Backpropagation-Free Parallel Deep Reinforcement Learning

William H. Guss
Machine Learning at Berkeley
2650 Durant Ave, Berkeley CA, 94720
`wguss@ml.berkeley.edu`

Mike Zhong
Machine Learning at Berkeley
Berkeley CA, 94720
`lol@gmail.com`

Utkarsh S
Machine Learning at Berkeley
Berkeley CA, 94720
`philkuz@ml.berkeley.edu`

Max Johansen
Machine Learning at Berkeley
Berkeley CA, 94720
`max@ml.berkeley.edu`

Abstract

In this paper we conjecture that an agent, envirionment pair (π, E) trained using DDPG with an actor network μ and critic network Q^π can be decomposed into a number of sub-agent, sub-environment pairs (π_n, E_n) ranging over every neuron in μ; that is, we show empircially that treating each neuron n as an agent $\pi_n : \mathbb{R}^n \to \mathbb{R}$ of its inputs and optimizing a value function Q^{π_n} with respect to the weights of π_n is dual to optimizing Q^π with respect to the weights of μ. Finally we propose a learning rule which simultaneously optimizes each π_n without error backpropogation achieving state of the art performance and speed across a variety of OpenAI Gym environments.

Todo list

1 Introduction

Introduction to DDPG and recent advances in deep RL.

Biological diffusion of dopamine in the brain $\implies$ error backpropagation is not biologically feasible.

Synthetic gradients are a step in the right direction, but still require eventual back propogation.

Therefore it is feasible that each neuron is maximizing the expectation on his future dopamine intake, and so we propose the following theorem.

2 Agent-Environment Value Decomposition

A high level description of the section.

2.1 Background

Recall the standard reinforcement learning setup. We say E is an *environment* if $E \overset{\text{def}}{=} (\mathcal{S}, \mathcal{A}, \mathcal{R}, T, r)$ where T describes transition probability measure $T\left(s_{t+1} \mid s_t, a_t\right)$ and $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is a reward function. Furthermore $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$ are the *state space, action space, and reward space* respectively. We restrict $\mathcal{R}$ to a compact subset of $\mathbb{R}$ and action space and state space to finite dimensional real vector spaces. As in DDPG we assume that the environment E is *fully observed*; that is, at any time step the state s_t is fully described by the observation presented, x_t, and not by the history $(x_1, a_1, \ldots, a_{t-1})$.

We define the policy for an agent to be $\pi : \mathcal{P}(\mathcal{A}) \times \mathcal{S} \to [0,1]$. In general the policy is a probability measure on some σ-algebra $\mathcal{M} \subset \mathcal{P}(\mathcal{A})$ conditioned on $\mathcal{S}$ so that $\pi(\mathcal{A} \mid s \in \mathcal{S}) = 1$. However, we will deal only with *deterministic* policies where for every s_t there is unique a_t so that $\pi(\{a_t\} \mid s = s_t) = 1$ and the measure is 0 otherwise. Thus we will abuse notation and define a *deterministic agent* by a policy function $\pi : \mathcal{S} \to \mathcal{A}$.

For a policy π the action-value function is the expected future reward under π by performing a_t at state s_t using the Bellman equation

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1}\sim E}\left[r(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))\right] \tag{2.1.1}$$

with $\gamma \in (0,1)$ a discount factor, and the second expectation removed because π is deterministic. [Some survey] provides an extensive exposition into a justification of this equation and choice for the action-value of π, so we will assume such a choice is a valid measure of performance.

Insert reference and make this a footnote

In deterministic policy gradient methods, we define an actor $\mu : \mathcal{S} \to \mathcal{A}$ and a critic $Q^{\mu} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and optimize $Q^{\mu}(s_t, \mu(s_t)$ with respect to the paramers θ^{μ} of μ. This method is provably the true policy gradient of μ if Q^{μ} is know. Recently (DDPG) utilizes the universality of DNNs in order to approximate both μ and Q^{μ} along with delayed weight-transfer networks to stabilize learning and prevent divergence as depicted in Figure 1. In order to decompose the action-value function we will make heavy use of this methodology at a scale local to each neuron in the flavor of (Synthetic gradients.)

