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 neurons 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. Our experiments indicate that normalizing via tanh activation is most effective and can possibly remove the need for search to set 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 (except the neurons of the final layer) 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 neurons 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

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

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 neurons are approximately zero) [Liu et al., 2019, Javed and White, 2019, Rafati and 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 the representations for x_1 and x_2. The best way to enforce sparsity is an active topic of research. A detailed discussion of related work can be found in Section 1.1.

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 $\tanh$, batch norm, or range 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 3.1, indicating that the experiments were reproducable and nearly identical results were obtained as in [Pan et al., 2019].

To answer the question in the second point we normalize the values passed into FTA using three methods: passing them into a $\tanh$, using batch norm, and using range norm (all to be defined in Section 3.2). 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, however we did not see any indications of this in our experiments, and obtained very positive results for $tanh$. We hypothesis that normalizing the values should solve the need to set tiling bounds using search, since normalizing will ensure that the values passed into FTA are in $[-1, 1]$, thus the tiling bound can always be set to $[-1, 1]$. We found that $tanh$ was most effective, while batch norm and range norm did not perform as well. Experiments highlighting these results are shown in Section 3.2.

1.1 Related Work

Representation learning can influence learning efficiency, positively through generalization but negatively through interference [Bullinaria, 1994, Rebuffi et al., 2017, Le et al., 2017, Liu et al., 2019]. Learned representations using neural networks are especially vulnerable to interference due to aggressive generalization, as updates to one features 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., 2019, 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 ex-

pressive 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., 2019] 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.

2 Preliminaries

In this section we introduce the Fuzzy Tiling Activation (FTA) and the Rectified Linear Unit (ReLU) activation. Moreover, we formally define the RL setting and the Deep Q-learning (DQN) algorithm which we use for our experiments.

2.1 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.

2.2 ReLU Activation

We will be comparing FTA to the ReLU activation later. Thus we formally introduce it now. ReLU is defined as the positive part of its argument. Formally

$$\phi(z) = \max(0, z)$$

2.3 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 $\mathcal{S}$.

2.4 Deep Q-Learning

In our experiments, we use Deep Q-Learning (DQN) [Mnih et al., 2013]. As explained in section 2.3, the goal of the agent is to interact with the environment in a way that maximizes its expected reward. The optimial action value function is defined as

$$Q^*(s, a) = \max_\pi Q_\pi(s, a)$$

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

$$Q_{t+1}(s, a) = \mathbf{E}[R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_t(s_{t+1}, a_{t+1}) | S_t = s, A_t = a, S_{t+1} = s_{t+1}]$$

This converges to optimal action value function, $Q_t \to Q^*, t \to \infty$. A NN is used as a function approximator with weights θ to estimate an action value function, $Q(s, a; \theta)$. The agent's experience $(s_{t-1}, a_{t-1}, s_t, R(s_{t-1}, a_{t-1}, s_t))$ at each timestep t is stored into a $replay\ memory$. A minibatch $\mathcal{B}_t$ of experiences is sampled randomly from the replay memory to calculate the loss function.

The NN is trained by minimizing the following loss function at each time step t and then performing a gradient descent update

$$L_t(\theta_t) = \sum_{s,a,r,s' \in \mathcal{B}_t} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{t-1}) - Q(s, a; \theta_t) \right]^2.$$

The agents behaviour policy is an ϵ-greedy policy, which takes a random action with probability ϵ and is greedy with respect to $Q(s_t, \cdot; \theta_t)$ otherwise.

3 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 $\mathcal{A}(s_t) = \{\texttt{do nothing}, \texttt{fire left engine}, \texttt{fire main engine}, \texttt{fire right engine}\}$. 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.

We use a student's t-distribution to display 95% confidence intervals for all of the plots in this section. The confidence intervals appear as the shaded regions in the plots. The validity of using such confidence intervals is investigated in Section 3.3.

3.1 Reproducibility Experiment

Following the same settings as in Pan et al. [2019], we use a two-layer neural network with a 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 $k = 40/2 = 20$.

