Policy Gradients

Some Slides from DeepMind x UCL RL Lecture Series

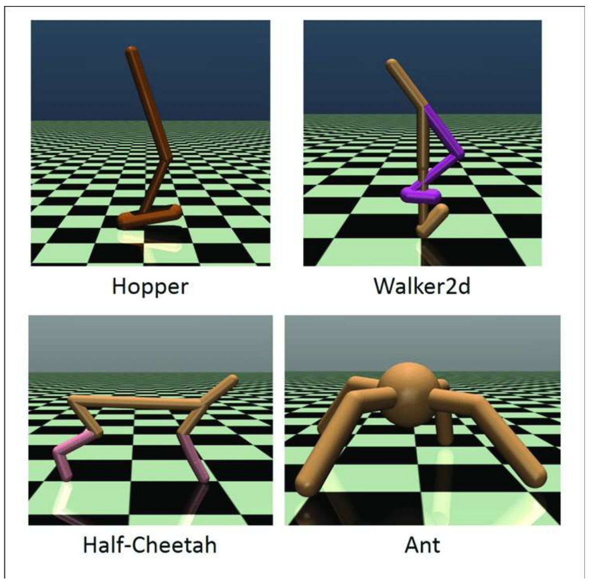
John Tan Chong Min

12 Oct 2022

Is Policy-Gradient that good?

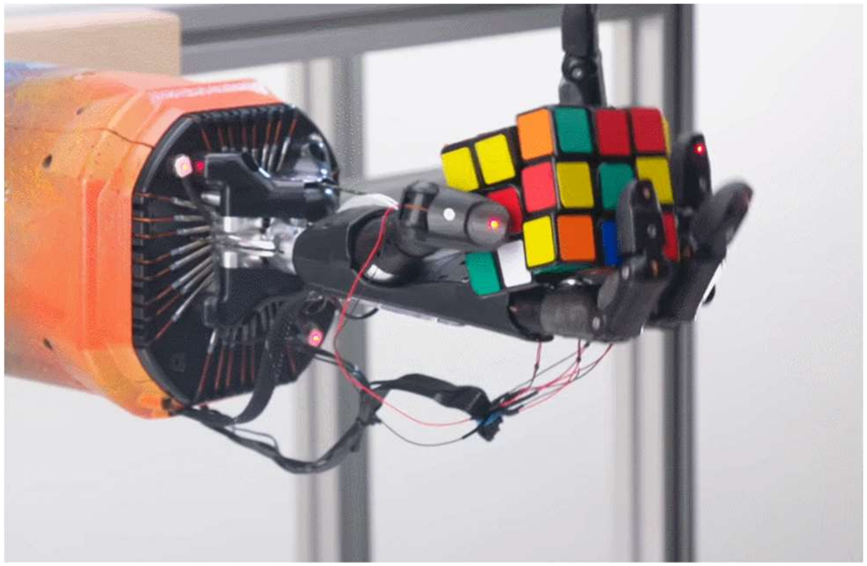
Proximal Policy Optimisation Success Cases

