Sparse Representation Neural Networks for Online Reinforcement Learning

by

Vincent Liu

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

Master of Science

in

Statistical Machine Learning

Department of Computing Science

University of Alberta

Abstract

In this thesis, we investigate sparse representations in reinforcement learning. We begin by discussing catastrophic interference in reinforcement learning with function approximation, and empirically investigating difficulties of online reinforcement learning in both policy evaluation and control. We then demonstrate that learning a control policy incrementally with a representation from a standard neural network fails in classic control domains, whereas learning with a sparse representation is effective. We provide evidence that the reason for this is that the sparse representation avoids catastrophic interference. Lastly, we discuss how to learn such sparse representations. We explore the idea of Distributional Regularizers, where the activation of hidden nodes is encouraged to match a particular distribution that results in sparse activation across time. We identify a simple but effective way to obtain sparse representations, not afforded by previously proposed strategies, making it more practical for further investigation into sparse representations for reinforcement learning.

Preface

This thesis is an original work by Vincent Liu. Parts of it are based on publication as Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White, "The Utility of Sparse Representations for Control in Reinforcement Learning". *In Proceedings of The Thirty-Third AAAI Conference on Artificial Intelligence*, 2019. Raksha Kumaraswamy and I developed the regularization method, designed and performed experiments, and wrote the paper together. Lei Le provided comments on the paper. Martha White was the supervisory author, provided comments and edited the paper for publication.

To my family.

Acknowledgements

I would like to thank my supervisor Martha White for her support. She has spent numerous hours discussing research with me, and deeply influenced me as a researcher. I would also like to thank Raksha Kumaraswamy. She helped me a lot when I started my journey in this field and always gave me useful suggestions. I am fortunate to meet a group of wonderful people #boba_guys: Wesley Chung, Sungsu Lim and Muhammad Zaheer. I am also thankful for my collaborators, friends and colleagues for their help: Yi Wan, Taher Jaferjee, Somjit Nath, Han Wang, Hengshuai Yao, Chenjun Xiao, Prof. Or Sheffet, Lei Le, Andrew Jacobsen, Roshan Shariff, Prof. Adam White, and Brendan Bennett. I would also like to thank all RLAI and AMII members for their help during the past two years.

Thank my parents and my sister for their encouragement and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Learning performance in artificial intelligence systems is highly dependent on the data representation. An effective representation captures important attributes of the state (or instance), as well as simplifies the estimation of predictors. Consider a reinforcement learning agent. A local representation enables the agent to more feasibly make accurate predictions for that local region, because the local dynamics are likely to be a simpler function than learning global dynamics. Additionally, such a representation can help prevent forgetting or interference [12, 33], by only updating local weights, as opposed to dense representations where any update would modify many weights. At the same time, it is important to have a distributed representation [7, 8], where the representation for an input is distributed across multiple features or attributes, promoting generalization and a more compact representation.

Such properties can be well captured by sparse representations: those for which only a few features are active for a given input (Figure 1.1). Enforcing sparsity promotes identifying key attributes, because it encourages the input to be well-described by a small subset of attributes. Sparsity, then, promotes locality, because local inputs are likely to share similar attributes (similar activation patterns) with less overlap to non-local inputs. In fact, many handcrafted features are sparse representations, including tile coding [51, 52], radial basis functions and sparse distributed memory [21, 42]. Other useful properties of sparse representations include invariance [17, 45]; decorrelated features [11]; improved computational efficiency for updating weights

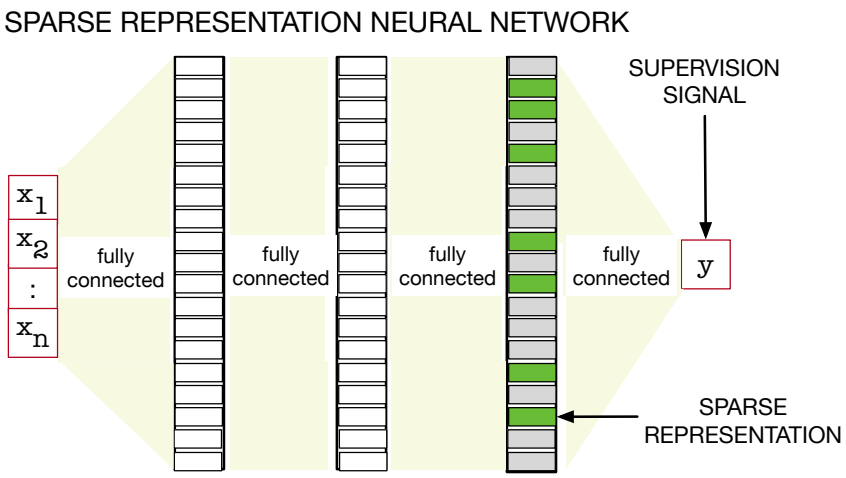


Figure 1.1: A neural network with dense connections producing a sparse representation: Sparse Representation Neural Network (SR-NN). The green squares indicate active (nonzero) units, making a sparse last hidden layer where only a small percentage of units are active. This contrasts a network with sparse connections—which is often also called sparse. Sparse connections remove connections between nodes, but are likely to still produce a dense representation.

in the predictor, as only weights corresponding to active features need to be updates; and enabling linear separability in the high-dimensional space [10], which facilitates the learning of a simple linear predictor. Further, sparse distributed representations have been observed in the brain [2, 38, 39].

Traditionally, sparse representations have been common for control in reinforcement learning, such as tile coding and radial basis functions [52]. They are effective for incremental learning, but can be difficult to scale to high-dimensional inputs because they grow exponentially with input dimension. Neural networks much more feasibly enable scaling to high-dimensional inputs, such as images, but can be problematic when used with incremental training. Instead, techniques like experience replay and target networks, inspired by batch methods such as fitted Q-iteration [43], have been necessary for many of the successes of control with neural networks. We provide some evidence in this paper that this modification is needed with dense, but not sparse, networks because of interference in the value function. Local representations, however, are much less likely to suffer from interference. Learned sparse representations, then, are a promising strategy to obtain the benefits

of previously common, fixed sparse representations with the scaling of neural networks.

Learning sparse representations, however, does remain a challenge. There have been some approaches developed to learning sparse representations incrementally, particularly through factorization approaches for dictionary learning [24, 29, 30] or for general sparse distributions [25, 37, 38, 40, 41, 53], like Boltzmann machines. In sparse coding, for example, the sparse representation learning problem is formulated as a matrix factorization, where input instances are reconstructed using a sparse, or small subset, of a large dictionary. Many of the methods for general sparse distribution, however, are expensive or complex to train and those based on sparse coding have been found to have serious out-of-sample issues [24, 26, 30].

There are fewer methods using feedforward neural network architectures. Certain activation functions—such as linear threshold units (LTU) [34] and rectified linear units (ReLU) [15]—naturally provide some level of sparsity, but provide no such guarantees. Early work on catastrophic interference investigated some simple heuristics for encouraging sparsity, such as node sharpening [12]. Though catastrophic interference was reduced, the resulting networks were still quite dense.[1] k-sparse auto-encoders [31] use a top-k constraint per instance: only the top k nodes with largest activations are kept, and the rest are zeroed. Winnner-Take-All auto-encoders [32] use a $k\%$ response constraint per node across instances, during training, to promote sparse activations of the node over time. These approaches, however, can be problematic—as we reaffirm in this work—because they tend to truncate non-negligible values or produce insufficiently sparse representations. Another line of work has investigated learning or specifying sparse activation functions for neural networks [3, 26, 41, 54], but used a sigmoid activation which is unlikely to result in sparse

[1] There have been strategies developed for catastrophic interference that rely on rehearsal or dedicating subparts of the network to particular tasks. This work is a complementary direction for understanding catastrophic interference for a sequential multi-task setting. We explore specifically the utility of sparse representations for alleviating interference for RL agents learning incrementally on one task, but do not necessarily imply that it is the only strategy to alleviate such interference. The comparisons in this work, therefore, focus on other strategies to learn sparse representations.

representations. They define sparsity based on norms of the vector, rather than activation level.

In summary, the main contributions of this dissertation are the following:

- We empirically demonstrate failures in online reinforcement learning with function approximation, more specifically, with representations produced by neural networks (Chapter 3).

- We demonstrate the utility of sparse representations for control in online reinforcement learning, and provide evidence that sparsity helps avoid interference (Chapter 4).

- We investigate a simple strategy to obtain sparse representations with neural networks: regularizing the distribution of activation to match a desired distribution. We propose a modification to the KL, called the Set-KL, and show that the resulting regularized objective produces effective sparse representations (Chapter 5).

Chapter 2

Background

In this chapter, we introduce some basics of reinforcement learning which are necessary to understand this thesis. We begin with a formal definition of a Markov Decision Process (MDP), and then present reinforcement learning algorithms for solving MDPs. Finally, we discuss linear function approximation and non-linear function approximation.

2.1 Markov Decision Process

In reinforcement learning (RL), an agent interacts with its environment, receiving observations and selecting actions to maximize a reward signal. We assume the environment can be formalized as a Markov decision process (MDP). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \Pr, R, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is an set of actions, $\Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and γ is the discount factor $\in [0, 1]$ which define the relative value of future rewards.

The agent's goal is to find a policy which maximizes the expected return. The return G_t is defined as the discounted sum of rewards:

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where R_t is the reward received at time t. A policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function mapping from the states to the actions[1]. Given a policy π, the value of a state

[1]For simplicity, we only consider deterministic policies here. In general, policies can be stochastic.

s is the expected return starting from state s and thereafter following policy π. Formally, we define the value function V^π by

$$V^\pi(s) := \mathbb{E}_\pi[G_t|S_t = s],\ \forall s \in \mathcal{S}$$

where $\mathbb{E}_\pi$ is the expectation given that actions are selected based on policy π. Similarly, the action-value of a state s and an action a is the expected return starting from state s, taking action a and following policy π. We define the action-value function by

$$Q^\pi(s,a) := \mathbb{E}_\pi[G_t|S_t = s, A_t = a],\ \forall s \in \mathcal{S} \text{ and } \forall a \in \mathcal{A}.$$

We also define the optimal value function as the best value that can be attained for any policy, i.e.,

$$V^*(s) := \max_\pi V^\pi(s),\ \forall s \in \mathcal{S}$$

and the optimal action-value function as

$$Q^*(s,a) := \max_\pi Q^\pi(s,a),\ \forall s \in \mathcal{S} \text{ and } \forall a \in \mathcal{A}.$$

A policy π is an optimal policy if it achieves the optimal action-value in all state action pairs, i.e.,

$$Q^\pi(s,a) = Q^*(s,a),\ \forall s \in \mathcal{S} \text{ and } \forall a \in \mathcal{A}.$$

We say that a policy π is greedy with respect to an action-value function Q, and write $\pi = \pi_Q$ if, for all $s \in \mathcal{S}$,

$$\pi_Q(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

It is known that a greedy policy with respect to Q^* is an optimal policy. Therefore, knowing Q^* is sufficient for behaving optimally.

