An Alternative to Backpropagation in Reinforcement Learning

Stephen Chung
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
minghaychung@umass.edu

Abstract

State-of-the-art deep learning algorithms mostly rely on gradient backpropagation to train a deep artificial neural network, which is generally regarded to be biologically implausible. For a network of stochastic units trained on a reinforcement learning task, one biologically plausible way of learning is to treat each unit as an agent and train all units by REINFORCE. In this case, only a global reward signal has to be broadcast to all units, and the learning rule given is local, which can be interpreted as reward-modulated spike-timing-dependent plasticity (R-STDP) that is observed biologically. Although this learning rule follows the gradient of return in expectation, it suffers from high variance and cannot be used to train a deep network in practice. In this paper, we propose an algorithm called MAP propagation that can reduce this variance significantly while retaining the local property of the learning rule. We show that the newly proposed algorithm can solve common reinforcement learning tasks at a similar speed to that of backpropagation when applied to an actor-critic network. We further show that MAP propagation can be implemented asynchronously, making the algorithm more biologically plausible than backpropagation.

1 Introduction

Gradient backpropagation [1] efficiently computes the gradient of an objective function with respect to parameters by iterating backward from the last layer of a multi-layer artificial neural network. However, backpropagation is generally regarded as being biologically implausible [2–6]. First, the learning rule given by backpropagation is non-local, since it relies on information other than input and output of a neuron-like unit during feed-forward computation, while synaptic plasticity observed biologically depends mostly on local information (e.g. spike-timing-dependent plasticity (STDP) [7]) and possibly some global signals (e.g. reward-modulated spike-timing-dependent plasticity (R-STDP) [7–9]). To address this issue, different local learning rules have been proposed. Energy-based models (e.g. the continuous Hopfield model [10]) can be trained by algorithms that are based on a local learning rule (e.g. contrastive Hebbian learning [11], contrastive divergence [12], equilibrium propagation [13]). Local learning rules have also been proposed for hierarchical auto-encoder networks [5]. These proposed algorithms apply to undirected networks in unsupervised learning tasks. A second reason backpropagation is considered biologically implausible is that learning by backpropagation requires separate feedforward and backpropagation phases, and it is unclear how a biological system can synchronize an entire network to precisely alternate between these two phases.

For a reinforcement learning task, a multi-layer artificial neural network can be trained efficiently by training the output unit by REINFORCE [14] while all hidden units are trained by backpropagation. In the case of hidden units being stochastic and continuous, we can still train the hidden units by

Preprint. Under review.

backpropagation using the re-parameterization trick [15]. Another possible way of training a network of stochastic units is to apply REINFORCE to all units, and it is shown that the learning rule is still an unbiased estimator of the gradient of return [14]. Another interpretation of training each unit by REINFORCE is to treat each unit as an reinforcement learning agent, with each of them trying to maximize the global reward. Such a network of reinforcement learning agents is called a coagent network, and [16–18] introduced relevant theories on coagent networks. The coagent network's idea goes back as far as [19], which trained a network of A_{r-p} units to solve simple reinforcement learning tasks. However, coagent networks can only solve simple tasks due to the high variance of the learning rule. The idea of coagent networks is also related to Klopf's hedonistic neuron hypothesis [20, 21].

REINFORCE, when applied on a Bernoulli-logistic unit, gives a three-factor learning rule which depends on a reinforcement signal, input and output of the unit. This is similar to R-STDP observed biologically, which depends on a neuromodulatory input, presynaptic spikes and postsynaptic spikes. It has been proposed that dopamine, a neurotransmitter, plays the role of neuromodulatory input in R-STDP and corresponds to the TD error concept from reinforcement learning. We refer readers to Chapter 15 of [22] for a review and discussion of the connection between REINFORCE and neuroscience.

Though REINFORCE is simple and biologically plausible, if it is used to train all units in a coagent network, it will suffer from high variance making it impractical to use in practice for all but the smallest networks. To address the high variance, we propose a new algorithm that significantly reduces the variance of learning rule when training all units by REINFORCE. We call the newly proposed algorithm MAP propagation. We show how it can apply to an actor-critic network with eligibility trace, where both the actor and critic network are trained with similar local learning rules modulated by global signals. Our experiments show that MAP propagation can solve common reinforcement learning tasks at a similar speed to that of backpropagation when applied to an actor-critic network.

We further propose an asynchronous version of MAP propagation, where all hidden units are trained asynchronously without the need for alternating between the feedforward and backpropagation phases as in backpropagation. This makes MAP propagation able to solve the two major problems of backpropagation: non-local and phase synchronization. MAP propagation provides a new class of reinforcement learning algorithms that do not require backpropagation and is more biologically plausible than backpropagation while maintaining a comparable learning speed to backpropagation. Our work also opens the prospect of the broader application of coagent networks in deep reinforcement learning.

2 Background and Notation

We will consider a Markov Decision Process (MDP). A MDP is a tuple given by $(\mathcal{S}, \mathcal{A}, P, R, \gamma, d_0)$, where $\mathcal{S}$ is the finite set of possible states of the environment (but this work extends to MDPs where state set is infinite) and the state at time t is denoted by S_t which is in $\mathcal{S}$. $\mathcal{A}$ is the finite set of possible actions and the action at time t is denoted by A_t which is in $\mathcal{A}$. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, is the transition function which describes the dynamics of the environment and is defined to be $P(s, a, s') := \Pr(S_{t+1} = s' | S_t = s, A_t = a)$, which has to be a valid probability mass function. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, defined to be $R(s, a) := E[R_t | S_t = s, A_t = a]$, and R_t is the reward at time t which is in $\mathbb{R}$. $\gamma \in [0, 1]$ is the discount factor. $d_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution, defined to be $d_0(s) := \Pr(S_0 = s)$ and has to be a valid probability mass function.

We are interested in finding a policy function $\pi : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ such that if action is sampled from policy function, that is, if $A_t | S_t \sim \pi(S_t, \cdot)$, then the expected return $E[G_t | \pi]$ is maximized, where return is defined to be $G_t := \sum_{s=t}^{\infty} \gamma^{s-t} R_s$. The value function for policy π is defined to be $V^\pi(s) := E[G_t | S_t = s, \pi]$.

In this work, we only consider a multi-layer network of stochastic units as the policy function. Let L be the number of layers in the network, including output layer. Denote H_t^l as the value of hidden units of layer l at time t. For simplicity we also define $H_t^0 := S_t$ and $H_t^L := A_t$, and $H_t := \{H_t^1, H_t^2, ..., H_t^{L-1}\}$ as sets of all hidden units. The conditional distribution of H_t^l is given by function $\pi_l(H_t^{l-1}, H_t^l; W^l) := \Pr(H_t^l | H_t^{l-1}; W^l)$, where W^l is the parameter for computing distribution of layer l. To sample action from the network, we iteratively samples $H_t^l \sim \pi_l(H_t^{l-1}, \cdot; W^l)$

from $l = 1$ to L. A common choice of π_l is $\pi_l(H_t^{l-1}, \cdot; W^l) = N(g(W^l H_t^{l-1}), \sigma^2)$ where g is the non-linear activation function such as softplus or rectified linear unit (ReLU).

3 Algorithm

3.1 MAP Propagation

Firstly, the gradient of return with respect to W^l (where $l \in \{1, 2, ..., L\}$) can be estimated by REINFORCE, which is also known as likelihood ratio estimator:

$$\nabla_{W^l} \mathbb{E}[G_t] = \mathbb{E}[G_t \nabla_{W^l} \log \Pr(A_t|S_t; W)] \tag{1}$$

We can show that (1) also equals $\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1}; W^l)]$, which is the REINFORCE update rule applied to each hidden unit with the same global reinforcement signal G_t:

Theorem 1. *Under the multi-layer network of stochastic units described above, we have, for $l \in \{1, 2, ..., L\}$:*

$$\mathbb{E}[G_t \nabla_{W^l} \log Pr(A_t|S_t; W)] = \mathbb{E}[G_t \nabla_{W^l} \log Pr(H_t^l|H_t^{l-1}; W^l)] \tag{2}$$

The proof is in Appendix A.1. Note that this theorem is also proved in [14], but we prove it here again using a different method. This shows that we can apply REINFORCE to each unit of the network, and the learning rule is still an unbiased estimator of the gradient of the return. Therefore, denoting α as the step size, we can learn the parameters of the network by:

$$W^l \leftarrow W^l + \alpha G_t \nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1}; W^l) \tag{3}$$

However, this learning rule suffers from high variance. It can be shown that (see the Appendix for the details):

$$\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})] = \mathbb{E}[G_t \, \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})|S_t, A_t]] \tag{4}$$

To reduce the variance, we can therefore replace $\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})$ in learning rule (3) by $\mathbb{E}[\nabla_{W^l} \log \Pr(H^l|H^{l-1})|S_t, A_t]$, such that we obtain the following learning rule:

$$W^l \leftarrow W^l + \alpha G_t \, \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})|S_t, A_t] \tag{5}$$

Since (3) is following gradient of return in expectation, and the expected update value of (3) and (5) is the same, we conclude that (5) is also a valid learning rule in the sense that it follows gradient of return in expectation. However, we note that $\mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})|S_t, A_t]$ in (5) is generally intractable to compute[1]. Instead, we propose to use maximum a posteriori (MAP) estimate to approximate this term:

$$\begin{aligned} &\mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})|S_t, A_t] \approx \nabla_{W^l} \log \Pr(\hat{H}_t^l|\hat{H}_t^{l-1}) \\ &\text{where } \hat{H}_t = \operatorname*{argmax}_{h_t} \Pr(H_t = h_t|S_t, A_t) \end{aligned} \tag{6}$$

Denote $\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})$ as D_t^l and we are interested in estimation of $\mathbb{E}[D_t^l|S_t, A_t]$. Essentially, REINFORCE uses a single sample of D_t^l to estimate $\mathbb{E}[D_t^l|S_t, A_t]$, while we propose to use the mode of D_t^l conditioned on S_t, A_t to estimate $\mathbb{E}[D_t^l|S_t, A_t]$. Though the former estimate is unbiased, it suffers from a high variance such that learning is extremely slow. The latter estimate is biased but has no variance.