MuJoCo



Atari

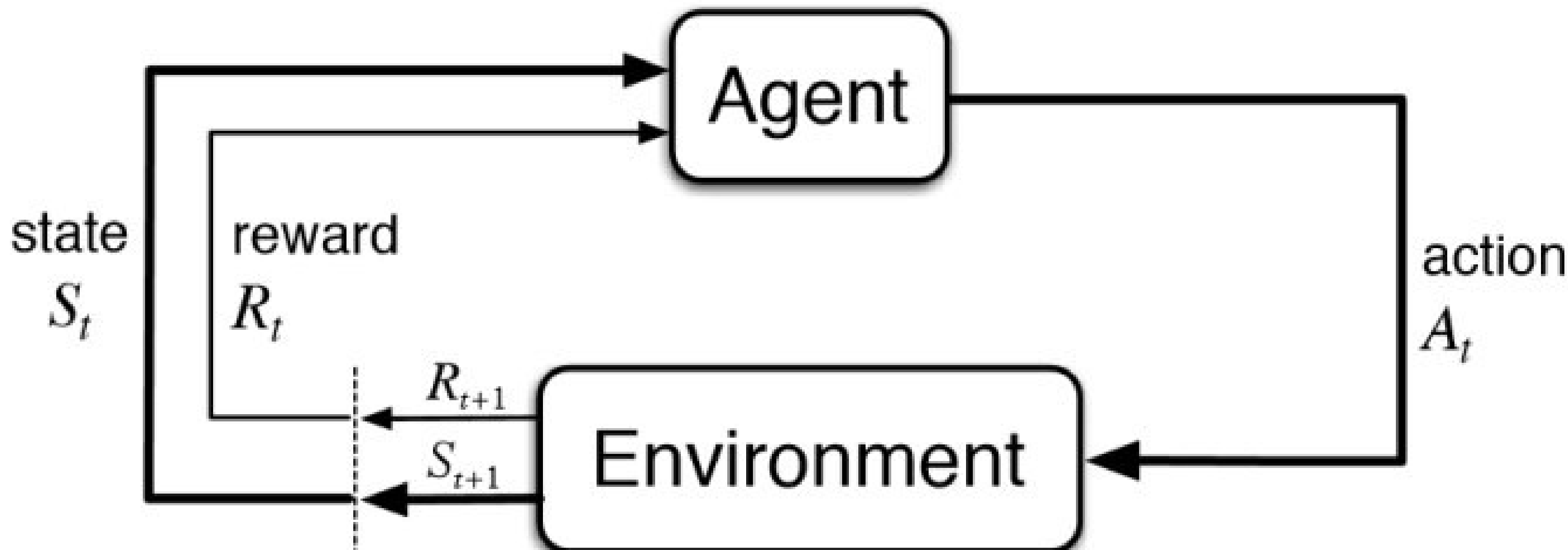
Proximal Policy Optimisation Success Cases



Proximal Policy Optimisation Success Cases

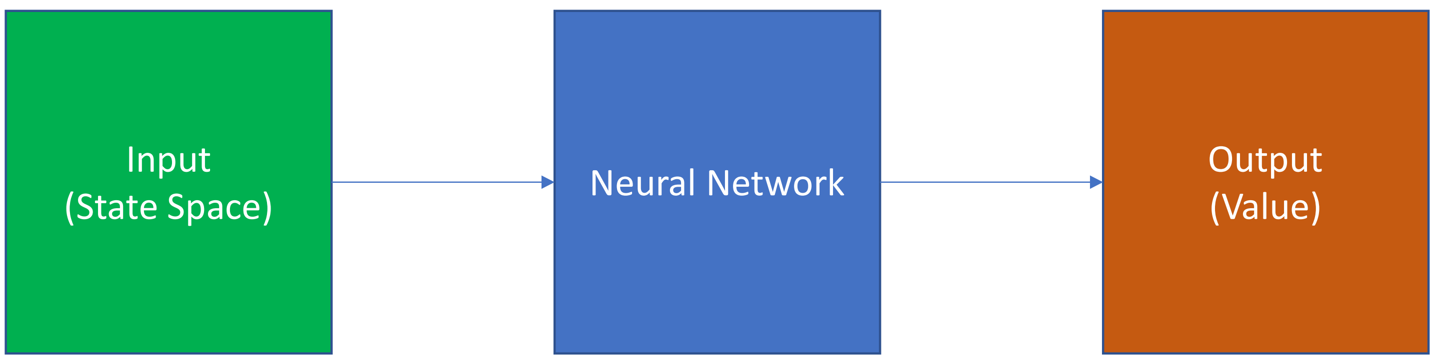
Recap on Reinforcement Learning

Reinforcement Learning Framework



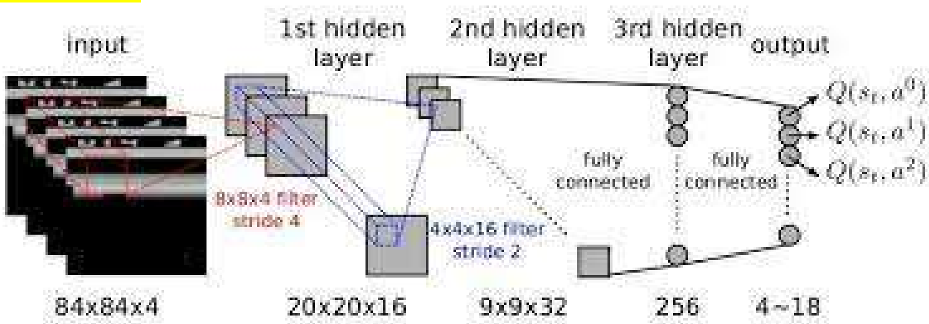
Previously in Reinforcement Learning (Value-based)

