Meta Reinforcement Learning on Neural Networks

Alexander D. Cai
Harvard College
Cambridge, MA 02138
`alexcai@college.harvard.edu`

Oliver Cheng
Harvard College
Cambridge, MA 02138
`ocheng@college.harvard.edu`

Abstract

In the field of reinforcement learning (RL), the subfield of meta-learning has recently gained popularity. Meta-learning proposes to not only learn the policy (which selects actions), but rather a flexible *RL algorithm* which learns the policy. This claims to make the training process more data-efficient and generalizable across tasks. We build off of work from Wang et al. [8] that investigates the implications for biological learning. The structure is as follows: an overarching *policy network* mimics the dopamine system by guiding a fully connected neural network to update its synaptic weights. Over time, the policy network learns an optimal learning rule for which to update the weights in the neural network. We empirically find that the optimal policy is to quickly jump to the optimal weights for the training distribution, which does not generalize as well as other common learning rules such as Stochastic Gradient Descent (SGD), Weight Perturbation (WP), and Node Perturbation (NP) on teacher-student setups and MNIST.

1 Introduction to Meta-Reinforcement Learning

In recent years there has been a growing amount of excitement about *meta-learning* in order to solve a wider set of problems. In the typical reinforcement learning setup, an agent interacts with an environment by taking actions and receiving rewards. The goal is to maximize the total reward received. We can formalize this as a Markov Decision Process (MDP) $\{\mathcal{S}, \mathcal{A}, P, R, \gamma\}$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ governs the state transitions, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Let π_θ be the parameterized policy. Thus for some MDP $\mathcal{M}$, the goal is to maximize the mean discounted reward

$$\mathcal{J}_{\mathcal{M}}(\theta) = \mathbb{E}_{a_{t=0}^{\infty} \sim \pi_\theta(s_t), s_{t+1} \sim P(s_t, a_t)} \left(\sum_t \gamma^t R(s_t, a_{t-1}) \right).$$

This framework gives us a way to view human beings and the nervous system as a reinforcement learning program where the human is the agent. One notable trait of the human brain is that it learns and maintains numerous distinct skills, as opposed to typical RL algorithms which can only optimize for a given task. To generalize, we instead seek to develop meta-learning algorithms, where rather than optimizing a single policy for a single task, we aim to learn an RL algorithm which finds optimal policies over a diverse distribution of tasks. This idea is motivated by the idea in psychology of "learning how to learn" and is conceptually closer to how the brain works [8].

2 (Multi-Agent) Reinforcement Meta-Learning to find Optimal Learning Rules

We model neural network training as follows.

Preprint. Under review.

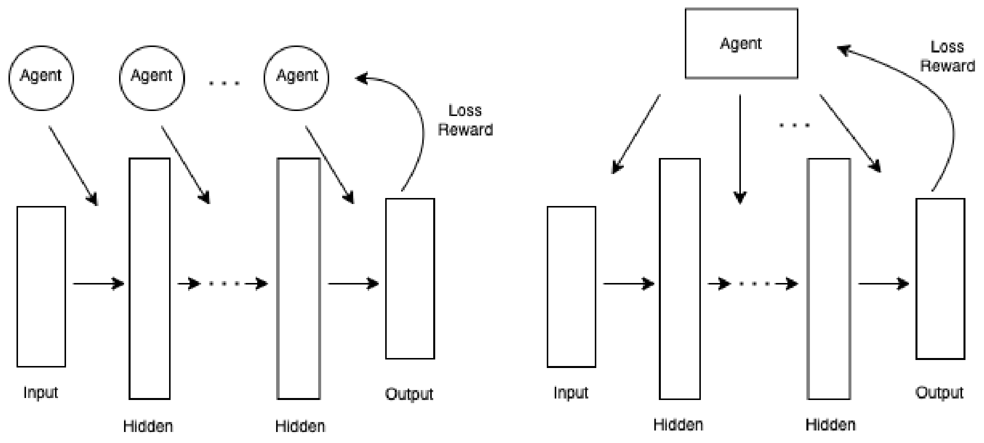

Figure 1: Our neural meta-learning paradigm, framed as a multi-agent problem (left) and a single-agent problem (right). We seek to find the parameters θ which maximize reward, which we define as the negative of the training loss on a fixed training dataset. We do this by optimizing the policy network's parameters using Proximal Policy Optimization [6]. In the multi-agent framing, each agent is in control of a set of weights and can only see the activations of its own weights and those of the previous agent. All agents receive the same total reward. In the single-agent framing, a single agent has complete control and knowledge of all weights.

