Sparse Representations in Deep RL: Encoding Information in Deep Architectures

Hunter Hobbs
James.Hobbs@Colorado.edu

Anuj Pasricha
Anuj.Pasricha@Colorado.edu

Matthew Resnick
Matthew.Resnick@Colorado.edu

***Abstract*—Is less truly more? We aim to investigate the problem of Catastrophic Interference through the lens of one area of potential solutions: sparse representations. We provide context for work in the area of sparse-coding and deep sparse representations, then justify, implement, and analyze the results of sparse pathway encoding, parameter-distribution-based regularization, and spatially packed representations. We show the successes and shortcomings of creating sparse representations in different ways, both in terms of task performance and the level of sparsity of the representation.**

I. Introduction

In this project, we aim to improve performance in tasks completed by a simulated robot and reduce model overhead by creating sparse representations in a deep neural controller. There is evidence from two perspectives on deep learning that supports sparse representations as an effective avenue. From the purely supervised learning perspective, there is a growing body of literature which supports the idea that the carefully constructed, sparely connected structure of a neural network is more important than the optimization of parameters for performance [1] [2] [3]. From a Reinforcement Learning (RL) perspective, the recent literature suggests that sparse representations may be highly effective at addressing the problem of Catastrophic Interference [4] [5]. And further, from both of these perspectives, sparse models often come with the added benefit of being less expensive to train, store, and compute a forward pass. Thus, in order to evaluate the effectiveness of these claims, we will compare the success of deep baseline models with that of models which employ notions of sparse pathway encoding (through specialized activation functions and regularization), Distributional Regularizers, and spatially packed representations.

We implement channel-out activation in Tensorflow and compare it's effects with that of maxout, ReLU, and SELU with and without dropout for the actor network in a DDPG model. We then see how a sparse representation can be created by combining what we know about the distributions produced by these activations with KL-divergence, in implementing Distributional Regularizers. Finally, the culmination of our investigation leads us to our creation of a modified PackNet, an algorithm for multi-task computer vision learning. This modified PackNet is the combination of information we gleaned from the prior two areas, whereby we show that we can spatially pack representations into a single network to allow for more stable single- and multi-task representations to be learned via pathway encoding and increased network capacity usage.

II. Background and Related Work

Retroactive interference is a term used in human psychology to describe the phenomena of difficulty in remembering learned material from the past because of newly learned material. In the field of Reinforcement Learning such a phenomena appears fairly commonly wherever the use of Artificial Neural Networks (ANNs) also appears. In deep RL, this problem is called Catastrophic Interference or Catastrophic Forgetting; a neural network containing a representation of a task will experience interference from learning a more recent task that completely deteriorates the old representation [6]. This could be a result of training on different regions of an environment for a single task or training on different tasks entirely. Kemker, McClure, Abitino, Hayes, and Kanan (2017) [7] present five approaches to addressing this problem, all with their own sub-approaches and varying degrees of success at mitigation: regularization, ensembling, rehearsal, dual-memory, and sparse-coding.

Early attempts to combat catastrophic forgetting were largely centered around the third and fourth of these. Neural Fitted Q Iteration [8] is a popular dual-memory attempt at remedying the issue, by providing the network with explicit information about past experience at each update of new data. Experience replay [9] was an early rehearsal approach, and some form of it is used very often in modern practice. The first listed approach, regularization, is a rather amorphous idea in ANNs, and it is focused broadly on generalization rather than catastrophic forgetting specifically. Ensemble methods like bagging [10] and boosting [11] are strong approaches to forgetting, but they come with a host of optimization issues and have seen more success in supervised learning than in RL. This leaves the last, curious approach, which has a long history and yet has not received significant attention until recently: sparse-coding.

The central idea behind sparse coding is to reduce the chance of competing internal representations, as the structure of the model becomes just as important as the learned values. Further, a sparse representation of a model allows for a significant compression effect, allowing for potentially reduced training time and/or overhead. Though there is an extensive history to sparse representations in ANNs, it has not appeared as extensively in the realm of RL. Many sparse-coding approaches in RL took the form of engineered features, as in

Tile Coding [12], Radial Basis Function Networks [13], and Sparse Distributed Memory [14]. There have been some deep RL approaches lately, especially from Le, Kumaraswamy, and White who created an objective function for policy evaluation via sparse coding [15] as well as formalizing Distribution Regularizers for use with Sparse Representation Neural Networks [4]. Hernandez-Garcia and Sutton added to these results by showing the possibility of being able to learn a sparse representation concurrently with an action-value function [5].

This leaves Reinforcement Learning open to a wide array of ideas floating around the field of supervised learning that need only be re-thought to be applied to RL's own problems. For this project we will be focusing on a few specific concepts. The first is sparse pathway encoding—developed through Maxout [16] and Channel-out [17], activation functions which, by nature, encode information for a given input sample on the network pathways itself. We will also explore Distributional Regularizers and their interactions with various induced parameter distributions. Finally, we will delve into more direct manipulations of network structure, namely by investigating and expanding upon ideas originally developed by Mallya and Lazebnik for computer vision applications such as packing multiple representations into a single network [18] and trainable weight masks [19]. Throughout our experiments, we'll compare results from a stabilizing implementation of a prioritized experience replay buffer as defined by T. Schaul et al. [20]

III. Methods

A. Sparse Pathway Encoding: Activation Functions

We begin by discussing maxout, an activation function conceived of by Goodfellow et. Al., wherein each hidden unit in a layer is pooled into size p, and then the activation of the pool is determined by the maximum of the p neurons within [16]. Maxout proved to be a powerful activation function because, as demonstrated in the original paper, it could approximate any piecewise linear function with p number of pieces, including ReLU, when combined with the dropout technique. The authors of maxout, however, did not go into detail about a proposed explanation for maxout's capabilities, especially as combined with dropout regularization. This was a task later taken up by the authors of channel-out, a successor activation function with greater versatility [17]. Channel-out works similarly to maxout, except the function that operates over the pooled neurons to determine the "winner" is generic, and furthermore the remaining activations are zeroed, not removed entirely. This makes channel-out adaptable to different output distributions of a layer and also means it does not require special architecture considerations as maxout does.

