FOURIER FEATURES IN REINFORCEMENT LEARNING WITH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In classic Reinforcement Learning (RL), encoding the inputs with a Fourier feature mapping is a standard way to facilitate generalization and add prior domain knowledge. In Deep RL, such input encodings are less common, since they could, in principle, be learned by the network and may therefore seem less beneficial. In this paper, we present experiments on Multilayer Perceptron (MLP) that indicate that even in Deep RL, Fourier features can lead to significant performance gains, in both rewards and sample efficiency. Furthermore, we observe that they increase the robustness with respect to hyperparameters, lead to smoother policies, and benefit the training process by reducing learning interference, encouraging sparsity, and increasing the expressiveness of the learned features. According to our experiments, other input preprocessings, such as random Fourier features or Polynomial features, do not give similar advantages. But a major bottleneck with conventional Fourier features is that they exponentially increase the number of features with the state dimension. We remedy this by proposing a simple, light version that only has a linear number of features, yet still maintains the benefits. Our experiments cover both shallow/deep, discrete/continuous, and on/off-policy RL settings. To the best of our knowledge, this is the first reported application of Fourier features in Deep RL.

1 INTRODUCTION

In Reinforcement Learning (RL), the performance of algorithms depends critically on data representation, i.e. the way the states of the system are represented as features. Choosing appropriate features for a task is an important way of adding prior domain knowledge, since redistributing information into states facilitates generalization. For linear function approximations, the data is usually hand-designed according to the task at hand and projected into a higher-dimensional space to facilitate linear separation (Sutton & Barto, 2018). Examples feature encodings used in classic RL for linear function approximation are Polynomial Features (Lagoudakis & Parr, 2003), Fourier Features (Konidaris et al., 2011) and Tile Coding (Albus, 1971). However, the main bottleneck of such feature encodings is that they do not scale to high-dimensional inputs as they grow exponentially with the dimension of the input.

In recent years, Deep RL has experienced dramatic growth in interest due the ability of Neural Networks (NN) to learn feature representations, allowing algorithms to learn complex tasks from raw sensory data without prior knowledge (Mnih et al., 2015; Schulman et al., 2017; Lillicrap et al., 2015; Haarnoja et al., 2018). In Deep Learning, it is common to apply min-max normalization (Bishop et al., 1995) or batch normalization (Ioffe & Szegedy, 2015). But preprocessing inputs with hand-designed features is less common since such features could, in principle, be learned by the network and thus may seem less beneficial. In recent work in Deep Learning (outside of RL), it has been shown that preprocessing inputs with Random Fourier Features (Rahimi & Recht, 2007) helps Multilayer Perceptrons (MLP) to learn high-frequency functions (Tancik et al., 2020; Wang et al., 2021) and improves the training performance for NN (Mehrkanoon & Suykens, 2018; Mitra & Kaddoum, 2021). In Deep RL, it has been observed that Tile Coding can improve performance, sample efficiency and robustness to hyperparameter variations by mitigating learning interference (Ghiassian & Huizhen Yu, 2018; Ghiassian et al., 2020; Liu et al., 2019). Fourier features have been primarily used in standard RL, where they show better performance compared to other features

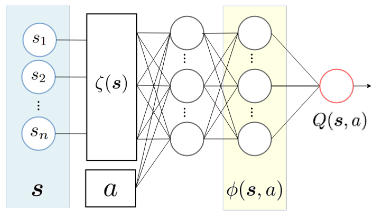


Figure 1: **Example of a 3-layers MLP fed with preprocessed inputs**. The state $\boldsymbol{s}$ is preprocessed by a feature function $\zeta(.)$ ($\zeta(\boldsymbol{s}) = \mathrm{FF}(\boldsymbol{s})$ for Fourier Features and $\zeta(\boldsymbol{s}) = \boldsymbol{s}$ for raw inputs) before being passed into the MLP with the action a. Learned features $\phi(\boldsymbol{s}, a)$ are used for the prediction of the Q-value $Q(\boldsymbol{s}, a)$.

encodings, including Tile Coding (Konidaris et al., 2011). However, to the best of our knowledge, the use of Fourier features has not yet been studied in Deep RL.

In this work, we empirically investigate the effects of preprocessing inputs with Fourier Features, as illustrated in Fig. 1, on Deep RL. Our main contributions are as follows:

- While Fourier Features are standard in classic Reinforcement Learning, we are, to our best knowledge, the first to suggest that **Fourier Features are beneficial in Reinforcement Learning with Neural Networks**.
- We study the **performance benefits of Fourier Features** across a range of tasks and cover both discrete and continuous environments. We observe significant performance gains, in both rewards and sample efficiency, and extend the range of usable hyperparameters. In our experiments, Fourier Features outperform other common input preprocessings.
- We **empirically investigate the effects of Fourier features on the learning process**. According to our experiments, Fourier features lead to smoother Neural Networks, mitigate learning interference, promote sparsity and increase the expressivity of learned features.
- We propose a **light, scalable version of Fourier Features** to avoid the exponential explosion of traditional Fourier Features, while maintaining much of their benefits.

2 Applying Fourier Features to Reinforcement Learning with Neural Networks

Fourier Features (FF) are generated by the **order-m Fourier Feature** function $\mathrm{FF} : \mathbb{R}^n \to \mathbb{R}^p$, mapping a state $\boldsymbol{s} \in \mathbb{R}^n$ into a p-dimensional feature space (Konidaris et al., 2011), with $p = (m+1)^n$. For $1 \leq i \leq p$, the feature i is given by:

$$\mathrm{FF}_i(\boldsymbol{s}) = \cos(\pi \psi(\boldsymbol{s})^\top \boldsymbol{c}^i), \tag{1}$$

where $\psi : \mathbb{R}^n \to [0,1]^n$ is a bijective normalizer and each coefficient vector $\boldsymbol{c}^i$ takes a value in $\{0, \ldots, m\}^n$ (one-to-one). The inner product $\psi(\boldsymbol{s})^\top \boldsymbol{c}^i$ determines the frequency along dimension i and creates interactions between state variables. The major bottleneck of Fourier features is that their dimension grows exponentially with the dimension n of the state space. To remedy this, we propose the following subset of Fourier features. We call **order-m Fourier Light Features** (FLF) the $n(m+1)$ Fourier Features where at most one of the elements of $\boldsymbol{c}^i$ is nonzero.

Overall Performance We apply Fourier Features (FF-NN) and Fourier Light Features (FLF-NN) on the off-policy Deep-Q Network (DQN) algorithm (Mnih et al., 2015) for the discrete action environments and on the on-policy Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) for continuous action environments and compare to the algorithms without encoding (NN). For the normalization, we compute the min/max normalization when state variables are bounded and a nonlinear normalization based on arctan otherwise. All hyperparameters are optimized for each environment for the NN approach (see details in Appendix J). For computation time reasons and a fair comparison, only learning rate and Fourier order are re-optimized for FF-NN and FLF-NN. Results of FLF-NN for deeper architectures (Appendix A) show that our conclusion remain valid.

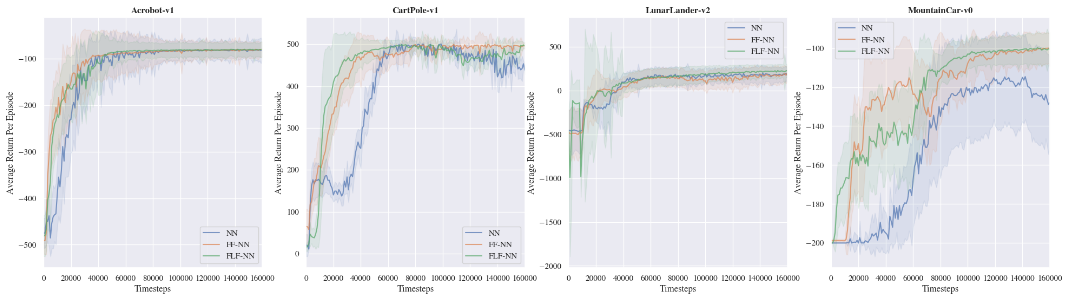


Figure 2: **The use of Fourier features improves performance and sample efficiency of DQN on discrete control tasks.** Evaluation learning curves of NN (blue), FF-NN (orange), and FLF-NN (green), reporting episodic return versus environment timesteps. Results are averaged over 30 trainings (different seeds), with shading indicating standard deviation.

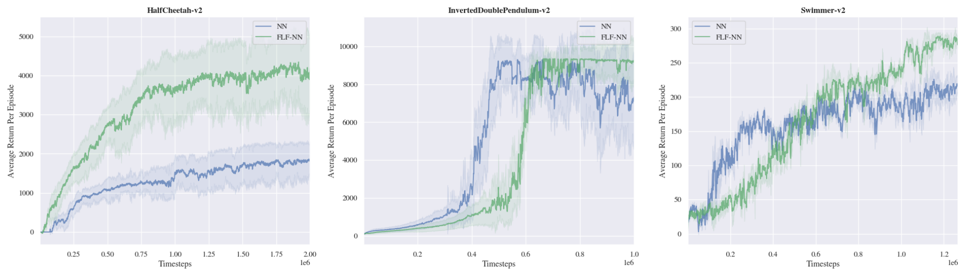


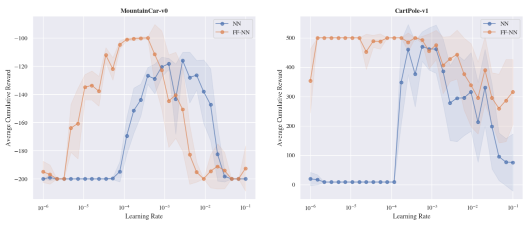
Figure 3: **The use of Fourier Light features improves the performance and sample efficiency of PPO on a continuous control task.** Evaluation learning curves of NN (blue) and FLF-NN (green), reporting episodic return versus environment timesteps. Results are averaged over 10 trainings with shading indicating the standard deviation.

Figure 2 shows the averaged returns per episode for DQN on four discrete-action environments from OpenAI Gym (Brockman et al., 2016). Figure 3 shows the averaged returns per episode of PPO on three continuous-action control tasks from Mujoco (Todorov et al., 2012). For the latter, we only test FLF because the number of traditional Fourier Features explodes due to the higher state dimension. In all tasks, FLF significantly improve upon the baseline (DQN/PPO), in both cumulative rewards and sample efficiency. Where the dimension is small enough so that we can apply FF, we obtain similar performance by FF and FLF.

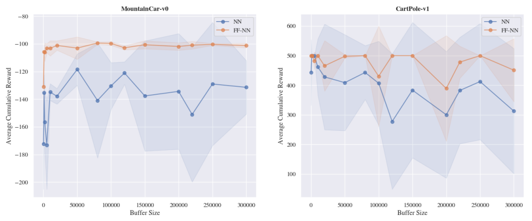
Robustness to Hyperparameter Changes RL algorithms can be very sensitive to hyperparameter changes (Henderson et al., 2018; Islam et al., 2017). The following experiments indicate that Fourier features reduce the sensitivity to hyperparameters. Figure 4a illustrates how the performance varies with the learning rate, keeping other hyperparameters constant. FF-DQN and FLF-DQN require a smaller learning rate, but perform well over a larger range compared to raw inputs. In Figures 4b and 4c we vary only the buffer size or target update frequency while keeping other hyperparameters fixed. In the cases where standard DQN shows large variations for different buffer sizes and frequencies, we observe that FF-DQN is both better and less sensitive. This indicates a more stable learning process, with potentially less interference (see also Section 3.3), and makes Fourier Features even more interesting to nonstationary and online problems. Additional results on other discrete action tasks are in Appendix C.