2.2 Reinforcement Learning Algorithms

In practice, the transition probability and reward function are usually unknown. The agent has to directly estimate Q^π from experience—a sequence

of states, actions and rewards. Here we describe three reinforcement learning algorithms: *Monte Carlo Methods*, *Sarsa* and *Q-Learning*.

Monte Carlo methods estimate the action-value function from sample returns, and improve the policy with respect to the estimated action-value function. We use Q to denote the estimated action-value function, where the value of each state-action pair can be separately stored. The agent executes its current policy until an episode ends, and updates the estimated action-value function for all state action pairs in the episode according to

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)]$$

where α is a step size and G_t is the actual return following time t.

Monte Carlo is an offline method, where all updates are performed at the end of an episode. On the other hand, Sarsa and Q-learning are online methods where an update is performed at each time step. The idea is based on *temporal difference learning* [50], which bootstraps from the current estimate as a target.

Sarsa updates the action-value function for the current behavior policy by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

and it acts near-greedily according to the estimated action-value function. It is an on-policy control method since it estimates the action-value for its behavior policy.

Q-learning [56] is an off-policy control algorithm. It directly estimates the optimal action-value function while following a behavior policy π. The update rule is

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

Q-learning converges to the optimal action-value Q^* with probability 1, under certain assumptions on the behavior policy and the sequence of step-size. In practice, we can use an ϵ-greedy policy as the behavior policy. The agent selects an action randomly with probability ϵ and selects a greedy action with respect to the current action-value estimate with probability $1 - \epsilon$.

2.3 Linear Function Approximation

In the previous section, we assumed that the state space was finite and the value of each state can be separately stored. These are called *tabular methods*. We can extend these methods to very large or continuous state space with *function approximation*. In such cases we seek to find a good function approximation to approximate the value functions.

The simplest function approximation is a linear function, where the true value function is approximated by a linear function parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^d$:

$$Q^\pi(s,a) \approx Q_{\mathbf{w}}(s,a) := \boldsymbol{\phi}(s,a)^\top \mathbf{w} \tag{2.1}$$

where $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ is the *representation* of a state action pair.

Similar to the tabular methods, we have Monte Carlo update:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha[G_t - \boldsymbol{\phi}(S_t, A_t)^\top \mathbf{w}_t]\boldsymbol{\phi}(S_t, A_t),$$

semi-gradient Sarsa update:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha[R_{t+1} + \gamma\boldsymbol{\phi}(S_{t+1}, A_{t+1})^\top \mathbf{w}_t - \boldsymbol{\phi}(S_t, A_t)^\top \mathbf{w}_t]\boldsymbol{\phi}(S_t, A_t),$$

and semi-gradient Q-learning's update:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha[R_{t+1} + \gamma \max_a \boldsymbol{\phi}(S_{t+1}, a)^\top \mathbf{w}_t - \boldsymbol{\phi}(S_t, A_t)^\top \mathbf{w}_t]\boldsymbol{\phi}(S_t, A_t).$$

The Sarsa and Q-learning updates are not in fact a true gradient descent method since they ignore the effect on the target when changing the weight vector. Therefore, we refer to such updates as *semi-gradient methods*.

There are various methods to build a representation function, including Polynomials, Fourier basis, coarse coding and tile coding [52]. Tile coding uses overlapping grids to partition the state space, to convert continuous states to binary feature vectors. Each partition is called a tiling, and each element of a partition is called a tile. The representations generated by it are sparse and distributed based on a static hashing technique.

2.4 Non-linear Function Approximation with Neural Networks

Linear function approximation with tile coding has made great success in solving reinforcement learning tasks [51]. However, designing good handcrafted features for linear functions requires non-trivial effort, especially for high-dimensional state space. Neural networks enable automating feature extraction from data. They are an approach to non-linear function approximation in reinforcement learning. This combination is usually called *deep reinforcement learning*.

A neural network typically uses a linear layer on the last layer, hence we can view a value function as a two-part approximation with a representation function and a linear weight:

$$Q^\pi(s,a) \approx Q_{\mathbf{w},\boldsymbol{\theta}}(s,a) := \boldsymbol{\phi}_{\boldsymbol{\theta}}(s,a)^\top \mathbf{w} \tag{2.2}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the weights in the last layer and $\boldsymbol{\phi}_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ is the *representation* function learned by the network with weights $\boldsymbol{\theta}$, composed of all the hidden layers in the network. The function $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s,a)$ corresponds to the last layer in the network, with $\boldsymbol{\theta}$ the weights of the network. The efficacy of the action-value approximation, therefore, relies on the representation $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s,a)$.

In most deep reinforcement learning algorithms, the representation and weight are learned concurrently. Let $\beta = [\mathbf{w}, \boldsymbol{\theta}]$, the online update for non-linear semi-gradient Q learning is

$$\beta_{t+1} \leftarrow \beta_t + \alpha[R_{t+1} + \gamma \max_a Q_{\beta_t}(S_{t+1},a) - Q_{\beta_t}(S_t,A_t)]\nabla_{\beta_t} Q_{\beta_t}(S_t,A_t).$$

where the gradient is a vector of first partial derivatives

$$\nabla_{\beta_t} Q_{\beta_t}(S_t,A_t) = (\frac{\partial Q_{\beta_t}(S_t,A_t)}{\partial \beta_{t_1}}, ..., \frac{\partial Q_{\beta_t}(S_t,A_t)}{\partial \beta_{t_N}}).$$

Recently, Mnih *et al.* [35] proposed the Deep-Q-Network (DQN), which uses experience replay [27] and a target network. It stores transitions in a replay buffer D when interacting with the environment. When it performs an update, it samples a mini-batch of transitions $\{(S_i, A_i, R_i, S_i')\}_{i=1}^B$ from D, and

optimizes the mean squared error between Q function and fixed targets:

$$\beta_{t+1} \leftarrow \beta_t + \alpha \frac{1}{B} \sum_{i=1}^{B} [(R_i + \gamma \max_{a'} \bar{Q}(S'_i, a') - Q_{\beta_t}(S_i, A_i)) \nabla_{\beta_t} Q_{\beta_t}(S_i, A_i)]$$

where $\bar{Q}$ is the target network which are updated periodically. These two techniques have been found to stabilize the training process [35, 43] and are widely used in most deep reinforcement learning algorithms.

Alternately, the representation parameters $\boldsymbol{\theta}$ and linear weight $\mathbf{w}$ can be optimized with different objectives [9], or the representation can be pre-trained and fixed while optimizing for the linear weight [6, 24]. In the later case, the approximation reduces to linear function approximation with a fixed representation $\boldsymbol{\phi}_{\boldsymbol{\theta}}$.

Chapter 3

Interference in Online Reinforcement Learning

In online reinforcement learning, an agent updates its value function or policy at each time step t. Online updating uses limited memory and computation, and can quickly react to new information. It is one of the important characteristics of temporal difference learning [50]. We are particularly interested in a fully online RL algorithm: updating the value function based on the current transition $\{S_t, A_t, R_t, S_{t+1}\}$ at each time step. A successful example of such online RL algorithms with function approximation is tile coding, which has been shown to solve benchmark RL tasks [51]. On the other hand, non-linear function approximation with neural networks often fails to solve simple RL tasks in a fully online setting [14].

When neural networks are used for function approximation, an update on the weights is likely to change the function globally. After seeing a sequence of new examples, the agent might forget previously learned information. This issue is referred to as *catastrophic interference* [13, 33, 49], and previous works on catastrophic interference focus on sequential supervised learning and multi-task learning [16, 23, 28, 44]. In this chapter, we investigate this phenomenon in online RL algorithms with function approximation in a single-task setting.

3.1 Defining Interference in RL

Interference is tied to generalization in function approximation. Therefore, in this section, we first study generalization between a pair of examples. Intuitively, when an agent performs an update based on an example, the update can generalize to another example positively (positive generalization), negatively (interference), or no effect (no generalization).

Let's consider a simple case: generalization in an estimated action-value function for policy evaluation. Given a policy π, we denote Q^π as the true action-value function and $Q_{\boldsymbol{\theta}}$ as the estimated action-value function, parameterized by the parameter $\boldsymbol{\theta}$. A natural objective function to minimize is the Mean Squared Value Error (MSVE) [52], defined as

$$J(\boldsymbol{\theta}) := \mathbb{E}_{s\sim d(\cdot),a\sim\pi(\cdot|s)}[(Q^\pi(s,a) - Q_{\boldsymbol{\theta}}(s,a))^2]$$

where d is the stationary distribution induced by the policy π.

In RL, we use some approximation to the true value as our target output. We denote the target output of the tth example by $U_t \in \mathbb{R}$. When we perform an update on the tth example (s_t, a_t), the update is computed by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha[U_t - Q_{\boldsymbol{\theta}_t}(s_t,a_t)]\nabla_{\boldsymbol{\theta}_t} Q_{\boldsymbol{\theta}_t}(s_t,a_t)$$

where α is the step size. The *pairwise interference* between two examples (s_t, a_t) and (s_i, a_i) can be defined as change in the objective function for the example (s_i, a_i) given the update based on the example (s_t, a_t):

$$PI(\boldsymbol{\theta}_t; (s_t,a_t), (s_i,a_i)) := J(\boldsymbol{\theta}_{t+1}; s_i, a_i) - J(\boldsymbol{\theta}_t; s_i, a_i). \tag{3.1}$$

If PI is negative, the error decreases for the example (s_i, a_i) and positive generalization occurs. Negative PI corresponds to a measure of positive generalization. If PI is positive, the error increases and interference occurs. The update based on example (s_t, a_t) results in unlearning of the example (s_i, a_i). Note that in tabular methods, there is no generalization between states so $PI = 0$ for any pair of examples.

