Faster Learning with a Team of Reinforcement Learning Agents

Stephen Chung
University of Massachusetts Amherst
minghaychung@umass.edu

Andrew G. Barto
University of Massachusetts Amherst
barto@cs.umass.edu

Abstract

Though backpropagation underlies nearly all deep learning algorithms, it is generally regarded as being biologically implausible. An alternative way of training an artificial neural network is through making each unit stochastic and treating each unit as a reinforcement learning agent, and thus the network is considered as a team of agents. As such, all units can learn via REINFORCE, a local learning rule modulated by a global reward signal that is more consistent with biologically observed forms of synaptic plasticity. However, this learning method suffers from high variance and thus the low speed of learning. The high variance stems from the lack of effective structural credit assignment. This paper reviews two recently proposed algorithms to facilitate structural credit assignment when all units learn via REINFORCE, namely *MAP Propagation* and *Weight Maximization*. In MAP Propagation an energy function of the network is minimized before applying REINFORCE, such that activities of hidden units are more consistent with the activities of output units. In Weight Maximization the global reward signal to each hidden unit is replaced with the change in the squared L^2 norm of the vector of the unit's outgoing weights, such that each hidden unit is trying to maximize the norm of its outgoing weights instead of the external reward. Experiments show that both algorithms can learn significantly faster than a network of units learning via REINFORCE, and have a comparable speed to backpropagation when applied in standard reinforcement learning tasks. In contrast to backpropagation, both algorithms retain certain biologically plausible properties of REINFORCE, such as having local learning rules and the ability to be computed asynchronously. Therefore these algorithms may offer insights for understanding possible mechanisms of structural credit assignment in biological neural systems.

Keywords: backpropagation, synaptic plasticity, structural credit assignment, multi-agent reinforcement learning, hierarchical reinforcement learning

Overview

The *error backpropagation algorithm* (backprop) 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 (ANN). However, backprop is generally regarded as being biologically implausible [1]. First, the learning rule given by backprop is non-local, as it relies on a feedback signal backpropagating from the downstream units, while biologically-observed synaptic plasticity depends mostly on local information (e.g. spike-timing-dependent plasticity (STDP) [2]) and possibly some global signals, e.g. reward-modulated spike-timing-dependent plasticity (R-STDP) [2]. Second, to compute the feedback signal, backprop requires synaptic symmetry in the forward and backward paths, which has not been observed in biological systems. Third, backprop cannot be implemented asynchronously across units since it requires the network to alternate precisely between the feedforward and feedback phase, and the feedback signal has to be backpropagated layer-by-layer. Nonetheless, recent work has shown that these issues may be overcome. For example, the feedback signal in backprop may be approximated locally by the difference in neural activities across time [3], and synaptic symmetry may not be necessary for backprop due to the 'feedback alignment' phenomenon [4], but asynchronous computation remains a major obstacle to biological plausibility.

Alternatively, each unit in an ANN can be made stochastic and learn via the REINFORCE learning rule [5], a special case of $A_{R-\lambda P}$ when $\lambda = 0$ [6]. This learning method has also been called 'node perturbation' [3], and is generally considered more biologically plausible than backprop. REINFORCE, when applied on Bernoulli-logistic units, gives a three-factor learning rule which depends on a reward signal in addition to a unit's input and output signals. This three-factor learning rule is closely related to R-STDP, which depends on pre-synaptic and post-synaptic activity plus a neuromodulatory input [7, Chapter 15]. Since the only non-local information required by the REINFORCE learning rule is the globally broadcasted reward signal, the three aforementioned issues of backprop regarding biological plausibility do not exist when all units learn via REINFORCE.

Another interpretation of training all units by REINFORCE relates to viewing each unit as a reinforcement learning (RL) agent, with each agent trying to maximize the same reward signal from the environment. We can thus view an ANN as a *team of agents* playing a cooperative game, a scenario where all agents receive the same reward; agents here refer to RL agents [7]. Such a team of agents is also known as *coagent network* [8]. The idea of solving a task by a team of agents has a long history, and we refer readers to [7, Chapter 15] for the related work. However, due to the weak correlation between the team's reward signal and the action of an agent in the team, this learning method is associated with high variance and thus the low speed of learning. The high variance stems from the lack of effective structural credit assignment.

