Journal of Artificial Intelligence Research 70 (2021) 497-543 Submitted 04/2020; published 01/2021

General Value Function Networks

Matthew Schlegel MKSCHLEG@UALBERTA.CA
Andrew Jacobsen AJJACOBS@UALBERTA.CA
Zaheer Abbas MZAHEER@UALBERTA.CA
Andrew Patterson AP3@UALBERTA.CA
Adam White AMW@UALBERTA.CA
Martha White WHITEM@UALBERTA.CA
Department of Computing Science and the Alberta Machine Intelligence Institute (Amii)
University of Alberta, Canada

Abstract

State construction is important for learning in partially observable environments. A general purpose strategy for state construction is to learn the state update using a Recurrent Neural Network (RNN), which updates the internal state using the current internal state and the most recent observation. This internal state provides a summary of the observed sequence, to facilitate accurate predictions and decision-making. At the same time, specifying and training RNNs is notoriously tricky, particularly as the common strategy to approximate gradients back in time, called truncated Back-prop Through Time (BPTT), can be sensitive to the truncation window. Further, domain-expertise—which can usually help constrain the function class and so improve trainability—can be difficult to incorporate into complex recurrent units used within RNNs. In this work, we explore how to use multi-step predictions to constrain the RNN and incorporate prior knowledge. In particular, we revisit the idea of using predictions to construct state and ask: does constraining (parts of) the state to consist of predictions about the future improve RNN trainability? We formulate a novel RNN architecture, called a General Value Function Network (GVFN), where each internal state component corresponds to a prediction about the future represented as a value function. We first provide an objective for optimizing GVFNs, and derive several algorithms to optimize this objective. We then show that GVFNs are more robust to the truncation level, in many cases only requiring one-step gradient updates.

1. Introduction

Most domains of interest are partially observable, where an agent only observes a limited part of the state. In such a setting, if the agent uses only the immediate observations, then it has insufficient information to make accurate predictions or decisions. A natural approach to overcome partial observability is for the agent to maintain a history of its interaction with the world. For example, consider an agent in a large and empty room with low-powered sensors that reach only a few meters. In the middle of the room, with just the immediate sensor readings, the agent cannot know how far it is from a wall. Once the agent reaches a wall, though, it can determine its distance from the wall in the future by remembering this interaction. This simple strategy, however, can be problematic if a long history length is needed (McCallum, 1996).

©2021 AI Access Foundation. All rights reserved.

State construction enables the agent to overcome partial observability, with a more compact representation than an explicit history. Because most environments and datasets are partially observable—in time series prediction, in modeling dynamical systems and in reinforcement learning—there is a large literature on state construction. These strategies can be separated into Objective-state and Subjective-state approaches.

Objective-state approaches specify a true latent space, and use observations to identify this latent state. An objective representation is one that is defined in human-terms, external to the agent's data-stream of interaction. They typically require an expert to provide feature generators or models of the agent's motion and sensor apparatus. Many approaches are designed for a discrete set of latent states, including HMMs (Baum & Petrie, 1966) and POMDPs (Kaelbling, Littman, & Cassandra, 1998). A classical example is Simultaneous Localization and Mapping, where the agent attempts to extract its position and orientation as a part of the state (Durrant-Whyte & Bailey, 2006). These methods are particularly useful in applications where the dynamics are well-understood or provided, and so accurate transitions can be used in the explicit models. When models need to be estimated or the latent space is unknown, however, these methods either cannot be applied or are prone to misspecification.

The goal of subjective-state approaches, on the other hand, is to construct an internal state only from a stream of experience. This contrasts objective-state approaches in two key ways. First, the agent is not provided with a true latent space to identify. Second, the agent need not identify a true latent state, even if there is one. Rather, it only needs to identify an internal state that is sufficient for making predictions about target variables of interest. Such a state will likely not correspond to objective quantities like meters and angles, but could be much simpler than the true latent state and can be readily learned from the data stream. Examples of subjective-state approaches to state construction include Recurrent Neural Networks (RNNs) (Hopfield, 1982; Lin & Mitchell, 1993), Predictive State Representations (PSRs) (Littman, Sutton, & Singh, 2001) and TD Networks (Sutton & Tanner, 2004).

RNNs have emerged as one of the most popular approaches for online state construction, due to their generality and the ability to leverage advances in optimizing neural networks. An RNN provides a recurrent state-update function, where the state is updated as a function of the (learned) state on the previous step and the current observations. These recurrent connections can be unrolled back in time, making it possible for the current RNN state to be dependent on observations far back in time. There have been several specialized activation units crafted to improve learning long-term dependencies, including long short-term memory units (LSTMs) (Hochreiter & Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho, van Merrienboer, Bahdanau, & Bengio, 2014). PSRs and TD Networks are not as widely used, because they make use of complex training algorithms that do not work well in practice (see McCracken & Bowling, 2005; Boots, Siddiqi, & Gordon, 2011 and Vigorito, 2009; Silver, 2012 respectively). In fact, recent work has investigated facilitating use of these models by combining them with RNNs (Downey, Hefny, Boots, Gordon, & Li, 2017; Choromanski, Downey, & Boots, 2018; Venkatraman, Rhinehart, Sun, Pinto, Hebert, Boots, Kitani, & Bagnell, 2017). Other subjective state approaches based on filtering can be complicated to extend to nonlinear dynamics, such as system identification approaches (Ljung, 2010) or Predictive Linear Gaussian models (Rudary, Singh, & Wingate, 2005; Wingate & Singh, 2006).

One issue with RNNs, however, is that training can be unstable and expensive. There are two well-known approaches to training RNNs. The first, Real Time Recurrent Learning (RTRL) (Williams & Zipser, 1989) relies on a recursive form to estimate gradients. This gradient computation is exact in the offline setting—when RNN parameters are fixed—but only an approximation when computing gradients online. RTRL is prohibitively expensive, requiring computation that is quartic in the hidden dimension size n. Low-rank approximations have been developed (Tallec & Ollivier, 2018; Mujika, Meier, & Steger, 2018; Benzing, Gauy, Mujika, Martinsson, & Steger, 2019) to improve computational efficiency, but these approaches to training RNNs remain less popular than the simpler strategy of Back propagation through time (BPTT).

BPTT explicitly computes gradients of the parameters, by using the chain rule back in time, essentially unrolling the recursive RNN computation. This approach requires maintaining the entire trajectory, which is infeasible for many online learning systems we consider here. A truncated form of BPTT (p-BPTT) is often used to reduce the complexity of training, where complexity grows linearly with p: $O(pn^2)$. Unfortunately, training can be highly sensitive to the truncation parameters (Pascanu, Mikolov, & Bengio, 2013), particularly if the dependencies back-in-time are longer than the chosen p—as we reaffirm in our experiments.

One potential cause of this instability is precisely the generality of RNNs. These systems require expertise in selecting architectures and tuning hyperparameters (Pascanu et al., 2013; Sutskever, 2013). This design space can already be difficult to navigate with standard feed-forward neural networks, and is exacerbated by the recurrence that makes the learning dynamics more unstable. Further, it can be hard to leverage domain expertise to constrain the space of RNNs, and so improve trainability. Specialized, complex architectures have been designed for speech recognition (Saon, Kurata, Sercu, Audhkhasi, Thomas, Dimitriadis, Cui, Ramabhadran, Picheny, Lim, et al., 2017) and NLP (Peters, Neumann, Iyyer, Gardner, Clark, Lee, & Zettlemoyer, 2018); redesigning such systems for new problems is an onerous task. Many general purpose architectural restrictions have been proposed, such as GRUs and skip connections (see Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2017 and Trinh, Dai, Luong, & Le, 2018 for thorough overviews). These methods all provide tools to design, and tune, better architectures, but still do not provide a simple mechanism for a non-expert in deep learning to inject prior knowledge.

An alternative direction, that requires more domain expertise than RNN expertise, is to use predictions as auxiliary losses. Auxiliary unsupervised losses have been used in NLP to improve trainability (Trinh et al., 2018). Less directly, auxiliary losses were used in reinforcement learning (Jaderberg, Mnih, Czarnecki, Schaul, Leibo, Silver, & Kavukcuoglu, 2017) and for modeling dynamical systems (Venkatraman et al., 2017), to improve the quality of the representation; this is a slightly different but nonetheless related goal to trainability. The use of predictions for auxiliary losses is an elegant way to constrain the RNN, because the system designers are likely to have some understanding of the relevant system components to predict. For the larger goals of AI, augmenting the RNN with additional predictions is promising because one could imagine the agent discovering these predictions autonomously—predictions by design are grounded in the data stream and learnable without human supervision. Nonetheless, the use of predictions as auxiliary tasks

provides a more indirect (second-order) mechanism to influence the state variables. In this work, we ask: is there utility in directly constraining states to be predictions?

To answer this question, we need a practical approach for learning RNNs, where the internal state corresponds to predictions. We propose a new RNN architecture, where we constrain the hidden state to be multi-step predictions, using an explicit loss function on the hidden state. In particular, we use general policy-contingent, multi-step predictions—called General Value Functions (GVFs) (Sutton, Modayil, Delp, Degris, Pilarski, White, & Precup, 2011)—generalizing the types of predictions considered in related predictive representation architectures (Rafols, Ring, Sutton, & Tanner, 2005; Silver, 2012; Sun, Venkatraman, Boots, & Bagnell, 2016; Downey et al., 2017). These GVFs have been shown to represent a wide array of multi-step predictions (Modayil, White, & Sutton, 2014). In this paper, we develop the objective and algorithm(s) to train these GVF networks (GVFNs).

We then demonstrate through a series of experiments that GVFNs can effectively represent the state and are much more robust to train, allowing even simple gradient updates with no gradients needed back-in-time. We first investigate accuracy on two time series datasets, and find that our approach is competitive with a baseline RNN and more robust to BPTT truncation length. We then investigate GVFNs more deeply in several synthetic problems, to determine 1) if robustness to truncation remains for a domain with long-term dependencies and 2) the impact of the prediction specification—or misspecification—on GVFN performance. We find that GVFNs have consistent robustness properties across problems, but that, unsurprisingly, the choice of predictions do matter, both for improving learning as well as final accuracy. Our experiments provide evidence that constraining states to be predictions can be effective, and raise the importance of better understanding what these predictions should be.

Our work provides additional support for the *predictive representation hypothesis*, that state-components restricted to be predictions about the future result in good generalization (Rafols et al., 2005). Constraining the state to be predictions could both regularize learning—by reducing the hypothesis space for state construction—and prevent the constructed state from overfitting to the observed data and target predictions. To date, there has only been limited investigation into and evidence for this hypothesis. Rafols et al. (2005) showed that, for a discrete state setting, learning was more sample efficient with a predictive representation than a tabular state representation and a tabular history representation. Schaul and Ring (2013) showed how a collection of optimal GVFs—learned offline—provide a better state representation for a reward maximizing task, than a collection of optimal PSR predictions. Sun et al. (2016) showed that, for dynamical systems, constraining state to be predictions about the future significantly improved convergence rates over auto-regressive models and n4sid. Our experiments show that RNNs with state composed of GVF predictions can have notable advantage over RNNs in building state with p-BPTT, even when the RNN is augmented with auxiliary tasks based on those same GVFs.

2. Problem Formulation

We consider a partially observable setting, where the observations are a function of an unknown, unobserved underlying state. The dynamics are specified by transition probabilities $\mathrm{P} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ with state space $\mathcal{S}$ and action-space $\mathcal{A}$. On each time step the

agent receives an observation vector $\mathbf{o}_t \in \mathcal{O} \subset \mathbb{R}^m$, as a function $\mathbf{o}_t = \mathbf{o}(z_t)$ of the underlying state $z_t \in \mathcal{S}$. The agent only observes $\mathbf{o}_t$, not z_t, and then takes an action a_t, producing a sequence of observations and actions: $\mathbf{o}_0, a_0, \mathbf{o}_1, a_1, \ldots$.

The goal for the agent under partial observability is to identify a state representation $\mathbf{s}_t \in \mathbb{R}^n$ which is a sufficient statistic (summary) of past interaction, for targets y_t. More precisely, such a *sufficient state* ensures that y_t given this state is independent of history $\mathbf{h}_t = \mathbf{o}_0, a_0, \mathbf{o}_1, a_1, \ldots, \mathbf{o}_{t-1}, a_{t-1}, \mathbf{o}_t$,

$$p(y_t|\mathbf{s}_t) = p(y_t|\mathbf{s}_t, \mathbf{h}_t) \tag{1}$$

or so that statistics about the target are independent of history, such as $\mathbb{E}[Y_t|\mathbf{s}_t] = \mathbb{E}[Y_t|\mathbf{s}_t, \mathbf{h}_t]$. Such a state summarizes the history, removing the need to store the entire (potentially infinite) history. Note here that this is a less stringent definition of sufficient state than used for PSRs (Littman et al., 2001), where the state is constructed for predictions about all future outcomes. We presume that the agent has a limited set of targets of interest, and needs to find a sufficient state for just those targets. For example, a potential set of targets is the observation vector on the next time step.

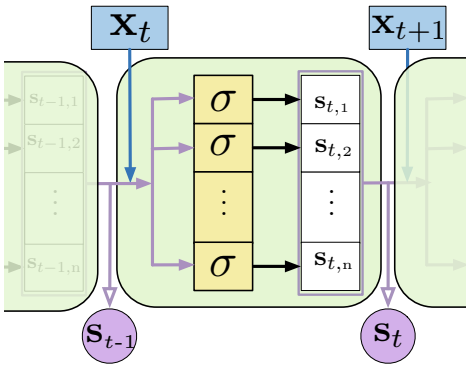
One strategy for learning such a state is with *recurrent neural networks* (RNNs), which learn a state-update function. Imagine a setting where the agent has a sufficient state $\mathbf{s}_t$ for this step. To obtain sufficient state for the next step, it simply needs to update $\mathbf{s}_t$ with the new information in the given observation and action $\mathbf{x}_{t+1} = [a_t, \mathbf{o}_{t+1}] \in \mathbb{R}^d$. The goal, therefore, is to learn a state-update function $f : \mathbb{R}^{n+d} \to \mathbb{R}^n$ such that

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{x}_{t+1}) \tag{2}$$

provides a sufficient state $\mathbf{s}_{t+1}$. The update function f is parameterized by a weight vector $\boldsymbol{\theta} \in \Theta$ in some parameter space Θ. An example of a simple RNN update function, for $\boldsymbol{\theta}$ composed of stacked vectors $\boldsymbol{\theta}^{(j)} \in \mathbb{R}^{n+d}$ for each hidden state $j \in \{1, \ldots, n\}$ is, for activation function $\sigma : \mathbb{R} \to \mathbb{R}$,

$$\mathbf{s}_{t+1} = \begin{bmatrix} \sigma\left(\begin{bmatrix} \mathbf{s}_t \\ \mathbf{x}_{t+1} \end{bmatrix}^\top \boldsymbol{\theta}^{(1)}\right) \\ \vdots \\ \sigma\left(\begin{bmatrix} \mathbf{s}_t \\ \mathbf{x}_{t+1} \end{bmatrix}^\top \boldsymbol{\theta}^{(n)}\right) \end{bmatrix}$$

depicted as



In many cases, learning a sufficient state under function approximation may not be possible. Instead, this state is approximated so as to improve prediction accuracy of the target y_t.

The goal in this work is to develop an efficient algorithm to learn this state-update function. Most RNN algorithms learn this state-update by minimizing prediction error to desired targets $y_t \in \mathbb{R}$ across time. For example, for $\hat{y}_t = \mathbf{s}_t^\top \mathbf{w}$ for weights $\mathbf{w} \in \mathbb{R}^n$, the loss

for $\boldsymbol{\theta}$ on time step t could be

$$\ell(\hat{y}_t, y_t) \stackrel{\text{def}}{=} (\hat{y}_t - y_t)^2 = (\mathbf{s}_t^\top \mathbf{w} - y_t)^2$$
$$= (f_{\boldsymbol{\theta}}(\mathbf{s}_{t-1}, \mathbf{x}_t)^\top \mathbf{w} - y_t)^2 = \Big(f_{\boldsymbol{\theta}}\big(f_{\boldsymbol{\theta}}(\mathbf{s}_{t-2}, \mathbf{x}_{t-1}), \mathbf{x}_t\big)^\top \mathbf{w} - y_t\Big)^2 = \dots$$

This objective, however, can be difficult to optimize. The weights $\boldsymbol{\theta}$ can influence the state variables far back in time, with small changes for early states resulting in big changes to the state many steps later. This sensitivity to the weights can result in both vanishing and exploding gradient problems (Pascanu et al., 2013). Even worse, the problem is under-constrained, particularly if there is a scalar target. The loss may encourage the weights to change the immediate state $\mathbf{s}_t$ quite a bit—just to reduce error for this single stochastic target—resulting in potentially destabilizing changes to the weights that influence states all the way back in time.

One strategy is to consider how to constrain the state variables, so as to avoid changes just due to the targets, and stochasticity in the targets. We pursue a strategy, inspired by predictive representations, where the state-update function is learned such that each hidden state is an accurate prediction about future outcomes, described in the next section.

3. Constraining State to be Predictions

Let us start in a simpler setting and explain how the hidden units could be trained to be n-horizon predictions about the future. Imagine you have a multi-dimensional time series of a power-plant, consisting of d sensory observations with the first sensor corresponding to water temperature. Your goal is to make a hidden node in your RNN predict the water temperature in 10 steps, because you think this feature is useful to make other predictions about the future.

This can be done simply by adding the following loss: $(\mathbf{s}_{t,1} - \mathbf{x}_{t+10,1})^2$. The combined loss $L_t(\boldsymbol{\theta})$ on time step t is

$$L_t(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \ell(\hat{y}_t, y_t) + (\mathbf{s}_{t,1} - \mathbf{x}_{t+10,1})^2 \tag{3}$$

where both $\hat{y}_t$ and $\mathbf{s}_t$ are implicitly functions of $\boldsymbol{\theta}$. This loss still encourages the RNN to find a hidden state $\mathbf{s}_t$ that predicts y_t well. There is likely a whole space of solutions that have similar accuracy for this prediction. The second loss constrains this search to pick a solution where the first state node is a prediction about an observation 10 steps into the future. This second term can be seen as a regularizer on the network, specifying a preference on the learned solution. In general, more than one state node—even all of $\mathbf{s}_t$—could be learned to be predictions about the future.

The difficulty in training such a state depends on the chosen targets. For example, long horizon targets—such as 100 steps rather than 10 steps into the future—can be high variance. Even if such a predictive feature could be useful, it may be difficult to learn accurately and could make the state-update less stable. Using n-horizon predictions also requires a delay in the update: the agent must wait 100 steps to see the target to update the state at time t.

We therefore propose to restrict ourselves to a class of prediction that have been shown to be more robust to these issues (van Hasselt & Sutton, 2015; Sutton et al., 2011; Modayil

et al., 2014). This class of predictions correspond to predictions of discounted cumulative sums of signals into the future, called General Value Functions (GVFs). We have algorithms to estimate these predictions online, without having to wait to see outcomes in the future. This property of GVFs is called *independence of span* (van Hasselt & Sutton, 2015), meaning learning can be achieved with computation and memory independent of the horizon. Such a property is doubly critical for predictions within an RNN, as it is more likely that we can actually learn these predictions sufficiently quickly to be usable as state. Further, there is some evidence that this class of predictions is sufficient for a broad range of predictions about the future (Sutton et al., 2011; Modayil et al., 2014; Momennejad & Howard, 2018; Banino, Barry, Uria, Blundell, Lillicrap, Mirowski, Pritzel, Chadwick, Degris, Modayil, et al., 2018; White, 2015; Pezzulo, 2008), and so the restriction to GVFs does not significantly limit representability. We therefore focus on developing an approach for this class of predictions within RNNs.

4. GVF Networks

In this section, we introduce GVF Networks, an RNN architecture where hidden states are constrained to predict policy-contingent, multi-step outcomes about the future. We first describe GVFs and the GVF Network (GVFN) architecture. In the following section, we develop the objective function and algorithms to learn GVFNs. There are several related predictive approaches, in particular TD Networks, that we discuss in Section 8, after introducing GVFNs.