Equation (3.1) can be approximated by a Taylor expansion:

$$
\begin{aligned}
&PI(\boldsymbol{\theta}_t; (s_t, a_t), (s_i, a_i)) \\
&\approx (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \frac{\partial J(\boldsymbol{\theta}_t; s_i, a_i)}{\partial \boldsymbol{\theta}_t} \\
&= 2\alpha [U_t - Q_{\boldsymbol{\theta}_t}(s_t, a_t)][Q^\pi(s_i, a_i) - Q_{\boldsymbol{\theta}_t}(s_i, a_i)] \nabla_{\boldsymbol{\theta}_t} Q_{\boldsymbol{\theta}_t}(s_t, a_t)^\top \nabla_{\boldsymbol{\theta}_t} Q_{\boldsymbol{\theta}_t}(s_i, a_i).
\end{aligned} \tag{3.2}
$$

This equation provides some insight into previous analysis of generalization in function approximation. The term $\nabla_{\boldsymbol{\theta}_t} Q_{\boldsymbol{\theta}_t}(s_i, a_i)^\top \nabla_{\boldsymbol{\theta}_t} Q_{\boldsymbol{\theta}_t}(s_t, a_t)$ is the neural tangent kernel (NTK) [20] of the Q function, which has been used to analyze generalization in the Q function across state-action pairs [1]. However, to determine whether positive generalization or interference occurs, we also need to know $[U_t - Q_{\boldsymbol{\theta}_t}(s_t, a_t)]$ and $[Q^\pi(s_i, a_i) - Q_{\boldsymbol{\theta}_t}(s_i, a_i)]$.

The PI is difficult to measure since it requires the true values. Nonetheless, it provides a way to think about interference in reinforcement learning. When we update the value of a state which has high PI with many other states, the update might cause instability in training. The PI informs the hypothesis in the next section.

3.2 Hypothesis about Interference in RL

Generalization has been extensively studied in supervised learning. We normally assume that we have a training set $D = \{(s_t, a_t), u_t\}_{t=1}^n$, where each input (s_t, a_t) is an identically and independently distributed (i.i.d.) sample from a fixed input distribution and the target output u_t is sampled from a fixed conditional distribution. However, in online reinforcement learning, we face new challenges for both prediction and control:

1. When an agent interacts with an environment, it receives a sequence of observations, which are likely to be temporally correlated.

2. The agent updates the value estimate based on one current sample, which might result in high variance of the gradient [55].

3. The agent uses its own estimates as targets, which depend on the parameters and hence are non-stationary.

Therefore, we aim to investigate the following hypothesis:

> *In a setting with significant pairwise interference, any of the three issues mentioned above results in instability in training and failure to learn accurate value estimates.*

3.3 Testing the hypothesis

We test our hypothesis in linear function approximation with a learned representation from a neural network. We design experiments to remove each issue individually with a different algorithm, and report the performance in both prediction and control.

3.3.1 Experimental setup

The experimental setup is as follows. We first pre-train a neural network on a batch of data to extract a representation, to be used for prediction or control. Learning a good representation $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s, a)$ in the case of finite actions can be transformed to learning a good representation of the form $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$, and using that to represent the action-value function as:

$$Q_{\mathbf{w},\boldsymbol{\theta}}(s, a) := \boldsymbol{\phi}_{\boldsymbol{\theta}}(s)^\top \mathbf{w}_a. \tag{3.3}$$

Here, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$ is the representation of the state s, which is used in conjunction with the linear predictor $\mathbf{w}_a$ to estimate action-values for action a across the state space. Note that this decomposition does not allow generalization over actions.

This learned representation is then fixed, and used for estimating action-values in prediction and control. We use a fully online Sarsa algorithm for learning the action-value function and a fully online Q-learning for learning the optimal action-value function, where only the weights $\mathbf{w}$ on the last layer are updated. Additional details on this experiment are provided in Section 3.4.

We choose this two-stage training regime to remove confounding factors in difficulties of training neural networks incrementally and also to keep a con-

sistent experimental setup with the next chapter. Our goal here is to identify the issues in learning value function online, not necessarily in learning the neural network. The networks are trained with an objective for learning values, on a large batch of data generated by a policy that covers the input space; the learned representations are capable of representing the true values. We investigate their utility for online learning. Outside of this carefully controlled experiment, we advocate for learning the representation incrementally, for the task faced by the agent.

An easy objective to train connectionist networks with simple backpropagation is the Mean Squared Temporal Difference Error (MSTDE) [50]. For a given policy, the MSTDE is defined as:

$$\mathbb{E}_{s\sim d(\cdot)}[(R_{t+1} + \gamma_{t+1}\boldsymbol{\phi}_{\boldsymbol{\theta}}(S_{t+1})^\top \mathbf{w}_v - \boldsymbol{\phi}_{\boldsymbol{\theta}}(S_t)^\top \mathbf{w}_v)^2 | S_t = s] \tag{3.4}$$

where $\gamma_{t+1} = \gamma$ if S_{t+1} is not a terminal state and $\gamma_{t+1} = 0$ if S_{t+1} is a terminal state. Here, d denotes the stationary distribution over the states induced by the given policy, and $\boldsymbol{\theta}$ and $\mathbf{w}_v$ are parameters that can be learned with stochastic gradient descent. Therefore, given experience generated by a policy that explores sufficiently in an environment, a strong function approximator (a dense neural network) can be trained to estimate useful features, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$.

3.3.2 Algorithms

To remove the issues of (1) correlated samples; (2) high variance of gradient based on one sample; and (3) non-stationarity of the data distribution, we employ (1) experience replay; (2) mini-batch updates; and (3) target networks. These are commonly-used techniques to enable online learning. Experience replay provides decorrelated samples from past experience and sampling from a replay buffer can be seen as sampling i.i.d. data from a set of transitions following mixed policies. Mini-batch updates average the gradients over a mini-batch of samples to reduce the variance of the update. Target networks reduce the non-stationarity of target outputs during training by updating the target outputs periodically.

We test all combinations of these three techniques, denoted as

1. *Buffer-Batch*: mini-batch updates sampling from a buffer.

2. *Batch*: mini-batch updates with recent samples.

3. *Buffer*: stochastic updates with a single example sampling from a buffer.

4. *Online*: update with an online sample.

5. *Target-Buffer-Batch*: Buffer-Batch plus target network.

6. *Target-Batch*: Batch plus target network.

7. *Target-Buffer*: Buffer plus target network.

8. *Target-Online*: Online plus target network.

3.3.3 Prediction

We report the Root Mean Squared Value Error (RMSVE) for predicted action-values, which is evaluated as follows:

$$\text{RMSVE} = \sqrt{\frac{\sum_{(s,a)\in D_{test}} (\hat{Q}(s,a) - Q^{\pi}(s,a))^2}{D_{test}}} \tag{3.5}$$

where D_{test} is a set of state and action pairs, $\hat{Q}$ is the estimated value function and $Q^{\pi}(s,a)$ is the true value function computed using Monte Carlo rollouts [46].

Figure 3.1 shows the learning curve with a constant step-size Sarsa in three benchmark domains: Mountain Car, Puddle World and Acrobot. Buffer-Batch achieves the lowest testing error across all domains, but using either Batch or Buffer is not as effective. Target-Batch also learns good predictions in Puddle World and Acrobot. The target network in general seems to provide little help and largely slows down learning. Overall, the result suggests that the learned representation is able to represent accurate action-values, however, learning fully incrementally (without i.i.d. samples and mini-batch update) with such representation is less effective.

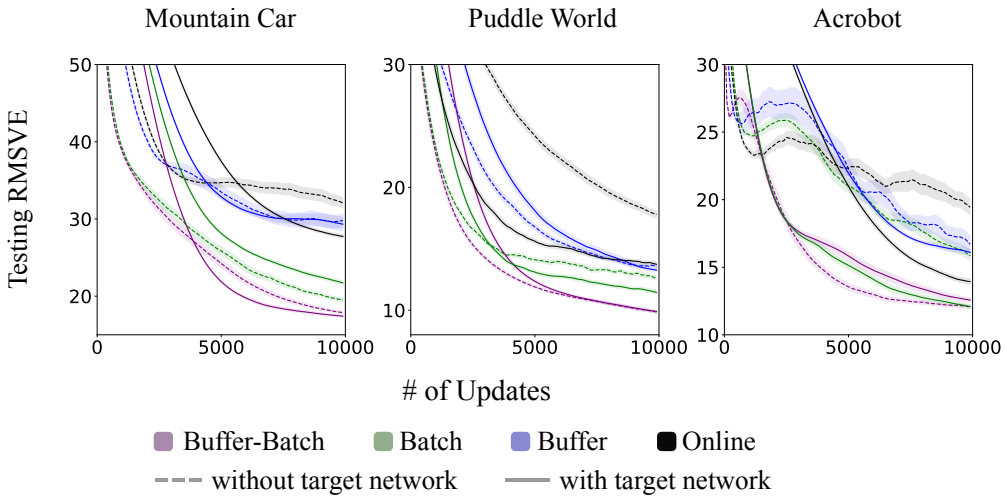


Figure 3.1: Learning curves for Sarsa(0). All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$) and one standard deviation of the average.

3.3.4 Control

Figure 3.2 shows the learning curve with a constant step-size Q-learning for control. Again, Buffer-Batch is important to obtain good performance. Target-Batch performs well in two domains but fails in Puddle World. Target network seems to be a critical technique here. We think the reason is that target distribution is much more unstable in control, where the policy changes at each time step. Hence, using stationary targets can significantly stabilize the training.

In summary, the results suggest that online RL, even with linear function approximation, suffers from interference due to non-i.i.d. samples, small mini-batch size, and non-stationary target distribution. Techniques like experience replay and target networks, hence, have been needed for many reinforcement learning algorithms [35, 43].

However, recall that our hypothesis states that these three issues are a problem under significant pairwise interference. An orthogonal approach to mitigate this problem is to reduce interference in function approximation. We discuss a potential strategy to decrease interference in the next chapter.