Since $\hat{H}_t$ is intractable to compute analytically, we can approximate it by running gradient ascent on $\log \Pr(H_t|S_t, A_t)$ as a function of H_t for fixed S_t and A_t, with H_t being initialized as the sample

[1] We have tested various methods such as rejection sampling and importance sampling to estimate this term, but experiment results show that the speed of it is still far behind backpropagation.

sampled from the network when sampling action A_t, such that H_t approaches $\hat{H}_t$. This assumes all hidden units to be continuous. Therefore, before applying learning rule (3), we first run gradient ascent on H for N steps:

$$H_t \leftarrow H_t + \alpha \nabla_{H_t} \log \Pr(H_t | S_t, A_t) \tag{7}$$

For $l = 1, 2, ..., L-1$, we can see that this is equivalent to (see the Appendix for the details):

$$H_t^l \leftarrow H_t^l + \alpha(\nabla_{H_t^l} \log \Pr(H_t^{l+1} | H_t^l) + \nabla_{H_t^l} \log \Pr(H_t^l | H_t^{l-1})) \tag{8}$$

Therefore, the update rule to H_t^l is local: it only depends on H_t^{l-1} and H_t^{l+1} and not other layers. The update rule is maximizing the probability of the value of a hidden unit given the value of units one layer below and above by updating the value of that hidden unit.

In the case of hidden units being normally distributed ($\pi_l(H_t^{l-1}, \cdot; W^l) = N(g(W^l H_t^{l-1}), \sigma^2)$), (8) becomes:

$$H_t^l \leftarrow H_t^l + \frac{\alpha}{\sigma^2}(g(W^l H_t^{l-1}) - H_t^l + ((H_t^{l+1} - g(W^{l+1} H_t^l)) \odot g'(W^{l+1} H_t^l))(W^{l+1})^T) \tag{9}$$

where $\odot$ denotes the elemental-wise multiplication. We can observe that the feedback weight $(W^{l+1})^T$ has to be symmetric of the feed-forward weight. Though symmetric weight is not observed in biological neurons, it has been proposed that the brain may have learning rules to calibrate this reciprocity such that the symmetric weight holds at the level of cortical columns [23, 24].

We can also add noise to H_t during each update such that it becomes stochastic gradient ascent. After updating H_t for N steps by (8), we can apply (3) to learn the weight of network. We call this algorithm MAP propagation and the pseudo-code can be found in Algorithm 1.

Similar to actor-critic networks [22], we can also train a critic network to predict $V^\pi(S_t)$, so G_t in (3) can be replaced by TD error $\delta_t = R_t + \gamma V^\pi(S_{t+1}) - V^\pi(S_t)$ and the whole algorithm can be implemented online. To better facilitate temporal credit assignment, we can also use eligibility traces to replace the gradient in (3), using the same idea of actor-critic networks with eligibility trace [22]. The pseudo-code of it can be found in Algorithm 2. How a critic network can be trained by MAP propagation is explained in Section 3.3.

3.2 Asynchronous MAP propagation

In Algorithm 2, there are four separate phases: feedforward phase, REINFORCE phase, MAP gradient ascent phase, and trace update phase. A global phase-wise synchronization across all layers is required, which makes MAP propagation biologically implausible. In the following, we consider merging all phases together to remove the need for phase synchronization.

In the feedforward phase, we sample action A_t by iteratively samples $H_t^l \sim \pi_l(H_t^{l-1}, \cdot; W^l)$ from $l = 1$ to L. Alternatively, we can sample action A_t by sampling from the stationary distribution of a Markov chain defined by the following dynamic:

Theorem 2. *The stationary distribution of the Markov chain $\mathcal{T}$ is $P(A_t | S_t = s)$, where $\mathcal{T}$ is defined by the following transition function (denote k as the time step in the Markov chain):*

$$\forall a, a', Pr(A_{t,k+1} = a' | A_{t,k} = a) := \mathbb{E}[\pi_L(H_t^{L-1}, a'; W^L) | S_t = s, A_t = a] \tag{10}$$

The proof is in Appendix A.2. Assuming π_L is always non-zero (which can be easily satisfied, such as using softmax function as the probability distribution of the output layer), then this Markov chain always converges to the stationary distribution $P(A_t | S_t = s)$. So an alternative method to sample action A_t is to run this Markov Chain sufficiently long and sample A_t from it.

However, we note that $\mathbb{E}[\pi_L(H_t^{L-1}, a'; W^L) | S_t = s, A_t = a]$ is intractable to compute. Similar to the previous section, we propose to approximate it by MAP estimation:

$$\mathbb{E}[\pi_L(H_t^{L-1}, a'; W^L) | S_t = s, A_t = a] \approx \pi_L(\hat{H}_t^{L-1}, a'; W^L) \tag{11}$$
$$\text{where } \hat{H}_t = \underset{h_t}{\operatorname{argmax}} \Pr(H_t = h_t | S_t = s, A_t = a)$$

We can again estimate $\hat{H}_t$ by doing gradient ascent on the probability using (8). Therefore, to take one step in the Markov chain $\mathcal{T}$ starting from $A_{t,k} = a$, we first do N' steps of gradient ascent using (8) to estimate $H_{t,N'} \approx \hat{H}_t$, then we sample the next value $A_{t,k+1}$ from $\pi_L(H_{t,N'}^{L-1}, \cdot; W^L)$.

Using this sampling method, we can observe that the update rules to hidden units in both feedforward and MAP gradient ascent phases use the same update formula (8). The only difference between the two phases is that the value of the output unit is repeatedly sampled from $\pi_L(H_t^{L-1}, \cdot; W^L)$ in the feedforward phase, while the output unit is clamped to the chosen action value in the MAP gradient ascent phase. Thus, this removes the need for hidden units to separate between the feedforward phase and the MAP gradient ascent phase.

Then we consider how to make the trace update phase asynchronous. Ideally, we update the trace when the MAP gradient ascent phase has just finished. But if we assume there is no phase synchronization, the hidden units in a network cannot differentiate between the feedforward phase and the MAP gradient ascent phase. However, we notice that during the feedforward phase, the value of hidden units will keep changing as the value of the output unit is being repeatedly sampled, and during the MAP gradient ascent phase, the value of hidden units will also keep changing before reaching the equilibrium point to which the MAP gradient ascent converge. Therefore, we propose to update the trace scaled by the inverse square of the momentum of hidden units' value, such that the trace accumulated mostly corresponds to the trace after the hidden units' value reached the equilibrium point:

$$\mathbf{g^l} \leftarrow \beta \mathbf{g^l} + (1-\beta)\Delta H^l \tag{12}$$

$$\mathbf{z}^l \leftarrow \gamma\lambda \mathbf{z}^l + \frac{1}{\epsilon + \alpha(\mathbf{g^l} \odot \mathbf{g^l})} \odot \nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l) \tag{13}$$

where $\mathbf{z^l}$ denotes the eligibility trace for layer l, $\mathbf{g^l}$ denotes the momentum for layer l, ΔH^l denotes the change in value of H^l. β, α, and ϵ are hyperparameters to control how the momentum is computed and used to scale the trace.

Finally, the REINFORCE phase occurs when a TD error is received. To sum up, the network's hidden units repeatedly perform the following steps without any phase synchronization: 1. Update the value H^l by (8), 2. Update the trace $\mathbf{z}^l$ and the momentum by (12) and (13), 3. Upon receiving TD error, update the weight by $W^l \leftarrow W^l + \alpha\delta z^l$, where δ denotes the TD error. The pseudo-code of it can be found in Algorithm 3.