We first need to extend the definition of GVFs (Sutton et al., 2011) to the partially observable setting, to use them within RNNs. The first step is to replace state with histories. We define $\mathcal{H}$ to be the minimal set of histories, that enables the Markov property for the distribution over next observation

$$\mathcal{H} = \left\{ \mathbf{h}_t = (\mathbf{o}_0, a_0, \ldots, \mathbf{o}_{t-1}, a_{t-1}, \mathbf{o}_t) \mid \begin{array}{l} \text{(Markov property) } \Pr(\mathbf{o}_{t+1}|\mathbf{h}_t, a_t) = \Pr(\mathbf{o}_{t+1}|\mathbf{o}_{-1}a_{-1}\mathbf{h}_t a_t), \\ \text{(Minimal history) } \Pr(\mathbf{o}_{t+1}|\mathbf{h}_t) \neq \Pr(\mathbf{o}_{t+1}|\mathbf{o}_1, a_1, \ldots, a_{t-1}, \mathbf{o}_t) \end{array} \right\} \quad (4)$$

A GVF question is a tuple (π, c, γ) composed of a policy $\pi : \mathcal{H} \times \mathcal{A} \to [0, \infty)$, cumulant $c : \mathcal{H} \times \mathcal{A} \times \mathcal{H} \to \mathbb{R}$ and continuation function[1] $\gamma : \mathcal{H} \times \mathcal{A} \times \mathcal{H} \to [0, 1]$, also called the discount. On time step t, the agent is in H_t, takes actions A_t, transitions to H_{t+1} and observes[2] cumulant C_{t+1} and continuation γ_{t+1}. The answer to a GVF question is defined as the value function, $V : \mathcal{H} \to \mathbb{R}$, which gives the expected, cumulative discounted cumulant from any history $\mathbf{h}_t \in \mathcal{H}$. The value function which can be defined recursively with a Bellman

1. The original GVF definition assumed the continuation was only a function of H_{t+1}. This was later extended to transition-based continuation (White, 2017), to better encompass episodic problems. Namely, it allows for different continuations based on the transition, such as if there is a sudden change from $\mathbf{h}_t$ to $\mathbf{h}_{t+1}$. We use this more general definition for this reason, and because the cumulant itself is already defined on the three tuple $(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})$.
2. Throughout this document, unbolded uppercase variables are random variables; lowercase variables are instances of that random variable; and bolded variables are vectors. When indexing into a vector on time step t, such as $\mathbf{h}_t$, we double subscript as $\mathbf{h}_{t,j}$ for the jth component of $\mathbf{h}_t$.

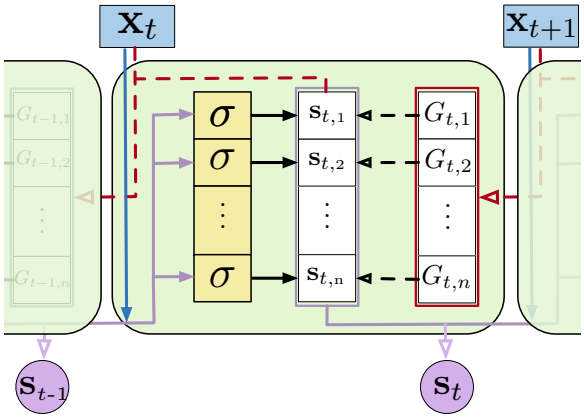


Figure 1: GVF Networks (GVFNs), where each state component $\mathbf{s}_{t,i}$ is updated towards the return $G_{t,i} \stackrel{\text{def}}{=} C^{(i)}_{t+1} + \gamma^{(i)}_{t+1}\mathbf{s}_{t+1,i}$ for the ith GVF. The solid forward arrows indicate how state is updated; in fact, the update is the same as a standard RNN. The difference is with the dotted lines, that indicate training. The dotted black arrows indicate the targets for the state components. The dotted red arrows indicate that the target $G_{t,i}$ are created using the observation and state on the next step.

equation as

$$V(\mathbf{h}_t) \stackrel{\text{def}}{=} \mathbb{E}[C_{t+1} + \gamma_{t+1}V(H_{t+1}) \mid H_t = \mathbf{h}_t, A_t \sim \pi(\cdot|\mathbf{h}_t)] \tag{5}$$
$$= \sum_{a_t \in \mathcal{A}} \pi(a_t|\mathbf{h}_t) \sum_{\mathbf{h}_{t+1} \in \mathcal{H}} \Pr(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t) \left[c(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}) + \gamma(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})V(\mathbf{h}_{t+1})\right].$$

The sums can be replaced with integrals if $\mathcal{A}$ or $\mathcal{O}$ are continuous sets. We assume that $\mathcal{H}$ is a finite set, for simplicity; the definitions and theory, however, can be extended to infinite and uncountable sets.

A GVFN is an RNN, and so is a state-update function f, but with the additional criteria that each element in $\mathbf{s}_t$ corresponds to a prediction—to a GVF. A GVFN is composed of n GVFs, with each hidden state component $\mathbf{s}_{t,j}$ trained such that at time step t, $\mathbf{s}_{t,j} \approx V^{(j)}(\mathbf{h}_t)$ for the jth GVF and history $\mathbf{h}_t$. Each hidden state component, therefore, is a prediction about a multi-step policy-contingent question. The hidden state is updated recurrently as $\mathbf{s}_t \stackrel{\text{def}}{=} f_{\boldsymbol{\theta}}(\mathbf{s}_{t-1}, \mathbf{x}_t)$ for a parametrized function $f_{\boldsymbol{\theta}}$, where $\mathbf{x}_t = [a_{t-1}, \mathbf{o}_t]$ and $f_{\boldsymbol{\theta}}$ is trained so that $\mathbf{s}_j \approx V^{(j)}(\mathbf{h}_t)$. This is summarized in Figure 1.

General value functions provide a rich language for encoding predictive knowledge. In their simplest form, GVFs with constant γ correspond to multi-timescale predictions referred to as Nexting predictions (Modayil et al., 2014). Allowing γ to change as a function of state or history, GVF predictions can combine finite-horizon prediction with predictions that terminate when specific outcomes are observed (Modayil et al., 2014).

To build some intuition, we provide some examples in Compass World. This environment is used in our experiments and depicted in Figure 2. Compass World is a grid world where the agent is only provided information about the color directly in front it. This world is partially observable, with all the tiles in the middle having a white observation, with the

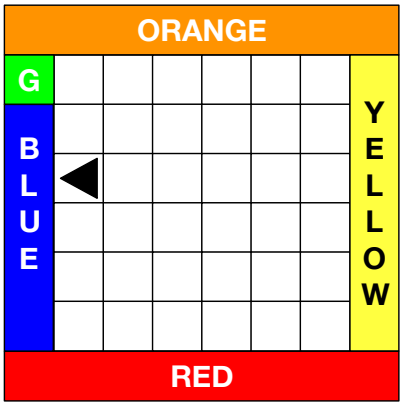


Figure 2: The Compass World: A partially observable grid world with observations of the color directly in front of the agent. **Actions:** The agent can take the actions Move Forward (one cell), Turn Left, and Turn Right. **Observations:** The agent observes the color of the grid cell it is facing. This means the agent can only observe a color if it is at the wall and facing outwards. The agent depicted as an arrow would see Blue. In the middle of the world, the agent sees White. **Goal:** The agent's goal is to make accurate predictions about which direction it is facing.

only distinguishing color information available to the agent at the walls. The actions taken by the agent are to move forward, turn left, or turn right.

In this environment, the agent might want to know if it is facing the red wall. This can be specified as a GVF question: "If I go forward until I hit a wall, what is the probability I will see red?". The policy is to always go forward. If the current observation is 'Red', then the cumulant is 1; otherwise it is zero. The continuation γ is 1 everywhere, except when the agent hits a wall and see a color; then it becomes zero. The sampled return from a state is 1.0 if the agent is facing the Red wall, because going forward will result in summing many zero plus a 1 right before termination. If the agent is not facing the Red wall, the return is 0, because the agent terminates when hitting the wall but only sees cumulants that are zero for the entire trajectory. Because the outcome is deterministic, the probabilities are 1 or 0.

The agent could also ask about how frequently it will see Red, within a horizon of about 10 steps. We can obtain an approximation to this question by using a constant continuation of $\gamma = 0.9$. The intuition for this comes from thinking of $1 - \gamma$ as a success probability for a geometric distribution: the probability of successfully terminating. The mean of this geometric distribution is $\frac{1}{1-\gamma}$—which in this case is $\frac{1}{1-0.9} = 10$—provides the expected number of steps until the first success. Recall that termination indicates that a return is cut-off, and so a cumulant is not included in the sum after termination. This probabilistic termination means that even if Red is seen after 10 steps, it will still be included in the return. However, it does indicate its contribution has been significantly decayed. This exponential prediction loses precision, and so the GVF only provides an approximation to this question.

The agent could also also ask if it will see Red, within a horizon of about 10 steps. In this case, the continuation would be 0.9 until the agent observed Red, at which point it would become zero (indicating termination). The GVF answer corresponds to a discounted probability of observing Red, with a smaller number if Red is observed further in the

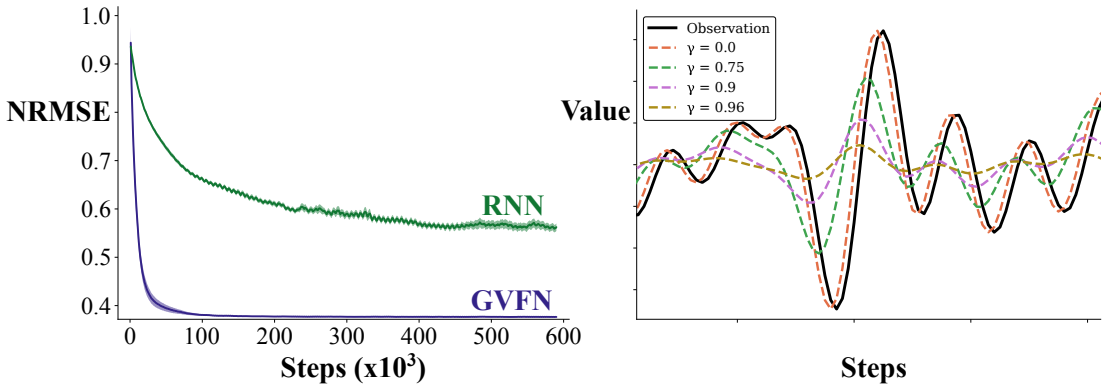


Figure 3: **(left)** Example learning curve for MSO with GVFNs and simple RNNs **(right)** The returns for different γ, corresponding to GVFs in the GVFN, for a small section of the MSO time series dataset. The dotted red line for $\gamma = 0$ looks overlayed with the time series plotted in black, but is actually the observation one step in the future.

future. If the agent always see Red in 1 step from $\mathbf{h}_t$, then it observes $C_{t+1} = 1$ and $\gamma_{t+1} = 0$ and the value is precisely 1. If the agent sees Red in 2 steps from $\mathbf{h}_t$, then $C_{t+1} = 0, \gamma_{t+1} = 0.9, C_{t+2} = 1$ and $\gamma_{t+2} = 0$ resulting in a value of 0.9. If the agent sees Red in 10 steps from $\mathbf{h}_t$, then the value is $0.9^9 \approx 0.4$. If just a few more steps into the future, say 15 steps, then the value would be 0.2. The magnitudes start to get quite low, indicating that it is less likely to observe Red in this window.

Notice that though we define the cumulants and continuation functions on the underlying (unknown) state $\mathbf{h}_t$, this is a generalization of defining it on the observations. The observations are a function of state; the cumulants and continuations γ that are defined on observations are therefore defined on $\mathbf{h}_t$. In the examples above, these functions were defined using just the observations. More generally, we consider them as part of a problem definition. This means they could be defined using short histories, or other separate summaries of the history. As we discuss in Section 6, we can also consider cumulants that are a function of our own predictions or constructed state.

A natural question is how these GVFs are chosen. This problem corresponds to the discovery problem for predictive representations. In this work, we first focus on the utility of this architecture, with simple heuristics or expert chosen GVFs. We briefly discuss simple ideas for discovery in Section 13, but leave a more systematic investigation of the discovery problem to future work.

5. A Case Study using GVFNs for Time Series Prediction

Before discussing the objective and training algorithms for GVFNs, we provide a simple demonstration of their use in a synthetic single-variable time series dataset to build intuition. GVFNs can be used for time series prediction by simply assuming that a fixed (unknown) policy generates the data. The GVFs within the network are assumed to have this same fixed unknown policy in common, but differ in the pair of continuation and cumulant functions. For a multi-variate time series, one GVF could have a cumulant corresponding to the first

entry of the observation on the next time step, and another GVF could use the second entry. Even for a single-variate time series, we can define meaningfully different cumulants. For example, one GVF could correspond to the probability that the observation becomes larger than 1. The cumulant would be zero until this event occurs, at which point it would be 1.

In Figure 3 (left) we provide a preliminary result using a GVFN to forecast 12-steps into the future on the single-variate Multiple Super-imposed Oscillator (MSO) time series dataset. We discuss the full empirical set-up in Section 9, and here simply provide some insights relevant to building intuition for how to use GVFNs.

The GVFN consists of a recurrent, constrained layer of 128 GVFS with γs spaced linearly in $[0.1, 0.97]$ to learn the state. To make predictions, we can additionally add feedforward layers from this recurrent layer; here we add a ReLu layer for additional nonlinearity in the prediction. For comparison, we also include a simple RNN, which similarily uses an additional ReLu layer after its recurrent layer. The prediction target is the observation 12 steps into the future. Both the RNN and GVFN have to wait 12 steps to see the accuracy of their prediction, delaying updates based on the target by 12 steps. The GVFN, however, can use the loss on the state at each step, and so more directly influence the value of states with the most recent observations. Both methods use p-BPTT, with truncation p. With a sufficiently high p, both perform well (see Section 9 for results with many p). We report the result here for $p = 1$, where the GVFN already obtains near-optimal performance.

It might be surprising that this simple GVFN, with GVFs only differing in continuation γ, can perform well. For time series data, however, such constant γ predictions provide anticipatory information about observations in the future. To see why, we plot the time series as well as returns for $\gamma \in \{0, 0.75, 0.9, 0.96\}$ as dotted lines in Figure 3 (right). These returns reflect the type of information that would be provided by a GVF prediction. At each time point t on the x-axis, we can see that the smaller γ, like $\gamma = 0.75$ as dotted green, anticipate the observations in a nearby window. If the time series is starting to rise in the near future, then the dotted green starts to rise right now. Returns can thus provide useful predictive information about increases and decreases that are expected to soon appear in the time series. Notice that the magnitude of the returns are approximately equal. For practical use, we want the magnitude of each GVF prediction to be similar, to avoid large differences in magnitude between state variables. With large γ, however, the return becomes large and so too does the value function. The standard fix to this is straightforward: each GVF uses a scaled cumulant of $(1 - \gamma)o_{t+1}$.

Notice, though, that there is a trade-off between anticipating a cumulant farther into the future and the precision of predictions about the future. Returns with lower continuations predict trends closer to when they occur in the dataset and have higher resolution. Returns with higher continuations anticipate changes further in the future, at the cost of smoothing over the detailed changes in the dataset. By using both lower and higher continuations, we hope to obtain the benefits of both. We further discuss this simple heuristic—GVFs with the same cumulant and varying γ—as a general purpose heuristic in Section 13.

6. The Objective for GVFNs

In this section, we introduce the objective function for GVFNs, that constrains the learned state to be GVF predictions. Each state component of a GVFN is a value function prediction,

and so is approximating the fixed point to a Bellman equation with history in Equation (5). The extension is not as simple as using a standard Bellman operator, however, because the GVFs are in a network. In fact, the Bellman equations are coupled in two ways: through composition—where one GVF can be the cumulant for another GVF as seen in section 11—and through the parametric recurrent state representation. We first discuss the Bellman network operator in Section 6.1, which extends the typical Bellman operator to allow for composition. We then explain how the coupling that arises from the recurrent state representation can be handled using a projected operator, and provide the objective for GVFNs, called the Mean-Squared Projected Bellman Network Error (MSPBNE), in Section 6.2. Then we discuss several algorithms to optimize this objective in Section 7.

The GVFN objective we introduce can be added to the standard RNN objective, to provide an RNN where the learned states are both useful for prediction of the target and encouraged—or regularized—to be GVF predictions. In this work, we only train GVFNs with the GVFN objective, without including the loss to a target, to focus the investigation on the utility of the proposed objective and on predictive features.

6.1 The Bellman Network Operator

To understand the Bellman network operator, it is useful to first revisit the Bellman operator for learning a single GVF. We assume the set of histories $\mathcal{H}$ is finite. Assume a tabular encoding for the values, $\mathbf{V}^{(j)} \in \mathbb{R}^{|\mathcal{H}|}$, for a GVF question $(\pi^{(j)}, c^{(j)}, \gamma^{(j)})$. The Bellman equation in 5 can be written as a fixed point equation, with Bellman operator

$$\mathbf{B}^{(j)}\mathbf{V}^{(j)} \stackrel{\text{def}}{=} \mathbf{C}^{(j)} + \mathbf{P}^{(j)}\mathbf{V}^{(j)} \tag{6}$$

where $\mathbf{C}^{(j)} \in \mathbb{R}^{|\mathcal{H}|}$ is the vector of expected cumulant values under $\pi^{(j)}$, with entries

$$\mathbf{C}^{(j)}(\mathbf{h}_t) \stackrel{\text{def}}{=} \sum_{a_t \in \mathcal{A}} \pi^{(j)}(a_t|\mathbf{h}_t) \sum_{\mathbf{h}_{t+1} \in \mathcal{H}} \Pr(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t) c^{(j)}(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}). \tag{7}$$

and $\mathbf{P}^{(j)} \in \mathbb{R}^{|\mathcal{H}| \times |\mathcal{H}|}$ is the matrix of values satisfying

$$\mathbf{P}^{(j)}(\mathbf{h}_t, \mathbf{h}_{t+1}) = \sum_{a_t \in \mathcal{A}} \pi^{(j)}(a_t|\mathbf{h}_t) \Pr(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t) \gamma^{(j)}(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}). \tag{8}$$

If the operator $\mathbf{B}^{(j)}$ is a contraction, then iteratively applying this operator converges to a fixed point. More precisely, if for any $\mathbf{V}_1^{(j)}, \mathbf{V}_2^{(j)} \in \mathbb{R}$, $\|\mathbf{B}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{B}^{(j)}\mathbf{V}_2^{(j)}\| < \|\mathbf{V}_1^{(j)} - \mathbf{V}_2^{(j)}\|$, then iteratively applying $\mathbf{B}^{(j)}$, as $\mathbf{V}_2^{(j)} = \mathbf{B}^{(j)}\mathbf{V}_1^{(j)}, \ldots, \mathbf{V}_{t+1}^{(j)} = \mathbf{B}^{(j)}\mathbf{V}_t^{(j)}, \ldots$, converges to a fixed point. Because temporal difference learning algorithms are based on this fixed-point update, the Bellman operator is central to the analysis of many algorithms for learning value functions, and is used in the definition of objectives for value estimation.

We can similarly define a Bellman operator that accounts for the relationships between GVFs in the network. Assume there are n GVFs, with $\mathbf{V} \in \mathbb{R}^{n|\mathcal{H}|}$ the stacked values for all the GVFs,

$$\mathbf{V} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{V}^{(1)} \\ \vdots \\ \mathbf{V}^{(n)} \end{bmatrix}. \tag{9}$$

The cumulants may now be functions of the values of other GVFs; we therefore explicitly write $\mathbf{C}_{\mathbf{V}}^{(j)}$. The Bellman network operator $\mathbf{B}$ is

$$\mathbf{B}\mathbf{V} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{C}_{\mathbf{V}}^{(1)} + \mathbf{P}^{(1)}\mathbf{V}^{(1)} \\ \vdots \\ \mathbf{C}_{\mathbf{V}}^{(n)} + \mathbf{P}^{(n)}\mathbf{V}^{(n)} \end{bmatrix}. \tag{10}$$

The Bellman network operator needs to be treated as a joint operator on all the GVFs because of compositional predictions, where the prediction on the next step of GVF j is the cumulant for GVF i. When iterating the Bellman operator $\mathbf{V}^{(j)}$ is not only involved in its own Bellman equation, but also in the Bellman equation for $\mathbf{V}^{(i)}$. Notice that if there were no compositions, the Bellman network operator would separate into individual Bellman operators, that operate on each $\mathbf{V}^{(j)}$ independently.