- Train the neural network to give the value of each state/state-action
- Can be trained using supervised learning approach
- Input: State space, Output: Value – *V(s)* or *Q(s, a)*



Previously in Reinforcement Learning (Value-based)

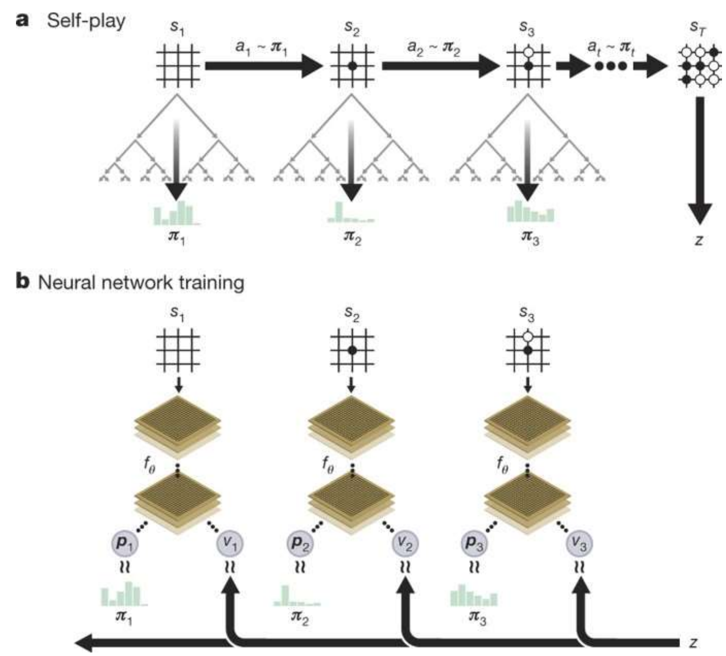
DQN



$$loss = \Big(\underbrace{\overset{\text{Reward}}{r} + \overset{\text{Decay Rate}}{\gamma} \max_{a`} \hat{Q}(s,a`)}_{\text{Target}} - \underbrace{Q(s,a)}_{\text{Prediction}}\Big)^2$$

$$a = \max_a Q(s, a; \theta)$$

AlphaZero



$$(\mathbf{p}, v) = f_\theta(s), \qquad l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2$$

