements. 2) The performance of recurrent learning algorithms trained with T-BPTT is highly sensitive to the truncation parameter, requiring much more computation for learning longer dependencies.

Auxiliary task discovery through generate-and-test (Chapter 6)
We propose an auxiliary task discovery algorithm based on the generate-and-test idea. The idea of generate-and-test has been originally studied in the context of representation learning where a set of features is produced by a random generator and evaluated using a tester (Mahmood and Sutton, 2013). High-utility features are retained and used by the base learning system while low utility features are replaced by newly generated features. We use a similar idea for auxiliary task discovery where the agent continually generates new auxiliary tasks, retains the high-utility ones, and replaces the auxiliary tasks with low-utility. We show the efficacy of the proposed method empirically.

Chapter 2

Background on Reinforcement Learning

This chapter sets the context for understanding the rest of the thesis by providing a background on reinforcement learning and the associated terminology and notation.

2.1 Online multi-step prediction and control

We consider the interaction of an agent with its environment as formalized by Markov Decision Processes (MDPs) for discrete time steps. At each time step t, the environment is in a state $S_t \in \mathcal{S}$, the agent performs an action $A_t \in \mathcal{A}$, the environment in turn emits a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and transitions to the next state S_{t+1}.

The next state and reward are determined according to the transition dynamics of the MDP $p(s', r|s, a)$:

$$p(s', r|s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, a_t = a\}$$

defined for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $r \in \mathcal{R}$.

The agent selects action according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\sum_{a \in \mathcal{A}} \pi(s, a) = 1$, $\forall s \in \mathcal{S}$.

We consider the problems of prediction and control. For the prediction problem, the goal of the agent is to approximate the expected *return* for a given policy π defined as:

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\end{aligned} \tag{2.1}
$$

where $\gamma \in [0, 1)$ is the discount factor.

The expected return starting from each state following a policy forms a function referred to as the *value function* and defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$$

The predictions associated with this notion of a value function are referred to as *multi-step predictions* as they look at multiple steps into the future as opposed to just the immediate next step.

These predictions can be learned *online*, that is updated at every time step, and independent of span (van Hasselt and Sutton, 2015) due to the recursive form of return:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

This is done using temporal-difference learning methods which we will discuss in the following section.

In the control setting, the goal of the agent is to find the *optimal policy* instead of estimating the value function for a given policy. The optimal policy is the policy that maximizes the expected return and is denoted by π^*. In this setting, it is common to use state-action value functions:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

The difference between the value functions and state-action value functions is that for the latter the expectation is conditioned on the state and the action whereas for the former the expectation is only conditioned on the state.

The action-value function corresponding to the optimal policy is called the optimal action-value function and is denoted by q^*:

$$q^*(s, a) \doteq \max_\pi q_\pi(s, a)$$

2.2 Temporal-difference learning

To estimate the value function, it is common to use semi-gradient temporal-difference learning (Sutton, 1988). More specifically, TD(0) is used to learn a parametric approximation $\hat{v}(s; \mathbf{w})$ by updating a vector of parameters $\mathbf{w} \in \mathbb{R}^{\mathbf{d}}$ as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta_t \nabla_{\mathbf{w}} \hat{v}(S_t; w) \tag{2.2}$$

where α denotes the step-size parameter and $\nabla_{\mathbf{w}} \hat{v}(S_t; \mathbf{w})$ is the gradient of the value function with respect to the parameters $\mathbf{w_t}$. δ_t denotes the TD error:

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}; \mathbf{w_t}) - \hat{v}(S_t; \mathbf{w_t})$$

The value function can be parameterized linearly as $\hat{v}(S_t; \mathbf{w_t}) = \mathbf{w_t}^\top \mathbf{x_t}$ with $\mathbf{x_t} \in \mathbb{R}^{\mathbf{d}}$ denoting the feature vector. In that case, $\nabla_{\mathbf{w}} \hat{v}(S_t; \mathbf{w}) = \mathbf{x_t}$.

Alternatively, the value function can be approximated by a neural network in which case $\nabla_{\mathbf{w}} \hat{v}(S_t; \mathbf{w})$ is calculated using backpropagation.

For the control setting, to estimate the state-action value functions, the control variant of TD(0), Q-learning (Watkins and Dayan, 1992) is commonly used. The update rule for Q-learning is similar to that of TD(0); however, $\hat{q}(S_t, A_t, \mathbf{w})$ is used instead of $\hat{v}(S_t, \mathbf{w})$ and the TD error is defined as:

$$\delta_t \doteq R_{t+1} + \gamma \text{max}_a \hat{q}(S_{t+1}, a; \mathbf{w}_t) - \hat{q}(S_t, A_t; \mathbf{w_t})$$

where the action with the highest value is used for forming the target of learning.

Q-learning is commonly integrated with the epsilon-greedy policy for action selection. In the epsilon-greedy policy, the optimal action is selected with $1-\epsilon$ probability, and with probability ϵ one of the actions is selected uniformly randomly. Here, ϵ can be used to control the exploration-exploitation tradeoff. Note that in Q-learning, the policy for which the action-value function is estimated is the greedy policy whereas the policy that is used for action selection is the epsilon-greedy policy.

When the policy being learned about is different from the policy used for action selection, learning is called *off-policy*. The policy being learned is called the target policy and the policy used for action selection is called the behavior policy. Off-policy learning makes it possible to learn about multiple target policies in parallel while behaving according to a single behavior policy. Off-policy learning also facilitates the learning of general value functions which we will discuss in the next section.

2.3 General value functions and nexting predictions

General value functions or GVFs are value functions with a generalized notion of target and termination (Sutton et al., 2011). More specifically, a GVF can be written as the expectation of the discounted sum of any signal of interest given that a specific policy is followed:

$$v_{\pi,\gamma,c}(s) \doteq \mathbb{E}_{\pi}[\sum_{k=0}^{\infty}(\prod_{j=1}^{k}\gamma(S_{t+j}))c(S_{t+k+1})|S_t = s, A_{t:\infty} \sim \pi]$$

where the signal of interest is denoted by c and referred to as the cumulant or pseudo reward. π is the target policy and γ sets the temporal horizon of the prediction and is referred to as the continuation function.

General state-action value function $q_{\pi,\gamma,c}(s,a)$ can be defined similarly with the difference that the expectation is conditioned on $A_t = a$ as well as $S_t = s$. GVFs can also be extended to a case where there are no actions or policies. We use this form of GVFs in our first and second contributions. In this case, the value function is defined as:

$$v_{\gamma,c}(s) \doteq \mathbb{E}[\sum_{k=0}^{\infty}(\prod_{j=1}^{k}\gamma(S_{t+j}))c(S_{t+k+1})|S_t = s] \tag{2.3}$$

Predictions corresponding to general value functions are sometimes called *nexting predictions* (Modayil et al., 2014).

2.4 Summary

In this chapter, we presented the background for reinforcement learning and explained concepts such as online multi-step prediction and control, temporal-difference learning, off-policy learning, GVFs, and nexting predictions.

Chapter 3

Diagnostic Benchmarks for State Construction

This chapter presents the first contribution of this thesis that is the proposal of three diagnostic benchmarks for state construction inspired by animal learning experiments. As mentioned in Chapter 1, state construction is an important challenge for enhancing the generality and applicability of reinforcement learning. Making progress on this challenge requires suitable benchmarks to enable the evaluation of existing solution methods as well as novel ideas.

In a sense, this contribution is the key contribution of the thesis as it provides the foundation for the next contribution which is an empirical study of state construction solution methods on the proposed benchmarks. This contribution was completed by my co-authors and me and is published in the Adaptive Behavior journal (Rafiee et al., 2022). The source code for the benchmarks is available at https://github.com/banafsheh-rafiee/Classical-conditioning-benchmarks-for-state-construction.

3.1 Diagnostic benchmarks

To make progress on the problem of state construction in reinforcement learning, there is a need for suitable benchmarks. The existing benchmarks in reinforcement learning are mostly based on the fully observable setting. The Arcade Learning Environment (ALE) exhibits minor partial observability, but frame-stacking can be used to construct a state that can achieve good perfor-

mance (Bellmare et al., 2013; Machado et al., 2018). OpenAI-Gym (Brockman et al., 2016) and MuJoCo (Todorov et al., 2012) offer a wide variety of tasks inspired by problems in robotics that are partially observable when using only visual inputs. However, the focus is mostly on continuous actions and high-dimensional inputs from joint angles and velocities. The DeepMind Lab contains several 3D simulation problems inspired by experiments in neuroscience (Beattie et al., 2016; Wayne et al., 2018). Researchers have used these problems to benchmark large-scale learning systems; unfortunately, such experiments require several billion steps of interaction and cloud-scale compute (Beattie et al., 2016; Wayne et al., 2018; Pariotto et al., 2020; Fortunato et al., 2019; Espeholt et al., 2018).

Diagnostic benchmarks serve different purposes than large-scale challenge problems. Diagnostic benchmarks are simple issue-oriented problems that illuminate the fundamental limitations of the existing methods. For example, the eight-state Black and White problem highlights the need for tracking in partially observable problems (Sutton et al., 2007), and DeepSea highlights how dithering exploration can be arbitrarily inefficient even in a grid world (Osband et al., 2019). Such diagnostic benchmarks isolate specific algorithmic issues, and progress on these problems represents progress on the specific issues.

Additionally, if a diagnostic benchmark has small compute requirements, then researchers can quickly evaluate new ideas and avoid the additional engineering complexity required to build high-performance, state-of-the-art architectures. Large problems often require complex architectures that can be difficult to analyze, and small implementation details can lead to incorrect conclusions (Engstrom et al., 2019; Tucker et al., 2018). Rigorous statistical analysis, experiment repetition, and ablations can be challenging in large-scale benchmarks because of the excessive computational requirements (see Machado et al. (2018); Henderson et al. (2018); Colas et al. (2018)).

3.2 Animal learning

We used ideas from animal learning experiments to design diagnostic benchmarks for state construction. Animals also have to do state construction. The study of multi-step prediction learning in the face of partial observability dates back to the origins of classical conditioning. Pavlov was perhaps the first to observe that animals form predictive relationships between sensory cues while training dogs to associate the sound of a metronome with the presentation of food (Pavlov, 1927). The animal uses the sound of a metronome (which is never associated with food in nature) to predict when the food will arrive, inducing a hardwired behavioral response.

The ability of animals to learn the predictive relationship between stimuli is critical for survival. These responses could be preparatory like a dog's salivation before food presentation or protective in case of anticipating danger like blinking to protect the eyes.

The study of prediction, timing, and memory in natural systems remains of chief interest to those who wish to replicate it in artificial systems. Researchers from the animal learning field have conducted careful experiments and investigated issues that have received little attention in computer science such as the temporal relationship of events. The animal learning experiments are also of interest because they have a simple setting: there are only a few signals that the animal has to make predictions about. However, they are complicated in the sense that complex computational models are required to solve them.

3.3 The trace conditioning benchmark

Our first diagnostic benchmark is inspired by classical conditioning experiments of the name trace conditioning. In trace conditioning, two stimuli are presented to the animal in sequence as shown in Figure 3.1. First, a conditioned stimulus or CS (the predictive trigger) which usually takes the form of a light or tone, is presented to the animal. Then an unconditioned stimulus

(US), such as a puff of air to the animal's eye, is presented which generates a behavioral response called the unconditioned response (UR)—the rabbit closes its inner eyelid. After enough pairings of the CS and US, the animal produces a conditioned response (e.g., closing the inner eyelid) after the CS—behaving in advance of the US.

This arrangement is interesting because there is a gap, called the *trace interval*, between the offset of the CS and the onset of the US where no stimuli are presented. Empirically we can only reliably measure the strength and timing of the animal's anticipatory behavior: the muscles controlling the inner eyelid. However, the common view is that the animal is making a multi-step prediction of the US triggered by the onset of the CS that grows in strength closer to the onset of the US (Schneiderman 1966; Sutton and Barto 1990, 2018), similar to the conditioned response in Figure 3.1.

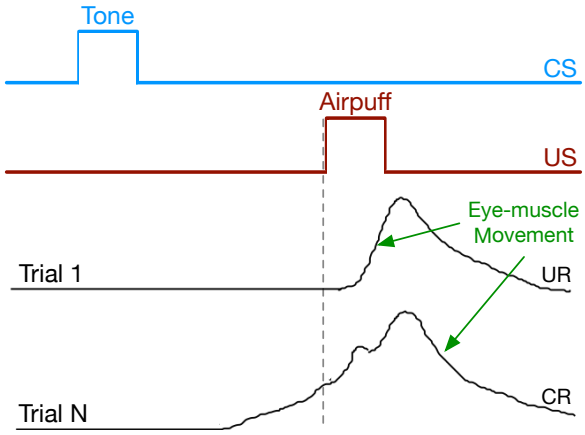

Figure 3.1: Eyeblink conditioning. After many pairings of the tone with the puff of air, the rabbit learns to close its inner eyelid (nictating membrane) before the puff of air is presented.

The mystery for both animal learning and artificial intelligence (AI) is how the agent fills the gap. No stimuli occur during the gap and yet the prediction of the US rises on each time step. There must be some temporal generalization of the stimuli occurring inside the animal.

Additionally, what is the form of the prediction being made, and what algorithm is used to update it? Previous work has suggested that the predictions resemble *discounted returns* used in reinforcement learning (Dickinson 1980;

Wagner 1978), sometimes called nexting predictions (Modayil et al., 2014), which can be learned using temporal difference learning and eligibility traces (i.e., TD(λ)). Indeed, it has been shown that the TD-model of classical conditioning emulate several phenomena observed in animals (Ludvig et al., 2012; Ludvig et al., 2008; Sutton and Barto 1990).

On the question of representation or agent state, the answer is less clear. TD-models can generate predictions consistent with the animal data, but only if the state representation fills the gap between the CS and US in the right way (Ludvig et al. 2012, 2009; Williams et al., 2017). A flag indicating the CS just happened, called the *presence representation*, will not induce predictions that increase over time, and a clock that keeps precise track of time is biologically unrealistic (Ludvig et al, 2012). Hand-designed temporal representations do reproduce the animal data well (Ludvig et al., 2012, 2008, 2009; Williams et al., 2017), but their generality remains unclear. Ideally, the learning system could discover for itself how to represent different stimuli over time in a way that (1) is useful across a variety of prediction tasks, and (2) requires computation and storage independent of the size of the trace interval. Animals do require more training to learn trace conditioning tasks with longer and longer trace intervals, but there is no evidence that the update mechanisms or representations fundamentally change as a function of the trace interval (Howard and Eichenbaum 2013).

We designed a benchmark inspired by trace conditioning experiments and refer to it as the *trace conditioning* benchmark. The trace conditioning benchmark consists of a sequence of trials, each of which consists of a number of discrete time steps. Each trial starts with the onset of the CS $\in \{0, 1\}$ which lasts for 4 time steps and is followed by a long gap, and then the US $\in \{0, 1\}$ which lasts for 2 time steps. The time from the CS onset to the US onset is called the *inter-stimulus interval* or ISI $\in \mathcal{N}$. We designed the benchmark such that the ISI is uniformly distributed between $L - \frac{L}{3}$ and $L + \frac{L}{3}$ where L is the benchmark parameter used to control the benchmark's level of difficulty. More specifically, ISI $\sim U(L - \frac{L}{3}, L + \frac{L}{3})$ where U denotes the discrete uniform distribution. The time from the US onset to the start of the next trial is

called the *inter-trial interval* (ITI) which is uniformly distributed between 80 and 120.

We also include several binary distractor stimuli that do not contain any information about the US. The distractors are drawn from a Poisson distribution with different frequencies and each lasts for 4 time steps. The frequency varies from distractor to distractor. One distractor occurs on average every 10 steps, another every 20 steps, and so on, up to one distractor that occurs every 100 steps on average. Note that they also occur during the ITI.

To formulate the problem of predicting the US, we used multi-step predictions of the US. More specifically, we used nexting predictions as defined in Equation 2.3 in Chapter 2 where the cumulant is the US and γ is set according to the ISI: $\gamma = 1 - \frac{1}{\mathbb{E}(\text{ISI})}$. This allows the time horizon of the return to match the ISI.

Figure 3.2 provides an example trial including the CS, US, and return for a case where ISI is uniformly distributed between 7 and 13. Note that as shown in Figure 3.2, the return reaches its maximum value just before the US onset and steps downward after. This happens because the discounted sum of future USs is maximal just before the US onset. This temporal profile is consistent with previous work on *Nexting* (Modayil et al, 2014) and computational modeling in animal learning (Ludvig et al, 2012).

To understand why this problem could be challenging for a learning system, consider learning to predict using the presence representation. In the presence feature representation, the components are one-to-one with the stimuli. More specifically, the presence feature representation is in form of a vector $\mathbf{x}_t = (x_t^1, x_t^2, \ldots, x_t^n, 1)^\top$ where n is the number of stimuli and the last feature is a bias feature. x_t^i is one if the ith stimulus is present on time step t and zero otherwise. The presence feature corresponding to the CS is active during the CS activation as shown in Figure 3.3. However, during the trace interval, between the offset of the CS and the onset of the US, no feature is active (only the bias feature, which has a small weight associated with it is active) and therefore, the trace interval is not represented by the presence representation. As a result, as shown in Figure 3.2, the presence representation has a close to

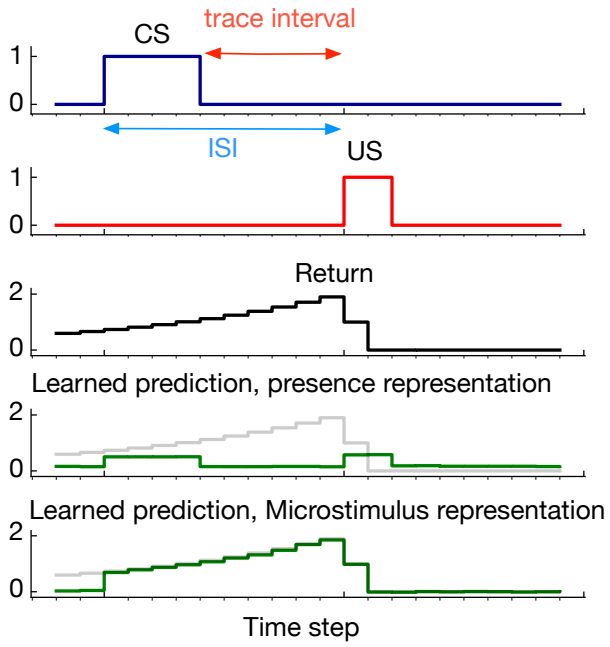

Figure 3.2: An example of learned predictions in the **trace conditioning** benchmark. The *return* is the target of prediction. Rows 4 and 5 show predictions using the presence and microstimulus representations after 200,000 time steps learning. Microstimulus successfully predicted the US, matching the return. The presence representation failed to predict the US as it did not have any active features during the trace interval. The predictions never go to zero like the return because all representations use a bias feature and even after 200,000 steps the predictions continue to update.

zero prediction during the trace interval.

To understand what a good prediction looks like, consider the predictions made by the microstimulus representation (Figure 3.2). The microstimulus representation was initially presented in prior work on computational modeling of classical conditioning (Ludvig et al., 2012, 2008; Hull 1939). Microstimulus forms an exponentially-weighted decaying memory of stimuli, or *stimulating trace*,[1] and then applies a non-linear mapping to produce the representation. Each component of stimulating trace, $\mathbf{y}_t$, corresponds to one component of the stimuli and is set to 1 at the onset of the corresponding stimulus and decays

[1]A stimulating trace of the stimuli is different from the eligibility traces (Sutton and Barto, 2018). Eligibility traces are part of the update mechanism and does not impact the representational capacity. Mozer was the first to investigate stimulating traces as input to neural network representation learning (Mozer 1989).

