# 11강. Policy Gradient methods

# Contents

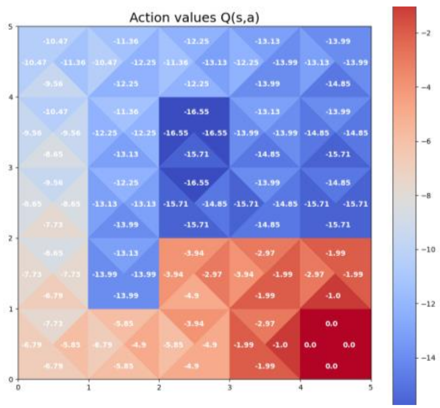
# Policy Gradient methods

# Overview

- So far, we learned Q value-based methods, the policy is defined based on Q values

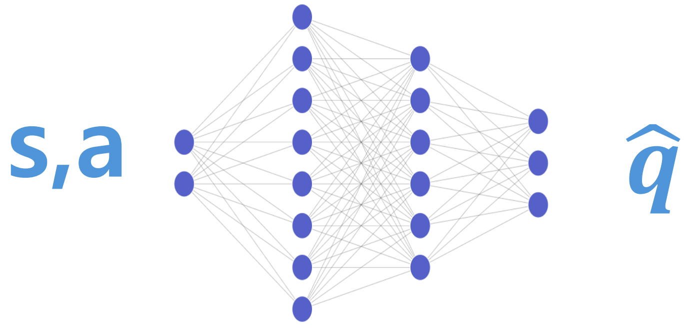
Value table:

$$a = \arg\max_{a} Q(s, a)$$




Function approximators:

$$a = \arg\max_{a} \hat{q}(s, a|\theta)$$



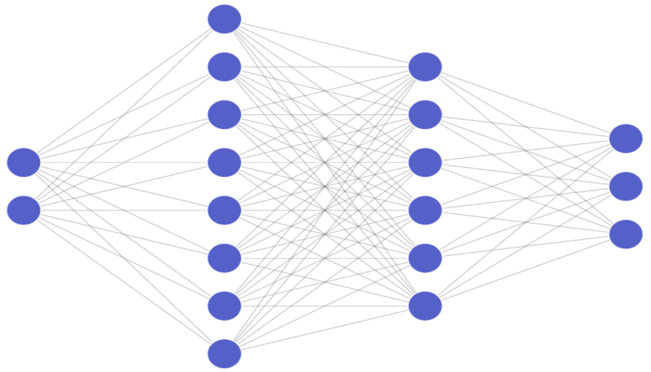
# Overview

- PG methods use a function approximator, not to estimate Q value, but to estimate the probabilities of taking each action:

$$\pi(\mathrm{a}|\mathrm{s},\theta) \in [0,1]$$

- The neural network is the policy (stochastic):

Input:
s=[$s_1$,$s_2$]



Output:
$\pi(S|\theta) = [p(\mathrm{a}_1), p(\mathrm{a}_2), p(\mathrm{a}_3)]$

# Advantages

- Value-based methods cannot represent stochastic policies in a simple way

Greedy policy:

$$\pi(a'|s)\begin{cases}1\ if\ a' = \arg\max_a \hat{q}(s,a|\theta)\\ 0\ else\end{cases}$$

Epsilon-greedy policy

$$(a'|s)\begin{cases}1-\varepsilon+\ \frac{\varepsilon}{|A|}\ if\ a' = \arg\max_a \hat{q}(s,a|\theta)\\ \frac{\varepsilon}{|A|}\qquad else\end{cases}$$

- Imagine a poker game where the agent has imperfect information

If $\pi$*(s) = [0.7,0.3], how do we represent it?

# Advantages

- The policy changes more smoothly during learning:

## Value-based methods:

when the maximum q-value changes, they choose a new action 100% of the time.

$$a \quad = \arg\max_a \hat{q}(s, a|\theta)$$

## Policy gradient methods:

the probability of choosing an action changes in small increments

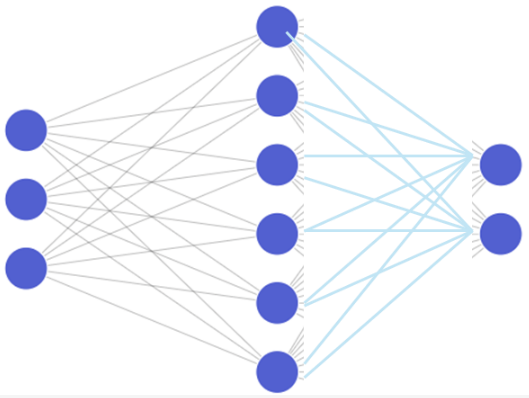
$$a \sim \pi(s|\theta)$$

The probabilities of taking an action gradually increase if the action turns out to be effective and gradually decrease if not

# Stochastic policy

- The neural network can be viewed as a function:

$$y = \varphi_2(\varphi_1(x \cdot w1) \cdot w2)$$



```
nural_network = nn.Sequential(
    nn.Linear(3, 6),
    nn.ReLU(),
    nn.Linear(6,2))
```