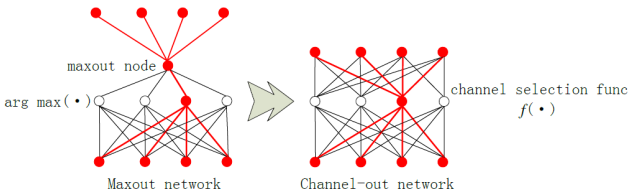

Fig. 1: *From [17]*

The importance of channel-out comes largely in its theoretical underpinnings. The authors of channel-out suggest that the output of each pool at each training step is information that is incorporated into the parameter optimization process, and hence information about the pathway through the network that a given sample may take is encoded into the network itself. They call this idea Sparse Pathway Encoding (SPE), and suggest that with dropout, a channel-out network can make full use of both pathway information and network capacity, as dropout itself works like a bagging model with sub-networks within the primary network.

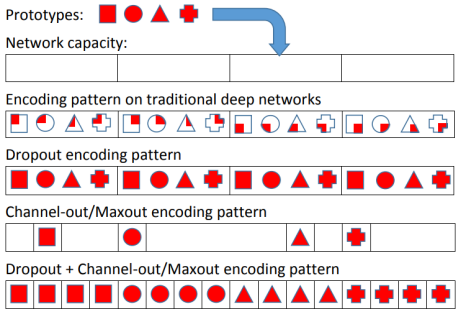

Fig. 2: *From [17] Simplified visual representation of channel-out.*

For this project, we intend to investigate how channel-out may encode state information in a DDPG model, and especially in the actor network. We wrote channel-out from scratch in Tensorflow, armed with both max and $absolutemax$ as pool-winner functions.

However, we note one major issue with both maxout and channel-out that we also intend to explore: dropout introduces a significant amount of variance into the model during training. In supervised learning, this variance has the effect of improving generalization error, but in reinforcement learning it has the effect of destabilizing training and possibly causing a representation to not be learned at all. Thus, we intend to show the dynamics of training with and without dropout on a primarily ReLU-activated model, a maxout model, a channel-out model, and a SELU model. SELU is a self-normalizing activation function asymptotically similar to ReLU with it's own version of dropout. It does not necessarily induce a sparse representation, but it during our experimentation it revealed important information about dropout as a technique for reinforcement learning. Later in this report we will dis-

cuss alternatives to dropout for increasing usage of network capacity, whether or not this usage is in conjunction with a sparsity-inducing activation function such as channel-out.

B. Distributional Regularizers

In his masters thesis, Vincent Liu wished to create a sparse representation in a deep model by encouraging the activations of a layer to fit a distribution which induces sparsity [21]. We will follow his logic to justify such a method of regularization for channel-out which circumvents the need for dropout.

First, consider each unit to be a random variable y with some empirical distribution $p_{\hat{\beta}_i}(\theta)$, where θ represents the layer's weights. Then,

$$p_{\hat{\beta}_i(\theta)}(y) = \int_{s \in S} p(s) p(\phi_{i,\theta}(s) = y) ds$$

where, for our purposes in DDPG experimentation, s represents a given state observation, $\phi_{i,\theta}$ represents a dimension of the state space, and y represents the loss target. If we can manipulate this distribution to be closer to a distribution with a higher density at zero, we will be, in effect, encouraging sparsity in the layer.

Liu suggests using Kullback-Leiber divergence, which by definition will give us a measure of difference between two distributions (in a similar sense that entropy demonstrates difference) [22], as a penalty term on the loss function to push the hidden units' distribution towards a desired distribution. Some of the most popular activation functions, including ReLU, at least asymptotically have a distribution which fits within the range $[0, \infty)$ [23] [24], and it is thus sensible to try to move the existing output distributions to an exponential distribution for these activations (restricting us to a positive domain and a high density about zero).

Liu proposes that a minimum sparsity can be specified by instead using a family of potentially desired distributions, as it may be too difficult to aim for one specific distribution, and further that by modifying KL divergence to include the choice of a minimum value out of the set of distributions is equivalent to clipping the original divergence (see [21] for more information and a proof). For exponential distributions, the divergence is described as

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

with modified loss function

$$J_{\text{sparse}}(\theta) = J(\theta) + \lambda_{KL} \sum_{i}^{\text{num units}} SKL(Q_B || p_{\hat{\beta}_i})$$

where β is the minimum desired level of sparsity for a given dimension and λ_{KL} is a tuneable scale factor. This provides us with a Distributional Regularizer for ReLU.

Channel-out is similar to ReLU at the node-level if the activating function is chosen to be max, and in fact can approximate ReLU, however negative values are technically allowed as output as we don't clip the max to zero. Hence, we should not attempt to move the output towards an exponential distribution, but rather towards something closer to the distribution of the values of the underlying weights, $\mathcal{N}(\mu, \sigma)$. Then,

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

and

$$\frac{\delta SKL(Q_B || p_{\hat{\beta}})}{\delta \hat{\beta}} = \frac{\hat{\beta}_i - \mu}{\sigma^2} \mathbb{1}[\hat{\beta} > \mu]$$

For this project, we wrote a few Distributional Regularizers from scratch in Tensorflow and evaluated their effect when applied to a few RL tasks, the results of which can be found in the subsequent "Results" section.

