The Utility of Sparse Representations for Control in Reinforcement Learning

Vincent Liu[1], Raksha Kumaraswamy[1], Lei Le[2], Martha White[1]
[1]Department of Computing Science, University of Alberta, Edmonton, Canada
{vliu1, kumarasw, whitem}@ualberta.ca
[2]Department of Computer Science, Indiana University Bloomington, Indiana, USA
leile@indiana.edu

Abstract

We investigate sparse representations for control in reinforcement learning. While these representations are widely used in computer vision, their prevalence in reinforcement learning is limited to sparse coding where extracting representations for new data can be computationally intensive. Here, we begin by demonstrating that learning a control policy incrementally with a representation from a standard neural network fails in classic control domains, whereas learning with a representation obtained from a neural network that has sparsity properties enforced is effective. We provide evidence that the reason for this is that the sparse representation provides locality, and so avoids catastrophic interference, and particularly keeps consistent, stable values for bootstrapping. We then 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.

Introduction

Learning performance in artificial intelligence systems is highly dependent on the data representation—the features. 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 (McCloskey and Cohen 1989; French 1991), 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 (Bengio 2009; Bengio, Courville, and Vincent 2013), 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). 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 hand-crafted features are sparse representations, including tile coding (Sutton 1996; Sutton and Barto 1998), radial basis functions and sparse distributed memory (Kanerva 1988; Ratitch and Precup 2004). Other useful properties of sparse representations—which can be seen as projecting data into a higher-dimensional space—include invariance (Goodfellow et al. 2009; Rifai et al. 2011); decorrelated features per instance (Földiák 1990); improved computational efficiency for updating weights in the predictor, as only weights corresponding to active features need to be updates; and enabling linear separability in the high-dimensional space (Cover 1965), which facilitates the learning of a simple linear predictor. Further, such sparse, distributed representations have been observed in the brain (Olshausen and Field 1997; Quian Quiroga and Kreiman 2010; Ahmad and Hawkins 2015).

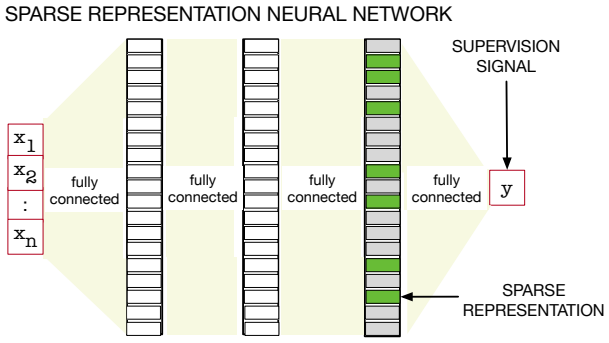


Figure 1: A neural network with dense connections producing a sparse representation: Sparse Representation Neural Network (SR-NN). The green squares indicate active (non-zero) units, making a sparse last hidden layer where only a small percentage of units are active. This contrasts a network with sparse connections—which is often also called sparse. Sparse connections remove connections between nodes, but are likely to still produce a dense representation.

Traditionally, sparse representations have been common for control in reinforcement learning, such as tile coding and radial basis functions (Sutton and Barto 1998). 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 target networks, inspired by batch methods such as fitted Q-iteration (Riedmiller 2005), have been necessary for many of the successes of control with neural networks. We provide some evidence in this paper that this modification is necessary with dense, but not sparse, networks because the reinforcement learning agent bootstraps off its own estimates. If the value in other states are overwritten, the agent will bootstrap off inaccurate estimate. Local representations, however, are much less likely to suffer from interference and these issues with bootstrapping. 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 (Mairal et al. 2009; 2010; Le, Kumaraswamy, and White 2017) or for general sparse distributions (Olshausen and Field 1997; Olshausen 2002; Teh et al. 2003; Ranzato et al. 2006; Ranzato, Boureau, and LeCun 2007; Lee et al. 2008), 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 (Mairal et al. 2009; Lemme, Reinhart, and Steil 2012; Le, Kumaraswamy, and White 2017).

There are fewer methods using feedforward neural network architectures. Certain activation functions—such as linear threshold units (LTU) (McCulloch and Pitts 1943) and rectified linear units (ReLU) (Glorot, Bordes, and Bengio 2011)—naturally provide some level of sparsity, but of course provide no such guarantees. Early work on catastrophic interference investigated some simple heuristics for encouraging sparsity, such as node sharpening (French 1991). Though catastrophic interference was reduced, the resulting networks were still quite dense.[1] k-sparse autoencoders (Makhzani and Frey 2013) 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 autoencoders (Makhzani and Frey 2015) 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 (Triesch 2005; Ranzato et al. 2006; Lemme, Reinhart, and Steil 2012; Arpit et al. 2015), but used a sigmoid activation which is unlikely to result in sparse representations. They define sparsity based on norms of the vector, rather than activation level.

In this work, we first highlight that learned sparse representations can significantly improve control performance, under an incremental learning setting, compared to dense neural networks. We visualize the activation of the hidden nodes for the sparse representation as well as the action-values for particular states. These provide evidence that locality helps avoid catastrophic interference and improves accuracy of action-values for bootstrapping. We then investigate a simple strategy for encouraging sparsity in neural networks: Distributional Regularizers. This approach flexibly enables any desired architecture, simply with the addition of a KL divergence on the activation level for a node. We show that direct use of such a regularizer can cause dead filters or collapse—activation concentrating on a few nodes—potentially explaining why this simple strategy has not yet found wide-spread use. We show that a simple clipping is sufficient to obtain effective sparse representations, and conclude with a comparison to several other strategies for obtaining a sparse representation on the same benchmark domains.

Background

In reinforcement learning (RL), an agent interacts with its environment, receiving observations and selecting actions to maximize a reward signal. The environment is formalized by a Markov decision process (MDP), with states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities $\Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, rewards $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and discount function $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ (White 2017).