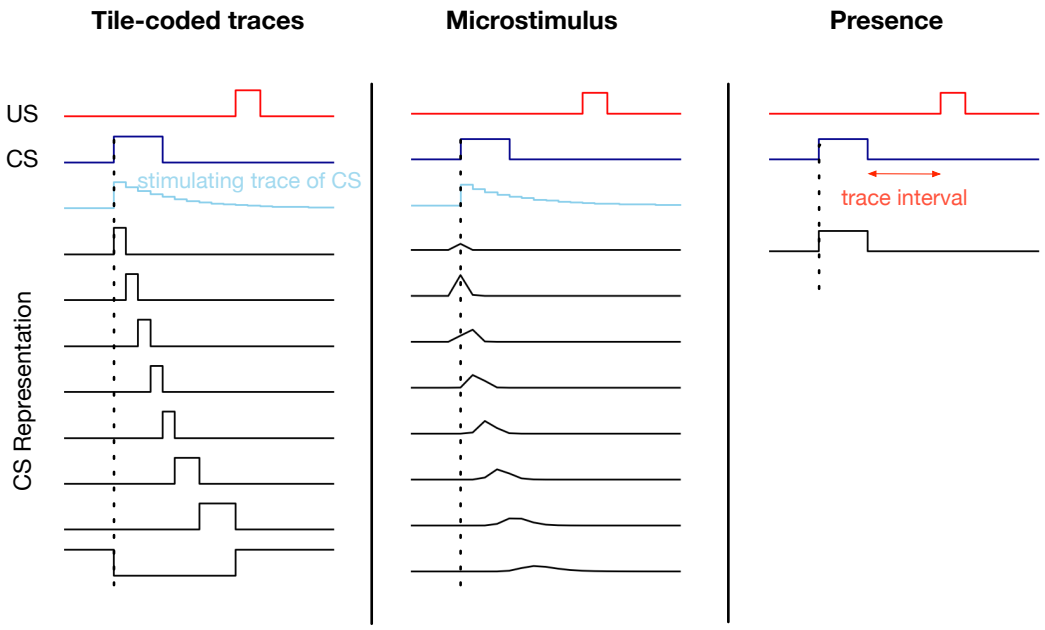

Figure 3.3: The stimulus representation for the tile-coded traces, micro stimulus, and presence representations. The presence representation does not have any active features during the trace interval. The tile-coded traces and microstimulus, however, represent the trace interval through multiple active features. This figure is adapted from Ludvig et al. (2012).

immediately after the stimulus onset following:

$$\mathbf{y_{t+1}} = \tau \mathbf{y_t}$$

where $0 < \tau < 1$ is the decay parameter. Note that the stimulating trace $\mathbf{y_t}$ also includes one component corresponding to the US.

Each component of the stimulating trace, y_t^i, is then coarse coded using k overlapping Gaussian basis functions resulting in k features:

$$f_j(y_t^i) = \frac{1}{\sqrt{2\pi}} exp(-\frac{(y_t^i - \mu_j)^2}{2\sigma^2}) \times y_t^i$$

where μ_j and σ are the mean and width of the basis functions. The mean of the basis functions are spaced linearly: $\mu_j = \frac{j}{k}$ where k is the total number of basis functions.

Another representation similar to microstimulus is tile-coded traces which uses tile coding (Sutton and Barto, 2018) to coarse code the stimulating traces. Tile coding works by applying a number of overlapping grids on the input

called tilings. In the case of tile coding, the quality of the representation depends on both the tile coding parameters and the exponential decay rate of the stimulating traces.

Figure 3.3 shows the CS representation made by the microstimulus and tile-coded traces representations. During the empty gap between the CS offset and the US onset, the microstimulus and tile-coded traces representations have active features constructed from the stimulating trace of the CS. As a result, they successfully associate the CS with the US, matching the return. See the predictions for the microstimulus representation in Figure 3.2.

3.4 The noisy patterning benchmark

Our second benchmark, the *noisy patterning* benchmark, is related to positive/negative patterning and biconditional patterning in psychology (Harris et al., 2008). It considers a situation where non-linear combinations of CSs activate the US. In negative patterning, each CS in isolation activates the US but their combination does not.

Interestingly, these problems correspond to logical operations like XOR, which are famously unsolvable by single-layer neural networks. While neural networks with more than one layer can easily learn patterning problems like XOR, some of the approaches considered in this work, such as microstimulus, fail to solve them. To make the benchmark more challenging we designed the benchmark such that multiple configurations of the CSs activate the US and added distractors and noise.

This benchmark includes n CSs and one US. There are k configurations of the CSs that activate the US. We refer to these configurations as activation patterns. Each activation pattern includes $n/2$ activated CSs and $n/2$ non-activated CSs which are picked randomly at the beginning and kept fixed throughout the experiment. We designed the benchmark such that in half of the trials, one of the activation patterns occurs. Each trial starts with the CSs getting a value of 0 or 1. If the value of the CSs matches an activation pattern, the US becomes 1 in 4 time steps. Therefore, ISI equals 4. In contrast

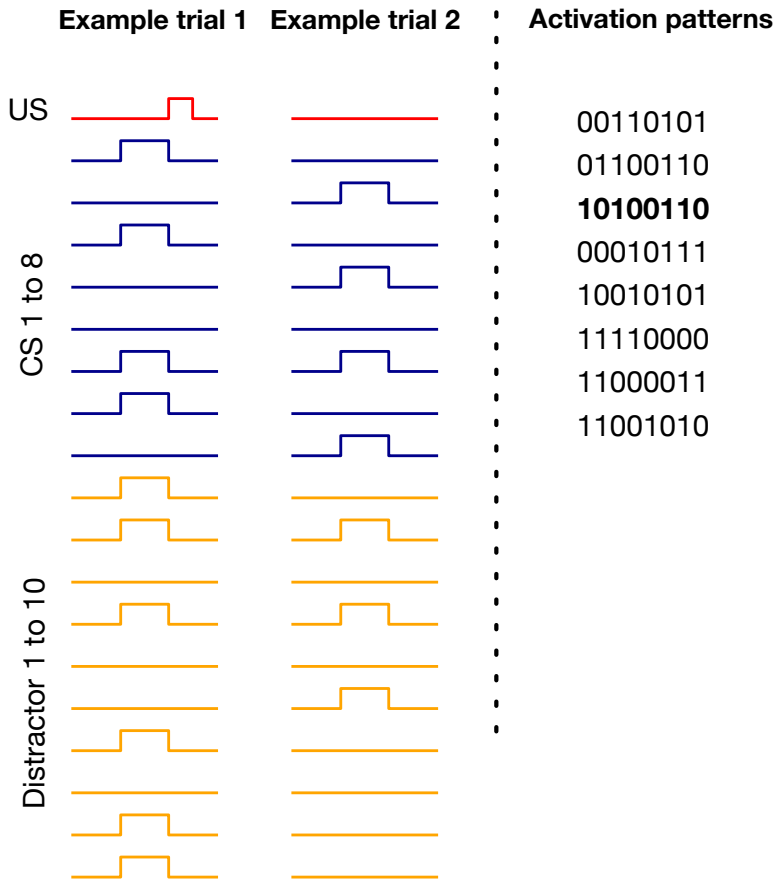

Figure 3.4: Example trials for noisy patterning in the case of 8 CSs, 8 activation patterns, 10 distractors, and 10 percent noise. The 8 activation patterns are shown on the right. `10100110` is one of the activation patterns. In the example trial on the left, the pattern of the CSs matches `10100110` and the US gets activated as a result. In the example trial on the right, however, the pattern of the CSs does not match any of the activation patterns resulting in the US remaining 0.

to the trace conditioning benchmark, the ISI is fixed. Similar to the trace conditioning benchmark, the ITI is uniformly distributed between 80 and 120.

The benchmark also includes m distractors, which occur at the same time as the CSs but do not contribute to the US activation. We also added noise such that in x percent of the trials, an activation pattern occurs but the US remains 0, or a non-activating pattern occurs and the US gets activated. γ is set to $1 - \frac{1}{\text{ISI}} = 0.75$.

Two example trials for a case with 8 CSs, 8 activation patterns, 10 distractors, and 10 percent noise are shown in Figure 3.4. In the example on the left,

the pattern of the CSs matches one of the 8 activation patterns. Therefore, the US gets activated. In the example on the right, however, the pattern of the CSs does not match any of the activation patterns. As a result, the US remains 0.

Just as we can control the difficulty level of the trace conditioning benchmark by changing, for example, the ISI, we can control the difficulty level of the noisy patterning benchmark by changing the key problem parameters — the number of CSs, the number of activation patterns, the number of distractors, and the level of noise.

3.5 The trace patterning benchmark

Our third benchmark is a combination of the first two and we refer to it as the *trace patterning* benchmark. In this benchmark, we put together the challenge of bridging the temporal gap between the stimuli, as posed by the trace conditioning benchmark, and the challenge of recognizing important patterns and disregarding distractors, as formulated in the noisy patterning benchmark. For a learner to do well on this benchmark, it has to both fill the trace interval and construct non-linear representations of the CSs.

In the trace patterning benchmark, the level of difficulty can be controlled by the ISI as well as the number of CSs, the number of activation patterns, the number of distractors, and the level of noise. In our empirical study, we use the trace patterning benchmark examining the effect of the ISI while keeping the rest of the benchmark's parameters fixed: 8 CSs, 8 activation patterns, 10 distractors, and 10% noise.

3.6 Conclusion

To effectively address the challenge of state construction, suitable benchmarks are required to evaluate the existing solution methods as well as novel ideas. In this chapter, we presented three diagnostic benchmarks for state construction inspired by animal learning. The proposed benchmarks are appealing for several reasons:

1. They are lightweight, facilitating experiment repetition and statistical analysis.

2. They are issue-oriented allowing us to study the fundamental issues within state construction in isolation.

3. They incorporate knobs to control the benchmark's level of difficulty.

Other researchers have also felt the appeal of the proposed benchmarks and used them in their experiments (Elelimy 2023; Javed et al., 2023; Samani and Sutton 2021).

In Chapter 5, we will use the proposed benchmarks to conduct an empirical study of prominent solution methods for state construction.

Chapter 4

Methods for State Construction

One of the goals of this dissertation is to answer the question of how well existing solution methods to state construction perform. To achieve this goal, this dissertation contributes diagnostic benchmarks for state construction presented in Chapter 3 and presents an empirical study of prominent solution methods on the proposed benchmarks in Chapter 5. This chapter surveys existing solution methods for state construction and discusses which of the methods we include in and leave out of the empirical study and why.

As mentioned in Chapter 1, state construction is defined as the construction of the agent state directly from the information provided by the environment. An approach to constructing the state is to use the state-update function which is computationally appealing due to its recursive form. The challenge of state construction is an old challenge dating back to the original work on Partially Observable Markov Decision Processes in 1960. Since then there has been a substantial amount of work each with its own development of the state update function. We will discuss some of these approaches in the next sections.

4.1 Recurrent neural networks

A common approach to state construction is to represent the agent state using the hidden state of recurrent neural networks (RNN) and update the agent state in a recurrent manner (Elman, 1990). In this case, the state-update function would be the RNN.

The internal weights of RNNs could be kept fixed or learned through gradi-

ent descent. RNNs with fixed randomly initialized internal weights are called echo state networks or ESNs (Jaeger 2001). In this case, the state-update function is a fixed function of the data stream.

The internal weights of RNNs can also be learned through gradient descent. To compute the gradient with respect to the weights of RNNs, backpropagation through time or BPTT is commonly used (Robinson and Fallside, 1987; (Werbos, 1988). In BPTT, backpropagation is run on the RNN unrolled in time. BPTT requires the network activations to be stored from the beginning of time and is expensive in terms of computation and memory. To address this issue, truncated backpropagation through time (T-BPTT) was introduced in which the computational graph is truncated T steps back in time (Williams and Peng 1990). The truncation length presents a trade-off. If the truncation window is short, long temporal dependencies cannot be learned as most recurrent neural networks cannot learn temporal relationships longer than T (Williams and Peng 1990). If the truncation window is long, the computation and memory costs will be high as they grow with T. Moreover, a recurrent network with a long truncation window is prone to the problem of vanishing/exploding gradients similar to that of a deep network. More complex recurrent architectures have been introduced to address the problem of vanishing/exploding gradients such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) which incorporate gating techniques.

An alternative to T-BPTT is real-time recurrent-learning or RTRL which itself computes an approximation of the true gradient. In RTRL, the gradient is computed recursively without the need for unrolling and storing the previous network activations. For a fully connected network, RTRL requires quartic computation in the number of hidden states per step which makes online implementation with even modestly sized networks challenging (Williams and Zipser 1989). Approximations of RTRL such as Unbiased Online Recurrent Optimization (UORO) (Tallec and Ollivier 2018), synthetic gradient methods (Jaderberg et al., 2017), and SnAp (Menick et al. 2020) approximate the gradient back in time. These approximate methods are either extremely high variance or significantly biased.

We included recurrent neural networks trained with gradient descent in our empirical study. Neural networks trained with gradient descent are perhaps the most successful approach to state construction. These methods are appealing for several reasons. They have made the construction of the state automatic and tailored to the task at hand. They also scale relatively well with computation. Finally, they do not need to have access to the underlying transition model of the environment. These characteristics make them suitable for the benchmarks proposed in the previous chapter.

While neural networks trained with gradient descent have hints to the final solution, they are not fully satisfactory. They are not suitable for continual learning: the ability of the network to learn deteriorates over time (Dohare, Mahmood, and Sutton 2021). Moreover, as discussed earlier, the existing methods for computing the gradient in recurrent neural networks are expensive in terms of computation and memory. The limitations of this class of methods motivate the need for additional analysis, one instance of which is presented in this thesis.

4.2 Transformers

An alternative to RNNs is the transformer model (Vaswani et al., 2017) which consist of attention mechanisms combined with a feed-forward neural network. Attention mechanisms identify which parts of a sequence are more important and produce a representation of each element in the sequence based on its degree of importance.

Transformers are commonly viewed as being more effective than RNNs due to their ability to handle long temporal associations and their scalability with computational resources which allows the use of large models. However, their computational requirements increase quadratically with the span of temporal dependencies.

Note that transformers are not an instantiation of the state-update function because they do not update the state recursively from the previous state. Instead, to handle partial observability, they stack sequences of past observa-

tions. The length of the sequence fed into transformers is referred to as the context length.

The best way to use transformers for the multistep prediction and control problems is still evolving (Parisotto et al., 2020; Parisotto and Salakhutdinov 2021; Loynd et al., 2020; Chen et al., 2021; Janner et al., 2021). Chen et al. (2021) and Janner et al. (2021) use transformers in the offline reinforcement learning setting where transformers are used to imitate offline data. Parisotto et al. (2020) and Parisotto and Salakhutdinov (2021) stack long sequences of past observations in order to learn long temporal dependencies.

We included transformers trained with backpropagation in our empirical study as they have recently achieved remarkable results in sequence modeling tasks.

4.3 Partially Observable Markov Decision Processes

The problem of state construction dates back to the introduction of Partially Observable Markov Decision Processes or POMDPs. POMDPs are the extension of Markov Decision Processes to the case of partial observability. In POMDPs, the environment state is referred to as the *latent state* and is never observed by the agent. Instead, the agent maintains a belief about the possible state of the environment. More specifically, the agent state is defined as a distribution over the environment state and referred to as the *belief state*. The belief state is incrementally updated using Bayes' rule assuming having access to the transition probability of the environment state (Monahan,1982).

POMDPs have had several significant applications. However, their applicability has remained limited for several reasons. First, the belief state is grounded in the environment state which is not directly observable by the agent. Second, the agent has to know the transition model for the environment state which in many cases is not plausible. Third, the belief state update is computationally expensive as it requires computation quadratic in the number of environment states. Finally, exact POMDP planning is intractable even for

problems with modest size.

We did not include POMDPs in our empirical study as they assume having access to the dynamics of the underlying latent state which does not apply to the animal learning benchmarks proposed in Chapter 3.

4.4 Predictive state representations

In predictive state representations or PSRs, the state is represented as a vector of predictions about the stream of observations and actions. Similar to POMDPs, the state is updated using Bayes' rule (Littman, Sutton, and Singh, 2001). Unlike the belief state in POMDPs, however, in PSRs the state is grounded in the stream of observations and actions which is directly observable by the agent.

In PSRs, a *test* is defined as a sequence of observations and actions. The outcome of a test is defined as the probability of the test happening given the data stream up to the current time step. If the outcome of all possible tests is known by the agent, the agent knows everything there is to know about the system. The set of tests sufficient for knowing the outcome of all tests is referred to as *core tests*. This set of tests is linearly independent and can be linearly combined to produce the outcome of any test. The state is defined as the probability of the core tests given the history.

There is a large body of work related to the idea of predictive state representations. Observable operator models (OOMs), a predecessor of PSRs, introduced the idea of predictions as state for the first time (Jaeger, 2000); PSRs can be thought of as the extension of OOMs to the control setting. Rudary and Singh (2003) extended PSRs to the nonlinear case. Transformed predictive state representations (TPSRs) use principle component analysis to discover a minimal set of core tests (Rosencrantz et al., 2004). Temporal-difference networks extended PSRs to include compositional and temporally abstract predictions (Sutton, Rafols, and Koop, 2005).

We excluded PSRs from our empirical study as learning the state in PSRs requires complicated training algorithms. Moreover, using PSRs requires the

discovery of the core tests which we do not readily have access to in the animal learning benchmarks.

4.5 Generate-and-test

The problem of state construction can be viewed as one of feature finding such that the underlying system performs well where *features* are elements of the state representation, such as hidden units in neural networks. Many of the approaches to state construction discussed so far perform feature finding from the stream of data. Perhaps the most successful methods within this group are neural networks trained with gradient descent. We can think of large neural networks trained with gradient descent as performing a massive parallel search in feature space (Frankle and Carbin, 2018).

Neural networks trained with gradient descent are specifically developed for settings where learning happens once on a large dataset as opposed to settings where there is an unending stream of data from which the agent has to continually learn (Dohare et al., 2023). More specifically, they greatly depend on the randomness in the weight initialization to find good features. Once exposed to new data, neural networks trained with gradient descent tend to forget what they have learned previously (McCloskey and Cohen, 1989; French 1999). More importantly, they tend to lose their ability to learn from new data (Dohare et al., 2023).

A promising approach to address the issues of forgetting and loss of plasticity is to continually inject randomness in feature search as done in a class of methods known as generate-and-test. Generate-and-test methods search for good features by continually generating new features, testing them, and replacing the useless ones with newly generated features (Mahmood and Sutton 2013; Samani and Sutton, 2021; Dohare et al., 2023; Shah 2023; Elsayed and Mahmood, 2023). This class of methods is specifically suitable for continual learning as they do not suffer from the problem of loss of plasticity as in Neural networks trained with gradient descent.

The idea of generate-and-test is orthogonal to gradient descent and can be

combined with it. In fact, the combination of generate-and-test with gradient descent has produced promising results in classification and regression tasks (Dohare et al., 2023).

We did not include generate-and-test methods in our empirical study. While the generate-and-test approach exhibits a lot of promise, it is not yet fully developed for partial observability. The third contribution of this thesis, however, is inspired by the idea of generate-and-test.

4.6 Neural networks with external memory

There is a class of neural networks that incorporate an external memory component. One example of neural networks with external memory is Neural Turing Mechin and its extensions (Graves et al., 2014; Graves et al., 2016; Wayne et al., 2018). This approach includes methods that learn memory operations resembling memory operations of a conventional computer through gradient-descent and from data.