C. Increasing Network Capacity with Representation Packing

In implementing Distributional Regularizers, we demonstrated that many deep reinforcement learning models are capable of incredibly expressive representations even if a majority of activations, throughout all of training and testing, remain zero. This indicates that significant sub-graphs of the model stay unused for the purpose of internal representation. The authors of PackNet [18] noticed a similar phenomena in computer vision models for image classification, and thus concluded that it should be possible to fit multiple representations within the same network by a series of cycles of parameter pruning, freezing, and retraining. We extend their algorithm for a few different situations in reinforcement learning, allowing us to greatly increase our usage of network capacity without the need for dropout.

We consider two situations in reinforcement learning: one where we wish to have a model learn a single task in an environment stably and robust to the random initialization of the environment, and one where we wish to have a single model learn multiple tasks. In both cases, our extension of the PackNet algorithm works identically (where, for the first situation, differently seeded environments can be treated as multiple tasks).

For the first task seen during training, we store the parameter initialization values and train the model normally via feed-forward calculation and backpropagation of loss gradient information. Then, after training is complete, we determine parameter salience by the L_1-norm of the parameters and identify the top $\frac{1}{T-t}$ fraction of parameters per layer, where T is the total number of tasks and t is the current task number, indexed from 0. This ensures that the parameters are ultimately divided roughly evenly for each task. These neurons are then frozen from future training by masking-out their corresponding gradient values, and another mask is created to be applied to each dense layer at testing time. Then, all but the frozen

parameters are reset to their initial values, and the process continues for the next task, until $t = T$ and the training is complete.

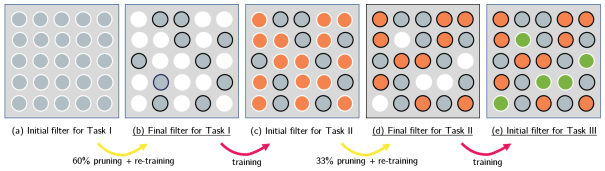

Fig. 3: *From [18] Diagram of PackNet for CNNs*

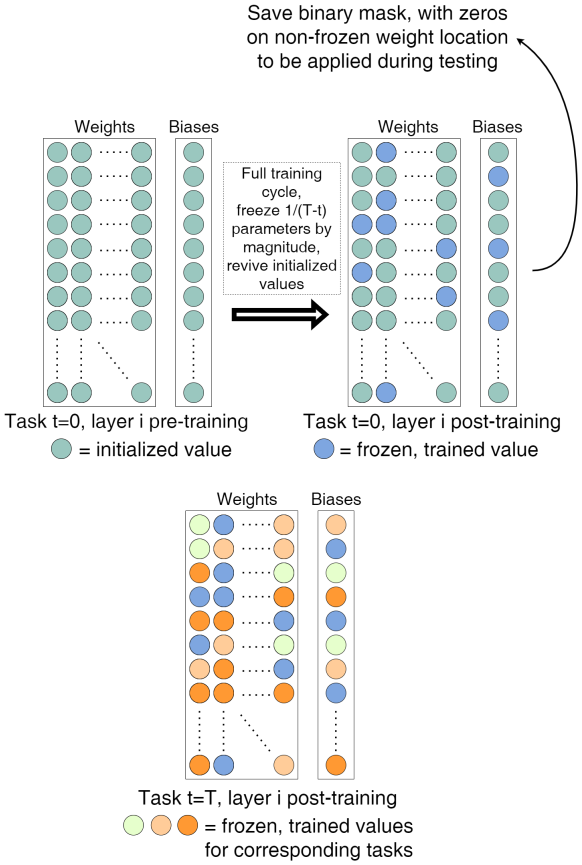

Fig. 4: *Our version of PackNet for DDPG*

D. Increasing Network Capacity by Storing Sparse Parameter Information

Catastrophic Interference for multiple tasks is not an easily avoidable problem, yet it does seem to be a natural and intuitive concept when trying to approximate functions for multiple environments. What is less intuitive is the interference that occurs when trying to learn a single task, where inputs to a network are of the same domain and result from observations of the same environment. We show a brief explanation of one view of this issue, and then propose a novel solution based on this view.

Assume we have a neural network which, for the sake of argument simplicity, is the actor within a DDPG model. We can represent this neural network as a mapping g from the state domain S to the action co-domain A, and parameterized by parameters $\theta \subset \Theta$. In training this network, we attempt to optimize θ such that, for any given set of states $S' \subset S$, $g(S'; \theta) \rightarrow A'$ where $A' \subset A$ maximizes rewards. Now assume we have two sets of states S_1, S_2 which by some metric d have all constituent states distant from all constituent states in the other set. Then there is an optimal set of parameters θ_1, θ_2 for each set, respectively, that allows g to map to the appropriate subset of the action co-domain for that state set (action subsets may very well overlap, and in practice often do). Training the network chronologically with S_1 first and S_2 second will cause $\theta \rightarrow \theta_1$ and then $\theta \rightarrow \theta_2$, so that $g(S_1; \theta) = g(S_1; \theta_2)$, and hence the problem of interference, as θ_1 is assumed to be the optimal set of parameters for S_1.

