ELEC96033: Deep Learning

Carlo Ciliberto and Krystian Mikolajczyk

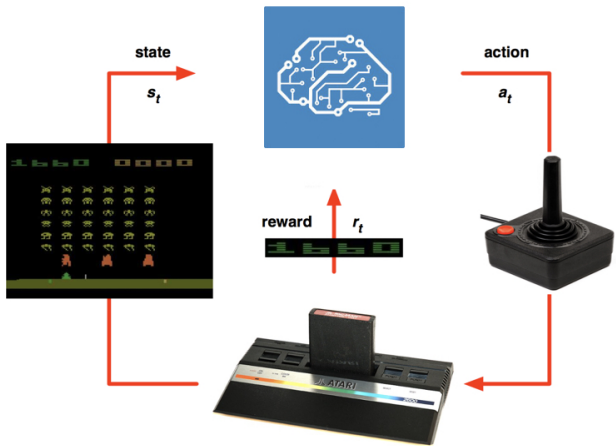
Department of Electrical and Electronic Engineering
Imperial College London

Previously... on Reinforcement Learning

We have focused on Markov Decision Processes (MDPs):

- **Key Spaces:** States $\mathcal{S}$, Actions $\mathcal{A}$ and Rewards $\mathbb{R}$,
- **Transition:** Probability of moving to a new state and reward $P(s', r|s, a)$,
- **Policy:** probability $\pi(a|s)$ of the agent choosing action a,
- **Goal:** find a policy that maximises the return (i.e. sum of all rewards)!

MDP Setting



Previously... on Reinforcement Learning

Scenarios considered:

- P unknown (need to estimate it!).
- $\mathcal{S}$ with both finite or infinite cardinality.
- $\mathcal{A}$ finite cardinality.
- π deterministic.

The Importance of Being Non-deterministic

Sometimes a deterministic policy can only go so far... For instance:

- Ambiguous states (e.g. aliased gridworld).
- Most multi-agent settings.

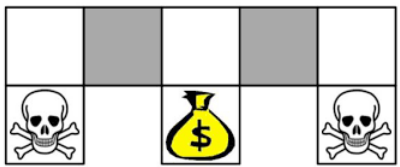
The Importance of Being Non-deterministic

Sometimes a deterministic policy can only go so far... For instance:

- **Ambiguous states (e.g. aliased gridworld).**
- Most multi-agent settings.

State: binary vector recording the position of walls (N, S, W, E)

Gray squares have same state!!



Aliased Gridlworld - Image credits to David Silver

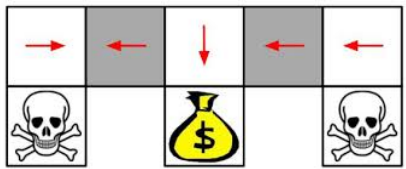
The Importance of Being Non-deterministic

Sometimes a deterministic policy can only go so far... For instance:

- **Ambiguous states (e.g. aliased gridworld).**
- Most multi-agent settings.

State: binary vector recording the position of walls (N, S, W, E)

Gray squares have same state!! A deterministic policy would get stuck



Aliased Gridlworld - Image credits to David Silver

The Importance of Being Non-deterministic

Sometimes a deterministic policy can only go so far... For instance:

- Ambiguous states (e.g. aliased gridworld).
- **Most multi-agent settings.**

If there is not an "always winning" strategy: deterministic == predictable (BAD!)

Today... on Reinforcement Learning

Today:

- P unknown (need to estimate it!).
- $\mathcal{S}$ with both finite or infinite cardinality.
- $\mathcal{A}$ with both finite or infinite cardinality.
- π non-deterministic.

Value and Action-Value Functions

We introduced the value function of a policy $V_\pi : \mathcal{S} \to \mathbb{R}$

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s_0 = s \right]$$

the expected return when following π starting from $s_0 = s$.

Similarly we introduced the action-value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s_0 = s, a_0 = a \right]$$

the expected return when following π after choosing action $a_0 = a$ from $s_0 = s$.

Leveraging, Estimating and Appriximating Q-functions

We observed that access to Q_π is key to attain an optimal deterministic policy.