Other external memory approaches include those equipped with an episodic memory system. These methods aim to rapidly learn from highly rewarding situations using the data stored in the episodic memory (Blundell et al., 2016; Pritzel et al., 2017). These methods search for past successful experiences stored in the replay buffer using nearest-neighbours non-parametric model (Blundell et al., 2016) or attention mechanisms (Pritzel et al., 2017).

We excluded neural networks with external memory from our empirical study. Although these approaches have shown promise in certain problems, we decided to focus on more widely used methods with more straightforward implementations such as recurrent neural networks and transformers.

4.7 Summary

This chapter discussed solution methods for state construction including the methods that we included and left out of the empirical study. Chapter 5 will present the result of a systematic study of some of these methods highlighting their strengths and weaknesses.

Chapter 5

Empirical Study of Prominent Recurrent Learning Methods on the Diagnostic Benchmarks

As mentioned in the introduction, this thesis aims to answer the question of how well current state construction solution methods perform. To address this question, this thesis provides diagnostic benchmarks and an empirical study of existing state construction methods on the proposed benchmarks. The diagnostic benchmarks were presented in Chapter 3. This chapter presents the empirical study of state construction methods on the diagnostic benchmarks with a focus on prominent recurrent learning methods.

This contribution has the most immediate contribution to the goal of gaining a better understanding of the problem of state construction and the existing solution methods because it provides a comprehensive set of results on the existing solution methods through a systematic study. The major results from this study are presented in the Adaptive Behavior journal in 2022.

In this chapter, we systematically study each method's performance as we vary the benchmarks' difficulty using key benchmark parameters. Also, as a minor contribution, we introduce a simple input augmentation scheme based on memory traces which improves the studied recurrent learning methods. This empirical study is a step toward identifying the limitations of prominent recurrent learning systems and designing new methods.

5.1 Methods

As mentioned in Chapter 4, there is a large body of work on state construction. Some of the approaches to state construction were discussed in Chapter 4. For the empirical study, we focused on neural networks trained with backpropagation. As discussed in Chapter 4, neural networks trained with backpropagation are appealing because they have made the construction of the state automatic and scale well as more computational resources are provided.

The first group of methods that we considered includes recurrent learning architectures combined with T-BPTT or RTRL for computing the gradient of the value function with respect to the network's weights. Within this group, the first recurrent architecture that we considered is Vanilla-RNN chosen for its simplicity. Additionally, we included LSTM and GRU architectures due to their ability to address the problem of vanishing/exploding gradients often encountered when using Vanilla-RNNs.

We followed standard practice in implementing recurrent neural networks trained with T-BPTT and RTRL. For T-BPTT with truncation length T, at each time step t, we unroll the RNN for T steps. We copy the value of the hidden state of the RNN at time $t-T-1$ from the forward pass at the previous time step to form the initial hidden state, $\mathbf{x}_{t-T-1}$, in the current forward pass. Then we pass the observation sequence $\mathbf{o}_{t-T}, ..., \mathbf{o}_t$ to the network one by one. After passing each observation, $\mathbf{o}_k$ where $t - T \leq k \leq t$, we compute the corresponding hidden state, $\mathbf{x}_k$, using the preceding hidden state, $\mathbf{x}_{k-1}$, and the observation $\mathbf{o}_k$, and compute the value function, $\hat{v}_k$. This results in a sequence of hidden states $\mathbf{x_{t-T}}, ..., \mathbf{x_t}$, and a sequence of value predictions $\hat{v}_{t-T}, ..., \hat{v}_t$. After computing the value predictions $\hat{v}_{t-T}, ..., \hat{v}_t$, we use them as a mini-batch to update the parameters of the network using backpropagation.

For RTRL, on the other hand, we do not do unrolling. Instead, we update the parameters throughout the training sequence on every time step, while carrying forward a stale Jacobian that tracks sensitivity to the old parameters (See Menick et al, 2022).

We also experimented with a group of methods with fixed representations

as baselines including the microstimulus and tile-coded traces representations (discussed in Chapter 3) and echo state networks (Jaeger 2001) in which the representation is constructed using a large recurrent neural network with fixed randomly initialized internal weights as discussed in Chapter 4. We believe that microstimulus and tile-coded traces are useful baselines to include due to their simplicity. Moreover, the effectiveness of microstimulus in has been shown in prior computational modeling work of classical conditioning (Ludvig et al., 2012, 2008).

The last group of methods that we explored is the combination of attention mechanisms with feed-forward neural networks, known as transformers. Transformer models have gained remarkable results in sequence modeling tasks. It is worthwhile to examine their performance on our proposed benchmarks. We specifically used a GPT model (Radford et al., 2018). The details of the GPT model and the results can be found in Section 5.6

For each of these representations, we used semi-gradient TD(λ) and ADAM optimizer (Kingma and Ba 2014). To evaluate the performance, we computed the Mean Squared Return Error (MSRE):

$$\text{MSRE} = \sum_{t}^{T} (\hat{v}(S_t, \mathbf{w_t}) - G_t)^2$$

where T is the total number of time steps.

For each diagnostic benchmark, we studied the performance of each method as we varied the key problem parameters. For the trace conditioning and trace patterning benchmarks, we studied the effect of the ISI. For the noisy patterning benchmark, we studied the effect of the number of CSs, the number of activation patterns, the number of distractors, and the level of noise

5.2 The trace conditioning benchmark

We studied the effect of the ISI on the performance of the baseline representation methods considering three cases: 1) short: ISI uniformly distributed between 7 and 13, 2) medium: ISI uniformly distributed between 14 and 26,

Problem	Representation Method	Number of Tiles/RBFs	Hidden Layer Size	Truncation Length	Step-size
Trace Conditioning and Trace Patterning	Presence	-	-	-	3e-6, 1e-5, 3e-5,1e-4, 3e-4, 1e-3
	Microstimulus	4, 8, 16, 32	-	-	
	Tile-coded-traces	2, 4, 8, 16	-	-	
	Vanilla-RNN	-	10, 20, 40	5, 10, 20, 40	
	GRU	-	10, 20, 40	5, 10, 20, 40	
	LSTM	-	10, 20, 40	5, 10, 20, 40	
	ESN	-	-	-	
Noisy Patterning	Presence	-	-	-	
	Vanilla-RNN	-	10, 20, 40	5	
	GRU	-	10, 20, 40	5	
	LSTM	-	10, 20, 40	5	
	ESN	-	-	-	

Table 5.1: Parameter sweeps for the three benchmarks.

and 3) long: ISI uniformly distributed between 20 and 40, with expected ISI equal to $10, 20, 30$ for the 3 settings respectively.

We swept over the parameters of each representation method. See Table 5.1. The parameter sweeps included the step-size for all the methods, the number of Tile/RBFs for tile-coded-traces/microstimulus, the hidden layer size for the RNNs and echo state network, and the spectral radius, input scaling, and internal connections density for the echo state network. For tile-coded traces, we used 2 tilings and for microstimulus, we set the standard deviation of the RBFs to 0.8. For RNNs trained with T-BPTT, we swept over T-BPTT truncation length. For all RNNs, one hidden layer was used.

We ran each method with each of its parameter settings for 5 runs and 2 million time steps. We then computed MSRE averaged over the 5 runs and selected the parameter setting that resulted in the lowest level of MSRE. After optimizing the parameters, we ran each method with its best parameter setting for 30 runs and averaged the result. We calculated standard errors for each method to measure how far the sample means are from the true population means. We then plotted the MSRE averaged over 30 runs and standard error bars with non overlapping standard error bars for two methods suggesting significant difference in their performance.

Figure 5.1 shows MSRE for fixed representations for short, medium, and long ISI. The y-axis is MSRE averaged over 30 runs. The level of error for the presence representation is shown with a dotted grey line for comparison.

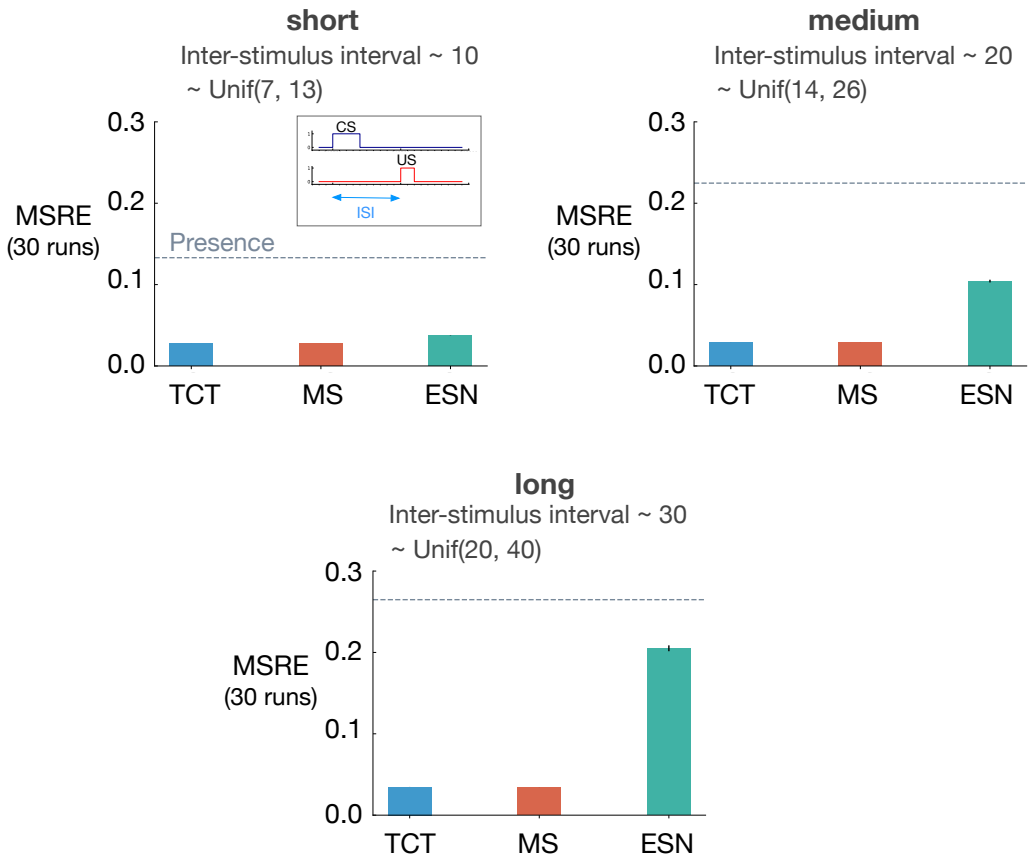

Figure 5.1: The interaction between ISI and truncation level in the **trace conditioning** benchmark for fixed representations: tile-coded traces (TCT), microstimulus (MS), and echo state network (ESN). Each subplot corresponds to one setting of short, medium, and long ISI. A mini picture of the CS and US timings is included in the upper left subplot. The y-axis is the MSRE. Lower is better. The results are calculated over 2 million steps and averaged over 30 runs. (Standard error bars are plotted but in some cases are not visible due to being small). The error level for the presence representation is plotted in each subplot as a dotted line for comparison. In the short setting, all methods performed well. Microstimulus and tile-coded traces performed well across all settings. The performance of the echo state network, however, deteriorated as ISI got larger.

The expert designed fixed representations of microstimulus and tile-coded traces performed well across all ISI settings; however, the echo state network failed to capture longer temporal dependencies. In the short setting, all fixed representations performed well. As ISI got larger, the echo state network performed worse and approached the level of error of the presence representation. This is likely due to the fact that echo state networks trade-off prediction

accuracy for computation.

Figure 5.2 shows MSRE for representations learned by T-BPTT and RTRL for short, medium, and long ISI. In each subplot, multiple bars are shown for each of Vanilla RNN, LSTM, and GRU architectures. For each architecture, the four left bars correspond to T-BPTT with $T = 5$, $T = 10$, $T = 20$, and $T = 40$. The right bar corresponds to the result for RTRL.

In the short setting, the representations learned by both T-BPTT and RTRL performed well for all architectures, reaching a much lower level of error compared to the presence representation.

RNNs trained with T-BPTT were sensitive to the length of the truncation window, and the sensitivity became more pronounced as ISI got larger (Figure 5.2). To better understand this, let us contrast the performance of T-BPTT with that of the RTRL variants, which are roughly equivalent to T-BPTT for $T = \infty$ (since when $T = \infty$, T-BPTT computes the gradient all the way back in time, resulting in a gradient roughly the same as the one computed by RTRL). In the medium setting, the T-BPTT variants for LSTMs and GRUs performed similarly to the RTRL counterparts only when the truncation window was greater than or equal to 20 – the expected ISI (Figure 5.2, top right subplot). This effect was even stronger in the long setting (Figure 5.2, bottom subplot). This result is one example of the efficacy of the trace conditioning benchmark as a diagnostic benchmark — it clearly isolates the trade-off introduced by the T-BPTT algorithm.

There was a significant drop in the performance of Vanilla RNNs as we increased the expected ISI and a larger truncation window did not help improve performance much. This is likely due to the vanishing gradient problem (Hochreiter et al., 2001). Vanilla RNN trained with RTRL also failed to capture longer dependencies. This is in contrast to the LSTM and GRU variants trained with RTRL.

Our results suggest that further algorithmic improvements are required for solving the trace conditioning problem. While the expert designed fixed representations perform robustly across all ISI settings, they do not automatically discover useful features and thus are not scalable. RTRL also performs well

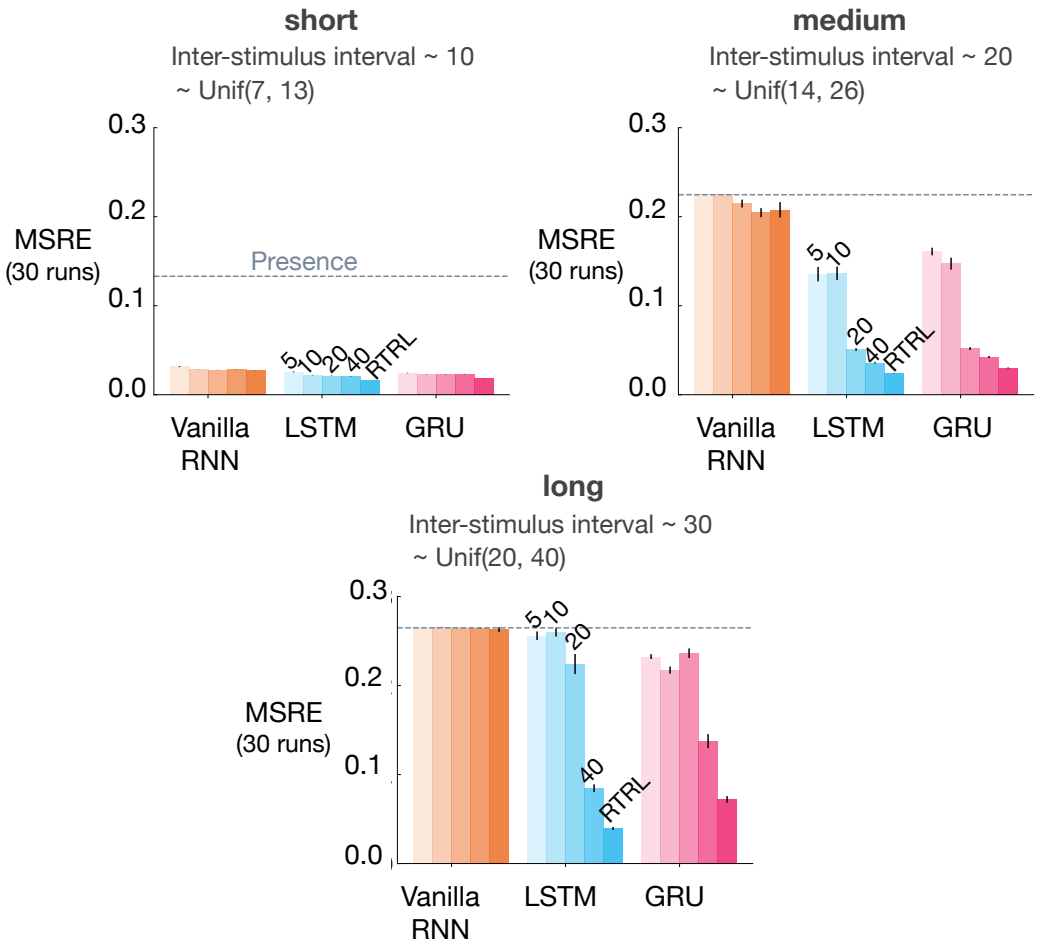

Figure 5.2: The interaction between ISI and truncation level in the **trace conditioning** benchmark for representations learned by T-BPTT and RTRL. Each subplot corresponds to one setting of ISI. In each subplot, multiple bars are plotted for Vanilla RNN, LSTM, and GRU. For each architecture, the left four bars correspond to T-BPTT with different truncation levels and the right bar corresponds to RTRL. The y-axis is the MSRE with lower better. The results are calculated over 2 million steps and averaged over 30 runs. Standard error bars are included in the plot. With short ISI all methods performed well and the T-BPTT based methods worked with all T's. In the medium setting, basic RNNs performed poorly, and LSTMs and GRUs required truncation at or greater than expected ISI (20) to perform well. In the long setting, none of the T-BPTT based methods performed well, even with T greater than expected ISI. Across all three problem settings, RTRL-based LSTMs achieved a low level of error.

in all cases; however, it is not computationally feasible. Finally, T-BPTT's performance is highly sensitive to the truncation parameter, requiring much more computation for learning longer temporal dependencies. Later we will

discuss a simple algorithm that we tried to improve performance.

5.3 The noisy patterning benchmark

Just as we can control the difficulty level of the trace conditioning benchmark by changing, for example, the ISI, we can also control the difficulty level of the noisy patterning benchmark by changing the key problem parameters — the number of CSs, the number of activation patterns, the number of distractors, and the level of noise. Using this flexibility, we experimented with the noisy patterning benchmark in two ways. First, we evaluated echo state network and several T-BPTT variants with truncation length 5 on three different levels of difficulty that we refer to as easy, medium, and hard.

We did not experiment with RTRL because with small ISI (= 4), T-BPTT with $T = 5$ performs as well as the idealized RTRL baseline. We also did not experiment with tile-coded traces and microstimulus because they independently represent each input and cannot predict patterns of CSs as they are combined with linear function approximation.

There was a consistent drop in performance, across all methods, as the level of difficulty was increased (Figure 5.3). Echo state network performed worse than all three recurrent variants trained with T-BPTT in all three configurations of the problem. This is likely due to the fact that echo state network's representation, which is randomly determined and fixed at the beginning of learning, is not suitable for capturing the activation patterns.

Example LSTM prediction profile plots for the noisy patterning benchmark are provided in Figure 5.4 for the medium and hard levels of difficulty. We are only showing 2 of the CSs and 2 of the distractors as examples. In both examples, an activation pattern occurred and the US got activated (i.e., the US activation was not due to noise). In the medium setting, LSTM successfully predicted the US, matching the return after the onset of the CS. However, in the hard setting, there was a mismatch between LSTM's prediction and the return.