One algorithm for on-policy control is Sarsa, where the agent updates its action-values for its current policy and acts near-greedily according to these action-values. The action-values for a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ are the expected return for that policy, starting from state s and action a:

$$Q^\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \quad (1)$$
$$\text{where, } G_t = R_{t+1} + \gamma_{t+1} G_{t+1}$$

These action-values can be estimated with function approximation, such as with neural network. Because the expected return is a real-value target, such a neural network typically uses a linear activation on the last layer:

$$Q^\pi(s, a) \approx \hat{Q}_{\mathbf{w},\boldsymbol{\theta}}(s, a) := \boldsymbol{\phi}_{\boldsymbol{\theta}}(s, a)^\top \mathbf{w} \quad (2)$$

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* 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 this representation $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s, a)$.

[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.

The Utility of Sparsity for Control

We begin by highlighting the utility of sparsity for control before discussing how to learn sparse representations. We show that two sparse representations—tile coding and sparse representation learned by a neural network (referred to as *SR-NN* from hereon)—both significantly improve stability 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 (Sutton and Barto 1998). We hypothesize that the main reason is due to catastrophic interference, which is much less problematic for the local representations typically provided by a sparse representations. We show both that SR-NN does appear to have more stable action-values for bootstrapping, and that the learned sparse representation is local, providing some evidence for this hypothesis.

We evaluate control performance on four benchmark domains: Mountain Car, Puddle World, Acrobot and Catcher. All domains are episodic, with discount set to 1 until termination. 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 of these domains. We provide details about the domains in the Appendix.

The experimental set-up is as follows. To extract a representation with a neural network, to be used for control, we pre-train the neural network on a batch of data with a mean-squared temporal difference error (MSTDE) objective and the applicable regularization strategies. The training data consists of trajectories generated by a fixed policy that explores much of the space in the various domains. For the SR-NN, we use our distributional regularization strategy, described in a later section. This learned representation is then fixed, and used by a (fully incremental) Sarsa(0) agent for learning a control policy, where only the weights $\mathbf{w}$ on the last layer are updated. The meta-parameters for the batch-trained neural network producing the representation and the Sarsa 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 ranges and objectives are provided in the Appendix.

We choose this two-stage training regime to remove confounding factors in difficulties of training neural networks incrementally. Our goal here is to identify if a sparse representation can improve control performance, and if so, why. The networks are trained with an objective for learning values, on a large batch of data generated by a policy that covers the space; the learned representations are capable of representing the optimal policy. We investigate their utility for fully incremental learning. Outside of this carefully controlled experiment, we advocate for learning the representation incrementally, for the task faced by the agent.

The learning curves for the four domains, with Tile-Coding (TC), SR-NN and NN , are shown in Figure 2. Both SR-NN and NN used two-layers, of size $[32, 256]$, with ReLU activations. The NNs performs surprisingly poorly, in some case

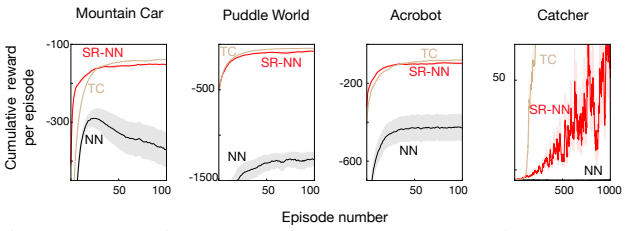


Figure 2: Learning curves for Sarsa(0) comparing SR-NN, Tile Coding and vanilla NN in the four domains.

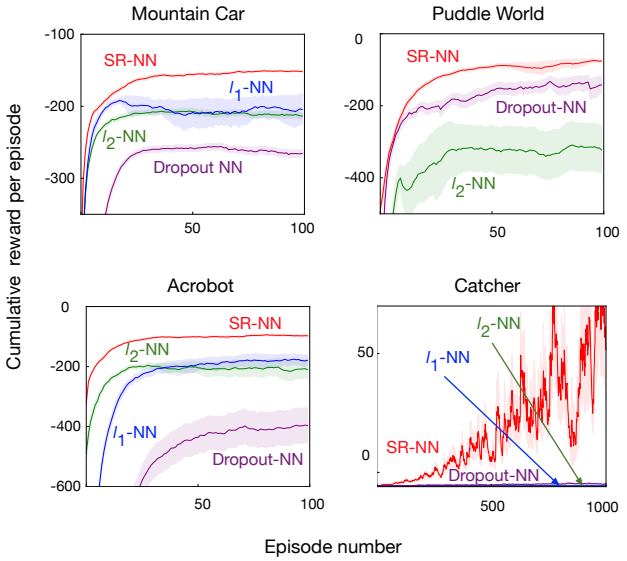


Figure 3: Learning curves for Sarsa(0) comparing SR-NN to the regularized representations. All representations except ℓ_1-NN in Puddle World could reach goals more than 70 times out of 100. ℓ_1 does poorly in Puddle World, and is not visible.

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, as expected. Specifically in Catcher, TC learns a close-to-optimal policy as the representation is powerful. 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. Yet, the sparsity of 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.

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 (Srivastava et al. 2014) (Dropout-NN). The ℓ_1 regularizer actually encourages weights to go to zero, reducing the number of connections, but does not necessarily provide a sparse representation. In Figure 3, 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