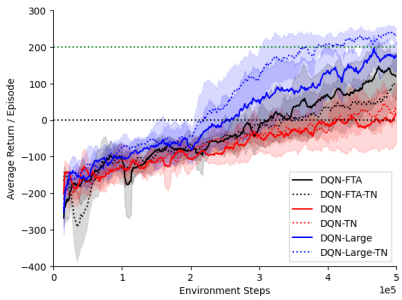

Figure 1: Evaluation learning curves of **DQN-FTA(black)**, **DQN(red)** and **DQN-Large(blue)** . The **dotted** line shows the algorithms trained with target network. The results are averaged over 10 runs.

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. This is the same behaviour that was observed by [Pan et al., 2019] for this experiment, thus we were able to reprodue the experiment.

To show the sensitivity of DQN-FTA to the tiling bounds, we repeat the experiment for DQN-FTA only, 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 depicts the results of the experiment. It can be seen that DQN-FTA in LunarLander performs the best, when we set u to a reasonably small value (approximately $u = 1$). However, it performs poorly when it is set too large ($u = 10, 50, 100$) or small ($u = 0.01$). With $u \in \{10, 50, 100\}$, performance improves by increasing the number of tiles k, but it gets worse when u is very small. The improved performance when the number of tiles k is increased can be understood by looking at the case when $u = 10$ and $k = 128$. Notice that $u = 10$ is 10 times larger than $u = 1$ and $k = 128$ is approximately 10 times larger than $k = 16$. Thus, for the range $[-1, 1]$ there will be approximately the same number of bins when $u = 10, k = 128$ as when $u = 1, k = 16$. This explanation explains why the green line ($u = 1$) when $k = 16$ is approximately the same as the red line ($u = 10$) when $k = 128$.

We found FTA sensitive to the tiling bound (u), hence only when choosing the the right value of u we observed that it outperforms both DQN and DQN-Large. We found that the best value $u = 1$ for DQN-FTA without a target network was $u = 1$. Figure 3 shows a plot of the above described

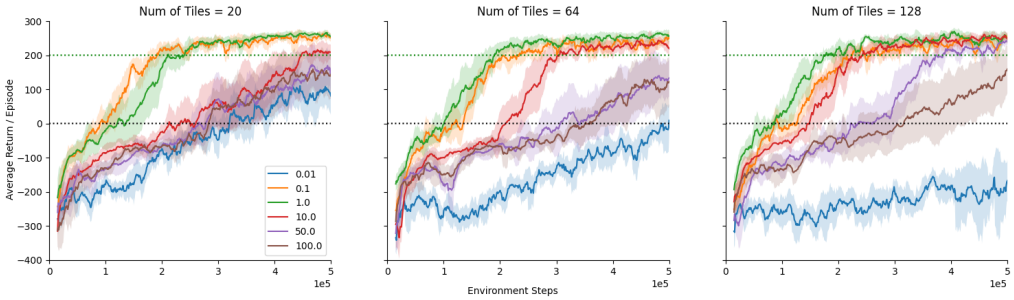

Figure 2: Evaluation learning curves of DQN-FTA using different values of tiling bounds: $u \in \{0.1, 1, 10, 50, 100\}, l = -u$, with different number of tiles $k \in \{16, 64, 128\}$. The results are averaged over 10 runs.

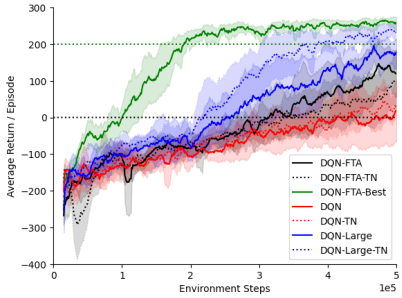

Figure 3: Evaluation learning curves of **DQN-FTA(black)** ($u = 20, k = 40$), **DQN(red)**, **DQN-Large(blue)** and **DQN-FTA(green)**($u = 1, k = 64$). The **dotted** line shows the algorithms trained with target network.

DQN-FTA network compared to the untuned DQN-FTA ($u = 20$) and DQN using ReLU activation instead of FTA. This clearly indicates that when the value of u is set appropriately, DQN-FTA can outperform DQN with ReLU activation.