3 Observed Effects on Training Neural Networks

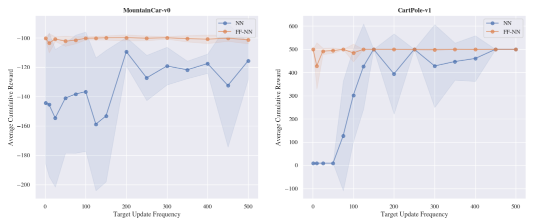
In this section, we look at different metrics in order to investigate why Fourier features help the Neural Network to learn better and faster. We study the effects of Fourier features on the sparsity and expressiveness of the NN, which in turn can reduce *catastrophic interference*. Catastrophic interference occurs when the learner ”forgets” what it has learned in the past by overwriting previous updates to better fit the learned function to recent data (McCloskey & Cohen, 1989; French,



(a) Learning Rate Variations over $n = 10$ trainings



(b) Buffer Size Variations over $n = 5$ trainings



(c) Target Update Variations over $n = 5$ trainings

Figure 4: **Fourier Features are more robust to learning rate, buffer size and target update frequency.** Cumulative reward over different hyperparameter variations, for NN (blue) and FF-NN (orange) on MountainCar-v0 and CartPole-v1. Results are averaged over n trainings and shading indicating the 95% confidence interval (CI).

1991). Such interference can significantly slow down learning and even prevent the network from converging to an optimal solution. In *sparse representations*, only few features are active (nonzero) for any given input, so that each update only impacts few weights and is less likely to interfere with other updates (Liu et al., 2019; Hernandez-Garcia & Sutton, 2019; Ghiassian et al., 2020; Pan et al., 2020). Another beneficial effect of sparsity is the promotion of locality, where similar inputs should produce similar features. It may be thus easier for the agent to make accurate predictions for an explored local region as the local dynamics are likely to be a simpler function than the global dynamics. Ghiassian et al. (2020) showed that Tile Coding improves the learning of sparse features. Enforcing sparsity can also promote *expressiveness* through the identification of key attributes by encouraging the input to be well-described by a small subset of attributes. To achieve good performance, a NN needs to extract expressive and fine-grained local features. This is particularly true when consecutive raw inputs are similar and small differences between inputs may lead to different actions. Kumar et al. (2020) and Luo et al. (2020) identified in RL an *implicit under-parameterization* of Neural Networks where under-parameterization is defined as an excessive aliasing of learned features, i.e., learned features are mapped into a much smaller subspace than the feature space that could be generated by the NN. Consequently, the neural network behaves as an under-parameterized network, generates less rich features and leads to poorer performance.

3.1 Sparsity of Learned Representations

We measure sparsity with two proxy measures: normalized overlap and instance sparsity (Liu et al., 2019; Hernandez-Garcia & Sutton, 2019; Pan et al., 2020). In the following, we take into account the number of *dead neurons*, i.e. neurons that have a zero response value for any input, and call the other m neurons *alive*. Let d the number of neurons in the penultimate layer. The *normalized activation overlap*, as proposed by Hernandez-Garcia & Sutton (2019), is

$$\text{overlap}(\phi(\boldsymbol{s}_1, a_1), \phi(\boldsymbol{s}_2, a_2)) = \frac{1}{m} \sum_{i=1}^{d} \mathbf{1}_{(\phi_\mathrm{i}(\mathrm{s}_1, \mathrm{a}_1) > 0) \wedge (\phi_\mathrm{i}(\mathrm{s}_2, \mathrm{a}_2) > 0)}. \tag{2}$$

When the normalized overlap between two representations is zero, there is no interference between their corresponding inputs. The normalization avoids misleadingly low scores in cases where only few neurons are alive. The *instance sparsity* is the percentage of active units in the feature vector $\phi(\boldsymbol{s}, a)$ for a given input value $(\boldsymbol{s}, a)$ (Liu et al., 2019). We estimate these sparsity measures using

Table 1: **Fourier features and Fourier Light features promote sparsity on discrete control tasks**. Sparsity scores with percentage of dead neurons, normalized activation overlap and instance sparsity obtained for DQN fed with raw inputs (NN), Fourier features (FF-NN) and Fourier Light features (FLF-NN), averaged across environment timesteps. Averages are taken across all timesteps and margins of error of the 95% confidence interval (CI) are computed over 30 trainings. Lower sparsity scores are better and better scores are in **bold**.

Architecture		**MountainCar-v0**	**Acrobot-v1**	**CartPole-v1**	**Catcher-v1**	**LunarLander-v2**
NN	Dead Neurons	0.47 ± 0.09	**0.0**	0.07 ± 0.02	0.02	0.0
	Normalized Overlap	0.72 ± 0.08	0.49 ± 0.04	0.63 ± 0.04	0.34 ± 0.01	0.30 ± 0.01
	Instance Sparsity	0.78 ± 0.07	0.64 ± 0.02	0.66 ± 0.03	0.48 ± 0.01	0.46 ± 0.02
FF-NN	Dead Neurons	**0.0**	0.01	**0.0**	**0.0**	**0.0**
	Normalized Overlap	$\mathbf{0.37 \pm 0.06}$	$\mathbf{0.05 \pm 0.02}$	$\mathbf{0.52 \pm 0.02}$	$\mathbf{0.20 \pm 0.02}$	$\mathbf{0.23 \pm 0.03}$
	Instance Sparsity	$\mathbf{0.57 \pm 0.05}$	$\mathbf{0.13 \pm 0.04}$	$\mathbf{0.60 \pm 0.02}$	$\mathbf{0.40 \pm 0.03}$	$\mathbf{0.40 \pm 0.02}$
FLF-NN	Dead Neurons	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	Normalized Overlap	0.43 ± 0.10	0.16 ± 0.02	0.79 ± 0.07	0.28 ± 0.01	0.39 ± 0.04
	Instance Sparsity	0.62 ± 0.08	0.30 ± 0.03	0.85 ± 0.06	0.45 ± 0.02	0.55 ± 0.04

a set $\mathcal{D}$ of state-action pairs drawn i.i.d from rollouts obtained with sub-optimal pre-trained policies and random policies; for details see Appendix D. Our results are summarized in Table 1; the corresponding curves as a function of environment steps can be found in Appendix E. In all tasks, FF results in lower (and thus better) normalized overlap and instance sparsity. There are no dead neurons in FF/FLF, suggesting a better use of the Neural Network capacity. However, FLF increases the sparsity in one instance (CartPole-v1), even though the learning performance of FLF-NN is better than NN on all instances. Hence, sparsity does not seem to be the only beneficial effect of FF/FLF.

3.2 Expressiveness of Learned Representations

To measure expressiveness, we compute the *effective rank* srank_δ of the learned feature matrix $\mathbf{\Phi}$ built on the set $\mathcal{D}$, normalized by the number of neurons in the penultimate layer (Kumar et al., 2020). This estimates the proportion of the sum of the k highest singular values $\sigma_1(\mathbf{\Phi}) \geq \ldots \geq \sigma_{\min(b,d)}(\mathbf{\Phi}) \geq 0$ of $\mathbf{\Phi}$ that capture $1-\delta$ (usually $\delta = 0.01$) of the sum of all singular values:

$$\text{srank}_\delta(\mathbf{\Phi}) = \frac{1}{d} \min \left\{ k : \frac{\sum_{i=1}^{k} \sigma_i(\mathbf{\Phi})}{\sum_{i=1}^{\min(b,d)} \sigma_i(\mathbf{\Phi})} \geq 1 - \delta \right\}, \tag{3}$$

where b is the number of state-action pairs used to build $\mathbf{\Phi}$ and d is the number of neurons in the penultimate layer of the NN. Intuitively, this quantity represents the number of ”effective” unique components of the feature matrix $\mathbf{\Phi}$ that form the basis for linearly approximating the targets. When the network aliases inputs by mapping them to a smaller subspace, $\mathbf{\Phi}$ has only a few active singular directions and $\text{srank}_\delta(\mathbf{\Phi})$ takes thus a small value.

Figure 5 shows the normalized effective rank over the environment timesteps of training. The learned features are more expressive for FF/FLF in all instances, which may induce a better use of the network capacity and explain better performance. These results are consistent with the absence of dead neurons for FF/FLF reported in Table 1. Features learned with FLF-NN are more expressive than those with FF-NN in most instances. In Deep RL, because targets are estimated with boostrapping, a *decrease in the effective rank* of learned features can lead to a sequence of NNs with potentially decreasing expressivity and results in degenerate behaviors, generalization problems and drops in performance (Kumar et al., 2020; Luo et al., 2020). In Figure 5, the curves with FF/FLF and raw data have a similar general trend, but in most instances, the decrease in the effective rank is less pronounced with FF/FLF. This suggests a more stable learning process with less catastrophic interference for FF-NN and FLF-NN. Additional results on discrete tasks can be found in Appendix F.

3.3 Learning Interference

To investigate interference, we estimate the *stiffness* of the gradient alignment (Fort et al., 2020)

$$\rho(\boldsymbol{s}, a, \boldsymbol{s}', a') = \cos(\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{s}, a), \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{s}', a')), \tag{4}$$

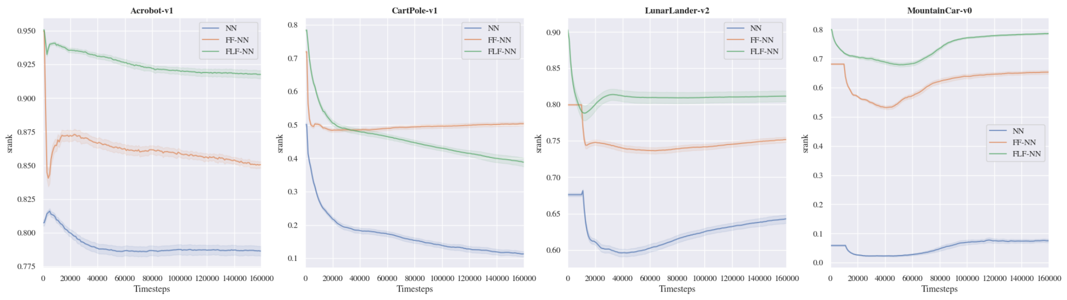


Figure 5: **The use of Fourier features, and Fourier Light features enhance the expressiveness of the learned features on discrete control tasks.** Learning curves of the normalized effective rank $\mathrm{srank}_\delta(\boldsymbol{\Phi})$ for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.

with parameters $\boldsymbol{W}$, loss L, and cosine similarity $\cos(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T\boldsymbol{v}/||\boldsymbol{u}||||\boldsymbol{v}||$. More context and related work are provided in Appendix G. A stiffness $\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')$ close to zero means that an update made to $(\boldsymbol{s}, a)$ does not affect the prediction in $(\boldsymbol{s}', a')$ whereas a negative value determines an interference. Based on stiffness, we consider four proxy measures for gradient interference evaluated on the DQN experience replay buffer $\mathcal{B}$:

- *Average of Stiffness* (AS): $\mathbb{E}_{(\boldsymbol{s},a),(\boldsymbol{s}',a')\sim\mathcal{B}}[\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')]$,
- *Average of Interference* (AI): $\mathbb{E}_{(\boldsymbol{s},a),(\boldsymbol{s}',a')\sim\mathcal{B}}[\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')|\rho(\boldsymbol{s}, a, \boldsymbol{s}', a') < 0]$ which only considers (negatively) interfering samples and determines the average of interference,
- *Interference Risk* (IR): $\mathrm{CVaR}_{0.9}(\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')) = \mathbb{E}[\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')|\rho(\boldsymbol{s}, a, \boldsymbol{s}', a') \leq \mathrm{VaR}_{0.9}(\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')) \wedge \rho(\boldsymbol{s}, a, \boldsymbol{s}', a') \leq 0]$ which is the conditional value at risk of interference where $\mathrm{VaR}_{0.9}(\rho(\boldsymbol{s}, a, \boldsymbol{s}', a'))$ is the 0.9-quantile of the distribution of $\rho(\boldsymbol{s}, a, \boldsymbol{s}', a')$,
- *Percentage of Interfering sample pairs* (PercI) within a minibatch sampled from $\mathcal{B}$.

Our results averaged across all timesteps are reported in Table 2 and curves showing the evolution of interference during the training can be found in Appendix G. In all cases, AS shows that an update with a state-action pair has less impact on other NN predictions with FF/FLF compared to raw inputs. This is confirmed by higher (better) AI and IR scores. Only for PercI we see worse scores with FF/FLF. However, similar contradictory results for methods improving sparsity were reported by Pan et al. (2020), which suggests a more complex relationship between PercI and actual learning performance. Our observations indicate that Fourier features help to generalize appropriately without overgeneralizing and leads to a more stable training and better performance. Interestingly, FLF-NN seem to have even less interference than FF-NN, while we observed sparser representations for FF-NN in Section 3.1. Such results suggest that even if sparsity mitigates catastrophic interference, FLF may have other beneficial effects that reduce catastrophic interference.