To use such a Bellman network operator, we need to ensure that iterating under this operator converges to a fixed point. For no composition, this result is straightforward, as it simply follows from previous results showing when the Bellman operator is a contraction. We state this explicitly below in Corollary 1. Under composition, we need to consider the effect of the current value function on the cumulant. Consequently, the operator may no longer be a simple linear projection of the values, followed by a sum of expected cumulants.

We first identify a necessary condition: the connections between GVFs must be acyclic. For example, GVF i cannot be a cumulant for GVF j, if j is already a cumulant for i. More generally, the connections between GVFs cannot create a cycle, such as $1 \to 2 \to 3 \to 1$. We provide a counterexample, where the Bellman network operator is not a contraction when there is a cycle, to illustrate that this condition is necessary.

We further place restrictions on the cumulant, if it is a function of other GVFs. In particular, we require that the cumulant is a Lipschitz function of the other value functions. Note that this restriction encompasses the setting for a non-compositional GVF, because the cumulant can be a constant w.r.t. these values. It also encompasses the setting we use in our experiments: that each cumulant is a linear function of the GVF values on the next step.

Assumption 1 (Acyclic Connections)**.** *The directed graph G is acyclic. G consists of n vertices, each corresponding to a GVF, and each directed edge (i, j) indicates that j is used in the cumulant for i.*

Assumption 2 (Lipschitz Compositional Cumulants)**.** *If GVF i has directed edges to $\{j_1, \ldots, j_k\}$, then the cumulant $c_{\mathbf{V}}^{(i)}(\mathbf{h}_{t+1})$ is Lipschitz in $\mathbf{V}^{(j_1)}, \ldots, \mathbf{V}^{(j_k)}$ with Lipschitz constant K_i. That is, for $\mathbf{V}_1, \mathbf{V}_2 \in \mathbb{R}^{n|\mathcal{H}|}$, $\|\mathbf{C}_{\mathbf{V}_1}^{(i)} - \mathbf{C}_{\mathbf{V}_2}^{(i)}\| \leq K_i \sum_{l=1}^{k} \|\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}\|$.*

Note that this assumption is satisfied if we assume that for some bounded weights $w_1, \ldots, w_k \in \mathbb{R}$, the cumulant must satisfy $c_{\mathbf{V}}^{(i)}(\mathbf{h}_{t+1}) = \sum_{l=1}^{k} w_l \mathbf{V}^{(j_l)}(\mathbf{h}_{t+1})$ or equivalently, $\mathbf{C}_{\mathbf{V}}^{(i)} = \sum_{l=1}^{k} w_l \mathbf{P}^{(j_l)}\mathbf{V}^{(j_l)}$. This is because $\mathbf{P}^{(j_l)}$ is a non-expansion, and so

$$\begin{aligned} \|\mathbf{C}_{\mathbf{V}_1}^{(i)} - \mathbf{C}_{\mathbf{V}_2}^{(i)}\| = \left\| \sum_{l=1}^{k} w_l \mathbf{P}^{(j_l)}(\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}) \right\| &\leq \sum_{l=1}^{k} |w_l| \|\mathbf{P}^{(j_l)}(\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)})\| \\ &\leq (\max_l |w_l|) \sum_{l=1}^{k} \|\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}\|. \end{aligned}$$

The third assumption is standard for showing Bellman operators are contractions, and is easily satisfied if the policy is proper: is guaranteed to visit at least one state where the continuation is less than 1.

Assumption 3 (Discounted Transitions are Contractions)**.** *For all* $j \in \{1, \ldots, n\}$, $\beta_j \stackrel{\text{def}}{=} \|\mathbf{P}^{(j)}\| < 1$, *where* $\|\cdot\|$ *is the spectral norm.*

With these three assumptions, we can prove the main result.

Theorem 1. *Under Assumptions 1-3, iterating* $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ *converges to a unique fixed point.*

Proof. We first prove that the sequence of value estimates converges (Part 1) and then that it converges to a unique fixed point (Part 2 and 3).

Part 1: *The sequence* $\mathbf{V}_1, \mathbf{V}_2, \ldots$ *defined by* $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ *converges to a limit* $\mathbf{V}^* \in \mathbb{R}^{n|\mathcal{H}|}$.

Because G is acyclic, we have a linear topological ordering of the vertices, $i_1, \ldots, i_n$: for each directed edge (i, j), i comes before j in the ordering. Therefore, starting from the last GVF $j = i_n$, we know that the Bellman operator $\mathbf{B}^{(j)}$ is a contraction with rate $\beta_j < 1$,

$$\|\mathbf{B}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{B}^{(j)}\mathbf{V}_0^{(j)}\| = \|\mathbf{P}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{P}^{(j)}\mathbf{V}_0^{(j)}\| \leq \beta_j \|\mathbf{V}_1^{(j)} - \mathbf{V}_0^{(j)}\|.$$

Therefore, iterating $\mathbf{B}$ for t steps results in the error

$$\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \leq \beta_j^t \|\mathbf{V}_1^{(j)} - \mathbf{V}_0^{(j)}\|$$

and as $t \to \infty$, $\mathbf{V}_t^{(j)}$ converges to its fixed point.

We will use induction for the argument, with the above as the base case. Assume for all $j \in \{i_k, \ldots, i_n\}$ there exists a ball of radius $\epsilon(t)$ where $\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \leq \epsilon(t)$ and $\epsilon(t) \to 0$ as $t \to \infty$. Consider the next GVF in the ordering, $i = i_{k-1}$.

Case 1: There are no outgoing edges from i. If i does not use another GVF j in its cumulant, then iterating with $\mathbf{B}$ independently iterates $\mathbf{V}_t^{(i)}$ with $\mathbf{B}^{(i)}$. Therefore, as above, $\mathbf{V}_t^{(i)}$ converges because the Bellman operator is a contraction. In this setting, clearly such an $\epsilon_i(t)$ exists because $\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \to 0$ as $t \to \infty$.

Case 2: The cumulant for GVF i is composed of the values for the set of GVFs $\mathcal{J} \subseteq \{i_k, \ldots, i_n\}$. The basic idea, formalized below, is that GVF i will be guaranteed to converge once the GVFs used to construct the become sufficiently accurate. The update is $\mathbf{V}_{t+1}^{(i)} = \mathbf{C}_{\mathbf{V}_t}^{(i)} + \mathbf{P}^{(i)}\mathbf{V}_t^{(i)}$. The change in $\mathbf{V}_t^{(i)}$ is

$$\begin{aligned}
\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| &= \|(\mathbf{C}_{\mathbf{V}_t}^{(i)} - \mathbf{C}_{\mathbf{V}_{t-1}}^{(i)}) + \mathbf{P}^{(i)}(\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)})\| \\
&\leq K_i \sum_{j \in \mathcal{J}} \|\mathbf{V}_t^{(j)} - \mathbf{V}_{t-1}^{(j)}\| + \beta_i \|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\| \\
&\leq nK_i\epsilon(t-1) + \beta_i \|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|.
\end{aligned}$$

In the first inequality, the first term is due to Lipschitz continuity of the cumulant and the second term is due to the fact that $\|\mathbf{P}^{(i)}\| = \beta_i$. In the second inequality, we know $\|\mathbf{V}_t^{(j)} - \mathbf{V}_{t-1}^{(j)}\| \leq \epsilon_j(t)$, under the inductive hypothesis. The second inequality is loose, as the

sum only involves $|\mathcal{J}| < n$ terms, but we use n for simplicity since the results goes through with this constant as well. For sufficiently large t, $\epsilon(t-1)$ can be made arbitrarily small. If $nK_i\epsilon(t-1) < (1-\beta_i)\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$, i.e., $\epsilon(t-1) < \frac{(1-\beta_i)}{nK_i}\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$ then

$$\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| \leq \tilde{\beta}_i\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\| \qquad \text{for some } \tilde{\beta}_i < 1$$

and so the iteration is a contraction on step t. Else, if $\epsilon(t-1) \geq \frac{(1-\beta_i)}{nK_i}\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$, then this implies the difference $\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\|$ is already within a small ball, with radius $nK_i\epsilon(t-1)/(1-\beta_i)$. As $t \to \infty$, the difference can oscillate between being within this ball, which shrinks to zero because $\epsilon(t)$ shrinks to zero, or being iterated with a contraction that also shrinks the difference. In either case, there exists an $\epsilon_i(t)$ such that $\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| \leq \epsilon_i(t)$, where $\epsilon_i(t) \to 0$ as $t \to \infty$.

By induction, we have such an ϵ_i for all GVFs in the network. Therefore, we know the sequence $\mathbf{V}_t^{(i)}$ converges.

Part 2: $\mathbf{V}^*$ *is a fixed point of* $\mathbf{B}$.

Because the Bellman network operator is continuous, the limit can be taken inside the operator

$$\mathbf{V}^* = \lim_{t\to\infty} \mathbf{V}_t = \lim_{t\to\infty} \mathbf{B}\mathbf{V}_{t-1} = \mathbf{B}\left(\lim_{t\to\infty} \mathbf{V}_{t-1}\right) = \mathbf{B}\mathbf{V}^*$$

Part 3: $\mathbf{V}^*$ *is the only fixed point of* $\mathbf{B}$.

Consider an alternative solution $\mathbf{V}$. Because of the uniqueness of fixed points under Bellman operators, all those GVFs that have non-compositional cumulants have unique fixed points and so those components in $\mathbf{V}$ must be the same as $\mathbf{V}^*$. All the GVFs next in the ordering that use those GVFs as cumulants must then also converge to a unique value, because their Bellman operators with fixed GVFs as cumulants have a unique fixed point. This argument continues for the remaining GVFs in the ordering. ■

Corollary 1. *Under Assumption 3 with non-compositional cumulants (no edges in G), iterating* $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ *converges to a unique fixed point.*

Proposition 1 (Necessity of Acyclic Composition)**.** *There exists transition function* $\mathrm{P} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0,1]$ *and policy* $\pi : \mathcal{S}\times\mathcal{A} \to [0,1]$ *such that, for two GVFs in a cycle, iteration with the Bellman network operator diverges.*

Proof. Assume there are two states, with the policy defined such that we get the following dynamics for the Markov chain

$$\mathbf{P}^\pi = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \tag{11}$$

Assume further that $\gamma = 0.95$. The resulting Bellman iteration is

$$\begin{aligned}
\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} &= \mathbf{P}^\pi\begin{bmatrix}\mathbf{V}^{(2)} \\ \mathbf{V}^{(1)}\end{bmatrix} + \gamma\mathbf{P}^\pi\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} \\
&= \mathbf{P}^\pi\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} + \mathbf{P}^\pi\begin{bmatrix} \gamma & 0 \\ 0 & \gamma \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} \\
&= \mathbf{P}^\pi\begin{bmatrix} \gamma & 1 \\ 1 & \gamma \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix}
\end{aligned}$$

Since the matrix $\mathbf{P}^\pi \begin{bmatrix} \gamma & 1 \\ 1 & \gamma \end{bmatrix}$ is an expansion, for many initial $\begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix}$ this iteration goes to infinity, such as initial $\mathbf{V}^{(1)} = \mathbf{V}^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. ■

6.2 The Objective Function for GVFNs

With a valid Bellman network operator, we can proceed to defining the objective function for GVFNs. The above fixed point equation assumes a tabular setting, where the values can be estimated directly for each history. GVFNs, however, have a restricted functional form, where the value estimates must be a parametrized function of the current observation and value predictions from the last time step. Under such a functional form, it is unlikely that we can exactly solve for the fixed point. Rather, we will solve for a projected fixed point, which projects into the space of representable value functions.

Define the space of functions as

$$\mathcal{F} = \Big\{ \mathbf{V}_{\boldsymbol{\theta}} = [\mathbf{V}_{\boldsymbol{\theta}}^{(1)}, ..., \mathbf{V}_{\boldsymbol{\theta}}^{(n)}] \in \mathbb{R}^{n|\mathcal{H}|} \;\; | \;\; \text{where } \boldsymbol{\theta} \in \Theta \;\; \text{and} \\ V_{\boldsymbol{\theta}}(\mathbf{h}_{t+1}) = f_{\boldsymbol{\theta}}([V_{\boldsymbol{\theta}}^{(1)}(\mathbf{h}_t), \dots, V_{\boldsymbol{\theta}}^{(n)}(\mathbf{h}_t)], \mathbf{x}_{t+1}) \;\; \text{when } \Pr(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{x}_{t+1}) > 0 \Big\} \tag{12}$$

Recall that $\mathbf{x}_{t+1} = [a_t, \mathbf{o}_{t+1}]$. We know $\Pr(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{x}_{t+1}) > 0$ only when $\mathbf{h}_{t+1} \equiv \mathbf{h}_t a_t \mathbf{o}_{t+1}$, and so expect this to only be true for one outcome $\mathbf{h}_{t+1}$. We write that $\mathbf{h}_{t+1}$ is equivalent, rather than equal, to the current history appended with action a_t and observation $\mathbf{o}_{t+1}$, because $\mathbf{h}_{t+1}$ might be shorter (more minimal): earlier actions and observations might not be needed. Define the projection operator

$$\Pi_{\mathcal{F}}(\mathbf{V}) \stackrel{\text{def}}{=} \min_{\hat{\mathbf{V}} \in \mathcal{F}} \|\mathbf{V} - \hat{\mathbf{V}}\|_{\mathbf{d}}^2 \qquad \text{where } \|\mathbf{V} - \hat{\mathbf{V}}\|_{\mathbf{d}}^2 \stackrel{\text{def}}{=} \sum_{\mathbf{h} \in \mathcal{H}} \mathbf{d}(\mathbf{h})(V(\mathbf{h}) - \hat{V}(\mathbf{h}))^2 \tag{13}$$

where $\mathbf{d} : \mathcal{H} \to [0, 1]$ is the sampling distribution over histories. Typically, we assume data is generated by following a behavior policy $\mu : \mathcal{H} \to [0, 1]$, and that $\mathbf{d}$ is the stationary distribution for this policy. The value functions for policies π_i are typically learned off-policy, since in general π_i will not equal μ. The behavior policy μ used to gather the data is different, or off of, the policy—or policies—that we are evaluating.

To obtain the projected fixed point solution, a natural goal is to minimize the following projected objective,

$$\min_{\boldsymbol{\theta} \in \Theta} \|\Pi_{\mathcal{F}} \mathbf{B}\mathbf{V}_{\boldsymbol{\theta}} - \mathbf{V}_{\boldsymbol{\theta}}\|_{\mathbf{d}}^2 \tag{14}$$

Unfortunately, this objective can be hard to compute, because the projection operator $\Pi_{\mathcal{F}}$ onto the nonlinear manifold can be intractable. Instead, we take the same approach as Maei, Szepesvári, Bhatnagar, and Sutton (2010), when defining the nonlinear MSPBE for learning value functions with neural networks and other nonlinear function approximators. The idea is to approximate the projection onto the nonlinear manifold by assuming it is locally linear. Then, we can use a linear projection operator, defined locally at the current set of parameters $\boldsymbol{\theta} \in \Theta$, spanned by the basis $\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h}) \stackrel{\text{def}}{=} \nabla_\theta \mathbf{V}_{\boldsymbol{\theta}}^{(j)}(\mathbf{h})$ for all $\mathbf{h} \in \mathcal{H}$ and GVFs j. Let $\Phi_{j,\boldsymbol{\theta}}$ correspond to the matrix of stacked $\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})^\top$ for all $\mathbf{h} \in \mathcal{H}$, having $|\mathcal{H}|$ rows. We further

define

$$\Phi_{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \begin{bmatrix} \Phi_{1,\boldsymbol{\theta}} \\ \vdots \\ \Phi_{n,\boldsymbol{\theta}} \end{bmatrix} \qquad \mathbf{D} \stackrel{\text{def}}{=} \text{diag}\begin{bmatrix} \mathbf{d} \\ \vdots \\ \mathbf{d} \end{bmatrix} \qquad \Pi_{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \Phi_{\boldsymbol{\theta}}(\Phi_{\boldsymbol{\theta}}^\top \mathbf{D}\Phi_{\boldsymbol{\theta}})^{-1}\Phi_{\boldsymbol{\theta}}^\top \mathbf{D}.$$

Using this locally linear approximation to the objective potentially expands the set of stationary points. The fixed points under the original projection are still fixed points under this locally linear approximation. But, there could be points that are fixed points under this locally linear approximation, that would not be under the original.

We call the final objective using this projection the MSPBNE[3], defined as

$$\text{MSPBNE}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \|\Pi_{\boldsymbol{\theta}}\mathbf{B}\mathbf{V}_{\boldsymbol{\theta}} - \mathbf{V}_{\boldsymbol{\theta}}\|_{\mathbf{d}}^2 \tag{15}$$

We show in the following lemma, with proof in Appendix A.1, that in can be rewritten in a way that makes it more amenable to compute and sample gradients.[4] We will use this reformulation to develop algorithms to minimize this objective in the next section.

Lemma 1. *The MSPBNE defined in Equation* (15) *can be rewritten as*

$$\textit{MSPBNE}(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})^\top W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta}) \tag{16}$$

where

$$W(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_d\left[\sum_{j=1}^n \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top\right] = \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^n \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})^\top \tag{17}$$

$$\boldsymbol{\delta}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_{j=1}^n \mathbb{E}_{d,\pi_j}\left[\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\right]$$

$$\delta_j(H,A,H') \stackrel{\text{def}}{=} c^{(j)}(H,A,H') + \gamma^{(j)}(H,A,H')V_{\boldsymbol{\theta}}^{(j)}(H') - V_{\boldsymbol{\theta}}^{(j)}(H).$$

From this reformulation, one can see that the MSPBNE objective is a weighted quadratic objective, with weighting matrix $W(\boldsymbol{\theta})$ on vector $\boldsymbol{\delta}(\boldsymbol{\theta})$. The objective is zero—and so minimal—when $\boldsymbol{\delta}(\boldsymbol{\theta}) = \mathbf{0}$. This is similar to the temporal difference (TD) learning fixed point criteria. In fact, TD implicitly optimizes the linear MSPBE, which corresponds to the above objective with $n = 1$ and fixed features that do not depend on the parameters. Once we have a projected Bellman error objective, we can take advantage of the many advances in formulating TD algorithms to optimized MSPBE objectives. Therefore, though this objective looks quite complex, there is substantial literature to facilitate minimizing the MSPBNE.

3. A variant of the MSPBNE has been introduced for TD networks (Silver, 2012); the above generalizes that MSPBNE to GVF Networks. Because it is a strict generalization, we use the same name.
4. Since developing the MSPBNE, an alternative approach to defining a nonlinear MSPBE has been developed using a conjugate form for the Bellman error (see Dai, He, Pan, Boots, & Song, 2017 and in-preparation work that makes the connection the MSPBE more explicit (Patterson, Ghiassian, Gupta, White, & White, 2021)). The extension here should be relatively straightforward, as we formulate the objective using histories.

7. Algorithms for the MSPBNE

The algorithms to optimize the MSPBNE are a relatively straightforward combination of standard algorithms for RNNs and the TD algorithms designed to optimize the MSPBE. To provide some intuition on these algorithms, and how to obtain this combination of TD and RNN algorithms, we begin with a simpler setting: extending TD to a recurrent setting, with one GVF. From there, we introduce two algorithms for the MSPBNE: Recurrent TD and Recurrent GTD.

