Meta Reinforcement Learning on Neural Networks

Alexander D. Cai
Harvard School of Engineering
Harvard College
Cambridge, MA 02138
alexcai@college.harvard.edu

Oliver Cheng
Department of Mathematics
Harvard College
Cambridge, MA 02138
ocheng@college.harvard.edu

Abstract

Reinforcement learning (RL) has recently introduced meta-learning to make the training process substantially more data-efficient, reducing the amount of data required by orders of magnitude. This brings back the idea that RL learning can be a model and bring insights to how the brain performs its learning. We build off of work from Wang et al. at DeepMind to explore how meta-RL could potentially model how neurons learn [4]. The structure is as follows: an overarching policy network mimcs 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 show that the optimal policy is to quickly jump to the optimal weights, which does not generalize as well as other common learning rules such as Stochastic Gradient Descent (SGD), Weight Perturbation (WP), and Node Perturbation (NP).

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. This is often used in reinforcement learning tasks. In a standard reinforcement learning task, we have a policy that governs how an agent transitions between states in an environment. There are rewards that the agent can receive, and the optimal policy is one that maximizes rewards. We can formalize this by defining a reinforcement learning regime as a Markov Decision Process (MDP) where S is the set of states, A is the set of actions, T is the transition distribution (how states transition given an action), and R is the reward, which is a function of the state. Let π_θ be the policy, which is parameterized by some θ. Thus for some "task" $\mathcal{T}$, the goal is thus to maximize the objective

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

The task $\mathcal{T}$ determines the rewards, and hence the policy that is optimized depend heavily on what the task $\mathcal{T}$ is.

Reinforcement learning is used to solve many problems such as Go, Chess, Atari games, etc. where the environment is unknown and is structured in a way which an agent must discover an optimal strategy. This framework gives us a way to view human beings and the nervous system as a reinforcement learning program where we humans are the agent. (Un)Fortunately, this is a bit too basic of a way of modeling human beings and the brain, as reinforcement learning networks have not taken over humanity yet (citation needed). One of the reasons is that rewards and the task that an agent needs to optimize a policy for is constantly changing in the real world. Thus we introduce the idea of meta-learning, where rather than optimizing a single policy for a single task $\mathcal{T}$, we optimize for a learning strategy on how to optimize a policy for a distribution of tasks $p(\mathcal{T})$. This idea is motivated by the idea in psychology of "learning how to learn" and is conceptually closer to how the brain

Preprint. Under review.

works [4]. With meta-learning, due to the way we are improving the way we are finding the policy, this is a way we can introduce the idea of "past experiences" to help train policy for tasks that are similar.

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

A very basic way to model a brain is as a simple fixed architecture 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 [4, 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.

PettingZoo and others... Hi alex can you do this part

Code for can be found at `www.gethackedforfree.net`

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 [3]. 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

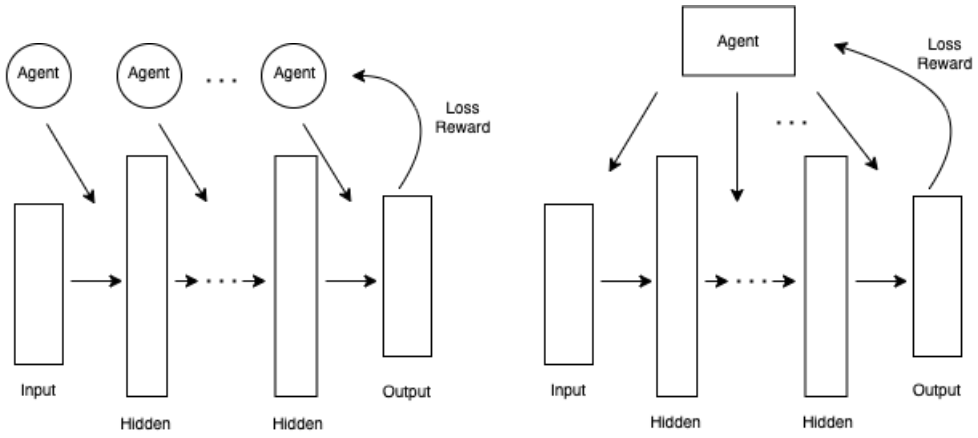

Figure 1: In both figures, we have a neural network as the lower loop. For any given timestep t, we have a policy $\pi(\theta_t)$ which governs the weight updates as the neural network performs `epochs = S` forward passes. There are a total of $\mathcal{T}$ different random initializations of weights at each timestep, each running for S epochs. At each epoch, the network emits a reward corresponding to the negative loss, which the agent will use to improve its policy for timestep $t+1$ to create $\pi(\theta_{t+1})$. On the left, we have a Multi-Agent Meta-RL system, where each agent is in control of a set of weights and can only see the previous layer's activations. Each agent receives the same total reward. On the right, we have a single agent that does all of the work of the agents in the multi-agent regime, thereby having complete knowledge of all weights.