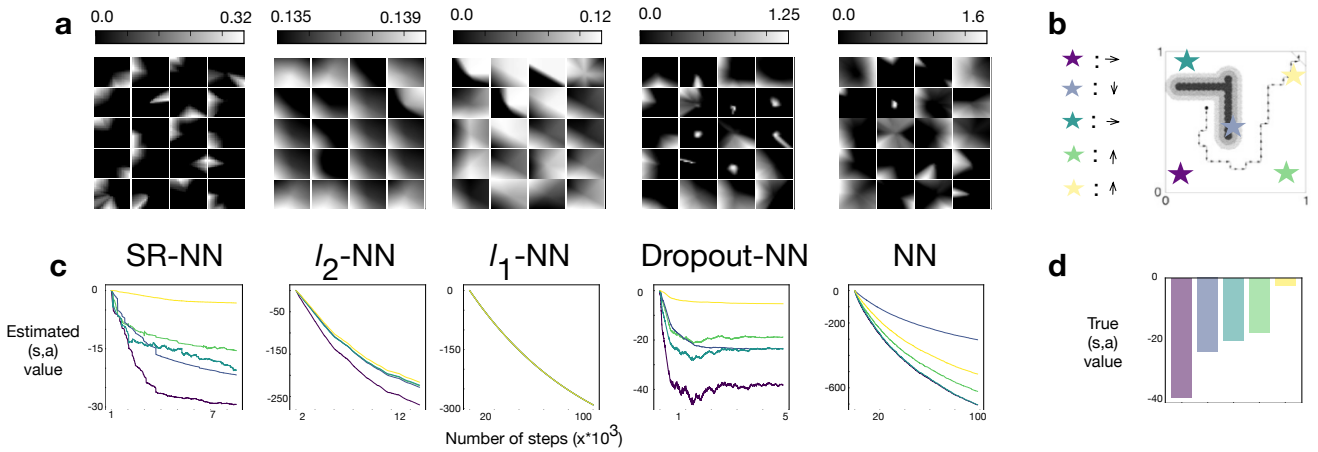


Figure 4: A study in Puddle World to investigate the effect of locality during on-policy control. (**a**) The activation maps for 20 randomly chosen neurons for different representations - each cell in the heatmap corresponds to the complete Puddle World state-space. The activation maps, and magnitude of activation of SR-NN are visibly sparser, and lower, when compared to Dropout and NN. ℓ_1 and ℓ_2 are quite dense in terms of activation area, whereas the magnitudes are really low - due to regularization of the network weights. (**b**) A visualization of the domain, denoting the selected state-action pairs used in the analysis. (**c**) The estimated state-action values for the selected configurations during on-policy control with Sarsa(0) ($\epsilon = 0.1$), while utilizing the specific representation of interest. (Sarsa(0) budget - 100 episodes with 1k episode cut-off for all representations). (**d**) The true state-action values for the selected configurations with an $\epsilon = 0.1$-optimal policy, estimated from 100k Monte Carlo rollouts.

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 except Puddle World. Interestingly, in this one domain, Dropout-NN appears to have learned a sparse representation, based on the heatmap shown in Figure 4. It has been observed that Dropout can at times learn sparse representations (Banino et al. 2018), but not consistently, as corroborated by our experiments.

We next investigate the hypothesis that locality is preventing catastrophic interference. We first investigate the locality of the representations, as well as examining the bootstrap values over time. We show results for Puddle World here, as it is an interpretable two-dimensional domain; similar experiments for other domains are in the Appendix. Figure 4(a) 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 this domain, as mentioned earlier.

To show the stability (or lack of stability) of bootstrap targets used during control, we select five states and evaluate their action-values for the optimal action over the course of learning. These states are distributed across the observation space, depicted in Figure 4(b). The bootstrap estimates, that correspond to the algorithm settings for the learning curves, are plotted in Figure 4(c). We can see that the relative ordering of the value estimates is maintained with SR-NN and

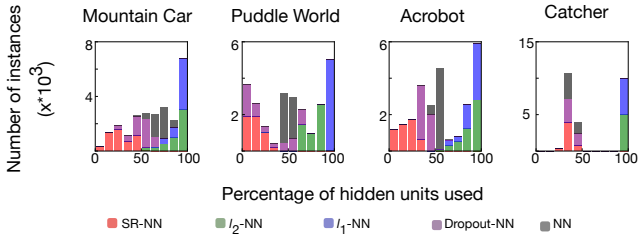


Figure 5: Instance sparsity comparing SR-NN to the regularized variants and vanilla NN. The percentage evaluation is designed to disregard units that are never active across all samples in the batch (dead units).

Dropout-NN, which were the two NNs effective for on-policy control, and that their values converge to near the true values (given in Figure 4(d)). The other representations, on the other hand, have very poor estimates. Moreover, these estimates seem to decrease together, suggesting interference is causing overgeneralization to reduce values in other states.

Finally, we report additional measures of locality, to determine if the successful methods are indeed sparse. The heatmaps provide some evidence of locality, but are more qualitative than quantitative. We report two qualitative measures: *instance sparsity* and *activation overlap*. 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 5, SR-NN has consistently low instance sparsity across all four domains, with slightly higher level in Catcher, potentially explaining the noisy behaviour in that domain. Once again, Dropout-NN is noticeably more instance sparse on Puddle World, but less so on other domains. The NN representation, which has no regularization, has some instance sparsity, likely due to simply using ReLU activa-

SR-NN	ℓ_2-NN	ℓ_1-NN	Dropout-NN	NN
8.8	111.5	142.5	31.2	54.0

Table 1: Activation overlap in Puddle World. The numbers are the average overlap over all pairs of selected states. For example, SR-NN has an average of 8.8 shared activation over all pais of 5 selected states defined in Figure 4 (a).

tion. Interestingly, ℓ_1-NN and ℓ_2-NN actually produced less instance sparsity.

Activation overlap, introduced by French(1991), reflects the amount of shared activation between any two inputs. We consider a variant of activation overlap that 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 $\mathbf{x}_2$:

$$\text{overlap}(\boldsymbol{\phi}(\mathbf{x}_1),\boldsymbol{\phi}(\mathbf{x}_2)) = \sum_j \mathbb{1}[(\boldsymbol{\phi}_j(\mathbf{x}_1) > 0) \wedge (\boldsymbol{\phi}_j(\mathbf{x}_2) > 0)].$$

We measure the activation overlap of the five chosen states, distributed across Puddle World. If the overlap between two representations is zero, the interference would be zero. Updating the value function with respect to one state, therefore, would not affect the other state's value. Table 1 shows the average overlap, and once again, a similar trend emerges where, SR-NN has significantly less overlap (about 8), with Dropout-NN showing the next least overlap (with about 30).