3.3 MAP Propagation for Critic Networks

Following, we consider how to apply the idea to a critic network. As the function of a critic network can be seen as approximating the scalar value $R_t + \gamma\hat{v}(S_{t+1})$, we consider how to learn a scalar regression task by MAP propagation in general.

A scalar regression task can be converted into a single-time-step MDP with the appropriate reward and $\mathbb{R}$ as the action set. For example, we can set the reward function to be $R(S, A) = -(A - A^*(S))^2$ (we dropped the subscript t as it only has a single time step), with A being the output of the network and $A^*(S)$ being the target output given input S. Therefore, the maximization of reward in the MDP minimizes the L2 distance between the predicted value and target value.

But this conversion is inefficient since the information of the optimal output is lost. Should A increase or decrease given the value of R? To address this issue, we suggest to use a baseline $b(S) = R(S, \mathbb{E}[A|S])$. Therefore, we have $-(A - A^*(S))^2 + (\mathbb{E}[A|S] - A^*(S))^2$ as the adjusted reward, and we obtain the following result:

Theorem 3. *Under the multi-layer network of stochastic units described above, we have, for $l \in \{1, 2, ..., L\}$:*

$$\begin{aligned}\nabla_{W^l} \mathbb{E}[-(A - A^*(S))^2] =& 2\,\mathbb{E}[(A^*(S) - \mathbb{E}[A|S])(A - \mathbb{E}[A|S])\nabla_{W^l} \log Pr(H^l|H^{l-1}; W^l)] \\ &- \nabla_{W^l} \mathbb{E}[(A - \mathbb{E}[A|S])^2]\end{aligned} \tag{14}$$

The proof is in Appendix A.3. We note that the last term, $-\nabla_{W^l} \mathbb{E}[(A - \mathbb{E}[A|S])^2]$, encourages the output of the network to have small variance, which can be removed if we consider adding the same

term as regularization for motivating exploration. Therefore, we can use the following learning rule for the task:

$$W^l \leftarrow W^l + \alpha(A^*(S) - \mathbb{E}[A|S])(A - \mathbb{E}[A|S])\nabla_{W^l} \log \Pr(H^l|H^{l-1}; W^l) \tag{15}$$

However, $\mathbb{E}[A|S]$ is generally intractable to compute exactly. We can use Monte Carlo estimation by running the network a few times, but this is inefficient. Instead, we suggest to use $\bar{A} := \mathbb{E}[A|H^{L-1}]$, the mean value of the action given value of the last hidden layer, to estimate this term. H^{L-1} here is obtained during sampling action A, before running update rule (8). $\bar{A}$ can usually be computed directly from H^{L-1}. For example, when the output unit's distribution is $N(H^{L-1}W^L, \sigma^2)$, we have $\bar{A} = H^{L-1}W^L$.

Though the use of $\bar{A}$ makes the baseline no longer valid for all but last layers since it depends on H^{L-1}, our experiment shows that the result is no worse than Monte Carlo estimation when combined with MAP propagation. Therefore, we suggest the following learning rule for a scalar regression task:

$$W^l \leftarrow W^l + \alpha(A^*(S) - \bar{A})(A - \bar{A})\nabla_{W^l} \log \Pr(H^l|H^{l-1}; W^l) \tag{16}$$

The pseudo-code is the same as the one in Algorithm 1 but with G in line 11 replaced by $(A^*(S) - \bar{A})(A - \bar{A})$.

We then consider applying the above method to train a critic network, where the action of the network, $A \in \mathbb{R}$, is an estimation of the current value. In a critic network, the target output is $R_t + \gamma A_{t+1}$. However, a more stable estimate of target output is $R_t + \gamma \bar{A}_{t+1}$. For example, when the output unit's distribution is $N(H_t^{L-1}W^L, \sigma^2)$, the difference between A_t and $\bar{A}_t$ is an independent Gaussian noise that can be removed. Therefore, we chose $A^*(S)$, the target output, as $R_t + \gamma \bar{A}_{t+1}$ and TD error as $\delta_t := \gamma \bar{A}_{t+1} + R_t - \bar{A}_t$. Substituting back into (16), the update rule for the critic network is:

$$W^l \leftarrow W^l + \alpha\delta_t(A_t - \bar{A}_t)\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1}; W^l) \tag{17}$$

which is almost the same as the update rule for the actor network except the additional term $(A_t - \bar{A}_t)$. The pseudo-code of training a critic network with eligibility trace using (Asynchronous) MAP propagation is the same as Algorithm 2 (3), except (i) Line 13 (28) is replaced with $\delta \leftarrow \gamma\bar{A} + R - \bar{A}'$ where $\bar{A} = \mathbb{E}[A|H^{L-1}]$ and $\bar{A}'$ is $\bar{A}$ in the previous time step, and (ii) the gradient term in line 19 (9, 24 and 34) is multipled by $(A - \bar{A})$.

Both the critic and the actor network can be trained together, and the TD error δ computed by the critic network can be passed to the actor network to facilitate temporal credit assignment.

4 Related Works

A major inspiration for our work is [5], which proposed training a deep generative model by using MAP estimate to infer the value of latent variables given observed variables in unsupervised learning tasks. They also proposed to learn the feedback weights such that the constraint of symmetric weight can be removed. This idea can possibly be applied to our algorithm as well to remove the requirement of symmetric weight given the similarity of the inference process. [25] explained the relationship of the inference of latent variables in energy-based models with backpropagation. [26] proposes training a network on supervised learning tasks with local learning rule based on MAP inference.

We can also view a multi-level network of stochastic units as a hierarchy of agents. In hierarchical reinforcement learning, [27] proposed learning a multi-level hierarchy with the use of hindsight action, which replaces the chosen action of the high-level agent with a more likely action instead. This is similar to our idea of replacing the action of all actors in hidden layers with the most probable action.

There is a large amount of literature on methods for training a network of stochastic units. A comprehensive review can be found in [15], which includes re-parametrization trick [28] and REINFORCE [14]. They introduce methods to reduce the variance of update rules such as baseline and critic. These ideas are orthogonal to the use of MAP estimate to reduce the variance of REINFORCE. [19, 16–18] introduce theories relating to training coagent networks.

[24, 29, 30] introduce biologically plausible learning rules based on reward prediction errors and attentional feedback. However, similar to backpropagation, these learning rules require a non-local feedback signal (besides the global reward signal) and global phase-wise synchronization, which is not required in MAP propagation.

5 Experiments

To test the algorithm, we first consider the multiplexer task [19], which can be considered as a single-time-step MDP. This is to test the performance of the algorithm as an actor. Then we consider a scalar regression task to test the performance of the algorithm as a critic. Finally, we consider some standard reinforcement learning tasks to test the performance of the algorithm as both actor and critic.

In all the below tasks, all networks considered have the same architecture: a two-hidden-layer network, with the first hidden layer having 64 units, second hidden layer having 32 units and output layer having one unit. All hidden units are normally distributed with mean given by a non-linear function g on linear transformation of previous layer and a fixed variance. That is, $P(H^l|H^{l-1};W^l) = N(g(H^{l-1}W^l),\sigma_l^2)$ for $l = 1, 2$. We chose $g(x) = \log(1 + \exp(x))$, which is the softplus function. For the network in multiplexer task and the actor network with discrete output, the output unit's distribution is given by softmax function on previous layer. That is, $P(A = a|H^{L-1};W^L) = \text{softmax}_a(TH^{L-1}W^L)$, where T is a scalar hyperparameter representing the temperature. For the network in scalar regression task, the critic network and the actor network with continuous output, the output unit's distribution is normally distributed with mean given by linear transformation of previous layer and a fixed variance. That is, $P(A|H^{L-1};W^L) = N(H^{L-1}W^L,\sigma_L^2)$. We used $N = 20$ in MAP propagation. Other hyper-parameters and details of experiments can be found in the Appendix.

5.1 Multiplexer Task

We consider the task of k-bit multiplexer here (similar to the one in [19] which is a 2-bit multiplexer) which can be considered as a single-time-step MDP. In a multiplexer, there are $k + 2^k$ binary inputs, where the first k bits represents the address and last 2^k bits represents the data, each of which is associated with an address. The desired output of the multiplexer is given by the value of data associated with the address, which is also binary. The action set is $\{-1, 1\}$ and we give a reward of 1 if the action of agent is the desired output and -1 otherwise. We consider $k = 5$ here so the dimension of state is 37.

We trained each network using 2,000,000 training samples and a batch size of 128 with Adam optimizer [31]. Batch here means that we compute weight update for each sample in a batch, then we update the weight using the average of these weight updates. Adam optimizer is applied by treating the weight update as gradient and is not applied to the update of hidden value in MAP propagation.

We used Algorithm 1 for MAP propagation. We consider two baselines: 1. Backpropagation, where the output unit is trained with REINFORCE and all hidden units are trained with backpropagation using the re-parametrization trick; 2. REINFORCE, where both output and hidden units are trained with REINFORCE.