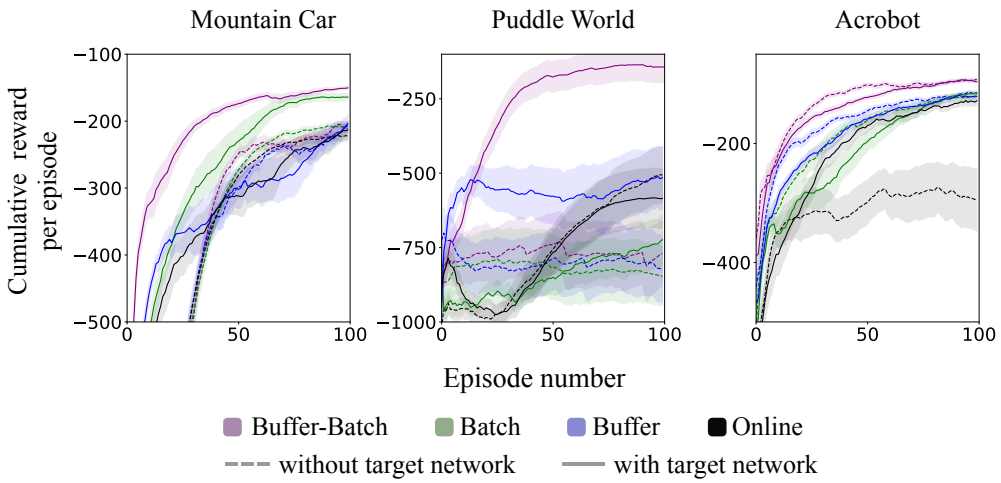


Figure 3.2: Learning curves for Q-learning. All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$) and one standard deviation of the average.

3.4 Experimental Details

Policies to generate training and testing data In Mountain Car, we use the standard energy pumping policy with 10% randomness. In Puddle World, by a policy that chooses to go North with 50% probability and East with 50% probability on each step. The data in Acrobot is generated by a near-optimal policy. We use these policies to generate training samples and testing samples. For Mountain Car, Puddle World and Acrobot, we have 1k episode cut-off. All domains are episodic, with discount set to 1 until termination.

Representation Learning Phase We used neural networks with two hidden layers. The first layer 32 hidden units. The second layer, which is the representation layer used for prediction, has 256 units. In representation learning phase, we optimized the neural network weights using Adam optimization [22] with a batch size of 64 and learning rate 0.001. The neural network weights are initialized based on He initialization [18]. That is, the neural networks weights are initialized with zero-mean Gaussian distribution with variance equals to $2/n_l$, where n_l is the number of input nodes for layer l.

Prediction & Control Performance The learned representations are then used for policy evaluation and control. The value function weight $\mathbf{w}$ is initialized with zero-mean Gaussian distribution with small variance. We use semi-gradient Sarsa and Q-learning with fixed learning rate, which is swept in the set:

$$\alpha \in \{0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001\}.$$

The range of grid search for other hyperparameters are as follows:

$$\text{target network update frequency} \in \{25, 50, 100, 200, 400\}$$
$$\text{buffer size} \in \{1000, 10000\}.$$

We fix the mini-batch size of 64 for Buffer-Batch and Batch. All the sweeps for selecting the hyperparameters across domains use 10 runs. We report the best hyperparameters based on the final RMSE for prediction and average return over 100 episodes for control.

Chapter 4

The Utility of Sparse Representation for Control

In this chapter, we highlight the utility of sparsity for control in an incremental setting. We show that two sparse representations—tile coding and sparse representation learned by a neural network (referred to as *SR-NN* from here-on)—both significantly improve stability and performance in control. We choose tile coding, a static representation, as a baseline to compare to, as it known to perform very well in the benchmark RL domains we experiment with [52]. We use a regularization method, described in a later chapter, to learn SR-NN. It is a more specific model than the SR-NN introduced in Figure 1.1. We provide the full details in Chapter 5.

We hypothesize that interference is much less problematic for representations which generalize locally, which are typically provided by sparse representations. We show that agents that learn sparse representations perform better, providing some evidence for this hypothesis.

4.1 Control Performance with SR-NN

We evaluate control performance on four domains: Mountain Car, Puddle World, Acrobot and Catcher. We choose these domains because they are well-understood, and typically considered relatively simple. A priori, it would be expected that a standard action-value method, like Sarsa, with a two-layer neural network, should be capable of learning a near-optimal policy in all four

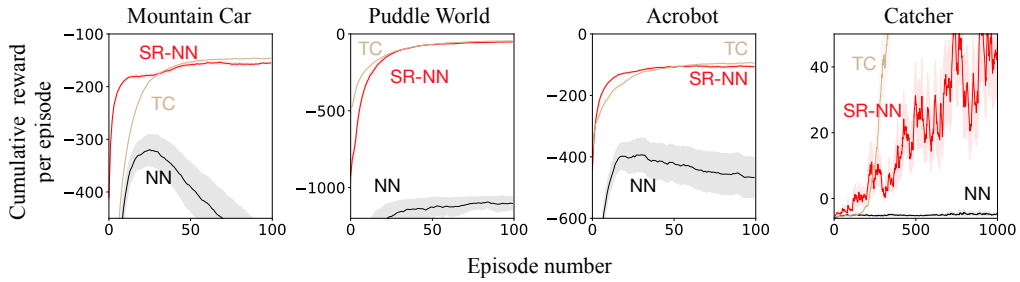


Figure 4.1: Learning curves for Sarsa(0) comparing SR-NN, Tile Coding and vanilla NN in the four domains. All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

of these domains.

The experimental setting is the same as the one in the previous chapter. We pre-train a representation with a MSTDE objective given a batch of data. The learned representation is then fixed, and used for estimating action-values in control for learning the (near) optimal behaviour policy in the environment. We use a fully online Sarsa algorithm for learning the control policy, where only the weights $\mathbf{w}$ on the last layer are updated. Both SR-NN and NN used two-layers, of size 32 and 256 respectively, with ReLU activations. The meta-parameters for the batch-trained neural network producing the representation and the control agent were swept in a wide range, and chosen based on control performance. The aim is to provide the best opportunity for a regular feed-forward network (NN) to learn on these problems, as it is more sensitive to its meta-parameters than the SR-NN. Additional details on hyper-parameter optimization are provided in Section 4.4.

The learning curves for the four domains, with Tile-Coding (TC), SR-NN and NN, are shown in Figure 4.1. The NNs performs surprisingly poorly, in some case increasing and then decreasing in performance (Mountain Car), and in others failing altogether (Catcher). In all the benchmark RL domains, the baseline sparse representation, TC, performs well and learns a nearly optimal policy, as expected. The learned SR-NN performs as well in all domains, and is effective for learning in Catcher, whereas NN performs really poorly in all domains, and does not learn anything in Catcher. Both SR-NN and

NN representations were trained in the same regime, with similar representational capabilities. In fact, the NN is the same representation from Figure 3.2, where it is able to learn a good policy with ER and target network. The representational ability of the NN is therefore not the problem. Yet, SR-NN enables the Sarsa(0) agent to learn, where the regular feed-forward NN does not. We investigate this effect further in the next sets of experiments, to better understand the phenomenon.

4.2 The Effect of Regularization

Overfitting is a problem for neural networks with limited training data [8]. In our experiments, representations are pre-trained with a fixed number of training data, hence, the learned representation can overfit and not generalize effectively. To determine if the main impact of the sparse representation is simply from regularization, preventing overfitting, we tested several regularization strategies for the neural network. These include ℓ_2 and ℓ_1 on the weights of the network (ℓ_2-NN and ℓ_1-NN respectively) and Dropout on the activation (Dropout-NN) [48]. The ℓ_1 regularizer encourages weights to go to zero, reducing the number of connections, but does not necessarily provide a sparse representation. The Dropout regularizer randomly zeros out activation in hidden layers during training, but does not necessarily provide a sparse representation during inference.

In Figure 4.2, we can see that regularization is unlikely to account for the improvements in control. SR-NN performs well across all domains, whereas none of the regularization strategies consistently perform well. ℓ_1-NN and ℓ_2-NN perform well in Mountain Car during early learning, but fail in other domains. Dropout-NN performs poorly in all domains.

Additionally, we show the learning curve during representation learning phase in Figures 4.3. The metric on the y-axis is the RMSVE for predicted state-values evaluated on a set of test states, similar as Equation (3.5) for action-values. The number of test states are 5000 for benchmark domains and 1000 for Catcher. Most algorithms converge within 50 epochs in Mountain Car,

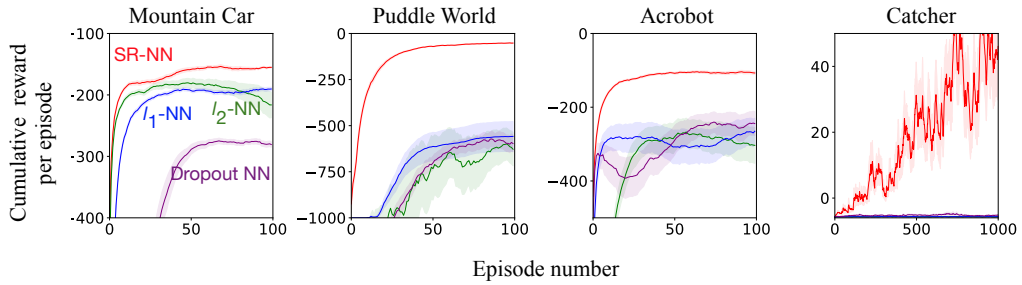


Figure 4.2: Learning curves for Sarsa(0) comparing SR-NN to the regularized representations. All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

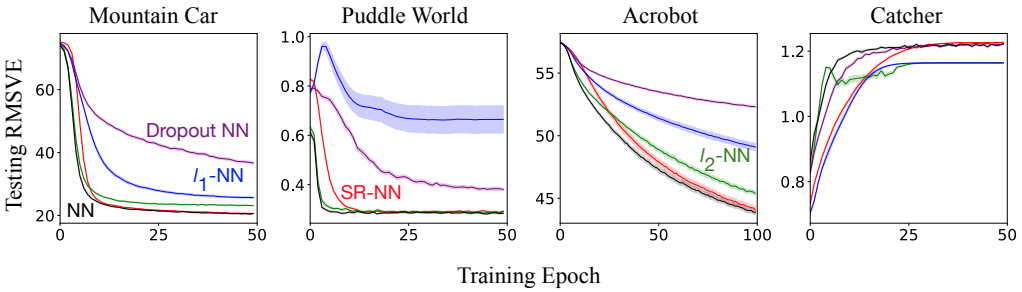


Figure 4.3: Learning curve during the representation learning phase with regularization methods. All curves are averaged over 30 runs.

Puddle World and Catcher, and 100 epochs in Acrobot as shown in the figures. Note that these representations are optimized for the MSTDE objective, not the RMSE objective. Nevertheless, these curves show the predictive ability of these learned representations. Interestingly, NN achieves the lowest testing error across all domains and regularization methods, especially Dropout-NN and ℓ_1-NN, seem to have higher testing error.

