Revisiting the Fuzzy Tilling Activation and How to Set its Hyperparameters

Muhammad Gohar Javed
Department of Electrical and Computer Engineering
University of Alberta
javed4@ualberta.ca

Tian Xiang Du
Department of Computer Science
University of Alberta
tdu@ualberta.ca

Amir Bahmani
Department of Computer Science
University of Alberta
bahmani1@ualberta.ca

Vlad Tkachuk
Department of Computer Science
University of Alberta
vtkachuk@ualberta.ca

Abstract

Sparsity has been shown to improve model performance on decision making problems with non-stationary data, such as online supervised learning and reinforcement learning (RL). Sparsity is when a large number of features in a neural network are approximately zero. The fuzzy tiling activation (FTA) has been proposed to enforce sparsity by design, and has been shown to outperform other activations, such as ReLU and $\tanh$, that do not enforce sparsity. However, a difficulty of using the FTA is that it is sensitive to a new *tiling bound* hyperparameter, which currently requires a search to be set effectively. In this work we do two things. First, we reproduce experiments comparing FTA with ReLU when using deep Q-learning (DQN) on the LunarLander RL environment, showing that indeed, for this environment FTA is no better than ReLU if the tiling bound is not set appropriately, Second, we empirically test if simple techniques for normalizing the activation values passed into the FTA cell can remove the need for search when setting the tiling bound.

1 Introduction

Neural networks (NN) have been shown to perform well on several challenging problems Brown et al. [2020], Krizhevsky et al. [2017], Mnih et al. [2013], Silver et al. [2017] Part of the reason for their success is due to their ability to learn good representations of the input data, which can then be used to effectively generalize. A representation of an input is defined as the values of all the neurons of the NN when the input is passed into the NN. The values of all the neurons are commonly called *features*. Unfortunately, when the NN is trained online (data recieved sequentially), interference can occur between the representations that are learned [Chandak et al., 2019, Caselles-Dupré et al., 2018, Madjiheurem and Toni, 2019]. Interference is loosely defined as when a newly learned representation affects the performance of the NN on previously seen data in a negative way (i.e. the old representation becomes worse due to the newly learned one). A potentially useful intuition for when interference can occur is when the representation for some input x_1 is dense (a large number of features are non-zero), since then, if another input x_2 is sufficiently different (uncorrelated) from x_1, its representation should likely be quite different. However, in order for a different representation to be learned for x_2 the NN will likely have to update the weights that contribute to the representation of x_1, thus making the representation for x_1 worse in order to learn a good representation for x_2.

A method that has been found to reduce interference, is to enforce sparse representations (a large number of features are approximately zero) [Liu et al., 2018, Javed and White, 2019, Rafati and

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Noelle, 2019]. We revisit the example above with inputs x_1 and x_2. If the representation for x_1 is sparse, then the representation that is learned for x_2 can use weights that are unused for the representation of x_1, thus causing no interference between thkkke features for x_1 and x_2. The best way to enforce sparsity is an active topic of research A detailed discussion of related works can be found in Section 2.

One way to enforce sparsity is by design, a method which does this is using a *fuzzy tiling activation* (FTA) [Pan et al., 2019]. FTA is an activation function that can be used at each layer of a NN, similar to ReLU or $\tanh$. Different from ReLU and $\tanh$ is that FTA has a vector output instead of a scalar output. FTA takes as input a scalar and outputs a one-hot vector. The output of FTA can be thought of as mapping a scalar $z \in \mathbb{R}$ to one of k evenly spaced bins in $\mathbb{R}^k$. A critical step to using FTA effectively is setting the tiling bound parameter u. This is because z should be in $[-u, u]$ to work well. However, just satisfying this is usually not enough for good performance (as one could easily set u very large). For instance if $u = 10$ and $k = 4$ then the bins would be $[-10, -5), [-5, 0), [0, 5), [5, 10]$ and if z really takes values in $[0, 1]$ then it will always map to the same bin (bin 3 since $[0, 1] \subset [0, 5)$). Thus, as discussed by [Pan et al., 2019], setting the tiling bound is important; however, no good solution exists for setting it other than performing a search. The sensitivity to the tiling bound was especially prominant in the work of [Pan et al., 2019] when using FTA with a deep Q-network (DQN) [Mnih et al., 2013] in the LunarLander RL environment.

