PhD Course
Advance Topics in Reinforcement Learning

Policy Gradient Methods
Second Part

Mirco Musolesi

mircomusolesi@acm.org

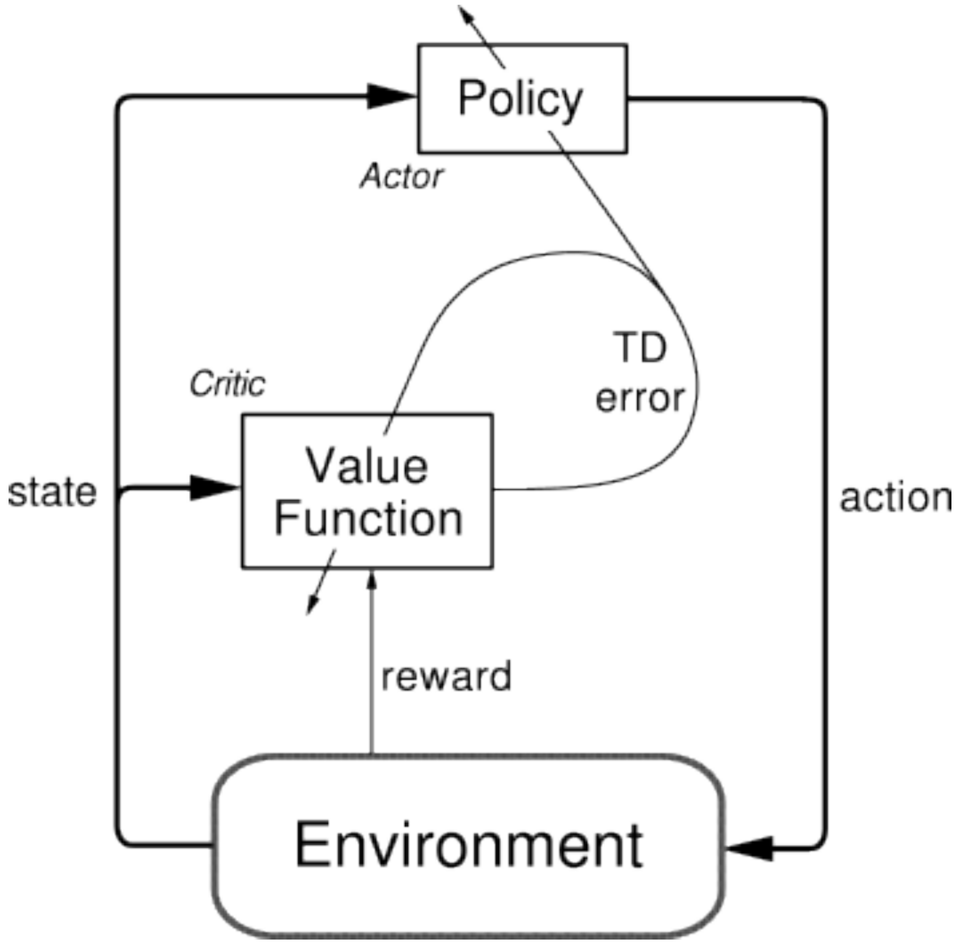
Actor-Critic Methods

- Methods that learn approximations to both policy and value functions are often called *actor-critic methods*, where *actor* is a reference to the learned policy and *critic* refers to the learned value function.
- We do not consider REINFORCE with baseline an *actor-critic method*, even if it learns both a policy and state-value functions.
- The reason is that its state-value function is used only as a baseline and not as a critic and not for updating the value estimate for a state from the estimated values of subsequent states (i.e., bootstrapping).
- This is an important distinction, since only through bootstrapping we introduce bias and an asymptotic dependence on the quality of the function approximation.
- Recall that the bias introduced through bootstrapping is often beneficial since it reduces variance and accelerates learning.