4.3 Evaluation of Learned Representations

We next investigate the hypothesis that sparse representations provide locality, which should help mitigate interference. Recall in the previous chapter, the magnitude of generalization in an estimated Q function for two examples $\mathbf{x}_i$ and $\mathbf{x}_j$ is proportional to the dot product of two gradients:

$$|PI(\mathbf{w}; (\mathbf{x}_i, a), (\mathbf{x}_j, a))| \propto |\nabla Q_{\mathbf{w}}(\mathbf{x}_i, a)^\top \nabla Q_{\mathbf{w}}(\mathbf{x}_j, a)| = |\boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j)|.$$

If two representations are orthogonal, interference would be zero. Updating the value function with respect to one state, therefore, would not affect the other state's value. Moreover, if the representations are non-negative, these representation are likely to be sparse and have little shared activation between samples.

Activation overlap, introduced by [12], reflects the amount of shared activation between any two inputs. We consider two variants of activation overlap. The first one measures the number of shared activation between two representations, $\boldsymbol{\phi}(\mathbf{x}_1)$ and $\boldsymbol{\phi}(\mathbf{x}_2)$, for two samples, $\mathbf{x}_1$, and $\mathbf{x}_2$:

$$\text{overlap}(\boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)) = \sum_k \mathbb{1}[(\boldsymbol{\phi}_k(\mathbf{x}_i) > 0) \wedge (\boldsymbol{\phi}_k(\mathbf{x}_j) > 0)],$$

where $\boldsymbol{\phi}_k(\mathbf{x})$ is the k-th component of the feature vector, and the second one measures the orthogonality of two representations:

$$\text{orthogonality}(\boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)) = \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j) / \left\|\boldsymbol{\phi}(\mathbf{x}_i)\right\|_2 \left\|\boldsymbol{\phi}(\mathbf{x}_j)\right\|_2 .$$

We normalize the magnitude by the norm of the representations. Note that the dot product is always non-negative because of ReLU activation. This orthogonality measure is the cosine similarity between two gradients, which has been used to quantify the degree of interference [47]. It is also similar to the ratio of the off-diagonal entry to the on-diagonal entry in the NTK [1]. We report the average overlap and average orthogonality over a subset of states S equally covering the state space in Puddle World and Mountain Car. Specifically, $S = \{(-1.2 + 0.17x, -0.07 + 0.014y) : x, y = 0, 1, 2, ..., 10\}$ in Mountain Car and $S = \{(0.1x, 0.1y) : x, y = 0, 1, 2, ..., 10\}$ in Puddle World.

Table 4.1 shows the activation overlap, and once again, SR-NN has less overlap and its feature vectors are more orthogonal. In Puddle World, the feature vectors of SR-NN are much more orthogonal than other representations, and correspondingly, SR-NN is the only one which performs well in the domain. In Mountain Car, the difference in orthogonality between SR-NN and other representations is not significant, and they seem to have similar control performance.

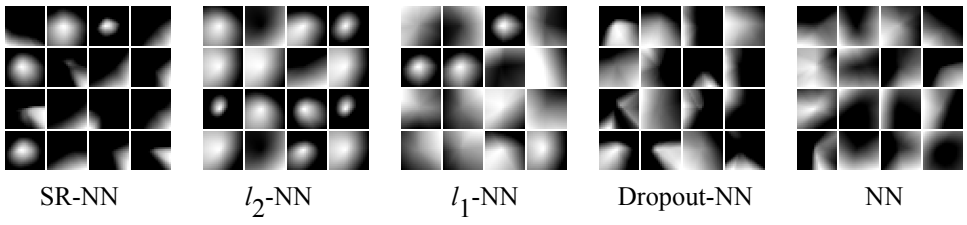
Mountain Car	SR-NN	ℓ_2-NN	ℓ_1-NN	Dropout-NN	NN
Overlap	**12.23**	90.18	122.19	74.65	109.64
Orthogonality	**0.66**	0.71	0.85	0.76	0.68
Non-dead Neurons	123.73	144.67	135.30	240.77	212.73

Puddle World	SR-NN	ℓ_2-NN	ℓ_1-NN	Dropout-NN	NN
Overlap	**15.59**	80.34	147.32	33.68	56.43
Orthogonality	**0.23**	0.77	0.95	0.48	0.80
Non-dead Neurons	216.60	147.80	147.43	216.30	176.60

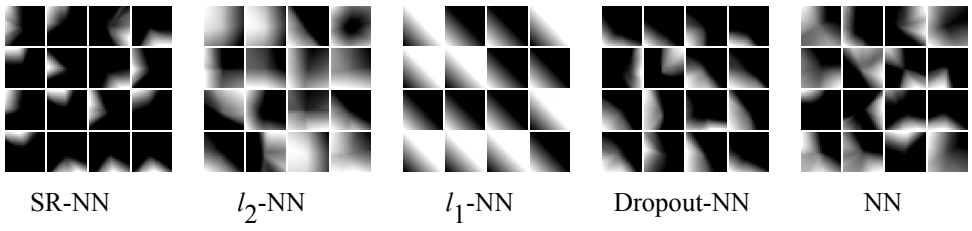
Table 4.1: Activation overlap in Mountain Car and Puddle World. Overlap and orthogonality are averaged over all state pairs. Non-dead neurons are the number of features which are non-zero for some states. The numbers are averaged over 30 runs. The standard errors are small (less than 1 for Overlap and Non-dead Neurons, and less than 0.01 for Orthogonality) so we do not report the error bars here.

To show the locality of the learned representations, we visualize the learned representations. Figure 4.4 shows the activation map of randomly selected hidden neurons with the different networks. We can see that each hidden neuron in SR-NN only responds to a local region of the input space, while some hidden neurons in NN respond to a large part of the space. Consequently, when one state is updated in a part of the space with the NN representation, it is more likely to significantly shift the values in other parts of the space, as compared to the more local SR-NN. The ℓ_2-NN, and ℓ_1-NN representations do not exhibit any discernible locality properties. Dropout-NN does achieve some degree of locality in Puddle World.

We also report a measure of sparsity, called *instance sparsity*, to determine if the successful methods are indeed sparse. Instance sparsity corresponds to the percentage of active units for each input. A sparse representation should be instance sparse, where most inputs produce relatively low percentage activation. As shown in Figure 4.5, SR-NN has consistently low instance sparsity across all four domains. Dropout-NN appears to have learned a sparse representation in Puddle World. It has been observed that Dropout can at times learn sparse representations [5], but not consistently, as corroborated by our experiments. The NN representation, which has no regularization, has some



(a) Mountain Car



(b) Puddle World

Figure 4.4: The activation maps for 16 randomly chosen neurons for different representations—each cell in the heatmap corresponds to the complete 2D state space.

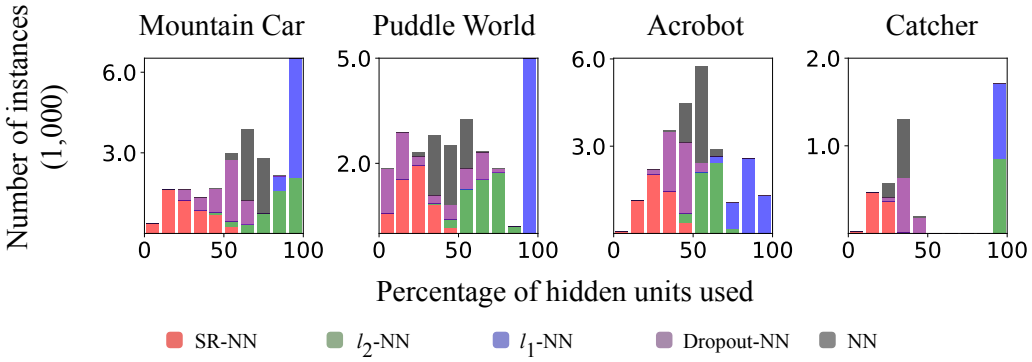


Figure 4.5: Instance sparsity comparing SR-NN to the regularized variants and vanilla NN. The percentage evaluation is designed to disregard units that are never active across all samples in the batch (dead units). The numbers are averaged over 30 runs.

instance sparsity, likely due to simply using ReLU activation. Interestingly, ℓ_1-NN and ℓ_2-NN actually produced less instance sparsity in some domains.

Overall, these results provide some evidence that (a) sparse representations can improve control performance in an incremental learning setting, (b) these sparse representations appear to provide locality and orthogonal feature vector. These results are a first step, and warrant further investigation. They do nonetheless motivate that learning sparse representations could be a promising direction for control in reinforcement learning.

4.4 Experimental Details

Policies to generate training and testing data For Mountain Car, Puddle World and Acrobot, the policies are described in the previous chapter. In Catcher, the agent chooses to move toward the apple with 50% probability, and selects a random action with 50% probability on each step; and gets only 1 life in the environment. We use these policies to generate training samples and testing samples. For Catcher, we have a 10k episode cut-off. All domains are episodic, with discount set to 1 until termination.

Tile coding hyperparameter We compare to Tile Coding (TC) representation, a well-known sparse representation, as the baseline. We experiment with several configurations for the fixed representation:

$$\text{number of tiles (N)} \in \{4, 8, 16\}$$
$$\text{number of tilings (D)} \in \{4, 8, 16\}.$$

We use a hash size of 4096, which is significantly larger than the feature size of 256, as used in the other learned representation models we compare to. The results shown in Figure 4.1 are for the best configuration of the static tile-coder after a sweep.

Neural networks hyperparameters The range of grid search for the representation hyperparameters are as follows:

$$\lambda_{KL} \in \{0.1, 0.01, 0.001\} \qquad \text{[SR-NN]}$$

$$\beta \in \{0.05, 0.1\} \qquad \text{[SR-NN]}$$

$$\lambda_{NN} \in \{0.1, 0.01, 0.001, 0.0001\} \qquad [\ell_1\text{-NN and } \ell_2\text{-NN}]$$

$$\text{dropout probability } p \in \{0.1, 0.2, 0.3, 0.4, 0.5\} \qquad \text{[Dropout-NN]}$$

For dropout, given the form of the supervision goal (MSTDE), the same dropout mask is chosen to generate the representation for both states S_{t+1} and S_t – this preserves dropouts role as regularizer w.r.t. the target. We have also experimented with different dropout masks for S_{t+1} and S_t, and the result suggests that it is not able to learn good representations even for prediction across all domains. For ℓ_1 and ℓ_2, regularization is not applied to the bias units.