The results is shown in Fig 1. We can observe that MAP propagation performs the best, followed closely by backpropagation. For REINFORCE, the learning is very slow due to inefficient structural credit assignment. The result demonstrates that MAP propagation can improve the learning speed of REINFORCE significantly by reducing the variance of weight update, such that its learning speed is comparable to (or better than) that of backpropagation.

5.2 Scalar Regression Task

Following we consider a scalar regression task which can be considered as a single-time-step MDP. The dimension of input is 8 and follows the standard normal distribution. The target output is computed by a one-hidden-layer network with weights chosen randomly. The action set is $\mathbb{R}$ and the reward is given by negative L2 distance between action and target output.

Similar to the previous task, we trained each network using 100,000 training samples and a batch size of 16 with Adam optimizer. For MAP propagation, we used Algorithm 1 and tested two variants. In

the first variant, we directly used negative L2 loss as the reward, which is labeled as 'MAP Prop (RL)'. In the second variant, we used (16) as the update rule in Algorithm 1, which is labeled as 'MAP Prop'. For the baseline, we trained the network with backpropagation using the re-parametrization trick by doing gradient descent on L2 loss.

The results is shown in Fig 1. We observe that if we directly use negative L2 loss as the reward, then the learning speed of MAP propagation is significantly lower than that of backpropagation since the information of optimal action is not incorporated. However, if we instead used (16) to incorporate the information of optimal action, then the learning speed of MAP propagation is comparable to that of backpropagation. The result demonstrates that MAP propagation can be used to train a network on scalar regression tasks efficiently.

5.3 Reinforcement Learning Task

Following we consider standard reinforcement learning tasks: Acrobot, CartPole, LunarLander, and continuous MountainCar in OpenAI's Gym [32][2]. The former three tasks have a discrete action set, while the last task has a continuous action set.

For MAP propagation, we used the actor-critic network with eligibility traces given by Algorithm 2 and 3. We did not use any batch update or any forms of regularization.

For the baseline, we used actor-critic with eligibility traces (episodic) from [22]. The model of baseline has the same architecture as the one in MAP propagation but with all hidden units being deterministic since we found the performance of deterministic units better than that of stochastic units. We trained the entire network with backpropagation and did not use any batch update. We employed entropy regularization [34] to encourage exploration. We did not include REINFORCE as a baseline since it showed almost no signs of learning in these tasks.

The average return over ten independent runs is shown in Fig 2. Let $\bar{G}$ denote the average return of all episodes. The mean and standard deviation of $\bar{G}$ over the ten runs can be found in Table 2.

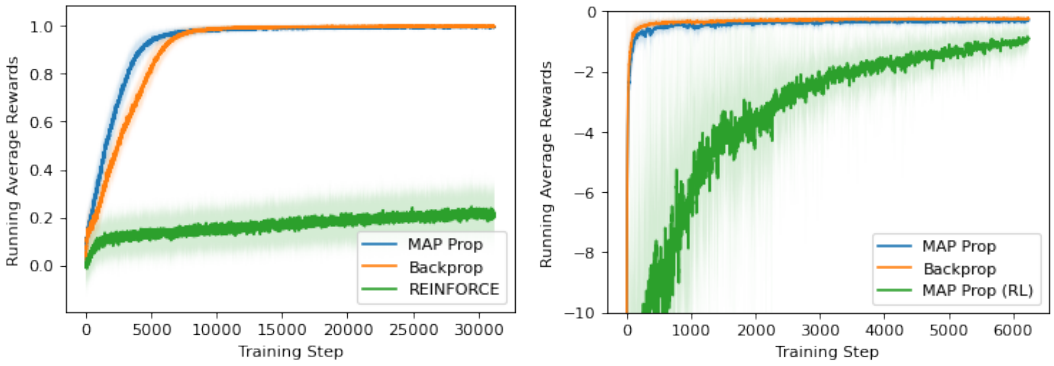
For all tasks, we observe that MAP propagation has a better performance than the baseline in term of the average return $\bar{G}$. The asynchronous MAP propagation also has comparable performance to the baseline. Its average return is lower than the baseline in LundarLander but higher than the baseline in the other tasks. This demonstrates that MAP Propagation can train a coagent network efficiently such that its learning speed is comparable to a standard reinforcement learning algorithm.

Although there are other algorithms, such as value-based methods, that can solve reinforcement learning tasks more efficiently, the present work aims to demonstrate that MAP propagation can be used to train a multi-layer actor-critic network at a comparable speed to backpropagation. This points to the possibility of using MAP propagation to replace backpropagation in training a multi-layer network in algorithms besides actor-critic networks. For example, MAP propagation can also be applied to other variants on actor-critic networks such as Proximal Policy Optimization [33].

We also notice that for baseline models, entropy regularization [34] is necessary for the agent to solve some tasks, else the agent will get stuck in early episodes. However, we have not employed any forms of regularization in MAP propagation. This may indicate that stochastic hidden units combined with MAP propagation can encourage more sophisticated exploration such that entropy regularization is unnecessary for the agent to solve the task. This is also a major motivation of coagent networks, given its hierarchical nature in selecting actions. Additional supporting evidence that MAP propagation has a more sophisticated exploration is given by its performance in MountainCar, a task where an agent can easily be stuck in the local optima. For the baseline model, we found that agents in most of the runs are stuck in local optima. In contrast, for both MAP propagation and asynchronous MAP propagation, agents in all runs can learn a policy that reaches the goal successfully. A detailed analysis of this can be found in the Appendix.

In the experiments above, we plotted the running average of returns against the number of the episode. However, it should be noted that the computation required for MAP propagation and backpropagation in each step is different since the time complexity of each step in MAP propagation is $O(N)$. This makes training of MAP Propagation take a much longer time than backpropagation. However, one

[2]We used Acrobot-v1, CartPole-v1, LunarLander-v2, and MountainCarContinuous-v0 in OpenAI's Gym for the implementation of the tasks.



(a) Multiplexer

(b) Scalar Regression

Figure 1: Running average rewards over last 10 episodes in multiplexer task and scalar regression task. Results are averaged over 10 independent runs, and shaded area represents standard deviation over the runs.

Table 1: Average return over all episodes.

	Acrobot		CartPole		LunarLander		MountainCar	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
MAP Propagation	-119.47	18.50	**464.23**	7.51	**101.81**	35.05	**61.90**	9.76
Asy. MAP Propagation	**-106.71**	17.43	418.61	29.15	76.84	15.90	46.24	13.77
Baseline	-143.09	11.62	373.54	48.28	90.26	16.37	8.55	26.11

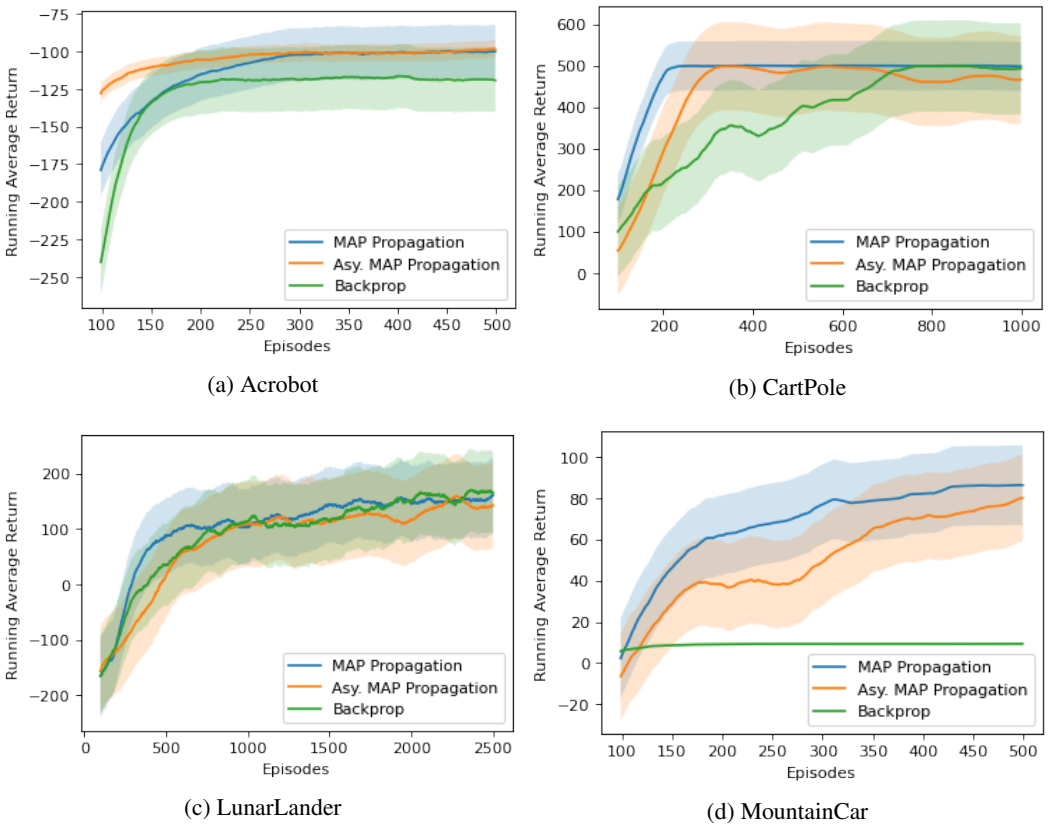
advantage of MAP propagation is that the update rule of (8) can be implemented asynchronously, while backpropagation requires all the upper layers to finish computing the gradient before updating the next layer.