Actor-Critic Methods

- REINFORCE with baseline is unbiased and converges asymptotically to a local minimum, but as all the the other Monte Carlo methods tends to learn slowly (high variance in the estimates).
- We have seen that with temporal distance methods we can remove these problems.
- For this reason, we use actor-critic methods with a bootstrapping critic, i.e., we update the value estimate for a state from the estimated values of subsequent state.
- We consider one-step actor-critic methods.
  - These are analogs to TD(0), Sarsa and Q-learning.

Actor-Critic Methods



One-step Actor-Critic Method

- The one-step actor-critic method replaces the full return of REINFORCE with the one-step return (and use a learned state-value function as baseline) as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha(R_{t+1} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}$$

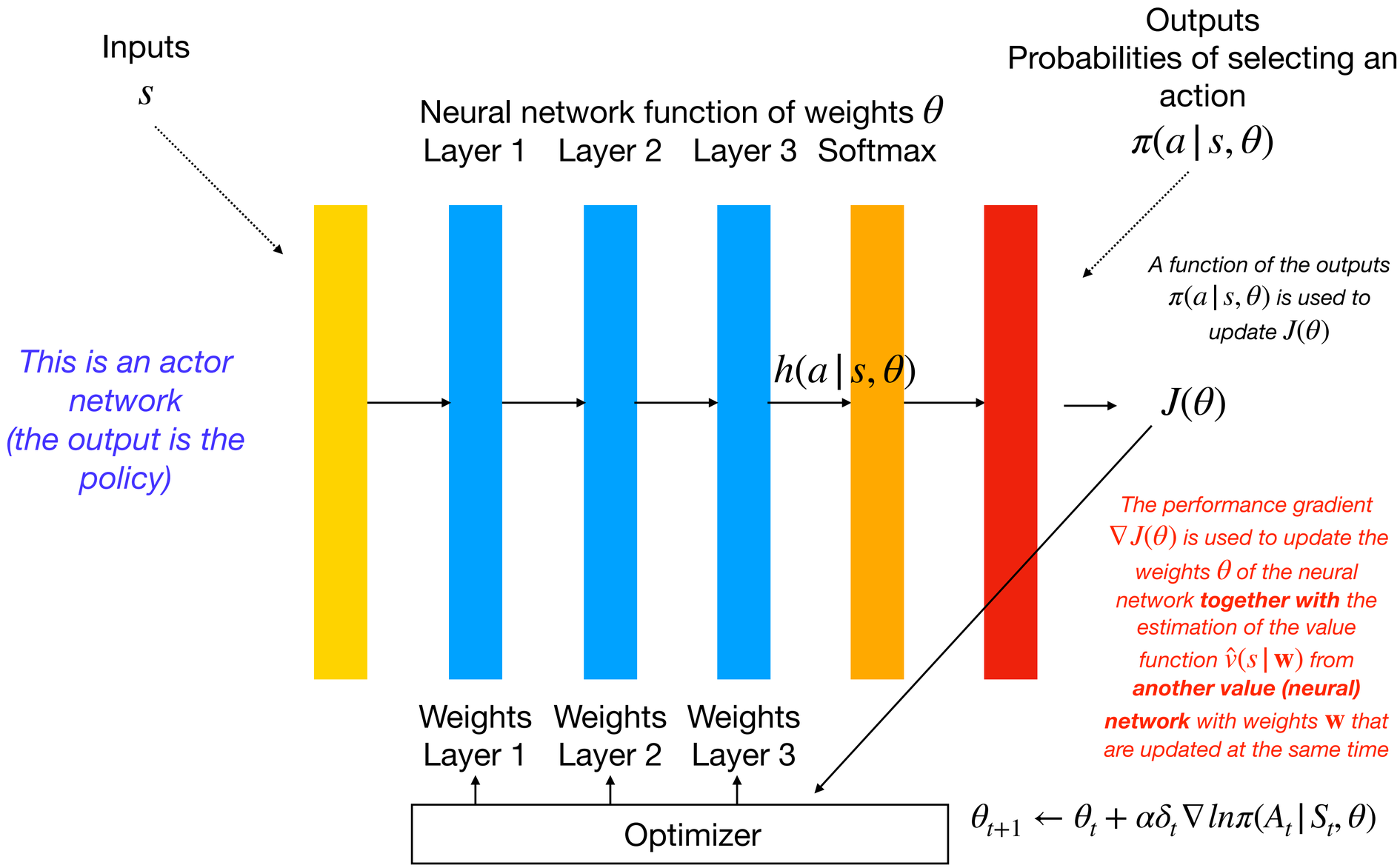
$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t\frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t \nabla ln\pi(A_t \mid S_t, \theta)$$