However, if we assume that it is possible for a network to exist which may approximately map both S_1 and S_2 to appropriate subsets of the action space, but no single set of parameters to allow such a mapping since the states are sufficiently distant from each other, then we propose the following solution:

Train to a set of parameters $\theta_3 \subset \Theta$ such that $\theta_1 \cup \theta_2 \subset \theta_3$, and add to g the additional parameter spaces $\phi \subset \mathbb{N}^{|\theta_\alpha|}$ and $\alpha \subset \mathbb{N}$. We define α to index the sets of states sufficiently distant from other sets of states and corresponding to parameter subset θ_α, and we define ϕ to represent the indices within θ_3 which correspond to θ_α. Thus we can simply train g with a shared set of parameters, keeping track of subsets of these parameters we want to use at evaluation time given a certain set of states, ignoring the remaining parameters. Since we assume the sets of states are, by some metric, distant from the other sets, we may use a clustering algorithm based on this sense of distance to identify each set and index them by α, producing the final mapping: $g(S_\alpha; \theta_3, \phi, \alpha)$.

The only remaining question is, why use ϕ at all, and not simply learn the individual θ_α? This question is largely answered by the technique used in [19]: storing only a set of indices is far easier and more efficient than attempting to store the equivalent of $max(\alpha)$ networks.

Our original plan was to extend the preceding justification to the case where the number of sets of states was arbitrarily greater than two to develop an algorithm that would cluster states via dimensionality reduction and then train sparse binary masks to be applied to a master network, in order to greatly diminish Catastrophic Interference within a single task. Due to time constraints, we were not able to implement such an algorithm, but we believe the justification in this section shows the potential for it to work well if implemented in the future.

IV. Results

In order to see the effectiveness of some of the more theoretical ideas, we chose a number of environments with which we could obtain benchmark results to compare with the results of our approaches. This will allow us to strip back some of the macroscopic issues that may arise from the more

complicated tasks, and simply look at the direct effects of these approaches.

A. *Sparse Pathway Encoding*

It is evident from our results (see Figure 12) that the environment has an important interaction with sparsity levels in a model. A simpler environment is much more resistant to sparsity and a more complex environment can benefit from a sparse representation (as PackNet shows) but it has to be more carefully constructed.

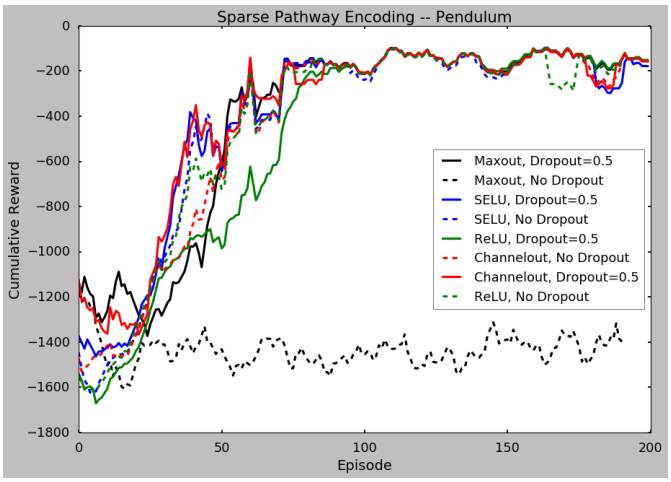

Fig. 5: *Cumulative rewards for SPE on Pendulum*

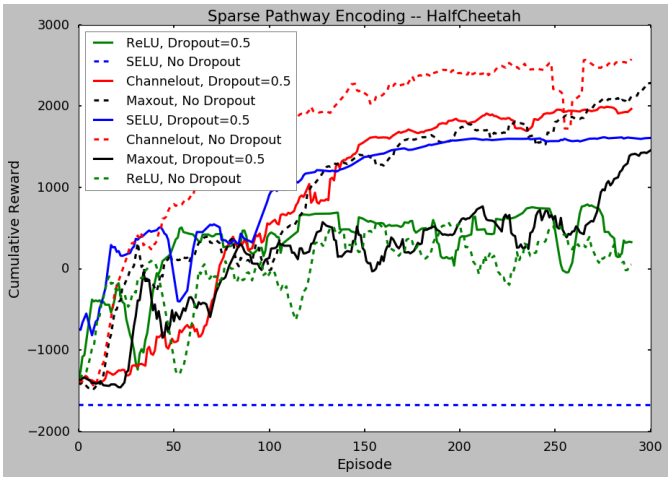

Fig. 6: *Cumulative rewards for SPE on HalfCheetah*

The individual results of training with the various activations and dropouts don't make the difference in learned representations very clear, but the results from testing do show a clear distinction:

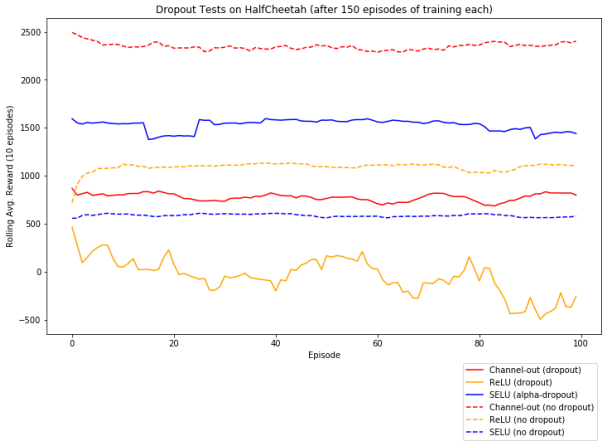

Fig. 7: *Rolling average rewards for SPE on HalfCheetah*

The first obvious note is that ReLU performs worse with dropout than without. This is expected, as dropout intentionally introduces some variance during training, a property that works well in supervised learning, but poorly in reinforcement learning. Channel-out, however, needs some properties of dropout to work properly, and yet it also performs poorly here when dropout is introduced. SELU needs dropout as well, but it uses a modified version, and hence here has the opposite trend, performing better when dropout is introduced. This tells us that the needed properties of dropout may still be accessed for an activation like channel-out, but we must rethink the approach, as traditional dropout has too significant of a drawback.