W1 =
$$\begin{bmatrix} w11 & w12 & w13 & w14 & w15 & w16 \\ w21 & w22 & w23 & w24 & w25 & w26 \\ w31 & w32 & w33 & w34 & w35 & w36 \end{bmatrix}$$

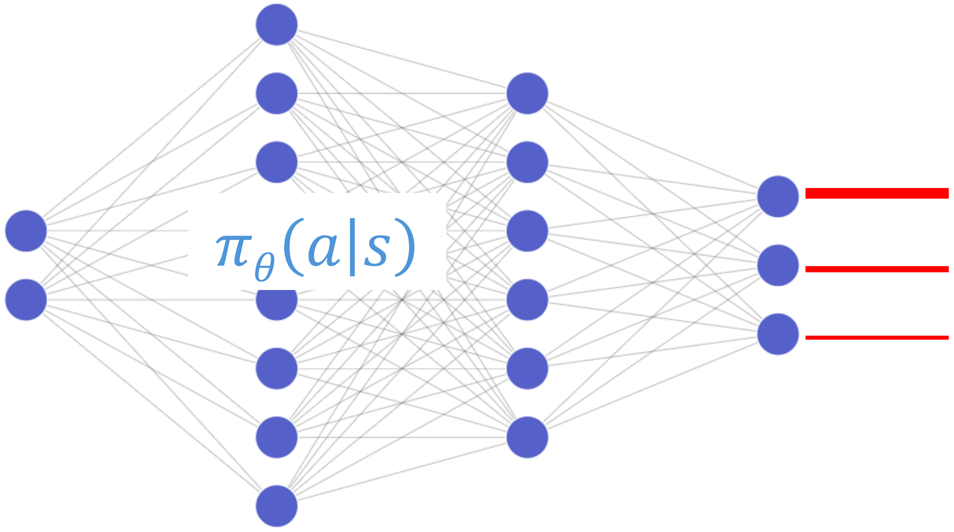
W2 = $$\begin{bmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \\ w41 & w42 \\ w51 & w52 \\ w61 & w62 \end{bmatrix}$$

- By changing its parameters W1, we can modify it to approximate the function we are interested in

# Stochastic policy

- By normalizing the neural network's output, softmax activation function is perfect tool to generate probability vector, y

$x=[x_1, x_2]$




$y= [\sigma(x_1),\sigma(x_2),\ldots,\sigma(x_3)]$

$\text{e.g.}, y= [0.6,0.39,0.01)]$

$$\sigma(xi) = \frac{e^{x_i}}{\sum e^{x_i}}$$

$$\sum_i \sigma(\mathrm{xi}) = 1$$

# Stochastic policy

- Discrete action space:


- Continuous action space:

# Policy performance

- If we want to find the optimal policy, we need to be able to compare them.

- We will define policy performance as: $J(\theta)$

the performance of the policy is a function of its parameters

$$J(\theta) = E^{\pi\theta}[R|s]$$

$$= V^{\pi\theta}(s)$$

$$= \sum_a \pi_\theta(a|s)\, Q^{\pi\theta}(s,a)$$

By changing the parameters of the neural net., we change the policy

# Policy performance

- If $J_{\pi 1}(\theta) > J_{\pi 2}(\theta)$, then we consider that $J_{\pi 1}(\theta)$ is better than $J_{\pi 2}(\theta)$

- Our goal is to find the $\theta$ values that maximize policy performance:

$$\pi * (a|s, \theta) = \arg\max_{a} J(\theta)$$

- We will use the experience samples that the agent obtains to approximate the policy's performance as $\hat{J}(\theta)$

# SGA

- We will approximate the optimal $\theta$ values by stochastic gradient ascent (SGA):

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

we'll take the gradient step in the direction of steepest ascent

where:

$$\nabla \hat{J}(\theta) = \left[ \frac{\partial \hat{J}(\theta)}{\partial \theta_1}, \frac{\partial \hat{J}(\theta)}{\partial \theta_2}, \dots, \frac{\partial \hat{J}(\theta)}{\partial \theta_n} \right]$$

# SGA

- Optimizing the policy using values that the agent can observe.
- $\nabla_\theta J(\theta) = \sum_a \nabla_\theta \pi_\theta(a|s)\, Q^{\pi\theta}(s,a)$

$$= \sum_a \underbrace{\pi_\theta(a|s)}_{\text{Distribution of collected data}}\, Q^{\pi\theta}(s,a)\, \underbrace{\nabla_\theta log\pi_\theta(a|s)}_{\text{Equal to } \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}}$$

$$= \mathrm{E}_{\mathrm{a}}{}^{\pi\theta}\, [Q^{\pi\theta}(s,a)\, \nabla_\theta log\pi_\theta(a|s)]$$

$$\approx \underbrace{\hat{E}}_{\text{Sample mean that follows } \pi\theta}\, [Q^{\pi\theta}(s,a)\, \nabla_\theta log\pi_\theta(a|s)]$$

# Policy gradient theorem

$$\underbrace{\nabla J(\theta)}_{\text{gradient of policy performance}} \propto \sum_s \underbrace{\mu(s)}_{\text{state distribution following policy } \pi} \sum_a \underbrace{q_\pi(s,a)}_{\text{q-value of a state-action pair following policy } \pi} \underbrace{\nabla \pi(a|s,\theta)}_{\text{Gradient of the probability of selecting action a.}}$$

gradient of policy performance

state distribution following policy $\pi$

q-value of a state-action pair following policy $\pi$

Gradient of the probability of selecting action a.