3.4 Smoothness of Neural Network

In Neural Networks, larger weights are associated with poorer relative performance on new data (Neyshabur et al., 2015; Bartlett et al., 2017; Neyshabur et al., 2017). Similarly, in Reinforcement Learning, regularization approaches that enforce small weight norms, such as weight decay, tend to produce better results (Farebrother et al., 2018; Liu et al., 2020; Cobbe et al., 2019). A common approach is to normalize weights to ensure that the learned layers are 1-Lipschitz, thereby improving the smoothness of the model, improving convergence (Salimans & Kingma, 2016; Gogianu et al., 2021), and reducing the generalization gap (Rosca et al., 2020; Gouk et al., 2021; Wang et al., 2019).

We investigate the smoothness by estimating bounds on the Lipschitz constant, taking the lower bounds from (Rosca et al., 2020) and the upper bound from (Gouk et al., 2021). See Appendix H for more context and details. In three out of the four tasks shown in Figure 6, FF features clearly improve the Lipschitz smoothness, while FLF do not have a clear effect. Additional metrics, based on the l_1, l_2, and l_∞ norm of different layers, indicate that FLF can lie between FF and raw data, sometimes even surpassing FF; see Appendix H.

Table 2: **Fourier features and Fourier Light features mitigate learning interference on discrete control tasks**. Interference measures with Average of Stiffness (AS), Average of Interference (AI), Interference Risk (IR), and Percentage of Interference within a batch (PercI) averaged across all timesteps for DQN fed with raw inputs (NN), Fourier features (FF-NN) and Fourier Light features (FLF-NN) on discrete control tasks. The symbol ↓ (↑) indicates that a lower (higher) score is better. Better interference measures are in **bold.**

Architecture			**MountainCar-v0**	**Acrobot-v1**	**CartPole-v1**	**Catcher-v1**	**LunarLander-v2**
NN	AS	↓	0.24	0.09	0.22	0.07	0.06
	AI	↑	−0.83	−0.60	−0.92	−0.63	−0.56
	IR	↑	−0.91	−0.92	−0.99	−0.94	−0.94
	PercI	↓	0.38	**0.43**	**0.39**	**0.45**	0.46
FF-NN	AS	↓	0.10	**0.03**	**0.05**	0.04	0.06
	AI	↑	**−0.47**	**−0.37**	**−0.73**	**−0.40**	**−0.49**
	IR	↑	**−0.87**	**−0.80**	−0.98	−0.84	**−0.92**
	PercI	↓	0.44	0.48	0.47	0.47	**0.46**
FLF-NN	AS	↓	**0.05**	**0.04**	**0.05**	**0.04**	**0.04**
	AI	↑	−0.54	**−0.38**	−0.86	−0.54	−0.67
	IR	↑	**−0.87**	**−0.79**	−0.98	**−0.82**	−0.94
	PercI	↓	0.47	0.47	0.47	0.47	0.48

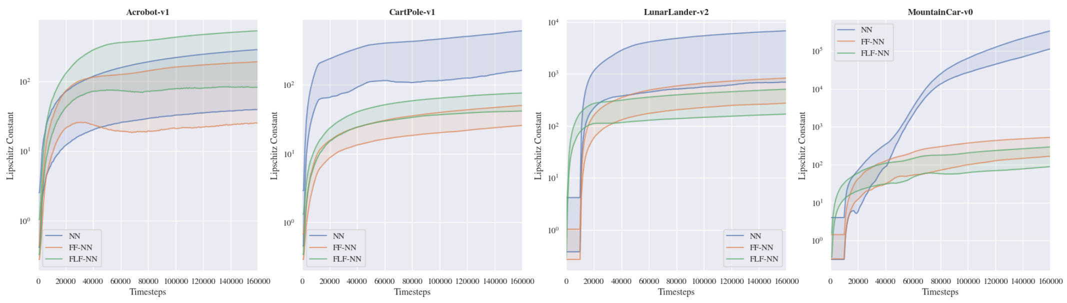


Figure 6: **Preprocessing inputs with Fourier features or Fourier Light features improve the smoothness of the Neural Network.** Lower and upper bounds on the Lipschitz constant of NN over training timesteps, for NN fed with raw inputs (blue), FF (orange), and FLF (green). Bounds are averaged over 30 trainings. A lower score is better.

4 Comparison with Other Input Preprocessings

We compare Fourier features with three standard input preprocessings: Polynomial Features (PF-NN), Random Fourier Features (RFF-NN), and Tile Coding (TC-NN). For more context, definitions, and experimental setup, see Appendix I. In the experiments shown in Fig. 7, none of the other input preprocessings achieve the performance of FF-NN/FLF-NN, even though we tuned their hyper-parameters through an extensive search. In the following, we evaluate the effects on the Neural Network using the metrics from Section. 3.

Sparsity measures, as reported in Table 3, suggest that standard input preprocessings degrade sparsity. Even Tile Coding, reported to promote sparsity in (Ghiassian et al., 2020), produces less sparse representations than standard DQN. Tile Coding, just as FF/FLF, does not have dead neurons, while PF-NN and RFF-NN increase the number of dead neurons.

Expressiveness is indicated by the learning curves of effective rank in Figure 8. PF-NN and RFF-NN generate poorer features than NN fed with raw inputs. As expected given the absence of dead neurons, TC-NN produces richer features than NN and is on par with FF/FLF-NN.

Interference scores, shown in Table 4, indicate that PF-NN, RFF-NN, and TC-NN highly interfere during the training with poor Average of Interference (AI) and Interference Risk (IR) scores. These results are consistent with the poor sparsity scores.

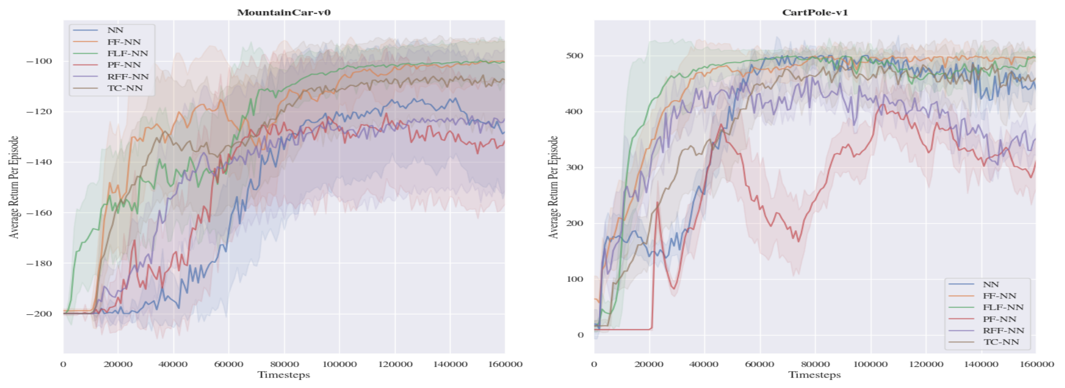


Figure 7: **Fourier Features/Fourier Light features outperform other standard input preprocessings on discrete control tasks.** Evaluation learning curves of NN (blue), FF-NN (orange), FLF-NN (green), PF-NN (red), RFF-NN (purple) and TC-NN (brown) reporting episodic return versus environment timesteps. Results are averaged over 30 trainings with shading indicating the standard deviation.

Table 3: **Input preprocessings do not necessarily promote sparsity in discrete control tasks**. Sparsity scores with dead neurons in % (DN), normalized overlap (NO) and instance sparsity (IS) obtained for DQN fed with raw inputs (NN), Fourier features (FF-NN), Fourier Light features (FLF-NN), Normalized inputs (NI-NN), Polynomial features (PF-NN), Random Fourier features (RFF-NN) and Tile Coding (TC-NN) on discrete control tasks averaged across all timesteps. Averages and margins of error of the 95% CI are over 30 trainings. Lower scores better.

Task		NN	FF-NN	FLF-NN	PF-NN	RFF-NN	TC-NN
MountainCar-v0	DN	0.47 ± 0.09	**0.0**	**0.0**	0.66 ± 0.08	0.48 ± 0.04	**0.0**
	NO	0.72 ± 0.08	$\mathbf{0.37 \pm 0.06}$	0.43 ± 0.10	0.80 ± 0.08	0.87 ± 0.06	0.77 ± 0.13
	IS	0.78 ± 0.07	$\mathbf{0.57 \pm 0.05}$	0.62 ± 0.08	0.84 ± 0.08	0.90 ± 0.05	0.86 ± 0.09
CartPole-v1	DN	0.07 ± 0.02	0.01 ± 0.00	**0.0**	0.23 ± 0.02	0.88 ± 0.07	**0.0**
	NO	0.63 ± 0.04	$\mathbf{0.52 \pm 0.02}$	0.79 ± 0.07	0.73 ± 0.07	0.58 ± 0.03	0.66 ± 0.06
	IS	0.66 ± 0.03	$\mathbf{0.60 \pm 0.02}$	0.85 ± 0.06	0.75 ± 0.05	$\mathbf{0.61 \pm 0.03}$	0.70 ± 0.04

Smoothness, as measured by the Lipschitz bounds shown in Figure 9, is improved by all input preprocessings, with TC/FF/FLF giving the best performance, followed by PF/RFF. In the shown instance, smoother networks correlate with better learning performance.

5 Conclusion

We studied the effect of Fourier feature encoding on Deep RL for a set of discrete and continuous control problems. We found that Fourier features provide a systematic increase in final performance, sample efficiency, learning stability, and robustness to hyperparameters. A detailed empirical analysis of the reason for this behavior showed that Fourier features improve the sparsity, expressiveness and smoothness of the NN, and reduce catastrophic interference during learning. Additionally, a light version of Fourier features with only a linear number of features compared to the input size leads to similar benefits.

We have just begun to understand how Fourier features and more broadly input preprocessing may improve neural network training. Our work suggests thus several promising future research directions. The first is to investigate whether Fourier features behave as an implicit regularizer and decrease the generalization gap. Second, we plan to further study the correlations between performance, sparsity, expressiveness, learning interference, and smoothness of neural networks. Our experiments show that preprocessing inputs may improve performance of Neural Networks in Deep RL, and there is a wide range of research directions that can be pursued. The code for reproducing all experiments will be made publicly available on github at the end of the double-blind review.

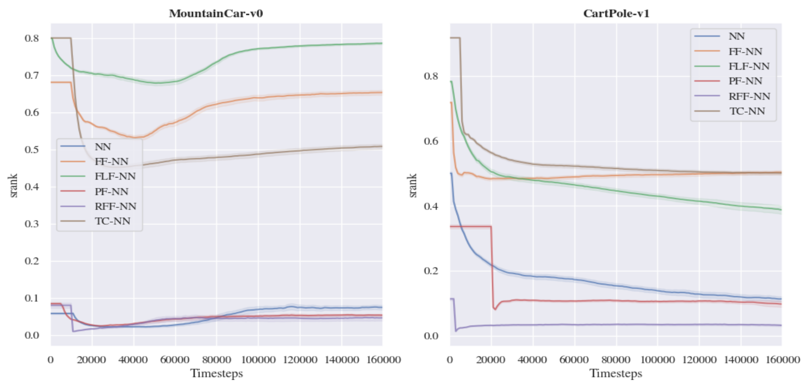


Figure 8: Learning curves of the normalized effective rank $\mathrm{srank}_\delta(\mathbf{\Phi})$ for NN fed with raw inputs (blue), Fourier features (orange), Fourier Light features (green), Polynomial features (red), Random Fourier Features (purple) and Tile Coding features (brown) on discrete control tasks. Results are averaged over 30 training with the shade that indicates the 95% CI.

Table 4: Interference measures with Average of Stiffness (AS), Average of Interference (AI), Interference Risk (IR), Percentage of Interference within a batch (PercI) and Average Q-Interference (AQI) averaged across all timesteps for standard input preprocessings on discrete control tasks where $\downarrow$ and $\uparrow$ mean lower and higher the score is better, respectively.