This paper reviews two recently proposed algorithms, namely *MAP Propagation* [9] and *Weight Maximization* [10], that facilitate structural credit assignment when training all units by REINFORCE. MAP propagation replaces the hidden units' outputs with their maximum a posteriori (MAP) estimates conditioned on the state and the selected action, or equivalently, minimizes an energy function of the network, before applying REINFORCE. For a network of normally distributed units (that is, an ANN with i.i.d Gaussian noise being added to the activation value of each unit), by minimizing the energy function of the network, the parameter update given by REINFORCE and backprop with the reparametrization trick becomes the same, thus establishing a connection between REINFORCE and backprop. In Weight Maximization we replace the global reward signal to each hidden unit with the change in the squared L^2 norm of its *outgoing weight*, which is defined as the vector of the weights by which the unit's outputs influence other units in the network. With the replaced reward signals, each hidden unit in the network is trying to maximize the norm of its outgoing weight. We prove that Weight Maximization approximately follows the gradient of reward in expectation, showing that every hidden unit maximizing the norm of its outgoing weight also approximately maximizes the external reward.

Our experiments show that ANNs trained with MAP Propagation or Weight Maximization can learn much faster than REINFORCE, such that the learning speed is comparable to backprop on standard RL tasks[1]. The experimental results are shown in Fig 1. It is worth noting that both MAP Propagation and Weight Maximization only assume a scalar reward signal from the environment, so they are more suitable for RL tasks instead of supervised learning (SL) tasks. It remains to be investigated how to best incorporate the knowledge of optimal network outputs into both algorithms for competitive performance in SL tasks.

MAP Propagation and Weight Maximization retain certain properties relevant to biological plausibility of REINFORCE, but neither algorithms can be implemented asynchronously. Alternative versions of MAP Propagation and Weight Maximization that allow asynchronous computation have been proposed. In *asynchronous MAP Propagation* (to be published) the feedforward phase is merged with the energy minimization phase to remove the requirement of alternating between phases. In *Weight Maximization with traces* [10] eligibility traces are employed to remove the need of waiting for downstream units to finish updating. A comparison of different biologically inspired algorithms according to biological plausibility properties is summarized in Table 1. Nonetheless, other properties relevant to biological plausibility of these

[1]To be precise, REINFORCE here refers to the case of all units implementing the REINFORCE learning rule, and backprop here refers to the case of output units implementing the REINFORCE learning rule and hidden units implementing backprop.