A very basic way to model a brain is as a simple feedforward neural network where each neuron is its own agent and we are optimizing a multi-agent reinforcement learning problem. Then, to imitate a model-based dopamine system which determines which synapses are reinforced, we consider a policy neural network which will update the weights for the neural net [8, 1]. The state space of this system is essentially the possible weights that this neural net can have.

From here we can construct two different systems. One is where a single agent gets information on all of the weights, and can take actions to improve weights accordingly. This is less biologically plausible since it would be difficult for any neuron in the brain to have access to have all of this data. The second system we consider is setting one agent for each layer. Each agent has information about the previous layer's weights and activations, and is able to control the weights to the next layer. In other words, the agent corresponding to the the ℓ-th layer, $\mathbf{x}^{(\ell)}$, has control over the weights $[\mathbf{W}^{(\ell)}]_{ij}$. This is more biologically plausible as this would imply each agent need not have connections to all the neurons in the network, but rather a local subset of them. Unfortunately, although the second system is conceptually closer to biology, the implementation with current libraries is shaky at best. Theoretically it would not perform any better than the first system, as each agent has less information to operate on, and since the policy is a deep neural network, the loss of this information would be a large hindrance to performance.

We use Proximal Policy Optimization (PPO) in order to progressively improve the policy. We can define our reward as the negative loss for the neural network, thus having smaller absolute loss will be a larger reward, and PPO will find a policy that optimizes for the largest expected reward. To compute the reward for any given policy, we take T different trajectories. For each trajectory t we find how the loss E_t decreases after a successive forward passes (1e5 epochs) through the neural network with weights specified by the agent/policy network. The reward is then transmitted as $\langle -E_t \rangle$ and the policy is updated. This process constitutes a single timestep.

Recall from SGD we have update rules that involve using the gradient to traverse through the loss landscape, in our case we are learning a learning rule and hence the meta "learning how to learn" concept. The goal is for the policy to converge to a learning rule that is optimal for the given task, which will not necessarily be SGD or learning with global error signals that we will look at below, but rather one that is more fine-tuned to the particular set of problems, for example learning from a teacher with noise or classifying digits in the MNIST dataset.

Code Implementation The project is implemented using the popular OpenAI Gym and Stable Baselines 3 libraries for reinforcement learning [2, 5]. We implement a custom Gym environment that takes the base neural network architecture as an input, along with the training dataset used for calculating the loss. Then at each timestep, the environment receives a candidate set of weights for the base network, and evaluates its loss on the given dataset.

We attempted to implement a multi-agent environment using the more advanced but flexible Tianshou library and the Petting Zoo library for MARL. This implementation is still a work in progress, but if completed, would allow different agents to control different parts of the base network [9, 7].

3 Comparison to Known Learning Rules

We compare this meta-learning, or learning learning rules, to accepted learning rules in the literature. SGD converges the quickest, but due to the weight transport problem, is not biologically plausible [4]. More biologically plausible alternatives involve learning from a global error signal, and involve perturbation-type learning rules. Global error signals have been observed in the brain, but it is still under much research exactly how the brain uses these error signals in order to learn and update neurons. We will look at NP and WP. We compare how our new model does for teacher-student learning and learning to classify the MNIST dataset.

The idea behind NP and WP is to take some perturbation ξ of either the weights or the nodes, and then see how that changes the objective or loss. The learning rule update then corresponds to scaling this perturbation by whether or not it increases or decreases the loss. Consider a general neural network with L layers and a data set $\{\mathbf{x}_i^{(0)}, \mathbf{y}_i\}_{i=1}^N$. Let $\mathbf{W}_\ell \in \mathbb{R}^{R_{\ell+1} \times R_\ell}$ where R_ℓ is the number of neurons in layer ℓ. We define the forward pass as

$$\mathbf{X}^{(\ell)} = \sigma\left(\mathbf{X}^{(\ell-1)}\mathbf{W}_{\ell-1}^\top\right). \tag{1}$$