One-step Actor Methods

- The value of the state is usually estimated with *another network* for example with a semi-gradient TD(0) method (other methods are possible).
- Please note that the weights $\mathbf{w}$ of this other network are learned at the same time, but independently.
- The network that learns the policy (that with weights θ) in our example is the actor network (i.e, it used to act).
- The network that learns the values (that with weights $\mathbf{w}$) is called critic network (i.e., it is used to “judge” the actions of the actor, which is implemented through the actor network).

Policy Network with One-step Actor Critic



One-step Actor-Critic Method

Input: a differentiable policy parametrisation $\pi(a \mid s, \theta)$, a differentiable state-value function parametrisation $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\theta} > 0$ and $\alpha^{w} > 0$

Initialise policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$

Loop forever (for each episode):

Initialise S (first state of episode)

Loop while S is not terminal

Select A using policy π

Take action A, observe S', R

$\delta \leftarrow R + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$

$\theta \leftarrow \theta + \alpha^{\theta} \delta \nabla ln \pi(A \mid S, \theta)$

$S \leftarrow S'$

A Recap

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t)$$

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla ln\pi(A_t \mid S_t, \theta_t)$$

Standard REINFORCE

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(S_t)) \nabla ln\pi(A_t \mid S_t, \theta_t)$$

REINFORCE with baseline

$$\theta_{t+1} = \theta_t + \alpha (G_t - \hat{v}(S_t, \hat{w})) \nabla ln\pi(A_t \mid S_t, \theta_t)$$

REINFORCE with value function estimation

Advantage Function

$$\theta_{t+1} = \theta_t + \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\,\nabla ln\pi(A_t \,|\, S_t, \theta_t)$$

$$\theta_{t+1} = \theta_t + \alpha(R_t + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\,\nabla ln\pi(A_t \,|\, S_t, \theta_t)$$

$$\theta_{t+1} = \theta_t + \alpha\delta\,\nabla ln\pi(A_t \,|\, S_t, \theta_t) \text{ with } \delta = R_{t+1} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

We will use the so-called *advantage function* expressed as follows:

$$\theta_{t+1} = \theta_t + \alpha\delta_{ADV}\,\nabla ln\pi(A_t \,|\, S_t, \theta_t) \text{ with } \delta_{ADV} = \hat{Q}(S_t, A_t, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

Advantage Actor Critic (A2C)

- We can stabilise learning further by using the advantage function as critic instead of the action value function.

- In practice instead of using

$$\delta = R_{t+1} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

we will use

$$\delta_{ADV} = \hat{Q}(S_t, A_t, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

- The latter expression is usually called the *advantage function*.

Advantage Actor Critic (A2C)

- The idea is that the advantage function calculates how better the selection the specific action A_t at state S_t is compared to the average value of the state S_t. In a sense, we are subtracting the mean value of the state from the value of the state-action pair.
- In practice, we do not need to quantify how good an action is in absolute sense, but only how much it is better than others on average.
- It is a relative advantage that we are interested in.
- Different types of advantage functions and approximate advantage functions have been proposed and evaluated in the past years.
- The only (important) condition is that they have to lead to the same expected value for the policy gradient, despite having different variances.

Asynchronous Advantage Actor-Critic (A3C)

- Asynchronous variants have been proposed in order to stabilise the process.
- It has been shown that that parallel actor-learners have a *stabilising* effect on training.
- This has also benefit in terms of performance, since it allows parallel execution.
- This method is usually called Asynchronous Advantage Actor-Critic.
- We are not going to discuss the details of this implementation. If you are interested please check the relevant paper.
- High-scalable implementation of the A3C algorithm has been proposed such as the IMPALA architecture.