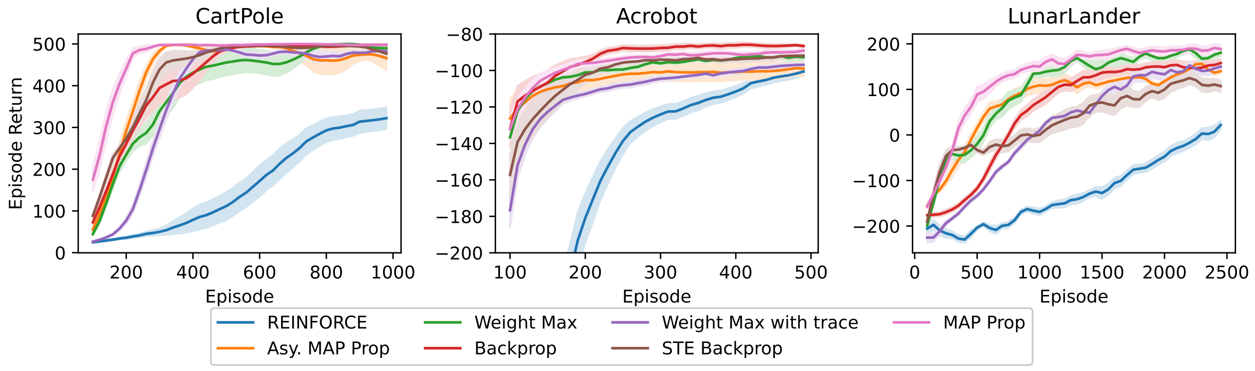

Figure 1: Episode returns in different RL tasks. Results are averaged over 10 independent runs, and shaded areas represent standard deviation over the runs. Curves are smoothed with a running average of 100 episodes. All networks have 64 and 32 units on the first and the second hidden layer respectively. REINFORCE, Weight Max and STE backprop use Bernoulli-logistic units; MAP Prop uses normally distributed units, and backprop uses Rectified Linear Units (ReLU). Hyper-parameters are selected based on manual tuning to optimize the average episode returns.

	Algorithm Type	Local learning rule	No symmetric feedback connections	Asynchronous computation across units
REINFORCE [5]	RL	✓	✓	✓
MAP Propagation [9]	RL	✓	✗	✗
Asy. MAP Propagation	RL	✓	✗	✓
Weight Max. [10]	RL	✓	✓	✗
Weight Max. with traces [10]	RL	✓	✓	✓
Backprop	SL	✗	✗	✗
Equilibrium Propagation [11]	SL	✓	✗	✗
Target Propagation [12]	SL	✓	✓	✗

Table 1: Comparison of biologically inspired algorithms on properties relevant to biological plausibility. Algorithm type: RL (Reinforcement Learning) - the algorithm only assumes the knowledge of a scalar reward signal from the environment; SL (Supervised Learning) - the algorithm assumes the knowledge of optimal network outputs from the environment. We call a learning rule local if it only depends on local variables (incoming weights, outgoing weights, activation values of the unit, incoming units, and outgoing units) and possibly a global scalar variable (e.g. reward signal in R-STDP). Backprop requires an error signal propagating from the outgoing units, so it is not local. Note that backprop can be combined with REINFORCE to train hidden units in RL tasks as in most deep RL algorithms.

algorithms remain to be investigated. For example, it is not yet clear if there exists a mechanism that allows changes in the efficacies of a neuron's outgoing synapses to influence changes in its incoming synapses.

From the computational perspective, MAP Propagation and Weight Maximization offer different advantages and disadvantages compared to backprop. Experiments showed that an ANN trained with MAP Propagation is less likely than bcakprop to become stuck in local optima in the MountainCar task [9], suggesting that stochastic activities of all units may lead to more effective exploration. However, each iteration of MAP Propagation is computationally more costly compared to each iteration of backprop due to the energy minimization phase. For Weight Maximization, experiments showed that it performs well when applied to discrete units such as Bernoulli-logistic units but performs poorly when applied on continuous units. Nonetheless, the ability to efficiently train discrete units and to localize optimization to each network unit may open the door to other RL methods for training ANNs.

By showing the possibility of efficient credit structural assignment, we hope that this work reinvigorates interest in training an ANN by REINFORCE alone, a biologically plausible method that has long been considered too slow for learning compared to backprop.

Algorithms

For simplicity, we consider an MDP with only immediate reward[2] defined by a tuple $(\mathcal{S}, \mathcal{A}, R, d_0)$, where $\mathcal{S}$ is the set of state, $\mathcal{A}$ is the set of action, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, and $d_0 : \mathcal{S} \rightarrow [0, 1]$ is the initial state distribution. Denoting the state, action, and reward by S, A, and R respectively, $\Pr(S = s) = d_0(s)$ and $\mathbb{E}[R|S = s, A = a] = R(s, a)$. We are interested in learning the policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ such that selecting actions according to $\Pr(A = a|S = s) = \pi(s, a)$ maximizes the expected reward $\mathbb{E}[R|\pi]$.