Where σ is some non-linearity such as ReLU. We use MSE loss for student teacher

$$E = \frac{1}{NR_L}\frac{1}{2}\|\mathbf{X}^{(L)} - \mathbf{Y}\|^2. \tag{2}$$

and for classification into $C = R_L$ classes let

$$p(\mathbf{x}_i^{(L)} = r) = \frac{\exp x_{j,r}^{(L)}}{\sum_{j=1}^C \exp x_{i,j}^{(L)}} \tag{3}$$

we use cross entropy loss

$$E = -\frac{1}{N}\sum_{i=1}^N \sum_{r=1}^C y_r \log p(\mathbf{x}_i^{(L)} = r)$$

In SGD, we use backpropagation to update the weight parameters by

$$\Delta_{SGD}\mathbf{W}_\ell = \eta \frac{\partial E}{\partial \mathbf{W}_\ell},$$

where the chain rule means we need to "transport" these values backwards through the neural network. On the other hand, in WP/NP, we have two forward passes. For the first, we define the same as 1, however for NP, we will let the perturbation be a vector $\xi \in \mathbb{R}^{N \times R_\ell}$ where all entries are distributed i.i.d $\mathcal{N}(0, \sigma)$. We can then define the forward pass as

$$\widetilde{\mathbf{X}}^{(\ell)} = \sigma\left(\widetilde{\mathbf{X}}^{(\ell-1)}\mathbf{W}_{\ell-1}^\top + \xi^{(\ell)}\right).$$

While for WP we take $\mathbf{\Xi}^{(\ell)} \in \mathbb{R}^{R_{\ell+1} \times R_\ell}$ where all entries are distributed i.i.d $\mathcal{N}(0, \sigma)$.

$$\hat{\mathbf{X}}^{(\ell)} = \sigma\left(\hat{\mathbf{X}}^{(\ell-1)}\left(\mathbf{W}_{\ell-1} + \mathbf{\Xi}^{(\ell-1)}\right)^\top\right).$$

We can then define two errors for each NP and WP respectively. For MSE

$$E_N = \frac{1}{NR_L}\frac{1}{2}\|\widetilde{\mathbf{X}}^{(L)} - \mathbf{Y}\|^2 \qquad E_W = \frac{1}{NR_L}\frac{1}{2}\|\hat{\mathbf{X}}^{(L)} - \mathbf{Y}\|^2.$$