As such, in this work we aim to do two things:

1. We aim to reproduce the FTA vs ReLU using DQN in LunarLander experiments presented in [Pan et al., 2019], to confirm that indeed FTA is sensitive to the tiling bound in this setting.
2. Next, we aim to test if the simple technique of normalizing the values (using $\tanh$, or batch norm) passed to FTA can remove the need for tuning the tiling bound parameter in FTA.

Experiments related to the first point are shown in section 4.1.

To answer the question in the second point we normalize the values passed into FTA using two methods: passing them into a $\tanh$, using batch norm. The method of using $tanh$ to normalize the input values was discussed in Pan et al. [2019], but not tested. Pan et al. [2019] mentioned that using this method might suffer from vanishing gradients, which we hope to confirm from this experiment. We hypothesis that normalizing the values should solve the need to set tiling bounds using search, since normalizing will ensure that then values passed into FTA are in $[-1, 1]$, thus the tiling bound can always be set to $[-1, 1]$. Experiments related to the second point are shown in section 4.2.

2 Background

Representation learning can influence learning efficiency, positively through generalization but negatively through interference [Bullinaria, 1995, Rebuffi et al., 2016, Le et al., 2017, Liu et al., 2018]. Learned representations using neural networks are especially vulnerable to interference due to aggressive generalization, as updates to one feature's estimates change the estimates of later features in unexpected ways, degrading model accuracy. Interference in features learned by neural networks is even worse when trained on temporally correlated data [Liu et al., 2020, Bengio et al., 2020, Zhang et al., 2022]

Several authors [Ghiassian et al., 2020, Liu et al., 2018, Javed and White, 2019, Hernandez-Garcia and Sutton, 2019] have shown that sparse representations can reduce interference in learning feature updates: only a small number of features are active for any given input, and thus each update is less likely to interfere with the weights of other features. Sparse feature spaces can still be highly expressive and interpretable, as outputs can be traced to the activation of the most input-discriminative features. However, learning such sparse representations in the online setting remains an open problem.

Existing strategies to ensure sparsity are to pre-train representations offline using regularizers [Liu et al., 2018] or meta-learning Javed and White [2019], or online methods using regularizers with replay buffers Hernandez-Garcia and Sutton [2019]. While sparsity regularizers do improve learning, they often lead to high levels of dead neurons and thus degrades the learned representation over time [Hernandez-Garcia and Sutton, 2019]. Kernel representations can also provide sparse representations online, but do not scale well to large problems due to its computational complexity [Pan et al., 2019].

Therefore, there is motivation to find a simpler method to ensure sparsity in learned representations that is easy to implement, train, and scale to larger input spaces.

FTA is an approach to achieve sparsity by design, where an activation function that can be applied to any layer of a neural network provides adjustable sparsity without the need for pre-training or adjustments to the loss function. The FTA function aggregates inputs of any feature layer into bins with differentiable, overlapping zones. Much like Tile Coding [Sherstov and Stone, 2005], FTA allows learning to be discriminative between different input values through binning, but unlike Tile Coding, FTA maintains input generalization through learning activations within the differentiable regions between bins. When applied to DQN and Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015], FTA has been empirically shown to improve agent performance for both discrete and continuous control environments, often outperforming activations like ReLU and Tanh [Pan et al., 2019]. Furthermore, FTA-applied networks can often perform better without target networks, circumventing the sample inefficiency problem inherent in their use [Pan et al., 2019]. A recent study by Wang et al. [2022] on transferring learned representations to networks performing auxiliary tasks found that FTA-based representations transferred better and more consistently than ReLU-based representations for less similar tasks.

3 Preliminaries

In this section we introduce the Fuzzy Tiling Activation (FTA) and formally define the RL setting.

3.1 Fuzzy Tiling Activation

First we introduce the simpler *tiling activation* (TA) and then extend it to the FTA. A TA is a function $\phi : \mathbb{R} \to \mathbb{R}^k$, which expects inputs $z \in [-u, u]$ and maps them to one-hot vectors (standard basis) in $\mathbb{R}^k$. The $[-u, u]$ is referred to as the tiling bound. For example, if $u = 10$ and $k = 4$, then any $z \in [-10, -5)$ would be mapped to $(1, 0, 0, 0) \in \mathbb{R}^k$, any $z \in [-5, 0)$ would be mapped to $(0, 1, 0, 0)$, any $z \in [0, 5)$ would be mapped to $(0, 0, 1, 0)$, and any $z \in [5, 10]$ would be mapped to $(0, 0, 0, 1)$. Pan et al. [2019] show that the TA can be implemented efficientely as follows. Assume you want evenly spaced bins of size $\delta > 0$, and $k = 2u/\delta$, where WLOG u was chosen such that it is divisable by δ. Define the tiling vector

$$\mathbf{c} = (-u, -u + \delta, l + 2\delta, \ldots, u - 2\delta, u - \delta) \in \mathbb{R}^k$$