Another limitation of MAP propagation is that it is sensitive to a large number of hyperparameters. For example, we found that the variance has to be larger, and the learning rate has to be smaller for units in deeper layers. This is in contrast to the baseline, which only has a few hyperparameters and does not require separate hyperparameters per layer. Further works can be done on investigating the choices of hyperparameters in MAP propagation.

6 Conclusion

In this paper, we propose a new algorithm that significantly reduces the variance of the update step when training all units in a network by REINFORCE. Many theoretical works have been done on coagent networks [16–18], but so far, there are no works tackling the extremely low learning speed of coagent networks, a critical problem that makes it inapplicable in practice. Our work presents a biologically plausible solution to this problem and opens the prospect of the broader application of coagent networks in deep reinforcement learning. Our experiments also show that MAP propagation can prevent local optima better compared to backpropagation by encouraging more sophisticated exploration.

In addition, the newly proposed algorithm solves two major problems of backpropagation regarding biological plausibility: the requirement of non-local feedback signals and the global phase-wise synchronization, which makes our algorithm more biologically plausible compared to backpropagation. Our experiments show that it can solve reinforcement learning tasks at a similar speed to that of backpropagation. This points to the possibility of learning a deeper network for more complex tasks such as Atari games without backpropagation. However, the algorithm has two major limitations compared to backpropagation: the difficulty in tuning hyperparameters and the large computational resources required.



(a) Acrobot

(b) CartPole

(c) LunarLander

(d) MountainCar

Figure 2: Running average returns over the last 100 episodes in Acrobot, CartPole, LunarLander and MountainCar. Results are averaged over 10 independent runs, and shaded area represents standard deviation over the runs.

The newly proposed algorithm may also provide insights into how learning occurs in biological systems. The idea that each neuron in a network learns independently by R-STDP, which is closely related to REINFORCE, is elegant and biologically plausible. However, although such learning follows gradient ascent on return in expectation, it will result in a very inefficient structural credit assignment since a single global signal is responsible for assigning credits to millions of neurons in the network. In MAP propagation, we solve this issue by having all hidden units settled to equilibrium while fixing the value of output units, which allows for more efficient structural credit assignment. Given a large number of feedback connections in biological systems, this may be one of the methods that biological systems employ to facilitate structural credit assignment. For example, [35] shows that the activity of neurons in the visual system that is responsible for the selected action will be enhanced by feedback connection, which is analogous to the updates of hidden units in MAP propagation.

One unsatisfying property of the proposed algorithm is that it relies on symmetric feedback weight, which is not observed biologically. However, this issue may be solved by learning the feedback weight separately by predicting values of the lower layer given values of the upper layer, as is done in [5]. Though symmetric weight is not observed in biological neurons, it has been proposed that the brain may have learning rules to calibrate this reciprocity such that the symmetric weight holds at the level of cortical columns [23, 24].

How can we build an asynchronous neuron-like unit such that when they are connected together, intelligent behavior automatically arises without the need for any global synchronization and intricate feedback connections? Our work presents one possible solution to this problem. This may help bridge the gap between machine learning and biologically learning, leading to a better understanding of the learning mechanism in the brain.

7 Acknowledgment

We thank Andy Barto who inspired this research and provided valuable insight and comments.

Algorithm 1: MAP Propagation: Monte-Carlo Policy-Gradient Control

Input: differentiable policy function: $\pi_l(H^{l-1}, H^l; W^l)$ for $l \in \{1, 2, ..., L\}$;
Algorithm Parameter: step size $\alpha > 0$, $\alpha_h > 0$; update step number $N \geq 1$;
discount rate $\gamma \in [0, 1]$;
Initialize policy parameter: W^l for $l := 1, 2, ..., L$;
Loop forever (for each episode):
  Generate an episode $S_0, H_0, A_0, R_1, ..., S_{T-1}, H_{T-1}, A_{T-1}, R_T$ following $\pi_l(\cdot, \cdot; W^l)$ for $l \in \{1, 2, ..., L\}$;
  Loop for each step of the episode $t = 0, 1, ..., T-1$**:**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    `/* MAP Gradient Ascent */`
    for $n := 1, 2, ..., N$ **do**
      $H_t^l \leftarrow H_t^l + \alpha_h(\nabla_{H_t^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l) + \nabla_{H_t^l} \log \pi_{l+1}(H_t^l, H_t^{l+1}; W^{l+1}))$ for $l := 1, 2, ..., L-1$;
    end
    `/* Apply REINFORCE */`
    $W^l \leftarrow W^l + \alpha G \nabla_{W^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l)$ for $l := 1, 2, ..., L$;

Algorithm 2: MAP Propagation: Actor Network with Eligibility Trace

Input: differentiable policy function: $\pi_l(H^{l-1}, H^l; W^l)$ for $l \in \{1, 2, ..., L\}$;
Algorithm Parameter: step size $\alpha > 0$, $\alpha_h > 0$; update step number $N \geq 1$;
trace decay rate $\lambda \in [0, 1]$; discount rate $\gamma \in [0, 1]$;
Initialize policy parameter: W^l for $l := 1, 2, ..., L$;
Loop forever (for each episode):
  Initialize S (first state of episode) ;
  Initialize zero eligibility trace $\mathbf{z^l}$ for $l := 1, 2, ..., L$;
  Loop while S **is not terminal (for each time step):**
    $H^0 \leftarrow S$;
    `/* 1. Feedforward Phase: Sampling actions */`
    Sample H^l from $\pi_l(H^{l-1}, \cdot; W^l)$ for $l := 1, 2, ..., L$;
    $A \leftarrow H^L$;
    `/* 2. REINFORCE Phase: Learning from TD error */`
    if episode not in first time step then
      Receive TD error δ from the critic network;
      $W^l \leftarrow W^l + \alpha \delta \mathbf{z^l}$ for $l := 1, 2, ..., L$;
    end
    `/* 3. MAP Gradient Ascent Phase: Gradient ascent on prob. */`
    for $n := 1, 2, ..., N$ **do**
      $H^l \leftarrow H^l + \alpha_h(\nabla_{H^l} \log \pi_l(H^{l-1}, H^l; W^l) + \nabla_{H^l} \log \pi_{l+1}(H^l, H^{l+1}; W^{l+1}))$ for $l := 1, 2, ..., L-1$;
    end
    `/* 4. Trace update phase: Accumulate eligibility trace */`
    $\mathbf{z}^l \leftarrow \gamma \lambda \mathbf{z}^l + \nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l)$ for $l := 1, 2, ..., L$;
    Take action A, observe S, R ;