Greedy policy with respect to Q_π: $\pi'(s) = \text{argmin}_{a \in \mathcal{A}} \; Q_\pi(s, a)$

Policy Improvement: the greedy update strategy

$$\pi_0 \xrightarrow{\text{greedy}} \pi_1 \xrightarrow{\text{greedy}} \pi_2 \to \cdots \to \pi_t \to \cdots \to \pi_*$$

converges to an optimal (deterministic) policy π_*!

Leveraging, Estimating and Appriximating Q-functions (Cont.)

Problem. In RL we do not have access to Q_π! Need to approximate it.

Two simultaneous forms of approximation:

- **Statistic:** use the samples (s, a, r, s', a') to obtain an estimator $\hat{Q}_\pi$ of Q_π,
- **Functional:** choose a model $Q(s, a|\theta)$ to model $\hat{Q}_\pi(s, a)$ (e.g. a neural net).

Several methods:

- Monte Carlo Sampling or Temporal Differences + Policy Improvement
- **SARSA:** "on-policy" method directly improving an ε-greedy policy,
- **Q-Learning:** "off-policy" method keeping track of target and explorarotry policies.

Policy Approximation

Today we plan on directly parametrizing $\pi = \pi_w$ with w a vector of parameters.

Goal. Find the w_* such that π_{w_*} attains maximum expected return.

Benefits:

- Learn non-deterministic policies
- Suited for $\#A = +\infty$ (No need to solve $\text{argmax}_{a \in \mathcal{A}} Q(s, a)$, possibly very hard for large $\mathcal{A}$)

Downsides:

- Convergence to local minima.
- Often high variance.

Policy Gradient

Goal. Find the w_* such that π_{w_*} maximizes the expected return

$$w_* = \underset{w}{\text{argmax}} \; J(w) \qquad\qquad J(w) = \mathbb{E}_{s_0}[V_{\pi_w}(s_0)]$$

with initial state s_0 sampled according to a "starting" distribution $\mu_0(s)$.

Approach. if we choose π_w to be differentiable (almost everywhere), then
we could compute $\nabla_w J(w)$ and perform gradient ascent with respect to w!

Problem. As usual, we do not have access to V_{π_w}
How do we derive $J(w)$ with respect to w to obtain $\nabla_w J(w)$?

One Step Episodes

one-step MDP. Let's start with a simple scenario:

- we sample the initial state $s_0 \sim \mu_0$,
- we sample an action $a \sim \pi(\cdot|s_0)$,
- we collect the next state and reward $(s', r) \sim P(s', r|s, a)$,
- we stop.

Therefore, $V_\pi(s_0) = \mathbb{E}_{a\sim\pi(\cdot|s_0)}\mathbb{E}_{(s',r)\sim P(\cdot|s,a)}[r]$ corresponds to the immediate return.

One Step Episodes (Cont.)

Therefore the expected return of π_w is

$$J(w) = \mathbb{E}_{s_0}\mathbb{E}_{a|s_0}\mathbb{E}_{r|s,a}[r] = \int_{\mathcal{S}}\int_{\mathcal{A}} \mathbb{E}_{r|s,a}[r]\ \pi_w(a|s_0)\ da\ d\mu_0(s_0)$$

By deriving w.r.t. w we obtain

$$\begin{aligned}\nabla_w J(w) &= \int_{\mathcal{S}}\int_{\mathcal{A}} \mathbb{E}_{r|s,a}[r]\ \nabla_w \pi_w(a|s_0)\ da d\mu_0(s_0) \\ &= \mathbb{E}_{s_0}\left[\int_{\mathcal{A}} \mathbb{E}_{r|s,a}[r]\ \nabla_w \pi_w(a|s_0)\ da\right]\end{aligned}$$

...it would seem a complicated object to compute or even just approximate!