Then the TA can be defined as

$$\phi(z) = \mathbb{1} - I_+(\max(\mathbf{c} - z\mathbb{1}, 0) + \max((z - \delta)\mathbb{1} - \mathbf{c}, 0))$$

where $\mathbb{1} \in \mathbb{R}^k$, and $I_+(\cdot)$ is an indicator function which returns 1 if the input is positive and zero otherwise, and is applied element wise to vectors.

An issue with the TA is that it has zero derivative almost everywhere. In order to solve this issue Pan et al. [2019] proposed modifying the TA for a fuzzy tiling activation (FTA). The FTA can be implemented as follows. Define a fuzzy version of the indicator function as

$$I_{\eta,+} = I_+(\eta - x)x + I_+(x - \eta)$$

where $\eta \geq 0$. Then the FTA can be defined as

$$\phi_\eta(z) = \mathbb{1} - I_{\eta,+}(\max(\mathbf{c} - z\mathbb{1}) + \max((z - \delta)\mathbb{1} - \mathbf{c}, 0)).$$

where $I_{\eta,+}$ is applied element wise to vectors The reason for the use of fuzzy in the name of FTA can be understood from the definition of $I_{\eta,+}$. The term $I_+(\eta - x)$ is 1 when $x < \eta$, while when $x > \eta$, $I_+(x - \eta)$ is 1. Thus, when $x < \eta$ we have that $I_{\eta,+}$ evaluates to x, giving a smoother (fuzzy) indicator function. Notice, that this means for $x < \eta$ the derivative is non-zero, allowing us to use backpropagation to train out networks. Also, when $\eta = 0$ the original indicator function I_+ can be recovered.

3.2 Reinforcement Learning

The reinforcement learning (RL) setting can be formulated as an markov decision process (MDP). Formally an MDP is characterized by the tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma)$, where $\mathcal{S}$ is that state space, $\mathcal{A}$ is the

action space, $\mathbb{P}$ is the transition probability kernel, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. A policy $\pi : \mathcal{S} \to \mathcal{M}_1$ is a map from states to distributions over actions, where $\mathcal{M}_1$ is the set of all probability measures over actions $\mathcal{A}$. Then, an agent (policy) interacts with an MDP as follows. At each time step $t \in 1, 2 \cdots$ the agent observes a state $s_t \in \mathcal{S}$. The agent takes an action according to its policy $a_t \sim \pi(s_t)$. The agent then transits to its next state according to $s_{t+1} \sim \mathbb{P}(s_t, a_t)$, and recieves a reward $R(s_t, a_t, s_{t+1})$.

An action value function under policy π is defined as

$$Q_\pi(s, a) = \mathbf{E}[G_t | S_t = s, A_t = a; A_{t+1:\infty} \sim \pi]$$

where $G_t = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$ is the return. The goal of the agent is maximize its expected reward from each state in $\mathcal{S}$.

4 Experiments

The LunarLander environment, from OpenAI Gym [Brockman et al., 2016] is a trajectory rocket optimization problem, where the goal is to land a moon lander on a landing pad. The state space of the environment $\mathcal{S}$ is compromised of 8 attributes; x and y coordinates, v_x and v_y horizontal and vertical velocities, θ and v_θ angle and angular velocity, and 2 booleans to show if the left or right leg is touching the ground. At each timestep t the action space $\mathcal{A}(s_t)$ = {`do nothing`, `fire left engine`, `fire main engine`, `fire right engine`}. The rewards for landing on the landing pad is between 100 to 140 points, with an additional 100 points if it stops on it, hence a solved episode is +200 points. Firing the main engine has -0.3 and the side engines -0.03 points penalty, and the lander gets -100 points if it crashes. In the event that the agent crashes, leaves the viewport, or remains stationary without touching another body, the episode will terminate.