Algorithm 3: Asynchronous MAP Propagation: Actor Network with Eligibility Trace

Input: differentiable policy function: $\pi_l(H^{l-1}, H^l; W^l)$ for $l \in \{1, 2, ..., L\}$;
Algorithm Parameter: step size $\alpha > 0$, $\alpha_h > 0$; update step number $N \geq 1$;
sample number $N, N' \geq 1$; trace decay rate $\lambda \in [0, 1]$; discount rate $\gamma \in [0, 1]$;
Initialize policy parameter: W^l for $l \in \{1, 2, ..., L\}$;
PROCEDURE: HIDDEN_UNITS_UPDATE(l)
    $\Delta H^l \leftarrow \alpha_h(\nabla_{H^l} \log \pi_l(H^{l-1}, H^l; W^l) + \nabla_{H^l} \log \pi_{l+1}(H^l, H^{l+1}; W^{l+1}))$;
    $H^l \leftarrow H^l + \Delta H^l$;
    $\mathbf{g^l} \leftarrow \beta \mathbf{g^l} + (1 - \beta)\Delta H^l$;
    $\mathbf{z}^l \leftarrow \gamma\lambda \mathbf{z}^l + \frac{1}{\epsilon + \alpha(\mathbf{g^l} \odot \mathbf{g^l})} \odot \nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l)$;
    if tde_queue[l] not empty **then**
        $\delta \leftarrow$ tde_queue[l].pop() ;
        $W^l \leftarrow W^l + \alpha\delta \mathbf{z^l}$;
    end
MAIN:
    Loop forever (for each episode):
        Initialize S (first state of episode) ;
        Initialize empty queue tde_queue[l] for $l \in \{1, 2, ..., L\}$;
        Initialize zero unit values H^l, zero momentum $\mathbf{g^l}$, and zero eligibility trace $\mathbf{z^l}$ for $l \in \{1, 2, ..., L\}$;
        Start threads HIDDEN_UNITS_UPDATE(l) for $l \in \{1, 2, ..., L-1\}$;
        Loop while S is not terminal (for each time step):
            /* Feedforward Phase: Repeatedly sample action */
            for $n := 1, 2, ..., N'$ **do**
                Sample A from $\pi_L(H^{L-1}, \cdot; W^L)$;
                $\mathbf{g^L} \leftarrow \beta \mathbf{g^L} + (1 - \beta)\Delta A$; // If A is discrete, then $\Delta A = +1$ if A has changed, else 0. If A is continuous, then ΔA is the change in value of A.
                $\mathbf{z}^L \leftarrow \gamma\lambda \mathbf{z}^L + \frac{1}{\epsilon + \alpha(\mathbf{g^L} \odot \mathbf{g^L})} \odot \nabla_{W^L} \log \pi_L(H^{L-1}, A; W^L)$;
                Wait for a short period of time;
            end
            /* REINFORCE Phase: Learning from TD error */
            if episode not in first time step then
                Receive TD error δ from the critic network;
                Push δ in tde_queue[l] for $l \in \{1, 2, ..., L-1\}$;
                $W^L \leftarrow W^L + \alpha\delta \mathbf{z^L}$;
            end
            /* MAP Gradient Ascent Phase: Clamped action */
            for $n := 1, 2, ..., N'$ **do**
                $\mathbf{g^L} \leftarrow \beta \mathbf{g^L}$;
                $\mathbf{z}^L \leftarrow \gamma\lambda \mathbf{z}^L + \frac{1}{\epsilon + \alpha(\mathbf{g^L} \odot \mathbf{g^L})} \odot \nabla_{W^L} \log \pi_L(H^{L-1}, A; W^L)$;
                Wait for a short period of time;
            end
            Take action A, observe S, R ;

A Proof

A.1 Proof of Theorem 1

$$\mathbb{E}[G_t \nabla_{W^l} \log \Pr(A_t|S_t;W)] \tag{18}$$

$$= \mathbb{E}[\frac{G_t}{\Pr(A_t|S_t;W)} \nabla_{W^l} \Pr(A_t|S_t;W)] \tag{19}$$

$$= \mathbb{E}[\frac{G_t}{\Pr(A_t|S_t;W)} \nabla_{W^l} \sum_{h_t} \pi_L(h_t^{L-1}, A_t; W^L)\pi_{L-1}(h_t^{L-2}, h_t^{L-1}; W^{L-1})...$$
$$\pi_2(h_t^1, h_t^2; W^2)\pi_1(S_t, h_t^1; W^1)] \tag{20}$$

$$= \mathbb{E}[\frac{G_t}{\Pr(A_t|S_t;W)} \sum_{h_t^{l+1}, h_t^l} \Pr(A_t, H_t^{l+1} = h_t^{l+1}, H_t^l = h_t^l|S_t)$$
$$\nabla_{W^l} \log \pi_l(h_t^{l-1}, h_t^l; W^l)] \tag{21}$$

$$= \mathbb{E}[G_t \sum_{h_t^{l-1}, h_t^l} \Pr(H_t^{l-1} = h_t^{l-1}, H_t^l = h_t^l|S_t, A_t)$$
$$\nabla_{W^l} \log \pi_l(h_t^{l-1}, h_t^l; W^l)] \tag{22}$$

$$= \mathbb{E}[G_t \, \mathbb{E}[\nabla_{W^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l)|S_t, A_t]] \tag{23}$$

$$= \mathbb{E}[\mathbb{E}[G_t|S_t, A_t] \, \mathbb{E}[\nabla_{W^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l)|S_t, A_t]] \tag{24}$$

$$= \mathbb{E}[\mathbb{E}[G_t \nabla_{W^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l)|S_t, A_t]] \tag{25}$$

$$= \mathbb{E}[G_t \nabla_{W^l} \log \pi_l(H_t^{l-1}, H_t^l; W^l)] \tag{26}$$

$$= \mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1}; W^l)] \tag{27}$$

(19) to (20) uses the fact that:

$$\Pr(A_t|S_t;W) = \sum_{h_t} \pi_L(h_t^{L-1}, A_t; W^L)\pi_{L-1}(h_t^{L-2}, h_t^{L-1}; W^{L-1})...\pi_2(h_t^1, h_t^2; W^2)\pi_1(S_t, h_t^1; W^1) \tag{28}$$

which marginalizes over all possible value of hidden units $H^1, H^2, ..., H^{L-1}$.
(20) to (21) uses the fact that:

$$\nabla_{W^l} \sum_{h_t} \pi_L(h_t^{L-1}, A_t; W^L)\pi_{L-1}(h_t^{L-2}, h_t^{L-1}; W^{L-1})...\pi_2(h_t^1, h_t^2; W^2)\pi_1(S_t, h_t^1; W^1) \tag{29}$$

$$= \sum_{h_t^{l+1}, h_t^l} \frac{\nabla_{W^l} \pi_l(h_t^{l-1}, h_t^l; W^l)}{\pi_l(h_t^{l-1}, h_t^l; W^l)} \sum_{h_t^1, ..., h_t^{l-1}, h_t^{l+2}, ..., h_t^{L-1}} \pi_L(h_t^{L-1}, A_t; W^L)$$
$$\pi_{L-1}(h_t^{L-2}, h_t^{L-1}; W^{L-1})...\pi_2(h_t^1, h_t^2; W^2)\pi_1(S_t, h_t^1; W^1) \tag{30}$$

$$= \sum_{h_t^{l+1}, h_t^l} \Pr(A_t, H_t^{l+1} = h_t^{l+1}, H_t^l = h_t^l|S_t) \nabla_{W^l} \log \pi_l(h_t^{l-1}, h_t^l; W^l) \tag{31}$$

For (30), we first move the term $\pi_l(h_t^{l-1}, h_t^l; W^l)$ out from inner summation since it is the only term that depends on W^l. Then we multiply back $\frac{\pi_l(h_t^{l-1}, h_t^l; W^l)}{\pi_l(h_t^{l-1}, h_t^l; W^l)}$ to get (30). (31) uses the fact that the inner summation in (30) is just $\Pr(A_t, H_t^{l+1} = h_t^{l+1}, H_t^l = h_t^l|S_t)$.
(21) to (22) uses the fact that $\frac{\Pr(A_t, H_t^{l+1} = h_t^{l+1}, H_t^l = h_t^l|S_t)}{\Pr(A_t|S_t)} = \Pr(H_t^{l+1} = h_t^{l+1}, H_t^l = h_t^l|S_t, A_t)$.
(22) to (23) uses the definition of expectation.
(23) to (24) uses the fact that, for any random variables Z and Y, $\mathbb{E}[\mathbb{E}[Z|Y]f(Y)] = \mathbb{E}[Zf(Y)]$. In our case, $Z = G_t$, $Y = (S_t, A_t)$ and $f(Y) = \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})|S_t, A_t]$.
(24) to (25) uses the fact that G_t is conditional independent with H_t^l, H_t^{l-1} given S_t, A_t.
(25) to (26) uses the law of total expectation: $\mathbb{E}_{X,Y}[f(X,Y)] = \mathbb{E}_Y[\mathbb{E}_X[f(X,Y)|Y]]$. In our case, $X = (G_t, H_t^l, H_t^{l-1})$, $Y = (S_t, A_t)$ and $f(X,Y) = G_t \nabla_{W^l} \log \Pr(H_t^l|H_t^{l-1})$.

A.2 Proof of Theorem 2

$$\sum_a \Pr(A_{t,k+1} = a' | A_{t,k} = a) \Pr(A_t = a | S_t = s) \tag{21}$$

$$= \sum_a \mathbb{E}[\pi_L(H_t^{L-1}, a'; W^L) | S_t = s, A_t = a] \Pr(A_t = a | S_t = s) \tag{22}$$

$$= \sum_a \int_h \pi_L(h, a'; W^L) \Pr(H^{L-1} = h | S_t = s, A_t = a) dh \Pr(A_t = a | S_t = s) \tag{23}$$

$$= \int_h \pi_L(h, a'; W^L) \sum_a \Pr(H^{L-1} = h | S_t = s, A_t = a) \Pr(A_t = a | S_t = s) dh \tag{24}$$

$$= \int_h \pi_L(h, a'; W^L) \sum_a \Pr(H^{L-1} = h, A_t = a | S_t = s) dh \tag{25}$$

$$= \int_h \pi_L(h, a'; W^L) \Pr(H^{L-1} = h | S_t = s) dh \tag{26}$$

$$= \int_h \Pr(A_t = a' | H^{L-1} = h, S_t = s) \Pr(H^{L-1} = h | S_t = s) dh \tag{27}$$