Here we restrict attention to policies computed by a multi-layer ANN consisting of L layers of stochastic units. Let H^l denote the vector of activation values of layer l. We also let $H^0 := S$ and $H^L := A$. For all $1 \leq l \leq L$, the distribution of H^l conditional on H^{l-1} is given by $\Pr(H^l_t = h^l | H^{l-1}_t = h^{l-1}; W^l) = \pi_l(h^{l-1}, h^l; W^l)$, where W^l is the parameter of layer l. We further assume[3] W^l is a matrix and $\pi_l(h^{l-1}, h^l; W^l) = \prod_i f(h^l_i, W^l_{:,i} h^{l-1})$ for some differentiable function f. To sample an action A from the network, we iteratively sample $H^l \sim \pi_l(H^{l-1}, \cdot; W^l)$ from $l = 1$ to L.

The gradient of reward with respect to W^l (where $l \in \{1, 2, ..., L\}$ in all discussion below unless stated otherwise) can be estimated by REINFORCE [5] :

$$\nabla_{W^l} \mathbb{E}[R|\pi] = \mathbb{E}[R \nabla_{W^l} \log \pi(S, A)|\pi] = \mathbb{E}[R \nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l)|\pi]. \tag{1}$$

Therefore, to perform gradient ascent on the reward, we can update parameters by adding $\alpha R \nabla_{W^l} \log \pi(S, A)$ to W^l, where α is the step size. $\nabla_{W^l} \log \pi(S, A)$ can be computed by backprop for deterministic and differentiable ANNs, or backprop with the reparametrization trick for some stochastic and continuous ANNs (e.g. a network of normally distributed units). For stochastic and discrete ANNs (e.g. a network of Bernoulli-logistic units), $\nabla_{W^l} \log \pi(S, A)$ cannot be computed by backprop or backprop with the reparametrization trick. But the second equality tells us that we can also update parameters by applying REINFORCE locally to all units:

$$W^l \leftarrow W^l + \alpha R \nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l). \tag{2}$$

This learning rule can applied in any stochastic ANNs. However, the parameter update in this learning rule has a large variance since a single reward R is used to evaluate the collective actions of all units. In the following we discuss two algorithms that help reduce this variance.

MAP Propagation - Let define an *energy function* by $E(h^1, h^2, ..., h^L; s) := -\log \Pr(H^1 = h^1, H^2 = h^2, ..., H^L = h^L | S = s)$. That is, the energy function measures the lack of compatibility between hidden units and output units. MAP Propagation minimizes the energy function of the network w.r.t. hidden units, before applying REINFORCE:

$$W^l \leftarrow W^l + \alpha R \nabla_{W^l} \log \pi_l(\hat{h}^{l-1}, \hat{h}^l; W^l), \tag{3}$$

where $\hat{h}^1, \hat{h}^2, ..., \hat{h}^{L-1} = \arg\min_{h^1, h^2, ..., h^{L-1}} E(h^1, h^2, ..., h^{L-1}, H^L; S)$ and $\hat{h}^L = H^L$. In other words, we try to make the activation values of hidden units to be more compatible with the selected action before applying REINFORCE. This energy minimization is equivalent to replacing the activation values of hidden units by their most probable activation values conditioned on the state and the selected action (i.e. MAP inference).

To intuitively understand how MAP Propagation improves learning, consider the following example. Suppose you are playing a computer game and there are two buttons (button A and B) on the controller, corresponding to action A and B. However, the controller is faulty and there is a 5% probability that the computer receives the opposite action. Suppose you pressed button A, and the computer game gave you a positive reward but showed that it received action B. Should you press button A or B more? You can learn in the long term by simply pressing the button associated with a positive reward more, which is button A in this case. But a more efficient way of learning is to press button B more since it is more likely that you have pressed button B. This example underlies the core idea of MAP Propagation - the most probable activation values of hidden units can replace their actual activation values in REINFORCE to speed up learning (see paper for detailed theoretical analysis).

The remaining question is how to minimize the energy function w.r.t. hidden units. For continuous-valued units, we can perform gradient descent on the energy function, and the update rule can be shown to be local. For discrete-valued units, we can use hill climbing methods or iteratively compute the minima for each unit.