To further highlight the configurability of the noisy patterning benchmark,

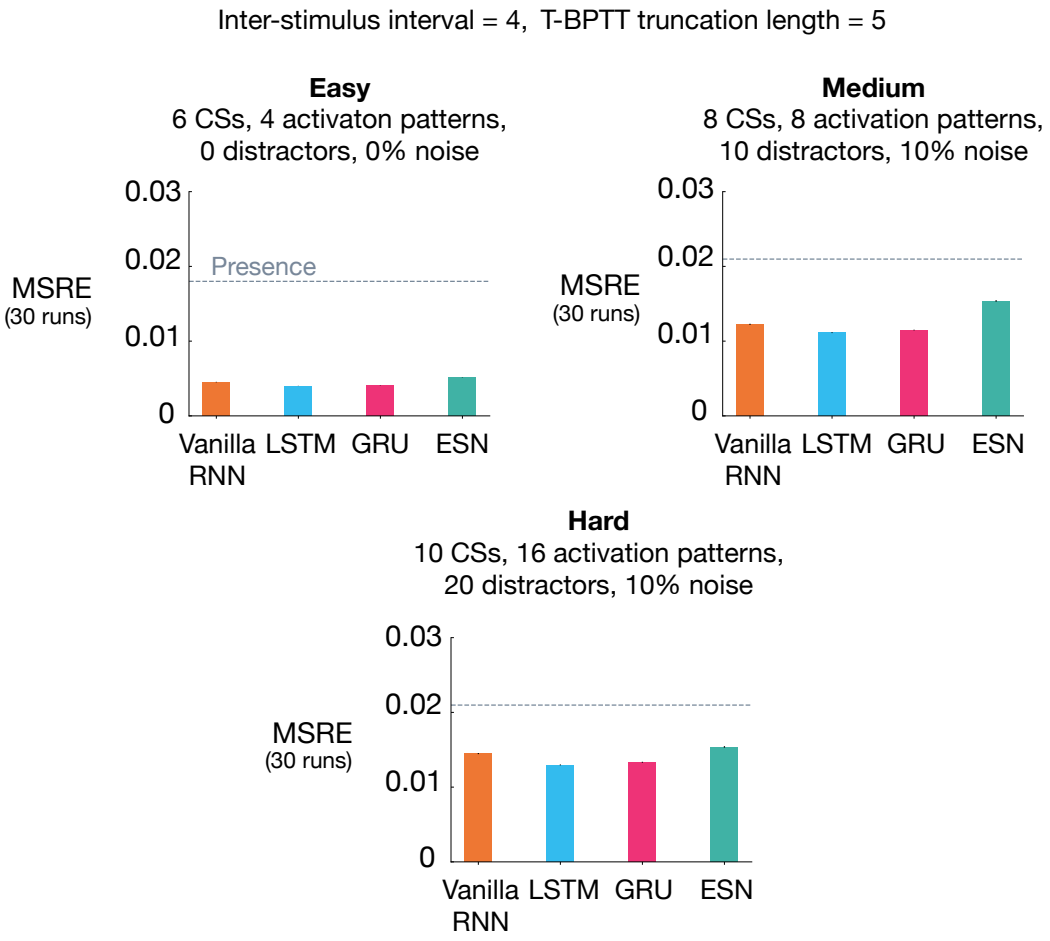

Figure 5.3: The **noisy patterning** benchmark with varying difficulty levels. The 4 bar plots show the MSRE of Vanilla-RNN, GRU, and LSTM trained with T-BPTT as well as the MSRE of echo state network for three different configurations of the problem: easy, medium, and hard. The results are for 2 million steps of training and averaged over 30 runs. The standard error bars are included. There was a consistent drop in performance, across all methods, from the easy setting to the hard one.

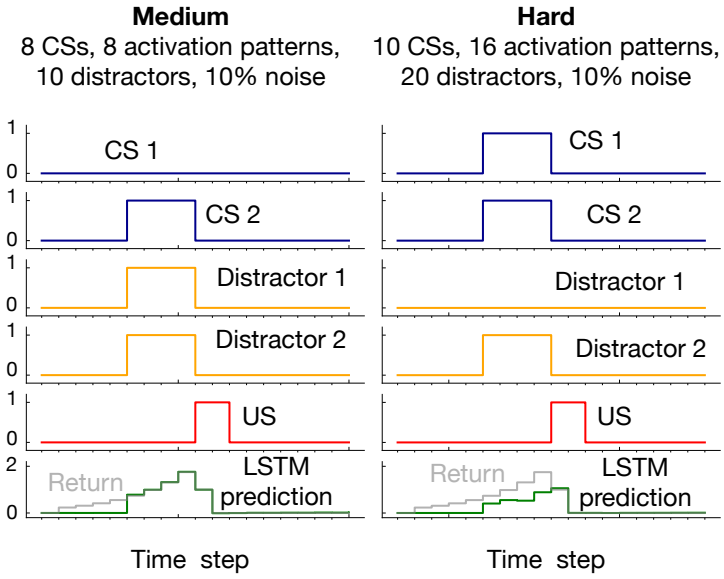

Figure 5.4: Example prediction profile plots for the noisy patterning benchmark in the medium setting and hard setting. Unlike Figure 3.4 where all the CSs and distractors were shown, in this figure only two of the CSs and distractors are shown as examples. In both cases, an activation pattern occurred as a result of which the US got activated. In the medium setting, LSTM prediction matched the return. In the hard setting, however, LSTM did not predict the US accurately.

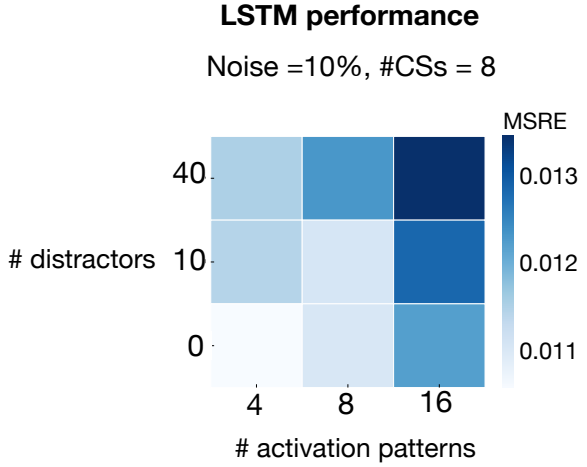

Figure 5.5: The performance of LSTM trained by T-BPTT in the **noisy patterning** benchmark. The performance of LSTM degraded as the number of distractors and activation patterns increased.

we evaluated the T-BPTT variant of LSTM across two dimensions: the number of activation patterns and the number of distractors. The results, presented as a heatmap of MSRE in Figure 5.5, show that the performance deteriorated as we made the problem more difficult across either dimension.

Taken together, these results demonstrate that the noisy patterning benchmark can be useful for systematically studying the scaling properties of the algorithms in isolation from the temporal dimension, by simply increasing the number of signals from half a dozen to tens of thousands.

5.4 The trace patterning benchmark

Similar to our experiments on the trace conditioning benchmark, we evaluated the baseline methods as we increased the ISI while keeping the rest of the problem parameters fixed (8 CSs, 8 activation patterns, 10 distractors, and 10% noise). The results for fixed representations and representations learned by T-BPTT and RTRL are provided in Figure 5.6 and 5.7 respectively.

The fixed representations performed poorly in all cases of short, medium, and long ISI, and their performance got worse as ISI got larger (Figure 5.6). The expert designed fixed representations of microstimulus and tile-coded traces independently represent each input (and not their combinations) and thus cannot learn accurate predictions; contextualizing the failure of the echo state network in this problem.

The T-BPTT algorithms showed sensitivity to the length of the truncation window (Figure 5.7). This is consistent with the findings from the trace conditioning experiments. One key difference, however, is that longer truncation parameters for the LSTM and GRU variants did not help as much as in the trace conditioning benchmark. Moreover, in contrast to the trace conditioning benchmark, the performance of the idealized RTRL baselines for the LSTM and GRU variants got worse considerably as we increased the ISI.

Example prediction plots for LSTM trained with T-BPTT are shown in Figure 5.8 in the case of expected ISI of 10 and 30. In both cases, a truncation length of 40 was used. While LSTM prediction profiles resemble the return in

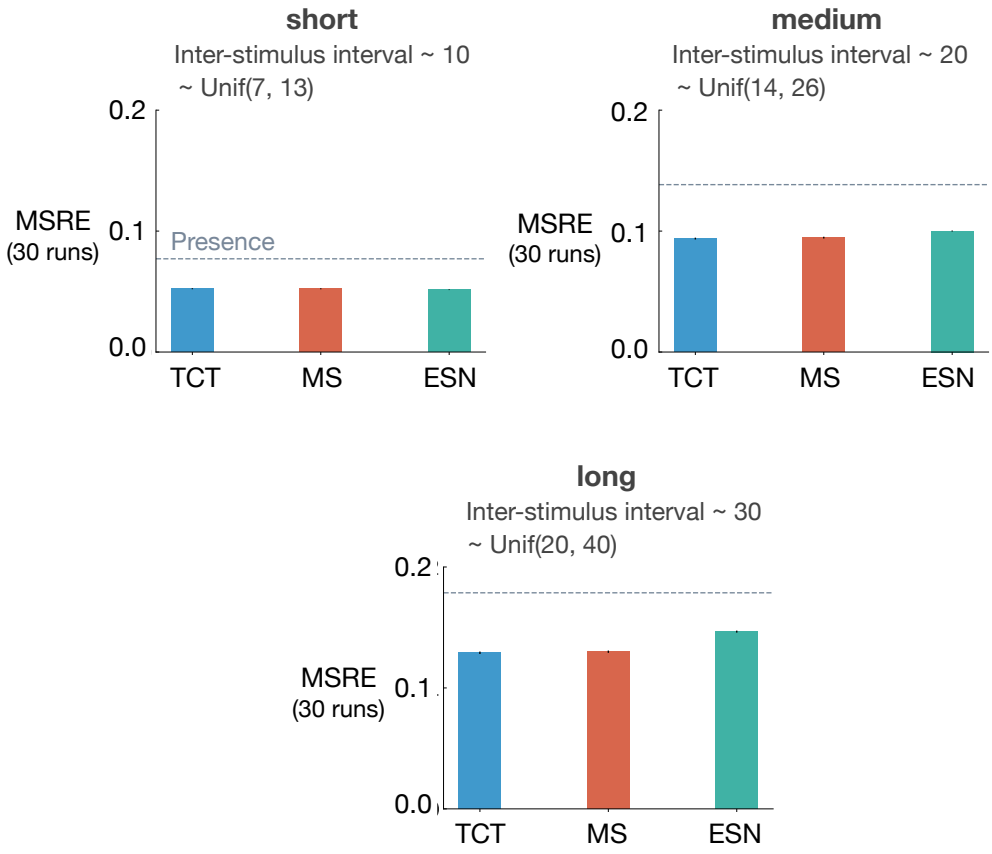

Figure 5.6: The impact of truncation level in the **trace patterning** benchmark for fixed representations. We used the exact same scheme as Figure 5.1 to visualize the performance in the trace patterning benchmark. Each plot corresponds to one setting of short, medium, and long ISI. Each bar reports the MSRE averaged over 30 runs. All methods were trained for 5 million steps. All fixed representations performed poorly. Tile-coded traces and microstimulus independently represent each input (not combinations) and thus cannot learn accurate predictions.

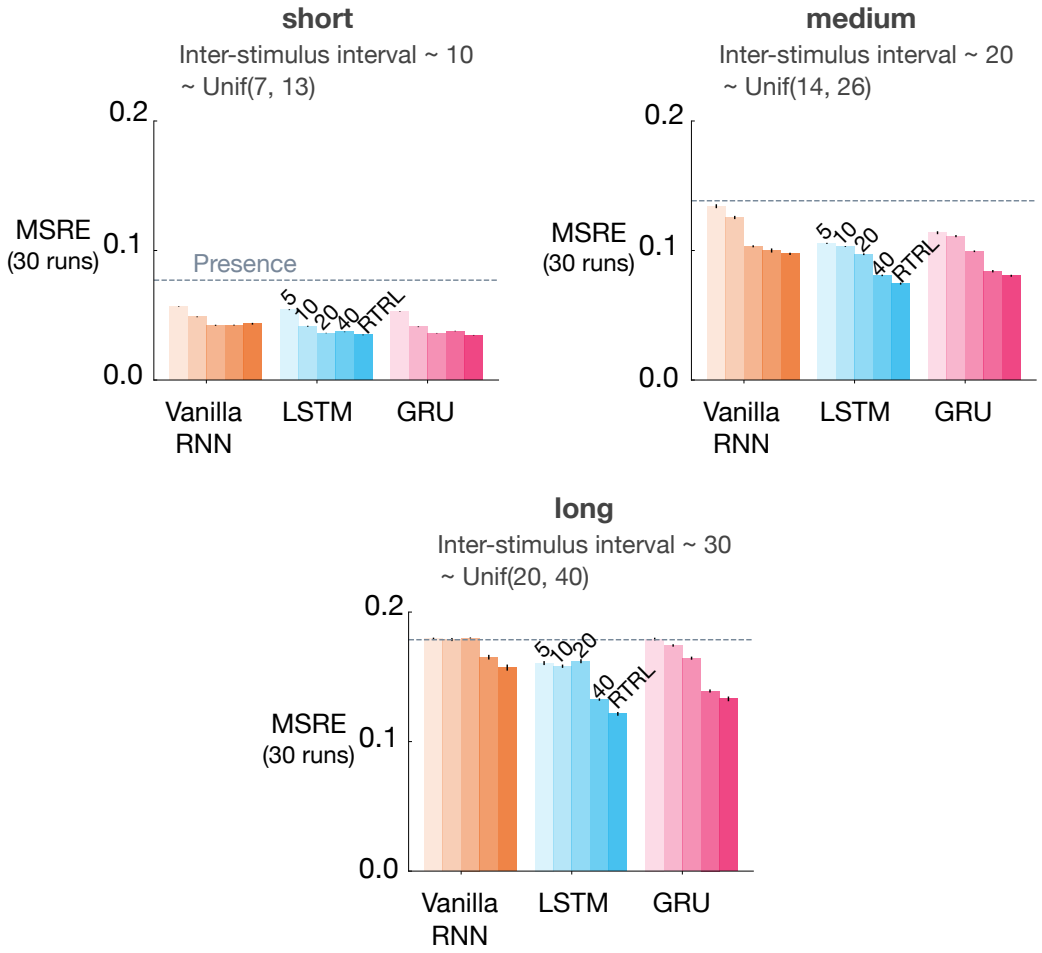

Figure 5.7: The impact of truncation level in the **trace patterning** benchmark for representations learned by T-BPTT and RTRL. Each subplot corresponds to one setting of short, medium, and long ISI and includes the error for Vanilla-RNN, LSTM, and GRU. For each architecture, multiple bars are shown with the left four bars corresponding to T-BPTT with different T's and the right bar corresponding to RTRL. The results are calculated over 5 million steps and averaged over 30 runs. Similar to the trace conditioning benchmark, the T-BPTT based methods showed sensitivity to the truncation parameter. The use of RTRL always improved performance; however, except for ISI$\sim$10 no methods performed well: they all reached a level of error close to the fixed representations in Figure 5.6.

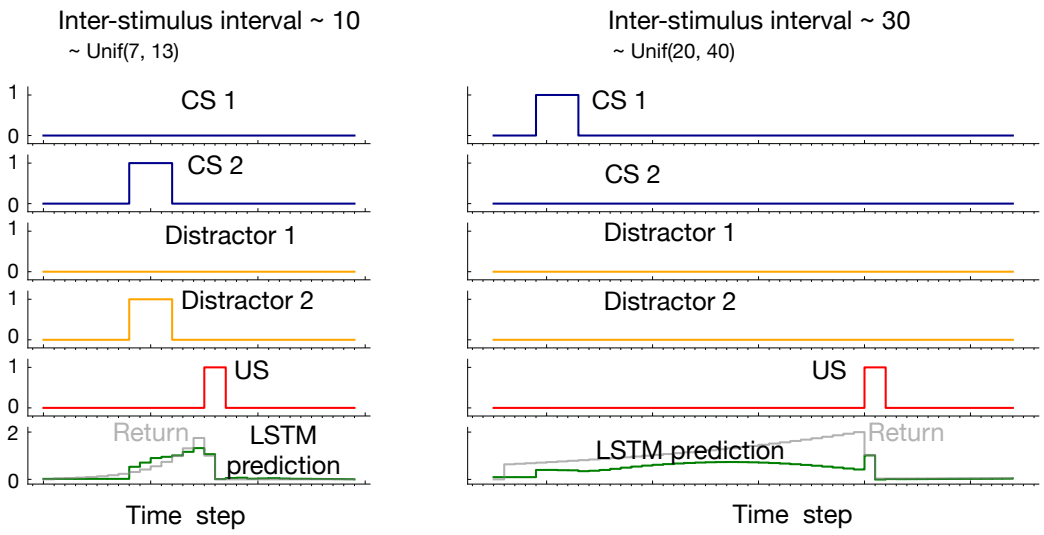

Figure 5.8: Example prediction profile plots for LSTM in the **trace patterning** benchmark in the the case of an expected ISI 10 and 30. LSTM was trained with T-BPTT and a truncation length of 40. Only two of the CS and distractors are shown as examples. In both cases, an activation pattern occurred as a result of which the US got activated. In the the case of expected ISI of 10, LSTM prediction resembled the return. In the case of longer ISI with the expectation of 30, however, LSTM did not predict the US accurately.

the case of expected ISI of 10, they fail to match the return in the case of the expected ISI of 30.

This result emphasizes the difficulty of the trace patterning benchmark — the tested recurrent networks struggle to achieve low error, even when they have access to better gradient approximations, as in the case of training with RTRL.

5.5 Combining stimulating traces with RNNs

Our experimental results highlight the limitations of the current learning methods. While the linear trace-based methods successfully bridge the temporal gap in the trace conditioning benchmark, their performance deteriorates when we introduce nonlinearities in the trace patterning benchmark. On the other hand, recurrent learning algorithms can simultaneously bridge the temporal gap and handle nonlinearities, but they can be expensive in computational and memory requirements

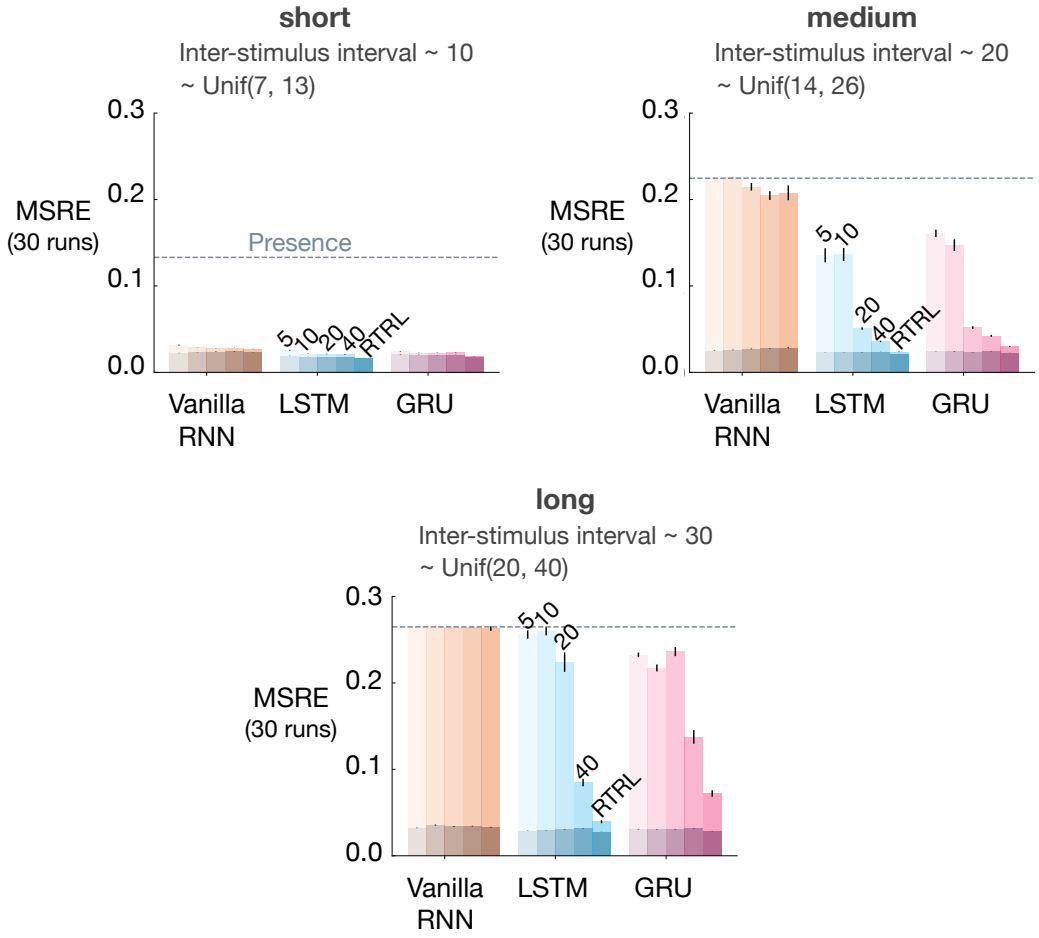

Figure 5.9: Results for combining stimulating traces with RNNs in the **trace conditioning** benchmark. We used the exact same scheme as Figure 5.2. Darker colors denote the combination of stimulating traces with the recurrent methods and lighter shades denote the recurrent methods. Each bar reports the MSRE averaged over 30 runs. The methods were trained for 2 million steps. The error bars denote the standard errors. Adding stimulating traces to the input of the Vanilla-RNN, GRU, and LSTM improved their performance in both T-BPTT and RTRL cases and made them less sensitive to the truncation length in the case of training with T-BPTT.