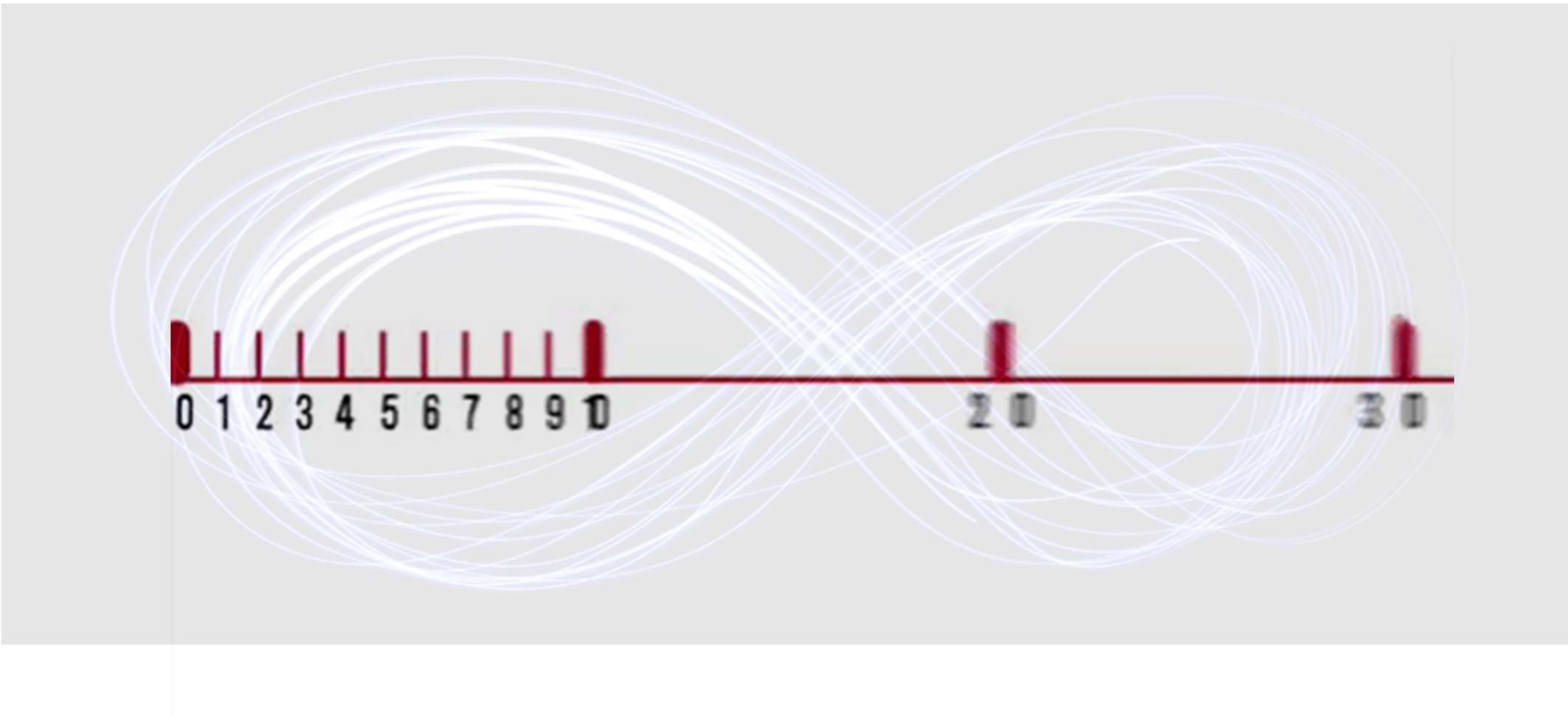
What is the limitation?

- Optimal action is chosen by maximum Q-value or maximum probability / visit counts in Monte Carlo Tree Search

- Only suitable for discrete action space

- Also, may not perform well in a stochastic environment!

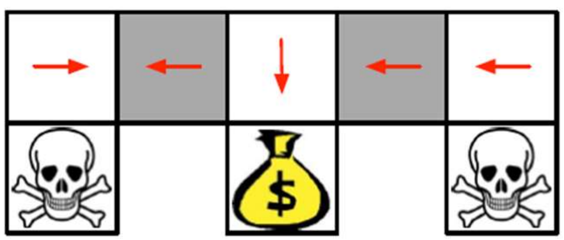
What if we have a continuous state space?



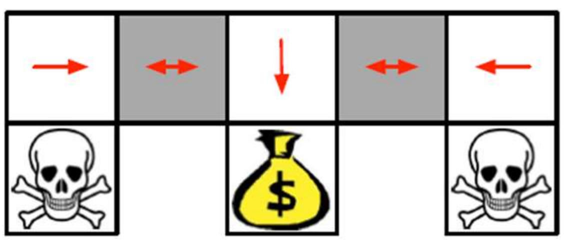
Aliased Gridworld

- You cannot differentiate between the two grey squares
- Value-based policies are deterministic, may get stuck
- Though this can be mitigated by **incorporating memory**