3.2 Normalizing Experiment

In our experiments, detailed in section 3.1, 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 increasing interference. This means there is a specific value of u which would work best for any specific environment. Since it is 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 three 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. And finally, using a Range Normalization 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.

Range Normalization is used to scale the values to fall within a specific range. We use Range Normalization to scale the inputs to FTA, to the range of -1 and 1. It is defined as

$$y = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot 2 - 1 \tag{3}$$

Similar to $tanh$, the reason for using Range Normalization is to ensure the range of inputs to FTA is between -1 and 1.

To test these hypothesis, we run experiments with $tanh$, Batch Norm and Range Norm as the layer preceding FTA on Lunar Lander. One, with the best performing FTA tiling bound ($u = 1$) and two, with the worst performing FTA tiling bounds ($u = 0.01, 100$). All other parameters and configurations are kept the same as those in section 3.1.

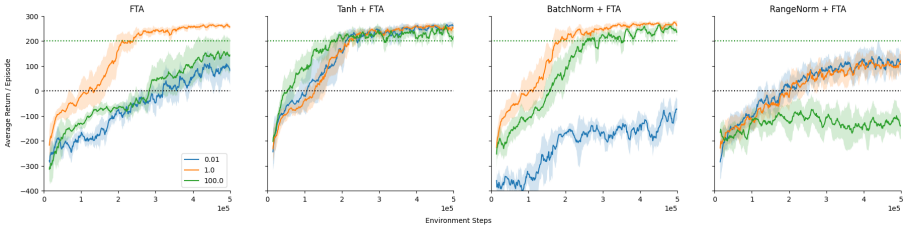

Figure 4: Evaluation learning curves plotting episodic return versus environment time steps for Batch Norm, Range Norm and $tanh$ as layers preceding FTA, compared to DQN-FTA without a normalization layer preceding FTA.

Figure 4 depicts the results of experiments comparing the performance of $tahn$, batch normalization and batch normalization as layers preceding FTA and DQN-FTA without normalization with the best and worst-performing tiling bounds in the LunarLander environment. The results show that using the $tanh$ function as a preceding layer to normalize the inputs of FTA leads to good performance with all tiling bounds and significant improvement over DQN-FTA with the worst tiling bounds ($u = 0.01, 100$). On the other hand, range normalization performs poorly, and its performance worsens with all the tiling bounds. While batch normalization improves the performance of DQN-FTA with the highest tiling bound, it performs poorly when the tiling bound is very small. Overall, it is clear that $tanh$ normalization is effective regardless of the tiling bound in the LunarLander environment.

3.2.1 Cartpole

To test the effectiveness of the $tanh$ normalizing technique across environements we also performed experiments on the CartPole environment from OpenAI Gym Brockman et al. [2016]. The CartPole environment begins with a pole balanced on top of a cart, and the goal is keep the pole balanced via moving the cart left or right. The state space of the CartPole environment contains four values: cart position. cart velocity, pole angle, pole angular velocity. The action space is: push cart left, and push cart right. Since the goal is to keep the pole from falling a reward of $+1$ is recieved for each step, and the episode terminates once the pole is at an angle greater than $\pm 12°$.

We first tested to see how DQN-FTA without a target network performs with three different tiling bounds ($u = 0.01, 1, 100$) and $k = 20$. The results can be seen on the left plot in Figure 5. Unlike in LunarLander, a tiling bound of $u = 100$ performs best, while $u = 1$ performs worse. Importantly,

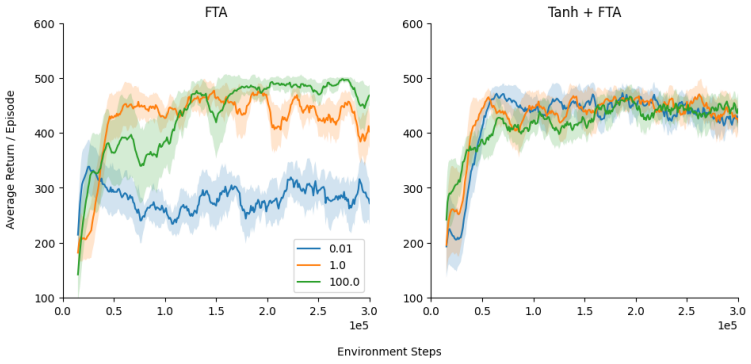

Figure 5: Evaluation learning curves plotting episodic return versus environment time steps for DQN-FTA without a target network (Left) and $tanh$ + DQN-FTA without a target network (Right).

this highlights the sensitivity of FTA to the tiling bound parameter in the CartPole environment as well as LunarLander.