# Policy gradient theorem

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\theta)$$

- The PG is,
  - proportional to the return of each action in each state,
  - multiplied by the gradient of the probability of taking that action in that state
  - and weighted by the frequency with which we observe each state following that policy

# Policy gradient theorem

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi(a|s,\theta)$$

- If an action in a state produces a positive return, we must increase the probability of picking that action to increase the return.
- If that action produces a negative return, the probability of taking that action must be reduced.

# REINFORCE

# Overview

- Policy gradient + Monte Carlo
- We will perform stochastic gradient ascent(SGA):

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

- To do this we will have to approximate the gradient of the policy performance estimate $\nabla \hat{J}(\theta)$ using samples collected from the environment

# Overview

- Note there's no bootstrapping, $Q^{\pi\theta}(s,a) = \mathrm{E}^{\pi\theta}[\mathrm{R}|\mathrm{s},\mathrm{a}]$
- $\nabla_\theta J(\theta) = \sum_a \nabla_\theta \pi_\theta(a|s)\, \underline{R_{s,a}}$

$$= \sum_a \underbrace{\pi_\theta(a|s)}_{\text{Distribution of collected data}} R_{s,a} \underbrace{\nabla_\theta log\pi_\theta(a|s)}_{\text{Equal to } \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}}$$

$$= \mathrm{E}_a{}^{\pi\theta}\,[R_{s,a} \nabla_\theta log\pi_\theta(a|s)]$$

$$\approx \underbrace{\hat{E}}_{\text{Sample mean that follows } \pi\theta}\,[R_{s,a} \nabla_\theta log\pi_\theta(a|s)]$$

# Overview

- We will perform stochastic gradient ascent(SGA):

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

- REINFORCE perform stochastic gradient ascent(SGA):

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \nabla \ln \pi(a|s, \theta)$$

# Overview

Initialize $\theta$ of the parameterized policy $\pi_\theta$ and learning rate $\eta$

While True

Generate an episode $e = (s_0, a_0, r_1, s_1, \cdots, s_{T-1}, a_{T-1}, r_T, s_T)$ using $\pi_\theta$

$\Delta_\theta = 0$

For $t \in 0 : T-1$

$$g_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}$$

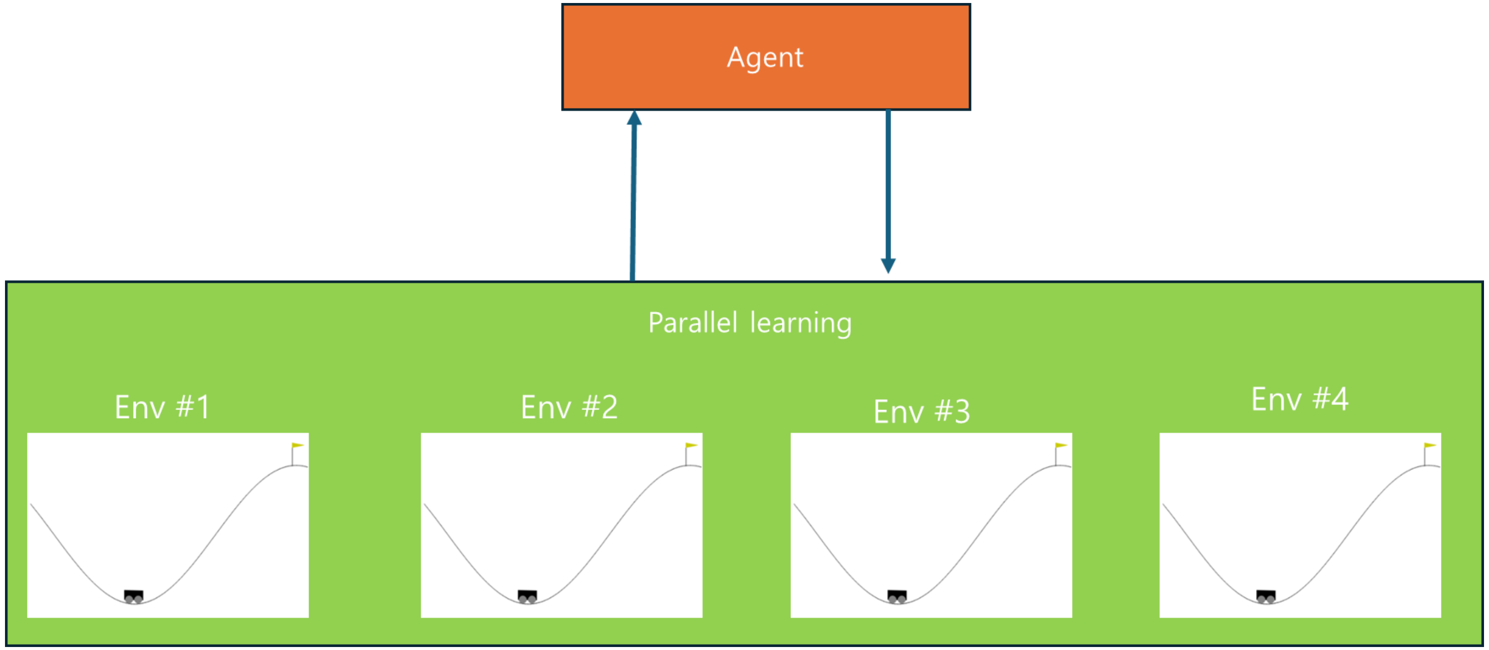
$$\Delta_\theta \leftarrow \Delta_\theta + g_t \nabla_\theta \log \pi_\theta(a|s)$$