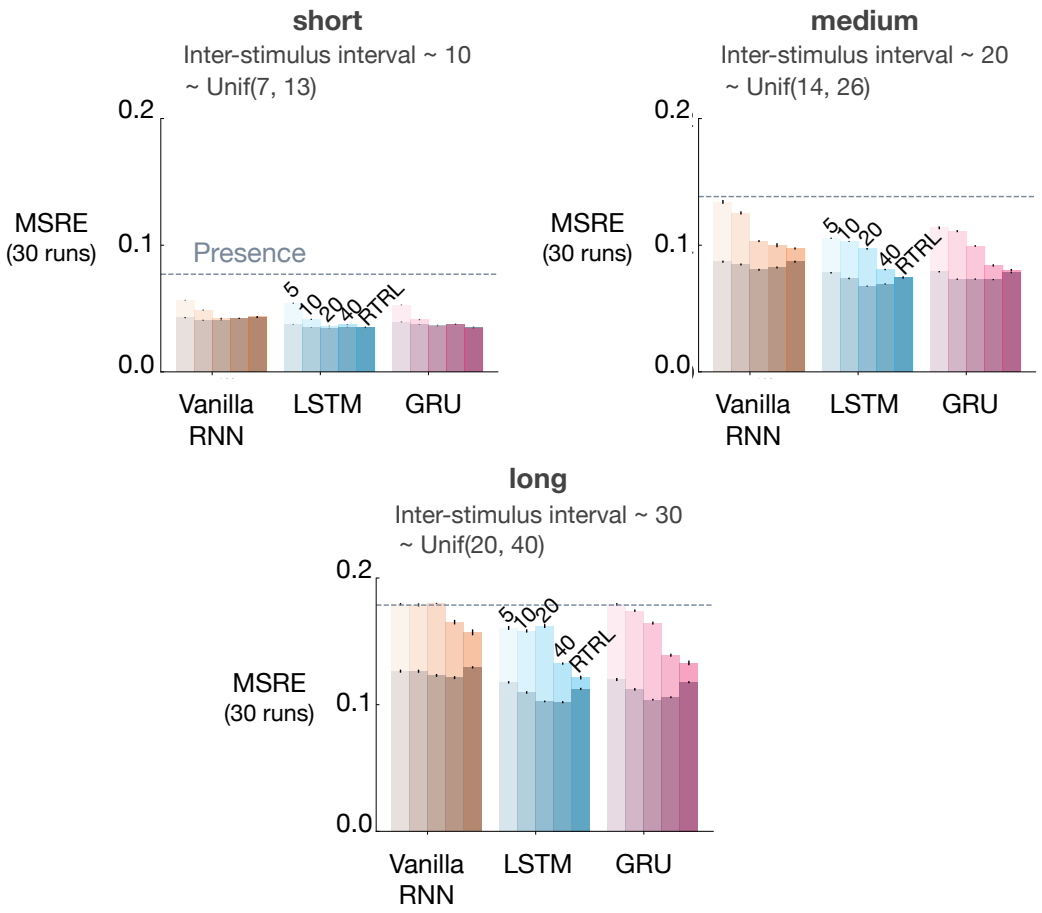

Figure 5.10: Results for combining stimulating traces with RNNs in the **trace patterning** benchmark. The naming conventions exactly match Figure 5.9, as does the general conclusion that stimulating traces improved performance but less so than in the trace conditioning benchmark.

In the case of T-BPTT, the memory requirements of RNNs grow linearly with the length of the truncation window, and learning long-term dependencies, as in the trace conditioning benchmark, requires a comparably long truncation window. In the case of RTRL, the computational complexity of RNNs grows quartically in the size of the hidden state, and learning patterns from a large number of signals, as in the noisy patterning quartically, requires a large hidden state. Ideally, we need training methods that scale well in computation and memory simultaneously.

As an example, we present a simple approach that scales well in computation and memory. We augment the RNNs with the stimulating memory traces of the observation. In particular, we feed an exponentially decaying trace of each stimulus, as described in tile-coded traces and microstimulus, as part of the input observation to the recurrent network.

Figure 5.9 and 5.10 show the effect of augmenting the RNNs with the stimulating memory traces of the observation respectively in the trace conditioning and trace patterning benchmarks. The results for RNNs fed with only the observation are also included in lighter shades for comparison.

When trained with T-BPTT, feeding the RNNs with stimulating traces significantly improved the performance for the Vanilla RNN, LSTM, and GRU variants in the trace conditioning benchmark. Moreover, it made the T-BPTT variants robust to the truncation length, achieving a similar level of error for all T's. This effect was more pronounced in the long setting (Figure 5.9). The reduced sensitivity of the T-BPTT variants to the truncation parameter suggests that feeding RNNs with stimulating traces enables them to achieve the same level of error with less computation.

When trained with RTRL, feeding the RNNs with stimulating traces helped improve the performance (Figure 5.9). The improvement was larger for Vanilla RNN than for the LSTM and GRU variants.

In the trace patterning benchmark, feeding the RNNs with the stimulating traces improved performance in both T-BPTT and RTRL variants but less so than in the trace conditioning benchmark.

While the space of ideas for fruitfully combining memory traces and RNNs

needs further investigation, this result shows how the proposed diagnostic benchmarks can help us search for general and scalable ideas for the online prediction problem.

5.6 Results for the GPT model

In this section, we provide the result of training a transformer model on the long setting of the trace patterning benchmark. Transformers have achieved remarkable results in sequence modeling tasks. It is commonly believed that their success is due to their ability to scale with computational resources through the use of large models. It would be interesting to see how they perform on our benchmark and how their computational expenses increase as we increase the difficulty of the benchmarks.

We used a GPT-style transformer, called minGPT[1], which includes blocks consisting of a multi-head attention module, layer normalizations, residual connections, and a feedforward network. See Figure 5.12. We set the number of blocks to one. At each time step, we stack the last k observations where k is referred to as the context length in transformers. Before passing the stack of observations to the block, we fed it into a perceptron with one layer and tanh nonlinearity. We then fed the output of the perceptron to the block. We used a context length of 40 because in the long setting of the trace patterning benchmark the ISI could be as large as 40. We used an embedding size of 64 and 4 heads for the multi-head attention module.

We ran minGPT for 5 runs and 5 million steps with different values for the step-size: $\{10^{-7}, 3^{-6}, 10^{-6}, 3^{-5}, 10^{-5}, 3^{-4}, 10^{-4}, 3^{-3}\}$. We then computed MSRE averaged over the 5 runs and picked the step-size that resulted in the lowest MSRE. Next, we ran minGPT with the best step-size for 30 runs and averaged MSRE.

Figure 5.12 shows the MSRE for the minGPT model. We have included the MSRE achieved by Vanilla-RNN, LSTM, and GRU trained with T-BPTT for $T = 40$ as well. Vanilla-RNN, LSTM, and GRU augmented with stimulating

[1] https://github.com/karpathy/minGPT

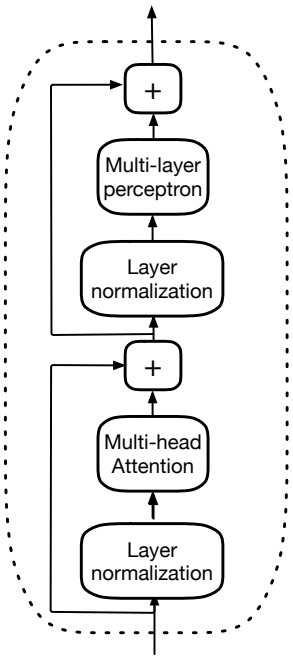

Figure 5.11: A block in the minGPT model consists of an attention module, layer normalizations, residual connections, and a multi-layer feedforward network.

traces are shown in darker shades. MinGPT achieved a lower level of error compared to the recurrent methods and a higher level of error compared to the recurrent methods augmented with stimulating traces.

While minGPT achieves a lower level of error compared to the RNN architectures trained with T-BPTT, its computational requirements increases quadratically with the length of the temporal association due to incorporating an attention module. To compare LSTM trained with T-BPTT with minGPT in terms of their computational requirements, we ran them for $10,000$ time steps for temporal associations of length $8, 32, 128, 512$ and computed the run times for 10 independent runs. We used a hidden layer size of 64 for LSTM and an embedding size 64 for minGPT. The truncation parameter for LSTM and the context length for minGPT were set equal to the length of the temporal association.

Figure 5.13 shows how the run time of LSTM and minGPT changes as a function of the length of the temporal association. LSTM's run time increased linearly with the length of the temporal association whereas minGPT's run

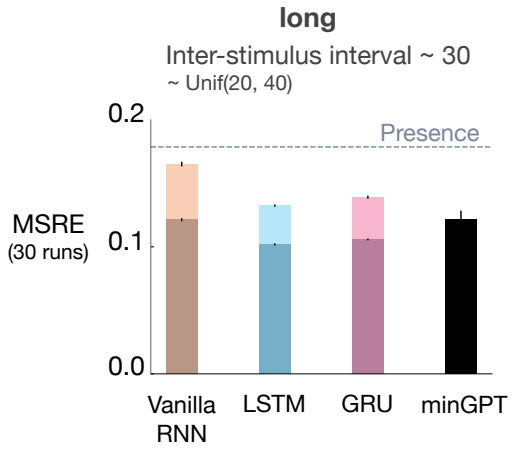

Figure 5.12: Results for minGPT in the **trace patterning** benchmark with ISI 30. The results for Vanilla-RNN, LSTM, and GRU trained with T-BPTT for $T = 40$ with and without stimulating traces are shown in light and dark shades. MinGPT achieved a lower level of error compared to the recurrent methods and a higher level of error compared to the recurrent methods augmented with stimulating traces.

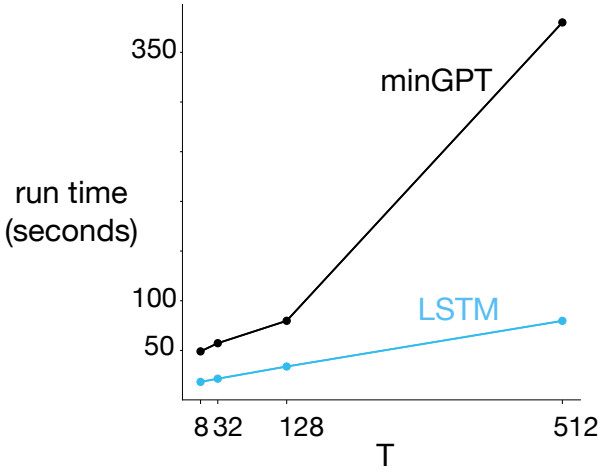

Figure 5.13: The run time of LSTM trained with T-BPTT and minGPT as a function of the length of the temporal association when ran for $10,000$ time steps. LSTM's run time increased linearly with the length of the temporal association whereas minGPT's run time approached a quadratic trend.

time increased more rapidly than linearly, approaching a quadratic trend. The computational data provided here, though empirically derived and specific to the implementations of the two methods, align with analytical expectations. Based on this result, minGPT used much more computation in order to achieve a lower level of error than LSTM.

5.7 Conclusions

The empirical study presented in this chapter together with the proposed benchmarks presented in Chapter 3 concludes the first major part of the thesis. Our empirical study suggests that the proposed benchmarks can be used to isolate and investigate fundamental challenges in state construction.

Our empirical study provided a granular understanding of prominent solution methods with a focus on recurrent learning architectures:

1. In the trace conditioning benchmark, vanilla RNN could not handle long temporal dependencies.

2. In the trace conditioning benchmark, gated architectures of LSTM and GRU exhibited significant sensitivity to the truncation parameter and did not perform as well as RTRL variants.

3. In the trace patterning experiments, all recurrent methods struggled when confronted with the combination of long temporal dependencies and the need to extract configuration patterns.

4. Feeding stimulating traces to RNNs trained with both T-BPTT and RTRL enhanced their performance and reduced the sensitivity of the T-BPTT variants to the truncation parameter.

5. MinGPT reached a lower level of error compared to the recurrent methods trained with T-BPTT. However, it required more computational resources and its computational requirements increased more rapidly with the length of the temporal association.

This thesis investigated methods whose computational complexity grows linearly or quadratically with the length of the temporal association. However, more stringent computational restrictions might be useful for future work. Many RL algorithms, like TD, can make and update long-horizon predictions with computation significantly less than the length of the prediction's horizon (van Hasselt and Sutton, 2015). This might also be possible in representation learning. Can the agent construct representations capable of overcoming dependencies back in time with computation and storage less than the length of the gap? While recurrent learning algorithms based solely on T-BPTT do not meet this requirement, our results show that some combination of stimulating traces and recurrent architectures may reduce the agent's dependency on the truncation level.

Moreover, there is a discrepancy between the speed of learning for natural and artificial systems; while animals learn eyeblink conditioning in about a few hundred trials, our baseline methods require thousands of trials to learn the task. Future research should investigate reasonable computational restrictions if we hope to discover representations as efficient as those used by animals.

Chapter 6

Auxiliary Task Discovery Through Generate and Test

This chapter presents the third contribution of this thesis which is a new method for auxiliary task discovery. *Auxiliary tasks* are prediction or control tasks auxiliary to the main task of maximizing the discounted sum of rewards. They can support the learning of the main task in several ways including by assisting state construction or exploration. This chapter presents a method for discovering auxiliary tasks that would assist state construction.

The third contribution is different from the first two contributions in that it presents a novel solution method whereas the previous contributions were mostly about existing solution methods. While the first two contributions presented comprehensive results by systematically studying different solution methods, the conclusions from the third contribution is more speculative as the approaches to the problem of auxiliary task discovery are still evolving.

In a sense, the contribution presented in this chapter has the highest potential to impact the field because it tackles a relatively unexplored but important problem. There are only a few existing solution methods in the literature that discover auxiliary tasks systematically.

6.1 Auxiliary tasks

As mentioned earlier, auxiliary tasks are prediction and control tasks about signals other than the long-term reward. Learning auxiliary tasks can po-

tentially assist the learning of the main task in several ways. The ability to predict and control different aspects of the environment can constitute a form of environment model (Sutton and Barto, 2018). Solving auxiliary tasks would result in options or temporally extended actions that could then be used for exploration or for constructing option models (Sutton et al., 2023). Finally, learning about auxiliary tasks can assist state construction.

There are several ways in which learning about auxiliary tasks can assist state construction. The agent state can be represented in terms of auxiliary tasks about the data stream as in predictive state representation (Littman, Sutton, and Singh, 2001). Predictions learned for auxiliary tasks can also be fed to the state update function (Ma 2020). Auxiliary tasks can also assist state construction by shaping the representation shared between the main task and the auxiliary tasks. For the rest of this chapter, we will focus on this last approach.

To share the representation between the main task and the auxiliary tasks, multi-headed neural networks are used where the last hidden layer acts as the state representation (Jaderberg et al., 2016). In this setting, each head corresponds to either the main task or one of the auxiliary tasks. The errors propagated from all heads make changes to the shared representation.

A common view is that auxiliary tasks can speed up learning the main task because they may be easier to learn and may require some of the same representation that is required for learning the main task (Jaderberg et al., 2016; Shelhamer et al., 2016; Mirowski et al., 2016). In environments with sparse reward structures, auxiliary tasks provide instantaneous targets for shaping the representation in the absence of reward. Auxiliary tasks can also arguably function as regularizers, improving the generalization and avoiding representation overfitting in RL (Dabney et al., 2020).

Auxiliary tasks could be posed by the agent. The problem of enabling agents to discover useful auxiliary tasks is referred to as the problem of *auxiliary task discovery*. Relying on human experts for designing auxiliary tasks is not ideal because it is challenging to know what auxiliary tasks will be useful in advance. Moreover, the results on hand-designed auxiliary tasks

are mixed: in some cases, auxiliary tasks result in substantial performance gain over the baselines whereas in other cases they achieve marginal improvements (Jaderberg et al., 2016) or even harm the performance (Shelhamer et al., 2016). Finally, the usefulness of auxiliary tasks might change over the course of learning.

Recently, there has been some progress in answering the question of what makes useful auxiliary tasks. Dabney et al. (2020) argue that learning about the value improvement path constitutes useful auxiliary tasks where the value improvement path is the sequence of value functions produced by the policy improvement process in RL. The usefulness of auxiliary tasks has been connected to how well the gradient direction proposed by them is aligned with the gradient direction of the main task (Lin et al., 2019; Du et al., 2020). Wang et al. (2022) investigated how different auxiliary tasks affect the properties of the representation learned by a DQN system with numerous auxiliary tasks when transferring from one task to another. While these works give hints to what constitutes useful auxiliary tasks, they do not provide a complete solution to the problem of auxiliary task discovery.

We explore a generate-and-test approach for auxiliary task discovery. The proposed generate-and-test method continually generates auxiliary tasks, evaluates them, and replaces those recognized to be least useful. This algorithm is the first instance of an auxiliary task discovery method based on the idea of generate-and-test. In the next sections, we will develop the generate-and-test method and show its effectiveness empirically.

6.2 Auxiliary task discovery through generate-and-test

The proposed method for auxiliary task discovery is based on a class of algorithms called generate-and-test. Generate-and-test was originally proposed as an approach to representation learning or feature finding where new features are continually generated using a generator, evaluated using a tester, and replaced if recognized as useless. (See Chapter 4.) This idea has a long

history in supervised learning (Sutton and Whitehead 1993; Mahmood and Sutton, 2013), and can even be combined with backprop (Dohare et al., 2023). The same basic structure can be applied to auxiliary task discovery, which we explain next.

We use generate-and-test for discovering and retaining auxiliary tasks that induce a representation useful for learning the main task. Our proposed generate-and-test method consists of a generator and a tester. The *generator* generates new auxiliary tasks and the *tester* evaluates the auxiliary tasks. The auxiliary tasks that are assessed as useful are retained while the auxiliary tasks that are assessed to be useless are replaced by newly generated auxiliary tasks. The newly generated auxiliary tasks will most likely have low utility. To prevent the replacement of newly generated auxiliary tasks, we calculate the number of steps since their generation and refer to that as their age. An auxiliary task can only be replaced if its age is bigger than some age threshold. Every T time steps, some ratio of the auxiliary tasks gets replaced. We refer to T as the replacement cycle and denote the replacement ratio by ρ. The pseudo-code for the proposed generate-and-test method is shown in Algorithm 1.

Note that the proposed method does not generate-and-test on features but on auxiliary tasks. It, however, does assess the utility of features and derives the utility of the auxiliary tasks from the utility of the features that they induced. We will explain this in more detail in the next section.