Consider first the on-policy TD update, without recurrence, assuming the true state $\mathbf{s}_t$ at time t is given:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_t \delta_t \nabla_{\boldsymbol{\theta}} V_{\boldsymbol{\theta}}(\mathbf{s}_t) \qquad \text{where } \delta_t \stackrel{\text{def}}{=} C_{t+1} + \gamma_{t+1} V_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}) - V_{\boldsymbol{\theta}}(\mathbf{s}_t).$$

With recurrence, where the state is estimated and so is a function of $\boldsymbol{\theta}$, the only difference to this update is in the computation of $\nabla_{\boldsymbol{\theta}} V_{\boldsymbol{\theta}}(\mathbf{s}_t)$, where $\mathbf{s}_t$ should instead be thought of $\mathbf{s}_t(\boldsymbol{\theta})$. This gradient now requires the chain rule, to account for the impact of $\boldsymbol{\theta}$ on the last state, and the state before then, and so on:

$$\frac{\partial V_{\boldsymbol{\theta}}(\mathbf{s}_t)}{\partial \boldsymbol{\theta}_i} = \frac{\partial V_{\boldsymbol{\theta}}(\mathbf{s}_t)}{\partial \mathbf{s}_t}^\top \frac{\partial \mathbf{s}_t}{\partial \boldsymbol{\theta}_i}$$

where $\mathbf{s}_t = f_{\boldsymbol{\theta}}(\mathbf{s}_{t-1}, \mathbf{x}_t)$. Computing this gradient back-in-time, $\nabla_{\boldsymbol{\theta}} \mathbf{s}_t$—which is also called the *sensitivity*—is precisely the aim of most RNN algorithms, including truncated BPTT and RTRL. Any algorithm that computes sensitivities can be used to obtain a TD update with recurrent connections to estimate the state.

For GVFNs, there are two differences: we need to account for off-policy sampling and the fact that state is itself composed of these value estimates, rather than being learned to estimate values. Value estimation within GVFNs requires off-policy updates, because the target policies π_j are not typically equal to the behavior policy μ. Therefore, we also need to include importance sampling ratios in the update

$$\rho_{t,j} \stackrel{\text{def}}{=} \frac{\pi_j(A_t|\mathbf{h}_t)}{\mu(A_t|\mathbf{h}_t)} \qquad \text{for all } j \in \{1, 2, \ldots, n\}.$$

This ratio multiplies the TD update, to adjust the expectation of the update to be as if action A_t had been taken under π_j rather than the behavior μ. For the second difference, the Recurrent TD update is actually even simpler because the value function itself is the state. For the j-th value function—which is the j-th state variable—we get that $\nabla_{\boldsymbol{\theta}} V_{\boldsymbol{\theta}}^{(j)}$ at time t is $\nabla_{\boldsymbol{\theta}} \mathbf{s}_{t,j}$. Notice that this gradient actually corresponds to using the above chain rule update, by using $V^{(j)}(\mathbf{s}_t) = \mathbf{s}_{t,j}$ as a selector function into the state variable.

The **Recurrent TD** update for GVFNs is

$$\begin{aligned}
\mathbf{s}_t &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t) && \triangleright \text{ where } \mathbf{x}_t \stackrel{\text{def}}{=} [a_{t-1}, \mathbf{o}_t] \\
\mathbf{s}_{t+1} &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_t, \mathbf{x}_{t+1}) && \triangleright \text{ where } \mathbf{x}_{t+1} \stackrel{\text{def}}{=} [a_t, \mathbf{o}_{t+1}] \\
\boldsymbol{\phi}_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}} \mathbf{s}_{t,j} && \triangleright \text{ Compute sensitivities using truncated BPTT} \\
\delta_{t,j} &\leftarrow C^{(j)}_{t+1} + \gamma^{(j)}_{t+1} \mathbf{s}_{t+1,j} - \mathbf{s}_{t,j} \\
\rho_{t,j} &\leftarrow \frac{\pi^{(j)}(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)} && \triangleright \text{ Policies can be functions of histories, not just of } \mathbf{o}_t \\
\boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha_t \left[\sum_{j=1}^{n} \rho_{t,j} \delta_{t,j} \boldsymbol{\phi}_{t,j} \right] && (18)
\end{aligned}$$

The TD update, however, is only an approximate semi-gradient update, even in the fully observable setting. To obtain exact gradient formulas, we turn to Gradient TD (GTD) algorithms. In particular we extend the nonlinear GTD strategy developed by Maei et al. (2010), to the MSPBNE. As above, we will immediately be able to use any algorithm to compute the sensitivities in the Recurrent GTD algorithm. But, the algorithm becomes more complex, simply because nonlinear GTD is more complex than TD even without recurrence.

We can use the following theorem to facilitate estimating the gradient. The main idea is to introduce an auxiliary weight vector, $\mathbf{w}$, to provide a quasi-stationary estimate of part of the objective. This proof and explicit derivation for the resulting Recurrent TD algorithm is given in the appendix. In the main body, we only provide the result for non-compositional GVFs: no GVFs predict the outcomes of other GVFs. This makes the algorithm easier to follow. We prove the more general result and derivation in Appendix A, in Theorem 3.

Theorem 2. *Assume that $V_{\boldsymbol{\theta}}(\mathbf{h})$ is twice continuously differentiable as a function of $\boldsymbol{\theta}$ for all histories $\mathbf{h} \in \mathcal{H}$ where $\mathbf{d}(\mathbf{h}) > 0$ and that $W(\cdot)$, defined in Equation (17), is non-singular in a small neighbourhood of $\boldsymbol{\theta}$. Assume further that there are no compositional GVFs in the GVFN: no GVFs has a cumulant that corresponds to another GVFs prediction. Then for $W(\boldsymbol{\theta})$ and $\boldsymbol{\delta}(\boldsymbol{\theta})$ defined in Lemma 1,*

$$\mathbf{w}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} W(\boldsymbol{\theta})^{-1} \boldsymbol{\delta}(\boldsymbol{\theta}) \tag{19}$$

$$\begin{aligned}
\hat{\delta}_{j,\theta}(H) &\stackrel{\text{def}}{=} \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top \mathbf{w}(\boldsymbol{\theta}) \\
\boldsymbol{\psi}(\boldsymbol{\theta}) &\stackrel{\text{def}}{=} \mathbb{E}_{d,\mu} \left[\sum_{j=1}^{n} \rho_j(H,A) \Big(\delta_j(H,A,H') - \hat{\delta}_{j,\theta}(H) \Big) \nabla^2 V^{(j)}_{\boldsymbol{\theta}}(H) \mathbf{w}(\boldsymbol{\theta}) \right] \qquad (20)
\end{aligned}$$

we get the gradient

$$-\tfrac{1}{2} \nabla MSPBNE(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta}) - \mathbb{E}_{d,\mu} \Big[\rho_j(H,A) \gamma^{(j)}(H,A,H') \hat{\delta}_{j,\theta}(H) \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H') \Big] - \boldsymbol{\psi}(\boldsymbol{\theta}) \tag{21}$$

We now have two additional terms to estimate beyond the standard sensitivities in a typical RNN gradient. First, we need to estimate this additional weight vector $\mathbf{w}$, given in

Equation (19). This can be done using standard techniques in reinforcement learning. Second, we also need to estimate a Hessian-vector product, given in Equation (20). Fortunately, this can be computed using R-operators, without explicitly computing the Hessian-vector product, using only computation linear in the length of the vector.

The **Recurrent GTD** update, for this simpler setting without composition, is[5]

$$
\begin{aligned}
\mathbf{s}_t &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t) \\
\mathbf{s}_{t+1} &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_t, \mathbf{x}_{t+1}) \\
\boldsymbol{\phi}_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}} \mathbf{s}_{t,j} && \triangleright \text{ Compute sensitivities using truncated BPTT} \\
\boldsymbol{\phi}'_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}} \mathbf{s}_{t+1,j} \\
\rho_{t,j} &\leftarrow \frac{\pi^{(j)}(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)} \\
\mathbf{v}_t &\leftarrow \nabla^2 \mathbf{s}_t \mathbf{w}_t && \triangleright \text{ Computed using R-operators, see Appendix A.3} \\
\hat{\delta}_{t,j} &\leftarrow \boldsymbol{\phi}_{t,j}^\top \mathbf{w}_t \\
\boldsymbol{\psi}_t &\leftarrow \sum_{j=1}^{n} (\rho_{t,j}\delta_{t,j} - \hat{\delta}_{t,j}) \mathbf{v}_t \\
\boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha_t \left[\sum_{j=1}^{n} \rho_{t,j}\delta_{t,j}\boldsymbol{\phi}_{t,j} - \rho_{t,j}\gamma_{j,t+1}\hat{\delta}_{t,j}\boldsymbol{\phi}'_{t,j} \right] - \alpha_t \boldsymbol{\psi}_t \qquad (22) \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \beta_t \left[\sum_{j=1}^{n} \rho_{t,j}(\delta_{t,j} - \hat{\delta}_{t,j})\boldsymbol{\phi}_{t,j} \right]
\end{aligned}
$$

The derivation for this algorithm is similar to the derivation for Gradient TD Networks (Silver, 2012), though for this more general setting with GVF Networks. We include additional algorithm details and derivations in Appendix A, including the general RGTD algorithm for compositional GVFs in equation (28).

As alluded to, there are a variety of possible strategies to optimize the MSPBNE for GVFNs. This variety arises from different strategies to optimize RNNs, back-in-time, as well as from the variety of strategies to optimize the MSPBE for value estimation. For example, we can compute sensitivities using truncated BPTT or RTRL and its many approximations. Similarly, for the MSPBE, there are a variety of different strategies to approximate gradients, because the gradient is not straightforward to sample. These including a variety of gradient TD methods—such as GTD and GTD2—saddlepoint methods and semi-gradient TD (see Ghiassian, Patterson, White, Sutton, & White, 2018 for a more exhaustive list).

8. Connections to Other Predictive State Approaches

The idea that an agent's knowledge might be represented as predictions has a long history in machine learning. The first references to such a predictive approach can be found in the work of Cunningham (1972), Becker (1973), and Drescher (1991), who hypothesized that

5. As mentioned above, we could have considered an alternative MSPBNE, using an in-development nonlinear MSPBE objective (Patterson et al., 2021). The resulting Recurrent GTD algorithm would look very similar, except the Hessian-vector product could be omitted: $\boldsymbol{\psi}_t$ is simply dropped in the update to θ.

agents would construct their understanding of the world from interaction, rather than human engineering. These ideas inspired work on predictive state representations (PSRs) (Littman et al., 2001), as an approach to modeling dynamical systems. Simply put, a PSR can predict all possible interactions between an agent and it's environment by reweighting a minimal collection of core test (sequence of actions and observations) and their predictions, without the need for a finite history or dynamics model. Extensions to high-dimensional continuous tasks have demonstrated that the predictive approach to dynamical system modeling is competitive with state-of-the-art system identification methods (Hsu, Kakade, & Zhang, 2012). PSRs can be combined with options (Wolfe & Singh, 2006), and some work suggests discovery of the core tests is possible (McCracken & Bowling, 2005). One important limitation of the PSR formalism is that the agent's internal representation of state must be composed exclusively of probabilities of action-observation sequences.

A PSR can be represented as a GVF network by using a myopic $\gamma = 0$ and compositional predictions. For a test $q = a_1\mathbf{o}_2$, for example, to compute the probability of seeing $\mathbf{o}_2$ after taking action a_1, the cumulant is 1 if $\mathbf{o}_2$ is observed and 0 otherwise; the policy is to always take action a_1; and the continuation $\gamma = 0$. To get a longer test, say $a_0\mathbf{o}_1a_1\mathbf{o}_2$, a second GVF can be added which predicts the output of the first GVF. For this second GVF, the cumulant is the prediction from the first GVF (which predicts the probability of seeing $\mathbf{o}_2$ given a_1 is taken); the policy is to always take action a_0; and the continuation is again $\gamma = 0$. Though GVFNs can represent a PSR, they do not encompass the discovery methods or other nice mathematical properties of PSRs, such as can be obtained with linear PSRs.

TD networks (Sutton & Tanner, 2004) were introduced after PSRs, and inspired by the PSR approach to state construction that is grounded in observations. GVFNs build on and are a strict generalization of TD networks. A TD network (Sutton & Tanner, 2004) is similarly composed of n predictions, and updates using the current observation and previous step predictions like an RNN. TD networks with options (Rafols et al., 2005) condition the predictions on temporally extended actions similar to GVF Networks, but do not incorporate several of the recent modernizations around GVFs, including state-dependent discounting and convergent off-policy training methods. The key differences, then, between GVF Networks and TD networks is in how the question networks are expressed and subsequently how they can be answered. GVF Networks are less cumbersome to specify, because they use the language of GVFs. Further, once in this language, it is more straightforward to apply algorithms designed for learning GVFs.

More recently, there has been an effort to combine the benefits of PSRs and RNNs. This began with work on Predictive State Inference Machines (PSIMs) (Sun et al., 2016), for inference in linear dynamical systems. The state is learned in a supervised way, by using statistics of the future k observations as targets for the predictive state. This earlier work focused on inference in linear dynamical systems, and did not state a clear connection to RNNs. Later work more explicitly combines PSRs and RNNs (Downey et al., 2017; Choromanski et al., 2018), but restricts the RNN architecture to a bilinear update to encode the PSR update for predictive state. In parallel, Venkatraman et al. (2017) proposed another strategy to incorporate ideas from PSRs into RNNs, without restricting the RNN architecture, called Predictive State Decoders (PSDs) (Venkatraman et al., 2017). Instead of constraining internal state to be predictions about future observations, statistics about future observations are used as auxiliary tasks in the RNN.

Of all these approaches, the most directly related to GVFNs is PSIMs. This connection is most clear from the PSIM objective (Sun et al., 2016, Equation 8), where the goal is to make predictive state match a vector of statistics about future outcomes. There are some key differences, mainly due to a focus on offline estimation in PSIMs. The predictive questions in PSIMs are typically about observations 1-step, 2-step up to k-steps into the future. To use such targets, batches of data need to be gathered and statistics computed offline to create the targets. Further, the state-update (filtering) function is trained using an alternating minimization strategy, with an algorithm called DAgger, rather than with algorithms for RNNs. Nonetheless, the motivation is similar: using an explicit objective to encourage internal state to be a predictive state.

A natural question, then, is whether the types of questions used by GVFNs provides advantages over PSIMs. Unlike k-step predictions in the future, GVFs allow questions about outcomes infinitely far into the far, through the use of cumulative discounted sums. Such predictions, though, do not provide high precision about such future events. As motivated in Section 3, GVFs should be easier to learn online. In our experiments, we include a baseline, called a Forecast Network, that uses k-step predictions as predictive features, to provide some evidence that GVFs are more suitable as predictive features for online agents.

9. Experiments in Forecasting

In this section, we compare GVFNs and RNNs on two time series prediction datasets, particularly to ask 1) can GVFNs obtain comparable performance and 2) do GVFNs allow for faster learning, due to the regularizing effect of constraining the state to be predictions.[6] We investigate if they allow for faster learning both by examining learning speed as well as robustness to truncation length in BPTT.

Datasets We consider two time series datasets previously studied in a comparative analysis of RNN architectures by (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017): the Mackey-Glass time series (previously introduced), and the Multiple Superimposed Oscillator.

The single-variate **Mackey-Glass (MG)** time series dataset is a synthetic data set generated from a time-delay differential equation:

$$\frac{\partial y(t)}{\partial t} = \alpha \frac{y(t-\tau)}{1 + y(t-\tau)^{10}} - \beta y(t). \tag{23}$$

We follow the learning setup in (Bianchi et al., 2017): we set $\tau = 17$, $\alpha = 0.2$, $\beta = 0.1$, and we take integration steps of size 0.1. We forecast the target variable y twelve steps into the future, starting from an initial value $y(0) = 1.2$. We generate $m = 600,000$ samples.

The **Multiple Superimposed Oscillator (MSO)** synthetic time series (Jaeger & Haas, 2004) is defined by the sum of four sinusoids with unique frequencies

$$y(t) = \sin(0.2t) + \sin(0.311t) + \sin(0.42t) + \sin(0.51t). \tag{24}$$

The resulting oscillator has a long period of $2000\pi \approx 6283.19$. Because we generate data using $t \in \mathbb{N}$, the oscillator effectively never returns to a previously seen state. These attributes

6. All code for these experiments can be found at `https://github.com/mkschleg/GVFN`

make prediction difficult with the MSO, as the model cannot rely on memory alone to make good predictions. We generate $m = 600,000$ samples and make predictions with a forecast horizon of $h = 12$.

Experiment Settings The focus in this work is on online prediction, and so we report online prediction error. At each step t, after observing $o_t = y(t)$, the RNN (or GVFN) makes a prediction $\hat{y}_t$ about the target y_t, which is the observation 12 steps into the future, $y_t = y(t + h)$. The magnitude of the squared error $(\hat{y}_t - y_t)^2$ depends on the scale of y_t. To provide a more scale invariant error, we normalize by the mean of the target—a mean predictor. Specifically, for each run, we report average error over windows of size 10000 with the mean predictor is computed for each window. This results in $m/10000$ normalized squared errors, where m is the length of the time series. We repeat this process 30 times, and average these errors across the 30 runs, and take the square root, to get a Normalized Root Mean Squared Error (NRMSE).

We fixed the values for hyperparameters as much as possible, using the previously reported value for the RNN and reasonable defaults for the GVFN. The stepsize is typically difficult to pick ahead of time, and so we sweep that hyperparameter for all the algorithms. We attempted to make the number of hyperparameters swept comparable for all methods, to avoid an unfair advantage. We do not tune the truncation length, as we report results for each truncation length $p \in \{1, 2, 4, 8, 16, 32\}$ for all the algorithms.

Algorithm Details The GVFN consists of a single layer of size 32 and 128 (for MG and MSO respectively), corresponding to horizon GVFs. As described in Section 5, each GVF has a constant continuation $\gamma^{(j)} \in [0.2, 0.95]$ and cumulant $C_t^{(j)} = \frac{1-\gamma^{(j)}}{y_t^{\max}} y(t)$, where $y_t^{\max}$ is an incrementally-computed maximum of the observations $y(t)$ up to time t. The GVFs are generated to linearly cover the range $[0.2, 0.95]$. This set is chosen as one of the simplest options that can be used without much domain knowledge. It is likely not the optimal set of GVFs for the GVFN, but represents a reasonable default choice. The GVFN is followed by a fully-connected layer with relu activations to produce a non-linear representation, which is linearly weighted to predict the target. The GVFN layer uses a linear activation, with clipping between [-10, 10], to help ensure state features remain bounded; again, this represented a simple rather than optimized choice.

The GVFN was trained using Recurrent TD with a constant learning rate and a batch size of 32. The weights for the fully-connected relu layer and the weights for the linear output are trained using ADAM, to minimize the mean squared error between the prediction at time t and target $y(t + h)$. We swept the stepsize hyperparameters: the learning rate for the GVFN $\alpha_{\text{GVFN}} = N \cdot 10^{-k}$ for $N \in \{1, 5\}$, $k \in \{3, \ldots, 6\}$, and the learning rate for the fully-connected and output layers $\alpha_{\text{pred}} = N \cdot 10^{-k}$ for $N \in \{1, 5\}$, $k \in \{2, \ldots, 5\}$.