$\theta \leftarrow \theta + \eta \cdot \Delta_\theta$

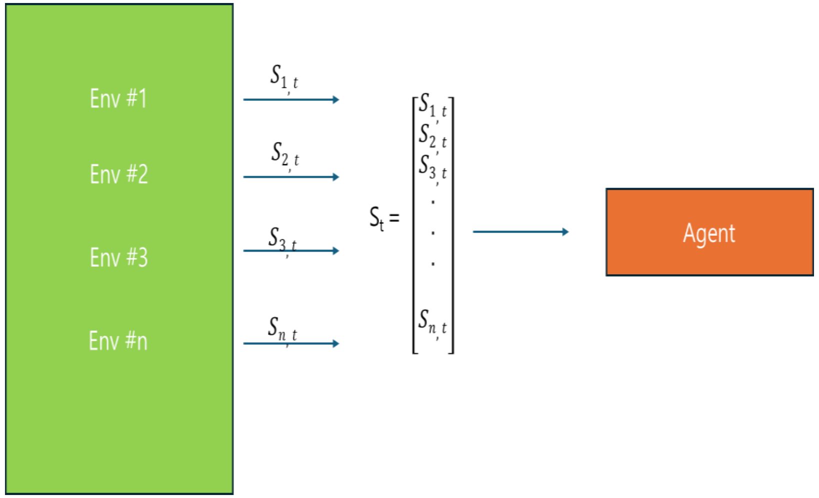
- Get an episode
- $g_t$ is the actual return obtained starting at time t of the episode(all the rewards obtained until the end of the episode)
- the gradient of the policy performance estimate $\Delta_\theta$ using samples collected from the environment
- Stochastic gradient ascent:

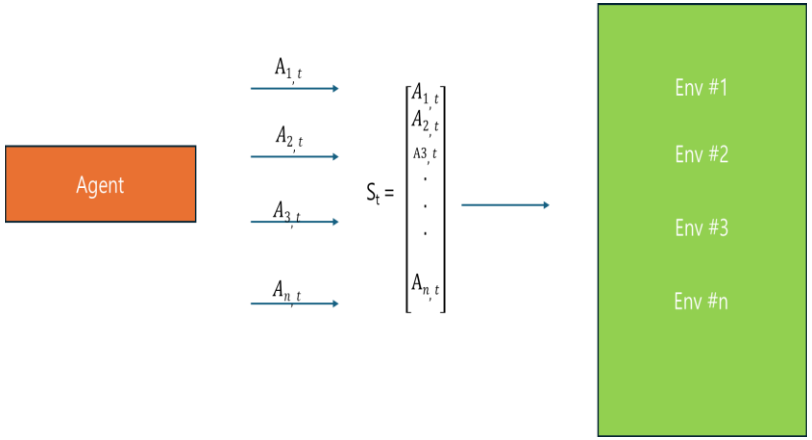
# Parallel learning

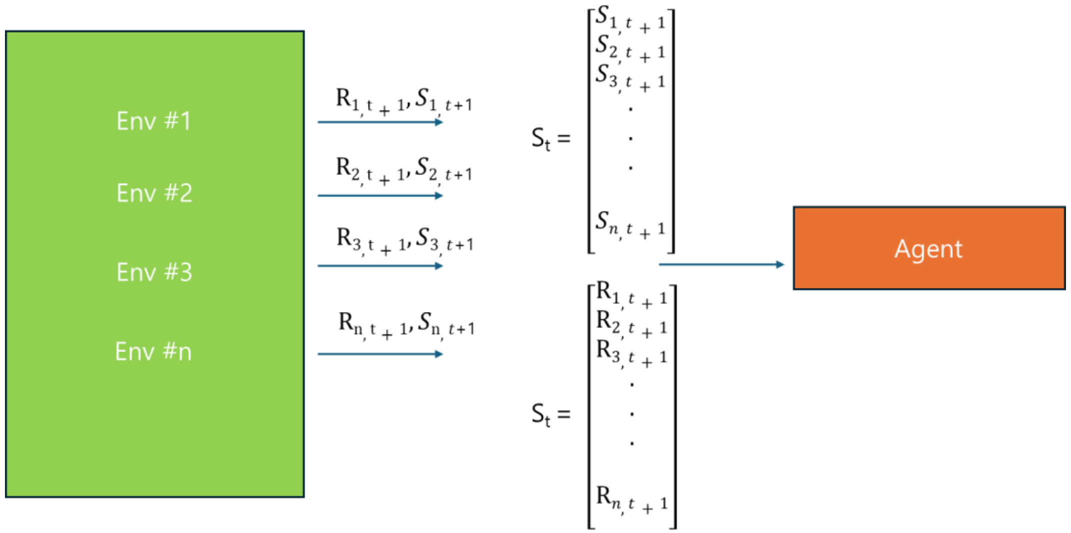
- Successive state tend to be very be very similar to the previous one, and this is called the time correlation problem
- The solution used with Policy Gradient methods



# Parallel learning



# Exploration Strategy

- We want to maintain the agent's exploration but we do not have mechanisms such as $\varepsilon$ -greedy policies

- Now the neural net. is our policy. How to incorporate an exploration mechanism into our neural network?