Control Performance The learned representations are used for control with fixed $\epsilon = 0.1$. The value function weight $\mathbf{w}$ is initialized with zero-mean Gaussian distribution with small variance. For sparse representations, we use semi-gradient Sarsa or Q-learning with fixed learnining rate. For dense representations, we use adaptive learning rate method RMSprop [19] with $\alpha = 0.9$. We do not use

The initial learning rate is swept in the set:

$$\alpha \in \{0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00001\}$$

All the sweeps for selecting the representation learning hyperparameters across domains use 10 runs. We report the best hyperparameters based on average return over episodes.

Chapter 5

Learning Sparse Representations with Neural Networks

In this chapter, we describe how to use Distributional Regularizers to learn sparse representations with neural networks. The idea was originally introduced for neural networks with Sigmoid activations in an unpublished set of notes [36], and as yet has not been systematically explored. When used out-of-the-box, we found important limitations in the learned representations, including from using Sigmoid activations instead of ReLU and from using the KL to a specific distribution. We explore the idea in-depth here, to make it a practical option for learning sparse representations. Besides, we introduce a Set Distributional Regularizer, which when paired with ReLU activations enables sparse representations to be learned, as we demonstrate in the experiments. We first describe how to define Distributional Regularizers on neural networks, and then discuss the extension to a Set Distributional Regularizer, and motivation for doing so.

5.1 Distributional Regularizers for Sparsity

The goal of using Distributional Regularizers is to encourage the distribution of each hidden node—across samples—to match a desired target distribution. In a neural network, we can view the hidden nodes, $Y_1, \ldots, Y_d$, as random variables, with randomness due to random inputs. Each of these random variables

Y_j has a distribution $p_{\hat{\beta}_j(\theta)}$, where the parameters $\hat{\beta}_j(\theta)$ of this distribution are induced by the weights $\boldsymbol{\theta}$ of the neural network:

$$p_{\hat{\beta}_j(\theta)}(y) = \int_{s \in \mathcal{S}} p(s) p(\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(s) = y) ds.$$

This provides a distribution over the values for the feature $\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(s)$, across inputs s. A Distributional Regularizer is a KL divergence $KL(p_\beta || p_{\hat{\beta}_j(\boldsymbol{\theta})})$ that encourages this distribution to match a desired target distribution p_β with parameter β.

Such a regularizer can be used to encourage sparsity, by selecting a target distribution that has high mass or density at zero. Consider a Bernoulli distribution for activations, with $Y_j \in \{0, 1\}$. Using a Bernoulli target distribution with $\beta = 0.1$, giving $p_\beta(Y = 1) = 0.1$, encodes a desired activation of 10%. As another example, for continuous nonnegative Y_j, the target distribution can be set to an exponential distribution $p_\beta(y) = \beta^{-1} \exp(-y/\beta)$, which has highest density at zero with expected value β. Setting $\beta = 0.1$ encourages the average activation to be 0.1 and increases density on $y = 0$.

The efficacy of this regularizer, however, is tied to the parameterization of the network, which should match the target distribution. For a ReLU activation, for example, which has a range $[0, \infty)$, a Bernoulli target distribution is not appropriate. Rather, for the range $[0, \infty)$, an exponential distribution is more suitable. For a Sigmoid activation, giving values between $[0, 1]$, a Bernoulli is reasonably appropriate. Additionally, the parametrization should be able to set activations to zero. The ReLU activation naturally enables zero values [15], by pushing activations to negative values. The addition of a Distributional Regularizer simply encourages this natural tendency, and is more likely to provide sparse representations. Activations under Sigmoid and tanh, on the other hand, are more difficult to encourage to zero, because they require highly negative input values or input values exactly equal to 0.5, respectively, to set the hidden node to zero. For these reasons, we advocate for ReLU for the sparse layer, with an exponential target distribution.

Finally, we modify this regularizer to provide a Set Distributional Regularizer, which does not require an exact level of sparsity to be achieved. It

can be difficult to choose a precise level of sparsity, making the Distributional Regularizer prone to misspecification. Rather, the actual goal is typically to obtain *at least* some level of sparsity, where some nodes can be even more sparse. For this modification, we specify that the distribution should match any of a set of target distributions Q_β, giving a *Set KL*: $\min_{p \in Q_\beta} KL(p||p_{\hat{\beta}_j(\boldsymbol{\theta})})$. Generally, this Set KL can be hard to evaluate. However, as we show below, it corresponds to a simple clipped KL-divergence for certain choices of Q_β, importantly including for exponential distributions where $Q_\beta = \{p_{\tilde{\beta}} \ | \tilde{\beta} \leq \beta\}$.

Theorem 1 (Set KL as a Clipped-KL)**.** *Let p_η be a one-dimensional exponential family distribution with the natural parameter η, $B = [\eta_1, \eta_2]$ be a convex set in the natural parameter space and $Q_B = \{p_\eta : \eta \in B\}$. Then the Set KL divergence*

$$SKL(Q_B||p_\eta) := \min_{p \in Q_B} KL(p||p_\eta) \tag{5.1}$$

is (a) non-negative (b) convex in η and (c) corresponds to a simple clipped form

$$SKL(Q_B||p_\eta) = \begin{cases} KL(p_{\eta_2}||p_\eta) & \text{if } \eta > \eta_2 \\ KL(p_{\eta_1}||p_\eta) & \text{if } \eta < \eta_1 \\ 0 & \text{else} \end{cases} \tag{5.2}$$

Proof. For exponential families, the KL divergence correspond to a Bregman divergence [4]:

$$KL(p_{\eta_1}||p_\eta) = D_F(\eta||\eta_1)$$

for a convex potential function F that depends on the exponential family. Hence, we have

$$SKL(Q_B||p_\eta) = \arg\min_{\tilde{\eta} \in B} D_F(\eta||\tilde{\eta})$$

If $\eta \in B$, this minimum over Bregman divergences is clearly zero. If $\eta < \eta_1$ and $\eta > \eta_2$, we have to consider the minimization. The Bregman divergence is not necessarily convex in the second argument. Instead, we can rely on

convexity of the set B. Taking the derivative of $D_F(\eta||\tilde{\eta})$ wrt $\tilde{\eta}$, we get

$$\begin{aligned}\frac{d}{d\tilde{\eta}}D_F(\eta||\tilde{\eta}) &= \frac{d}{d\tilde{\eta}}\left[F(\eta) - F(\tilde{\eta}) - (\eta-\tilde{\eta})\frac{d}{d\tilde{\eta}}F(\tilde{\eta})\right] \\ &= -\frac{d}{d\tilde{\eta}}F(\tilde{\eta}) + \frac{d}{d\tilde{\eta}}F(\tilde{\eta}) - (\eta-\tilde{\eta})\frac{d^2}{d\tilde{\eta}^2}F(\tilde{\eta}) \\ &= -\frac{d^2}{d\tilde{\eta}^2}F(\tilde{\eta})(\eta-\tilde{\eta})\end{aligned}$$

Now because F is convex, $-\frac{d^2}{d\tilde{\eta}^2}F(\tilde{\eta})$ is always negative. The derivative, then, is negative when $\tilde{\eta} < \eta$, indicating $\tilde{\eta}$ should be increased to decrease $D_F(\eta||\tilde{\eta})$. Similarly, when $\tilde{\eta} > \eta$, the derivative is positive, indicating $\tilde{\eta}$ should be decreased to decrease $D_F(\eta||\tilde{\eta})$. This derivative, then, points $\tilde{\eta}$ to the boundaries when $\eta \notin B$, respectively to the boundary points closest to η. □

Corollary 1 (SKL for Exponential Distributions)**.** *For p_β an exponential distribution, with natural parameter $\eta = -\beta^{-1}$, and $B = (0, \beta]$, then*

$$SKL(Q_B||p_{\hat{\beta}}) = \begin{cases} \log\hat{\beta} + \frac{\beta}{\hat{\beta}} - \log\beta - 1 & \text{if } \hat{\beta} > \beta \\ 0 & \text{else} \end{cases} \tag{5.3}$$

We use the SKL in Corollary 1, to encode a sparsity level of at least β—rather than exactly β—for the last layer in a two-layer neural network with ReLU activations. This regularizer was used to encourage sparse activations for SR-NN in the preceding chapter.

The overall loss function is

$$J_{sparse}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda_{KL}\sum_{i}^{d} SKL(Q_B||p_{\hat{\beta}_j})$$

where $J(\boldsymbol{\theta})$ is the vanilla objective function, d is the dimension of representation, and λ_{KL} controls the sparsity regularization. We include pseudocode for optimizing the regularized objective with the SKL for Exponential Distributions in Algorithm 1. Note that this algorithm is not used online, since we learn the representation for a batch of data. The pseudocode is given for the offline batch setting.

Algorithm 1 Optimizing the regularized objective

1: Initialize neural networks weights based on He initialization [18]: for each layer l and each element ij of the weight matrix $\mathbf{W}_{ij}^{(l)} \sim \mathcal{N}(0, \frac{2}{n_l})$ and $\mathbf{b}^{(l)} = \mathbf{0}$ where n_l the number of input nodes for layer l.
2: **while** not converge to a minimum **do**
3: Draw m samples $\{(x_1, y_1), ..., (x_m, y_m)\}$ from the dataset uniformly at random
4: Forward pass: compute the activation h_{ij} in the last layer for each sample x_i and each feature j.
5: Backward Pass: for $j = 1, ..., k$, compute $\hat{\beta}_j = \sum_{i=1}^{m} h_{ij}/m$ and the gradient:

$$\frac{\partial SKL(Q_B||p_{\hat{\beta}_j})}{\partial \hat{\beta}_j} = (\frac{1}{\hat{\beta}_j} - \frac{\beta}{\hat{\beta}_j^2})\mathbb{1}[\hat{\beta}_j > \beta]$$

6: Update each weight $\boldsymbol{\theta} \in \{\forall l, \mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$ with the gradient:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \lambda_{KL} \sum_{j=1}^{k} \frac{\partial SKL(Q_B||p_{\hat{\beta}_j})}{\partial \hat{\beta}_j} \frac{\partial \hat{\beta}_j}{\partial \boldsymbol{\theta}}$$

7: **end while**

5.2 Evaluation of Distributional Regularizers

In this section, we investigate the efficacy of Distributional Regularizers for obtaining sparsity. There are a variety of possible choices with Distributional Regularizers, including activation function and corresponding target distribution and using a KL versus a Set KL. In this section, we investigate some of these combinations, particularly focusing on the difference in sparsity and performance when using (a) KL and SKL; (b) Bernoulli distribution (with sigmoid), Exponential distribution (with ReLU) and Gaussian distribution (with ReLU); and (c) previous strategies to obtain sparse representations versus the proposed variant of the Distributional Regularizer.