We compare to RNNs, LSTMs, and GRUs [7]. The network architecture is similar to the GVFN for all recurrent models. The RNN size is set to 32 for MG and 128 for MSO, while the GRU and LSTM have 8 hidden units for MG and 128 for MSO. Notice how the GRU and LSTM have fewer hidden units than the RNN and GVFN for the MG experiment. This roughly accounts for the increased complexity of the LSTMs and GRUs as compared to the GVFN and RNN. While this was needed to make all the models competitive in MG,

7. We use standard implementations found in Flux (Innes, 2018).

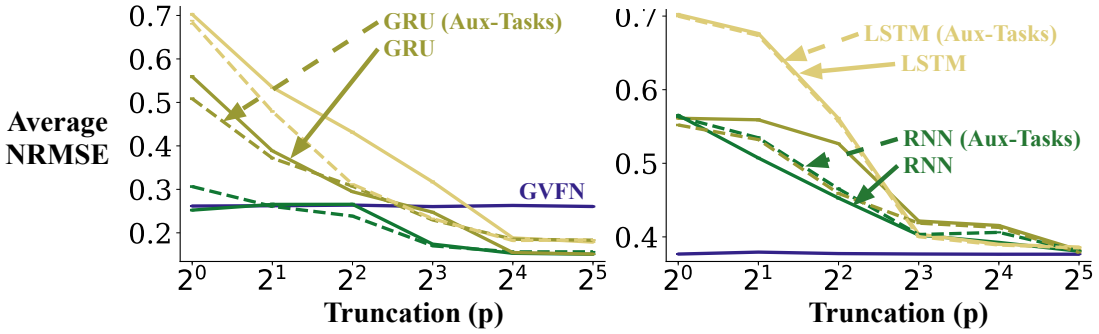


Figure 4: Truncation sensitivity for the (**left**) Mackey-Glass and (**right**) Multiple Superimposed Oscillator datasets. Errors are calculated using the normalized root mean squared error (NRMSE) averaged over the last 10k steps for the training results ± 1 standard error over 30 independent runs.

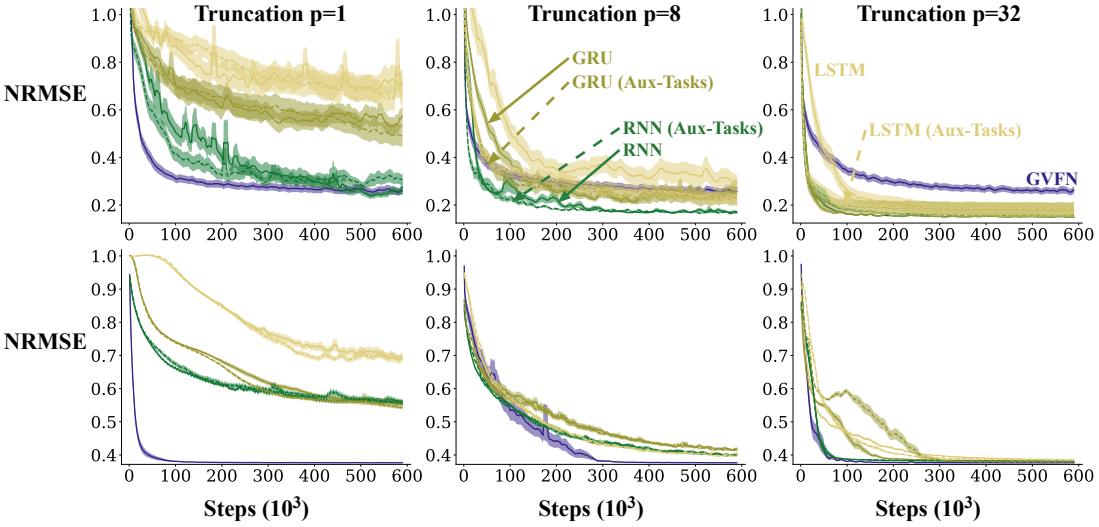


Figure 5: Learning curves for the (**top**) Mackey-Glass and (**bottom**) Multiple Superimposed Oscillator datasets. We are reporting the normalized root mean squared error (NRMSE) normalized to the performance of the windowed average baseline. We use the average of 30 independent runs ± the standard error.

we found the GVFNs performed well in MSO even with the same number of hidden units as the GRU and LSTMs. We trained these models using p-BPTT—specifically with the ADAM optimizer with a batch size of 32—to minimize the mean squared error between the prediction at time t and $y(t+h)$. We swept the learning rate $\alpha = 2^{-k}$ with $k \in \{1, \ldots, 20\}$.

Finally, we also compare to RNNs with the 128 GVFs as auxiliary tasks. The augmented RNN has the same architecture as above, but with an additional set of output heads. The additional GVF heads are the same as those used by the GVFN, and are trained with TD. The gradient information from the GVFs is back-propagated through the network, influencing the representation. The augmented RNN was tuned over the same values as the RNN. The

goal for adding this baseline is to gauge if there is an important difference in using the GVFs to directly constrain the state, as opposed to indirectly as auxiliary tasks. It further ensures that the RNN is given the same prior knowledge as the GVFN—namely the pertinence of these predictions—to avoid the inclusion of prior knowledge as a confounding factor.

All RNNs and GVFNs include a bias unit, as part of the input as well as in all layers. All methods have similar computation per step, particularly as they are run with the same truncation levels p.

Results We first show overall results across the truncation level in p-BPTT in Figure 4. Three results are consistent across both datatsets: 1) GVFNs can obtain significantly better performance than RNNs with small p; 2) GVFNs are surprisingly robust to truncation level, providing almost the same performance across p; and 3) auxiliary tasks in the RNN do not provide consistent benefits across models and datasets. GVFNs provide a strict improvement on the MSO dataset. The result on MG is more nuanced. As truncation levels increase, the RNN's performance significantly improves and then passes the GVFN. This might suggest some bias in the specification of the GVFs. As is typical with regularization or imposing an inductive bias, it can improve learning—here allowing for much more stable learning with small p—but can prevent the solution from reaching the same prediction accuracy. In some cases, if we are fortunate, the inductive bias is strictly helpful, constraining the solution in the right way so as to incur minimal bias but improve learning. In MSO, it's possible the GVF specification was more appropriate and in MG less appropriate.

To gain more detailed insight into the behavior of the algorithms across truncation levels, we show learning curves for $p \in \{1, 8, 32\}$ in Figure 5. All the approaches learn more slowly for $p = 1$, but the RNNs are clearly impacted more significantly. In MSO, the GVFN has a clear advantage in terms of learning speed. This is not true in MG, where once $p \geq 8$, the RNN performs better and learns faster. The GVFN objective here may actually be difficult to optimize, but it allows the agent to make progress constructing a useful state, whereas the signal from the error to the targets is insufficient.

10. Investigating Performance under Longer Temporal Dependencies

In this section, we investigate the utility of constraining states to be predictions, for an environment with long temporal dependencies. We use Compass World, introduced in Section 4 (see Figure 2), which can have long temporal dependencies, because the random behavior can stay in the center of the world for many steps, observing only the color white. The observation is encoded with two bits per color: one to indicate the agent observes that color, and the other to indicate another color is observed. The behavior policy chooses randomly between moving one-step forward; turning right/left for one step; moving forward until the wall is reached (*leap*); or randomly selecting actions for k steps (*wander*). The full observation vector is encoded based on which action was taken, and includes a bias unit.

We chose five hard-to-learn GVFs with predictions corresponding to the wall the agent is facing. These predictions are not learnable without constructing an internal state. These five questions correspond to leap questions. The leap question is defined as having a cumulant of 1 in the event of seeing a specific wall (orange, yellow, red, blue, green), and a continuation function defined as $\gamma = 0$ when any color is observed—when the agent is facing a wall—and $\gamma = 1$ otherwise.

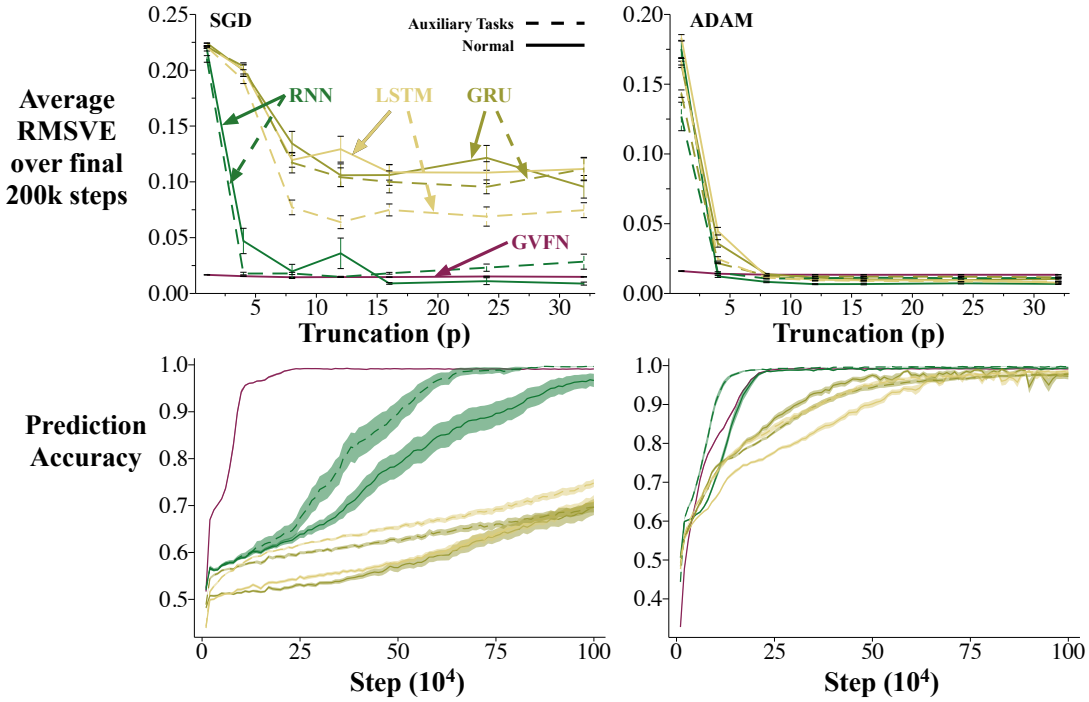


Figure 6: Results averaged over 30 runs ± one standard error. The dashed lines correspond to each RNN type augmented with auxiliary tasks, namely here the terminating horizon GVFs. The plots on the **(left)** are for a constant learning rate swept in range $\{0.1 \times 1.5^i; i \in [-10, 5]\} \cup \{1.0\}$. The plots on the **(right)** are for the ADAM optimizer with learning rate swept in range $\{0.01 \times 1.5^i; i \in \{-18, -16, \ldots, 0\}\}$. The **(top)** row shows sensitivity over truncation measured by the average root mean squared value error (RMSVE) over the final 200000 steps of training. The **(bottom)** row shows learning curves for $p = 4$ for prediction accuracy. We check if the prediction is correct by predicting the color of five with the highest GVF output, where the GVF prediction corresponds to a probability of facing that wall. When averaged over a window (10000 steps in our case) this results in a percentage of correct predictions during that time span.

We use the same architecture for both RNNs and GVFNs; the main difference is that for the GVFN we constrain the hidden state to be GVF predictions. The GVFN uses 40 GVFs: 8 GVFs per color. The 8 GVFs for a color correspond to **Terminating Horizon** GVFs. This means that they have a cumulant of 1 when seeing that color, and zero otherwise; they have a $\gamma = 1 - 2^k$ for one of 8 $k \in \{-7, -6, \ldots, -1\}$; they terminate—$\gamma$ becomes zero—when any color is observed; and the policy is to always go forward. These GVFs are similar to the horizon GVFs in time series prediction, except that termination occurs when a wall is reached and the policy is off-policy. The RNN similarly uses 40 hidden units for the recurrent layer. For RNNs, we use the hyperbolic tangent and the sigmoid function for GVFNs. We used sigmoids instead for GVFNs, because the returns are always nonnegative; otherwise, these two activations represent a similar architectural choice.

We found treating the input action a_t specially significantly improved performance of both the RNN and GVFN. This is done by specifying separate weight vectors $\{w_a \in \mathbb{R}^n; \forall a \in \mathcal{A}\}$ for each action the agent can take. The hidden state is then calculated as $\mathbf{s}_{t+1} = \sigma(w_{a_t}^\top [\mathbf{x}_{t+1}, \mathbf{s}_t])$, where σ is the activation function. For the GRUs and LSTMs, this architectural modification is not straightforward; instead we pass the action as a one-hot encoding.

All the approaches share the same structure following the recurrent layer. The state $\mathbf{s}_t$ is passed to a 32-dimensional hidden layer with relu activation, and then is linearly weighted to produce the predictions for the five hard-to-learn GVFs: $\hat{\mathbf{y}}_t = \text{relu}(\mathbf{s}_t^\top \mathbf{F})\mathbf{W}$ where $\mathbf{F} \in \mathbb{R}^{40\times 32}$ and $\mathbf{W} \in \mathbb{R}^{32\times 5}$. All methods include a bias unit on every layer.

The performance for increasing p, as well as learning curves for $p = 8$, are show in Figure 6. Again, we obtain a several clear conclusions. 1) The GVFN is again highly robust to truncation level, reaching almost perfect accuracy with $p = 1$. 2) The GVFN can learn noticeably faster with smaller p, such as $p = 4$, and the differences disappear for larger p. 3) The auxiliary tasks do not provide near the same level of benefit as the GVFN, though unlike the time series results, there does in fact seem to be some benefit. 4) All the methods are improved when using ADAM—especially the LSTMs and GRUs—though GVFNs are effective even with constant stepsizes.

11. Investigating Poorly Specified GVFNs

In the previous Compass World and Forecasting experiments, the GVFNs were robust to truncation. In fact, computing one-step gradients was sufficient for good performance. A natural question is when we can expect this to fail. We hypothesize that this robustness to truncation relies on appropriately specifying the GVFs in the GVFN. Poorly specified GVFs could both (a) make it so that the GVFN is incapable of constructing a state that can accurately predict the target and (b) make training difficult or unstable. In this section, we test this hypothesis by testing several choices for the GVFs in the GVFN in Compass World.

We consider three additional GVFN specifications: two that include intentional (but realistic) misspecifications and one that should be an improvement on the Terminating Horizon GVFN. The first misspecification, which we call the **Horizon** GVFN, causes the hidden states to have widely varying magnitudes. These GVFs are similar to the Terminating Horizon GVFs, except that they do not include termination when a color is observed. This means the true expected returns can be quite large, up to $\frac{1}{1-\gamma}$ (e.g., $\frac{1}{1-0.99} = 100$) if the agent is already immediately in front of the wall with that color. The policy is to go forward, and so if the agent is already facing the wall and receives a cumulant of 1, it will see a 1 forever onward, resulting in a return of $\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$.

The second misspecification provides a minimal set of sufficient predictions, but ones that are harder to learn. A natural choice for this is to use the five hard-to-learn predictions themselves, which is clearly sufficient but may be ineffective because we cannot learn them quickly enough to be a useful state. We call this the **Naive** GVFN, because it naively assumes that representability is enough, without considering learnability.

Finally, we also consider a specification that could improve on the more generic Terminating Horizon GVFN, that we call the **Expert Network**. This network also has 40 GVFs, but ones that are hand-designed for Compass World. This GVFN is a modified version of the TD network designed for Compass World (Sutton, Rafols, & Koop, 2005). The GVFs

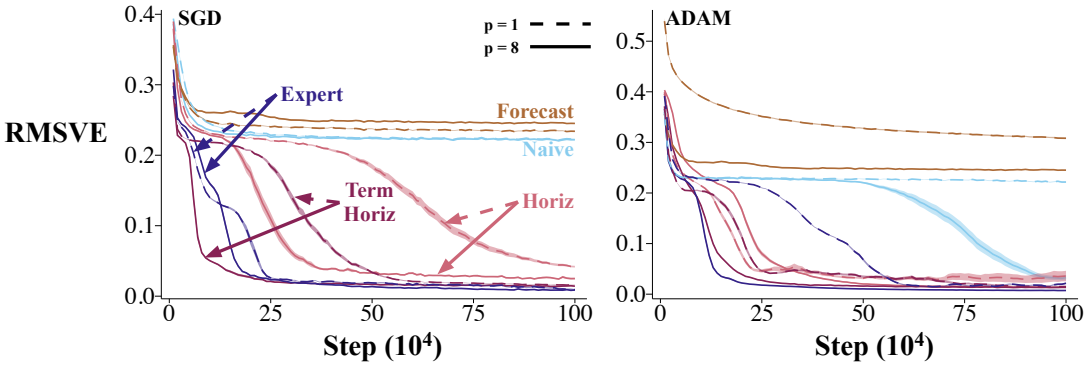


Figure 7: Learning curves for **(dashed)** $p = 1$ and **(solid)** $p = 8$ for various GVFN specifications and the Forecast networks. The GVFN is labeled TermHorizon, to highlight that it is composed of terminating horizon GVFs. Learning rates were chosen as in Figure 6, where the left plot corresponds to using a constant stepsize and the right to using the ADAM optimizer. The errors were averaged over 30 independent runs, to get the final learning curves $\pm$ standard error.

are defined similarly for the 5 colours. There are 3 myopic GVFs: a myopic GVFs consists of a myopic termination ($\gamma = 0$ always) and a cumulant of the color bit. Each myopic GVF has a persistent policy, which takes one action forever. Since there are three actions there are three myopic GVFs. These myopic GVFs indicate whether the agent is right beside the color (ahead, to the left or to the right). There is 1 leap GVF where the policy goes forward always, the cumulant is again the color bit and $\gamma = 1$ except when a color is observed, giving $\gamma = 0$. There are 2 GVFs with a persistent policy (left, right) with myopic termination and a cumulant of the previous leap GVF's. These compositional GVFs let the agent know if they were to first turn right (or left) and then go forward, would they see the color. There are 2 leap GVFs with cumulants of the myopic GVFs. Finally, there is 1 GVF with uniform random policy with $\gamma = 0$ at a wall event and $\gamma = 0.5$ otherwise.

As a baseline, we also include what we call a **Forecast** network, which uses k-horizon predictions for the hidden state instead of GVFs. The architecture of the Forecast network is otherwise the same as the GVFN. We use a set of horizons $\mathcal{K} = \{1, 2, \ldots, 8\}$, for each of the non-white observations, resulting in a hidden state size of 40. To train these networks online we keep a buffer of $p + \max(\mathcal{K})$ observations, using the first p observations in the BPTT calculation and the next k observations to determine the targets of the network. We then recover the most recent hidden state to train the evaluation GVFs as we would with the RNN and GVFN architectures. More specifically, at time step t, we update state $\mathbf{s}_{t-k}$ with observations $\mathbf{o}_{t-k+1}, \ldots, \mathbf{o}_t$.

Learning curves for all the GVFN specifications, as well as the Forecast network, with $p = 1$ and $p = 8$, are reported in Figure 7. The results indicate that the specification can have a big impact. The two misspecified GVFNs perform noticeably worse than the Terminating Horizon GVFN. As expected, the Naive GVFN is eventually able to learn, with enough steps, $p = 8$ and the ADAM optimizer. It is sufficient to obtain a good state, but poor learnability prevents it from playing a useful role. The Horizon GVFN, which has potentially high magnitude GVF predictions, is closer in performance to the Terminating

Horizon GVFN, but clearly worse. The Expert GVFN, on the other hand, can get to a lower error, though it does not have a clear advantage in terms of learning speed or robustness to p; this slower learning could again be potentially due to the fact that these expert GVFs were more difficult to learn than the simpler terminating horizon GVFs. Finally, the Forecast network performed very poorly. This is not too surprising in this environment. When considering a k-horizon prediction, the target is often zero, with the occasional one. This is generally a hard learning problem, as the resulting prediction loss does not provide a useful constraint. These results clearly show specifying the GVFs used to constrain the hidden state is an important consideration when using GVFNs, and could be the difference between learnable and not learnable representations.

12. Comparing Recurrent GTD and Recurrent TD

TD networks with a simple TD network update rule—no backprop through time—have been shown to have divergence issues on a simple six-state domain, called Ringworld (Tanner & Sutton, 2005). In fact, Gradient TD networks (Silver, 2012) were introduced precisely to solve this problem. Because GVFNs are a strict generalization of TD networks, we can set the GVFN to get the same problematic setting if we use a simple TD update (RTD with $p = 1$). This raises a natural question of if Recurrent TD (RTD) similarly has divergence issues, and if we need to use Recurrent GTD (RGTD).