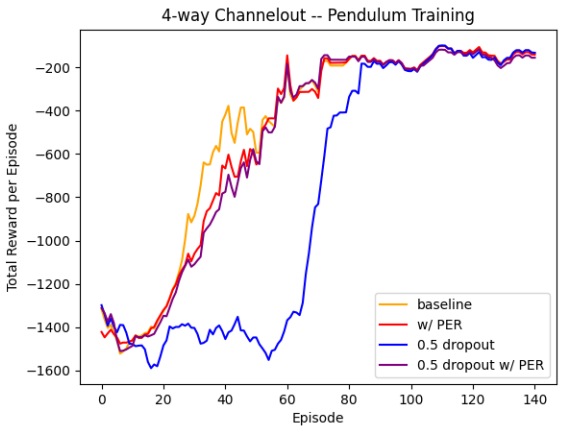

Fig. 8: *Cumulative rewards during training for Pendulum environment with Prioritized Experience Replay buffer*

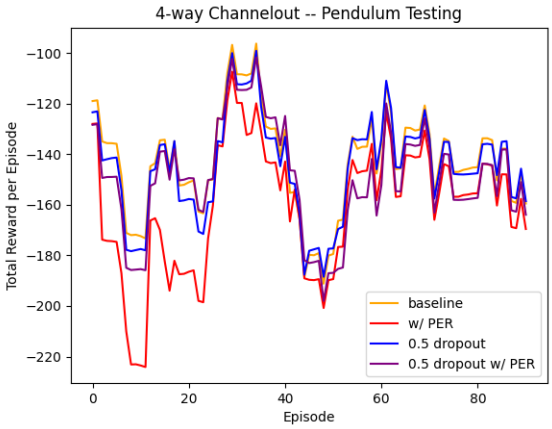

Fig. 9: *Cumulative rewards during testing for Pendulum environment with Prioritized Experience Replay buffer*

With a Prioritized Experience Replay buffer, channel-out trains a bit more stably, but testing results do not show as much promise. Since channel-out uses observation information to encode pathway-specific information and PER uses this information to sample observations with training, it is likely that channel-out does not see the variety of observations needed to encode all relevant pathway information. That is, channel-out needs to see important observations just as much as it needs to see unimportant observations during training.

B. Distributional Regularizers

ExpDR + ReLU is the only Distributional Regularizer model that worked well in our experimentation (see Figures 10 and 11), which is interesting for two reasons. One is that our hypothesis about GaussianDR and channelout was wrong, which isn't too surprising because channel-out can't necessarily guarantee zero-output; it can only do this at the pool-level. This is troublesome for using Distributional Regularizers for other functions because ReLU just so happens to have a property that works well for Distributional Regularizers, that of naturally and easily outputting zero (that is, for half of the domain). The second reason this result is interesting is still because of ReLU: in Liu's thesis, he found that Gaussian DR + ReLU also worked well, which we did not find ourselves. This may indicate that Distributional Regularizers in general work better for the DQN algorithm than they do with DDPG. More investigation would be needed to see this for sure, and if this were true, and important question is why? If it has something to do with discrete versus continuous action spaces, rather than the specific algorithms themselves, this would provide important information for how one develops sparse representations given a particular environment.

We see the opposite issue in applying PER to Distributional Regularizers as we did applying it to channel-out, as training is a bit less stable but it performs better in testing. The reasoning is just as symmetrical, as DR benefits from seeing more important observations during training, as the level of sparsity is high, hence only the most important information should be given the privilege of non-zero activation.