In our experiments, we use Deep Q-Learning (DQN) [Mnih et al., 2013]. As explained in section 3.2, the goal of the agent is to interact with the environment in a way that maximizes its expected reward.

the optimial action value function under policy π is defined as:

$$Q^*(s, a) = \max_\pi \mathbf{E}[G_t | S_t = s, A_t = a; A_{t+1:\infty} \sim \pi]$$

In Q-Learning action value function is iteratively updated by using Bellman equation to converge to the optimal action value function:

$$Q_{t+1}(s, a) = \mathbf{E}[r + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) | S_t = s, A_t = a]$$

This converges to optimal action value function, $Q_t \to Q^*, t \to \infty$. A neural network is used as function approximator with weights θ to estimate action value function, $Q(s, a; \theta) \approx Q^*(s, a)$, and it is trained by minimizing the loss function, that changes by each iteration i:

$$L_i(\theta_i) = \mathbf{E}_{s,a\sim\rho(.)}\big[(y_i - Q(s, a; \theta_i))^2\big],$$

Where $y_i = \mathbf{E}_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$. The loss is computed by taking the expectation over the state-action pairs, s, a, that are sampled from the current *behavior policy* $\rho(s, a)$. The expected value of the squared difference between the target value y_i and the predicted action-value $Q(s, a; \theta_i)$ is then computed. The target value y_i is a measure of the long-term rewards that can be obtained by taking a given action in a given state. It is computed by taking the expected value of the sum of the immediate reward r and the discounted future rewards $\gamma \max_{a'} Q(s', a'; \theta_{i-1})$, where s' is a future state that is obtained by taking action a in state s, and a' is the action that maximizes the expected future rewards. The expectation is taken over the distribution of future states s' obtained by sampling from the environment with some probability ε.

Agent's experience at each timestep is stored into a *replay memory*, a random minibatch of expriences are samepled randomly from this memory to calculate the loss function, and the neural network is trained by performing a gradient descent on the loss function.

4.1 Reproducibility Experiment

Following the same settings as in Pan et al. [2019], we use a two-layer neural network with different acivation function in the last hidden layer as the main difference. For DQN and DQN-Large, we

use ReLU activation function in all the layers, with the exception of the last layer which is linear. Since DQN with FTA activation function expands the number of features, its last hidden layer is $k = (l - u)/\delta$ times bigger than the DQN, Hence we use DQN-Large with the last hidden layer of the same size of DQN-FTA with and without the target network.

For DQN-FTA in the last layer we set $[l, u] = [-20, 20]$ and $\eta = \delta = 2.0$, hence $k = 40/2 = 20$.

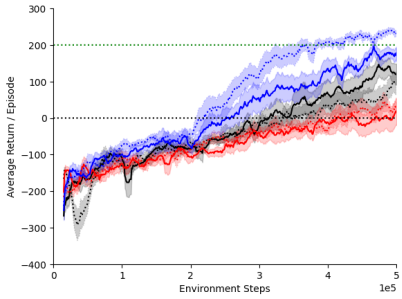

Figure 1: Evaluation learning curves of **DQN-FTA(black)**, **DQN(red)** and **DQN-Large(blue)** . The **dotted** line shows the algorithms trained with target network. The results are averaged over 10 runs, with shaded area showing the standard error.

In figure 1, we observe that, while DQN-FTA with and without the target network performs slightly better than DQN, DQN-Large outperforms both DQN and DQN-FTA.

To show the sensitivity of tiling bounds, we repeat the experiment for DQN-FTA with different values of $u \in \{0.1, 1, 10, 50, 100\}, l = -u$ with different number of tiles $k \in \{16, 24, 128\}$ for each tiling bounds.

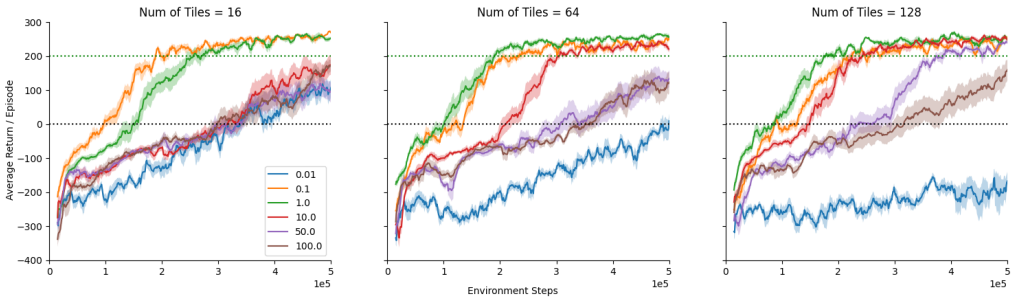

Figure 2: Evaluation learning curves of DQN-FTA using different values of tiling bounds: $u \in \{0.1, 1, 10, 50, 100\}, l = -u$, with different number of tiles $k \in \{16, 64, 128\}$. The results are averaged over 10 runs.