In all of our experiments so far, we have opted for the simpler RTD algorithm, rather than the full gradient algorithm RGTD, because empirically we found little difference between the two. RTD, unlike the simple TD update rule, does in fact compute gradients back-in-time, and so should be a more sound update. Further, once we use truncated BPTT, even RGTD is providing a biased estimate of the gradient. But nonetheless RTD—which is built on the semi-gradient TD update—does drop more of the gradient than RGTD. It is likely that RGTD is needed in some cases. But it is possible that for most settings, RTD provides a reasonable interim choice between the simple TD network learning rule, and the more complex RGTD.

In this section, we test RTD and RGTD on Ringworld, to see if they perform differently on this known problematic setting. Note that for $p = 1$, RTD reduces to the simple TD network learning rule, and so we expect poor performance.

Ring World is a six-state domain (Tanner & Sutton, 2005) where the agent can move left or right in the ring. All the states are indistinguishable except state six. The observation vector is simply a two bit binary encoding indicating if the agent is in state six or not. The agent behaves uniformly randomly. The goal is to predict the observation bit on the next time step. The environment itself is not too difficult for state-construction; rather a particular TD network causes divergence from the simple TD update rule. The corresponding GVFN consists of two chains of compositional GVFs: one chain for always go right and one chain for always going left. In the first chain, the first GVF is a myopic GVF, that has as cumulant the observed bit after taking action Right, with $\gamma = 0$. This first GVF predicts the observation one step into the future. The second GVF has the first GVFs prediction as a cumulant after taking action Right, with $\gamma = 0$. This second GVF predicts the observation two steps into the future. There are five GVFs in each chain, for a total of 10 GVFs in the GVFN.

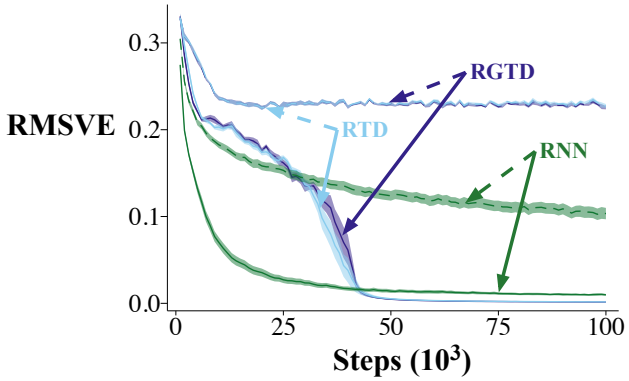


Figure 8: Learning curves for $p = 1$ and $p = 2$ averaged over 10 runs with fixed window smoothing of 1000 steps, in the Ringworld environment. Learning rates chosen from a sweep over $\alpha \in \{0.1 \times 1.5^i; i \in \{-10, -9, \ldots, 6\}\}$ for the RNN and learning rates $\alpha \in \{0.1 \times 1.5^i; i \in \{-6, -9, \ldots, 8\}\}$ and $\beta = \{0.0, 0.01\}$ corresponding to RTD and RGTD respectively. All approaches needed only $p = 2$ to learn, including the baseline RNN included for comparison.

Figure 8 shows the results of the Ring World experiments for truncation $p = 1$ and $p = 2$. The GVFNs for both RTD and RGTD needed only $p \geq 2$ to learn effectively. We also include a baseline RNN of the same architecture, that indicates that the GVFN specification does negatively impact performance. But, with even just $p = 2$, any convergence issues seem to disappear. In fact, RTD and RGTD perform very similarly. The fact that Ringworld is not problematic for RTD is by no means a proof that RTD is sufficient, especially since Ringworld was designed to be a counterexample for the simple TD network update not for RTD. But, it is one more datapoint that RTD and RGTD perform similarly. In future work, we will be investigating a counterexample for RTD, to better understand when it might be necessary to use RGTD.

13. A Discussion on Discovering GVFs for the GVFN

In this work, we were constrained to hand-designed GVFs for the GVFNs. While we were able to show several benefits of the framework, this limitation is apparent in Section 11 where we found that poorly specified GVFs—where we intentionally picked GVFs to have magnitude issues or that were difficult to learn—made the GVFN perform poorly as compared to the RNN. This outcome highlights the importance of the next question about how to improve the selection of predictive questions for a GVFN, and how to make this discovery process automatic and situated in the agent's stream of experience. An approach to discovery will also enable GVFNs to be applied to problems in which a set of GVF questions is not immediately apparent, and problems where our simple heuristic methods would create a set too large to manage computationally.

Previous approaches to discovery in predictive representations have focused on finding a set of predictions that would enable the agent to answer all predictive questions accurately. This objective is trying to find a sufficient statistic of the history for all predictions, and has been discussed in various forms (Subramanian, Sinha, Seraj, & Mahajan, 2020). This is the

approach typically taken in PSRs and a usual criteria when approaching a POMDP problem. This criteria falls naturally from the POMDP specification, where the assumption is there is a true underlying latent state which the agent can determine from enough interactions with the system. We conjecture that finding such a state is not feasible in large complex problems, and searching for such a state would be a poor use of a finite set of computational resources. Instead, the agent should focus on finding a set of questions which is useful for the agents overarching goals—for example, maximizing the return in the control problem.

In the following section, we describe several prior approaches to discovery applicable to the GVFN framework, develop a simple approach to a discovery framework for future testing, and discuss various ways of specifying GVFs by hand for the GVFN.

13.1 Previous Approaches

There are two main families of approaches to discovery of GVFs for GVFNs: generate-and-test and gradient descent.

Generate and test is a natural algorithmic approach when considering a search problem through a complex unordered (or not obviously ordered) space. The core of the approach is to propose GVFs through a generator and approximate their utility for the downstream task through a proxy measure. This approach has been used for representation discovery (Mahmood & Sutton, 2013; Javed, White, & Bengio, 2020). The simplest setting where such a generate-and-test approach could be used is time series forecasting, as the predictions are on-policy and so policies do not have to be proposed by the generated. Further, practitioners can apply their prior knowledge in creating the cumulant and continuation functions considered by the generator. There are, however, some simple strategies for generating policies, which we discuss in Section 13.2.

A generate and test algorithm has been developed for TD networks (Makino & Takagi, 2008). The process of discovery involves creating new predictions built entirely from existing structures: senses or predictions. By building new predictions from existing predictions, it facilitates the creation of compositional structures. The system proposed in Makino and Takagi (2008) determines when a node (i.e. a prediction or sense) should be expanded on using three criteria. They then expand these nodes in specific ways to ask a broad set of compositional questions. TD networks do not include policies—rather they include action primitives—so the approach does not directly extend. However, the idea of iteratively creating such compositional structures does extend. For example, in this work, the expert network considered in Section 11 was composed of compositional GVFs. Compositional GVFs could be generated simply by using existing GVFs as the cumulant for the new GVF.

Meta-gradient descent uses gradient descent to learn meta-parameters that affect learning performance. The meta-parameters could correspond to initialization of a model for later fine tuning (Finn, Abbeel, & Levine, 2017), a set of GVF auxiliary tasks to improve representation learning in Atari (Veeriah, Hessel, Xu, Rajendran, Lewis, Oh, van Hasselt, Silver, & Singh, 2019) or parameterized options (Bacon, Harb, & Precup, 2017). This approach splits the problem into two optimization problems: an inner problem and an outer problem. The inner optimization consists of the usual control or prediction procedure, where the agent seeks to maximize the discounted return or lower prediction error. The

outer optimization calculates gradients through this procedure, with respect to the meta-parameters.

For example, to learn a set of GVFs as auxiliary tasks, Veeriah et al. (2019) parameterized the cumulant and continuation functions. They did not need to parameterize the policies for the GVFs as they assumed on-policy prediction: the policy π for the GVF is the current policy. These meta-parameters are optimized in the outer loop to produce auxiliary tasks that improve control performance in the inner loop. For our setting, we could similarly parameterize GVF questions, including the policy. This meta-gradient approach was reasonably effective for discovering GVFs as auxiliary tasks, though the procedure is expensive and has some trainability issues. Nonetheless, it is a reasonable direction for pursuing discovery for GVFNs.

13.2 Investigating a Simple Generate and Test Strategy for GVF Discovery

We base this simple discovery framework on algorithms described for representation search (Mahmood & Sutton, 2013) focusing on two main components: an evaluator, and a generator. The evaluator is responsible for testing GVFs and removing unused GVFs. The generator proposes new GVFs from a set of possible GVFs. We summarize our framework in Figure 9. The key questions are how GVFs are evaluated and how new ones proposed. Our goal here is simply to demonstrate one avenue for discovery in GVFNs, rather than to develop an algorithm for discovery; we therefore opt for what we believe are some of the simplest choices.

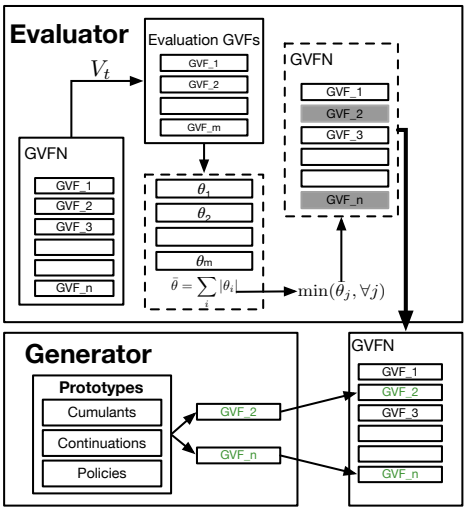


Figure 9: The discovery framework.

To evaluate the usefulness of a GVF we look at the magnitude of the associated weight in the external tasks using the GVFN. We assume the state vector is used linearly to make predictions, with θ_j corresponding to state s_j and so to the jth GVF. We evaluate all the GVFs every $N \in \mathbb{N}$ steps and prune the lowest $\epsilon \in [0, 1]$ percentage, i.e., prune $\lfloor n\epsilon \rfloor$ least useful GVFs of the full set of n GVFs. Other criteria have been proposed for evaluation, such as using traces of the weight magnitudes and considering internal weights (Mahmood & Sutton, 2013). As mentioned above, we opt for the simplest choice that is still reasonably effective.

We generate new GVFs randomly from a set of GVF primitives. We define a set of basic types of cumulants, continuations and policies from which to randomly sample. For continuations, we consider *myopic discounts* ($\gamma = 0$), *horizon discounts* ($\gamma \in (0, 1)$) and *terminating discounts* (the discount is set to $\gamma \in (0, 1]$ everywhere, except for at an event, which consists of a transition (o, a, o')). For cumulants, we consider *stimuli cumulants* (the cumulant is one of the observations, or taking on 0 or 1 if the observation fulfills some criteria (e.g. a threshold)) and *compositional cumulants* (the cumulant is the prediction of another GVF). We also use *random cumulants* (the cumulant is a random number generated from a

zero-mean Gaussian with a random variance sampled from a uniform distribution); we do not expect these to be useful, but rather use it to define what we call a dysfunctional GVF to test pruning. For the policies, we propose *random policies* (an action is chosen at random) and *persistent policies* (always follows one action).

The resulting GVF primitives consist of a triplet (c, γ, π) where each is randomly chosen from these basic types. For example, a randomly generated GVF could consist of a myopic continuation, a stimuli cumulant on observation bit one and a random policy. This would correspond to predicting the first component of the observation vector on the next step, assuming a random action is taken. As another example, a randomly generated GVF could consist of a termination continuation with $\gamma = 0.9$, a stimuli cumulant which is 1 when the observation is zero and is otherwise zero otherwise and a persistent policy with action forward. This GVF corresponds to predicting the likelihood of seeing the observation change from active ('1') to inactive ('0'), given the agent persistently moves forward, within the horizon of about $(1 - \gamma)^{-1} = 10$ steps.

We could also have considered parameterized continuations, cumulants and policies and randomly sample from that set. This set, however, is large. The GVF primitives can be seen as a prior over the full set of GVFs, which is too large from which to randomly generate. Without this prior we expect the discovery approach to still work but to take even longer than the experiments we present here.

We evaluate the performance of our system on two experiments in Compass World (Sutton et al., 2005). Both experiments use the five hard-to-learn GVFs as the targets for the GVFN, introduced in Section 10. These questions correspond to a question of "which wall will I hit if I move forward forever?". The first experiment, Figure 10 (left), provides a sanity check that the evaluation strategy prunes dysfunctional representational units. We initialize the GVF network with 200 GVFs: 45 used to form the expert crafted TD network (Sutton et al., 2005), and 155 defective GVFs predicting noise $\sim \mathcal{N}(0, \sigma^2)$. We report the learning curve and pruned GVFs over 12 million steps. The second experiment, Figure 10 (right), uses the full discovery approach to find a representation useful for learning the evaluation GVFs. We report the learning curves of the evaluative GVFs over 100 million steps.

These experiments have many similarities to the experiments above, but there is one key differences worth noting. Instead of using RTD or RGTD, we used TD(λ); see Appendix A.4 for the update equations. We found that this was sufficient to learn the expert network specification in a reasonable number of steps, and is significantly simpler than the other algorithms. Note that we did not use this algorithm in the above comparisons with RNNs, for two reasons. First in the cases where the target was not a return, it is not possible to use eligibility traces, as they are designed for predicting expected returns. Second, as far as we are aware, the eligibility trace calculation for neural networks with several output nodes has not been formally derived nor tested.

The results indicate that even a simple generate and test approach can be effective for discovery in GVFNs. The first figure shows that the pruning approach gradually removes the dysfunctional GVFs, without pruning the expert GVFs. Eventually, once the agent has mostly removed all the dysfunctional GVFs, it is then forced to prune the expert GVFs and prediction performance begins to drop. Of course, in practice, the agent would not prune all its GVFs; in this experiment we simply continue the pruning until the end to avoid biasing when we stop the agent. The second plot shows that iteratively pruning and generating

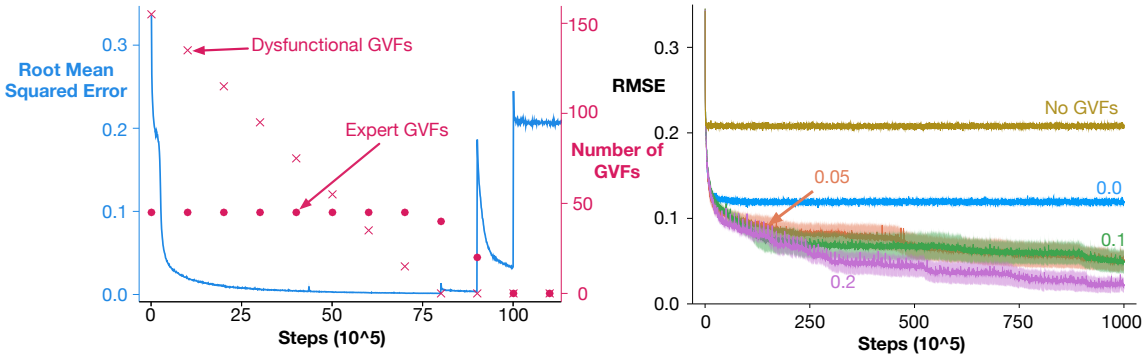


Figure 10: **(left)** Pruning predictive units occurs every million steps with no regeneration $\alpha = 0.001, \lambda = 0.9, \epsilon = 0.1, \sigma^2 = 1$ **(right)** Learning curves of the evaluative GVFs $N = 1000000, \epsilon = \text{labeled}, \alpha = 0.001, \lambda = 0.9, n = 100$, over 5 runs with standard error denoted by the shaded region.

new GVFs significantly improves on using an initial random set. For $\epsilon = 0.2$, which means about 20% of GVFs are pruned in each pruning phase, the prediction error continues to decrease until it almost reaches 0 and is almost as good as the set of hand-design GVFs used in previous experiments.

The goal of this experiment was to answer: is it possible to discover useful GVFs for a GVFN, even in simple settings? A negative answer would mean that GVFNs might have limited applicability. A demonstration that it is possible provides some evidence that this is a tractable problem for which even simple solutions can help us make traction. This demonstration, however, by no means shows an ideal or even efficient algorithm and there is ample room for improvement. Primarily, the random generation strategy does not take into account the current set of proposed predictions, potentially resulting in redundancy. A more principled method would look to generate a wide variety of predictions dependent on the current set of predictions. Another issue is the proxy used to determine a prediction's usefulness. Currently, the system will prune GVFs that are not directly useful, even if they are the cumulant for a useful GVF. The cumulant for the useful GVF is replaced by a new random GVF. This could reduce the quality of the predictive state or cause other instabilities within the GVFN. A simple approach is to define usefulness based also on compositional utility, not just on utility for the prediction task. The usefulness of a GVF should be higher if it is used by a GVF that is itself heavily relied on for accurate predictions, versus if it is only used by less useful GVFs.

13.3 Heuristics to specify GVFNs

Through testing GVFNs in several domains we have developed some rules of thumb for choosing GVFs which can be used today. In our time-series experiments, we found selecting GVFs with constant $\gamma^{(j)} \in [1 - 2^{-j}]$ to be surprisingly effective across the settings with fixed policies—namely the time series datasets. This is encouraging as these specifications on the surface seem simpler to discover than something as complex as the Expert network in Compass World. A set of discounts selected linearly across a range was also effective.

We also found that including GVFs which have a pseudo-termination at a known event (known due to expert knowledge) and a cumulant which is only active at this event improved learning performance considerably (see the performance of the Terminating-Horizon network in Section 11).

14. Discussion and Conclusions

In this work, we made a case for a new recurrent architecture, called GVF Networks. GVFNs constrain the hidden layer to correspond to predictions about the future, and so can be seen as a regularized or constrained RNN architecture. We first derive a sound fixed-point objective for these networks. We then show in experiments that GVFNs can outperform various RNN architectures with a much smaller truncation in BPTT. We demonstrated this phenomena on time series data as well as a RL prediction environment designed to have longer term dependencies. The goal of the paper was to further investigate the predictive representation hypothesis, where we asked if it is useful for trainability to restrict hidden states to be predictions. The work provided simpler algorithms than previous related work, such as TD networks, to test this hypothesis, as well as some evidence that restricting hidden state to be prediction can be beneficial. We finally investigated the impact of the specification of these predictions, and demonstrated that careful curation—an expert set—of GVFs could improve performance, but that relatively simple heuristics were also quite effective. We also found, though, that poorly specified GVFs—where we intentionally picked GVFs to have magnitude issues or that were difficult to learn—made the GVFN perform poorly as compared to the RNN.

In addition to trainability, constraining features to be predictions has other potential benefits we did not directly demonstrate in this work, primarily for transfer and adapting to changes in the environment. Predictive features can be useful for transfer because they can encode knowledge about dynamics that remain consistent, even when the agent has to make a new prediction or find a policy for a different reward function. Further, forgetting is a known problem with neural networks when transferring to new problems (McCloskey & Cohen, 1989; Kemker, McClure, Abitino, Hayes, & Kanan, 2018; Maltoni & Lomonaco, 2019); by primarily using the GVF prediction loss for the state update, it could alleviate some of these forgetting issues. The predictions learned in the state could also provide important information about the agent's experience, such as features that predict surprise (Günther, Kearney, Dawson, Sherstan, & Pilarski, 2018), knowledge about the variability of the environment (Sherstan, Bennett, Young, Ashley, White, White, & Sutton, 2018), and other various statistics (Modayil et al., 2014; White, 2015; Sherstan, 2020).

A related point is that predictive features could facilitate adapting more quickly when (part of) the world changes. There are two reasons for this. First, even if the targets for a GVF change, the GVF itself might still be a useful prediction to use in the state. By directly updating state to correspond to predictions, the features update quickly to respond to the change. This is in contrast to an RNN, where the feature update is more indirect. Second, even if parts of the world change, many predictions may remain accurate and pertinent. These features will remain stable even under the change, and only a smaller number of features, that need to change, will change. This property allows us to better re-use previous learning and promote stability.