- We will incentivize the agent to keep the entropy of its policy as high as possible

$$H(X) = -\sum_{x \in X} p(x) \cdot \ln p(x)$$

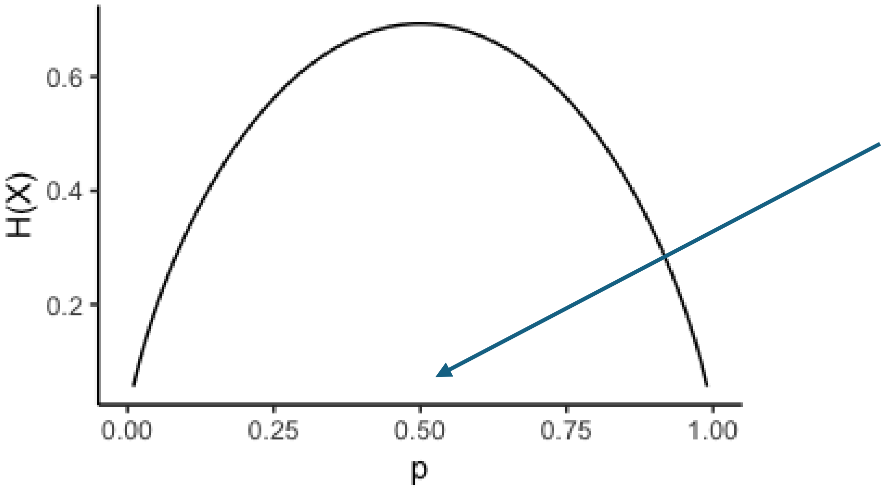
# Exploration Strategy

- What is Entropy? the level of uncertainty of a random variable

e,g.,

if $p(X = x_1) = 1, p(X = x_2) = 0$, $H(X) = -[1 \cdot \ln(1) + 0 \cdot \ln(0)] = 0$

if $p(X = x_1) = 0.5, p(X = x_2) = 0.5$, $H(X) = -[0.5 \cdot \ln(0.5) + 0.5 \cdot \ln(0.5)] \approx 0.6931$




- Max. point where our confidence in our predictions will go down.
- Max. point where it surprises us

# Exploration Strategy

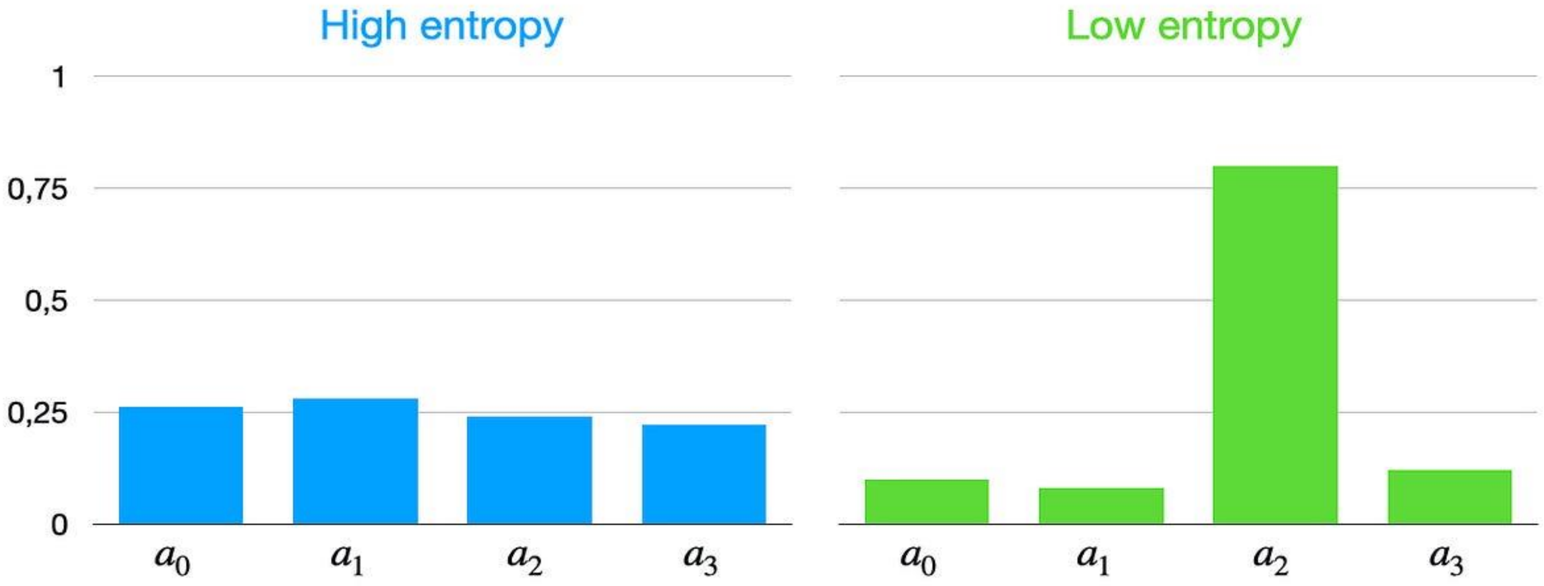
- What is the entropy of a policy?
- Uncertainty in the action to be selected in a state:

$$H_\pi(A_t) = -\sum_{a \in At} \pi(a|St) \cdot \ln \pi(a|St)$$

- Random variable($A_t$) is the action the policy will choose in a state.
- The entropy is computed by multiplying the probability of choosing each action by its logarithm and adding up the results

# Exploration Strategy

- What is the entropy of a policy?
- Imagine that we have 4 actions available



# Exploration Strategy

- What is the entropy of a policy?
- In order for the agent to explore the environment. We must keep the entropy of the policy high

- We add the entropy to the function to be maximized:

$$\theta_{t+1} = \theta_t + \alpha \left[ \gamma^t G_t \ \nabla \ln \pi(a|s,\theta) + \boxed{\beta \nabla H(\pi)} \right]$$

- Advantages in optimizing a policy:

  Exploration, Robustness, Policy refinement

# Parallel REINFORCE learning

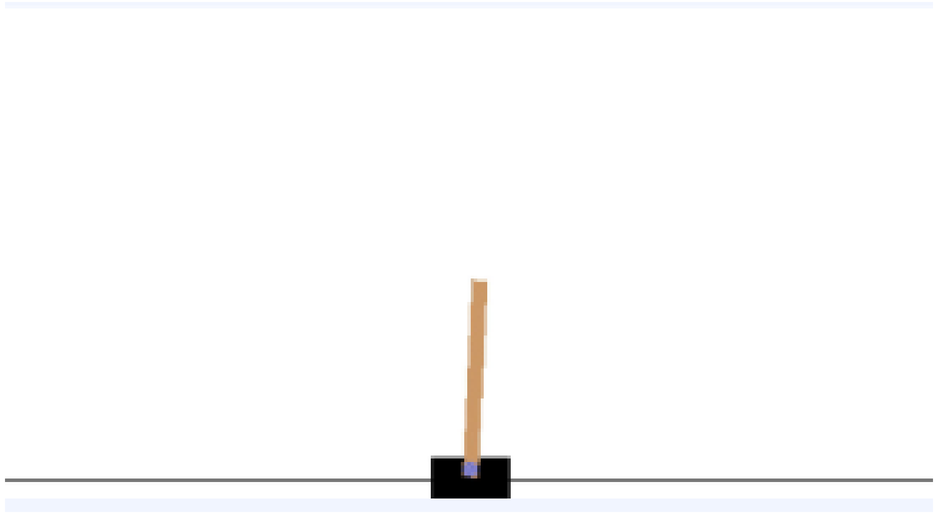
**Algorithm 1** REINFORCE

**Input:** $\alpha$ learning rate, $\gamma$ discount factor.
Initialize parallel environments $E$
Initialize policy parameters $\boldsymbol{\theta}$
**for** episode in 1..N **do**
    Use $\pi(s|\boldsymbol{\theta})$ to collect $|E|$ trajectories: $\boldsymbol{S_0}, \boldsymbol{A_0}, \boldsymbol{R_1}, \cdots, \boldsymbol{R_T}$
    $\boldsymbol{G} = \vec{0}$
    **for** t = T-1..0 **do**
        $\boldsymbol{G} = \boldsymbol{R_t} + \gamma \boldsymbol{G}$
        Compute entropy regularization: $\boxed{\boldsymbol{H_t} = -\sum_a \pi(a|\boldsymbol{S_t}) \ln \pi(a|\boldsymbol{S_t})}$
        $\hat{J}(\boldsymbol{\theta}) = \gamma^t \boldsymbol{G} \ln \pi(\boldsymbol{A_t}|\boldsymbol{S_t}, \boldsymbol{\theta}) \boxed{+ \boldsymbol{H_t}}$
        $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla \hat{J}(\boldsymbol{\theta})$
    **end for**
**end for**

We also try to maximize the entropy of the policy

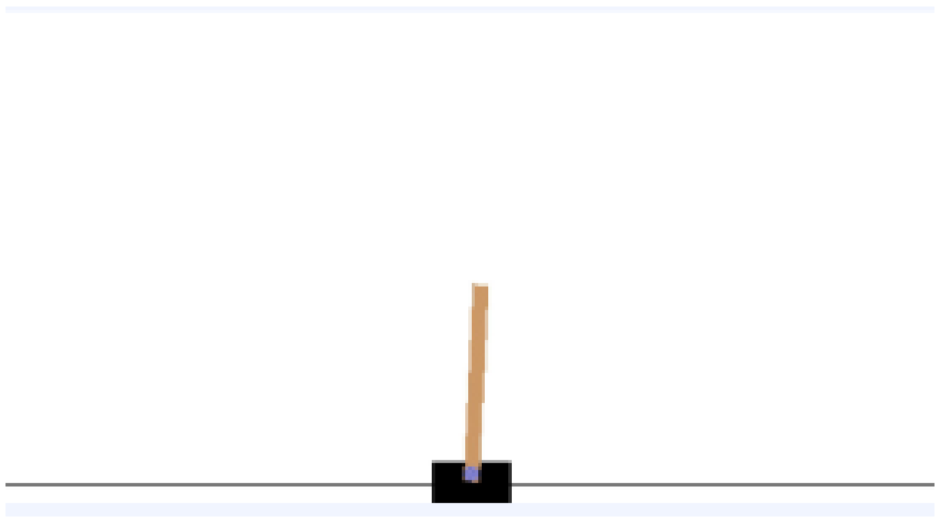
# Code Ex.

