Lecture overview

Lecture covers chapter 13 in Sutton & Barto [1] and examples from David Silver [2]

1. **Policy-based methods**
   - definition
   - characteristics
   - deterministic vs stochastic policies

2. **Policy gradients**
   - gradient-based estimator
   - Monte Carlo REINFORCE

3. **Actor-critic methods**
   - definition
   - algorithm
   - extensions
**Policy-based methods introduction**

**Definition:** policy-based methods

Last week, we used a function approximator to estimate the value function:

\[ \hat{v}(s, w) \approx v_\pi(s), \]

and for control we estimated \( Q \):

\[ \hat{q}(s, a, w) \approx q_\pi(s, a). \]

This week we will estimate policies:

\[ \pi_\theta(a|s) = P(a|s, \theta) \]

Given a state, what's the distribution over actions?

**Example:** what's the optimal policy?
Policy-based methods characteristics

This approach has the following advantages:

- Can be more efficient than calculating the value function
- Better convergence guarantees
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

And the following disadvantages:

- Converges on local rather than global optimum
- Inefficient policy evaluation with high variance

Example: continuous action spaces
Policy-based methods \textbf{deterministic vs stochastic policies}

**Example:** deterministic vs stochastic policies

Deterministic policy for feature vectors describing the walls around a state:

\[ a = \pi(s, \theta) \]

Stochastic policy:

\[ a \sim \pi(s, \theta) \]

Example from [2].
**Definition:** gradient estimators

While we could optimise $\theta$ for non-differentiable functions using approaches such as genetic algorithms or hill climbing, ideally we want to use a gradient based estimator:

$$L_{PG}(\theta) = \hat{E}_t \left[ \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \hat{A}_t \right]$$

where $\hat{A}_t$ is an estimate of the ‘advantage’ (difference between the return and the state values, you could also replace $\hat{A}_t$ with $q(s, a)$ instead for higher variance). The expectation $\hat{E}_t$ is an empirical average over a finite batch of samples [3]. Typically $\pi$ follows a categorical distribution (softmax) or a Gaussian for continuous action spaces.

Therefore we empirically follow the gradient that maximizes the likelihood of the actions that give the most advantage.
**Definition:** Monte Carlo REINFORCE

REINFORCE estimates the return in the previous equation by using a Monte Carlo estimate [4].

- Initialise some arbitrary parameters $\theta$
- Iteratively sample episodes
- Calculate the complete return from each step
- For each step again, update in the gradient times the sample return

**Algorithm:** Monte Carlo REINFORCE

**PyTorch example:**

```python
# initialise $\theta$ with random values
$\pi = \text{PolicyNetwork}(\theta)$

while (True):
    # sample episode following $\pi$
    $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \sim \pi$

    for $t$ in range($T - 1$):
        $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta)$
```
**Definition:** actor-critic methods

We combine policy gradients with action-value function approximation, using two models that may (optionally) share parameters.

- We use a **critic** to estimate the \( Q \) values:
  \[
  q_w(s, a) \approx q^{\pi_\theta}(s, a)
  \]
- We use an **actor** to update the policy parameters \( \theta \) in the direction suggested by the critic.

**Example:** Actor critic

- I rotate the piece
- Really bad action

Image from freecodecamp.org
**Definition:** actor-critic

Putting this together, actor-critic methods use an approximate policy gradient to adjust the actor policy in the direction that maximises the reward according to the critic:

\[ \theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s, a) q_{w}(s, a) \]

**Algorithm:** Actor-Critic *(PyTorch)*

```
# initialise s, \theta, w randomly
# sample a \sim \pi_{\theta}(a|s)
for t in range(T):
    sample r_t and s' from environment(s,a)
    sample a' \sim \pi_{\theta}(a'|s')
    \theta \leftarrow \theta + \alpha q_{w}(s,a) \nabla_{\theta} \ln \pi_{\theta}(a|s) \quad \# \text{update actor}
    \delta_t = r_t + \gamma q_{w}(s',a') - q_{w}(s,a) \quad \# \text{TD error}
    w \leftarrow w + \alpha \delta_t \nabla_{w} q_{w}(s,a) \quad \# \text{update critic}
    a \leftarrow a', s \leftarrow s'
```
This has introduced the foundations, hopefully now you have a good platform to read about the extensions to this.

1. **Recommended further study (papers & code):**
2. **Recommended further study (theory & STAR):**

Recommended extensions include:
- Advantage actor critic (A3C & A2C) [5]
- Experience replay & prioritised replay [6]
- Proximal policy optimisation [3]
- Rainbow (combining extensions) [7]
Take Away Points

Summary

In summary:

- Policy gradients open up many new extensions
- Choose extensions to reduce variance to stabilise training
- Consider regularisation to encourage exploration
- Going off-policy gives better exploration
- It's possible for the actor and critic to share some lower layer parameters, but be careful about it
- Experience replay can increase sample efficiency (where simulation is expensive)