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

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

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 networks with a hidden layer, and finds computational and stability issues that suggest biological implausibility [2]. 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.

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 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 [2]. 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 was balanced by choosing more trajectories for each timestep. Of course, this ended up creating a fairly general learning rule that wasn't able to improve substantially at all.

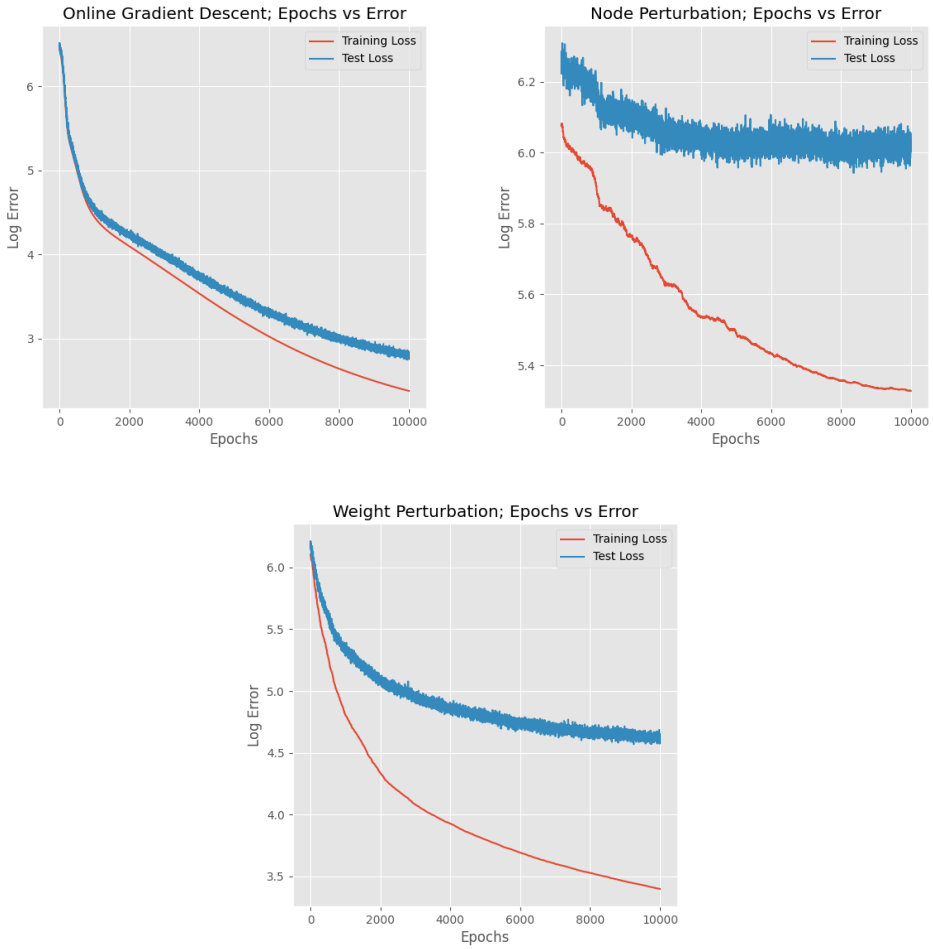

Figure 2: Discuss the results from teacher-student here

4 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.

Our Meta-MARL attempts include maybe this is garbage lol

In the Theory of Neural Computation, we can see reinforcement-learning-esque regimes. In particular researchers have found that the dopamine system seems to follow the TD algorithm, a reinforcement learning algorithm. In particular, the error function of the TD algorithm operates in the same way to how dopamine neurons learn their firing rates in order to associate stimuli with future rewards. In this case, the dopamine neuron is the agent and it can change its firing rate as an action.

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`.