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 (c) this locality reduces interference and improves accuracy of bootstrap values in Sarsa(0). 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. In the next section, we discuss how we actually obtain such sparse representations (SR-NN).

Distributional Regularizers for Sparsity

In this section, we describe how to use Distributional Regularizers to learn sparse representations with neural networks.[2] We introduce a Set Distributional Regularizer, which when paired with ReLU activations enables sparse representations to be learned, as we demonstrate in the next section. 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.

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 (Glorot, Bordes, and Bengio 2011), 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*

$$SKL(Q_B||p_\eta) := \min_{p\in Q_B} KL(p||p_\eta) \tag{3}$$

[2]The idea was originally introduced for neural networks with Sigmoid activations in an unpublished set of notes (Ng 2011), 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.

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

$$SKL(Q_B||p_\eta) = \begin{cases} KL(p_{\eta_2}||p_\eta) & \text{if } \eta > \eta_2 \\ KL(p_{\eta_1}||p_\eta) & \text{if } \eta < \eta_1 \\ 0 & \text{else} \end{cases} \quad (4)$$

Proof. For exponential families, the KL divergence correspond to a Bregman divergence (Banerjee et al. 2005):

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

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*

$$SKL(Q_B||p_{\hat{\beta}}) = \begin{cases} \log\hat{\beta} + \frac{\beta}{\hat{\beta}} - \log\beta - 1 & \text{if } \hat{\beta} > \beta \\ 0 & \text{else} \end{cases} \quad (5)$$

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 section. We include pseudocode for optimizing the regularized objective with the SKL, in Algorithm 1 in the Appendix.

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 versus SKL; (b) Sigmoid (with a Bernoulli target distribution) versus ReLU (with an Exponential target distribution); and (c) previous strategies to obtain sparse representations versus the proposed variant of the Distributional Regularizer.

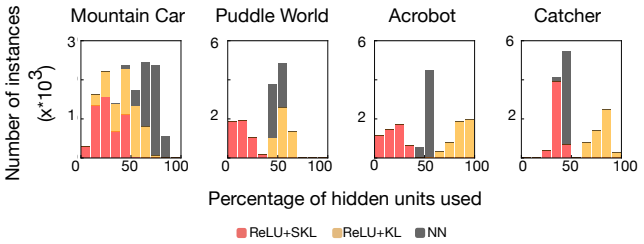


Figure 6: Instance sparsity as evaluated on a batch of test data comparing ReLU+KL and ReLU+SKL to NN. While ReLU+KL can make representations denser than just NN, ReLU+SKL always results in sparser representations.

In the first set of experiments, we compare the instance sparsity of KL to Set KL, with ReLU activations and Exponential Distributions (ReLU+KL and ReLU+SKL). Figure 6 shows the instance sparsity for these, and for the NN without regularization. Interestingly, ReLU+KL actually reduces sparsity in several domains, because the optimization encouraging an exact level of sparsity is quite finicky. ReLU+SKL, on the other hand, significantly improves instance sparsity over the NN. This instance sparsity again translates into control performance, where ReLU+KL does noticeably worse than ReLU+SKL across the four domains in Figure 7. Despite the poor instance sparsity, ReLU+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 Sigmoid (with a Bernoulli target distribution) versus ReLU (with an Exponential target distribution). We included both KL and Set KL, giving the combinations ReLU+KL, ReLU+SKL, SIG+KL, and SIG+SKL. We expect Sigmoid with Bernoulli to perform significantly worse—in terms of sparsity levels, locality and performance—because the Sigmoid activation makes it difficult to truly get sparse representations. This hypothesis is validated in the learning curves in Figure 7 and the heatmaps for Puddle World in Figure 8. SIG+KL and SIG+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. The heatmaps in Figure 8 show that both versions, SIG+KL and SIG+SKL, cover large porions of the space, and do not have local activations for hidden nodes. In fact, SIG+KL and SIG+SKL use all the hidden nodes for all the samples across domains, resulting in no instance sparsity.

Next, 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 activations are zeroed (Makhzani and Frey 2013) (k-sparse-NN); and Winner-Take-All NNs that

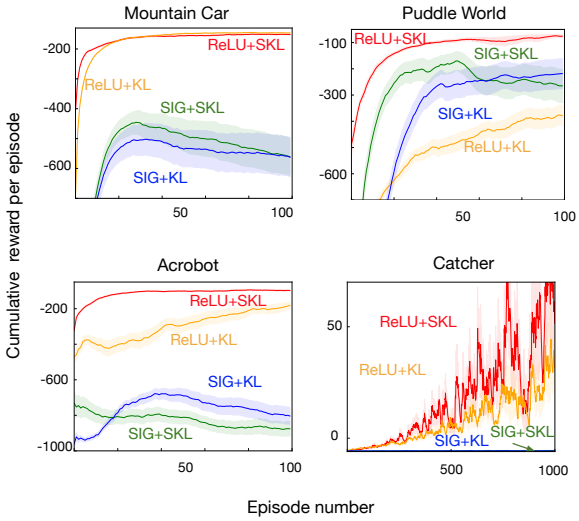


Figure 7: Learning curves for Sarsa(0) with different Distributional Regularizers.

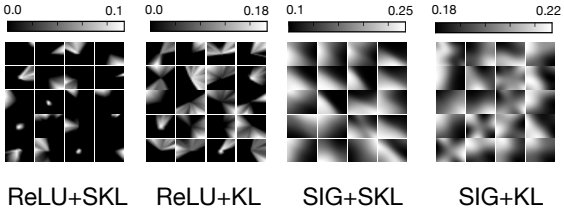


Figure 8: Heatmaps of activations with different Distributional Regularizers in Puddle World.

keep the top k% of the activations per node across instances, to promote sparse activations of nodes over time (Makhzani and Frey 2015) (WTA-NN).[3]

We include learning curves and instance sparsity for these methods, for a ReLU activation, in Figures 9 and 10. Results for the Sigmoid activation are included in the Appendix. 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 all domains but Catcher, whereas ℓ_1R-NN is effective only during early learning in Mountain Car. From the instance sparsity plots in Catcher, we see that ℓ_1R-NN and ℓ_2R-NN produce highly sparse (2%-3% instance sparsity), potentially explaining its poor performance. While similar instance sparsity was effective in Puddle World, this is unlikely to be true in general. This was with considerable parameter optimization for the regularization parameter.