- Upload 'REINFORCE_CartPole.ipynb' file onto Colab
- Upload 'utils.py' file onto Colab
- Upload 'parallel_env.py' file onto Colab
- Add 'pip install numpy==1.23.1



# Code Ex.

- Cartpole: move a cart (black) such that it balances a pendulum (brown) without moving too far from the center.
- State: The agent observes current position and velocity of the cart, as well as angle and velocity of the pole (cart position, cart velocity, pole angle, pole angular velocity)
- Action: It can act by pushing the cart to the left (value 0) or to the right (value 1).
- Reward:+1 for every step



# Actor-Critic

# Overview

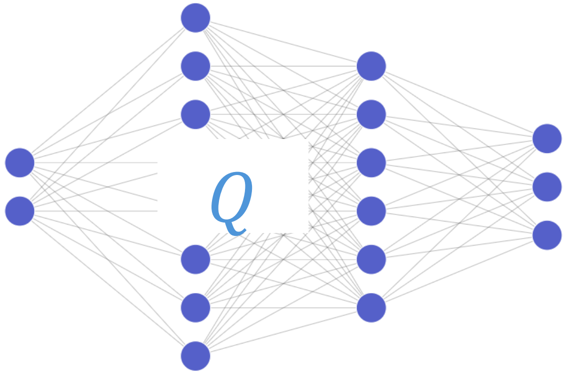
- 'Monte-Carlo' Policy Gradient method
  - $\mathrm{J}(\theta) = \sum_a \pi_\theta(a|s) R_{s,a}$

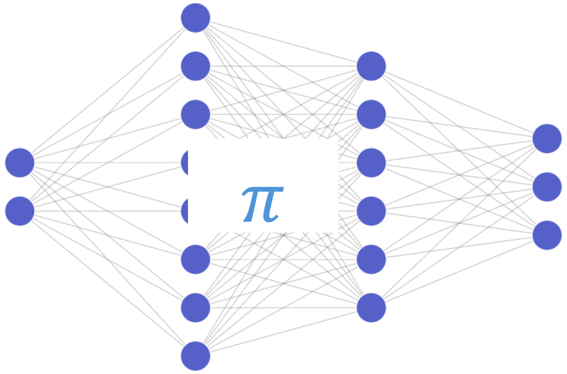
- 'Actor-Critic' method requires Q or V to calculate PG
  - $\mathrm{J}(\theta) = \sum_a \pi_\theta(a|s)\, Q^{\pi\theta}(s,a)$
  - $\mathrm{J}(\theta) = \sum_{s'} \sum_a p(s'|s,a)\, \pi_\theta(a|s)\, [r + \gamma \mathrm{V}^{\pi\theta}(s)]$

# Overview

- TD + Policy Gradient
  - Optimize the neural network during the episode by bootstrapping the value of the next state
  - Recall Temporal Difference (TD) learn at each time step

- Agent has both actor and critic networks



# Advantage Actor-Critic(A2C)

# Overview

Advantage Actor-Critic

$\widehat{\mathrm{A}}\mathrm{dv}_{\pi}(s,a)$ $\pi(a \mid s, \theta)$ $\hat{v}(s \mid \omega)$

# Overview

- Excess return of choosing the $\mathrm{a}$ action instead of following the policy:

$$\mathrm{Adv}_{\pi}(s,a) = q_{\pi}(s, a) - v_{\pi}(s)$$

$$\mathrm{Adv}_{\pi}(s,a) = q_{\pi}(s, a) - \sum_{\mathrm{a}} \pi(\mathrm{a}|\mathrm{s}) \mathrm{q\pi}(\mathrm{s}, \mathrm{a})$$

- We want to reinforce the actions that obtain better results and discourage those that obtain worse results

# Overview

$$\mathrm{Adv}_{\pi}(s,a) = q_{\pi}(s, a) - v_{\pi}(s)$$

$\mathrm{Adv}_{\pi}(s,a) > 0$ :
Taking the action, a is **better** than simply following the policy

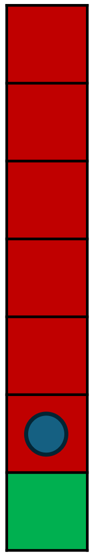
$\mathrm{Adv}_{\pi}(s,a) < 0$:
Taking the action a is **worse** than following the policy