Deterministic



Stochastic



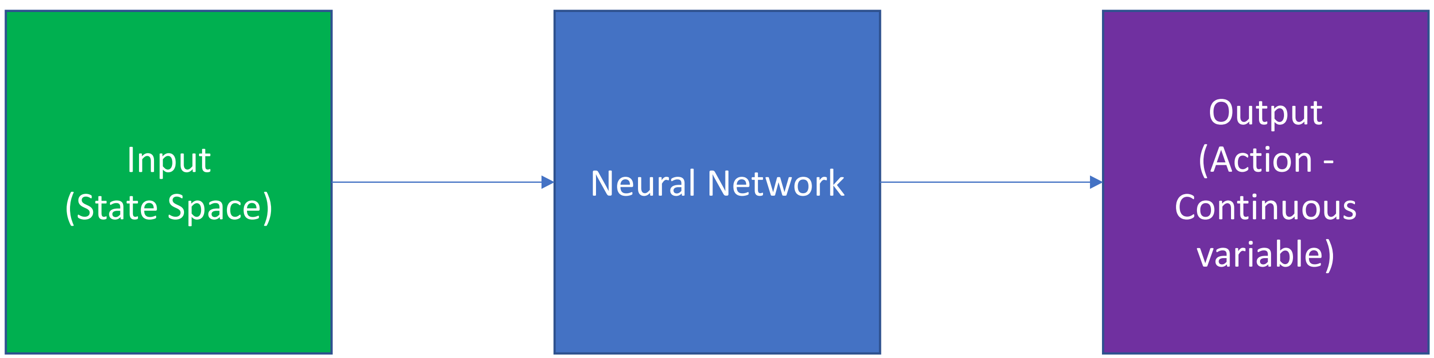
Policy Gradients

A method which works on continuous action spaces

A method which works on stochastic environments (but takes so long to train!!!)

Policy Gradient Framework

- Train the neural network to give the action output directly
- Can be trained using supervised learning approach
- Input: State space, Output: Optimal action



$$\pi_{\boldsymbol{\theta}}(a|s) = p(a|s, \boldsymbol{\theta})$$

Advantages / Disadvantages of Policy-Based method

- Advantages:
  - Better convergence properties
  - Effective in high-dimension or continuous action spaces
  - Can learn stochastic policies

- Disadvantages:
  - Typically converges to local rather than global optimum
  - Evaluating a policy is inefficient and has high variance

How to train policy?

- Goal: given **policy $\pi_\theta(s, a)$**, find best **parameters θ**
- How do we measure the quality of a policy π_θ?
- In episodic environments we can use the **average total return per episode**
- In continuing environments we can use the **average reward per step**

Episodic return objective

- **Episodic-return objective**:

$$
\begin{aligned}
J_{\mathrm{G}}(\boldsymbol{\theta}) &= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right] \\
&= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}}[G_0] \\
&= \mathbb{E}_{S_0 \sim d_0}[\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_t \mid S_t = S_0]] \\
&= \mathbb{E}_{S_0 \sim d_0}[v_{\pi_{\boldsymbol{\theta}}}(S_0)]
\end{aligned}
$$

where d_0 is the start-state distribution

Average reward objective

- **Average-reward objective**

$$
\begin{aligned}
J_{\mathrm{R}}(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[R_{t+1}\right] \\
&= \mathbb{E}_{S_t \sim d_{\pi_{\boldsymbol{\theta}}}}\left[\mathbb{E}_{A_t \sim \pi_{\boldsymbol{\theta}}(S_t)}\left[R_{t+1} \mid S_t\right]\right] \\
&= \sum_s d_{\pi_{\boldsymbol{\theta}}}(s) \sum_a \pi_{\boldsymbol{\theta}}(s, a) \sum_r p(r \mid s, a) r
\end{aligned}
$$

where $d_\pi(s) = p(S_t = s \mid \pi)$ is the probability of being in state s in the long run
Think of it as the ratio of time spent in s under policy π

Policy Optimisation

- Policy based reinforcement learning is an **optimization** problem
- Find $\boldsymbol{\theta}$ that maximises $J(\boldsymbol{\theta})$
- We will focus on **stochastic gradient ascent**, which is often quite efficient (and easy to use with deep nets)
- Some approaches do not use gradient
  - Hill climbing / simulated annealing
  - Genetic algorithms / evolutionary strategies