Conclusion

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 incremental learning setting, and provide some evidence that this

[3]Both k-sparse-NNs and WTA-NNs were introduced for autoencoders, though the idea can be applied more generally to NNs. We additionally tested these methods with autoencoders, but performance was significantly worse.

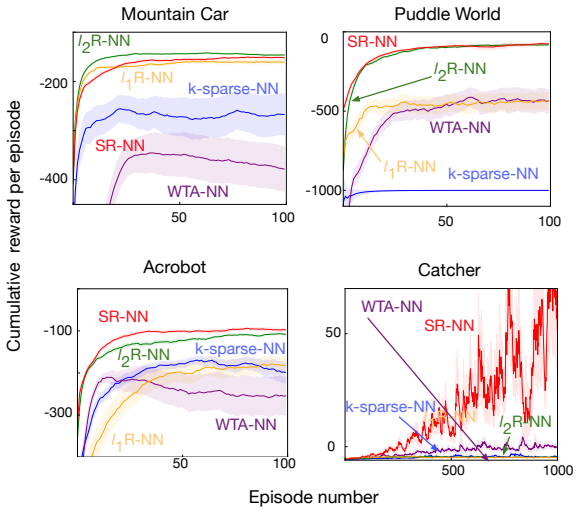


Figure 9: Learning curves for Sarsa(0) comparing SR-NN to previous proposed sparse representations learning strategies.

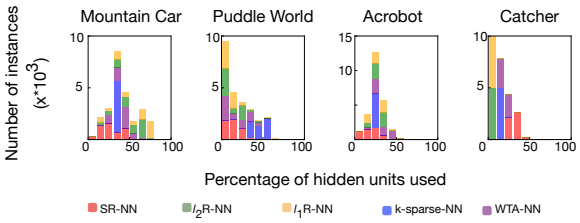


Figure 10: Instance sparsity comparing SR-NN to previous proposed sparse representations learning strategies.

is because the locality of the representation reduces catastrophic interference which would otherwise overwrite bootstrap values. 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, with the next best method—which only fails in one domain—being a simple methods that uses an ℓ_2 regularizer on activations.

This work highlights an important phenomenon that arises in control, 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 an agent remains in a particular area of the space for a long time. In reinforcement learning, however, this problem is magnified by the fact that the agent uses its own estimates as targets. If estimates change incorrectly due to interference, there could be a cascading effect. 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. We hope for this work to spur further empirical investigation into how widespread this issue is, and further algorithmic development into learning sparse representations for reinforcement learning.

References

Ahmad, S., and Hawkins, J. 2015. Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory.

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

Banerjee, A.; Merugu, S.; Dhillon, I. S.; and Ghosh, J. 2005. Clustering with bregman divergences. *Journal of Machine Learning Research* 6(Oct):1705–1749.

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*.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*.

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

Földiák, P. 1990. Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*.

French, R. M. 1991. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Annual Cognitive Science Society Conference*.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*.

Goodfellow, I.; Lee, H.; Le, Q. V.; Saxe, A.; and Ng, A. Y. 2009. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*.

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

Kanerva, P. 1988. *Sparse Distributed Memory*. MIT Press.

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

Le, L.; Kumaraswamy, R.; and White, M. 2017. Learning sparse representations in reinforcement learning with sparse coding. *arXiv:1707.08316*.

Lee, H.; Ekanadham, C.; information, A. N. A. i. n.; and 2008. 2008. Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems*.

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

Mairal, J.; Bach, F.; Ponce, J.; Sapiro, G.; and Zisserman, A. 2009. Supervised dictionary learning. In *Advances in Neural Information Processing Systems*.

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

Makhzani, A., and Frey, B. 2013. k-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

Makhzani, A., and Frey, B. 2015. Winner-take-all autoencoders. In *Adv. in Neural Information Processing Systems*.

McCloskey, M., and Cohen, N. J. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation*.

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

Ng, A. 2011. Sparse autoencoder. *CS294A Lecture notes*.

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

Olshausen, B. A. 2002. Sparse Codes and Spikes. In *Probabilistic Models of the Brain*.

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

Ranzato, M.; Poultney, C. S.; Chopra, S.; and LeCun, Y. 2006. Efficient Learning of Sparse Representations with an Energy-Based Model. In *Adv. in Neural Info. Process. Sys.*

Ranzato, M.; Boureau, Y.-L.; and LeCun, Y. 2007. Sparse Feature Learning for Deep Belief Networks. In *Advances in Neural Information Processing Systems*.

Ratitch, B., and Precup, D. 2004. Sparse distributed memories for on-line value-based reinforcement learning. In *Machine Learning: ECML PKDD*.

Riedmiller, M. 2005. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*.

Rifai, S.; Vincent, P.; Muller, X.; Glorot, X.; and Bengio, Y. 2011. Contractive auto-encoders: Explicit invariance during feature extraction. In *Inter. Conf. on Machine Learning*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press Cambridge.

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

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*.

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

Triesch, J. 2005. A Gradient Rule for the Plasticity of a Neuron's Intrinsic Excitability. In *ICANN*.

White, M. 2017. Unifying Task Specification in Reinforcement Learning. In *Inter. Conf. on Machine Learning*.

Additional algorithmic details

Algorithm 1 Optimizing the regularized objective

1: Initialize neural networks weights based on He initialization (He et al. 2015): 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 i.i.d. samples $\{y_1, ..., y_m\}$ from the true data distribution

4: For $j = 1, ..., k$, compute $\hat{\beta}_j = \sum_{i=1}^{m} y_{ij}/m$ and the gradient:

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

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

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

In general, we advocate for learning the representation incrementally, for the task faced by the agent. However, for our experiments, we learned the representations first to remove confounding factors. We detail that learning regime here.