In addition to testing these potential benefits as a next step, there are a few algorithmic extensions that are promising as well. The architecture proposed here is only tested in the online prediction setting, but we expect to see similar benefits in other settings in which RNNs are employed. One of particular interest is in applying GVFNs to the control setting. This can be easily done through having the final network output be a state-action value function as in a Deep Q-Network (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, et al., 2015; Hausknecht & Stone, 2015), or by using the state of the GVFN as input to an actor-critic algorithm such as Impala (Espeholt, Soyer, Munos, Simonyan, Mnih, Ward, Doron, Firoiu, Harley, Dunning, et al., 2018). Predictive representations built using GVFs have been shown to be advantageous in real-world control applications, such as autonomous driving (Graves, Nguyen, Hassanzadeh, & Jin, 2020), and we expect GVFNs to share similar properties in settings where state construction is necessary.

Finally, in this work, we constrained ourselves to GVFNs where all hidden states are predictions. A natural extension is to consider a GVFN that only constrains certain hidden states to be predictions and otherwise allows other states to simply be set to improve prediction accuracy for the targets. This modification could provide the improved stability of GVFNs, but improve representability. Additionally, GVFNs could even be combined with other RNN types, like LSTMs, by simply concatenating the states learned by the two RNN types. Overall, GVFNs provide a complementary addition to the many other RNN architectures available, particularly for continual learning systems with long temporal dependencies; with this work, we hope to expand interest and investigation further into these promising architectures.

15. Acknowledgements

We would like to thank the Alberta Machine Intelligence Institute, IVADO, NSERC and the Canada CIFAR AI Chairs Program for the funding for this research, as well as Compute Canada for the computing resources used for this work. We would also like to thank Marc Bellemare for helpful comments about extensions to infinite sets for the set of histories.

Appendix A. Algorithmic Details and Derivations

In this section we provide the derivation of Recurrent-GTD from the MSPBNE, including off-policy corrections, and the details in recursively calculating the gradients of a GVFN back through time using an RTRL derivation, with details easily extended to BPTT. We also briefly discuss the forecast networks used briefly in the main text.

A.1 Re-expressing the MSPBNE

The MSPBNE was derived for on-policy prediction questions, for TD Networks (Silver, 2012). The main extension is to allow for (1) off-policy prediction, which is straightforward to do using importance sampling ratios and (2) extension to continuation functions. Note that we assume that the target policies π_i do not change as the state estimate changes; rather, they are functions of history. We first show the result without importance sampling ratios, as they are implicit in the expectations. We then provide a corollary with explicit importance sampling ratios.

Lemma 1. *The MSPBNE defined in Equation* (15) *can be rewritten as*

$$MSPBNE(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})^\top W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta}) \tag{16}$$

where

$$W(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_d\left[\sum_{j=1}^{n} \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top\right] = \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{n} \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})^\top \tag{17}$$

$$\boldsymbol{\delta}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_{j=1}^{n} \mathbb{E}_{d,\pi_j}\left[\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\right]$$

$$\delta_j(H,A,H') \stackrel{\text{def}}{=} c^{(j)}(H,A,H') + \gamma^{(j)}(H,A,H')V_{\boldsymbol{\theta}}^{(j)}(H') - V_{\boldsymbol{\theta}}^{(j)}(H).$$

Proof. Starting with equation (15) and for $\Delta_{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \mathbf{B}\mathbf{V}_{\boldsymbol{\theta}} - \mathbf{V}_{\boldsymbol{\theta}}$, we get

$$\begin{aligned}
\text{MSPBNE}(\boldsymbol{\theta}) &= \|\Pi_{\boldsymbol{\theta}}\mathbf{B}\mathbf{V}_{\boldsymbol{\theta}} - \mathbf{V}_{\boldsymbol{\theta}}\|_{\mathbf{d}}^2 \\
&= \|\Pi_{\boldsymbol{\theta}}\left[\mathbf{B}\mathbf{V}_{\boldsymbol{\theta}} - \mathbf{V}_{\boldsymbol{\theta}}\right]\|_{\mathbf{d}}^2 \\
&= \|\Pi_{\boldsymbol{\theta}}\Delta_{\boldsymbol{\theta}}\|_{\mathbf{d}}^2
\end{aligned}$$

We can wrap the projection operator around the full TD error $\Delta_{\boldsymbol{\theta}}$, because it has no affect on $\mathbf{V}_{\boldsymbol{\theta}}$ which is already in the space. We then plug in the definition of $\Pi_{\boldsymbol{\theta}}$

$$\begin{aligned}
\Pi_{\boldsymbol{\theta}}^\top\mathbf{D}\Pi_{\boldsymbol{\theta}} &= \mathbf{D}^\top\Phi_{\boldsymbol{\theta}}(\Phi_{\boldsymbol{\theta}}^\top\mathbf{D}\Phi_{\boldsymbol{\theta}})^{-1}\Phi^\top\mathbf{D} \\
\|\Pi_{\boldsymbol{\theta}}\Delta_{\boldsymbol{\theta}}\|_{\mathbf{d}}^2 &= \Delta_{\boldsymbol{\theta}}^\top\Pi_{\boldsymbol{\theta}}^\top\mathbf{D}\Pi_{\boldsymbol{\theta}}\Delta_{\boldsymbol{\theta}} \\
&= \Delta_{\boldsymbol{\theta}}^\top\mathbf{D}^\top\Phi_{\boldsymbol{\theta}}(\Phi_{\boldsymbol{\theta}}^\top\mathbf{D}\Phi_{\boldsymbol{\theta}})^{-1}\Phi^\top\mathbf{D}\Delta_{\boldsymbol{\theta}}
\end{aligned} \tag{25}$$

As in prior gradient TD work we then convert the matrix operations to expectation forms.

$$\begin{aligned}
\Phi_{\boldsymbol{\theta}}^\top\mathbf{D}\Phi_{\boldsymbol{\theta}} &= \sum_{j=1}^{n}\sum_{\mathbf{h}\in\mathcal{H}} \mathbf{d}(\mathbf{h})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})\phi_{j,\boldsymbol{\theta}}(\mathbf{h})^\top = \mathbb{E}_d\left[\sum_{j=1}^{n}\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top\right] \\
&= W(\boldsymbol{\theta}) \\
\Phi_{\boldsymbol{\theta}}^\top\mathbf{D}\Delta_{\boldsymbol{\theta}} &= \sum_{j=1}^{n}\sum_{\mathbf{h}\in\mathcal{H}} \mathbf{d}(\mathbf{h})\boldsymbol{\phi}_{j,\theta}(\mathbf{h})\sum_{a\in\mathcal{A}}\pi_j(a|\mathbf{h})\mathbb{E}[\delta_j(\mathbf{h},a,H')] = \sum_{j=1}^{n}\mathbb{E}_{d,\pi_j}\left[\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\right] \\
&= \boldsymbol{\delta}(\boldsymbol{\theta})
\end{aligned}$$

Then substituting into equation (25), we get the result $\text{MSPBNE}(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})^\top W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta})$. ■

Now we do not actually get samples according to π_j; instead, we get them according to the behaviour μ. Throughout this work, we have assumed a coverage property for μ. This means that the behaviour policy μ satisfies $\mu(a|\mathbf{h}) > 0$ if any $\pi_j(a|\mathbf{h}) > 0$ for policies $\pi_1, \ldots, \pi_n$.

Corollary 2. *For importance sampling ratios* $\rho_j(a|\mathbf{h}) \stackrel{\text{def}}{=} \frac{\pi_j(a|\mathbf{h})}{\mu(a|\mathbf{h})}$ *and*

$$\begin{aligned}\boldsymbol{\delta}_\mu(\boldsymbol{\theta}) &\stackrel{\text{def}}{=} \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{n} \rho_j(H,A)\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\right] \\ &= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{a\in\mathcal{A}} \mu(a|\mathbf{h}) \sum_{j=1}^{n} \rho_j(a|\mathbf{h}) \mathbb{E}\left[\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\mathbf{h})|H=\mathbf{h},A=a\right]\end{aligned}$$

then we can show that $\boldsymbol{\delta}_\mu(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})$ *and so we can write*

$$MSPBNE(\boldsymbol{\theta}) = \boldsymbol{\delta}_\mu(\boldsymbol{\theta})^\top W(\boldsymbol{\theta})^{-1} \boldsymbol{\delta}_\mu(\boldsymbol{\theta})$$

Proof. The key is simply to show that $\boldsymbol{\delta}_\mu(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})$, because $W(\boldsymbol{\theta})$ depends only on d, not on the policies π or μ. This is straightforward with the typical cancellation in importance sampling ratios

$$\begin{aligned}\boldsymbol{\delta}_\mu(\boldsymbol{\theta}) &= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{a\in\mathcal{A}} \mu(a|\mathbf{h}) \sum_{j=1}^{n} \rho_j(a|\mathbf{h}) \mathbb{E}\left[\delta_j(\mathbf{h},a,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(h)|H=h,A=a\right] \\ &= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{n} \sum_{a\in\mathcal{A}} \mu(a|\mathbf{h}) \rho_j(a|\mathbf{h}) \mathbb{E}\left[\delta_j(\mathbf{h},a,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(h)|H=h,A=a\right] \\ &= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{n} \sum_{a\in\mathcal{A}} \pi_j(a|\mathbf{h}) \mathbb{E}\left[\delta_j(\mathbf{h},a,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(h)|H=h,A=a\right] \\ &= \boldsymbol{\delta}(\boldsymbol{\theta}).\end{aligned}$$

■

From here on, therefore, we assume that $\boldsymbol{\delta}(\boldsymbol{\theta})$ is defined more generally as the above $\boldsymbol{\delta}_\mu(\boldsymbol{\theta})$, since they result in the same objective but this more general expression more obviously highlights off-policy sampling.

A.2 Deriving Recurrent-GTD

Now that the objective is written in its expectation form, the gradients can be take with respect to the weight parameter. The main body stated the result for a simplified setting (Theorem 2), to make it simpler to understand the result. We provide the more general result here, for compositional GVFs.

Theorem 3. *Assume that* $V_{\boldsymbol{\theta}}(\mathbf{h})$ *is twice continuously differentiable as a function of* $\boldsymbol{\theta}$ *for all histories* $\mathbf{h} \in \mathcal{H}$ *where* $\mathbf{d}(\mathbf{h}) > 0$ *and that* $W(\cdot)$*, defined in Equation* (17)*, is non-singular in a small neighbourhood of* $\boldsymbol{\theta}$*. Then for*

$$\begin{aligned}\boldsymbol{\delta}(\boldsymbol{\theta}) &\stackrel{\text{def}}{=} \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{n} \rho_j(H,A)\delta_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)\right] \\ \mathbf{w}(\boldsymbol{\theta}) &= W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta}) \\ \boldsymbol{\psi}(\boldsymbol{\theta}) &= \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{n} \Big(\rho_j(H,A)\delta_j(H,A,H') - \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top \mathbf{w}(\boldsymbol{\theta})\Big)\nabla^2 V_{\boldsymbol{\theta}}^{(j)}(H)\mathbf{w}(\boldsymbol{\theta})\right]\end{aligned}$$

we get the gradient

$$-\frac{1}{2}\nabla MSPBNE(\boldsymbol{\theta}) = -\mathbb{E}_{d,\mu}\Big[\sum_{j=1}^{n}\rho_j(H,A)\nabla_{\boldsymbol{\theta}}\delta_j(H,A,H')\boldsymbol{\phi}_{j,\theta}(H)^\top\Big]\mathbf{w}(\boldsymbol{\theta}) - \boldsymbol{\psi}(\boldsymbol{\theta}) \tag{26}$$

$$= \boldsymbol{\delta}(\boldsymbol{\theta}) - \boldsymbol{\psi}(\boldsymbol{\theta}) \tag{27}$$
$$- \mathbb{E}_{d,\mu}\Big[\sum_{j+1}^{n}\rho_j(H,A)\Big[\sum_{i=1}^{n}c(j,i)\boldsymbol{\phi}_{i,\boldsymbol{\theta}}(H) + \gamma_j(H,A,H')\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H')\Big]\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H)^\top\mathbf{w}(\boldsymbol{\theta})\Big]$$

Proof. For simplicity in notation below, we drop the explicit dependence on the random variable H in the expectations.

$$\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H) \to \boldsymbol{\phi}_{j,\boldsymbol{\theta}}, \qquad \boldsymbol{\phi}_{j,\boldsymbol{\theta}}(H') \to \boldsymbol{\phi}'_{j,\boldsymbol{\theta}}$$
$$\delta_j(H,A,H') \to \delta_j, \qquad \rho_j(H,A) \to \rho_j$$

Further, we will use ∂_i to indicate the partial derivative with respect to $\boldsymbol{\theta}_i$. We also assume all expectations are with respect to d, and μ. We use J to denote the MSPBNE, which from Lemma 1 and Corollary 2, can be written $J(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})^\top W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta})$. When applying the product rule

$$\partial_i J(\boldsymbol{\theta}) = 2(\partial_i\boldsymbol{\delta}(\boldsymbol{\theta}))^\top\mathbf{w}(\boldsymbol{\theta}) + \boldsymbol{\delta}(\theta)^\top\partial_i W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\theta)$$
$$\partial_i\boldsymbol{\delta}(\boldsymbol{\theta}) = \mathbb{E}\Big[\sum_{j=1}^{n}\rho_j\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}}\delta_j + \boldsymbol{\phi}_{j,\boldsymbol{\theta}}\partial_i\delta_j\Big]$$
$$\partial_i W(\boldsymbol{\theta})^{-1} = -W(\boldsymbol{\theta})^{-1}\partial_i W(\boldsymbol{\theta})W(\boldsymbol{\theta})^{-1} = -2W(\boldsymbol{\theta})^{-1}\mathbb{E}\Big[\sum_{j=1}^{n}(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]W(\boldsymbol{\theta})^{-1}$$

Recall that $\mathbf{w}(\boldsymbol{\theta}) = W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\boldsymbol{\theta})$, and that $W(\boldsymbol{\theta})$ is symmetric, giving

$$\begin{aligned}\boldsymbol{\delta}(\theta)^\top\partial_i W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\theta) &= -2\boldsymbol{\delta}(\theta)^\top W(\boldsymbol{\theta})^{-1}\mathbb{E}\Big[\sum_{j=1}^{n}(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]W(\boldsymbol{\theta})^{-1}\boldsymbol{\delta}(\theta)\\ &= -2\mathbf{w}(\boldsymbol{\theta})^\top\mathbb{E}\Big[\sum_{j=1}^{n}(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}})\boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]\mathbf{w}(\boldsymbol{\theta})\\ &= -2\mathbf{w}(\boldsymbol{\theta})^\top\mathbb{E}\Big[\sum_{j=1}^{n}\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}})^\top\Big]\mathbf{w}(\boldsymbol{\theta})\end{aligned}$$

The last line follows from the fact that the transpose of a scalar is equal to the scalar. Here we transpose the whole expression, leading to a transpose of the outer-product inside the sum. Additionally,

$$\begin{aligned}\partial_i\boldsymbol{\delta}(\boldsymbol{\theta})^\top\mathbf{w}(\boldsymbol{\theta}) &= \mathbb{E}\Big[\sum_{j=1}^{n}\rho_j\delta_j(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}}) + \rho_j\boldsymbol{\phi}_{j,\boldsymbol{\theta}}\partial_i\delta_j\Big]^\top\mathbf{w}(\boldsymbol{\theta})\\ &= \mathbb{E}\Big[\sum_{j=1}^{n}\rho_j\delta_j(\partial_i\boldsymbol{\phi}_{j,\boldsymbol{\theta}})^\top\Big]\mathbf{w}(\boldsymbol{\theta}) + \mathbb{E}\Big[\sum_{j=1}^{n}\rho_j\partial_i\delta_j\boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]\mathbf{w}(\boldsymbol{\theta})\end{aligned}$$

Grouping the terms with $(\partial_i \boldsymbol{\phi}_{j,\boldsymbol{\theta}})$, we get

$$\begin{aligned}
&\mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \delta_j (\partial_i \boldsymbol{\phi}_{j,\boldsymbol{\theta}})^\top\Big]\mathbf{w}(\boldsymbol{\theta}) - \mathbf{w}(\boldsymbol{\theta})^\top \mathbb{E}\Big[\sum_{j=1}^{n} \boldsymbol{\phi}_{j,\boldsymbol{\theta}} (\partial_i \boldsymbol{\phi}_{j,\boldsymbol{\theta}})^\top\Big]\mathbf{w}(\boldsymbol{\theta}) \\
&= \mathbb{E}\Big[\sum_{j=1}^{n} \Big(\rho_j \delta_j - \mathbf{w}(\boldsymbol{\theta})^\top \boldsymbol{\phi}_{j,\boldsymbol{\theta}}\Big)(\partial_i \boldsymbol{\phi}_{j,\boldsymbol{\theta}})^\top \mathbf{w}(\boldsymbol{\theta})\Big] \\
&= \boldsymbol{\psi}_i(\boldsymbol{\theta})
\end{aligned}$$

where the last follows from the definition of $\nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}(\boldsymbol{\theta})$, which is the gradient vector composed of partial derivatives $\boldsymbol{\psi}_i(\boldsymbol{\theta})$. Therefore,

$$\begin{aligned}
\partial_i J(\boldsymbol{\theta}) &= 2\partial_i \boldsymbol{\delta}(\boldsymbol{\theta})^\top \mathbf{w}(\boldsymbol{\theta}) + \boldsymbol{\delta}(\theta)^\top \partial_i W(\boldsymbol{\theta})^{-1} \boldsymbol{\delta}(\theta) \\
&= 2\boldsymbol{\psi}_i(\boldsymbol{\theta}) + 2\mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \partial_i \delta_j \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big]
\end{aligned}$$

which proves Equation (26). Now we can further simplify the second term, using the fact that $\boldsymbol{\phi}_{j,\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} V_{j,\boldsymbol{\theta}}$, giving

$$\nabla_{\boldsymbol{\theta}} \delta_j = \nabla_{\boldsymbol{\theta}} c_{j,\boldsymbol{\theta}} + \gamma_j \boldsymbol{\phi}'_{j,\boldsymbol{\theta}} - \boldsymbol{\phi}_{j,\boldsymbol{\theta}}.$$

Now notice that

$$\begin{aligned}
\mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \nabla_{\boldsymbol{\theta}} \delta_j \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big] &= \mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \big(\nabla_{\boldsymbol{\theta}} c_{j,\boldsymbol{\theta}} + \gamma_j \boldsymbol{\phi}'_{j,\boldsymbol{\theta}} - \boldsymbol{\phi}_{j,\boldsymbol{\theta}}\big) \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big] \\
&= -\mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \boldsymbol{\phi}_{j,\boldsymbol{\theta}} \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]\mathbf{w}(\boldsymbol{\theta}) + \mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \big(\nabla_{\boldsymbol{\theta}} c_{j,\boldsymbol{\theta}} + \gamma_j \boldsymbol{\phi}'_{j,\boldsymbol{\theta}}\big) \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big]
\end{aligned}$$

Because $\mathbf{w}(\boldsymbol{\theta}) = W(\boldsymbol{\theta})^{-1} \boldsymbol{\delta}(\boldsymbol{\theta})$,

$$\mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \boldsymbol{\phi}_{j,\boldsymbol{\theta}} \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top\Big]\mathbf{w}(\boldsymbol{\theta}) = W(\boldsymbol{\theta})\mathbf{w}(\boldsymbol{\theta}) = \boldsymbol{\delta}(\boldsymbol{\theta})$$

Putting this all together, we get that

$$\begin{aligned}
-\tfrac{1}{2}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= -\boldsymbol{\psi}_i(\boldsymbol{\theta}) - \mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \nabla_{\boldsymbol{\theta}} \delta_j \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big] \\
&= -\boldsymbol{\psi}(\boldsymbol{\theta}) + \boldsymbol{\delta}(\boldsymbol{\theta}) - \mathbb{E}\Big[\sum_{j=1}^{n} \rho_j \big(\nabla_{\boldsymbol{\theta}} c_{j,\boldsymbol{\theta}} + \gamma_j \boldsymbol{\phi}'_{j,\boldsymbol{\theta}}\big) \boldsymbol{\phi}_{j,\boldsymbol{\theta}}^\top \mathbf{w}(\boldsymbol{\theta})\Big]
\end{aligned}$$

completing the proof. ■

The resulting Recurrent GTD algorithm explicitly learns a second set of weights $\mathbf{w}$, to perform this update. In our implementation, we use a particular form of composition, namely

that the cumulant for a GVF is a linear weighting of the predictions of some of the other GVFs on the next time step. If we let $c(i,j)$ indicate the weight on the ith GVF in the cumulant for the jth GVF, then we get that $\nabla_{\boldsymbol{\theta}} c_{j,t} = \sum_{i=1}^{n} c(j,i)\boldsymbol{\phi}'_{i,t}$.