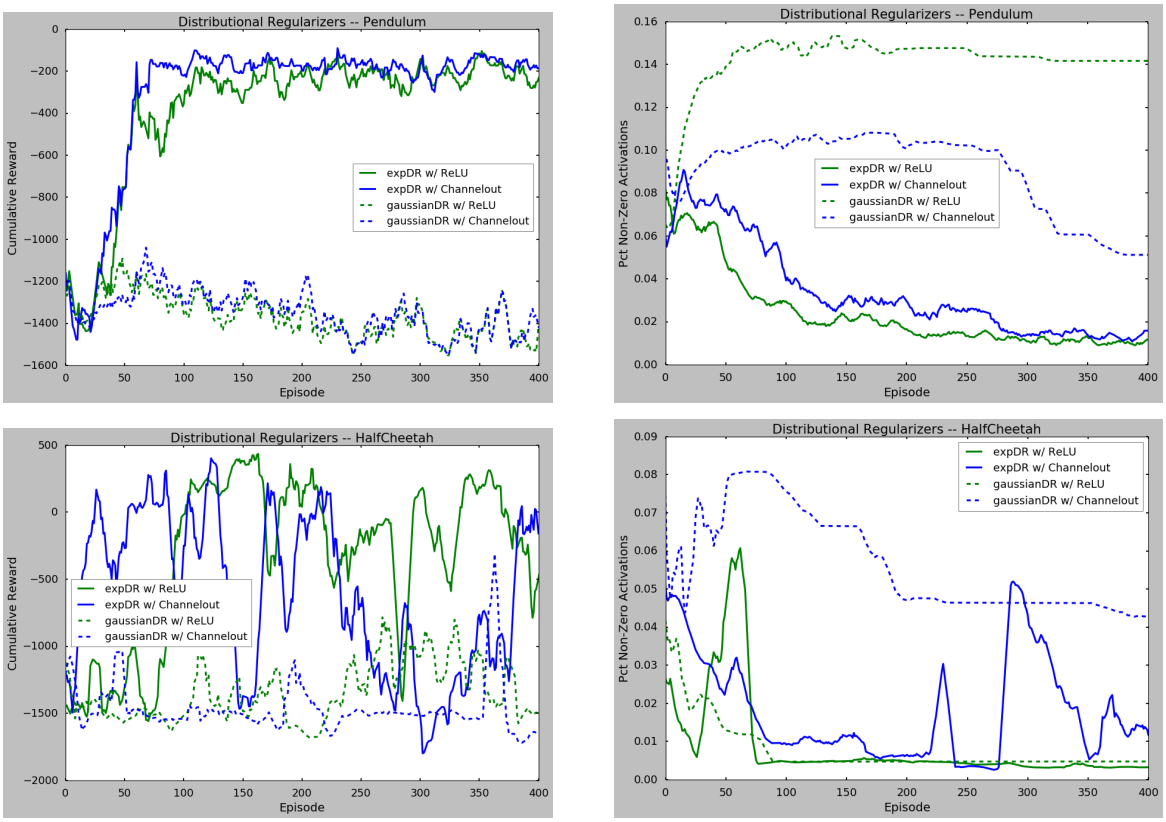

Fig. 10: Cumulative rewards (left column) and the percentage of non-zero activations (right column) for 2 different environments (top row: Pendulum, bottom row: HalfCheetah) under the distributional regularizer setup. Sparsity is achieved in both environments under each DR setup, however exponential DR yields better sparsity and higher rewards than gaussian DR in the same number of iterations.

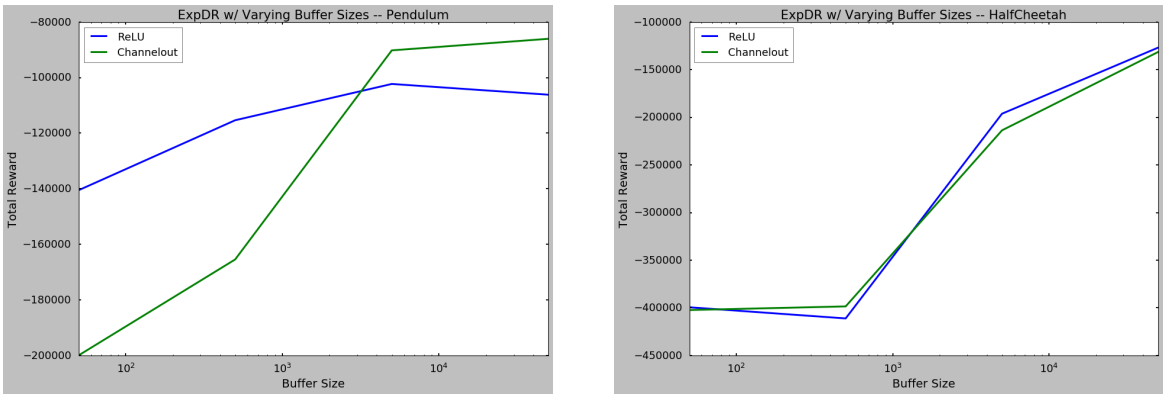

Fig. 11: Cumulative rewards for best performing DR (exponential) tested for 4 different experience replay buffer sizes (5e1, 5e2, 5e3, 5e4) for 2 different environments (left: Pendulum, right: HalfCheetah). Smaller buffer sizes lead to lower rewards, i.e., a narrow scope yields more correlated samples that prevent the agent from learning a broader, more representative policy.

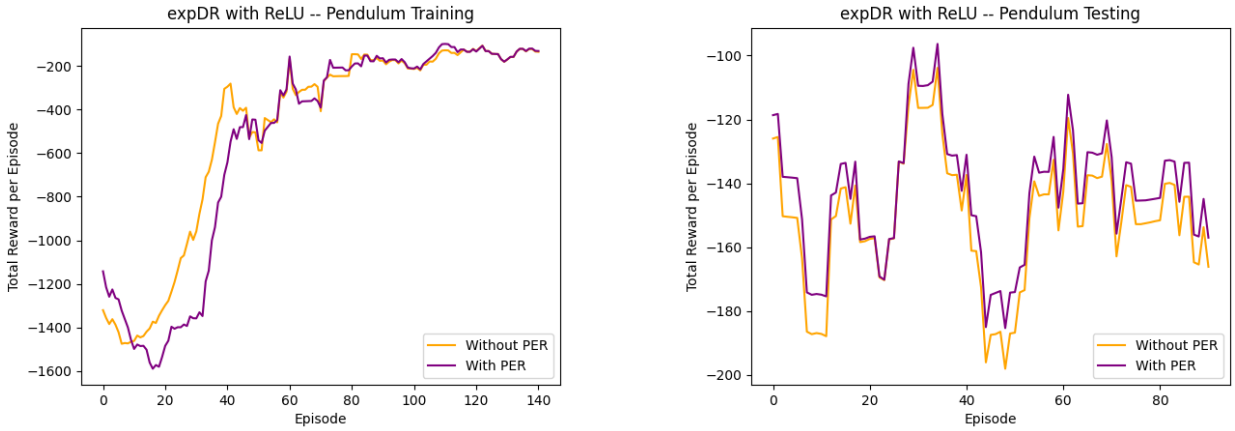

Fig. 12: Training and testing cumulative rewards for Pendulum environment; Comparing results with Prioritized Experience Replay buffer [20].

C. *Increasing Usage of Network Capacity with PackNet*

We implemented a modified PackNet algorithm (as discussed in the Methods section) for use with the DDPG learning algorithm in Tensorflow. Here, we demonstrate its ability to train moderately well over multiple random seeds of a single environment, and especially make the trained model test well on new seeds.