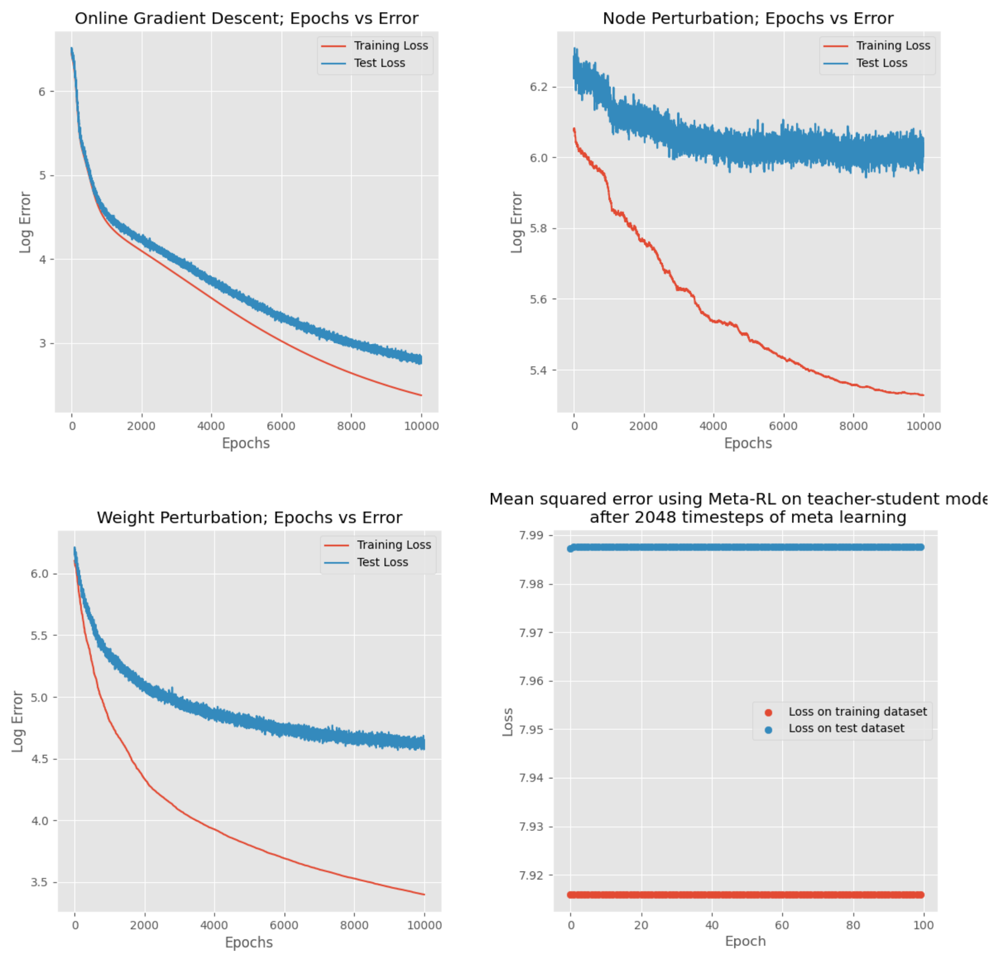

Figure 2: We find that the policy that the meta-RL system finds is to instantly jump to a certain set of weights, and then perform no further updates. The learning rates were chosen to optimize the loss, in the range of $\eta = 1.0$ to 0.1. $\sigma = \frac{1}{L_y L_h}$.

For classification

$$E_N = -\frac{1}{N}\sum_{i=1}^{N}\sum_{r=1}^{C} y_r \log p(\widetilde{\mathbf{x}}_i^{(L)} = r) \qquad E_W = -\frac{1}{N}\sum_{i=1}^{N}\sum_{r=1}^{C} y_r \log p(\hat{\mathbf{x}}_i^{(L)} = r)$$

From here we can simply transmit the global error signal $E - E_N$ or $E - E_W$ and update our weights according to this error signal. In particular, we update parameters as follows for NP

$$\Delta_{NP}\mathbf{W}_\ell = \frac{\eta}{\sigma}(E - E_N)\sum_{i=1}^{R_{\ell+1}} \mathbf{x}_i^{(\ell)} \xi^{(\ell+1)\top}$$

While for WP we update as

$$\Delta_{WP}\mathbf{W}_\ell = \frac{\eta}{\sigma}(E - E_W)\mathbf{\Xi}^{(\ell)}$$

The idea is that perturbations which cause decrease the objective will be added to the parameters of the model. In expectation we have that up to a constant

$$\langle \Delta_{NP}\mathbf{W}_\ell \rangle_{\xi^{(\ell)}} = \langle \Delta_{WP}\mathbf{W}_\ell \rangle_{\mathbf{\Xi}^{(\ell)}} = \Delta_{SGD}\mathbf{W}_\ell.$$

Node and weight perturbation are often used in small teacher-student learning cases where there is only a single perceptron. Recent work by Hiratani et al. explores node perturbation in neural