Gradient Ascent

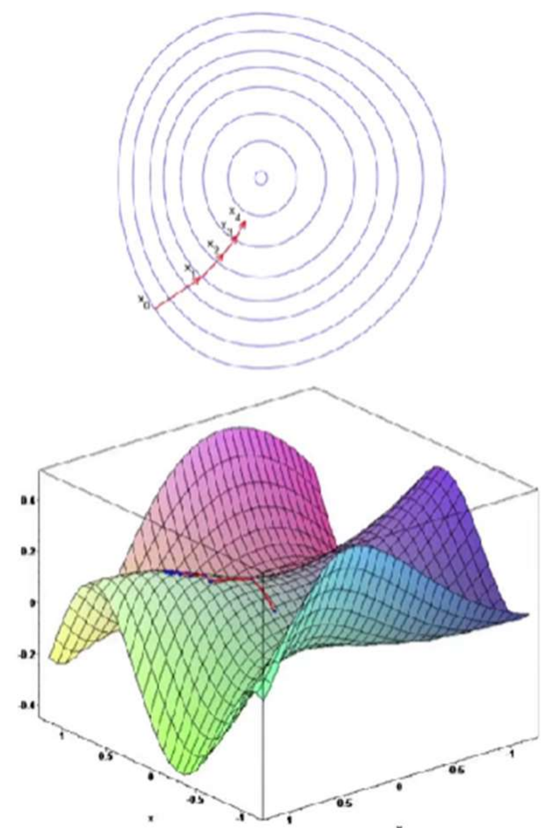
- Idea: ascent the gradient of the objective $J(\boldsymbol{\theta})$

$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- Where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the **policy gradient**

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} \end{pmatrix}$$

- and α is a step-size parameter
- Stochastic policies help ensure $J(\boldsymbol{\theta})$ is smooth (typically/mostly)



Score Function Trick

Let $r_{sa} = \mathbb{E}\left[R(S, A) \mid S = s, A = s\right]$

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta}[R(S, A)] &= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s)\, r_{sa} \\
&= \sum_s d(s) \sum_a r_{sa}\, \nabla_\theta \pi_\theta(a|s) \\
&= \sum_s d(s) \sum_a r_{sa}\, \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s)\, r_{sa}\, \nabla_\theta \log \pi_\theta(a|s) \\
&= \mathbb{E}_{d,\pi_\theta}[R(S, A)\, \nabla_\theta \log \pi_\theta(A|S)]
\end{aligned}$$

Policy-gradient Update

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A|S) R(S, A)]$$

- This is something we **can** sample
- Our stochastic policy-gradient update is then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha R_{t+1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_t}(A_t|S_t)$$

- In expectation, this is the following the actual gradient
- So this is a pure (unbiased) stochastic gradient algorithm
- Intuition: increase probability for actions with high rewards

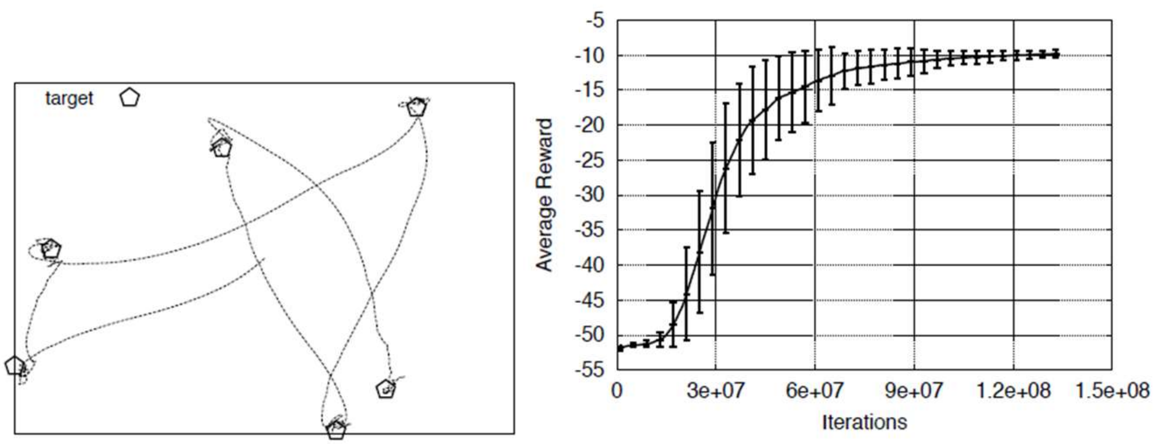
Gaussian Policy

- As example, consider a **Gaussian policy**
- E.g., mean is some function of state $\mu_{\theta}(s)$
- For simplicity, lets consider fixed variance of σ^2 (can be parametrized as well)
- Policy is Gaussian, $A_t \sim \mathcal{N}(\mu_{\theta}(S_t), \sigma^2)$
  (here μ_{θ} is the mean — not to be confused with the behaviour policy!)
- The gradient of the log of the policy is then

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{A_t - \mu_{\theta}(S_t)}{\sigma^2} \nabla \mu_{\theta}(s)$$

- Intuition: if return was high, move $\mu_{\theta}(S_t)$ toward A_t

Puck-World Environment



Monte Carlo is sample inefficient!