We next tested to see if applying $tanh$ normalization before the FTA layer can improve the performance of $u = 0.01, 1$ to be similar to that of $u = 100$. The right plot in Figure 5 illustrates that indeed, applying $tanh$ normaliztion caused for DQN-FTA to perform similar for all the tiling bounds we tested ($u = 0.01, 1, 100$). Thus, the experimental results in the CartPole environment further confirm that using $tanh$ normalization before an FTA layer can potentially remove the need for tuning the tiling bound u.

3.3 Distribution of Returns

We observed that DQN-FTA had noticeably different returns at 500k steps between seeds, wherein some seeds performed significantly better than others. To better understand agent performance with respect to initialization stochasticity, we investigate the distribution of mean episodic returns for ReLU, FTA, and FTA-$\tanh$ over various time steps. Additionally, student's t confidence intervals and averaging over runs in evaluation learning curves assumes that mean episodic returns are uni-modal and normally distributed; Violating these assumptions would misrepresent the robustness of the agent and the average return to which it converges. A bimodal distribution of average episodic returns at the final stage of learning (ie: 500k steps) would indicate that an agent is sensitive to weight initialization or environmental stochasticity, and therefore has inconsistent learning and unreliable behavior.

Figure 6 shows the distribution of mean episodic returns for DQN, (untuned) DQN-FTA, and DQN-FTA-$tanh$. DQN and DQN-FTA has the same parameters as in 3.1, and DQN-FTA-$tanh$ has the same parameters as in 3.2. The FTA agents have the same number of bins, $k = 20$, with DQN-FTA-$tanh$ differing from DQN-FTA by a $tanh$ activation layer before the FTA layer. We can observe some general distribution characteristics and trends; For DQN and DQN-FTA, distributions of returns at the beginning of learning appear uni-modal and shift away from the mean of their distributions near the end of learning, while the distribution of returns for DQN-FTA-$tanh$ remain uni-modal and appear to converge towards the mean of its distribution. At 495k time steps, DQN and DQN-FTA distributions appear bi-modal while the DQN-FTA-$tanh$ distribution appears strongly uni-modal and more normally distributed. While 50-runs sample size per agent is relatively small hypothesis testing, our preliminary results in LunarLander show that 1) DQN and DQN-FTA distributions of returns may violate the normality and uni-modality assumptions near the end of training, an important observation on performance stability under stochasticity in the environment and weight initializations, and 2) DQN-FTA-tanh appears to maintain those assumptions throughout learning, illustrating that $tanh$ normalization can improve FTA stability.

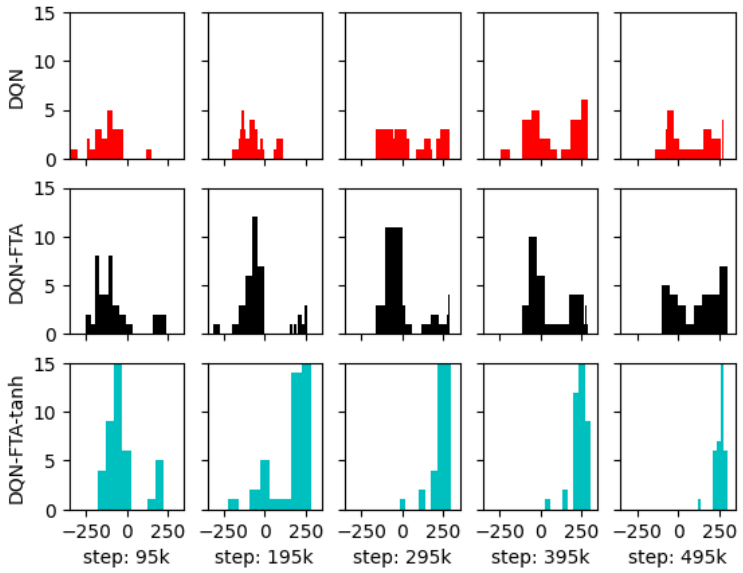

Figure 6: Histogram of mean episodic returns for **DQN(red)**, **DQN-FTA(black)**, and **DQN-FTA-$tanh$(cyan)** over time step learning progression in LunarLander. There are 50 runs per agent, with the mean episodic return values at each time step divided into 10 bins between the range [-350, 350]. All three agents do not use target networks.