Here we provides details of the Set KL for Bernoulli distributions and Gaussian distributions. Let p_ρ be a Bernoulli distribution, with natural parameter ρ, $B = [0, \rho]$ and $Q_B = \{p_\eta : \eta \in B\}$, then SKL for Bernoulli distributions is

$$SKL(Q_B||p_{\hat{\beta}_j}) = \begin{cases} \rho \log \frac{\rho}{\hat{\beta}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\beta}_j} & \text{if } \hat{\beta}_j > \rho \\ 0 & \text{else.} \end{cases}$$

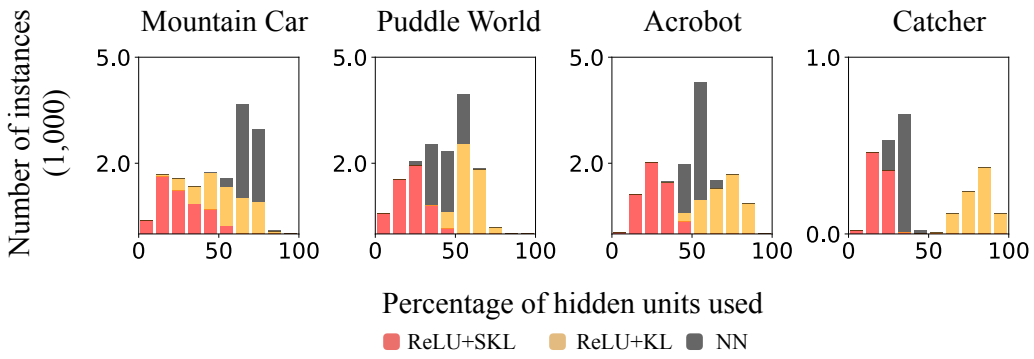


Figure 5.1: Instance sparsity as evaluated on a batch of test data comparing Exp+KL and Exp+SKL to NN. While Exp+KL can make representations denser than just NN, Exp+SKL always results in sparser representations. The numbers are averaged over 30 runs.

Similarly, let $p_{\mu,\sigma}$ be a Gaussian distribution, with natural parameter (μ, σ), $B = (-\infty, \mu]$ and $Q_B = \{p_{\eta,\sigma} : \eta \in B\}$, then SKL for Gaussian distributions is

$$SKL(Q_B||p_{\hat{\beta}_j,\sigma}) = \begin{cases} \frac{\sigma^2+(\hat{\beta}_j-\mu)^2}{2\sigma^2} - \frac{1}{2} & \text{if } \hat{\beta}_j > \mu \\ 0 & \text{else} \end{cases}$$

In the first set of experiments, we compare the instance sparsity of KL to Set KL, with ReLU activations and Exponential Distributions (Exp+KL and Exp+SKL). Figure 5.1 shows the instance sparsity for these, and for the NN without regularization. Interestingly, Exp+KL actually reduces sparsity in several domains, because the optimization encouraging an exact level of sparsity is quite finicky. Exp+SKL, on the other hand, significantly improves instance sparsity over the NN. This instance sparsity again translates into control performance, where Exp+KL does noticeably worse than Exp+SKL across the four domains in Figure 5.2. Despite the poor instance sparsity, Exp+KL does actually seem to provide some useful regularity, that does allow some learning across all four domains. This contrasts the previous regularization strategies, ℓ_2, ℓ_1 and Dropout, which all failed to learn on at least one domain, particularly Catcher.

In the next set of experiments, we compare with different distributional

regularizers: a Gaussian distribution with ReLu activation and a Bernoulli distribution with Sigmoid activation. We included both KL and Set KL, giving the combinations Exp+KL, Exp+SKL, Gau+KL, Gau+SKL, Ber+KL, and Ber+SKL. We expect Sigmoid with Bernoulli to perform significantly worse—in terms of sparsity levels and performance—because the Sigmoid activation makes it difficult to truly get sparse representations. This hypothesis is validated in the learning curves in Figure 5.2. Ber+KL and Ber+SKL perform poorly across domains, even in Puddle World, where they achieved their best performance. Unlike ReLU with Exponential, here the Set KL seems to provide little benefit. Gau+SKL and Gau+KL perform well across all domains, but Gau+KL has a slower learning in Puddle World. The instance sparsity in Figure 5.3 shows that Gau+SKL and Gau+KL have similar sparsity levels. In fact, since we use ReLU activation, the Set KL equals to the regular KL when $\mu = 0$ and they are close when μ is small. This explains why Gau+SKL and Gau+KL have similar performance in our experiments.

We compare to previously proposed strategies for learning sparse representations with neural networks. These include using ℓ_1 and ℓ_2 regularization on the activation (denoted by ℓ_1R-NN and ℓ_2R-NN respectively); k-sparse NNs, where all but the top k activation are zeroed [31] (k-sparse-NN); and Winner-Take-All NNs that keep the top k% of the activation per node across instances in the minibatch during training, to promote sparse activation of nodes over time [32] (WTA-NN). These approaches, however, can be problematic because they tend to truncate non-negligible values or produce insufficiently sparse representations. Both k-sparse-NNs and WTA-NNs were introduced for autoencoders, though the idea can be applied more generally to NNs. We tested these methods with autoencoders, but performance was significantly worse.

We include learning curves and instance sparsity for these methods, for a ReLU activation, in Figures 5.4 and 5.5. Neither WTA-NN nor k-sparse-NN are effective. We found the k-sparse-NN was prone to dead units, and often truncates non-negligible value. Surprisingly, ℓ_2R-NN performs comparably to SR-NN in Mountain Car and Acrobot, whereas ℓ_1R-NN is effective only during early learning in Mountain Car. From the instance sparsity plots in Puddle

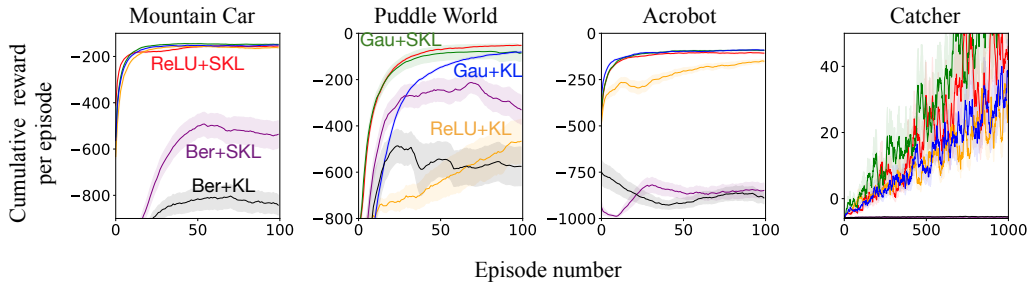


Figure 5.2: Learning curves for Sarsa(0) with different Distributional Regularizers. All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

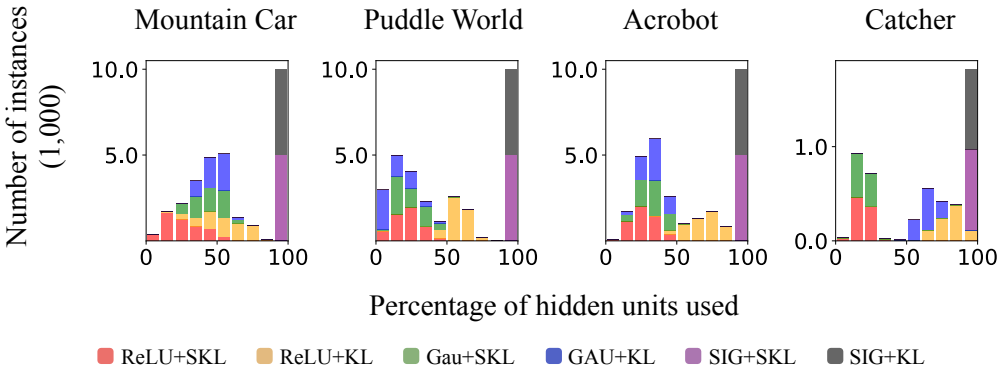


Figure 5.3: Instance sparsity as evaluated on a batch of test data comparing with different Distributional Regularizers. The numbers are averaged over 30 runs.

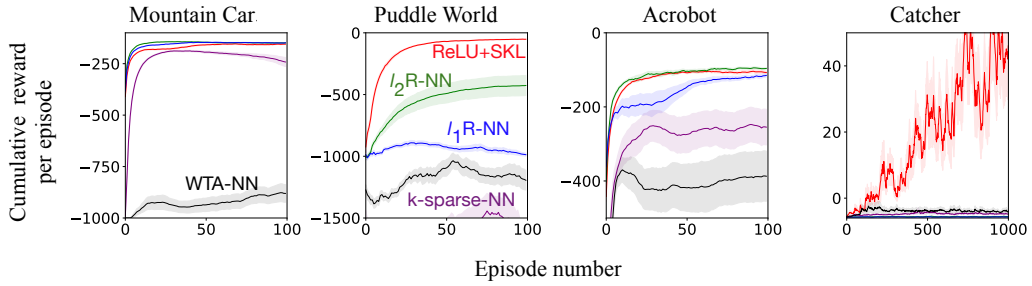


Figure 5.4: Learning curves for Sarsa(0) comparing SR-NN to previous proposed sparse representations learning strategies. All learning curves are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

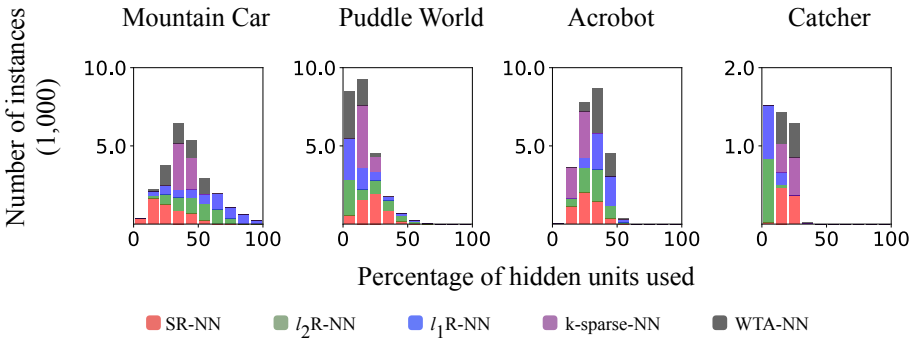


Figure 5.5: Instance sparsity comparing SR-NN to previous proposed sparse representations learning strategies. The numbers are averaged over 30 runs.