Cite lilicrap

Make DDPG figure

Cite deepmind

2.2 Towards Neurocomputational Decomposition of Q^{μ}

In order to decompose the Q^{μ} algorithm we will abstractly define a neurocomputational agent in terms of an operator on voltages with no restrictions on the topology of the network, and then relate the action-value function of the whole agent to those which are dfined for each individual neuron in the network.

If $\mathcal{V}$ is an N-dimensional vector space then a *neurocomputational agent* is a tuple $\mathcal{N} = (\mu, \epsilon, \delta, K, \Theta, \sigma, V)$ such that:

- $\epsilon : \mathcal{S} \to N_I \subset \mathcal{V}$ encodes the state into the voltages of *input neurons*, a subspace N_I of the voltages $\mathcal{V} \subset \mathbb{R}^N$ of every neuron in the network. Specifically $\epsilon(s_t) = \mathrm{proj}_{N_I}(s_t)$.
- $\delta : \mathcal{V} \to \mathcal{A}$ decodes the voltages of the *output neurons* $N_O \subset V$ into an action.
- $K : \mathcal{V} \to \mathcal{V}$ is the linear voltage graph transition function of the graph representing the topology of $\mathcal{N}$, parameterized by θ.
- $\Theta : \mathcal{V} \to \mathcal{V}$ is a nonlinear inhibition function.
- $\sigma : \mathcal{V} \to \mathcal{V}$ is the elementwise application of some activation function to the voltage vector.
- $V : \mathbb{N} \to \mathcal{V}$ is the voltage of $\mathcal{N}$ at a discrete internal timestep τ such that

  $$V(\tau+1) \overset{\text{def}}{=} \sigma\left(K\Theta[V(\tau)]\right) + I(\tau), \qquad V(0) \overset{\text{def}}{=} 0. \tag{2.2.1}$$

  where $I(\tau)$ is some input function.
- $\mu : \mathcal{S} \to \mathcal{A}$ is the deterministic policy for $\mathcal{N}$. For some agents, the internal time τ is not in sync with t. For example if $\mathcal{N}$ standard ℓ layer DNN, then the policy decodes a voltage after an ℓ step delay; that is, if s_t observed at τ, then $\mu(s_t) = V(\tau + \ell)$, and $I(\tau) = \epsilon(s_t)$, but $I(\tau + n) = 0$ when $n \leqslant \ell$.

It is not hard to see that this definition encompasses any DQN or DDPG network with either reccurent or non recurrent layers. Additionally other paradigms such as the leaky integrattor are neurocomputational agents. (See appendix.)

Now let E and $\mathcal{N}$ be defined as above. If n is some neuron in $\mathcal{N}$, assume that the voltage of n is a function of $\mathcal{V}$.Then define the a deterministic *sub-environment* $E^n = (\mathcal{V}, \mathbb{R}, \mathcal{R}, T^n, r^n)$. In this case the transition describes how the voltage of n affects the voltage of $\mathcal{N}$ that is,

$$T\left(V(\tau+1) \mid V(\tau) \in \mathcal{V}, V(\tau+1)_n = a^n_{\tau} \in \mathbb{R}\right) = 1. \tag{2.2.2}$$

In other words the voltage of the neuron n at time step $\tau + 1$ is a function of $V(\tau)$ and the new state presented is $V(\tau + 1)$ such that the new voltage of n is a_τ^n. The reward function is $r^n(V(\tau + \ell), a_{\ell+\tau}) = r_t(s_t, \mu(s_t))$ if s_t presented at τ and $\mu(s_t)$ decodes $\mathcal{N}$ at time $\tau + \ell$, otherwise $r^n(v, a) = 0$.