$$= \int_h \Pr(A_t = a', H^{L-1} = h | S_t = s) dh \tag{28}$$

$$= \Pr(A_t = a' | S_t = s) \tag{29}$$

A.3 Proof of Theorem 3

$$\nabla_{W^l} \mathbb{E}[-(A - A^*(S))^2] \tag{30}$$

$$= \mathbb{E}[-(A - A^*(S))^2 \nabla_{W^l} \log \Pr(A_t | S_t; W)] \tag{31}$$

$$= \mathbb{E}[-(A - A^*(S))^2 \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}; W^l)] \tag{32}$$

$$= \mathbb{E}[\left(-(A - A^*(S))^2 + (\mathbb{E}[A|S] - A^*(S))^2\right) \nabla_{W^l} \log \Pr(H^l | H^{l-1}; W^l)] \tag{33}$$

$$= \mathbb{E}[\left(-(A - \mathbb{E}[A|S] + \mathbb{E}[A|S] - A^*(S))^2 + (\mathbb{E}[A|S] - A^*(S))^2\right) \nabla_{W^l} \log \Pr(H^l | H^{l-1}; W^l)] \tag{34}$$

$$= \mathbb{E}[\left(2(A^*(S) - \mathbb{E}[A|S])(A - \mathbb{E}[A|S]) - (A - \mathbb{E}[A|S])^2\right) \nabla_{W^l} \log \Pr(H^l | H^{l-1}; W^l)] \tag{35}$$

$$= 2\,\mathbb{E}[(A^*(S) - \mathbb{E}[A|S])(A - \mathbb{E}[A|S]) \nabla_{W^l} \log \Pr(H^l | H^{l-1}; W^l)] - \nabla_{W^l} \mathbb{E}[(A - \mathbb{E}[A|S])^2] \tag{36}$$

(30) to (31) uses REINFORCE in (1).
(31) to (32) uses Theorem 1.
(32) to (33) uses $(\mathbb{E}[A|S] - A^*(S))^2$ as baseline which is a valid baseline since it does not depend on H or A.
(33) to (35) is simple algebra.
(35) to (36) converts $\mathbb{E}[(A - \mathbb{E}[A|S])^2 \nabla_{W^l} \log \Pr(H^l | H^{l-1}; W^l)]$ back to $\nabla_{W^l} \mathbb{E}[(A - \mathbb{E}[A|S])^2]$ by REINFORCE and Theorem 1.

$$\mathrm{Var}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1})] \tag{37}$$

$$= \mathbb{E}[\mathrm{Var}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] + \mathrm{Var}[\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] \tag{38}$$

$$= \mathbb{E}[G_t^2 \,\mathrm{Var}[\nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] + \mathrm{Var}[G_t \,\mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] \tag{39}$$

(37) to (38) uses the law of total variance: $\mathrm{Var}[g(X, Y)] = \mathbb{E}[\mathrm{Var}[g(X)|Y]] + \mathrm{Var}[\mathbb{E}[g(X)|Y]]$. In our case, $X = (G_t, H_t^l, H_t^{l-1})$, $g(X) = G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1})$ and $Y = (S_t, A_t)$.
(38) to (39) uses the assumption that G_t is deterministic function of S_t and A_t, so it can be pulled out from inner variance and expectation. Note that this assumption can be fulfilled if we use a learned q-value $q(s, a) := \mathbb{E}[G_t | S_t = s, A_t = a]$ to replace G_t in the learning rule.

A.4 Proof of (4)

$$\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1})] = \mathbb{E}[G_t \, \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]]$$

The detailed steps for (4) are:

$$\begin{aligned}
&\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1})] && (39)\\
=&\mathbb{E}[\mathbb{E}[G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] && (40)\\
=&\mathbb{E}[\mathbb{E}[G_t | S_t, A_t] \, \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] && (41)\\
=&\mathbb{E}[G_t \, \mathbb{E}[\nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1}) | S_t, A_t]] && (42)
\end{aligned}$$

(39) to (40) uses the law of total expectation: $\mathbb{E}_{X,Y}[f(X,Y)] = \mathbb{E}_Y[\mathbb{E}_X[f(X,Y)|Y]]$. In our case, $X = (G_t, H_t^l, H_t^{l-1})$, $Y = (S_t, A_t)$ and $f(X,Y) = G_t \nabla_{W^l} \log \Pr(H_t^l | H_t^{l-1})$.
(40) to (41) uses the fact that G_t is conditional independent with H_t^l, H_t^{l-1} given S_t, A_t.
(41) to (42) uses the fact that, for any random variable Z and Y, $\mathbb{E}[\mathbb{E}[Z|Y] f(Y)] = \mathbb{E}[Z f(Y)]$.

A.5 Details of (7) to (8)

From (7) to (8), we use the followings (recall we uses the notation $H_t^0 := S_t$ and $H_t^L := A_t$ for convenience):

$$\begin{aligned}
&\nabla_{H_t^l} \log \Pr(H_t | S_t, A_t) && (43)\\
=&\nabla_{H_t^l} \left(\log \Pr(H_t, A_t | S_t) - \log \Pr(A_t | S_t) \right) && (44)\\
=&\nabla_{H_t^l} \log \Pr(H_t, A_t | S_t) && (45)\\
=&\nabla_{H_t^l} \log \left(\prod_{i=0}^{L-1} \Pr(H_t^{i+1} | H_t^i) \right) && (46)\\
=&\nabla_{H_t^l} \left(\sum_{i=0}^{L-1} \log \Pr(H_t^{i+1} | H_t^i) \right) && (47)\\
=&\nabla_{H_t^l} \log \Pr(H_t^{l+1} | H_t^l) + \nabla_{H_t^l} \log \Pr(H_t^l | H_t^{l-1}) && (48)
\end{aligned}$$

B Details of Experiment

We used Algorithm 1 in the multiplexer and scalar regression experiment, and the hyperparameters can be found in Table 2. We used a different learning rate α for each layer of the network, and we denote the learning rate for the i^{th} layer to be α_i. We denote the variance of the Gaussian distribution for the i^{th} layer to be σ_i^2. For the update rule in line 12 of the pseudo-code, we used Adam optimizer [31] instead, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. These hyperparameters are selected based on manual tuning to optimize the learning curve. We did the same manual tuning for the baseline models.

We used Algorithm 2 and 3 in the experiment on reinforcement learning tasks, and the hyperparameters can be found in Table 3 and Table 4. We have not used any batch update in our experiment, and we used a discount rate of 0.98 for all tasks. The update size of hidden units, α_h, is selected to be 0.5 times the variance of the unit.

For the hyper-parameters in computing the eligibility trace for asynchronous MAP propagation, we used ϵ =1e-5, β =0.7 and α =1e3 in all experiments. For the purpose of consistency, we ran a single time of HIDDEN_UNITS_UPDATE(l) for all layers when the algorithm is at line 25 and line 35.

For the update rules in line 14 of Algorithm 2 and line 12 and 30 of Algorithm 3, we used Adam optimizer [31] instead, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Again, these hyperparameters are selected based on manual tuning to maximize the average return across all episodes. We did the same manual tuning for the baseline models.

We annealed the learning rate linearly such that the learning rate is reduced to $\frac{1}{10}$ of initial learning rate at 50000 and 100000 steps for CartPole and Acrobat respectively, and the learning rate remains

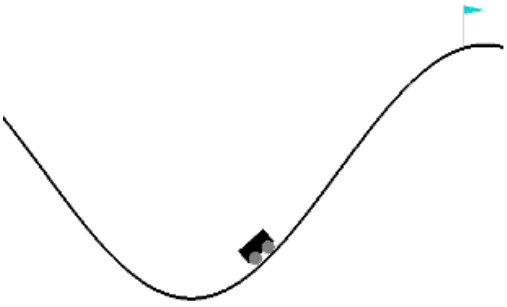


Figure 3: Illustration of MountainCar.

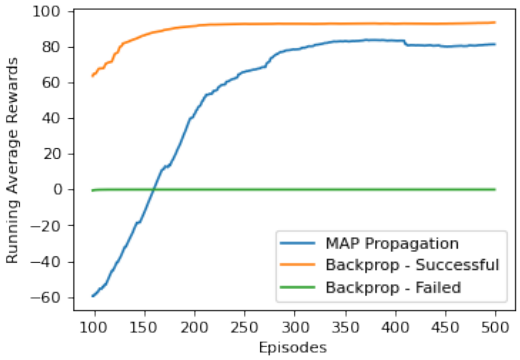


Figure 4: Running average rewards over the last 100 episodes in MountainCar of a sampled run of MAP propagation, a successful and a failed run of the baseline model.