First, we trained PackNet with both ReLU and channel-out activations on three random seeds of the pendulum-v0 environment and compared the results to the same training scheme but with baseline ReLU and channel-out networks:

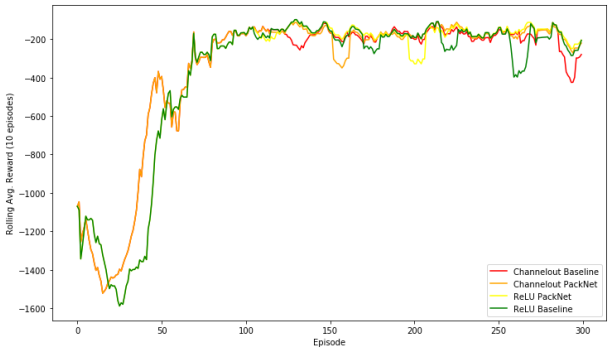

Fig. 13: *Training dynamics for three different random seeds of pendulum-v0. 100 episodes per seed.*

Clearly, these have very similar training dynamics, however the mean rewards are a bit more telling as to overall performance:

Model	Mean Reward
Channel-out Baseline	-351.41
Channel-out PackNet	**-344.58**
ReLU PackNet	-366.80
ReLU Baseline	-378.31

It can be seen by this that PackNet with channel-out has a slight edge over the baseline channel-out model, and both perform better than the ReLU models. It is to be expected that channel-out would in general train better over multiple seeds and a longer training time, as we expect information about specific states (notably as the environment is initialized in a random state) to be encoded into the network with channel-out, as discussed in "Methods." The fact that channel-out's PackNet version performs best is promising, but we further look at tests on the the trained models with the last seed trained and with a completely new, unseen seed:

Model	Mean Reward
Channel-out Baseline (last)	-151.76
Channel-out Baseline (new)	-149.01
Channel-out PackNet (last)	**-135.91**
Channel-out PackNet (new)	-149.37
ReLU PackNet (last)	-151.03
ReLU PackNet (new)	-161.27
ReLU Baseline (last)	-157.56
ReLU Baseline (new)	-148.78

As can be seen in the results in the above table, PackNet with channel-out performed markedly better than all other models when testing on the last task that was seen during training. This result comes even though training was fairly similar, and all models trained to very similar values. For this environment, PackNet with channel-out, the baseline channel-out, and the baseline ReLU all perform very similarly on the new seed, so we decided to test the models again on a slightly more advanced environment, PyBullet's HalfCheetah.

For the Cheetah environment, we only trained over two random seeds for all models due to the significantly longer training time, however we believe this actually yielded a stronger result. Training over just the two random seeds, the channelout baseline and PackNet models again trained similarly, but testing on PackNet with several new random seeds showed a clear improvement over the baseline.

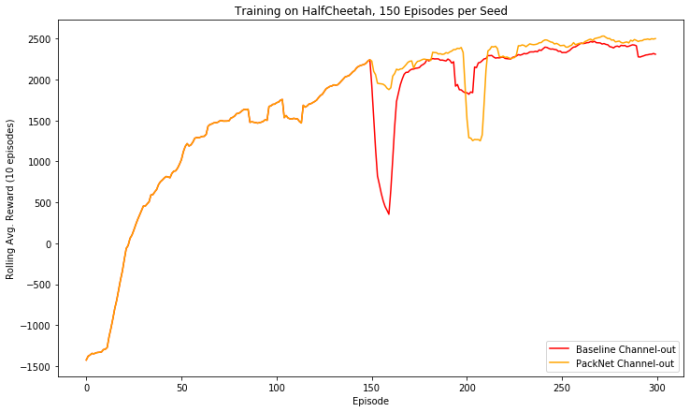

Fig. 14

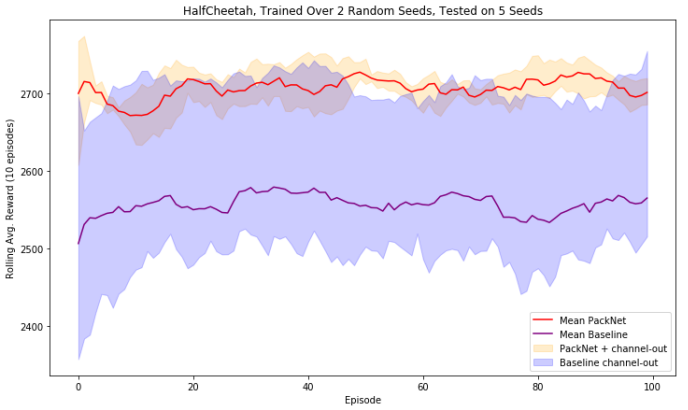

Fig. 15

Here it can be seen that over five runs with random seeds (the five unique seeds were tested on both models) there is a significant difference between the PackNet and baseline channelout models, and much greater consistency for PackNet. But to separate out what of this effect was from PackNet and what was from the addition of channel-out, we also show results for a baseline model and a PackNet model trained and tested the same way but with the ReLU activation function:

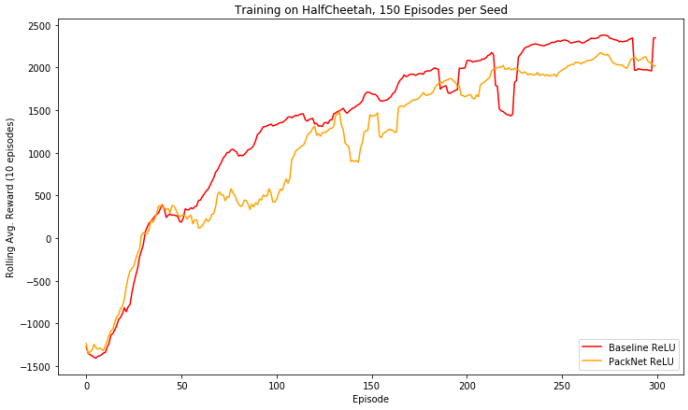

Fig. 16

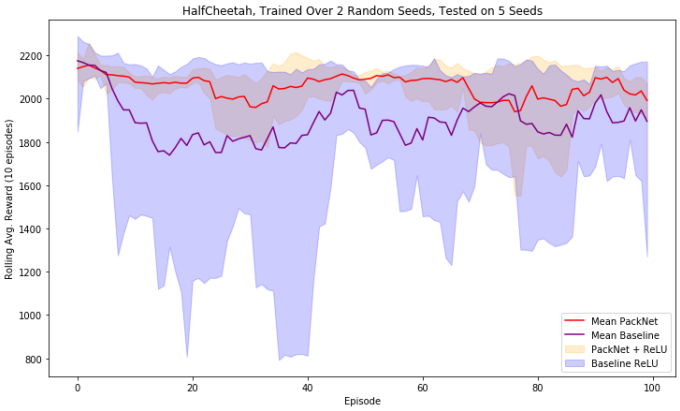

Fig. 17

The contrast between Figure 15 and Figure 17 makes the difference clear, as the distinction between the baseline and PackNet models diminished substantially with the use of ReLU instead of channel-out. The consistency of the PackNet results prevailed with the ReLU model, but the averages of each set of tests for ReLU were ultimately much closer together, even overlapping. This signifies that the combination of PackNet and channel-out is important; PackNet provides the means for improved network capacity usage, but in order for that capacity to be used *effectively*, channel-out must be in place to capture and encode information into the network pathways. Furthermore, PackNet may not be an all-around substitute for dropout in the realm of reinforcement learning, but it certainly could serve in the same role for activation functions, like channel-out and maxout, which require the extra capacity usage to work as intended. These results have only been gleaned from the two environments we had the time to test on, so there is certainly room for more experimentation before concrete claims could be made in general.

We had wanted to cast the preceding PackNet approach to situations with multiple different tasks, as opposed to just different seeds of the same task, but we were unfortunately not able to get a sufficient multi-task environment set up in time to test the multi-task properties. However, we believe the results of the single-task multi-seed cases demonstrate PackNet's ability to increase network capacity usage and enable channel-out's full pathway-encoding power. Hopefully this will in fact lead to better-than-vanilla multi-task learning, but only future investigation will tell.

V. Conclusions and Future Work

In this project, we investigated the effects of sparse representations in deep reinforcement learning as a remedy for catastrophic interference. We explored the dynamics of a DDPG model empowered with sparse pathway encoding via channel-out activation, distributional regularizers based on a KL-divergence penalty, and spatially packed representations with a modified PackNet. In experimentation with sparse pathway encoding, we found that though some activation functions that induce sparse representations require dropout to work

well in supervised learning, they perform worse with dropout in reinforcement learning due to the introduction of extra variance. However, we also saw that an activation function which requires a modified version of dropout still performs better with it, indicating that we need only re-think how we increase network capacity usage in order to get activations like channel-out to work effectively.

Our experimentation with Distributional Regularizers, we found that using a Gaussian distribution as a target with channel-out activation does not improve performance, and further that using a Gaussian target with ReLU performs worse in a DDPG than in a DQN. We suspect this is a result of either the algorithms themselves or that discrete and continuous action spaces interact with sparse representations differently, but we need to investigate further to find out.

Finally, we found that our modified PackNet must use an activation function like channel-out to work well, and when they are used together they are able to learn more consistent and better-performing representations for a single task than a model that has one or the other but not both. We claim that this is because channel-out captures observation information and encodes it into network pathways, while PackNet increases network capacity usage so the pathway information is more abundant and expressive. Further, we claim that PackNet can act as a limited alternative to dropout in Deep RL for some activation functions. Because of the motivation for these claims, we think that PackNet with channel-out may also perform well in a multi-task setting, but further testing is required to know for sure. This is in addition to the clustering + sparse trainable masks multi-task algorithm we would like to further explore, discussed in subsection *D* of the Methods.

REFERENCES

[1] A. Gaier and D. Ha, "Weight agnostic neural networks," 2019.

[2] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018.

[3] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020.

[4] V. Liu, R. Kumaraswamy, L. Le, and M. White, "The utility of sparse representations for control in reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4384–4391, 07 2019.

[5] J. F. Hernandez-Garcia and R. S. Sutton, "Learning sparse representations incrementally in deep reinforcement learning," 2019.

[6] M. McCloskey and N. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation - Advances in Research and Theory*, vol. 24, pp. 109–165, Jan. 1989.

[7] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," 2017.

[8] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," vol. 3720, pp. 317–328, 10 2005.

[9] L. ji Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," in *Machine Learning*, pp. 293–321, 1992.

[10] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *Ann. Statist.*, vol. 29, pp. 1189–1232, 10 2001.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[13] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, pp. 1481 – 1497, 10 1990.

[14] B. Ratitch and D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning," in *Proceedings of the 15th European Conference on Machine Learning (ECML 2004)* (J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, eds.), vol. 3201 of *Lecture Notes in Computer Science*, pp. 347–358, Springer, 2004.

[15] L. Le, R. Kumaraswamy, and M. White, "Learning sparse representations in reinforcement learning with sparse coding," 2017.

[16] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," 2013.

[17] Q. Wang and J. JaJa, "From maxout to channel-out: Encoding information on sparse pathways," 2013.

[18] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," 2017.

[19] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," 2018.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] V. Liu, "Sparse representation neural networks for online reinforcement learning," Master's thesis, University of Alberta, 2019.

[22] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, pp. 79–86, 03 1951.

[23] G. Zhang and H. Li, "Effectiveness of scaled exponentially-regularized linear units (serlus)," 2018.

[24] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.