Now define an *neuromorphically local agent* $\mu^n : \mathcal{V} \to \mathbb{R}$ so that $V(\tau) \mapsto \langle \sigma(K\Theta[V(\tau)]) + I(\tau), e_n \rangle$ where e_n is the basis vector for the n^{th} neuron's voltage. Immediately we can extend the action-value function to this sub-environment and agent,

$$Q^{\mu^n}(v_\tau, a_\tau^n) = \mathop{\mathbb{E}}_{V(\tau+1)\sim E^n}\left[r(v_\tau, a_\tau^n) + \gamma Q^{\mu^n}(V(\tau+1), \mu^n(V(\tau+1)))\right]. \tag{2.2.3}$$

Provided with the previous definitions, the following question arises: does deterministic policy gradient learning on $\mathcal{N}$, specifically μ on E, *commute* with performing the same operation simultaneously on every neuromorphically local agent μ^n comprising $\mathcal{N}$ and their respective sub-environments E^n? Supposing that we have the true Q^μ function and μ is optimal with respect to Q^μ, then it is intuitive, but not obvious, that every μ^n should behave optimally with respect to an infinite time horizion, but will reverse hold? In answer to this question, we propose the following conjecture.

IN PROGRESS: Write conjecture on decomposition which is free of neural configuration. Subject to change in later versions of ArXiv paper

Conjecture 2.2.1.

Emperical justification of the iff using the following experiment (s).

1. Training a network on Atari using DDPG and plotting average critic functions for neurons using window.

2. Possibly others.

Therefore we propose the following learning rule in aims to evidence the reverse, training μ using simultaneous optimization on all Q_n w.r.t π_n's weights.

3 Decentralized Deep Determinstic Policy Gradient Learning

Proposal of the rule. Linear approximation of the Q function for every neuron is good enough, (experimentally).

Implications of the rule to DDPG

Implications of the rule to entirely recurrent networks (infinite time horizion and NO unrolling since the environment the local actions of the neuron which globally recur to that neuron again are *encoded* into Q_n; large time horizion probably implies that better regresser needed for Q_n.)

Parallelism, no error backprop, and only 2x operations, but no locking on GPU, so all can be run sumultaneously if we cache!

4 Results

To validate the new learning rule we throw a fuck ton of experiments together on the following list (or better using OpenAI Gym).

```
blockworld1 1.156 1.511 0.466 1.299 -0.080 1.260
```

```
blockworld3da 0.340 0.705 0.889 2.225 -0.139 0.658
canada 0.303 1.735 0.176 0.688 0.125 1.157
canada2d 0.400 0.978 -0.285 0.119 -0.045 0.701
cart 0.938 1.336 1.096 1.258 0.343 1.216
cartpole 0.844 1.115 0.482 1.138 0.244 0.755
cartpoleBalance 0.951 1.000 0.335 0.996 -0.468 0.528
cartpoleParallelDouble 0.549 0.900 0.188 0.323 0.197 0.572
cartpoleSerialDouble 0.272 0.719 0.195 0.642 0.143 0.701
cartpoleSerialTriple 0.736 0.946 0.412 0.427 0.583 0.942
cheetah 0.903 1.206 0.457 0.792 -0.008 0.425
fixedReacher 0.849 1.021 0.693 0.981 0.259 0.927
fixedReacherDouble 0.924 0.996 0.872 0.943 0.290 0.995
fixedReacherSingle 0.954 1.000 0.827 0.995 0.620 0.999
gripper 0.655 0.972 0.406 0.790 0.461 0.816
gripperRandom 0.618 0.937 0.082 0.791 0.557 0.808
hardCheetah 1.311 1.990 1.204 1.431 -0.031 1.411
hopper 0.676 0.936 0.112 0.924 0.078 0.917
hyq 0.416 0.722 0.234 0.672 0.198 0.618
movingGripper 0.474 0.936 0.480 0.644 0.416 0.805
pendulum 0.946 1.021 0.663 1.055 0.099 0.951
reacher 0.720 0.987 0.194 0.878 0.231 0.953
reacher3daFixedTarget 0.585 0.943 0.453 0.922 0.204 0.631
reacher3daRandomTarget 0.467 0.739 0.374 0.735 -0.046 0.158
reacherSingle 0.981 1.102 1.000 1.083 1.010 1.083
walker2d 0.705 1.573 0.944 1.476 0.393 1.397
```

1. Show that training decentralized policy gradient $\implies$ total policy optimization

2. Show speed improvements on update step through parallelism (samples per second vs DDPG).

3. Show results are comparable with the state of the art.

5 Conclusion

We wrecked deep reinforcement learning using biological inspiration.

5.1 Future Work

Would like to try the method with full recurrent networks and purely asynchronous implementation of leaky integration networks.

Would like to prove the conjecture. List possible methods of proof.