- Continuous actions exert small force on puck
- Puck is rewarded for getting close to target
- Target location is reset every 30 seconds
- Policy is trained using variant of Monte-Carlo policy gradient

Reduce Variance

- Updates can be very “jumpy” based on value of action
- Let’s reduce the variance by subtracting a baseline independent of action
  - A good baseline is the TD-error

- Note that, in general

$$
\begin{aligned}
\mathbb{E}\left[b\nabla_{\theta}\log\pi(A_t|S_t)\right] &= \mathbb{E}\left[\sum_a \pi(a|S_t) b \nabla_{\theta}\log\pi(a|S_t)\right] \\
&= \mathbb{E}\left[b\nabla_{\theta}\sum_a \pi(a|S_t)\right] \\
&= \mathbb{E}\left[b\nabla_{\theta}1\right] \qquad = 0
\end{aligned}
$$

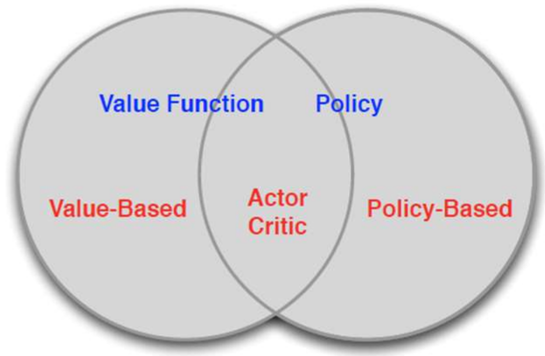
- This is true if b does not depend on the action (but it can depend on the state)
- Implies we can subtract a **baseline** to reduce variance

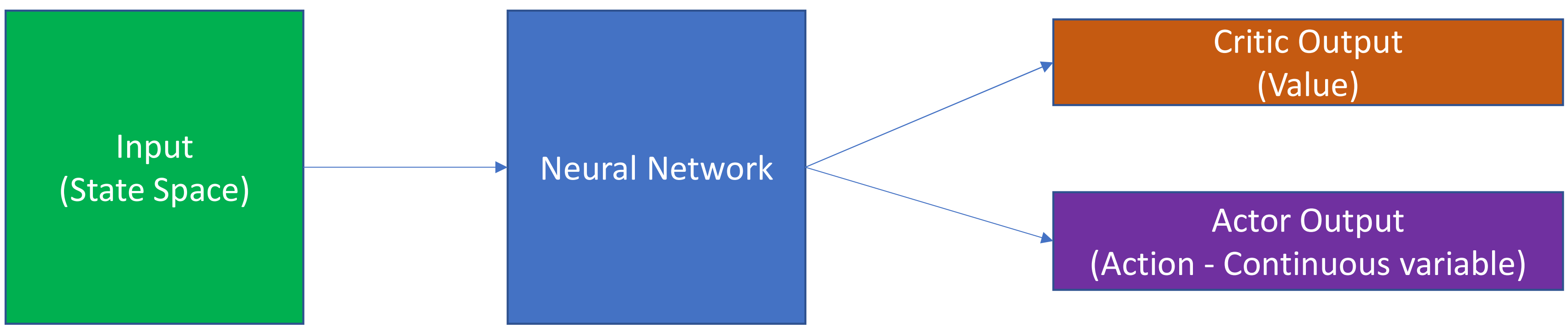
$$
\theta_{t+1} = \theta_t + \alpha(R_{t+1} - b(S_t))\nabla_{\theta}\log\pi_{\theta_t}(A_t|S_t)\,.
$$

Actor-Critic Framework

Actor-Critic

- Actor learns policy π(a|s)
- Critic learns value function *V(s)*
- Actor updates policy using guidance from critic





Actor-Critic Update

Critic Update parameters $\boldsymbol{w}$ of $v_{\boldsymbol{w}}$ by TD (e.g., one-step) or MC