The problem of 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 from Equation (2) as:

$$\hat{Q}_{\mathbf{w},\boldsymbol{\theta}}(s, a) := \boldsymbol{\phi}_{\boldsymbol{\theta}}(s)^\top \mathbf{w}_a \tag{6}$$

Here, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$ is the linear 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. Under a given policy, like the action-values $Q^\pi(s, a)$, corresponding state-values, $V^\pi(s)$, are defined as:

$$V^\pi(s) := \mathbb{E}[G_t | S_t = s]$$
$$\text{where, } G_t = R_{t+1} + \gamma_{t+1} G_{t+1}$$

An easy objective to train connectionist networks with simple backpropagation is the Mean Squared Temporal Difference Error (MSTDE) (Sutton 1988). For a given policy, the MSTDE is defined as:

$$\sum_{s \in \mathcal{S}} \mathbf{d}(s) \mathbb{E}[\delta_t^2 | S_t = s] \tag{7}$$

$$\text{where, } \delta_t := R_{t+1} + \gamma_{t+1} \boldsymbol{\phi}_{\boldsymbol{\theta}}(S_{t+1})^\top \mathbf{w}_v - \boldsymbol{\phi}_{\boldsymbol{\theta}}(S_t)^\top \mathbf{w}_v$$

Here, $\mathbf{d}$ denotes the stationary distribution over the states induced by the given policy, and $\boldsymbol{\theta}$ and $\mathbf{w}_v$ are parameters that can be estimated 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, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$. These features can then be used for estimating action-values in on-policy control for learning the (close-to) optimal behaviour policy in the environment.

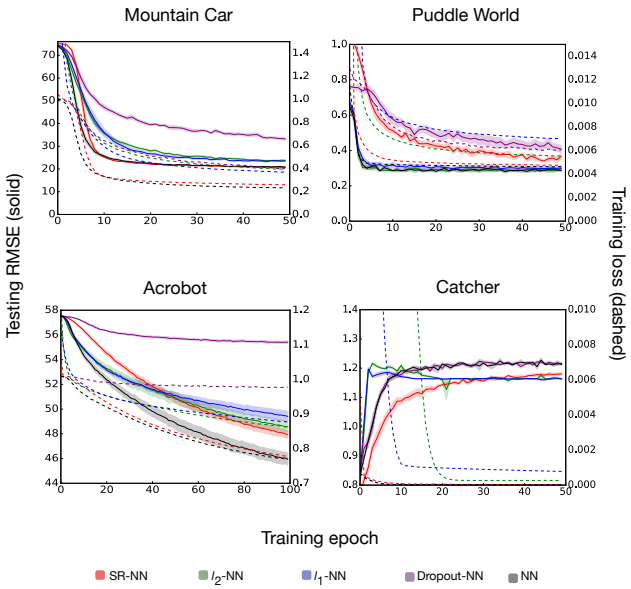


Figure 11: Learning curve during the training of neural networks with different regularization methods.

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. 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.

Tile Coding

We compare to Tile Coding (TC) representation, a well-known sparse representation, as the baseline. TC uses overlapping grids on the observation space, to convert a continuous space to a discrete dimensional space. The representations generated by it are sparse and distributed based on a static hashing technique. We experiment with several configurations for the fixed representation, particularly with grid-sizes(N) in $\{4, 8, 16\}$ and number of tilings (D) in $\{8, 16, 32\}$. We use a hash size of 8192, which is significantly larger than the largest feature size of 256, as used in the other learned representation models we compare to. The results shown in Figure 3 are for the best configuration of the static tile-coder after a sweep.

Training neural networks

Architecture and optimizer: 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. We optimized the neural network weights using Adam optimization (Kingma and Ba 2014) with a batch size of 64. The neural network weights are initialized based on He initialization (He et al. 2015). That is, the neural networks weights are initialized with zero-mean Gaussian distribution

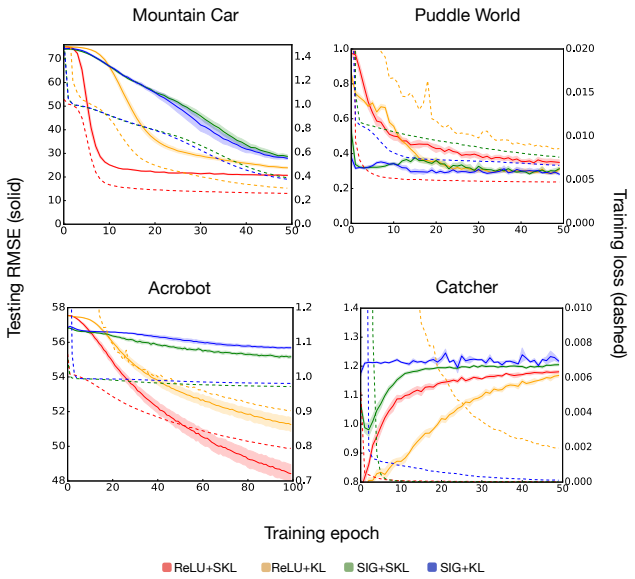


Figure 12: Learning curve during the training of neural networks with different distributional regularizers.

with variance equals to $2/n_l$, where n_l is the number of input nodes for layer l.

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

$$\begin{aligned}
\lambda_{KL} &\in \{0.1, 0.01, 0.001\} \\
\beta &\in \{0.05, 0.1, 0.2\} \\
\lambda_{NN} \text{ for } \ell_1 \text{ and } \ell_2 &\in \{0.1, 0.01, 0.001, 0.0001\} \\
\text{dropout probability } p &\in \{0.1, 0.2, 0.3, 0.4, 0.5\} \\
k \text{ for k-sparse} &\in \{16, 32, 64, 128\} \\
k \text{ for WTA} &\in \{6.25\%, 12.5\%, 25\%, 50\%\}
\end{aligned}$$

Algorithmic choices: 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 (Makhzani and Frey 2013). If used in conjunction with a distributional regularizer, the top-k nodes are chosen before application of the distributional regularizer. 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[4] – this preserves dropouts role as regularizer w.r.t. the target, and promotes diversity in learning.