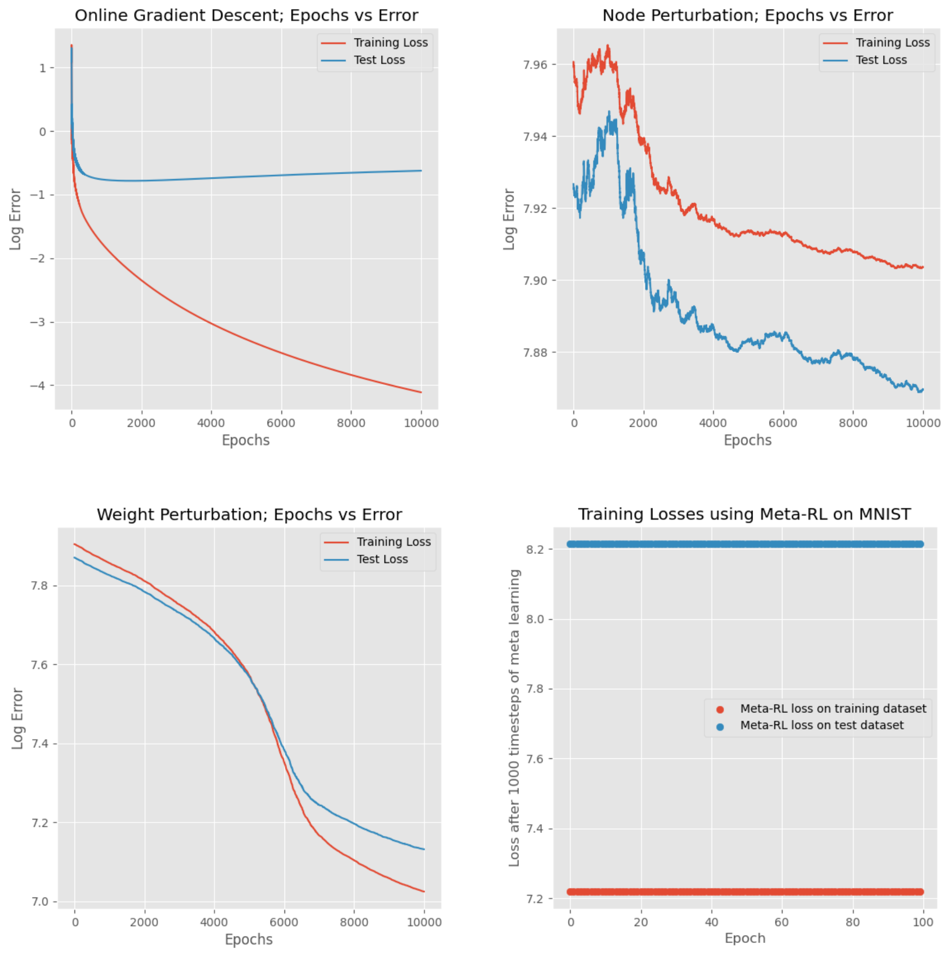

Figure 3: Again, we see the Meta-RL NN learn its weights instantly, and as a result have a constant loss for both training and testing. NP is set to $\eta = 10^{-4}$, which is unstable. NP will converge for $\eta \ll 10^{-6}$, but the number of epochs needed to make any reasonable progress was too high to be computationally feasible. $\sigma = \frac{1}{L_y L_h}$.

networks with a hidden layer, and finds computational and stability issues that suggest biological implausibility [3]. Nonetheless, these perturbation techniques remain an interesting way to investigate possible learning rules for neurons.

Training on Student-Teacher (Fig. 2) For the student-teacher model, we consider a neural network with one hidden layer. We set the three layer's width as $L_x = 10, L_h = 100, L_y = 10$. We have a teacher with a hidden set of weights $\mathbf{W}_h^*, \mathbf{W}_y^*$ that generates a dataset $N = 512$ with noise $\zeta_i \in \mathbb{R}^{L_y}$ for $i = 1, \ldots, N$, where each element of ζ_i is i.i.d $\mathcal{N}(0, \sigma_t)$. For our cases we set $\sigma_t = 0.01$ as node-perturbation will be very noisy and unstable otherwise. We define our dataset then as $\{\mathbf{x}_i^{(0)}, \mathbf{y}_i\}_{i=1}^N$ where each element of $\mathbf{x}_i$ is i.i.d standard normal, and

$$\mathbf{y}_i = \sigma\left(\sigma(\mathbf{x}_i \mathbf{W}_h^{*\top})\mathbf{W}_y^{*\top}\right) + \zeta_i$$

Where we used ReLU as the non-linearity for σ. We are thus optimizing a single-hidden layer neural network

$$\hat{\mathbf{y}}_i = \sigma\left(\sigma(\mathbf{x}_i \mathbf{W}_h^\top)\mathbf{W}_y^\top\right).$$

We are using MSE loss as in 2.