Tasks			NN	FF-NN	FLF-NN	PF-NN	RFF-NN	TC-NN
MountainCar-v0	AS	$\downarrow$	0.24	0.1	**0.05**	0.21	0.14	0.1
	AI	$\uparrow$	−0.83	**−0.47**	−0.54	−0.81	−0.88	−0.86
	IR	$\uparrow$	−0.91	**−0.87**	**−0.87**	−0.95	−0.93	−0.93
	PercI	$\downarrow$	**0.38**	0.44	0.47	0.39	0.43	0.44
CartPole-v1	AS	$\downarrow$	0.22	**0.05**	**0.05**	0.13	0.49	**0.07**
	AI	$\uparrow$	−0.92	**−0.73**	−0.86	−0.84	−0.97	−0.95
	IR	$\uparrow$	−0.99	−0.98	−0.98	**−0.87**	−0.99	−0.97
	PercI	$\downarrow$	0.39	0.47	0.47	0.43	**0.25**	0.46

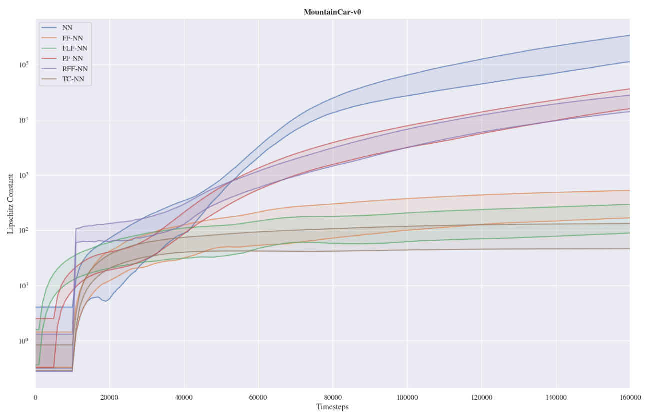


Figure 9: Lower and upper bounds on the Lipschitz constant of NN over training timesteps, of NN (blue), FF-NN (orange), FLF-NN (green), PF-NN (red), RFF-NN (purple), and TC-NN (brown). Bounds are averaged over 30 trainings with shading indicating the 95% CI

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

James S Albus. A theory of cerebellar function. *Mathematical biosciences*, 10(1-2):25–61, 1971.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30:6240–6249, 2017.

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pp. 767–777. PMLR, 2020.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Stanislav Fort, Paweł Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks, 2020.

R. French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 1991.

Sina Ghiassian and Richard S. Sutton Huizhen Yu, Banafsheh Rafiee. Two geometric input transformation methods for fast online reinforcement learning with neural nets. *arXiv*, 2018.

Sina Ghiassian, Banafsheh Rafiee, Yat Long Lo, and Adam White. Improving performance in reinforcement learning by breaking generalization in neural networks. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, pp. 438–446, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450375184.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Florin Gogianu, Tudor Berariu, Mihaela Rosca, Claudia Clopath, Lucian Busoniu, and Razvan Pascanu. Spectral normalisation for deep reinforcement learning: an optimisation perspective. *arXiv preprint arXiv:2105.05246*, 2021.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL `http://arxiv.org/abs/1812.05905`.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of AAAI Conference on Artificial Intelligence, (AAAI-18)*. AAAI Press, 2018.

J. Fernando Hernandez-Garcia and Richard S. Sutton. Learning sparse representations incrementally in deep reinforcement learning. *arXiv*, 2019.

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

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8:293–321, 1992. doi: 10.1007/BF00992699. URL `https://doi.org/10.1007/BF00992699`.

Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4384–4391, Jul. 2019. doi: 10.1609/aaai.v33i01.33014384. URL `https://ojs.aaai.org/index.php/AAAI/article/view/4349`.

Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization-an empirical study on continuous control. In *International Conference on Learning Representations*, 2020.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6470–6479, 2017.

Xufang Luo, Qi Meng, Di He, Wei Chen, and Yunhong Wang. I4r: Promoting deep reinforcement learning by the indicator for expressive representations. In *IJCAI*, pp. 2669–2675, 2020.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Siamak Mehrkanoon and Johan AK Suykens. Deep hybrid neural-kernel networks using random fourier features. *Neurocomputing*, 298:46–54, 2018.

Rangeet Mitra and Georges Kaddoum. Random fourier feature based deep learning for wireless communications. *arXiv preprint arXiv:2101.05254*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

Behnam Neyshabur. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pp. 1376–1401. PMLR, 2015.

Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. Exploring generalization in deep learning. *Advances in Neural Information Processing Systems*, 30:5947–5956, 2017.

Yangchen Pan, Kirby Banman, and Martha White. Fuzzy tiling activations: A simple approach to learning sparse representations online. In *International Conference on Learning Representations*, 2020.

Andy Patterson. Pyfixedreps. `https://github.com/andnp/PyFixedReps`, 2020.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Antonin Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 1177–1184, 2007.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6553–6564, 2017.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

Mihaela Rosca, Theophane Weber, Arthur Gretton, and Shakir Mohamed. A case for new neural network smoothness constraints. In Jessica Zosa Forde, Francisco Ruiz, Melanie F. Pradier, and Aaron Schein (eds.), *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pp. 21–32. PMLR, 12 Dec 2020. URL `https://proceedings.mlr.press/v137/rosca20a.html`.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29:901–909, 2016.

Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 3839–3848, 2018.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Huan Wang, Stephan Zheng, Caiming Xiong, and Richard Socher. On the generalization gap in reparameterizable reinforcement learning. In *International Conference on Machine Learning*, pp. 6648–6658. PMLR, 2019.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

A Fourier Features and Fourier Light Features in Deeper Architectures

In this section, we present results obtained with hyperparameters in the original code base for deeper architectures (Raffin, 2020) to confirm our results found in Section 2 that Fourier Light features with NN lead to better performance, in both reward and sample efficiency. Hyperparameter settings can be found in Appendix J.

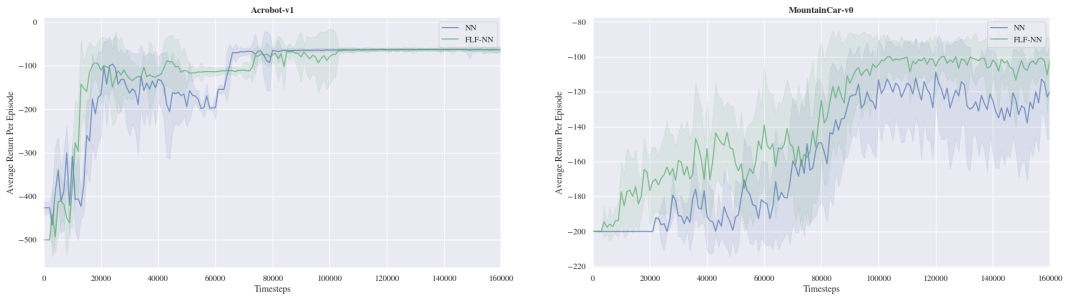


Figure 10: Evaluation learning curves of NN (blue) and FLF-NN (green) with DQN on discrete control tasks, reporting episodic return versus environment timesteps. Results are averaged over 10 trainings (different seeds), with shading indicating standard deviation.

B Learning Curve of DQN on Catcher-v1

Learning curves for the Catcher-v1 task (Tasfi, 2016) investigated in Section 3 are reported in Figure 11. Hyperparameter settings can be found in Appendix J.

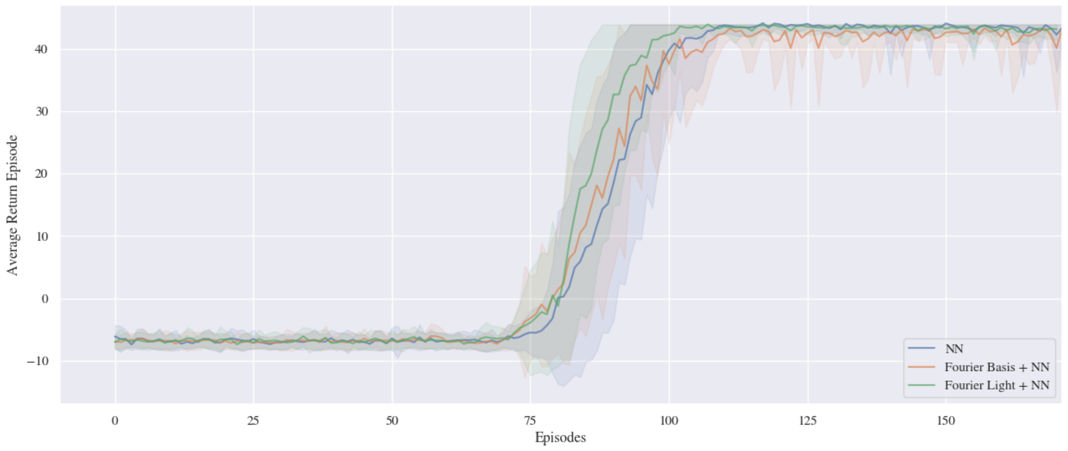
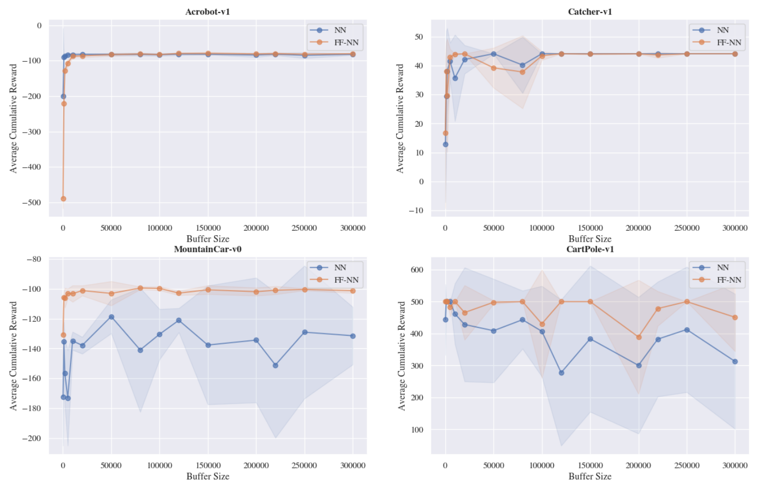


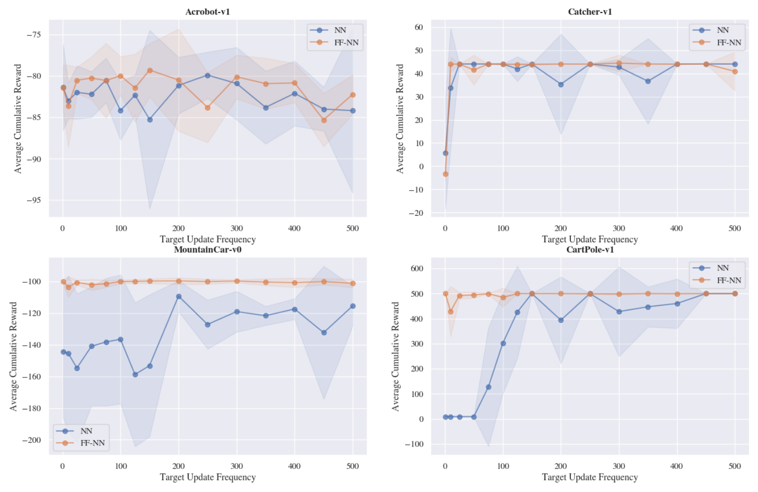
Figure 11: Learning curves of NN (blue) and FLF-NN (green) with DQN on Catcher-v1, reporting episodic return versus environment timesteps. Results are averaged over 30 trainings (different seeds), with shading indicating standard deviation.

C Further results on hyperparameter variations for DQN

In RL, the data distribution is constantly shifting because of both the changing policy and shifting training targets, e.g with bootstrapping. In addition, in most RL problems data are correlated and hurt training of NNs. Experience replay buffers (Lin, 1992; Mnih et al., 2015) and target networks (Mnih et al., 2015) were introduced in RL to mitigate these interference problems and have become critical in training of many deep RL algorithms including DQN. However, it is at the cost of higher computational and memory cost and a slower offline learning (Plappert et al., 2018). Previous works also highlighted difficulties to properly tune the buffer size where an either too small or too big buffer can have a negative effect in performance (Zhang & Sutton, 2017). In Figures 12a and 12b we vary only the buffer size or target update frequency while keeping other hyperparameters fixed over 5 trainings. In the cases where standard DQN shows large variations for different buffer sizes and frequencies, we observe that FF-DQN is both better and less sensitive (except for the Acrobot-v1 task). Such results suggest a more stable training.