Grid-search evaluation metric: The learned representations are then used for on-policy control in Sarsa(0) with fixed $\epsilon = 0.1$. The value function for Sarsa is initialized with zero-mean Gaussian distribution with small variance. For sparse representations, we use semi-gradient Sarsa with step decay learnining rate. For dense representatinos, we use adaptive learnining rate method RMSprop (Hinton, Srivastava, and Swersky 2012). The initial learning rate for Sarsa(0)

[4]We have 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.

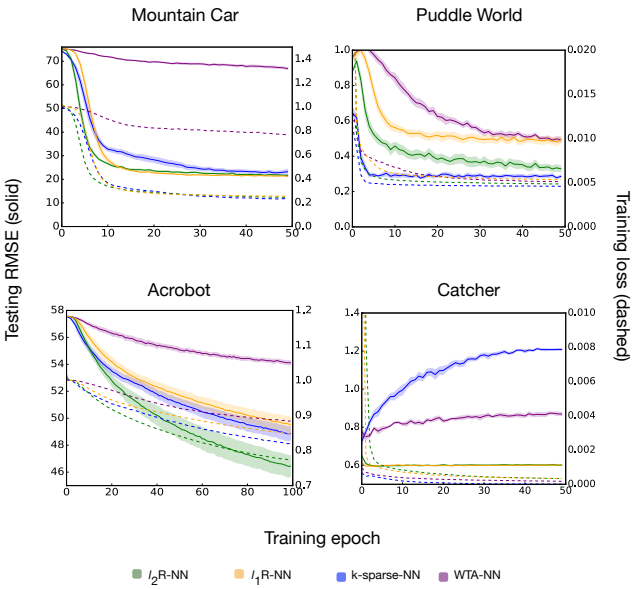


Figure 13: Learning curve during the training of neural networks with different sparsity inducing approaches.

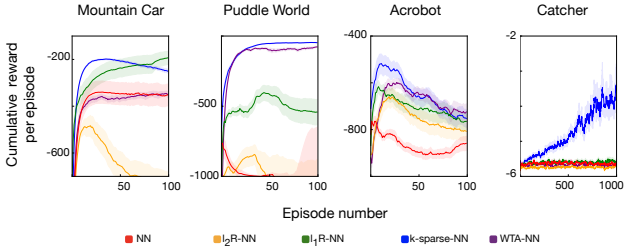


Figure 14: Learning curves for Sarsa(0) comparing various sparsity inducing networks with Sigmoid activation.

is swept in the set:

$$\alpha_0 \in \{0.1, 0.04, 0.01, 0.004, 0.001, 0.0004, 0.0001\}$$

All the sweeps for selecting the representation learning hyperparameters across domains use 50 epochs and 10 runs.

Learning curves: The chosen hyperparameters are used to train a good representation (saturated testing loss – 100 epochs for Acrobot, and 50 epochs for other domains), following which it is used for on-policy control with Sarsa(0). While the control performance is focused on in the main paper, the learning curve during the representation training phase is shown in Figures 11, 12 and 13. The metric on the y-axis is the Root Mean Squared Error (RMSE), which is

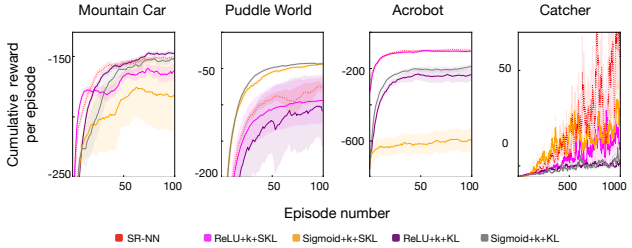


Figure 15: Learning curves for Sarsa(0) comparing various variants of distributional regularizers with k-sparse.

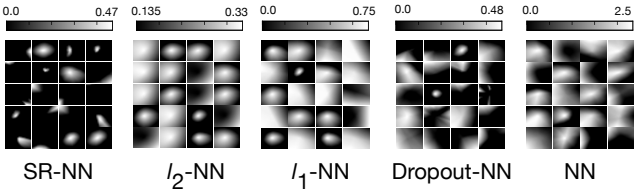


Figure 16: Heatmaps of activations comparing SR-NN to different regularization strategies and NN in Mountain Car.

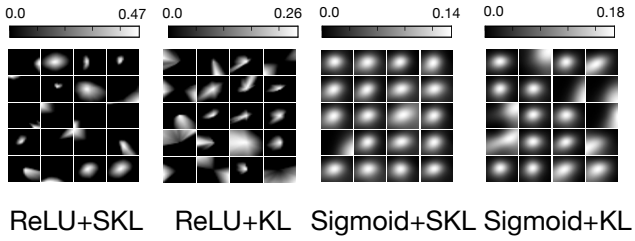


Figure 17: Heatmaps of activations with different Distributional Regularizers in Mountain Car.

evaluated as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{\mathbf{x} \in \mathbf{X}_{test}} (\hat{V}(\mathbf{x}) - V^*(\mathbf{x}))^2}{\mathbf{X}_{test}}}$$

where $\mathbf{X}_{test}$ is the set of test states for which the representations have been extracted, $\hat{V}(\mathbf{x})$ is the estimated value of state $\mathbf{x}$ and $V^*(\mathbf{x})$ is the true value of state $\mathbf{x}$ computed using Monte Carlo rollouts. The number of test states are 5000 for benchmark domains and 1000 for Catcher. Most algorithms converge to a good solution within 50 epochs in Mountain Car, Puddle World and Catcher, and 100 epochs in Acrobot as shown in the curves. The curves are for representation purposes and only averaged over 5 runs. All learning curves for Sarsa(0) are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

More results

Control curves

We perform the evaluation of sparsity inducing networks with Sigmoid activation. Figure 14 shows the performance of Sarsa(0) with representations learned by different networks. k-sparse and WTA performs well in Puddle World, however, none of these representations are effective across all domains.