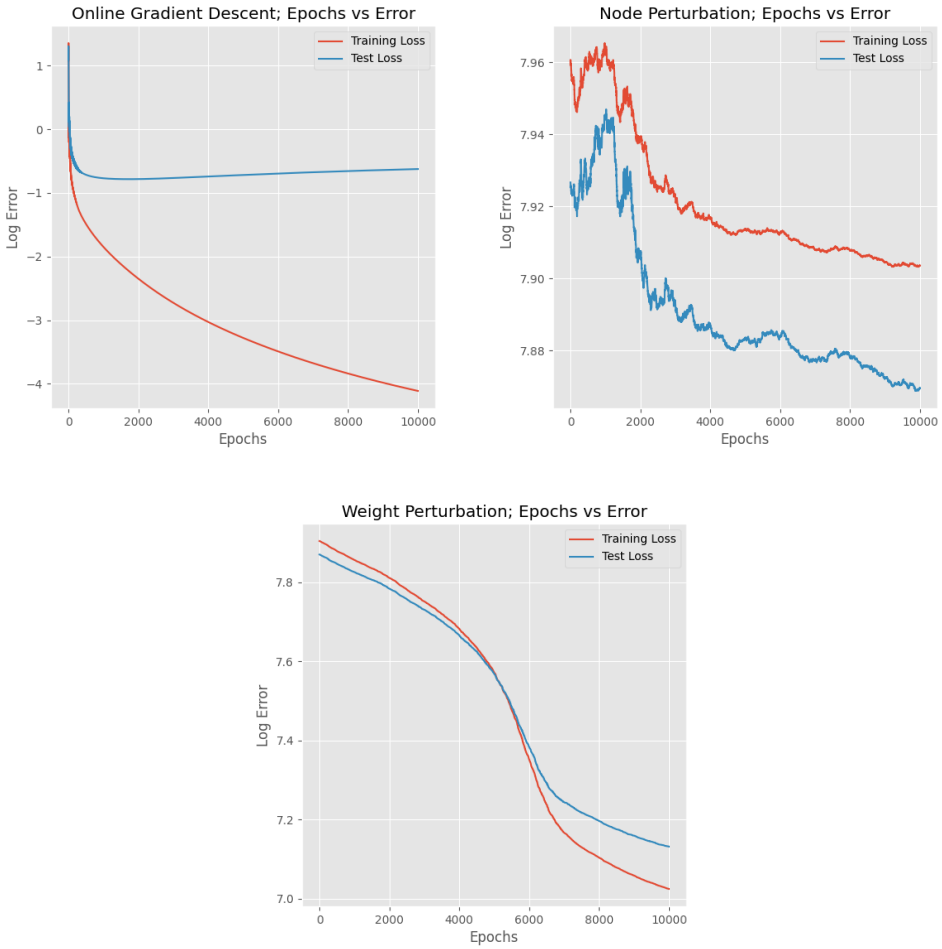

Figure 3: Discuss the results from mnist here

[2] 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=X0CKM7QV5k`.

[3] 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.

[4] 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.

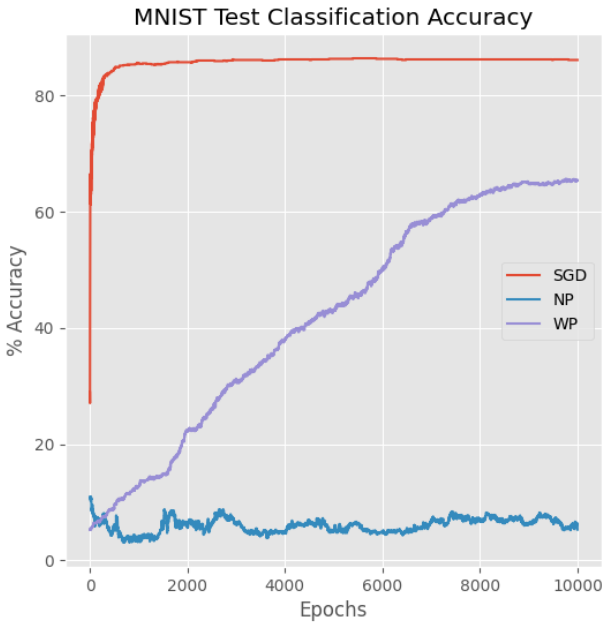

Figure 4: Discuss the results from mnist here