- We can estimate the advantage of an action as:

$$\hat{A}\mathrm{dv}_{\pi}(s,a) = r(s, a) + \gamma\hat{v}_{\pi}(s') - \hat{v}_{\pi}(s)$$

# Update rule: REINFORCE

Assume agent gets -1 reward each time spent

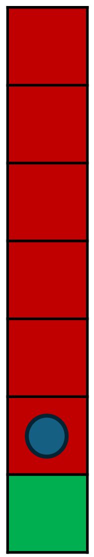


- REINFORCE: $$\theta_{t+1} = \theta_t + \alpha \gamma^t \underline{G_t} \nabla \ln \pi(A_t | S_t, \theta_t)$$

The best is -1

- If $G_t < 0$, discourages the actions that will have negative return
  - If an action led to a negative return, that action was disincentivized even if it was the least bad action available
  - Eventually it gets to the best policy, but a long time to get there
  - Using the return, the agent was not able to make these distinctions

# Update rule: A2C



- A2C: $$\theta_{t+1} = \theta_t + \alpha \gamma^t \underbrace{\widehat{\mathrm{Adv}}_t}_{\mathrm{Adv}(s,\, \downarrow)} \nabla \ln \pi(A_t | S_t, \theta_t)$$

- If $\widehat{\mathrm{Adv}}_t > 0$, reinforces the actions that are better
  - In a bad state, the advantage function helps the agent understand which action is the lesser evil and helps it to reinforce that action
  - If action, ↓ is lesser evil, Adv(s, ↓) is positive
  - It also allows it to see which action is the worst in a very good state and disincentivize it

# SGA

- Where the advantage is the temporal difference error

$$\widehat{\mathrm{Adv}}_t = R_{t+1} + \gamma \boxed{\hat{v}(s_{t+1} \mid \omega)} - \boxed{\hat{v}(s_t \mid \omega)}$$

- Bootstrapped value. We don't need to the end of episode to update the NN
- Bootstrapped value: the estimate of the rest of the rewards that we expect to get starting from the next state

- The baseline eliminates the effect of the state that we're in.
- Subtract this because we want to evaluate only the merit of that action

# REINFORCE vs. A2C

- REINFORCE:
- Used true return: $R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$
- Unbiased empirical return, but with high variance

- A2C:
- Used bootstrapping: $R_{t+1} + \gamma \hat{v}(s_{t+1}) - \hat{v}(s_t)$
  - Learning occurs during the episode
  - we don’t need to the end of episode to update the NN.
- Reducing variance accelerates learning
  - Bootstrapping introduces bias but reduces variance

**Algorithm 1** Advantage Actor-Critic (A2C)

**Input:** $\alpha$ learning rate, $\gamma$ discount factor.
Initialize parallel environments $E$
Initialize policy network parameters $\boldsymbol{\theta}$
Initialize value network parameters $\boldsymbol{w}$
**for** episode in 1..N **do**
    Initialize parallel environments $E$ and obtain initial states $\boldsymbol{S_0}$
    **for** t= 0..T-1 **do**
        $\boldsymbol{A_t} \sim \pi(\boldsymbol{S_t}|\boldsymbol{\theta})$
        Execute the actions in the environments $E$ and obtain $\boldsymbol{R_{t+1}}, \boldsymbol{S_{t+1}}$
        Update the value network with SGD:

$$L(\boldsymbol{w}) = \frac{1}{|E|}\left[\boldsymbol{R_{t+1}} + \gamma v(\boldsymbol{S_{t+1}}|\boldsymbol{w}) - v(\boldsymbol{S_t}|\boldsymbol{w})\right]^2 \tag{1}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla L(\boldsymbol{w}) \tag{2}$$

        Update the policy network with SGA:

$$\boldsymbol{A\hat{d}v_t} = \boldsymbol{R_{t+1}} + \gamma v(\boldsymbol{S_{t+1}}|\boldsymbol{w}) - v(\boldsymbol{S_t}|\boldsymbol{w}) \tag{3}$$

$$\hat{J}(\boldsymbol{\theta}) = \gamma^t \boldsymbol{A\hat{d}v_t} \ln \pi(\boldsymbol{A_t}|\boldsymbol{S_t}, \boldsymbol{\theta}) \boxed{+ \boldsymbol{H_t}} \tag{4}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla \hat{J}(\boldsymbol{\theta}) \tag{5}$$

    **end for**
**end for**

$$H(X) = -\sum_{x \in X} p(x) \cdot \ln p(x)$$

# Code Ex.

- Arcrobot: swing the lower part of a two-link robot up to a given height.
- Observations space: The agent observes current positions and velocities of the joints.
- Action space: It can act by applying positive torque (value 0), no torque (value 1), or negative torque (value 2) only to the joint between the two links.


| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $\cos(\theta_1)$ | $\sin(\theta_1)$ | $\cos(\theta_2)$ | $\sin(\theta_2)$ | $\dot{\theta}_1$ | $\dot{\theta}_2$ |

| Action | 0 | 1 | 2 |
|---|---|---|---|
| Torque | -1 | 0 | 1 |

# Code Ex.

- Upload 'A2C_Acrobot.ipynb' file onto Colab
- Upload 'utils.py' file onto Colab
- Upload 'parallel_env.py' file onto Colab
- Add 'pip install numpy==1.23.1