The learning curves for various k-sparse networks with distributional regularizers are in Figure 15. It suggests that k-sparse (ReLU+k+SKL) provides no improvement over just using distributional regularizer for ReLU activation (SR-NN).

Activation heatmaps

The activation heatmaps for randomly selected neurons (excluding dead neurons) in Mountain Car with different regularization stratergies are shown in Figure 16, and with differnt Distributional Regularization designs are shown in Figure 17. Heatmaps for sparsity inducing networks with ReLU activations and Sigmoid activation, for Mountain Car and Puddle World are shown in Figure 18.

	Mountain Car	Puddle World
SR-NN	16.8	8.8
ℓ_2-NN	112.3	111.5
ℓ_1-NN	109.5	142.5
Dropout-NN	72.5	31.2
NN	106.5	54.0
ReLU+KL	36.8	71.4
SIG+SKL	256.0	256.0
SIG+KL	256.0	256.0
k-sparse-NN	36.6	61.8
WTA-NN	24.8	6.5
ℓ_2R-NN	30.0	3.8
ℓ_1R-NN	10.5	0.4

Table 2: Activation overlap in Mountain Car and Puddle World. For Mountain Car, the numbers are the average overlap over all pairs of selected states defined in Figure 19.

Bootstrap values

The bootstrap values comparing SR-NN to different regularization strategies, and NN are shown in Figure 19. Since it is not easy to visualize 4-dimensional space, we only include the bootstrap value result of Mountain Car here.

Activation overlap

We show the overlap of representations learned by different networks in Table 2 for Mountain Car and Puddle World. ℓ_2R-NN and ℓ_1R-NN have low overlap values. However, the regularizers tend to push many neurons to be activated for a really small region to reduce penalty as shown in Figure 18. SR-NN, on the other hand, learns a more distributed representation.

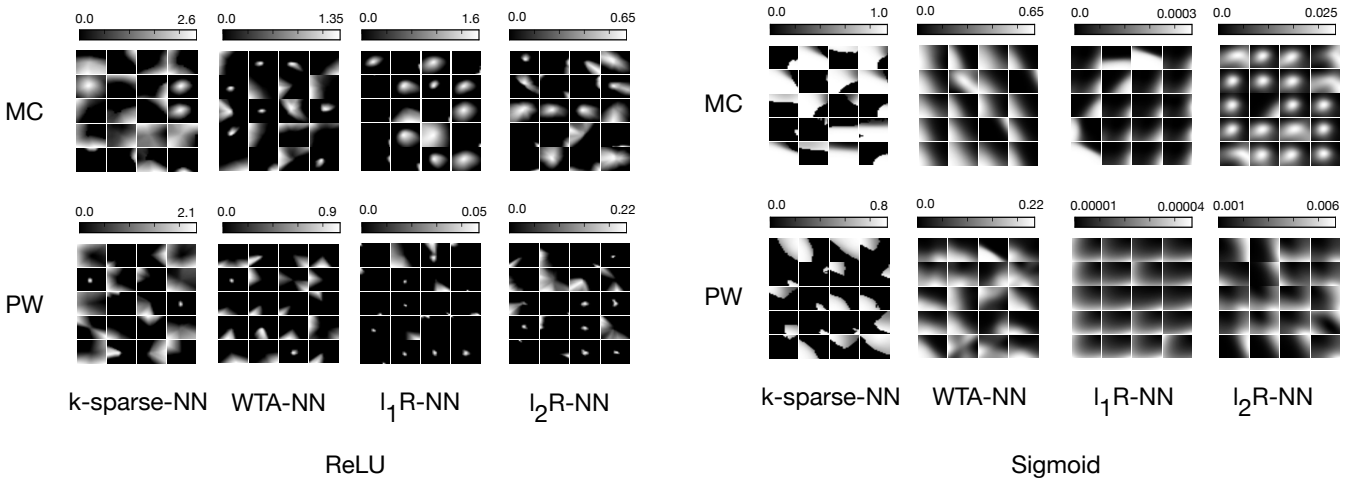
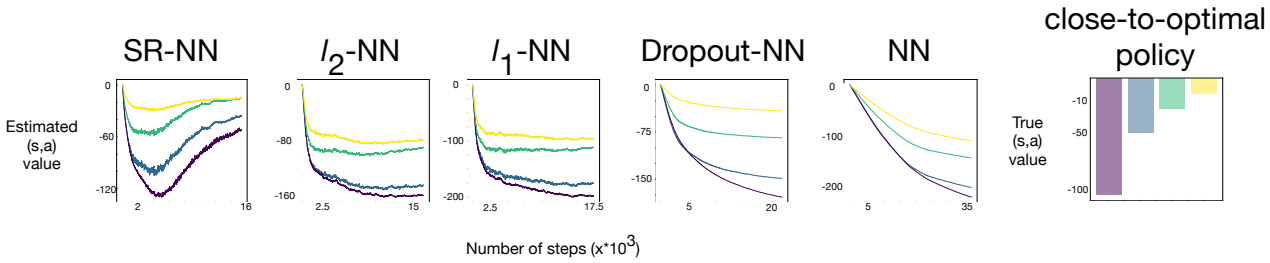


Figure 18: Heatmaps of activations for nodes from other networks which aim to generate sparse representations (ReLU and Sigmoid activation).



<valley,-ve>: reverse <firstHillTop,+ve>: accelerate <valley,+ve>: accelerate <secondHillTop,+ve>: accelerate

Figure 19: This plot compares the bootstrap estimates of SR-NN to various regularization stratergies during on-policy control for the chosen 4 state-action pairs denoted in the following format in the legend: <car-position,car-velocity>:action. Again, we see that the relative ordering of bootstrap values is maintained with SR-NN, and it tends towards the true values of the ($\epsilon = 0.1$)-optimal policy. The optimal policy estimates (currently) use 10k Monte Carlo rollouts with a powerful close-to-optimal tile-coder policy.