Actor Update $\boldsymbol{\theta}$ by policy gradient

function ONE-STEP ACTOR CRITIC

Initialise s, $\boldsymbol{\theta}$

for t = 0, 1, 2, … **do**

Sample $A_t \sim \pi_{\boldsymbol{\theta}}(S_t)$

Sample R_{t+1} and S_{t+1}

$\delta_t = R_{t+1} + \gamma v_{\boldsymbol{w}}(S_{t+1}) - v_{\boldsymbol{w}}(S_t)$ [one-step TD-error, or **advantage**]

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta\, \delta_t\, \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(S_t)$ [TD(0)]

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \delta_t\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)$ [Policy gradient update (ignoring γ^t term)]

How to prevent drastic policy updates?

Trust Region Policy Optimisation (TRPO)

2.1 Policy Gradient Methods

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The most commonly used gradient estimator has the form

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right] \tag{1}$$

In TRPO [Sch+15b], an objective function (the "surrogate" objective) is maximized subject to a constraint on the size of the policy update. Specifically,

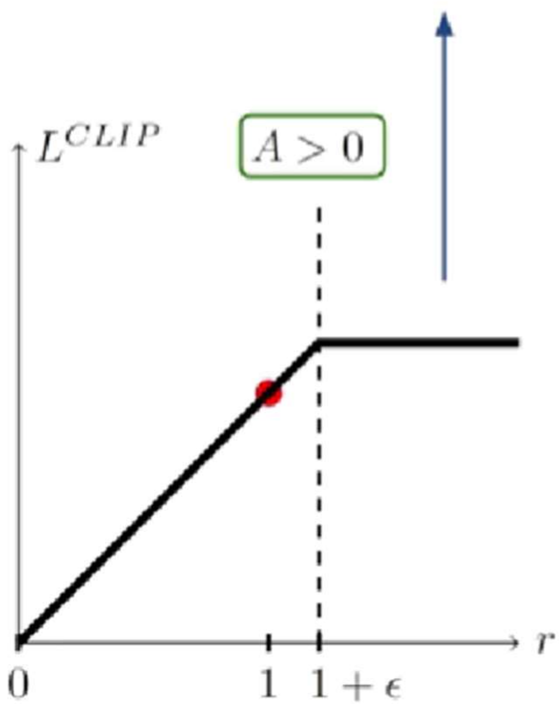
$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \tag{3}$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \tag{4}$$

Here, θ_{old} is the vector of policy parameters before the update. This problem can efficiently be approximately solved using the conjugate gradient algorithm, after making a linear approximation to the objective and a quadratic approximation to the constraint.

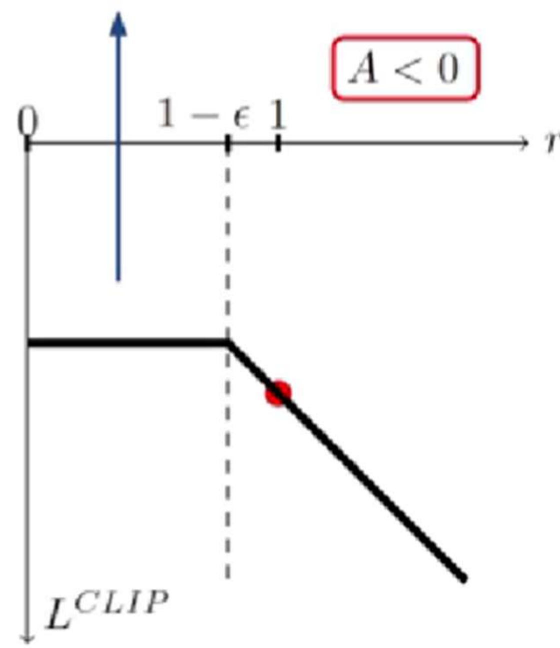
If the action was **good…**

… and it became a lot more probable after the last gradient step, don't keep updating too much or else it might get worse!



If the action was **bad…**

… and it just became a lot less probable, don't keep reducing it's likelihood too much for now!



$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)\Big]$$

Overall PPO Loss

Combining these terms, we obtain the following objective, which is (approximately) maximized each iteration:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \tag{9}$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{\text{targ}})^2$.

Discussion