(a) Buffer Size



(b) Target Update Frequency

Figure 12: **Fourier Features are more robust to buffer size and target update frequency.** Cumulative reward over different buffer size values (Figure 12a) and target update frequencies (Figure 12b) for NN (blue) and FF-NN (orange) on MountainCar-v0 and CartPole-v1. Results are averaged over 5 trainings with shading indicating the 95% CI.

D Experimental Setup for Observing the Effects on Training Neural Networks

We restrict our analysis to DQN with the same hyperparameter setting used in Sect. 2. We estimate both sparsity and the expressiveness measures during the training over the same learned feature matrix $\mathbf{\Phi}(\mathcal{D}) = (\phi(\boldsymbol{s}_1, a_1), \ldots, \phi(\boldsymbol{s}_b, a_b))^T$ every $1,000$ environment timesteps, where $\mathcal{D} := \{(\boldsymbol{s}_1, a_1), \ldots, (\boldsymbol{s}_b, a_b)\}$ is a datatset of $b = 3,000$ state-action pairs. State-action pairs in $\mathcal{D}$ are drawn i.i.d from rollouts obtained with sub-optimal pre-trained policies and random policies. This construction of $\mathcal{D}$ aims to cover state-action pairs likely to be used during the DQN learning. Consequently, our estimations of the percentage of dead neurons are less restrictive than the true percentage of dead neurons. Nevertheless, we believe that measuring sparsity scores over $\mathcal{D}$ makes more sense since it removes neurons only active in parts of the state space that are less likely to be visited by the agent.

E Sparsity Learning Curves for DQN on Discrete Control Tasks

Normalized overlap as function of environment steps corresponding are reported in Figure 13. Average of these score across timesteps can be found in Table 1. We estimate sparsity scores every $1,000$ timesteps and average them over 30 trainings.

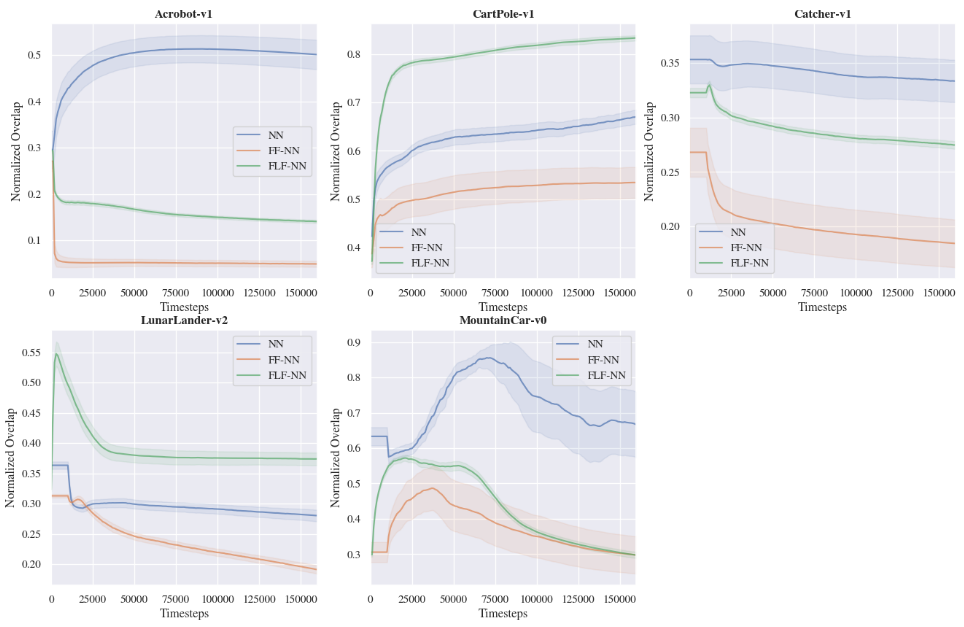


Figure 13: Learning curves of the normalized overlap for NN (blue), FF-NN (orange), and FLF-NN (green) with DQN on discrete control tasks, reporting normalized overlap versus environment timesteps. Results are averaged over 30 trainings (different seeds), with shading indicating the 95% CI.

F Normalized Effective Rank Curves for DQN on Discrete Control Tasks

The normalized effective rank $\mathrm{srank}_\delta(\mathbf{\Phi})$ (Section 3.2) as a function of environment steps is reported for DQN on discrete control tasks in Figure 14. We estimate normalized effective rank every $1,000$ timesteps and average them over 30 trainings.

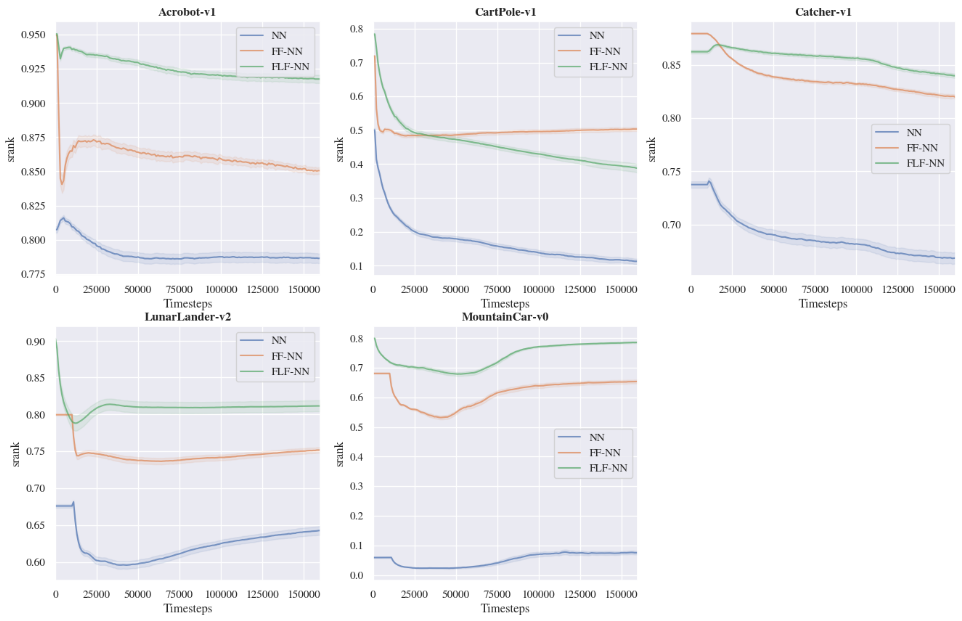


Figure 14: Learning curves of the normalized effective rank $\mathrm{srank}_\delta(\mathbf{\Phi})$ for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.

G Interference Learning Curves for DQN on Discrete Control Tasks

Connection between Gradient Alignment and Stiffness. The learning interference at a time t with parameters $\boldsymbol{W}$ and a loss L is defined as (Lopez-Paz & Ranzato, 2017; Riemer et al., 2018)

$$L(\boldsymbol{W}_{t+1}; \boldsymbol{s}, a) - L(\boldsymbol{W}_t; \boldsymbol{s}, a).$$

Assuming a small learning rate η and using the Taylor series expansion on the loss L, we have for an update made to $\boldsymbol{W}$ with $(\boldsymbol{s}_t, a_t)$:

$$\begin{aligned} L(\boldsymbol{W}_{t+1}; \boldsymbol{s}, a) - L(\boldsymbol{W}_t; \boldsymbol{s}, a) &\approx \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t; \boldsymbol{s}_t, a_t)^T (\boldsymbol{W}_{t+1} - \boldsymbol{W}_t) \\ &= -\eta \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t; \boldsymbol{s}_t, a_t)^T \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t; s, a) \end{aligned} \tag{5}$$

where the quantity $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t; \boldsymbol{s}_t, a_t)^T \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t; \boldsymbol{s}, a)$ is the *gradient alignment* (Bengio et al., 2020; Lopez-Paz & Ranzato, 2017; Riemer et al., 2018; Schaul et al., 2019). The positiveness or negativeness of this quantity determines whether the update is constructive (i.e positive generalization) or destructive (i.e. interference) on $(\boldsymbol{s}, a)$. The stiffness measure used in Section 3.3 is based on gradient alignment (Fort et al., 2020) by normalizing loss gradients.

Evolution of Interference During the Training. We estimate interference every $1,000$ timesteps and average them over 30 trainings. For each point in the learning curves, we randomly draw 64 samples from the experience replay buffer to estimate the interference. All interference measures are defined in Section 3.3.

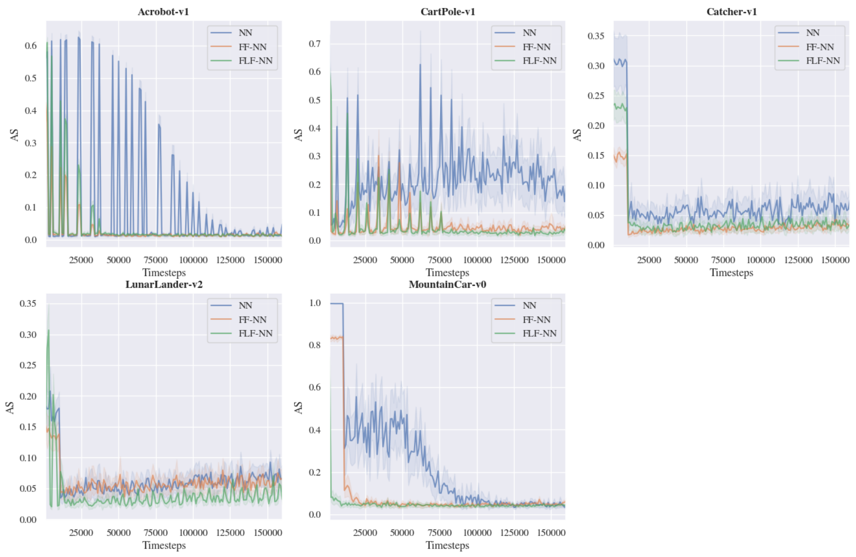


Figure 15: Learning curves of the **Average Stiffness** (AS) for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green). Results are averaged over 30 trainings with shading indicating the 95% CI.

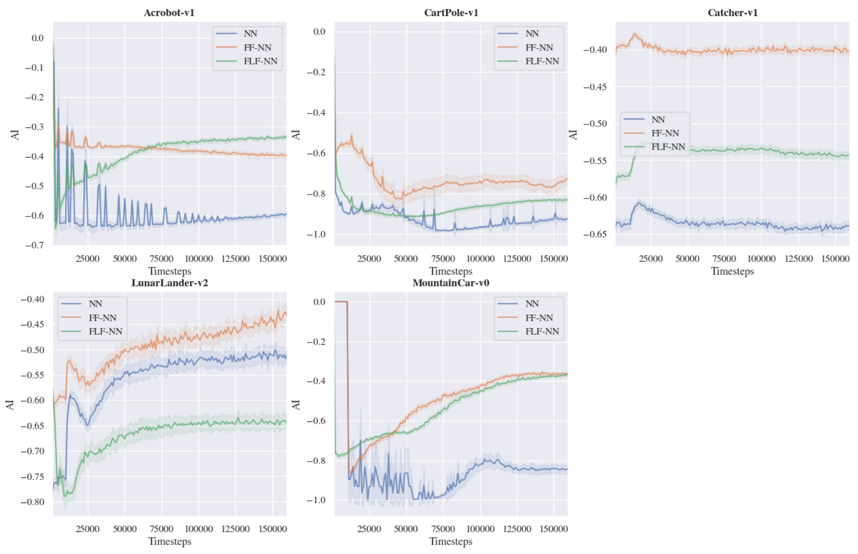


Figure 16: Learning curves of the **Average Interference** (AI) for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green). Results are averaged over 30 trainings with shading indicating the 95% CI.