6.3 The proposed tester

We propose a tester that evaluates the auxiliary tasks based on how useful the features induced by them are for the main task. It is challenging to recognize which auxiliary tasks induce useful representations. Our proposed tester does this in two phases. First, it evaluates how good each feature is based on how much it contributes to the approximation of the main task action-value function. Here we define the features to be the output of the neural network's last hidden layer after applying the activation function. Next, the tester identifies

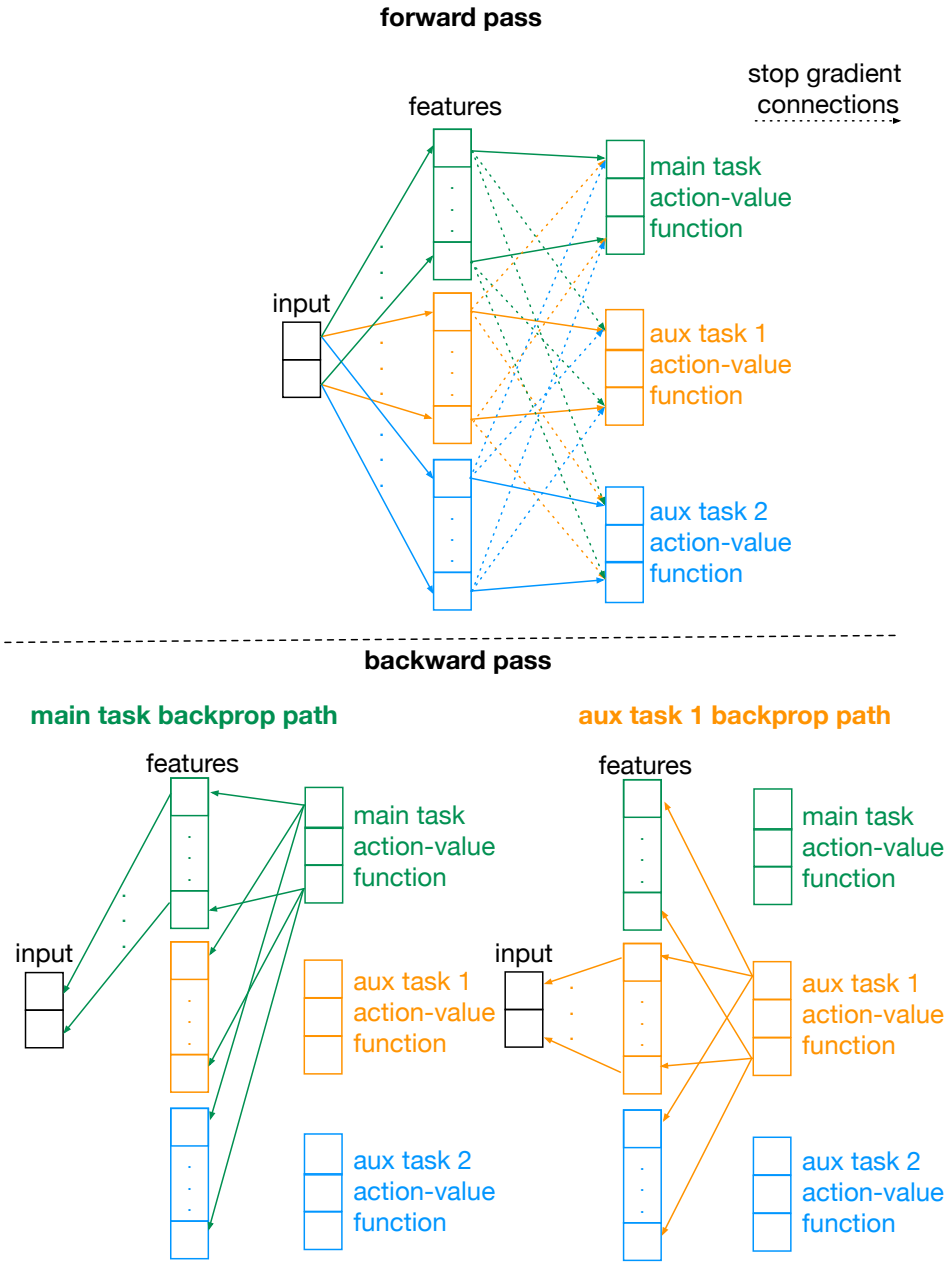

Figure 6.1: The forward pass, backward pass for the main task, and backward pass for auxiliary task 1 when using the Master-User strategy for learning auxiliary tasks alongside the main task. All features are used by all tasks in the forward pass but only modified through the gradient backpropagated from one task. The dotted arrows show stop-gradient connections. The gradients do not go back any further from these connections. When using the Master-User strategy, it is clear which auxiliary task was responsible for inducing which feature.

Algorithm 1 Generate-and-test for auxiliary task discovery

Input: number of auxiliary tasks n, age threshold μ, replacement cycle T, replacement ratio ρ
Initialization:
generate n auxiliary tasks using the generator
randomly initialize the base learning network
set age a_i for each auxiliary task to zero
for Every time step **do**
    do a DQN step to update the base learning network
    Increase a_i by one for $i = 1, \ldots, n$
    update the utility of all auxiliary tasks using the tester
    for Every T time steps **do**
        Find $n\rho$ auxiliary tasks with the lowest utilities such that $a_i > \mu$
        replace the $n\rho$ auxiliary tasks with new auxiliary tasks generated by the generator
        reinitialize the input and output weights of the features induced by the $n\rho$ auxiliary tasks
        reset a_i to zero for the $n\rho$ auxiliary tasks
    end for
end for

which auxiliary task was responsible for shaping which features.

Let us first consider the problem of recognizing which auxiliary tasks get credit for shaping which features. When following the standard practice of jointly learning the main task and the auxiliary tasks, recognizing which feature was influenced the most by which auxiliary task is challenging. This is because all features are jointly shaped by all the tasks, both auxiliary and main. To address this issue, we use a strategy for learning the representation where all features are used by all tasks in the forward pass; however, each feature is only modified through the gradient backpropagated from one task. See Figure 6.1. This learning strategy is similar to the Master-User algorithm proposed for continual recurrent learning (Javed et al., 2023). Therefore, we refer to this learning strategy as the Master-User strategy. When using the Master-User strategy, it is clear which auxiliary task was responsible for inducing which feature.

Next, we require a measure of feature usefulness. To assess each feature, the proposed tester looks at the magnitude of the outgoing weights from the

feature to the main task action-value function for all actions. The greater the magnitude is, the more important the feature is. The tester also considers the trace of the magnitude of each feature: the greater the trace of the feature is, the more it contributes to the approximation of the main task action-value function. The magnitude of the weights times the trace of the magnitude of the feature represents how much the feature contributes to the approximation of the main task action-value function. Therefore, the utility of feature k is defined as:

$$u_k = \bar{f}_k \times \sum_a |w_{ka}^{\text{main}}| \tag{6.1}$$

where the utility of feature k is denoted by u_k. w_{ka}^{main} is the output weight from feature k to the main action-value function for action a. $\bar{f}_k$ is a *trace* of feature k defined as:

$$\bar{f}_k \leftarrow (1-\tau)\bar{f}_k + \tau f_k \tag{6.2}$$

where f_k is the value of feature k at the current time step and $0 < \tau < 1$ is the trace parameter. This assessment method is similar to what has been used in generate-and-testing on features (Mahmood and Sutton, 2013).

After assessing the utility of the features, the utility of each auxiliary task is set to the sum of the utility of the features shaped by it:

$$u(\text{aux}^i) = \sum_{k \in F^i} u_k$$

where F^i are the features shaped by auxiliary task i.

6.4 The random generator

We combined the proposed tester with a simple generator that randomly generates auxiliary tasks. The auxiliary tasks are formulated as *subgoal-reaching* GVFs where the continuation function returns 0 at the subgoals and 1 elsewhere. (See Chapter 2 for an explanation of GVFs.) The cumulant is -1 everywhere and the policy is greedy.

The subgoals are uniformly randomly selected from the observation space, meaning the agent is learning many policies to reach different parts of the observation space in addition to solving the main task.

6.5 Experimental setup

In this section, we provide empirical results supporting the efficacy of the proposed generate-and-test method for auxiliary task discovery. We include results on two gridworld environments: four-rooms and maze. We also include results on the pinball environment (Konidaris and Barto, 2009), which is widely used in skill chaining, option discovery, and recently model-based planning (Lo et al., 2022). We chose these environments so that we could easily visualize the discovered auxiliary tasks and easily design good and bad auxiliary tasks as baselines. All environments are episodic.

In the gridworld environments, the goal is to learn the shortest path from the start state to the goal. The start and goal states are denoted by S and G respectively in Figure 6.2 and 6.4. At each cell, four actions are available: up, down, left, and right. There is some degree of stochasticity when transitioning from one state to another: with probability 0.5 the agent will transition in the same direction as the selected action, otherwise, it will transition in one of the remaining directions with equal probability. The observation space is described with a one-hot representation with the index corresponding to the agent's position being 1. The reward is -1 on each time step.

In the pinball environment, a small ball should be navigated to the goal in a maze-like environment with simplified ball physics. In Figure 6.4, the pinball environment is shown with the ball shown by a grey circle. The goal and start states are denoted by S and G respectively. Collision with the obstacles causes the ball to bounce. The observation space is continuous and is described by x, y, $\dot{x}$, $\dot{y}$. The start location and goal location are at $(0.8, 0.5)$ and $(0.1, 0.1)$ respectively. The action space includes 5 actions of increasing or decreasing $\dot{x}$ or $\dot{y}$ and no change to $\dot{x}$ and $\dot{y}$. The reward is -1 at each time step. There is no episode cutoff.

Note that in the original pinball environment, the agent receives a special reward of $10,000$ upon arrival at the goal. Instead, we gave a reward of -1 (like every other step) so that the scale of the action-value function for the main task and the auxiliary tasks would not be too different. When learning multiple tasks in parallel, the contribution of each task is determined by the scale of the corresponding value function (Hessel et al., 2019). Therefore, when the scale of value functions is very different, we would need to scale the reward of the main task and the cumulants of the auxiliary tasks appropriately. This issue requires an additional hyper-parameter that would give our method an advantage if tuned. For this work, we decided to focus on the case where the scale of the value function for the main task and the auxiliary tasks are similar.

As the base learning system, we used DQN with Adam optimizer which is a standard choice. We used a neural network with one hidden layer and *tanh* activation function. (We used *tanh* activation function so that the induced features would be all in the same range of $(-1, 1)$; however, our proposed tester should work well when other activation functions are used too. This can be investigated in future work.) For the girdworld environments, the one-hot observation vector was fed to the neural network. The hidden layer size for the baseline with no auxiliary task in four-rooms and maze were 50 and 500 respectively. For the pinball environment, the 4-dimensional observation was normalized and fed to the neural network. The hidden layer size was 500. The replay buffer size for the four-rooms, maze, and pinball environments were 500, 1000, and $10,000$ respectively. For all environments, we used a batch-size of 16. The target network update frequency for the gridworld and pinball environments were set to 100 and 200 respectively. We set all these hyper-parameters through an informal search, seeking a configuration for the base learning system that would yield reasonable results.

6.6 Evaluating the proposed tester

To see how well the proposed tester evaluates the auxiliary tasks, we designed good and bad auxiliary tasks in the four-rooms environment. The hand-

designed auxiliary tasks were formulated as subgoal-reaching GVFs with the good and bad hand-designed auxiliary tasks having hallway and corner subgoals respectively. See Figure 6.2. We used the Master-User architecture when learning the hand-designed auxiliary tasks.

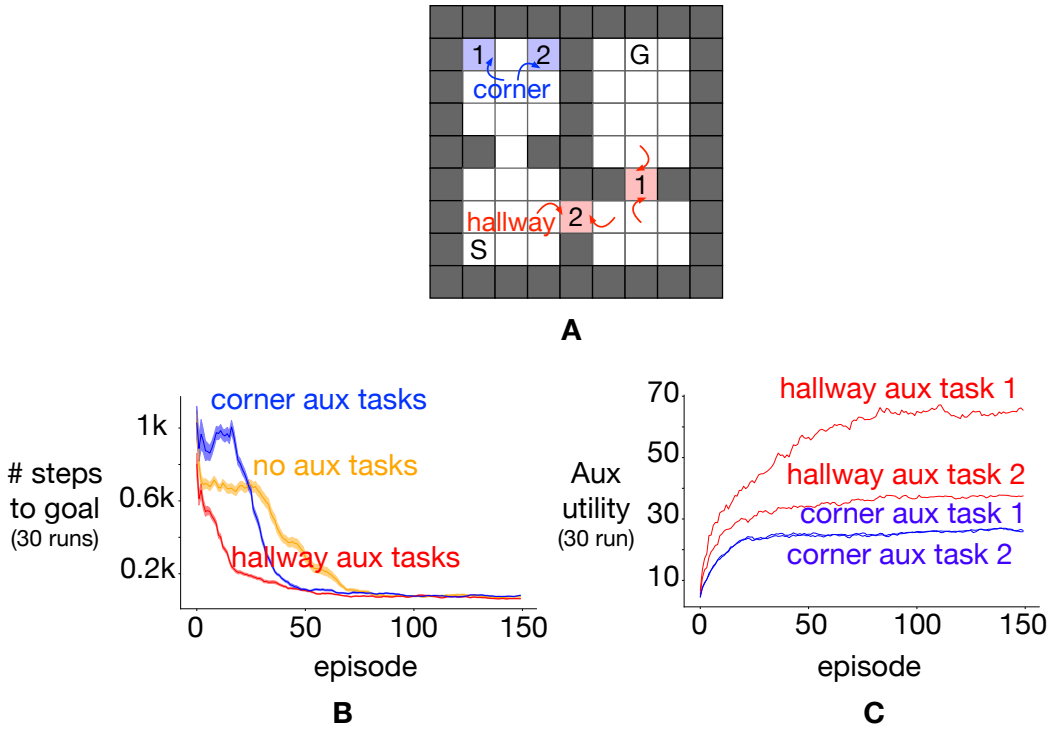

Figure 6.2: A: The four-rooms environment with the subgoals corresponding to the good and bad hand-designed auxiliary tasks shown in red and blue respectively. B: Hallway auxiliary tasks improved the performance in terms of learning speed. The corner auxiliary tasks made learning slower in the early episodes. C: The proposed tester evaluated the hand-designed auxiliary tasks well, giving higher utility to the hallway auxiliary tasks. The results are averaged over 30 runs and the shaded regions depict the standard error.

When learning the auxiliary tasks alongside the main task using the Master-User strategy, the gradient backpropagated from the main task only modifies $\frac{1}{\#\text{auxiliary tasks}+1}$ percent of the features. For example, in the case of learning the hallway auxiliary tasks, there are 2 auxiliary tasks. Therefore, the gradient backpropagated from the main task only modifies 33.3% of the features.

When including auxiliary tasks, we adjusted the hidden layer size of the network such that the total number of learnable parameters is roughly equal across methods. For example, in the four-rooms environment, for the case of

no auxiliary task there are a total of $49 \times 50 + 50 \times 4 = 2650$ parameters, with an observation size of (network input size) 49 , hidden layer size of 50, and 4 actions (network output size). For the baseline with hallway auxiliary tasks, we used a smaller hidden layer size of 43 so that it results in roughly the same number of total parameters: $49 \times 43 + 43 \times 4 \times 3 = 2623$. In all the experiments, we kept the number of learnable parameters roughly equal across methods with and without auxiliary tasks.

The hallway auxiliary tasks improved learning in terms of learning speed as we expected (Figure 6.2, bottom left graph). The corner auxiliary tasks, on the other hand, hurt the performance in the early episodes and resulted in suboptimal performance in comparison to the hallway auxiliary tasks. As we mentioned earlier, when including auxiliary tasks, we used a smaller hidden layer size such that the total number of learnable parameters is equal to the case of no auxiliary tasks. It is interesting that in this environment with such a small observation space and no partial observability, dedicating a considerable percentage of the learnable parameters to learning auxiliary tasks can result in better performance compared to dedicating all the parameters to learning the main task.

The proposed tester evaluated the hallway and corner auxiliary tasks well, assigning much higher utility to the hallway auxiliary tasks and clearly indicating the corner tasks are bad (Figure 6.2, bottom right graph). The utility of all hand-designed auxiliary tasks started around the same point. However, the utility of the good auxiliary tasks reached a much higher level compared to the bad auxiliary tasks over time.

We also conducted a more thorough experiment in which we included all the cells as subgoals. There were a total of 39 subgoals. We used a neural network with a hidden layer of size 240 so each task would modify 6 features. We observed that the tester gave higher scores to the auxiliary tasks with subgoals in the top right and bottom right rooms, that is subgoals closer to the goal state (Figure 6.3, bottom right).

To test whether the auxiliary tasks with subgoals closer to the goal state are indeed more useful auxiliary tasks in four-rooms, we conducted another

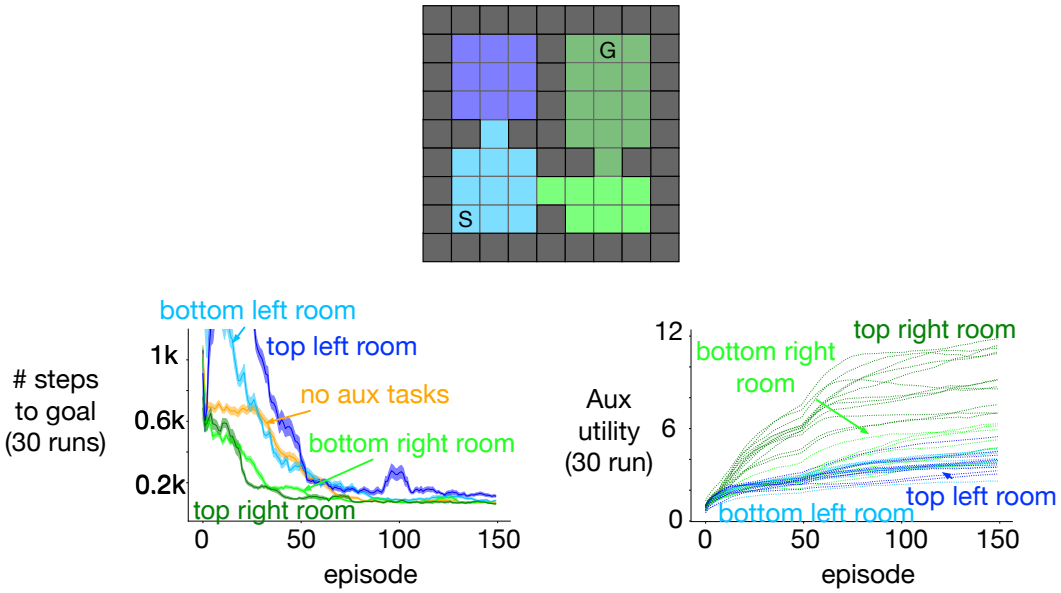

Figure 6.3: The tester gave higher scores to the auxiliary tasks with subgoals in the top right and bottom right rooms as shown in the bottom right subplot. The auxiliary tasks from the top right and bottom right rooms accelerated learning and were indeed more useful as shown in the bottom left subplot.

experiment where we considered auxiliary tasks with subgoals from each room separately and compared the result with the baseline with no auxiliary tasks. More specifically, to evaluate the subgoals from each room, in each run, we picked 5 subgoals from that room uniformly randomly and learned the corresponding auxiliary tasks alongside the main task. We did this so the number of auxiliary tasks from each room would be the same. We observed that the auxiliary tasks from the top right and bottom right rooms accelerated learning and were indeed more useful (Figure 6.3, bottom left).