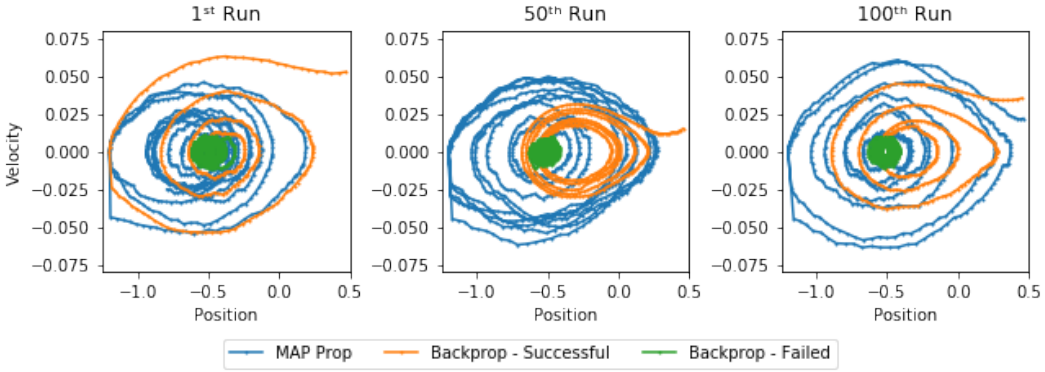


Figure 5: State trajectories on the 1st, 50th and 100th episode of the selected runs. If the position is larger than 0.45, then the agent reaches the goal. Although the agent trained by MAP propagation did not reach the goal in both the 1st and 50th run, the agent explored a large amount of state space, which is in contrast to the failed baseline that stays at the center. For the baseline model, if it does not reach the goal on the first run, it will quickly be stuck in the local optima.

unchanged afterward. We also annealed the learning rate linearly for the baseline model. We found that this can make the final performance more stable. For LunarLander and MountainCar, we did not anneal the learning rate. For MountainCar, we bound the reward by ± 5 to stabilize learning.

C Experiments on MountainCar

In the continuous version of MountainCar, the state is composed of two scalar values, which are the position and the velocity of the car, and the action is a scalar value corresponding to the force applied to the car. The goal is to reach the peak of the mountain on the right, where a large positive reward is given. However, to reach the peak, the agent has to first push the car to the left in order to gain enough momentum to overcome the slope. A small negative reward that is proportional to the force applied is given on every time step to encourage the agent to reach the goal with minimum force. The illustration of MountainCar is shown in Fig 3.

One locally optimum policy is to apply zero force at every time step so that the car stays at the bottom. In this way, the return is zero since no force is applied. In our experiment, we found that most runs of the actor critic model trained by backpropagation (the baseline model) got stuck in this locally optimum policy. However, in several runs, the baseline model can accidentally reach the goal in early episodes, which makes the agent able to learn quickly and achieve an asymptotic reward of over +90,

slightly higher than that of MAP propagation. The learning curve of a successful and a failed run of the baseline model is shown in Fig 4.

In contrast, for both MAP propagation and asynchronous MAP propagation, agents in all runs can learn a policy that reaches the goal successfully. However, this comes at the expense of a slower learning and a slightly worse asymptotically performance, as seen in Fig 4. This is likely due to the larger degree of exploration in MAP propagation, which can be illustrated by the state trajectories shown in Fig 5.

We used the same variance for the output unit in both MAP propagation and the baseline model. We found that even using a larger variance cannot prevent the baseline model from being stuck in local optima. This illustrates that MAP propagation allows more sophisticated exploration instead of merely more exploration. In MAP propagation, each unit in the network can be regarded as an agent exploring its own action space, thus allowing exploration in different levels of the hierarchy. This may explain the exploration behavior observed in MAP propagation.

From this experiment, we see that the learning dynamic of MAP propagation is very different from that of backpropagation. MAP propagation can prevent local optima better but at the expense of slower learning and worse asymptotic performance. This may be explained by the more sophisticated exploration of MAP propagation due to its exploration in different levels of the hierarchy.

Table 2: Hyperparameters used in the multiplexer and scalar regression experiment.

	Multiplexer	Scalar Regression
Batch Size	128	16
N	20	20
α_1	5e-2	5e-2
α_2	5e-4	5e-6
α_3	5e-6	5e-8
α_h	1	5e-2
σ_1^2	1	5e-2
σ_2^2	1	5e-2
σ_3^2	n.a.	5e-2
T	1	n.a.

Table 3: Hyperparameters used for MAP propagation in Acrobat, Cartpole, Lunarlander and MountainCar experiment.

	Acrobat		CartPole		LunarLander		MountainCar	
	Critic	Actor	Critic	Actor	Critic	Actor	Critic	Actor
N	20	20	20	20	20	20	20	20
α_1	2e-2	1e-2	2e-2	2e-2	1e-2	2e-3	1e-2	4e-3
α_2	2e-5	1e-5	2e-5	2e-5	1e-5	2e-6	1e-5	4e-6
α_3	2e-6	1e-6	2e-6	2e-6	1e-6	2e-7	1e-6	4e-7
σ_1^2	0.03	0.06	0.03	0.03	0.003	0.06	0.003	0.03
σ_2^2	0.1	0.2	0.1	0.1	0.01	0.2	0.01	0.1
σ_3^2	0.1	n.a.	0.1	n.a.	0.01	n.a.	0.05	0.5
T	n.a.	4	n.a.	2	n.a.	2	n.a.	n.a.
λ	.95	.95	.95	.95	.97	.97	.97	.97

Table 4: Hyperparameters used for asynchronous MAP propagation in Acrobat, Cartpole, Lunarlander and MountainCar experiment.

	Acrobat		CartPole		LunarLander		MountainCar	
	Critic	Actor	Critic	Actor	Critic	Actor	Critic	Actor
N, N'	20	20	20	20	20	20	20	20
α_1	2e-2	4e-3	2e-2	1e-2	1e-2	1e-3	1e-2	2e-3
α_2	2e-5	4e-6	2e-5	1e-5	1e-5	1e-6	1e-5	2e-6
α_3	2e-6	4e-7	2e-6	1e-6	1e-6	1e-7	1e-6	2e-7
σ_1^2	0.03	0.03	0.03	0.03	0.003	0.06	0.003	0.06
σ_2^2	0.1	0.1	0.1	0.1	0.01	0.2	0.01	0.2
σ_3^2	0.5	n.a.	0.5	n.a.	0.05	n.a.	0.05	1
T	n.a.	1	n.a.	2	n.a.	2	n.a.	n.a.
λ	.997	.997	.997	.997	.997	.997	.997	.997

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[2] Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

[3] Pietro Mazzoni, Richard A Andersen, and Michael I Jordan. A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, 88(10):4433–4437, 1991.

[4] Randall C O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938, 1996.

[5] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[6] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[7] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[8] Răzvan V Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural computation*, 19(6):1468–1502, 2007.

[9] Verena Pawlak, Jeffery R Wickens, Alfredo Kirkwood, and Jason ND Kerr. Timing is not everything: neuromodulation opens the stdp gate. *Frontiers in synaptic neuroscience*, 2:146, 2010.

[10] John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

[11] Javier R Movellan. Contrastive hebbian learning in the continuous hopfield model. In *Connectionist models*, pages 10–17. Elsevier, 1991.

[12] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[13] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[14] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[15] Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*, 2019.

[16] Philip S Thomas. Policy gradient coagent networks. In *Advances in Neural Information Processing Systems*, pages 1944–1952, 2011.

[17] James E Kostas, Chris Nota, and Philip S Thomas. Asynchronous coagent networks.

[18] Philip S Thomas and Andrew G Barto. Conjugate markov decision processes. In *ICML*, 2011.

[19] Andrew G Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4(4):229–256, 1985.

[20] A Harry Klopf. *Brain function and adaptive systems: a heterostatic theory*. Number 133. Air Force Cambridge Research Laboratories, Air Force Systems Command, United . . . , 1972.

[21] A Harry Klopf. *The hedonistic neuron: a theory of memory, learning, and intelligence*. Toxicology-Sci, 1982.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23] Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in neural information processing systems*, pages 976–984, 2019.

[24] Isabella Pozzi, Sander Bohte, and Pieter Roelfsema. Attention-gated brain propagation: How the brain can implement reward-based error backpropagation. *Advances in Neural Information Processing Systems*, 33, 2020.

[25] Yoshua Bengio and Asja Fischer. Early inference in energy-based models approximates back-propagation. *arXiv preprint arXiv:1510.02777*, 2015.

[26] James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5):1229–1262, 2017.

[27] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017.

[28] Diederik Kingma and Max Welling. Efficient gradient-based inference through transformations between bayes nets and neural nets. In *International Conference on Machine Learning*, pages 1782–1790, 2014.

[29] Pieter R Roelfsema and Arjen van Ooyen. Attention-gated reinforcement learning of internal representations for classification. *Neural computation*, 17(10):2176–2214, 2005.

[30] Jaldert O Rombouts, Sander M Bohte, and Pieter R Roelfsema. How attention can create synaptic tags for the learning of working memories in sequential tasks. *PLoS Comput Biol*, 11(3):e1004060, 2015.

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

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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

[34] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[35] Pieter R Roelfsema and Anthony Holtmaat. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience*, 19(3):166, 2018.