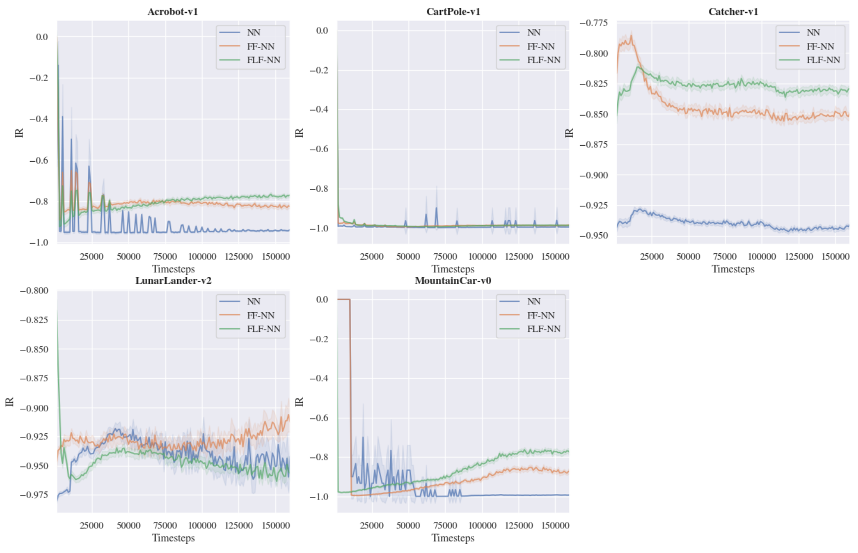


Figure 17: Learning curves of the **Interference Risk** (IR) for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green). Results are averaged over 30 trainings with shading indicating the 95% CI.

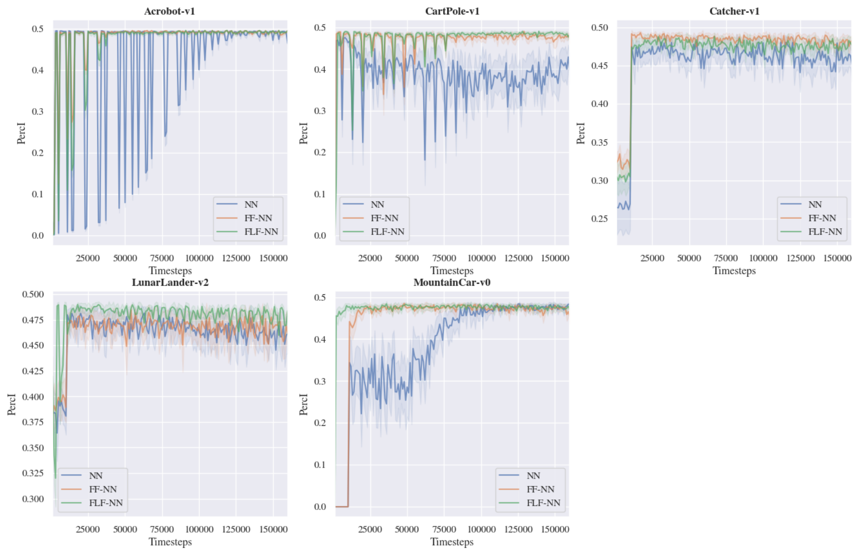


Figure 18: Learning curves of the **Percentage of Interference** (PercI) for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green). results are averaged over 30 trainings with shading indicating the 95% CI.

H SMOOTHNESS OF NEURAL NETWORKS FOR DQN ON DISCRETE CONTROL TASKS

Smoothness of the Neural Network. The exact computation of the Lipschitz constant for a NN is NP-hard (Scaman & Virmaux, 2018), but lower bounds and upper bounds can be estimated. The lower bound is obtained by computing for each state-action pairs $(\boldsymbol{s}, a)$ in a dataset $300,000$ state-action pairs the norm of the gradient of the Q-value with respect to the state-action pairs and we report the largest norm encountered (Rosca et al., 2020). Whereas to establish the upper bound, we compute the Lipschitz constants of each layer in isolation and multiply them (Gouk et al., 2021). Under the l_2 and l_1 norm, the upper bound of the Lipschitz constant of an MLP is given by the spectral norm and the maximum absolute column sum norm measure of the weight matrix (Neyshabur, 2017; Gouk et al., 2021). We measure these bounds every $1,000$ timesteps of the training and we average results over 30 trainings.

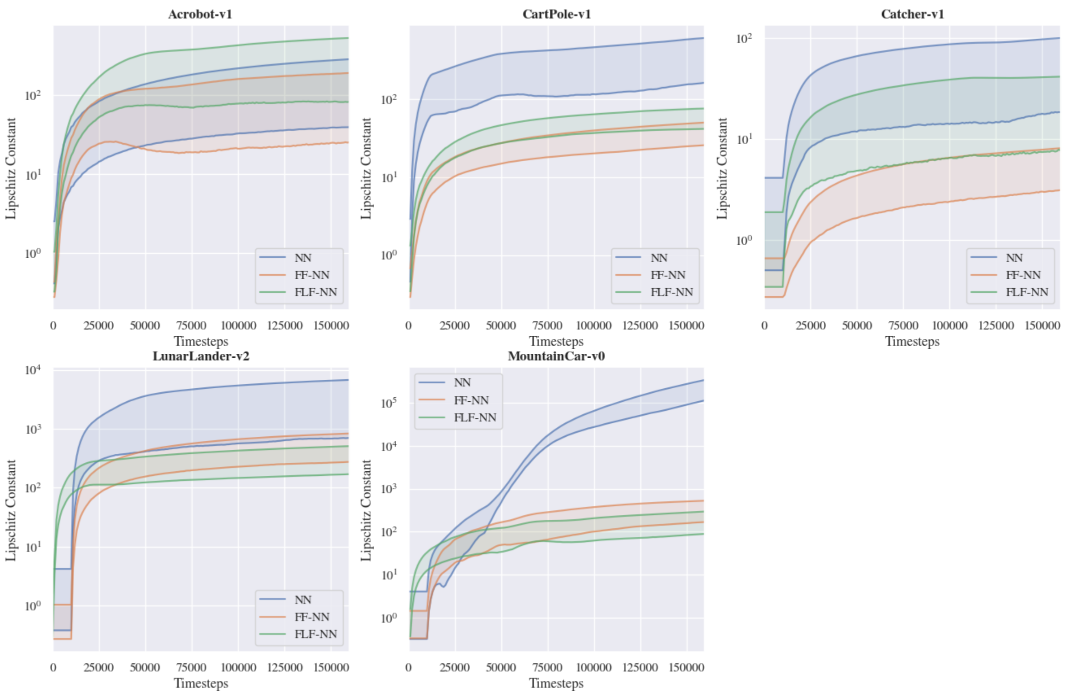
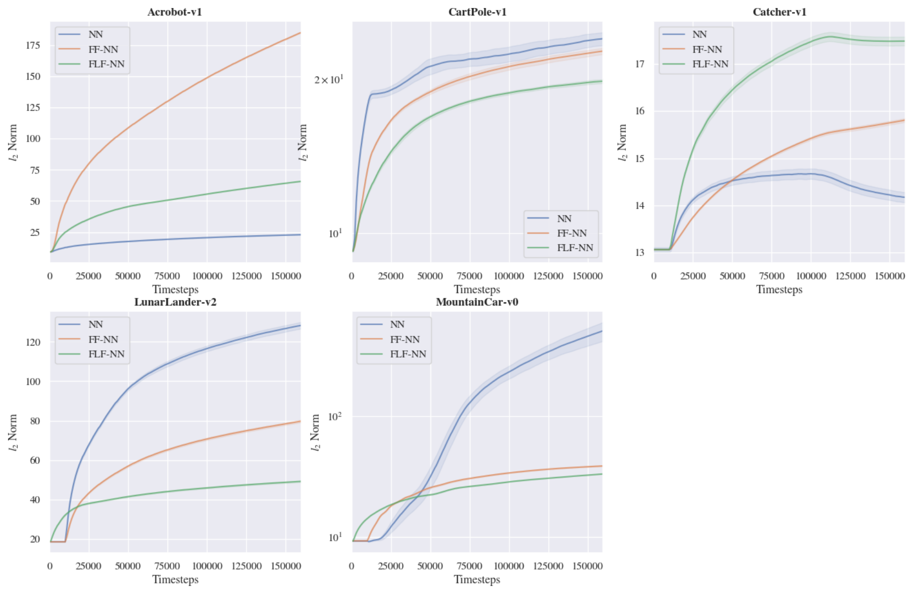
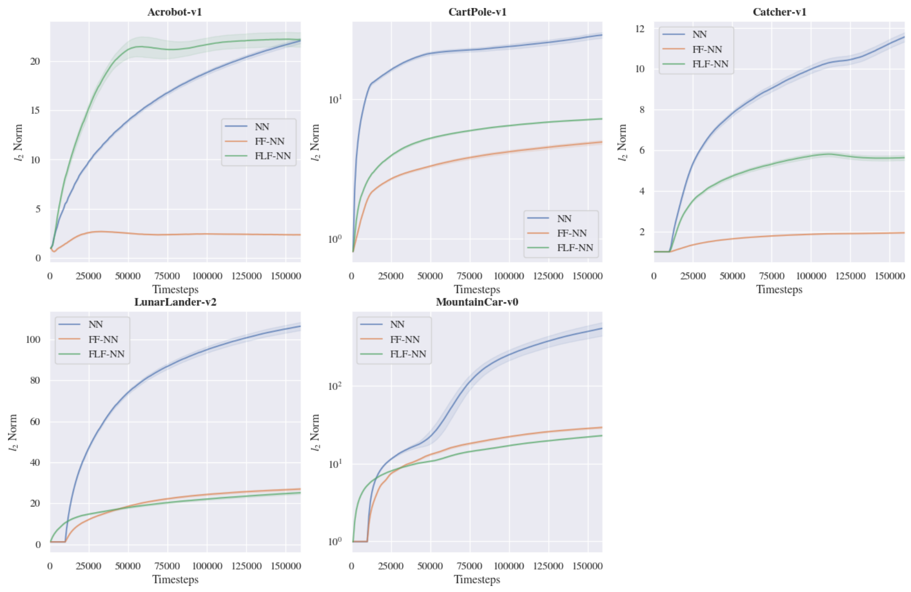


Figure 19: Learning curves of the lower and upper bounds of the Lipschitz constant of NN for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.

Simplicity of the Neural Network. Liu et al. (2020) observed that on many tasks, smaller policy weight norms correlate with better generalization ability. Figure 20, Figure 21, Figure 22 visualize the l_2, l_1, l_{inf} weight norms on both layers, respectively. In most cases, FF and FLF reduce the l_2 and l_1 weight norms especially in the second layer.