6.7 Evaluating the generate-and-test method

In this section, we provide the result of combining the base learning system with the proposed generate-and-test method. The generate-and-test method uses the combination of the *random generator* and our proposed tester. The random generator produces subgoal-reaching auxiliary tasks with the subgoals randomly picked from the observation space. More specifically, in the grid-world environments, the subgoals are cells in the grid. In the pinball environ-

ment, the subgoals are determined by (x, y) and once the ball is within radius 0.035 of a subgoal, it is assumed that the agent has reached the subgoal.

We included two baselines for comparison which included the base learning system with no auxiliary tasks and fixed random auxiliary tasks. All the auxiliary tasks were in the form of subgoal-reaching tasks. For the fixed random auxiliary tasks, the subgoals were randomly picked from the observation space and kept fixed throughout learning. The number of auxiliary tasks for the baseline with fixed random auxiliary tasks was set equal to the number of auxiliary tasks for the generate-and-test method.

We systematically swept the step-size parameter and report the performance of the best to ensure a fair comparison. To do so, we ran the baseline with no auxiliary tasks with different values of the step-size for 10 runs. We used the step-size that resulted in the lowest final error (last 10 percent episodes) and reran the baseline with the best step-size for 30 runs to get the final results. We repeated this process for all methods. For four-rooms, maze, and pinball the sweep over the step-sizes included $\{0.000625, 0.0025, 0.01, 0.04\}$, $\{0.00025, 0.001, 0.004\}$, and $\{0.00125, 0.0025, 0.005, 0.01, 0.02\}$.

The generate-and-test method has hyper-parameters of its own: 1) number of auxiliary tasks 2) age threshold 3) replacement cycle 4) replacement ratio 5) trace parameter. We set these hyper-parameters through an informal search. For the gridworld environments, we used 5 auxiliary tasks, an age threshold of 0, a replacement cycle of 500 steps, a replacement ratio of 0.2, and a trace parameter 0.05. For the pinball environment, we used 4 auxiliary tasks, an age threshold of 0, a replacement cycle of 500 steps, a replacement ratio of 0.25, and a trace parameter 0.01.

First, let us take a look at the learning curves for the four-rooms environment shown in Figure 6.4. The proposed generate-and-test method improved over the baseline with no auxiliary tasks bridging the gap between the baseline with no auxiliary tasks and the baseline with hallway auxiliary tasks. Note that generate-and-test is slower than the baseline with hallway auxiliary tasks. This is because generate-and-test is searching the space of auxiliary tasks, starting with random ones, testing them, and retaining those recog-

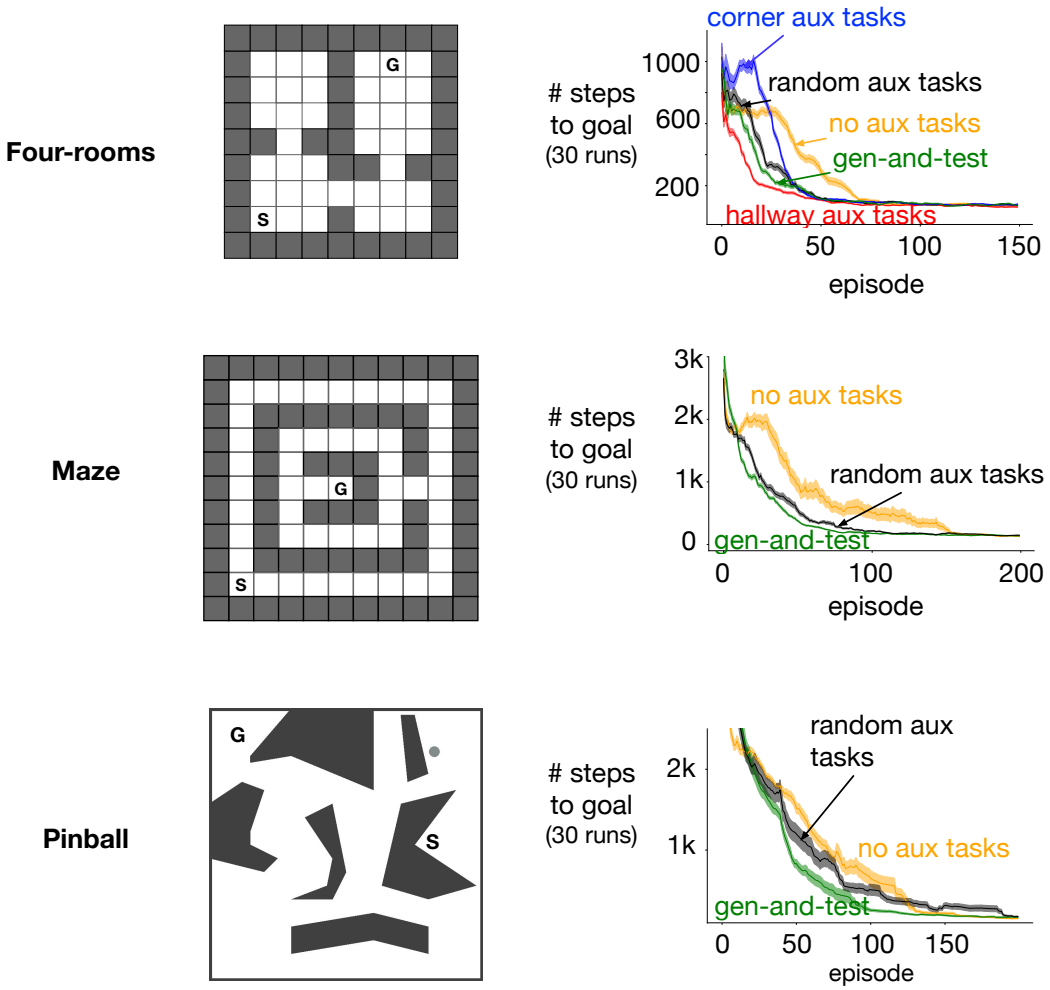

Figure 6.4: The learning curves for the proposed generate-and-test method (green), the baseline with no auxiliary tasks (orange), and the baseline with fixed random auxiliary tasks (black). The results are averaged over 30 runs and the shaded regions depict the standard error. The proposed generate-and-test method improved over the baseline with no auxiliary tasks. Generate-and-test also outperformed the baseline with fixed random auxiliary tasks. Fixed random auxiliary tasks also resulted in performance gain over the baseline with no auxiliary tasks.

nized as useful whereas the baseline with hallway auxiliary tasks starts with reasonably good auxiliary tasks from the beginning. The generate-and-test method outperformed the baseline with no auxiliary tasks in the maze and pinball environments as well (Figure 6.4).

Generate-and-test also outperformed the baseline with fixed random auxiliary tasks in all three environments. This suggests that the choice of the auxiliary tasks was important and generate-and-test discovered and retained useful auxiliary tasks.

Interestingly, the fixed random auxiliary tasks resulted in significant performance gain over the baseline with no auxiliary tasks in the gridworld environments (Figure 6.4). This is in line with the findings from the literature suggesting that random GVFs can form good auxiliary tasks for reinforcement learning (Zheng et al., 2021). In the pinball environment, however, the performance gain for the fixed random auxiliary tasks was small. This could be because the number of random auxiliary tasks is only 4 which is relatively small for the pinball environment.

The auxiliary tasks discovered and retained by generate-and-test are shown in Figure 6.5. To plot the discovered auxiliary tasks, we ran the generate-and-test method for 30 runs and stored the auxiliary tasks that were retained. The green squares correspond to the discovered auxiliary tasks in the gridworld environments. Darker green indicates that the cell was chosen as a subgoal in many runs.

For the pinball environment, the discovered auxiliary tasks are shown in green circles. In the gridworld environments, the subgoals corresponding to the discovered auxiliary tasks were close to the goal states. In the pinball environment, the discovered auxiliary tasks were more concentrated in the central areas—reasonable way-points on the path to the goal.

6.8 The feature-attainment generator

In this section, we propose and investigate a new auxiliary task generator that has better scaling potential compared to the random generator. While the

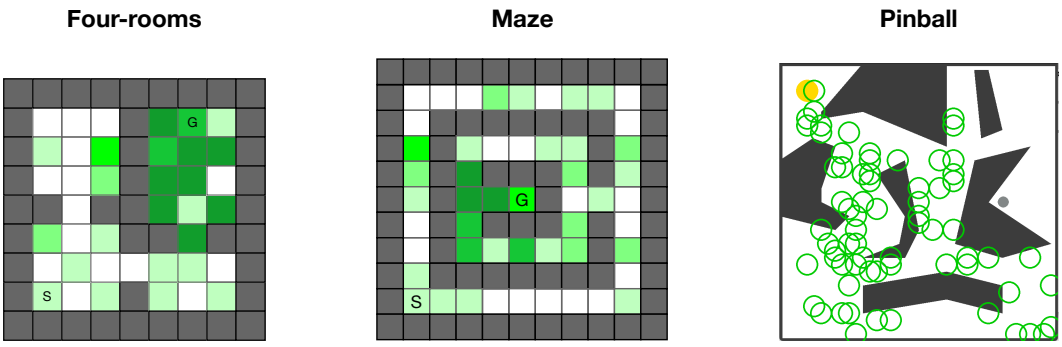

Figure 6.5: Example discovered auxiliary tasks in the three environments. Generate-and-test discovered reasonably good auxiliary tasks: in the grid-world environments, the subgoals corresponding to the discovered auxiliary tasks were close to the goal states. In the pinball environment, the discovered auxiliary tasks were more concentrated in the central areas.

random subgoal-reaching generator worked well in our experiments, it may not be feasible in cases with a large observation space where the space of possible subgoals is large. To improve the scalability of our generate-and-test method, we propose that the discovery method searches in the space of auxiliary tasks that are about features rather than searching in the space of auxiliary tasks that are about the input observations.

The new generator is based on the idea of *feature-attainment* recently introduced for option discovery in planning (Sutton et al., 2023). In feature-attainment auxiliary tasks, the goal is to maximize an individual feature of interest (or component of the representation layer) which we refer to as the target feature. For feature-attainment auxiliary tasks, the continuation function returns 0 when the target feature has its maximum value and 1 otherwise. Recall that we use tanh activation functions, and thus the maximum value a feature can take on is one. The cumulant is -1 everywhere and the policy is greedy.

When generating feature-attainment auxiliary tasks, the question is what the target features should be. We propose that the generator picks the target features only from the features induced by the main task. We designed the generator such that when picking from the features induced by the main task, it looks at the score of the features calculated according to Equation 6.1 and

picks the ones with the highest scores. Therefore, the new generate-and-test method will be searching in the space of auxiliary tasks that are about the features that 1) contribute the most to the main task action value function 2) are induced by the main task.

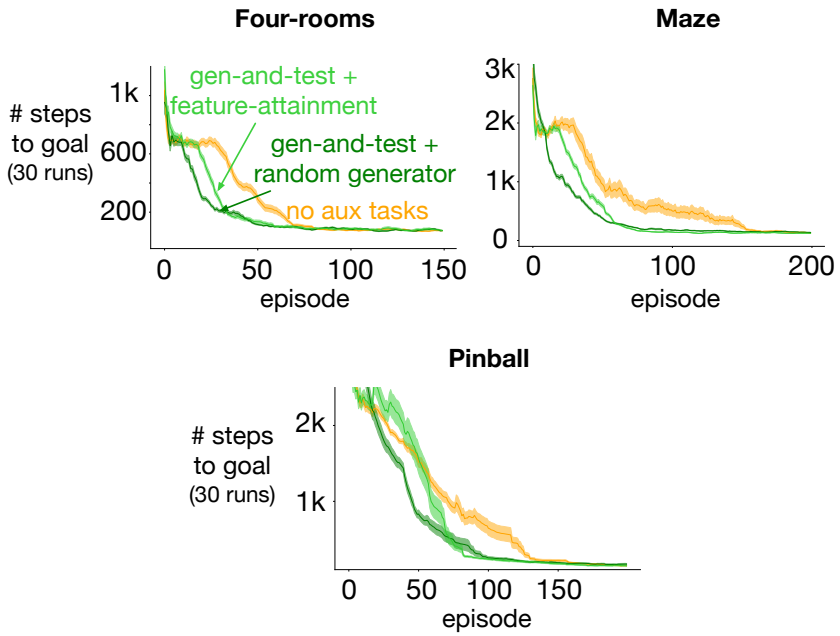

Figure 6.6: The learning curves for the proposed generate-and-test method with the feature-attainment generator (lime green) and the baseline with no auxiliary tasks (orange). The results are averaged over 30 runs and the shaded regions depict the standard error. The proposed generate-and-test method with the feature-attainment generator improved over the baseline with no auxiliary tasks. The random generator resulted in faster learning compared to the feature-attainment generator. However, the feature-attainment is potentially more scalable than the random generator.

We tested the generate-and-test method with the feature-attainment generator on the three environments. We used 3, 8, 4 auxiliary tasks for four-rooms, maze, and pinball respectively. For four-rooms and maze, we used a replacement cycle of 2000 steps, a replacement ratio of 0.2, and a trace parameter 0.05. For the pinball environment, we used a replacement cycle of 1000 steps, a replacement ratio of 0.25, and a trace parameter 0.01.

The generate-and-test method with the feature-attainment generator improved over the baseline with no auxiliary tasks across the three environments (Figure 6.6). For the first few episodes, the generate-and-test method with

the feature-attainment generator did not improve over the baseline with no auxiliary tasks, unlike the variant with the random subgoal-reaching generator. We speculate that this is because the features are not useful in early learning and thus the feature attainment auxiliary tasks are somewhat arbitrary. With further learning, the features become more relevant to the main task progressively becoming better auxiliary tasks.

6.9 Related work

Veeriah et al. (2019) proposed a method for auxiliary task discovery using meta-gradient. Meta-gradient methods are higher-level learning methods that adapt the meta-parameters of the base learning system, such as step-sizes and eligibility traces parameters, through gradient descent (Xu et al., 2018). Veeriah et al. (2019) applied meta-learning to discovering auxiliary tasks defined via General Value Functions by adapting the parameters that define the goal (cumulant) and termination functions via gradient-descent. Their proposed method considers the consequence of changing the continuation function and cumulant meta-parameters on the subsequent performance of the agent. Generally speaking, these meta-learning approaches are notoriously difficult to tune (Antoniou et al., 2018) and require large amounts of training data and compute as changing the meta-parameters at a time step affects both the update to the parameters at the next time step and all future updates.

The meta-learning and generate-and-test approaches explore two completely different approach to the problem of auxiliary task discovery. Rather than being viewed as mutually exclusive alternatives, they can be considered as complementing each other. Meta-learning approaches can be augmented with generate-and-test mechanisms where auxiliary tasks discovered using generate-and-test get further refined using meta-learning. Similar ideas have produced promising results in representation learning where simple generate-and-test significantly improved classification and regression performance when combined with backpropagation (Dohare et al., 2021). We leave the study of the comparison and combination of meta-learning approaches with generate-and-

test mechanisms to future work as such an effort is worthy of an entire study on its own.

He et al. (2022) have also proposed a method for auxiliary tasks discovery using an evolutionary search strategy. They consider auxiliary tasks in form of predictions about the elements of the sequence of states, actions, and rewards given other elements of the sequence of states, actions, and rewards. He et al. (2020) used their proposed method to search auxiliary losses over a small set of environments. Then, they showed the efficacy of the auxiliary loss discovered over the small set of environments on a wider range of environments. Their approach diverges from the goal of continually searching for useful auxiliary losses, instead focusing on identifying a globally effective auxiliary loss.

Other works related to auxiliary task discovery include the works on general value function networks (Schlegel et al., 2021) and hindsight experience replay (Andrychowicz et al., 2017). Schlegel et al. (2021) argue that generate-and-test is a reasonable avenue for the problem of discovery in general value function networks. While they do not develop a complete discovery method, they show that the measure of feature usefulness proposed by Mahmood and Sutton (2013) is effective in discarding dysfunctional GVFs. Hindsight experience replay (Andrychowicz et al., 2017) can also be thought of as performing some form of discovery where the states encountered at the end of episodes and stored in the replay buffer are used as subgoals.

Finally, a topic closely related to auxiliary task discovery is that of option discovery. Options are temporally extended actions that also include a notion of termination (Sutton et al., 1999). Options and auxiliary tasks are closely related as solving each auxiliary task produces an option. There is a large body of work on option discovery including discovering options based on bottleneck states (McGovern and Barto, 2001), from the successor representation (Ramesh et al., 2019; Machado et al., 2023), by maximizing diversity (Eysenbach et al., 2018), or by utilizing meta-gradient (Veeriah et al., 2021).

Although, many of the works on option discovery rely on designers' intuitions regarding what constitutes good options (e.g., bottleneck states, diversity), they have hints to general principles for designing agents capable of

discovering and evaluating options automatically. The general principles underlying some of the option discovery methods could be applied to auxiliary task discovery, and conversely, insights from auxiliary task discovery could inform the development of new option discovery approaches.

6.10 Conclusions

This chapter presented a method for discovering auxiliary tasks useful for state construction. The proposed method is the first instance of an auxiliary task discovery method based on the idea of generate-and-test. We showed for the first time that a generate-and-test style search for auxiliary tasks, using a fairly naive random generator, can be surprisingly effective. Through careful experimentation, we showed that:

1. The proposed tester reasonably evaluates the auxiliary tasks.

2. The generate-and-test method improves over the baseline with no auxiliary tasks.

Despite significant interest, the problem of auxiliary task discovery is still an open problem. The generate-and-test method proposed in this chapter is a step toward designing generic methods for auxiliary task discovery. However, it is by no means the final algorithm for auxiliary task discovery.

The proposed method is limited in various ways. The tester enforces the use of the Master-User architecture which may not be applicable in deeper networks. The random generator is not feasible in environments with a big observation space, such as pixel-based environments. The proposed method introduces new hyper-parameters and the sensitivity of the method to these hyper-parameters is not clear yet. There is a big space of ideas to try to improve both the proposed method and our understanding of it. We will discuss some of these ideas in the next chapter.

Chapter 7

Conclusion

This thesis considered the problem of state construction in reinforcement learning, that is the problem of designing agents with the ability to construct the state directly from the sensorimotor data stream. Our objective was to understand how well existing solution methods perform and to improve them. We took a step toward this objective through three contributions. The first two contributions focused on the first part of the objective, whihc is to understand existing solution methods. The third contribution included a new algorithmic idea for improving upon existing solution methods.

The first contribution of the thesis provided a tool for assessing state construction methods. The tool proposed for assessing state construction methods consists of a suite of diagnostic benchmarks. We believe that the proposed benchmarks will help make progress on the problem of state construction by facilitating the development and evaluation of new ideas. Moreover, their low computational demand and configurability make them appealing for careful systematic studies. In our experimental results in the second contribution, we found the proposed benchmarks effective for isolating and investigating fundamental challenges in state construction. The broader community has started to adopt the benchmarks for testing new ideas (Elelimy 2023; Javed et al., 2023; Samani and Sutton 2021).

The second contribution of the thesis benchmarked prominent solution methods for state construction. This part did not involve any major new algorithmic idea. Instead, it provided a comprehensive examination of existing

solution methods. To benchmark prominent solution methods, we systematically studied them on the proposed benchmarks with a focus on recurrent learning methods. Our empirical results revealed some of the limitations of the existing solution methods, as discussed in Chapter 5, suggesting that recurrent architectures trained with backpropagation are highly sensitive to the truncation parameter requiring much more computation and memory for learning longer temporal dependencies. This could potentially make recurrent architectures infeasible for learning long temporal associations, although they have infinite memory in principle. We were able to reduce the sensitivity of recurrent architectures to the truncation parameters using the simple idea of feeding stimulating traces to RNNs. Our empirical results also suggested that transformers can effectively learn temporal associations; however, their computational requirements increase significantly with the length of the temporal association.