World and Catcher, we see that ℓ_1R-NN and ℓ_2R-NN produce highly sparse (<5% instance sparsity), potentially explaining its poor performance. This was with considerable parameter optimization for the regularization parameter.

In conclusion, previous approaches such as ℓ_1R-NN, ℓ_2R-NN and k-sparse-NN are prone to collapsing to a few elements and resulting in dead units. This problem often occurs in sparse coding as well as autoencoders with instance sparsity regularization [31, 32]. Distributional regularizers, however, naturally avoid this problem by forcing each unit to be sparsely active over its lifetime. We empirically evaluate theses methods in reinforcement learning tasks, and the results suggest that distributional regularizers are able to learn effective sparse representations, especially with ReLu activation and Exp+SKL.

5.3 Experimental Details

We used the same neural network architecture as the experiments in the previous chapter. The range of grid search for the representation hyperparameters

are as follows:

$$\lambda_{KL} \in \{0.1, 0.01, 0.001\} \quad \text{[Distributional Regularizers]}$$
$$\beta \in \{0.05, 0.1\} \quad \text{[Exp+KL and Exp+SKL]}$$
$$\rho \in \{0.05, 0.1\} \quad \text{[Ber+KL and Ber+SKL]}$$
$$\mu \in \{0.0, 0.05, 0.1\} \quad \text{[Gau+KL and Gau+SKL]}$$
$$\lambda_{NN} \in \{0.1, 0.01, 0.001, 0.0001\} \quad [\ell_1\text{R-NN and } \ell_2\text{R-NN}]$$
$$k \in \{16, 32, 64, 128\} \quad \text{[k-sparse-NN]}$$
$$k \in \{6.25\%, 12.5\%, 25\%, 50\%\} \quad \text{[WTA-NN]}$$

For the KL and Set KL for Gaussian distributions, we fix $\sigma = 1$. For k-sparse networks, only the top-k hidden units in the representation layer are activated. We also use scheduling of sparsity level described in the original paper [31].

Chapter 6

Discussion and Future Work

We summarize the contributions presented in this dissertation, followed by a discussion on future research directions.

6.1 Summary of Contributions

In this work, we investigate using and learning sparse representations with neural networks for control in reinforcement learning. We show that sparse representations can significantly improve control performance when used in an online learning setting, and provide some evidence that this is because the sparsity of the representation reduces catastrophic interference which would otherwise overwrite value estimates. We formalize Distributional Regularizers, with a practically important extension to a Set KL, for learning a Sparse Representation Neural Network (SR-NN). We provide an empirical investigation into the sparsity properties and control performance under different Distributional Regularizers, as well as compared to other algorithms to obtain sparse representations with neural networks. We conclude that SR-NN performs consistently well across domains.

This work highlights an important phenomenon that arises in online reinforcement learning, beyond the typical issues with catastrophic interference. Interference is typically considered for sequential multi-task learning, where previous functions are forgotten by training on a new task. Interference could occur even in a single-task setting, if the agent has sequences of correlated observations. In reinforcement learning, however, this problem is magnified by

the fact that the data distribution is changing at each time step during training. This work provides some first empirical steps, in a carefully controlled set of experiments, to identify that this could be an issue, and that sparse representations could be a promising direction to alleviate the problem.

6.2 Future Directions

In this dissertation, we show sparse representation with linear function approximation improves control performance. A fist next-step extension is to apply the same strategy to non-linear function approximation with multi-layer neural networks. In backpropagation, the first partial derivatives w.r.t. the weight $W_{ij}^{(l)}$ in layer l is

$$\frac{\partial J(W)}{\partial W_{ij}^{(l)}} = \frac{\partial J(W)}{\partial z_i^{(l+1)}} a_j^{(l)}.$$

where $z_i^{(l+1)}$ is the ith input (before activation function) in layer $l+1$ and $a_j^{(l)}$ is the jth activation in layer l. If each layer has sparse activation, the gradient vector w.r.t. the weights would be sparse and, therefore, have less interference. Applying Distributional Regularizers in each layer is a straightforward method to achieve sparsity in each layer, however, we hypothesize that the regularizer would strongly penalize the first few layers and generate an ineffective function approximation. Another line of research is to develop algorithms to learn sparse representations and the value function concurrently and incrementally, where the value function is linear w.r.t. sparse representations. Hence, a single sparse activation layer is sufficient to reduce interference in the value function. Two-timescale networks [9] could be a useful framework for this line of research.

This dissertation opens up interesting research directions on interference and sparse representation in reinforcement learning. We hope for this work to spur further empirical investigation into what we believe is a widespread issue, and further algorithmic development into learning sparse representations for reinforcement learning.

References

[1] J. Achiam, E. Knight, and P. Abbeel, "Towards characterizing divergence in deep q-learning," *arXiv:1903.08894*, 2019. 13, 24

[2] S. Ahmad and J. Hawkins, "Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory," 2015. arXiv: 1503.07469. 2

[3] D. Arpit, Y. Zhou, H. Ngo, and V. Govindaraju, "Why regularized auto-encoders learn sparse representation?" In *International Conference on Machine Learning*, 2015. 3

[4] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, 2005. 31

[5] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, *et al.*, "Vector-based navigation using grid-like representations in artificial agents," *Nature*, 2018. 25

[6] M. G. Bellemare, W. Dabney, R. Dadashi, A. A. Taıga, P. S. Castro, N. L. Roux, D. Schuurmans, T. Lattimore, and C. Lyle, "A geometric perspective on optimal representations for reinforcement learning," *arXiv:1901.11530*, 2019. 10

[7] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, 2009. 1

[8] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. 1, 22

[9] W. Chung, S. Nath, A. Joseph, and M. White, "Two-timescale networks for nonlinear value function approximation," 2018. 10, 40

[10] T. M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition.," *IEEE Trans. Electronic Computers*, 1965. 2

[11] P. Földiák, "Forming sparse representations by local anti-Hebbian learning," *Biological Cybernetics*, 1990. 1

[12] R. M. French, "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks," in *Annual Cognitive Science Society Conference*, 1991. 1, 3, 24

[13] ——, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, 1999. 11

[14] S. Ghiassian, H. Yu, B. Rafiee, and R. S. Sutton, "Two geometric input transformation methods for fast online reinforcement learning with neural nets," *arXiv:1805.07476*, 2018. 11

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011. 3, 30

[16] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv:1312.6211*, 2013. 11

[17] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in Neural Information Processing Systems*, 2009. 1

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision*, 2015. 18, 33

[19] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012. 28

[20] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in neural information processing systems*, 2018. 13

[21] P. Kanerva, *Sparse Distributed Memory*. MIT Press, 1988. 1

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014. 18

[23] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, 2017. 11

[24] L. Le, R. Kumaraswamy, and M. White, "Learning sparse representations in reinforcement learning with sparse coding," *arXiv:1707.08316*, 2017. 3, 10

[25] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Advances in neural information processing systems*, 2008. 3

[26] A. Lemme, R. F. Reinhart, and J. J. Steil, "Online learning and generalization of parts-based image representations by non-negative sparse autoencoders," *Neural Networks*, 2012. 3

[27] L.-J. Lin, "Reinforcement learning for robots using neural networks," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993. 9

[28] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017. 11

[29] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *Journal of Machine Learning Research*, 2010. 3

[30] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning," in *Advances in Neural Information Processing Systems*, 2009. 3

[31] A. Makhzani and B. Frey, "k-sparse autoencoders," *arXiv:1312.5663*, 2013. 3, 35, 37, 38

[32] ——, "Winner-take-all autoencoders," in *Advances in Neural Information Processing Systems*, 2015. 3, 35, 37

[33] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," *Psychology of Learning and Motivation*, 1989. 1, 11

[34] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, 1943. 3

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015. 9, 10, 17

[36] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, 2011. 29

[37] B. A. Olshausen, "Sparse Codes and Spikes," in *Probabilistic Models of the Brain*, 2002. 3

[38] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vision Research*, 1997. 2, 3

[39] R. Quian Quiroga and G. Kreiman, "Measuring sparseness in the brain: Comment on bowers (2009).," 2010. 2

[40] M. Ranzato, Y.-L. Boureau, and Y. LeCun, "Sparse Feature Learning for Deep Belief Networks.," in *Advances in Neural Information Processing Systems*, 2007. 3

[41] M. Ranzato, C. S. Poultney, S. Chopra, and Y. LeCun, "Efficient Learning of Sparse Representations with an Energy-Based Model.," in *Advances in Neural Information Processing Systems*, 2006. 3

[42] B. Ratitch and D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning," in *Machine Learning: ECML PKDD*, 2004. 1

[43] M. Riedmiller, "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*, 2005. 2, 10, 17

[44] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," *arXiv:1810.11910*, 2018. 11

[45] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *International Conference on Machine Learning*, 2011. 1

[46] T. Sajed, W. Chung, and M. White, "High-confidence error estimates for learned value functions," *arXiv preprint arXiv:1808.09127*, 2018. 16

[47] T. Schaul, D. Borsa, J. Modayil, and R. Pascanu, "Ray interference: A source of plateaus in deep reinforcement learning," *arXiv:1904.11455*, 2019. 24

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, 2014. 22

[49] R. S. Sutton, "Two problems with back propagation and other steepest descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986. 11

[50] ——, "Learning to predict by the methods of temporal differences," *Machine learning*, 1988. 7, 11, 15

[51] ——, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, 1996. 1, 9, 11

[52] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. 2018. 1, 2, 8, 12, 20

[53] Y. W. Teh, M. Welling, S. Osindero, and G. E. Hinton, "Energy-Based Models for Sparse Overcomplete Representations.," *Journal of Machine Learning Research*, 2003. 3

[54] J. Triesch, "A Gradient Rule for the Plasticity of a Neuron's Intrinsic Excitability," in *ICANN*, 2005. 3

[55] C. Wang, X. Chen, A. J. Smola, and E. P. Xing, "Variance reduction for stochastic gradient optimization," in *Advances in Neural Information Processing Systems*, 2013. 13

[56] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, 1992. 7