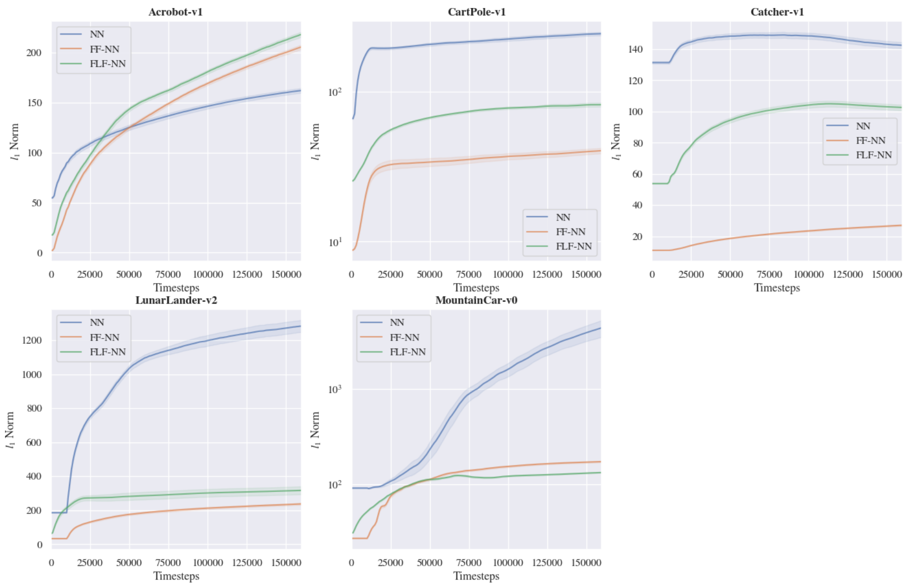


(a) In the First Layer

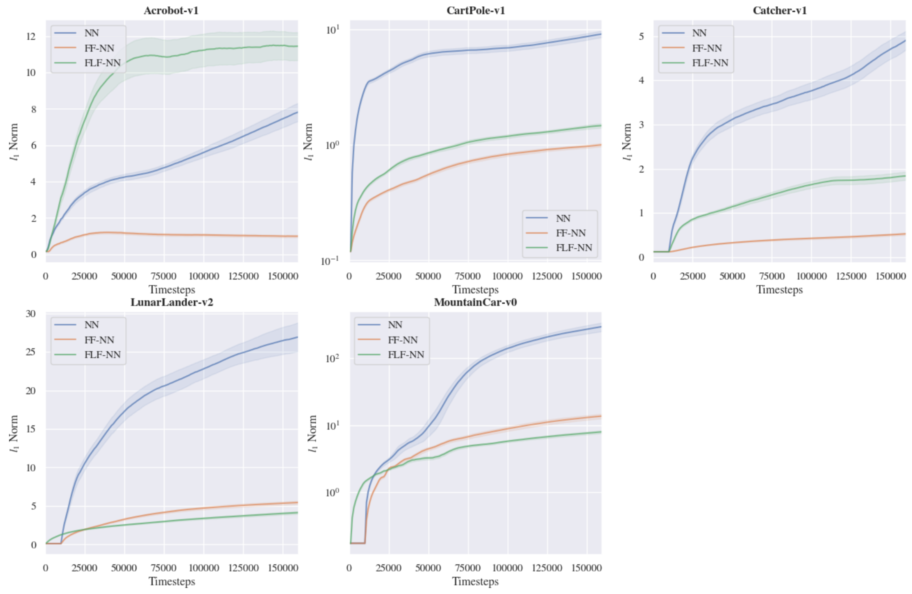


(b) In the Second Layer

Figure 20: Learning curves of the l_2 weight norm for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.

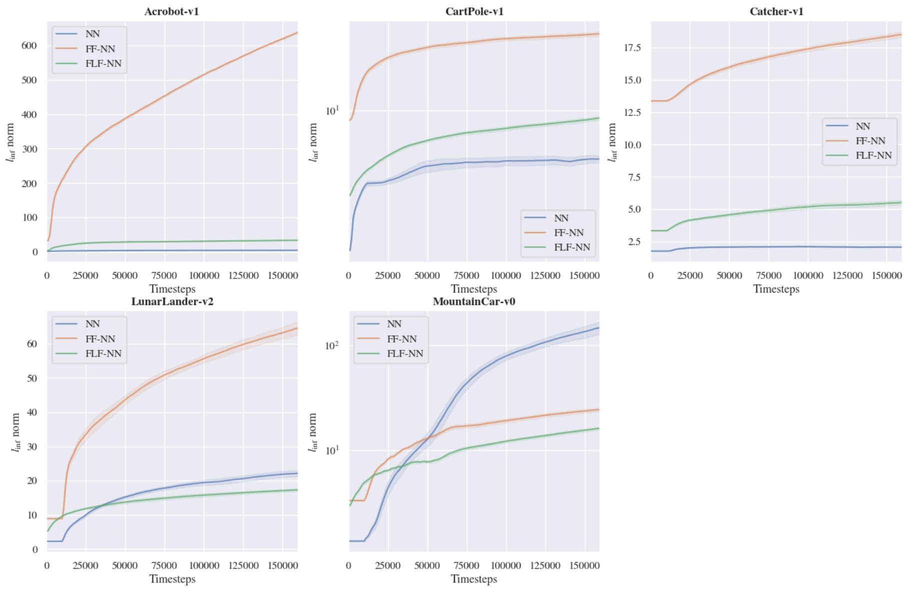


(a) In the First Layer

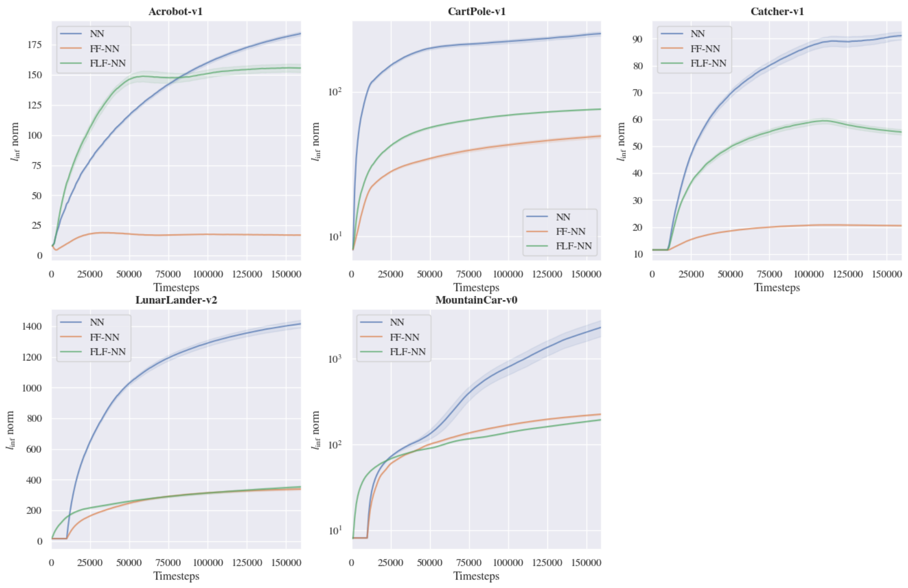


(b) In the Second Layer

Figure 21: Learning curves of the l_1 weight norm for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.



(a) In the First Layer



(b) In the Second Layer

Figure 22: Learning curves of the l_{inf} weight norm for NN fed with raw inputs (blue), Fourier features (orange), and Fourier Light features (green), averaged over 30 trainings with shading indicating the 95% CI.

I Other Input Preprocessings

Polynomial Features (PF). The feature vector consists of all polynomial combinations of the state variables with degree less than or equal to a specified degree (Lagoudakis & Parr, 2003; Sutton & Barto, 2018).

Random Fourier Features (RFF). Random Fourier Features were initially introduced to approximate an arbitrary stationary kernel function by exploiting Bochner's theorem (Rahimi & Recht, 2007). Recent works have shown promising results where RFF boost performance of deep neural networks (Mehrkanoon & Suykens, 2018), reduce the probability of misclassification (Mitra & Kaddoum, 2021), or can facilitate MLPs to learn high-frequency functions (Tancik et al., 2020). In RL, Rajeswaran et al. (2017) used them with Natural Policy Gradient to outperform performance obtained with NNs. The i-th feature of the Random Fourier Feature mapping $\mathrm{RFF} : \mathbb{R}^n \to \mathbb{R}^p$ is

$$\mathrm{RFF}_i(\boldsymbol{s}) = {}^2/\sqrt{p}\cos\Big(\psi(\boldsymbol{s})^T \boldsymbol{c}_i + b_i\Big) \tag{6}$$

where $\boldsymbol{c} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_p)$, $\boldsymbol{b} \sim \mathcal{U}(0, 2\pi)$, ψ is a normalizer, and p is the number of features we want to generate. The term $2/\sqrt{p}$ is used as normalization factor to reduce the variance of the estimates. RFF and Fourier features have a very close definition, except that the vector $\boldsymbol{c}$ creating interaction between state variables is sampled from a normal distribution. RFF are studied to understand if it is rather the structure or the choice of $\boldsymbol{c}$ that can explain the good performances.

Tile Coding (TC). Tile Coding (Albus, 1971; Sutton & Barto, 2018) is a generalization of state aggregation, in which we cover the state space with overlapping grids (tilings) where each grid divides the state space into small squares (tiles). The representation of a state for each tiling is a one-hot vector of dimension the number of tiles that has one for the tile where the state is in and zero otherwise. Concatenation of one-hot vectors for each tiling forms Tile Coding features. A nice property of Tile Coding is that generalization occurs to states other than the one trained if those states fall within any of the same tiles. Ghiassian et al. (2020) already proposed to preprocess inputs of Neural Network with Tile Coding to promote sparsity of learned representations and obtain better performance.

Experiments. We study performance on two discrete control environments, MountainCar-v0 and CartPole-v1 using the DQN algorithm, optimizing hyperparameter with Optuna (Appendix J), but consider the same MLP architecture for DQN and each input preprocessing. All experiments are averaged over 30 runs (30 different random seeds), with offline evaluations performed on the policy every $1,000$ training/environment timesteps.

J Reproducing Experiments

The entire code for reproducing all experiments will be made publicly available on github at the end of the double-blind review.

Discrete Control Tasks. We compare the performance of DQN on five discrete-action environments from OpenAI Gym (Brockman et al., 2016): Acrobot-v1, CartPole-v1, LunarLander-v2, MountainCar-v0, and Catcher-v1 (Tasfi, 2016). We use a MLP architecture with a single hidden layer.

Continuous Control Tasks. We compare performance on three continuous-action control tasks from Mujoco (Todorov et al., 2012): HalfChettah-v2, InvertedDoublePendulum-v2, and Swimmer-v2. Because of the higher state dimension, we were only able to test Fourier Light Features. We use a MLP with two hidden layers.All experiments are averaged over 10 runs, with offline evaluations performed on the policy every $1,000$ environment timesteps.

Common Settings. The first for discrete action domains are from OpenAI Gym (Brockman et al., 2016) with version 0.18.0 and continuous action domains from Mujoco (Todorov et al., 2012). The discrete action environment Catcher-v1 is from PyGame-Learning-Environment (Tasfi, 2016). For Deep Reinforcement Learning implementations, we adopt the code from StableBaselines-3 (Raffin et al., 2019) with version 0.10.0 based on Pytorch 1.8.0. We use Adam optimizer (Kingma & Ba, 2014), Xavier initializer (Glorot & Bengio, 2010), and ReLU activation functions across all experiments. We evaluate offline each algorithm every $1,000$ training/environment timesteps.

Table 5: **Sampling Values for DQN**

Hyperparameter	Range
Number of Hidden Layers	1
Number of Neurons per Hidden Layer	$\{16, 32, 64, 128, 256\}^1$
Batch Size	$\{16, 32, 64, 100, 128, 256, 512\}$
Replay Buffer Size	$\{1e4, 5e4, 1e5, 1e6\}$
Discount Factor	$\{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999\}$
Learning rate	$[1e-5, 1]$
Target Update Frequency	$\{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999\}$
Train Frequency	$\{1, 4, 8, 16, 128, 256, 1000\}$
Exploration Fraction	$[0, 0.5]$
Final Value of Random Action Probability	$[0, 0.2]$
Fourier Order (FF)	$\{1, 2, 3, 4, 5\}$

Hyperparameter Sampling on Discrete Control Tasks. Since hyperparameters for the discrete control tasks were not included at the time of experiments, we tune hyperparameters with Optuna 2.4.0 (Akiba et al., 2019) for each discrete control task. We sample hyperparameter values in Table 5 using the TPE (Tree-structured Parzen Estimator) algorithm (Bergstra et al., 2011). Since TPE can be very sensitive to the scores of the first trials, we run 5 independent hyperparameter research where one research is composed of 500 trials. Each trial corresponds to a training of $120,000$ timesteps with hyperparameters sampled by TPE and the score of each trial is based on the return of 100 rollouts of the learned policy. Because Deep RL algorithms are highly unreliable (Henderson et al., 2018; Islam et al., 2017), we take the 15 best hyperparameter settings returned by Optuna and run 5 additional trainings of $150,000$ timesteps. The best hyperparameter setting we take is the one with the better final average return over the 5 trainings.

For searching the learning rate and Fourier Order of FF and FLF, we adopt the same strategy by doing just one research with Optuna. For the Fourier Light order, we sample values between 1 and the Fourier Light order corresponding to the number of traditional Fourier features found previously for the research for FF.