It can be seen that FTA in LunarLander performs the best, when we set u to a reasonably small value. However, it performs poorly when it is set to a large or very small number. With $u \in \{10, 50, 100\}$, performance imporves by increasing the number of tiles k, but it worsens when u is very samll.

We found FTA sensitive to tiling bound, Hence by coosing the the right value of u, We observe that it ourperforms both DQN and DQN-Large. Figure 3 shows DQN-FTA without target network and $u = 1, k = 64$, compared to other settings.

4.2 Normalizing Experiment

In our experiments, detailed in section 4.1, it can be observed that the performance of FTA is quite sensitive to the tiling bound $[-u, u]$. This reinforces the observation made in Pan et al. [2019]. In figure 2 we can see that the best performance on LunarLander was achieved with $u = 1$, and it got worse as we increased or decreased u. It can be inferred intuitively, that for a small value of u, many inputs to FTA may be out of its range providing zero gradients. On the other hand, for a large value of u, many inputs may activate the same tile. Both resulting in many dead neurons and

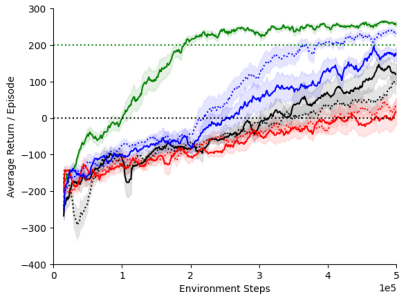

Figure 3: Evaluation learning curves of **DQN-FTA(black)** ($u = 20, k = 40$), **DQN(red)**, **DQN-Large(blue)** and **DQN-FTA(green)**($u = 1, k = 64$). The **dotted** line shows the algorithms trained with target network.

increasing interference. This means there is a specific value of u which would work best for any specific environment. Since its not straightforward to observe the range of outputs from a Neural Network layer, u can not be set mathematically. It has to be tuned manually by sweeping over a range of values in multiple experiments, which increases the cost of a project.

We hypothesize that if the inputs to FTA were scaled to a certain range, the performance would be less sensitive to the tiling bounds. We try two different methodologies to achieve this. First, using a Batch Normalization [Ioffe and Szegedy, 2015] layer before FTA. Second, using a $tanh$ activation layer before FTA.

Batch Normalization scales its input x to a learned mean β and variance γ. It is defined as

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \tag{1}$$

By using a Batch Norm layer before FTA, we expect it to learn the best γ and β for any particular tiling bound of FTA.

On the other hand, $\tanh$ is an activation function which maps the input to a continuous range of values between -1 and 1. The larger the input, the closer the output is to 1 and the smaller the input, the closer the output is to -1. This range of outputs can be controlled by multiplying them with a scalar. It is defined as

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$

The motivation behind using $tanh$ is to control the range of inputs to FTA, so that a standard tiling bound of $u = -l = 1$ would work in every case.

To test these hypothesis, we run two experiments each with $tanh$ and Batch Norm as the layer preceding FTA on Lunar Lander. One, with the best performing FTA tiling bounds and two, with the worst performing FTA tiling bounds. All other parameters and configurations are kept the same as those in section 4.1. Figure 4 shows the evaluation learning curves plotting episodic return versus environment time steps for all four of these experiments.

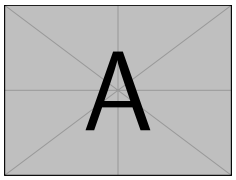

Figure 4: Evaluation learning curves plotting episodic return versus environment time steps for Batch Norm and $tanh$ as layers preceding FTA.

5 Discussion

In this paper, we investigate the properties of FTA through empirical experimentation as an extension to the original FTA paper by [Pan et al., 2019]. We first reproduced the learning curve evaluation for DQN-FTA, DQN, and DQN-Large in LunarLander, and found similar trends. When comparing the untuned FTA agent ($u = 20$) against the tuned ReLU agents, we found that FTA and DQN-Large reached higher average returns by 5×10^5 time steps than Pan et al. [2019]. It is currently unknown as to why our FTA and DQN-Large implementations perform better than the original paper. There are many potential reasons: stochasticity in our runs and initialization, our specific seeds used to evaluate performance, or even the implementation framework (we used PyTorch while Pan et al. [2019] used TensorFlow) could all influence the empirical outcome of experiments. While we only average over 10 runs compared to the 20 performed in the original paper, this should not have contributed to the performance discrepancy given our low standard error between runs. We will contact the original author to address any implementation differences, but further investigation into the plausible causes and further experimentation are required.