4 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 relatively tight confidence intervals between runs. Further investigation into the plausible causes may be useful; however, we believe that since the same trends between networks were observed it is sufficient to claim we were able to reproduce the LunarLander experiments from [Pan et al., 2019].

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 found 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.

While Pan et al. [2019] addressed the *out-of-boundary* and *gradient vanishing* problem when $z \notin [l, u]$, we address the over-generalization problem due to incorrect bounds wherein the majority of $z \in [l, u]$ aggregate into a small range of the available bins. For example, if the inputs to FTA are $z = x^\top w \in [-1, 1]$ while the range $[l, u] = [-20, 20]$ and $\delta = \eta = 2$ such that number of bins $k = 20$ and $c = (l, l + \delta, l + 2\delta, ..., u - 2\delta, u - \delta, u) = (-20, -18, -16, ..., 16, 18, 20)$ (like the LunarLander experiment in section 3.1), then $FTA(z)$ will only bin z into two bins, between the ranges [-2,0] and [0,2]. When the $[l, u]$ bounds extend beyond the standard input to activation range,

the sparsity provided by FTA is limited due to the number of unused bins. To address the bound selection problem, we perform normalizing experiments using range normalization, $tanh$ activation, and batch normalization. We found that $tanh$ normalization is bound-agnostic and improves the performance of FTA without requiring additional tuning for the environment. In section 3.2, we show that $tanh$-normalized FTA performs the same regardless of the tiling bound $[l, u]$ given a fixed number of bin k. When comparing against the agents in the LunarLander experiments, $tanh$-normalized DQN-FTA performs as well as a tuned FTA and better than all the other agents near the end of learning. $tanh$-normalized DQN-FTA also performs better than untuned FTA in Cartpole, extending the robustness $tanh$ normalization within other environments where the input range for $z \in [l, u]$ is unknown and agents are untuned.

Lastly, we evaluate the distribution of mean episodic returns for DQN, (untuned) DQN-FTA, and DQN-FTA with tanh normalization in LunarLander. We find that near the end of learning, DQN and DQN-FTA have more bi-modal distributions whereas DQN-FTA with $tanh$ normalization is more uni-modal. This potentially demonstrates that tanh-normalized FTA improves agent stability to randomness in the initialization of network weights and environment settings compared to agents using ReLU or untuned FTA. Untuned DQN-FTAs bi-modally distributed returns could also potentially explain the poor performance in LunarLander reported by Pan et al. [2019]. Although our distribution experiments are preliminary, and that hypothesis testing with significantly more sample runs is required to more conclusively address distribution normality and modality, figure 6 illustrates how inconsistent the returns can be for an untuned FTA agent. Furthermore, our results emphasize the sensitivity of FTA tiling bounds $[l, u]$ and the importance of a bound-agnostic solution to FTA for its reliability and widespread adoption.

Comparing against the agents in the LunarLander experiments of section 3.1, $tanh$-normalized DQN-FTA performs as well as a tuned FTA and better than all the other agents near the end of learning. The experiments performed in the CartPole environment indicate a similar conclusion. Evaluating the robustness of $tanh$-normalized DQN-FTA across more environments is left as a useful future work.

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.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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. *Submitted to Psychological Review*, 1994.

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. *arXiv preprint arXiv:2003.07417*, 2020.

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. *Advances in Neural Information Processing Systems*, 32, 2019.

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.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

V. Liu, R. Kumaraswamy, L. Le, and M. White. The utility of sparse representations for control in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4384–4391, 2019.

V. Liu, A. White, H. Yao, and M. White. Towards a practical measure of interference for reinforcement learning. *arXiv preprint arXiv:2007.03807*, 2020.

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.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

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.

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.