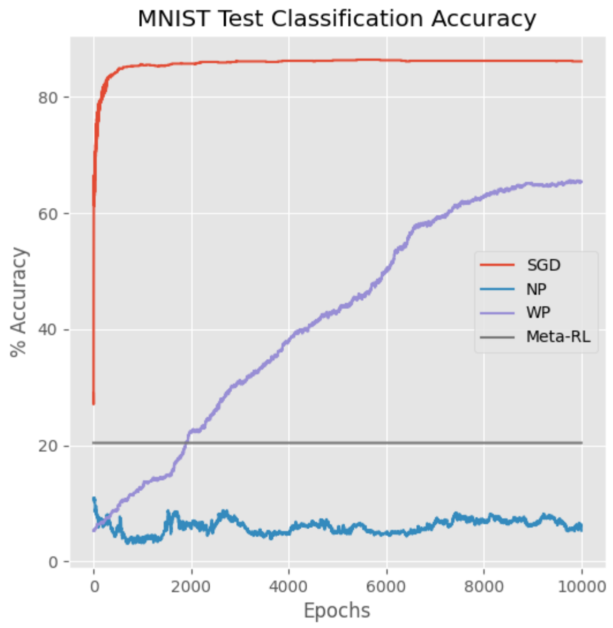

Figure 4: Test accuracy for Meta-RL is constant due to the policy network choosing the optimal weights during training instantly. NP is under performing due to being excessively noisy.

Training on MNIST (Fig. 3) Going to MNIST, the problem becomes much more noisy. In order to decrease noise, we first preprocess by standardizing and projecting the data onto the first 10 principal components ($\sim 99\%$ of the variance), and decrease the learning rate by a factor of 100. It is also necessary to increase the hidden layer width $L_h = 1000$, as otherwise the model did not have sufficient complexity. We train on a subset of the data $N = 512$. We use the same single-hidden layer architecture, with cross entropy loss as in 3 with $C = 10$ for the 10 different digits. As observed by Hiratani et al., the amount of training time required for these global error signal models to converge for MNIST is much higher (1e5) which was unfeasible for computationally [3]. We find the accuracy to be decent (50%) but clearly not optimized as SGD by itself can reach 80% testing accuracy on 10 principal components Fig. 4. In order to run without weights exploding, we standardized the weights at each step, thereby providing more stability at the cost of expressivity.

We found that one of the issues in our experiment was the tendency for the learning rule to choose a learning rule that updated the weights according to exactly the optimal weights of a training example, and hence substantial overfitting. This could be balanced by choosing more trajectories for each timestep.

4 Conclusion and Further Steps

Meta-Reinforcement Learning in the Brain Wang et al. discussed the an notion of "meta-learning" in the brain where the prefrontal cortex acts as a recurrent neural network which triggers actions and internally holding a notion of the value of a state. Phasic Dopamine (DA) is suggested to act in a similar way to how our policy network changed the weights of the neural network. In particular, the synaptic weights are adjusted by the model-free RL DA system, where DA releases are similar to the reward prediction error in temporal-difference RL algorithms. The reason why meta-RL fits this structure better than standard RL is because of the ability to use past experiences the brain has in order to learn an optimal policy, and this is clearly seen in humans and animal brains in general [8].

Computation Limits and Further Steps Wang et al. were able to achieve results relating meta-RL to the prefrontal cortex and dopamine system with the large computational power of DeepMind [8]. Our results show how the parameter size of models of this nature must be substantially larger in order to achieve results that are generalizable to even classic tasks such as MNIST or student-teacher. Further steps in this research include exploring a wider distribution of tasks $p(\mathcal{T})$ in order to harness the generalizability of meta-RL. Given more computational power it would also be interesting to see if the multi-agent meta-RL system has promise to generalize more as the optimal weights would be less clear due to the lack of information.

Code for this project can be found at `https://github.com/RubberNoodles/meta-MARL-neuronal-NN`.

References

[1] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis. Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*, 23(5):408–422, 2019. URL `https://www.sciencedirect.com/science/article/pii/S1364661319300610`.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[3] N. Hiratani, Y. Mehta, T. P. Lillicrap, and P. E. Latham. On the stability and scalability of node perturbation learning. 2022. URL `https://openreview.net/forum?id=XOCKM7QV5k`.

[4] M. P, A. RA, and J. MI. A more biologically plausible learning rule for neural networks. *Proc Natl Acad Sci U S A*, 15:4433–7, 1991.

[5] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[7] J. K. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sulivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. L. Williams, Y. Lokesh, R. Sullivan, and P. Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

[8] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick. Prefrontal cortex as a meta-reinforcement learning system. *Nature Neurosci*, 21: 860–868, 2018.

[9] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022. URL `http://jmlr.org/papers/v23/21-1127.html`.