Our second experiment reproduces the bound sensitivity experiment comparing FTA with different tiling bounds, but also across different numbers of tiles. Our results share the conclusion that moderately small values for u ($u \in 0.1, 1$) perform better than the extremes ($u \in 0.01, 10, 50, 100$) [Pan et al., 2019]. We also find that increasing the number of tiles improves the performance of larger tiling bounds ($u \in 50, 100$) while an extremely small u gets worse ($u = 0.01$). This is expected, as increasing the number of tiles given a large tiling bound reduces δ, thereby reducing interference between bins. In addition, increasing the number of tiles given an extremely small tiling bound greatly increases $delta$ and results in high interference between bins.

To address the bound selection problem, we design normalizing experiments using $tanh$ activation and batch normalization. Pan et al. [2019] mention that $tanh$ would not be an effective strategy for ensuring $z \in [l, u]$ given the gradient vanishing problem, so we will provide empirical data to evaluate this hypothesis. With a principle focus of FTA being its ease of use, we propose using batch normalization to automatically bound $z \in [l, u]$ without additional tuning.

Lastly, we will reproduce the learning curves comparing FTA against ReLU for Acrobot, MountainCar, and Cartpole, to investigate if our performance discrepancy exists in other environments.

References

E. Bengio, J. Pineau, and D. Precup. Interference and generalization in temporal difference learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 767–777. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/bengio20a.html.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

J. A. Bullinaria. Representation , learning , generalization and damage in neural network models of reading aloud. 1995.

H. Caselles-Dupré, M. Garcia-Ortiz, and D. Filliat. Continual state representation learning for reinforcement learning using generative replay. *arXiv preprint arXiv:1810.03880*, 2018.

Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. Thomas. Learning action representations for reinforcement learning. In *International conference on machine learning*, pages 941–950. PMLR, 2019.

S. Ghiassian, B. Rafiee, Y. L. Lo, and A. White. Improving performance in reinforcement learning by breaking generalization in neural networks. *CoRR*, abs/2003.07417, 2020. URL https://arxiv.org/abs/2003.07417.

J. F. Hernandez-Garcia and R. S. Sutton. Learning sparse representations incrementally in deep reinforcement learning. *ArXiv*, abs/1912.04002, 2019.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

K. Javed and M. White. Meta-learning representations for continual learning. *CoRR*, abs/1905.12588, 2019. URL http://arxiv.org/abs/1905.12588.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

L. Le, R. Kumaraswamy, and M. White. Learning sparse representations in reinforcement learning with sparse coding. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2067–2073, 2017. doi: 10.24963/ijcai.2017/287. URL https://doi.org/10.24963/ijcai.2017/287.

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, 09 2015.

V. Liu, R. Kumaraswamy, L. Le, and M. White. The utility of sparse representations for control in reinforcement learning. *CoRR*, abs/1811.06626, 2018. URL http://arxiv.org/abs/1811.06626.

V. Liu, A. White, H. Yao, and M. White. Towards a practical measure of interference for reinforcement learning. *CoRR*, abs/2007.03807, 2020. URL https://arxiv.org/abs/2007.03807.

S. Madjiheurem and L. Toni. Representation learning on graphs: A reinforcement learning application. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3391–3399. PMLR, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Y. Pan, K. Banman, and M. White. Fuzzy tiling activations: A simple approach to learning sparse representations online. *arXiv preprint arXiv:1911.08068*, 2019.

J. Rafati and D. C. Noelle. Learning sparse representations in reinforcement learning. *arXiv preprint arXiv:1909.01575*, 2019.

S. Rebuffi, A. Kolesnikov, and C. H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016. URL `http://arxiv.org/abs/1611.07725`.

A. A. Sherstov and P. Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and L. Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31882-8.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359, 2017.

H. Wang, E. Miahi, M. White, M. C. Machado, Z. Abbas, R. Kumaraswamy, V. Liu, and A. White. Investigating the properties of neural network representations in reinforcement learning, 2022. URL `https://arxiv.org/abs/2203.15955`.

T. Zhang, X. Wang, B. Liang, and B. Yuan. Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. doi: 10.1109/TNNLS.2022.3162241.