The **Recurrent GTD** update is

$$
\begin{aligned}
\mathbf{s}_t &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t) \\
\mathbf{s}_{t+1} &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_t, \mathbf{x}_{t+1}) \\
\boldsymbol{\phi}_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}} \mathbf{s}_{t,j} && \triangleright \text{ Compute sensitivities using truncated BPTT} \\
\boldsymbol{\phi}'_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}} \mathbf{s}_{t+1,j} \\
\rho_{t,j} &\leftarrow \frac{\pi_j(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)} \\
\mathbf{v}_t &= \nabla^2 \mathbf{s}_t \mathbf{w}_t && \triangleright \text{ Computed using R-operators, see Appendix A.3} \\
\boldsymbol{\psi}_t &= \sum_{j=1}^{n} (\rho_{j,t}\delta_{j,t} - \boldsymbol{\phi}_{j,t}^\top \mathbf{w}_t)\mathbf{v}_t && (28) \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \Bigg[\sum_{j=1}^{n} \rho_{j,t}\delta_{j,t}\boldsymbol{\phi}_{j,t} - \rho_{j,t}\Big[\nabla_{\boldsymbol{\theta}} c_{j,t} + \gamma_{j,t+1}\boldsymbol{\phi}'_{j,t}\Big]\boldsymbol{\phi}_{j,t}^\top \mathbf{w}_t - \boldsymbol{\psi}_t \Bigg] \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \beta_t \Bigg[\sum_{j=1}^{n} \rho_{j,t}\Big(\delta_{j,t} - \boldsymbol{\phi}_{j,t}^\top \mathbf{w}_t\Big)\boldsymbol{\phi}_{j,t} \Bigg]
\end{aligned}
$$

A.3 Computing gradients of the value function back through time

In this section, we show how to compute $\boldsymbol{\phi}_t$, which was needed in the algorithms. Recall from Section 7 that we set $V^{(j)}(\mathbf{s}_{t+1}) = \mathbf{s}_{t+1,j}$, and using $\mathbf{z}_{t+1} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{s}_t \\ \mathbf{x}_{t+1} \end{bmatrix}$ let $\mathbf{s}_{t+1,j} = \sigma\left(\mathbf{z}_{t+1}^\top \boldsymbol{\theta}^{(j)}\right)$ for some activation function $\sigma : \mathbb{R} \to \mathbb{R}$. For both Backpropagation Through Time or Real Time Recurrent Learning, it is useful to take advantage of the following formula for *recurrent sensitivities*

$$
\begin{aligned}
\frac{\partial V^{(i)}(S_{t+1})}{\partial \boldsymbol{\theta}_{(k,j)}} &= \dot{\sigma}(\mathbf{z}_{t+1}^\top \boldsymbol{\theta}^{(i)}) \left(\left(\frac{\partial \mathbf{z}_{t+1}}{\partial \boldsymbol{\theta}_{(k,j)}} \right)^\top \boldsymbol{\theta}^{(i)} + (\mathbf{z}_{t+1})_j \delta^\kappa_{i,k} \right) \\
&= \dot{\sigma}(\mathbf{z}_{t+1}^\top \boldsymbol{\theta}^{(i)}) \left(\left[\frac{\partial V^{(1)}(S_t)}{\partial \boldsymbol{\theta}_{(k,j)}}, ..., \frac{\partial V^{(n)}(S_t)}{\partial \boldsymbol{\theta}_{(k,j)}}, \mathbf{0}^\top \right] \boldsymbol{\theta}^{(i)} + (\mathbf{z}_{t+1})_j \delta^\kappa_{i,k} \right)
\end{aligned}
$$

where δ^κ is the Kronecker delta function and $\dot{\sigma}(\cdot)$ is shorthand for the derivative of σ w.r.t its scalar input. Given this formula, BPTT or RTRL can simply be applied.

For Recurrent GTD—though not for Recurrent TD—we additionally need to compute the Hessian back in time, for the Hessian-vector product. The Hessian for each value function is a $n((d+n)) \times n((d+n))$ matrix; computing the Hessian-vector product naively would cost at least $O(((d+n)+n)^2 n^2)$ for each GVF, which is prohibitively expensive. We can avoid this using R-operators also known as Pearlmutter's method (Pearlmutter, 1994).

The R-operator $\mathcal{R}\{\cdot\}$ is defined as

$$
\mathcal{R}_{\mathbf{w}}\Big\{ \mathbf{g}(\boldsymbol{\theta}) \Big\} \stackrel{\text{def}}{=} \left. \frac{\partial \mathbf{g}(\boldsymbol{\theta} + r\mathbf{w})}{\partial r} \right|_{r=0}
$$

for a (vector-valued) function $\mathbf{g}$ and satisfies

$$\mathcal{R}_{\mathbf{w}}\Big\{\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\Big\} = \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\mathbf{w}.$$

Therefore, instead of computing the Hessian and then producting with $\mathbf{w}_t$, this operation can be completed in linear time, in the length of $\mathbf{w}_t$.

Specifically, for our setting, we have

$$\mathcal{R}_w\Big\{\dot\sigma(\mathbf{z}_t^\top\boldsymbol{\theta})[\nabla_{\boldsymbol{\theta}}\mathbf{z}_t^\top\boldsymbol{\theta} + \mathbf{z}_t^\top\nabla_{\boldsymbol{\theta}}\boldsymbol{\theta}]\Big\} = \frac{\partial}{\partial r}\Big(\dot\sigma(\mathbf{z}_t^\top(\boldsymbol{\theta}+r\mathbf{w})[\nabla_{\boldsymbol{\theta}}\mathbf{z}_t^\top(\boldsymbol{\theta}+r\mathbf{w}) + \mathbf{z}_t^\top\nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}+r\mathbf{w})]\Big)\Big|_{r=0}$$

To make the calculation more managable we seperate into each partial for every node k and associated weight j.

$$\begin{aligned}
\frac{\partial V^{(i)}(S_{t+1},\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_{(k,j)}} &= \dot\sigma(\mathbf{z}_{t+1}^\top\boldsymbol{\theta}^{(i)})(\eta_{t+1})_{i,k,j}\\
(\eta_{t+1})_{i,k,j} &= ((\mathbf{u}_t)_{k,j}^\top\boldsymbol{\theta}^{(i)} + (\mathbf{z}_{t+1})_j\delta_{i,k})\\
(\mathbf{u}_t)_{k,j} &= \left[\frac{\partial V^{(1)}(S_t)}{\partial\boldsymbol{\theta}_{(k,j)}},...,\frac{\partial V^{(n)}(S_t)}{\partial\boldsymbol{\theta}_{(k,j)}},\mathbf{0}^\top\right]^\top\\
\boldsymbol{\xi}_t &= \left[\frac{\partial V^{(1)}(S_t)}{\partial r},...,\frac{\partial V^{(n)}(S_t)}{\partial r},\mathbf{0}^\top\right]^\top
\end{aligned}$$

$$\begin{aligned}
(\mathcal{R}_t)_{w,V} &= \left[\mathcal{R}_w\left\{\frac{\partial V^{(1)}(S_{t-1})}{\partial\boldsymbol{\theta}_{(k,j)}}\right\},...,\mathcal{R}_w\left\{\frac{\partial V^{(n)}(S_{t-1})}{\partial\boldsymbol{\theta}_{(k,j)}}\right\},\mathbf{0}^\top\right]^\top\\
\mathcal{R}_w\left\{\frac{\partial V^{(i)}(S_{t+1},\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_{(k,j)}}\right\} &= \frac{\partial^2 V^{(i)}(S_{t+1},\boldsymbol{\theta}+r\mathbf{w})}{\partial r\partial\boldsymbol{\theta}_{(k,j)}}\Big|_{r=0}\\
&= \ddot\sigma\Big(\mathbf{z}_{t+1}^\top(\boldsymbol{\theta}^{(i)}+r\mathbf{w}_i)\Big)\Big(\boldsymbol{\xi}_t^\top(\boldsymbol{\theta}^{(i)}+r\mathbf{w}_i) + \mathbf{z}_{t+1}^\top\mathbf{w}_i\Big)(\eta_{t+1})_{i,k,j}\\
&\quad + \dot\sigma\Big(\mathbf{z}_{t+1}^\top(\boldsymbol{\theta}^{(i)}+r\mathbf{w}_i)\Big)\Big((\mathcal{R}_t)_{w,V}^\top(\boldsymbol{\theta}^{(i)}+r\mathbf{w}) + (\mathbf{u}_t)_{k,j}^\top w_i + (\boldsymbol{\xi}_t)_j\delta^\kappa_{k,i}\Big)\Big|_{r=0}\\
&= \ddot\sigma\Big(\mathbf{z}_{t+1}^\top\boldsymbol{\theta}^{(i)}\Big)\Big(\boldsymbol{\xi}_t^\top(\boldsymbol{\theta}^{(i)}) + \mathbf{z}_{t+1}^\top\mathbf{w}_i\Big)(\eta_{t+1})_{i,k,j}\\
&\quad + \dot\sigma\Big(\mathbf{z}_{t+1}^\top\boldsymbol{\theta}^{(i)}\Big)\Big((\mathcal{R}_t)_{w,V}^\top\boldsymbol{\theta}^{(i)} + (\mathbf{u}_t)_{k,j}^\top\mathbf{w}_i + (\boldsymbol{\xi}_t)_j\delta^\kappa_{k,i}\Big)\\
\frac{\partial V^{(i)}(S_t)}{\partial r} &= \dot\sigma(\mathbf{z}_t^\top\boldsymbol{\theta}^{(i)})(\boldsymbol{\xi}_{t-1}^\top\boldsymbol{\theta}^{(i)} + \mathbf{z}_t^\top w_i)
\end{aligned}$$

A.4 TD(λ) for learning GVFNs

For many of the experiments we used Recurrent TD with no back-propagation through time $p = 1$. This algorithm only adjusts parameters to minimize immediate TD error. In

many cases, this was sufficient, but at times it was slow and increasing p improved learning. Another strategy is to use traces to obtain credit assignment back-in-time. The TD-error on this step can be attributed to state values back-in-time, with the **TD(λ) algorithm**

$$
\begin{aligned}
\mathbf{s}_t &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t) \\
\mathbf{s}_{t+1} &\leftarrow f_{\boldsymbol{\theta}_t}(\mathbf{s}_t, \mathbf{x}_{t+1}) \\
\mathbf{g}_{t,j} &\leftarrow \nabla_{\boldsymbol{\theta}_j} f_{\boldsymbol{\theta}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t) && \triangleright \text{ gradient given } \mathbf{s}_{t-1}\text{, no BPTT} \\
\mathbf{e}_{t,j} &\leftarrow \mathbf{g}_{t,j} + \gamma_{t,j}\lambda \mathbf{e}_{t-1,j} && \triangleright \text{ eligibility trace, } 0 \leq \lambda \leq 1 \\
\delta_{t,j} &\leftarrow C^{(j)}_{t+1} + \gamma_{t+1,j}\mathbf{s}_{t+1,j} - \mathbf{s}_{t,j} \\
\boldsymbol{\theta}_{t+1,j} &\leftarrow \boldsymbol{\theta}_{t,j} + \alpha_t \delta_{t,j} \mathbf{e}_{t,j}
\end{aligned} \tag{29}
$$

Notice the difference to Recurrent TD and Recurrent GTD, that the weights for each GVF are updated independently. This difference arises because the gradient computations for back-in-time, for the sensitivities, is what couples the updates. Without these sensitivities, the immediate gradient of the value $\mathbf{g}_{t,j}$ is independent for each GVF.

References

Bacon, P.-L., Harb, J., & Precup, D. (2017). The Option-Critic Architecture.. In *AAAI Conference on Artificial Intelligence*.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*.

Baum, L. E., & Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*.

Becker, J. D. (1973). A model for the encoding of experiential information. *Computer Models of Thought and Language*.

Benzing, F., Gauy, M. M., Mujika, A., Martinsson, A., & Steger, A. (2019). Optimal kronecker-sum approximation of real time recurrent learning. In *International Conference on Machine Learning*.

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). An overview and comparative analysis of Recurrent Neural Networks for Short Term Load Forecasting.. *arXiv:1705.04378*.

Boots, B., Siddiqi, S., & Gordon, G. (2011). Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*.

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *International Conference on Machine Learning*.

Choromanski, K., Downey, C., & Boots, B. (2018). Initialization matters: Orthogonal Predictive State Recurrent Neural Networks. In *International Conference on Learning Representations*.

Cunningham, M. (1972). *Intelligence: Its Organization and Development*. Academic Press.

Dai, B., He, N., Pan, Y., Boots, B., & Song, L. (2017). Learning from conditional distributions via dual embeddings. In *International Conference on Artificial Intelligence and Statistics*, pp. 1458–1467.

Downey, C., Hefny, A., Boots, B., Gordon, G. J., & Li, B. (2017). Predictive State Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*.

Drescher, G. L. (1991). *Made-up minds: a constructivist approach to artificial intelligence*. MIT press.

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping. *IEEE Robotics and Automation Magazine*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., & White, A. (2018). Online Off-policy Prediction.. *arXiv:1811.02597*.

Graves, D., Nguyen, N. M., Hassanzadeh, K., & Jin, J. (2020). Learning predictive representations in autonomous driving to improve deep reinforcement learning. *arXiv preprint arXiv:2006.15110*.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232.

Günther, J., Kearney, A., Dawson, M. R., Sherstan, C., & Pilarski, P. M. (2018). Predictions, surprise, and predictions of surprise in general value function architectures. In *AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, pp. 22–29.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *AAAI Conference on Artificial Intelligence*.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*.

Hopfield, J. J. (1982). Neural Network and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences of the United States of America*.

Hsu, D., Kakade, S., & Zhang, T. (2012). A spectral algorithm for learning Hidden Markov Models. *Journal of Computer and System Sciences*.

Innes, M. (2018). Flux: Elegant Machine Learning with Julia. *Journal of Open Source Software*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*.

Jaeger, H., & Haas, H. (2004). Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*.

Javed, K., White, M., & Bengio, Y. (2020). Learning causal models online. *arXiv preprint arXiv:2006.07461*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*.

Kemker, R., McClure, M., Abitino, A., Hayes, T., & Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. *AAAI Conference on Artificial Intelligence*.

Lin, L.-J., & Mitchell, T. M. (1993). Reinforcement learning with hidden states. In *International Conference on Simulation of Adaptive Behavior*.

Littman, M. L., Sutton, R. S., & Singh, S. (2001). Predictive representations of state. In *Advances in Neural Information Processing Systems*.

Ljung, L. (2010). Perspectives on system identification. *Annual Reviews in Control*.

Maei, H., Szepesvári, C., Bhatnagar, S., & Sutton, R. (2010). Toward Off-Policy Learning Control with Function Approximation. In *International Conference on Machine Learning*.

Mahmood, A. R., & Sutton, R. S. (2013). Representation Search through Generate and Test.. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors*.

Makino, T., & Takagi, T. (2008). On-line discovery of temporal-difference networks. In *Proceedings of the 25th international conference on Machine learning*, pp. 632–639.

Maltoni, D., & Lomonaco, V. (2019). Continuous learning in single-incremental-task scenarios. *Neural Networks*, *116*, 56–73.

McCallum, R. A. (1996). Learning to use selective attention and short-term memory in sequential tasks. In *International Conference on Simulation of Adaptive Behavior*.

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

McCracken, P., & Bowling, M. H. (2005). Online discovery and learning of predictive state representations. In *Advances in Neural Information Processing Systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Modayil, J., White, A., & Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*.

Momennejad, I., & Howard, M. W. (2018). Predicting the future with multi-scale successor representations. *biorxiv*.

Mujika, A., Meier, F., & Steger, A. (2018). Approximating real-time recurrent learning with random kronecker factors. In *Advances in Neural Information Processing Systems*, pp. 6594–6603.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks.. In *International Conference on Machine Learning*.

Patterson, A., Ghiassian, S., Gupta, D., White, A., & White, M. (2021). Investigating Objectives for Off-policy Value Estimation in Reinforcement Learning. *In Preparation*.

Pearlmutter, B. A. (1994). Fast Exact Multiplication by the Hessian. *Neural Computation*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

Pezzulo, G. (2008). Coordinating with the future: the anticipatory nature of representation. *Minds and Machines*.

Rafols, E. J., Ring, M. B., Sutton, R. S., & Tanner, B. (2005). Using predictive representations to improve generalization in reinforcement learning. In *International Joint Conference on Artificial Intelligence*.

Rudary, M., Singh, S., & Wingate, D. (2005). Predictive linear-Gaussian models of stochastic dynamical systems. In *Conference on Uncertainty in Artificial Intelligence*.

Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., Cui, X., Ramabhadran, B., Picheny, M., Lim, L.-L., et al. (2017). English conversational telephone speech recognition by humans and machines. In *Interspeech*.

Schaul, T., & Ring, M. (2013). Better generalization with forecasts. In *International Joint Conference on Artificial Intelligence*.

Sherstan, C. (2020). *Representation and General Value Functions*. Ph.D. thesis, University of Alberta.

Sherstan, C., Bennett, B., Young, K., Ashley, D. R., White, A., White, M., & Sutton, R. S. (2018). Directly estimating the variance of the {\lambda}-return using temporal-difference methods. In *Conference on Uncertainty in Artificial Intelligence*.

Silver, D. (2012). Gradient Temporal Difference Networks. In *European Workshop on Reinforcement Learning*.

Subramanian, J., Sinha, A., Seraj, R., & Mahajan, A. (2020). Approximate information state for approximate planning and reinforcement learning in partially observed systems. *arXiv preprint arXiv:2010.08843*.

Sun, W., Venkatraman, A., Boots, B., & Bagnell, J. A. (2016). Learning to filter with predictive state inference machines. In *International Conference on Machine Learning*.

Sutskever, I. (2013). *Training recurrent neural networks*. Ph.D. thesis, University of Toronto.

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

Sutton, R. S., Rafols, E. J., & Koop, A. (2005). Temporal Abstraction in Temporal-difference Networks.. In *Advances in Neural Information Processing Systems*.

Sutton, R. S., & Tanner, B. (2004). Temporal-Difference Networks. In *Advances in Neural Information Processing Systems*.

Tallec, C., & Ollivier, Y. (2018). Unbiased Online Recurrent Optimization. In *International Conference on Learning Representations*.

Tanner, B., & Sutton, R. S. (2005). Temporal-Difference Networks with History.. In *International Joint Conference on Artificial Intelligence*.

Trinh, T. H., Dai, A. M., Luong, M.-T., & Le, Q. V. (2018). Learning longer-term dependencies in rnns with auxiliary losses. In *International Conference on Machine Learning*.

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

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., & Singh, S. (2019). Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, pp. 9306–9317.

Venkatraman, A., Rhinehart, N., Sun, W., Pinto, L., Hebert, M., Boots, B., Kitani, K., & Bagnell, J. (2017). Predictive-State Decoders: Encoding the Future into Recurrent Networks. *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

Vigorito, C. M. (2009). Temporal-Difference Networks for Dynamical Systems with Continuous Observations and Actions.. In *Conference on Uncertainty in Artificial Intelligence*.

White, A. (2015). *Developing a predictive approach to knowledge*. Ph.D. thesis, University of Alberta.

White, M. (2017). Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*.

Williams, R. J., & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.. *Neural Computation*.

Wingate, D., & Singh, S. (2006). Mixtures of predictive linear gaussian models for nonlinear, stochastic dynamical systems. In *AAAI Conference on Artificial Intelligence*.

Wolfe, B., & Singh, S. P. (2006). Predictive state representations with options.. In *International Conference on Machine Learning*.