The third contribution of the thesis improved upon existing solution methods by introducing a new algorithmic idea for auxiliary task discovery. The new algorithmic idea is a method for discovering auxiliary tasks that would assist state construction. It is based on the idea of generate-and-test where auxiliary tasks are continually generated, evaluated, and replaced if recognized to be useless. We showed the efficacy of the proposed discovery method empirically using fully observable problems.

We did not try the proposed auxiliary task discovery method on the proposed benchmarks because we wanted to focus on the simpler case of full observability. However, the proposed discovery method could be potentially tested on the trace-patterning benchmark where there are multiple signals to make predictions about.

7.1 Future Directions

This section presents a few interesting directions to pursue in the future. The first future direction is to further investigate RNNs augmented with stimulating traces proposed in Chapter 5. The focus of Chapter 5 was to study

existing solution methods for state construction. However, as a minor contribution, we also proposed the new algorithmic idea of augmenting RNNs with stimulating traces. In our experiments on the trace-conditioning and trace-patterning benchmarks, we observed that augmenting RNNs with stimulating traces improved their performance and reduced their sensitivity to the truncation parameter. An interesting future direction is to investigate whether RNNs augmented with simulating traces would be applicable in more complex partially observable problems.

A second future direction is to propose additional benchmarks building up on the benchmarks proposed in this thesis. For example, we can extend the proposed benchmark to the case of non-stationarity where solution methods would require to do some form of tracking. This can be done by varying the ISI in trace conditioning and trace patterning or changing the activation patterns in noisy patterning and trace patterning.

The third future direction, which is the logical next step to the proposal of non-stationary benchmarks, is to study existing solution methods on them. As discussed in Section 4, training large neural networks with backpropagation, which is the prevailing approach to state construction, is not suitable for the case of continual learning due to the problem of loss of plasticity. It would be interesting to investigate the problem of loss of plasticity on the benchmarks that include non-stationarity.

The fourth future direction is to investigate the feature-attainment generator proposed for auxiliary task discovery in Chapter 6 more thoroughly. The results presented in Chapter 6 demonstrated that the feature attainment generator can find useful auxiliary tasks. The next step is to try it in environments where random generation would be infeasible such as pixel-based environments. We could also explore reward-respecting feature attainment subtasks (Sutton et al., 2022), that further promote auxiliary tasks that are useful for the main task.

The fifth idea for extending the work done in this thesis in a new direction is to design a new generator for the generate-and-test method. The generator constitutes an important part of the generate-and-test method. An idea to

explore for designing the generator is to sample subgoals from what the agent has already experienced and is stored in the replay buffer. To sample from the replay buffer, higher priority could be given to subgoals associated with higher temporal difference errors. The intuition behind this choice is that subgoals associated with higher temporal difference errors might be more interesting to learn about. This idea is similar to the idea of hindsight experience replay (Andrychowicz et al., 2017).

The sixth future direction is to explore new ideas for the tester in the generate-and-test discovery method. The tester proposed in this thesis evaluates the auxiliary tasks based on how useful the features induced by them are for the main task. To compute the utility of the features, we considered the weight magnitude and trace of activation of the feature. There are many other metrics in the representation learning literature for evaluating features. One example is the dropout pruner proposed by Shah (2023). The dropout pruner randomly sets some of the weights of the network to zero and measures how that affects the performance. This idea could be applied to features as well. Another metric for feature evaluation is proposed by Elsayed and Mahmood (2023). One future direction is to incorporate these other feature evaluation metrics into the proposed auxiliary task discovery method.

The seventh future direction is to remove the requirement of using the master-user architecture in the proposed generate-and-test method. We proposed to use the master-user architecture in order to identify which auxiliary task was responsible for inducing each feature. However, the agent may be able to do credit assignment in a smarter way. For example, it could use the gradient backpropagated from each auxiliary task to each feature to recognize which auxiliary task gets credit for inducing which feature. Removing this constraint would increase the applicability of the generate-and-test method significantly.

The eighth future direction is to investigate the sensitivity of the proposed generate-and-test method to its hyper-parameters. The proposed generate-and-test method introduced a set of new hyper-parameters, such as the replacement ratio and replacement time. Systematic studies are required to

study the method's sensitivity to these hyper-parameters and to explore ideas for improving it in case it requires careful tuning of these hyper-parameters.

The ninth direction to pursue is to use the generate-and-test idea for option discovery. The idea of generate-and-test can be applied to discovering options or temporally extended actions (Sutton, Precup, and Singh, 1999). To develop a generate-and-test method for option discovery, we require a generator and a tester. The tester requires a measure of option usefulness. One such measure could be how much the discovered options would help exploration. The generator could generate options using the trajectories in the replay buffer, giving a higher probability to trajectories that resulted in higher temporal-difference error, similar to the idea of the hindsight experience replay (Andrychowicz et al., 2017).

The last idea for future work is to combine meta-learning approaches with the proposed generate-and-test method. The meta-learning approaches can be slow due to their computational demand. However, they can be run in the background to refine the auxiliary tasks discovered by generate-and-test. It would be interesting to test the combination of meta-learning with generate-and-test in the three tested environments and see if their combination will result in discovering good auxiliary tasks, outperforming both approaches in isolation.

7.2 Closing

The problem of state construction is an old problem with a long history in control theory, robotics, and machine learning. While a lot of progress has been made on state construction, there are still fundamental problems that need to be addressed. We studied some of these issues in this thesis such as the computational complexity of existing solution methods which grows linearly or quadratically with the length of temporal associations that they want to learn. we also alluded to some of the fundamental issues like the problem of loss of plasticity in continual learning settings.

On the problem of auxiliary task discovery, we only laid out the territory.

Auxiliary task discovery is certainly a challenging problem with only a few existing solution methods. We believe that our proposed method is a step towards designing agents with the ability to pose subproblems for themselves. The field has only started exploring solution methods for the problem of discovery and our proposed method is by no means the conclusive solution.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, P., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, 30.

Antoniou, A., Edwards, H., and Storkey, A. (2018). How to train your MAML. *arXiv preprint arXiv:1810.09502*.

Beattie C., Leibo J. Z., Teplyashin D., Ward T., Wainwright M., Küttler H., Lefrancq A, Green S, Valdés V., Sadik A., Schrittwieser J., Anderson K., York S., Cant M., Cain A., Bolton A., Gaffney S., King H., Hassabis D., Legg S. and Petersen S. (2016) Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bellemare M. G., Naddaf Y., Veness J. and Bowling M. (2013) The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47: 253–279.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. (2016). Model-free episodic control. *arXiv preprint arXiv:1606.04460*.

Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J. and Zaremba W. (2016) Openai gym. *arXiv preprint arXiv:1606.01540*.

Chen L., Lu K., Rajeswaran A., Lee K., Grover A., Laskin M., Abbeel P., Srinivas A. and Mordatch I. (2021) Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34.

Cho K., Van Merrienboer B., Bahdanau D. and Bengio Y. (2014) On the prop-

erties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*

Colas C., Sigaud O. and Oudeyer P. (2018) How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295* .

Dabney, W., Barreto, A., Rowland, M., Dadashi, R., Quan, J., Bellemare, M. G., and Silver, D. (2021, May). The value-improvement path: Towards better representations for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 8, pp. 7160-7168).

Dickinson A. (1980) *Contemporary animal learning theory*, volume 1. CUP Archive.

Dohare, S., Hernandez-Garcia, J., Rahman, P., Sutton, R., and Mahmood, A. R. (2023). Loss of plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Farajtabar, M., Pascanu, R., and Lakshminarayanan, B. (2018). Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224.*

Elelimy, E. M. (2023). *Real Time Recurrent Learning with Complex-Valued Trace Units*. M.Sc. Thesis, University of Alberta, Edmonton.

Elman J. L. (1990) Finding structure in time. *Cognitive science* 14(2): 179–211.

Elsayed, M., and Mahmood, A. R. (2023). Utility-based Perturbed Gradient Descent: An Optimizer for Continual Learning. *arXiv preprint arXiv:2302.03281.*

Engstrom L., Ilyas A., Santurkar S., Tsipras D., Janoos F., Rudolph L. and Madry A. (2019) Implementation matters in deep rl: A case study on ppo and trpo. In: *International Conference on Learning Representations*.

Espeholt L., Soyer H., Munos R., Simonyan K., Mnih V., Ward T., Doron Y., Firoiu V., Harley T., Dunning I., Legg S. and Kavukcuoglu K. (2018) IMPALA: Scalable distributed deep- RL with importance weighted actor-learner architectures. In: *Proceedings of the 35th International Conference*

on Machine Learning, Proceedings of Machine Learning Research, volume 80. Stockholm Sweden: PMLR, pp. 1407–1416.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., and Singh, S. (2021). Bootstrapped meta-learning. *arXiv preprint arXiv:2109.04504*.

Fortunato M., Tan M., Faulkner R., Hansen S., Badia A. P., Buttimore G., Deck C., Leibo J. Z. and Blundell C. (2019) Generalization of reinforcement learners with working and episodic memory. In: *Advances in Neural Information Processing Systems*. pp. 12469–12478.

Frankle, J., and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635.*

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4), 128-135.

Gehring J, Auli M., Grangier D., Yarats D. and Dauphin Y.N. (2017) Convolutional sequence to sequence learning. In: *International Conference on Machine Learning.* PMLR, pp. 1243–1252.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401.*

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J. and Badia, A.P., Hermann, K.M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626), 471-476.

Harris J. A., Livesey E. J., Gharaei S. and Westbrook R. F. (2008) Negative patterning is easier than a biconditional discrimination. *Journal of Experimental Psychology: Animal Behavior Processes* 34(4): 494.

He, T., Zhang, Y., Ren, K., Liu, M., Wang, C., Zhang, W., Yang Y., and Li, D. (2022). Reinforcement learning with automated auxiliary loss search. *Advances in Neural Information Processing Systems*, 35, 1820-1834.

Henderson P., Islam R., Bachman P., Pineau J., Precup D. and Meger D. (2018) Deep reinforcement learning that matters. In: *Proceedings of the Association for the Advancement of Artificial Intelligence,* volume 32.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and Van Hasselt, H. (2019, July). Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 3796-3803).

Hochreiter S. and Schmidhuber J. (1997) Long short-term memory. *Neural Computation* 9(8): 1735–1780.

Howard M. W. and Eichenbaum H. (2013) The hippocampus, time, and memory across scales. *Journal of Experimental Psychology: General* 142(4): 1211.

Hull C.L. (1939) The problem of stimulus equivalence in behavior theory. *Psychological Review* 46(1): 9.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural computation*, 12(6), 1371-1398.

Jaeger H. (2001) The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *German National Research Center for Information Technology GMD Technical Report* 148(34): 13.

Janner M., Li Q. and Levine S. (2021) Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems* 34.

Javed, K., Shah, H., Sutton, R. S., and White, M. (2023). Scalable Real-Time Recurrent Learning Using Columnar-Constructive Networks. *Journal of Machine Learning Research*, 24, 1-34.

Kingma D.P. and Ba J. (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Konidaris, G., and Barto, A. (2009). Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems, 22*.

Lin, X., Baweja, H., Kantor, G., and Held, D. (2019). Adaptive auxiliary task weighting for reinforcement learning. *Advances in neural information processing systems, 32.*

Littman, M., Sutton, R. S., and Singh S. (2001). Predictive representations of state. *Advances in neural information processing systems*, 14.

Lo, C., Mihucz, G., White, A., Aminmansour, F., and White, M. (2022). Goal-space planning with subgoal models. *arXiv preprint arXiv:2206.02902.*

Loynd R., Fernandez R., Celikyilmaz A., Swaminathan A. and Hausknecht M. (2020) Working memory graphs. In: *International Conference on Machine Learning*. PMLR, pp. 6404–6414.

Ludvig E. A., Sutton R. S. and Kehoe E. J. (2008) Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural computation* 20(12): 3034–3054.

Ludvig E. A., Sutton R. S., Verbeek E. and Kehoe E. J. (2009) A computational model of hippocampal function in trace conditioning. In: *Advances in Neural Information Processing Systems*. pp. 993–1000.

Ludvig, E. A., Sutton, R. S., and Kehoe, E. J. (2012). Evaluating the TD model of classical conditioning. *Learning & behavior, 40(3)*, 305-319.

Ma C. (2020) *An Exploration of Predictive Representations of State.* M.Sc. Thesis, University of Alberta, Edmonton.

Machado M. C., Bellemare M. G., Talvitie E., Veness J., Hausknecht M. and Bowling M. (2018) Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61: 523–562.

Machado, M. C., Barreto, A., Precup, D., and Bowling, M. (2023). Temporal abstraction in reinforcement learning with the successor representation. *Journal of Machine Learning Research*, 24(80), 1-69.

Mackintosh N. J. (1974) *The Psychology of Animal Learning.* Academic Press.

Mahmood, A. R., and Sutton, R. S. (2013). Representation Search through Generate and Test. In *AAAI* Workshop: Learning Rich Representations from Low-Level Sensors (Vol. 10, No. 2908225.2908228).

Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., Newton, J., Parzen, E., and Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of forecasting*, 1(2), 111-153.

McCloskey, M., and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation* (Vol. 24, pp. 109-165). Academic Press.

McGovern, A., and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the International Conference on Machine Learning*.

Menick J., Elsen E. Evci U., Osindero S., Simonyan K. and Graves A. (2020) Practical real time recurrent learning with a sparse approximation. In: *International Conference on Learning Representations.*

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673.*

Modayil J., White A. and Sutton R. S. (2014) Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior* 22(2): 146–160.

Monahan, G. E. (1982). State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science, 28(1)*:1–16.

Nath S., Liu V., Chan A., Li X., White A. and White M. (2019) Training recurrent neural networks online by learning explicit state variables. In: *International Conference on Learning* Representations.

Osband I., Van Roy B., Russo D. J. and Wen Z. (2019) Deep exploration via randomized value functions. *Journal of Machine Learning Research* 20(124): 1–62.

Parisotto E., Song F., Rae J., Pascanu R., Gulcehre C., Jayakumar S., Jaderberg M., Kaufman R. L., Clark A., Noury S., Botvinick M., Heess N. and Hadsell R. (2020) Stabilizing transformers for reinforcement learning. In: *Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research,* volume 119. PMLR, pp. 7487–7498.

Parisotto E. and Salakhutdinov R. (2021) Efficient transformers in reinforcement learning using actor-learner distillation. In: *International Conference on Learning Representations*.

Pavlov I.P. (1927) *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*, volume 3. london: oxford University Press.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017, July). Neural episodic control. In *International conference on machine learning* (pp. 2827-2836). PMLR.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *arXiv preprint arXiv:1801.06146*.

Rafiee, B., Abbas, Z., Ghiassian, S., Kumaraswamy, R., Sutton, R. S., Ludvig, E. A., and White, A. (2022). From eye-blinks to state construction: Diagnostic benchmarks for online representation learning. *Adaptive behavior*, 31(1), 3-19.

Rafiee, B., Ghiassian, S., Jin, J., Sutton, R., Luo, J., and White, A. (2023, November). Auxiliary task discovery through generate-and-test. In *Conference on Lifelong Learning Agents* (pp. 703-714). PMLR.

Ramesh, R., Tomar, M., and Ravindran, B. (2019). Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731.*

Robinson, A. J., and Fallside, F. (1987). *The utility driven dynamic error propagation network* (Vol. 1). Cambridge: University of Cambridge Department of Engineering.

Rosencrantz, M., Gordon, G., and Thrun, S. (2004, July). Learning low dimensional predictive representations. In *Proceedings of the twenty-first international conference on Machine learning* (p. 88).

Rudary, M., and Singh, S. (2003). A nonlinear predictive state representation. *Advances in neural information processing systems*, 16.

Samani, A., and Sutton, R. S. (2021). Learning agent state online with recurrent generate-and-test. *arXiv preprint arXiv:2112.15236.*

Schlegel, M., Jacobsen, A., Abbas, Z., Patterson, A., White, A., and White, M. (2021). General value function networks. *Journal of Artificial Intelligence Research*, 70, 497-543.

Schneiderman N. (1966) Interstimulus interval function of the nictitating membrane response of the rabbit under delay versus trace conditioning. *Journal of Comparative and Physiological Psychology* 62(3): 397.

Shah, H. (2023). *Greedy Pruning for Continually Adapting Networks.* M.Sc. Thesis, University of Alberta, Edmonton.

Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. (2016). Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307.*

Sutton R. S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning* 3(1): 9–44.

Sutton R. S. and Barto A. G. (1990) Time-derivative models of pavlovian reinforcement. In M. Gabriel and J. Moore (Eds.). *Learning and Computational Neuroscience: Foundations of Adaptive Networks* : 497–537.

Sutton, R. S., Whitehead, S. D., (1993). Online learning with random representations. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 181-211.

Sutton, R. S., Rafols, E., and Koop, A. (2006). Temporal abstraction in temporal-difference networks. In *Advances in neural information processing systems*, pp. 1313–1320.

Sutton R. S., Koop A. and Silver D. (2007) On the role of tracking in stationary environments. In: *Proceedings of the 24th International Conference on Machine Learning*. pp. 871–878.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011, May). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2* (pp. 761-768).

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Sutton, R. S., Machado, M. C., Holland, G. Z., Szepesvari, D., Timbers, F., Tanner, B., and White, A. (2023). Reward-respecting subtasks for model-based reinforcement learning. *Artificial Intelligence*, 324, 104001.

Tallec C. and Ollivier Y. (2018) Unbiased online recurrent optimization. In: *International Conference on Learning Representations*.

Thon, M. (2017). *Spectral Learning of Sequential Systems* (Doctoral dissertation, Jacobs University Bremen).

Todorov E., Erez T. and Tassa Y. (2012) Mujoco: A physics engine for model-based control. In: 2012 *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026– 5033.

Tucker G., Bhupatiraju S., Gu S., Turner R., Ghahramani Z. and Levine S. (2018) The mirage of action-dependent baselines in reinforcement learning. In: *International Conference on Machine Learning*. PMLR, pp. 5015–5024.

van Hasselt, H., and Sutton, R. S. (2015). Learning to predict independent of span. *arXiv preprint arXiv:1508.04582*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H, Silver, D., and Singh, S. (2019). Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems,* 32.

Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H., Silver, D., and Singh, S. (2021). Discovery of options via meta-learned subgoals. *Advances in Neural Information Processing Systems*, 34, 29861-29873.

Wagner A.R. (1978) Expectancies and the priming of stm. *Cognitive Processes in Animal Behavior* : 177–209.

Wang, H., Miahi, E., White, M., Machado, M. C., Abbas, Z., Kumaraswamy, R., Liu, V., and White, A. (2022). Investigating the Properties of Neural Network Representations in Reinforcement Learning. *arXiv preprint arXiv:2203.15955.*

Watkins, C. J., and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3), 279-292.

Wayne G., Hung C., Amos D., Mirza M., Ahuja A., Grabska- Barwinska A., Rae J., Mirowski P., Leibo J. Z., Santoro A., Gemici M., Reynolds M., Harley T., Abramson J., Mohamed S., Rezende D., Saxton D., Cain A., Hillier C., Silver D., Koray K., Botvinick M., Hassabis D. and Lillicrap

T. (2018) Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760* .

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4), 339-356.

Williams D. A., Todd T. P., Chubala C. M. and Ludvig E. A. (2017) Intertrial unconditioned stimuli differentially impact trace conditioning. *Learning & Behavior* 45(1): 49–61.

Williams R. J. and Zipser D. (1989) A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1(2): 270–280.

Williams R. J. and Peng J. (1990) An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation* 2(4): 490–501.

Xu, Z., van Hasselt, H. P., and Silver, D. (2018). Meta-gradient reinforcement learning. *Advances in neural information processing systems, 31.*