Weight Maximization - Let define the *outgoing weight* of hidden unit i on layer l by the vector $W^{l+1}_{i,:}$ (the i^{th} row of matrix W^{l+1}), i.e. the weights connecting from that unit to units on the next layer. Weight Maximization uses a different

[2]The algorithms in this paper can be applied in general MDPs by replacing the reward with the sum of discounted reward (i.e. return) or TD error, and can be applied in supervised learning tasks by replacing the reward with the negative loss.

[3]This assumption is not necessary for MAP Propagation.

approach from MAP Propagation. In learning rule (2), all units receive the same reward signal R. It is natural to ask whether there exists a more specific signal that evaluates the activation value of each unit, such that a more precise structural credit assignment is possible. Inspired by this idea, Weight Maximization replaces the external reward signal R to each hidden unit by the change in the squared L^2 norm of its outgoing weight. The motivation of using the change in the norm of a unit's outgoing weight as a reward signal is based on the idea that the norm of a unit's outgoing weight roughly reflects the contribution of the unit in the network. For example, if the hidden unit is useful in guiding action, then the output unit will learn a large weight associated with it. Conversely, if the hidden unit is outputting random noise, then the output unit will learn a zero weight associated with it.

Assuming that the next layer is also learning, the outgoing weight $W^{l+1}_{i,:}$ changes on every step, and we denote this change (before multiplying the step size α) by $\Delta W^{l+1}_{i,:}$. Thus, the change in the squared L^2 norm of the outgoing weight can be expressed as ($\cdot$ denotes the dot product):

$$||W^{l+1}_{i,:} + \alpha \Delta W^{l+1}_{i,:}||_2^2 - ||W^{l+1}_{i,:}||_2^2 = 2\alpha \Delta W^{l+1}_{i,:} \cdot W^{l+1}_{i,:} + \mathcal{O}(\alpha^2). \tag{4}$$

We propose to ignore $\mathcal{O}(\alpha^2)$ since the step size is usually very small, and thus we define the reward signal to unit i on layer l by:

$$R^l_i := \begin{cases} 2\alpha \Delta W^{l+1}_{i,:} \cdot W^{l+1}_{i,:} & \text{for } l \in \{1, 2, ..., L-1\}, \\ R & \text{for } l = L. \end{cases} \tag{5}$$

The learning rule is the same as (2) but with the new reward signal ($W^l_{:,i}$ denotes the i^{th} column of matrix W^l):

$$W^l \leftarrow W^l + \alpha \Delta W^l, \text{where } \Delta W^l_{:,i} = R^l_i \nabla_{W^l_{:,i}} \log \pi_l(H^{l-1}, H^l; W^l). \tag{6}$$

(5) and (6) together defines the learning rule for all units in Weight Maximization. It can be proved that the update rule is still approximately following the gradient of rewards in expectation.

However, (5) and (6) require iterating backward from the top layer since units need to wait for the upper layer to finish learning to compute the reward R^l_i. To remove this iteration requirement, we consider the case where all units are learning at the same time step. Then, it can be seen that the reward signal to a unit on layer l is delayed by $L - l$ time steps as there are $L - l$ upper layers and each layer requires one time step to update. The problem of delayed reward is well studied in RL, and one prominent and biologically plausible solution is eligibility traces. By using eligibility traces, the algorithm can be implemented in parallel for all layers, and there are no distinct feedforward and feedback phases for the whole network (see paper for details of Weight Maximization with eligibility traces).

References

[1] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[2] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[3] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020.

[4] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, no. 1, pp. 1–10, 2016.

[5] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[6] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 360–375, 1985.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[8] P. S. Thomas, "Policy gradient coagent networks," in *Advances in Neural Information Processing Systems*, pp. 1944–1952, 2011.

[9] S. Chung, "Map propagation algorithm: Faster learning with a team of reinforcement learning agents," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[10] S. Chung, "Learning by competition of self-interested reinforcement learning agents," in *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022. To appear.

[11] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[12] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515, Springer, 2015.