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Published at ICML 2016

IMPALA

IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

Lasse Espeholt [*1] **Hubert Soyer** [*1] **Remi Munos** [*1] **Karen Simonyan** [1] **Volodymyr Mnih** [1] **Tom Ward** [1]
Yotam Doron [1] **Vlad Firoiu** [1] **Tim Harley** [1] **Iain Dunning** [1] **Shane Legg** [1] **Koray Kavukcuoglu** [1]

Abstract

In this work we aim to solve a large collection of tasks using a single reinforcement learning agent with a single set of parameters. A key challenge is to handle the increased amount of data and extended training time. We have developed a new distributed agent IMPALA (Importance Weighted Actor-Learner Architecture) that not only uses resources more efficiently in single-machine training but also scales to thousands of machines without sacrificing data efficiency or resource utilisation. We achieve stable learning at high throughput by combining decoupled acting and learning

separately. We are interested in developing new methods capable of mastering a diverse set of tasks simultaneously as well as environments suitable for evaluating such methods.

One of the main challenges in training a single agent on many tasks at once is scalability. Since the current state-of-the-art methods like A3C (Mnih et al., 2016) or UNREAL (Jaderberg et al., 2017b) can require as much as a billion frames and multiple days to master a single domain, training them on tens of domains at once is too slow to be practical.

We propose the **Imp**ortance Weighted **A**ctor-**L**earner **A**rchitecture (IMPALA) shown in Figure 1. IMPALA is capable of scaling to thousands of machines without sacrificing training stability or data efficiency. Unlike the popular

Published at ICML 2018

Soft-Actor Critic (SAC)

- Key (open) problems:
  - Very high sample complexity;
  - Brittleness of the convergence.
- One solution is *regularisation*.
- The Soft-Actor Critic algorithm has been introduced with this goal.
- The key idea is to regularise the expression using an entropy term. The actor aims at maximising the expected reward while also maximising entropy (i.e., exploration).

Soft-Actor Critic (SAC)

- SAC is based on a revised objective function as follows:

$$J(\pi) = \sum_{t=0}^{T} E_{\pi}[R_t + \alpha \mathscr{H}(\pi)]$$

where $\mathscr{H}(\pi)$ is the entropy of the policy over the distribution of the marginals of the trajectory distribution (induced by the policy π). α is called the *temperature parameter.*

- The idea of using entropy for increasing exploration is used widely in reinforcement learning, and, in any situation in which the goal is learning by exploration.

Soft-Actor Critic (SAC)

Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja [1] **Aurick Zhou** [1] **Pieter Abbeel** [1] **Sergey Levine** [1]

Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

Published at ICML 2018

Trust Region Policy Optimization (TRPO)

- TRPO updates policies by taking the largest possible step to improve performance while satisfying a special constraint on how close the new and the old policies are allowed to be.
- The constraint is expressed in terms of KL divergence, a measure of “distance” between probability distributions.
- The definition of the Kullback-Leibler divergence is the following:

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) log(\frac{P(x)}{Q(x)})$$

Trust Region Policy Optimization (TRPO)

- A simple interpretation of the KL divergence of P from Q is the expected excess surprise from using Q as a model instead of P when the actual distribution is P.

- TRPO enforces policy updates to stay within a “true region” using KL-divergence constraints.

Trust Region Policy Optimization (TRPO)

- Let π_θ denote a policy with parameters θ. The theoretical TRPO update is:

$$\theta_{t+1} = \underset{\theta}{argmax} \mathscr{L}(\theta_t, \theta) \text{ s.t. } D_{KL}(\theta || \theta_t) \leq const$$

where $\mathscr{L}(\theta_t, \theta)$ is the surrogate advantage, a measure of how policy π_θ performs relative to the old policy π_{θ_t} using data from the old policy:

$$\mathscr{L}(\theta_t, \theta) = \underset{s,a \sim \pi_{\theta_t}}{E} [\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \delta_{ADV}^{\pi_{\theta_t}}(s, a)]$$

and

Trust Region Policy Optimization (TRPO)

$D_{KL}(\theta||\theta_t)$ is the average KL divergence between policies across visited by the old policy:

$$D_{KL}(\theta||\theta_t) = \underset{s\sim\pi_{\theta_t}}{E}[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta_t}(\cdot||s))]$$

The theoretical TRPO is solved using an approximate solution by means of a Taylor expansion.

Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization

John Schulman JOSCHU@EECS.BERKELEY.EDU
Sergey Levine SLEVINE@EECS.BERKELEY.EDU
Philipp Moritz PCMORITZ@EECS.BERKELEY.EDU
Michael Jordan JORDAN@CS.BERKELEY.EDU
Pieter Abbeel PABBEEL@CS.BERKELEY.EDU
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Abstract

We describe an iterative procedure for optimizing policies, with guaranteed monotonic improvement. By making several approximations to the theoretically-justified procedure, we develop a practical algorithm, called Trust Region Policy Optimization (TRPO). This algorithm is similar to natural policy gradient methods and is effec-

Tetris is a classic benchmark problem for approximate dynamic programming (ADP) methods, stochastic optimization methods are difficult to beat on this task (Gabillon et al., 2013). For continuous control problems, methods like CMA have been successful at learning control policies for challenging tasks like locomotion when provided with hand-engineered policy classes with low-dimensional parameterizations (Wampler & Popović, 2009). The inability of ADP and gradient-based methods to consistently

Proximal Policy Optimization

- PPO is motivated by the same question as TRPO: how can we have the biggest improvement step on a policy using the data that we currently have, without stepping too far so that we cause performance collapse?

- Where TRPO tries to solve the problem with a complex second-order method (essentially an optimisation constraint problem), PPO is a family of first-order methods that use a set of mechanisms to keep the new policies close to the old ones.

- It is important to note that the PPO methods are significantly simpler to implement and, empirically, they perform as well as TRPO.

- PPO is an on-policy algorithm.

PPO Variants

- There are two PPO variants:
  - PPO-Penalty
  - PPO-CLIP

PPO-Penalty

- **PPO-Penalty**:
  - It approximately solves a KL-constrained update like TRPO, but it penalises the KL-divergence instead of making it a constraint.
  - It automatically adjusts the penalty over the course of the training so that is scaled approximately.
  - Since PPO-Penalty is not widely used, we are not going to cover it in detail. Please refer to the paper for a full description of the approach.

PPO-CLIP

- **PPO-CLIP**:
  - It does not have a KL-divergence term in the objective.
  - Actually, it is not based on the resolution of a constraint problem at all.
  - It essentially relies on specialised clipping of the objective function to remove incentives for the new policy to get far from the old policy.
  - It is the version of PPO that is widely used and we will study it in detail.

PPO-CLIP

- PPO updates policies via the following update:

$$\theta_{t+1} = \underset{\theta}{argmax} \underset{s,a \sim \pi_{\theta_t}}{E} [L(s, a, \theta_t, \theta)],$$

typically taking multiple steps of (usually mini batch) stochastic gradient descent to maximise the objective:

$$L(s, a, \theta_t, \theta) = min(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \delta^{\pi_{\theta_t}}_{ADV}, clip(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)}, 1 - \epsilon, 1 + \epsilon) \delta^{\pi_{\theta_t}}_{ADV}(s, a))$$

in which ϵ is a hyper-parameter, which quantifies how far the new policy is allowed to be “far” from the old one.