Hyperparameter Sampling on Continuous Control Tasks. On continuous tasks, we keep the hyperparameters from the original codebase, StableBaselines-3-zoo (Raffin, 2020). For searching the learning rate and Fourier Light Order in continuous control tasks, we adopt the same strategy as the one detailed in the previous subsection by doing just one research of 500 trials with Optuna. In this setting, each trial corresponds to a training of $800,000$ timesteps with hyperparameters sampled by TPE. Similarly, the best hyperparameter setting we take is the one with the better final average return over 5 trainings of $800,000$ timesteps. The range of values for the Fourier Light order in continuous control tasks is between 1 and 50.

Normalization. Before being passed into the Fourier features mapping or Fourier Light features mapping, input data need to be normalized by a normalizer $\psi : \mathbb{R}^n \to [0, 1]^n$ (Section 2). We compute either the min/max normalization when state variables are bounded or we apply a nonlinear transformation based on $\arctan$ where:

$$\psi_i(\boldsymbol{s}) = \frac{\arctan(\frac{\boldsymbol{s}_i - \boldsymbol{t}_i}{\boldsymbol{c}_i})}{\pi} + 0.5 \tag{7}$$

with $\boldsymbol{t}$ the normalizer shift parameter and $\boldsymbol{c}$ the normalizer scale parameter. $\boldsymbol{t}$ and $\boldsymbol{c}$ are arbitrarily chosen to obtain $\psi_i(\boldsymbol{s}_{\text{max}}) = 0.9$ and $\psi_i(\boldsymbol{s}_{\text{min}}) = 0.1$ where $\boldsymbol{s}_{\text{max},i}$ and $\boldsymbol{s}_{\text{min},i}$ are respectively the maximum and minimum state variable i values observed over $5 \times 5,000$ rollouts generated by 5 suboptimal policies.

Training timesteps. For DQN, we run $160,000$ timesteps for Acrobot-v1, CartPole-v1, Catcher-v1, LunarLander-v1 and MountainCar-v0. For Catcher-v1, we fix the episode length to 500 timesteps. For PPO, we run $2e8$ timesteps for HalfCheetah-v2 and Swimmer-v2 and $1e8$ timesteps for InvertedDoublePendulum-v2.

Implementation Details for Standard Input Preprocessing. For generating Polynomial features we use Scikit-Learn (Buitinck et al., 2013) and for Tile Coding features we adopt the implementation of Patterson (2020). For Random Fourier features and Tile Coding features, inputs are first normalized with the same normalization defined for Fourier features/Fourier Light features. In our first experiments, we noticed that just changing the learning rate was not sufficient to achieve convergence. We then keep the same MLP architecure and found another hyperparameter setting with Optuna by using the same hyperparameter ranges reported in Table 5. According to the experiments performed by Ghiassian et al. (2020), we set the number of tilings to 8 and the number of tiles to 4.

Hyperparameter Setting. The following tables report the hyperparameter values found using the approach described above.

Table 6: DQN Hyperparameter Setting found for **Acrobot-v1**.

Hyperparameter	DQN	FF-DQN	FLF-DQN
Number of Neurons per Hidden Layer	256	256	256
Number of Hidden Layers	1	1	1
Batch Size	256	256	256
Replay Buffer Size	1000000	1000000	1000000
Discount Factor	0.995	0.995	0.995
Learning Rate	$2.20e-04$	$5.25e-04$	$1.40e-03$
Target Update Frequency	450	450	450
Gradient Steps	256	256	256
Train Frequency	256	256	256
Exploration Fraction	0.032343	0.032343	0.032343
Final Value of Random Action Probability	0.101575	0.101575	0.101575
Fourier Order	–	3	10

Table 7: DQN Hyperparameter Setting found for **CartPole-v1**.

Hyperparameter	DQN	FF-DQN	FLF-DQN
Number of Neurons per Hidden Layer	256	256	256
Number of Hidden Layers	1	1	1
Batch Size	128	128	128
Replay Buffer Size	10000	10000	10000
Discount Factor	0.99	0.99	0.99
Learning Rate	$1.10e-03$	$5.85e-05$	$5.32e-05$
Target Update Frequency	350	350	350
Gradient Steps	8	8	8
Train Frequency	16	16	16
Exploration Fraction	0.038783	0.038783	0.038783
Final Value of Random Action Probability	0.069806	0.069806	0.069806
Fourier Order	–	3	7

Table 8: DQN Hyperparameter Setting found for **Catcher-v1**.

Hyperparameter	DQN	FF-DQN	FLF-DQN
Number of Neurons per Hidden Layer	512	512	512.0
Number of Hidden Layers	1	1	1
Batch Size	256	256	256
Replay Buffer Size	100000	100000	100000
Discount Factor	0.99	0.99	0.99
Learning Rate	$8.72e-04$	$8.17e-05$	$4.37e-04$
Target Update Frequency	250	250	250
Gradient Steps	2	2	2
Train Frequency	8	8	8
Exploration Fraction	0.087147	0.087147	0.087147
Final Value of Random Action Probability	0.166414	0.166414	0.166414
Fourier Order	$-$	4	6

Table 9: DQN Hyperparameter Setting found for **LunarLander-v2**.

Hyperparameter	DQN	FF-DQN	FLF-DQN
Number of Neurons per Hidden Layer	1024	1024.0	1024
Number of Hidden Layers	1	1	1
Batch Size	100	100	100
Replay Buffer Size	1000000	1000000	1000000
Discount Factor	0.99	0.99	0.99
Learning Rate	$1.47e-03$	$1.34e-04$	$8.29e-05$
Target Update Frequency	50	50	50
Gradient Steps	128	128	128
Train Frequency	128	128	128
Exploration Fraction	0.141211	0.141211	0.141211
Final Value of Random Action Probability	0.188717	0.188717	0.188717
Fourier Order	$-$	1	9

Table 10: DQN Hyperparameter Setting found for **MountainCar-v0**.

Hyperparameter	DQN	FF-DQN	FLF-DQN
Number of Neurons per Hidden Layer	256	256	256
Number of Hidden Layers	1	1	1
Batch Size	64	64	64
Replay Buffer Size	100000	100000	100000
Discount Factor	0.995	0.995	0.995
Learning Rate	$3.40e-03$	$4.43e-04$	$3.93e-04$
Target Update Frequency	200	200	200
Gradient Steps	128	128	128
Train Frequency	128	128	128
Exploration Fraction	0.420173	0.420173	0.420173
Final Value of Random Action Probability	0.093216	0.093216	0.093216
Fourier Order	$-$	4	9

Table 11: PPO Hyperparameter Setting of **HalfCheetah-v2** (Raffin, 2020)

Hyperparameter	PPO	FLF-PPO
Number of Neurons per Hidden Layer	256	256
Number of Hidden Layers	2	2
Batch Size	64	64
Clip Range	0.1	0.1
Learning Rate	$2.0633e-05$	$6.8897e-05$
Number of Epochs	20	20
GAE Parameter	0.92	0.92
Entropy Coefficient	$4.0176e-04$	$4.0176e-04$
Max Gradient Norm	0.8	0.8
Number of Steps	512	512
Value Function Coefficient	0.5096	0.5096
Fourier Order	–	3

Table 12: PPO Hyperparameter Setting of **InvertedDoublePendulum-v2** (Raffin, 2020)

Hyperparameter	PPO	FLF-PPO
Number of Neurons per Hidden Layer	256	256
Number of Hidden Layers	2	2
Batch Size	64	64
Clip Range	0.1	0.1
Learning Rate	$2.0633e-05$	$6.8897e-05$
Number of Epochs	20	20
GAE Parameter	0.92	0.92
Entropy Coefficient	$4.0176e-04$	$4.0176e-04$
Max Gradient Norm	0.8	0.8
Number of Steps	512	512
Value Function Coefficient	0.5096	0.5096
Fourier Order	–	3

Table 13: PPO Hyperparameter Setting of **Swimmer-v2** (Raffin, 2020)

Hyperparameter	PPO	FLF-PPO
Number of Neurons per Hidden Layer	256	256
Number of Hidden Layers	2	2
Batch Size	32	32
Clip Range	0.3	0.3
Learning Rate	$5.4972e-05$	$5.0205e-05$
Number of Epochs	10	10
GAE Parameter	0.95	0.95
Entropy Coefficient	$5.5476e-02$	$5.5476e-02$
Max Gradient Norm	0.6	0.6
Number of Steps	512	512
Value Function Coefficient	0.38782	0.38782
Fourier Order	–	3

Table 14: DQN Hyperparameter Setting found for **Polynomial Features**.

Hyperparameter	CartPole-v1	MountainCar-v0
Number of Neurons per Hidden Layer	256	256
Number of Hidden Layers	1	1
Batch Size	512	128
Replay Buffer Size	50000	100000
Discount Factor	0.9999	0.98
Learning Rate	$2.57e-04$	$6.54e-03$
Target Update Frequency	450	250
Gradient Steps	8	4
Train Frequency	8	16
Exploration Fraction	0.011846	0.084238
Final Value of Random Action Probability	0.038869	0.120751
Polynomial Degree	5	2

Table 15: DQN Hyperparameter Setting found for **Random Fourier Features**.

Hyperparameter	CartPole-v1	MountainCar-v0
Number of Neurons per Hidden Layer	256.0	256
Number of Hidden Layers	1.0	1
Batch Size	512	100
Replay Buffer Size	1000000	1000000
Discount Factor	0.9990000000000001	0.98
Learning Rate	$1.40e-02$	$1.67e-05$
Target Update Frequency	1	300
Gradient Steps	32	128
Train Frequency	128	256
Exploration Fraction	0.025779	0.345071
Final Value of Random Action Probability	0.138125	0.193399
Number of RFF	256	512

Table 16: DQN Hyperparameter Setting found for **Tile Coding Features**.

Hyperparameter	CartPole-v1	MountainCar-v0
Number of Neurons per Hidden Layer	256.0	256.0
Number of Hidden Layers	1.0	1.0
Batch Size	64	256
Replay Buffer Size	10000	1000000
Discount Factor	0.995	0.98
Learning Rate	$8.96e-05$	$7.95e-05$
Target Update Frequency	300	200
Gradient Steps	4	256
Train Frequency	4	256
Exploration Fraction	0.246283	0.316252
Final Value of Random Action Probability	0.000285	0.069128
Number of Tilings	8.0	8.0
Number of Tiles per Tiling	4.0	4.0

Table 17: DQN Hyperparameter Setting of **Acrobot-v1** used in Appendix A (Raffin, 2020)

Hyperparameter	DQN	FLF-DQN
Number of Neurons per Hidden Layer	256	256.0
Number of Hidden Layers	3	3.0
Batch Size	256	256
Replay Buffer Size	100000	100000
Discount Factor	0.999	0.999
Learning Rate	$8.09e-04$	$9.95e-04$
Target Update Frequency	200	200
Gradient Steps	1	1
Train Frequency	1	1
Exploration Fraction	0.296189	0.296189
Final Value of Random Action Probability	0.000437	0.000437
Fourier Order	$-$	3

Table 18: DQN Hyperparameter Setting of **CartPole-v1** used in Appendix A (Raffin, 2020)

Hyperparameter	DQN	FLF-DQN
Number of Neurons per Hidden Layer	256.0	256.0
Number of Hidden Layers	2.0	2.0
Batch Size	64	64
Replay Buffer Size	100000	100000
Discount Factor	0.99	0.99
Learning Rate	$2.30e-03$	$1.34e-05$
Target Update Frequency	10	10
Gradient Steps	128	128
Train Frequency	256	256
Exploration Fraction	0.16	0.16
Final Value of Random Action Probability	0.04	0.04
Fourier Order	$-$	10

Table 19: DQN Hyperparameter Setting of **MountainCar-v0** used in Appendix A (Raffin, 2020)

Hyperparameter	DQN	FLF-DQN
Number of Neurons per Hidden Layer	256	256
Number of Hidden Layers	2	2
Batch Size	128	128
Replay Buffer Size	10000	10000
Discount Factor	0.98	0.98
Learning Rate	$4.00e-03$	$5.04e-04$
Target Update Frequency	600	600
Gradient Steps	8	8
Train Frequency	16	16
Exploration Fraction	0.2	0.2
Final Value of Random Action Probability	0.07	0.07
Fourier Order	$-$	4