PPO-CLIP

- A simplified version of PPO-CLIP is the following:

$$L(s, a, \theta_t, \theta) = min(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \delta_{ADV}^{\pi_{\theta_t}}(s, a), g(\epsilon, \delta_{ADV}^{\pi_{\theta_t}}(s, a)))$$

where

$$g(\epsilon, \delta_{ADV}) = \begin{cases} (1 + \epsilon)\delta_{ADV} & \text{when } \delta_{ADV} \geq 0 \\ (1 - \epsilon)\delta_{ADV} & \text{when } \delta_{ADV} < 0 \end{cases}$$

PPO-CLIP

- Let's consider the case in which $\delta_{ADV} \geq 0$: suppose the advantage for that state-action pair positive. In this case the contribution to the objective reduces to:

$$L(s, a, \theta_t, \theta) = min(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)}, (1 + \epsilon))\delta_{ADV}^{\pi_{\theta_t}}(s, a)$$

- Because the advantage is positive, the objective will increase if the action become more likely, i.e., if $\pi_\theta(a \mid s)$ increases. But the min in this term puts a limit to how much the objective can increase. Once $\pi_\theta(a \mid s) > (1 + \epsilon)\pi_{\theta_k}(a \mid s)$, then the min is considered and this terms hits a ceiling of $(1 + \epsilon)\delta_{ADV}^{\pi_{\theta_t}}(s, a)$.

PPO-CLIP

- Let's consider now the case in which $\delta_{ADV} < 0$: in this case the contribution reduces to:

$$L(s, a, \theta_t, \theta) = max(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)}, (1 - \epsilon))\delta_{ADV}^{\pi_{\theta_t}}(s, a)$$

- Because the advantage is negative, the objective will increase if the action becomes less likely, that is $\pi_\theta(a, s)$ decreases. But the max term is then considered and this puts a limit to how much the objective can increase. Once $\pi_\theta(a \mid s) < (1 - \epsilon)\pi_{\theta_k}(a \mid s)$, this term hits a ceiling of $(1 - \epsilon)\delta_{ADV}^{\pi_{\theta_t}}(s, a)$.

Comparison between TRPO and PPO

- TRPO enforces policy updates to stay within a “trust region” using KL divergence constraints requiring second-order optimisation.
- PPO achieves similar results with the simpler clipping mechanism, making it:
  - Computationally more efficient;
  - Easier to implement!

Proximal Policy Optimization (PPO)

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

Approximating Advantage Functions

- Methods have been developed for approximating the advantage functions.
- An important method is the *Generalized Advantage Estimation (GAE)*, which is based on the idea of using an exponentially-weighted estimator of the advantage function.
- The GAE method is at the basis of the practical implementation of both TRPO and PPO.

Generalised Advantage Estimation (GAE)

- In particular the GAE advantage $\hat{\delta}_t^{GAE} = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}$

where:

- $\delta_t = r_t + \gamma V(S_{t+1}) - V(S_t)$ is the temporal difference (TD) error at time t;
- γ is the discount factor;
- λ is the GAE parameter.

Generalised Advantage Estimation (GAE)

HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION

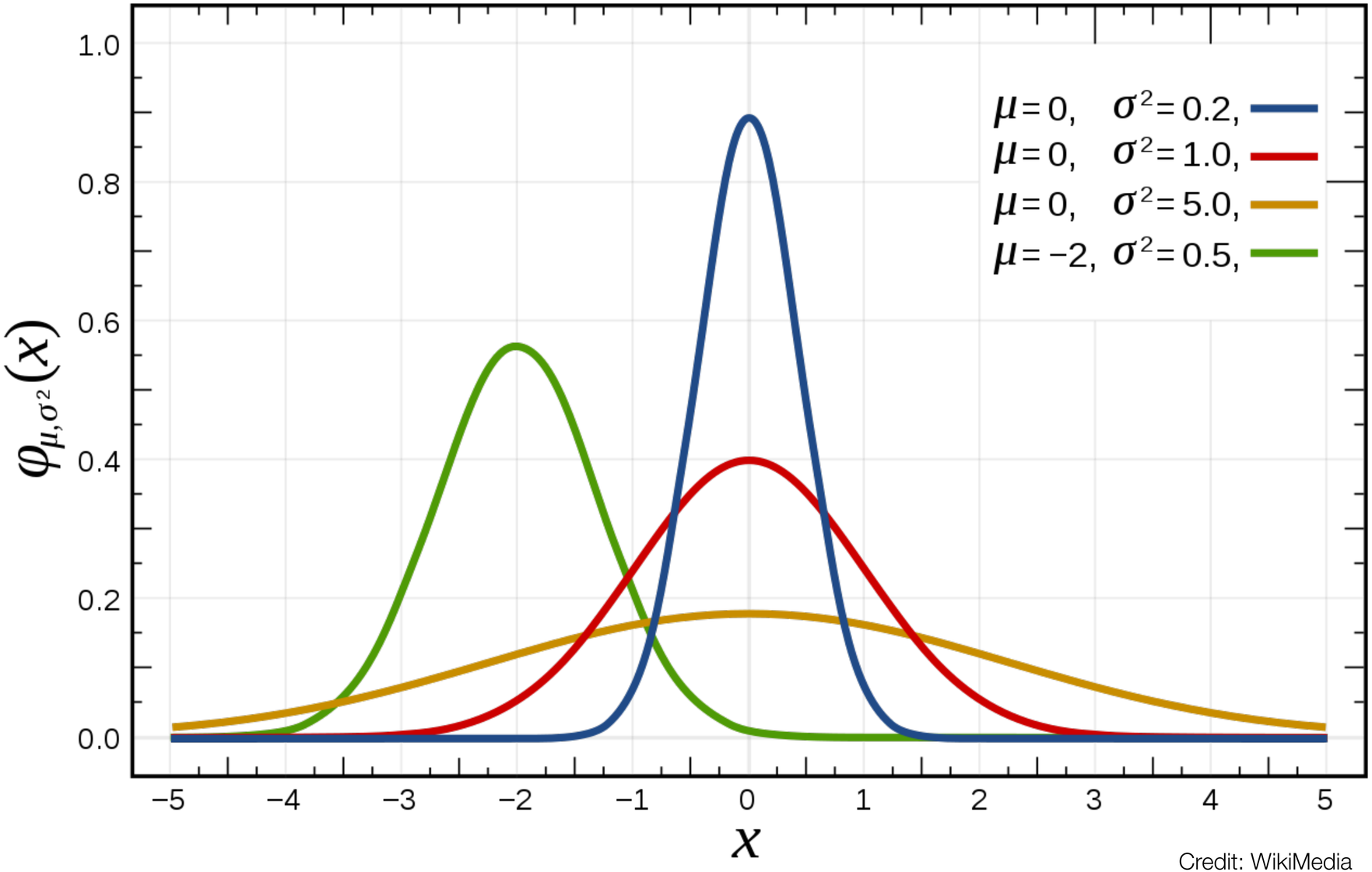
John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{joschu,pcmoritz,levine,jordan,pabbeel}@eecs.berkeley.edu

ABSTRACT

Policy gradient methods are an appealing approach in reinforcement learning because they directly optimize the cumulative reward and can straightforwardly be used with nonlinear function approximators such as neural networks. The two main challenges are the large number of samples typically required, and the difficulty of obtaining stable and steady improvement despite the nonstationarity of the incoming data. We address the first challenge by using value functions to substantially reduce the variance of policy gradient estimates at the cost of some bias, with an exponentially-weighted estimator of the advantage function that is analogous to TD(λ). We address the second challenge by using trust region optimization procedure for both the policy and the value function, which are represented by

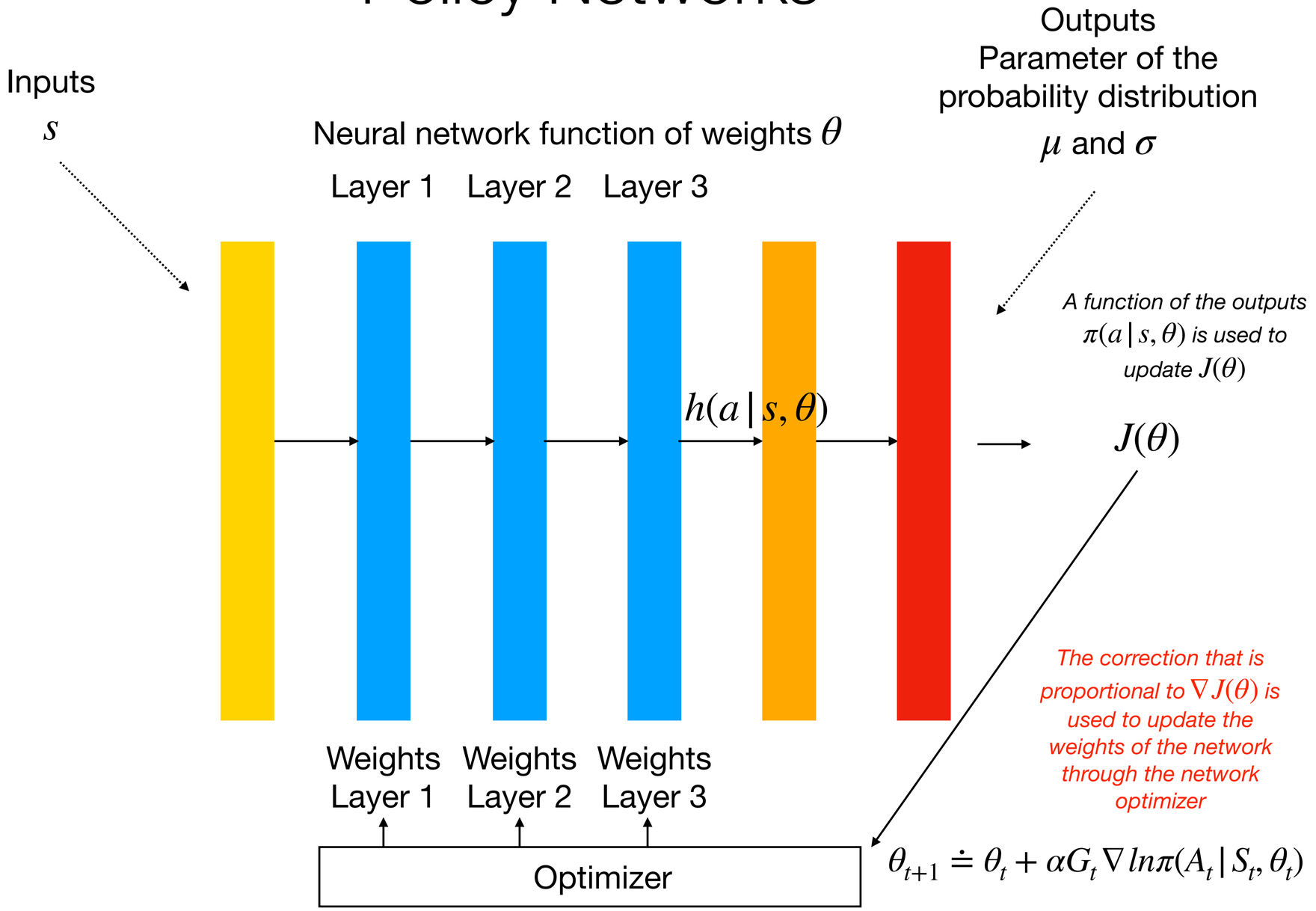
Continuous Action Space

- In this lecture, we have examined the case of discrete actions, but it is worth noting that the policy networks can be used to learn continuous actions (think about moving the wheel of a car).
- In that case the output is not a *discrete* probability distribution $\pi(a \mid s, \theta)$ on the state space (i.e., at the end a set of discrete values), but a *continuous* one.
- In that case we will not learn the discrete probability for each actions, but the parameters of a probability distribution.
- For example, we can use a Gaussian distribution and learn the parameters $mean(s)$ and $variance(s)$.
- Then, we will sample from that distribution for extracting the action to be taken.



Credit: WikiMedia

Policy Networks



References

- Chapter 13 of Barto and Sutton. Introduction to Reinforcement Learning. Second Edition. MIT Press 2018.
- For A2C, SAC, TRPO and PPO, please refer to the original papers.
- For TRPO and PPO, please also refer to the material on Open AI Spinning up RL (the material about TRPO and PPO is partially adapted from it).