The Log-derivative Trick

However, we can leverage the following trick:

$$\nabla_w \pi_w(s|a) = \pi_w(s|a) \frac{\nabla_w \pi_w(s|a)}{\pi_w(s|a)} = \pi_w(s|a) \nabla_w \log \pi_w(s|a)$$

Which leads to the following characterization for the gradient fo $J(w)$

$$\begin{aligned} \nabla_w J(w) &= \mathbb{E}_{s_0} \left[\int_{\mathcal{A}} \mathbb{E}_{r|s,a}[r] \ \nabla_w \pi_w(a|s_0) \ da \right] \\ &= \mathbb{E}_{s_0} \left[\int_{\mathcal{A}} \mathbb{E}_{r|s,a}[r] \ \nabla_w \log \pi_w(a|s_0) \pi_w(a|s_0) \ da \right] = \mathbb{E}_{s_0,a,r} \left[\ \nabla_w \log \pi_w(a|s) \ r \ \right] \end{aligned}$$

The Log-derivative Trick

Why do we care of having $\nabla_w J(w) = \mathbb{E}_{s_0,a,r}\left[\ \nabla_w \log \pi_w(a|s)\ r\ \right]$?

Because now the gradient is an expectation...

- ...of quantities that we can sample: $s_0 \sim \mu_0,\ a \sim \pi_w(\cdot|s),\ r \sim P(\cdot|s,a)$,
- ...of objects that we can compute: $\nabla_w \log \pi_w$ (since we are designing π_w).

Therefore we can:

- Approximate $\nabla_w J(w)$ with $\hat{\nabla}_w J(w)$ (e.g. by sampling and averaging).
- Perform a step of gradient ascent $w \leftarrow w + \alpha \hat{\nabla}_w J(w)$
- Repeat.

The Policy Gradient Theorem

In the general setting (i.e. we do not stop after just the first step)

Theorem. *For any differentiable policy* π_w

$$\nabla_w J(w) = \mathbb{E}_{s\sim\mu_w}\mathbb{E}_{a\sim\pi_w(\cdot|s)}\left[\ \nabla_w \log \pi_w(a|s)\ Q_\pi(s,a)\ \right]$$

where s *are sampled according to the stationary distribution* μ_w *on the states induced by the starting distribution* μ_0 *and the policy* π_w*.*

Main difference with the one-step case: we replaced...

- the immediate reward $\mathbb{E}_{r|s,a}[r]$,
- with the **ideal** action-value function $Q_\pi(s,a)$.

The REINFORCE Algorithm

Initialize. Set initial parameters w for the policy π_w

For each episode obtained following policy π_w: $(s_0, a_0, r_1, \ldots, a_{T-1}, r_T)$

- **Compute** the returns $g_t = \sum_{\tau=t+1}^{T} \gamma^{\tau-1} r_\tau$
- **For** $t = 1, \ldots T-1$
  - Perform one step of gradient ascent $w \leftarrow w + \alpha \nabla_w \log \pi_w(a_t|s_t)\ g_t$
    (As usual, g_t acts as the empirical proxy for Q_π).

Now that we have an algorithm, let us consider some explicit parametrization for π_w...

Example: Softmax Policy

Consider $f_w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a function model (e.g. a Neural Net) parametrized by w.

Softmax Policy. We parametrize $\pi_w(a|s) = \dfrac{e^{f_w(s,a)}}{\int_{\mathcal{A}} e^{f_w(s,a)}\ da}$

Log Gradient. The derivative of $\log \pi_w$ is easy to compute:

$$\nabla_w \log \pi_w(s|a) \quad = \quad \nabla_w \log \frac{e^{f_w(s,a)}}{\int_{\mathcal{A}} e^{f_w(s,a)}\ da} \quad = \quad \nabla_w f_w(s,a) - \nabla_w \log \int_{\mathcal{A}} e^{f_w(s,a)}\ da$$

But now

$$\nabla_w \log \int_{\mathcal{A}} e^{f_w(s,a)}\ da = \frac{\int_{\mathcal{A}} \nabla_w e^{f_w(s,a)}\ da}{\int_{\mathcal{A}} e^{f_w(s,a)}\ da} = \int_{\mathcal{A}} \nabla f_w(s,a) \underbrace{\left[\frac{e^{f_w(s,a)}}{\int_{\mathcal{A}} e^{f_w(s,a)}\ da}\right]}_{\pi_w(a|s)} da = \mathbb{E}_a[\nabla f_w(s,a)]$$

Example: Softmax Policy (Cont.)

Hence, for the softmax policy

$$\pi_w(a|s) = \frac{e^{f_w(s,a)}}{\int_{\mathcal{A}} e^{f_w(s,a)}\ da}$$

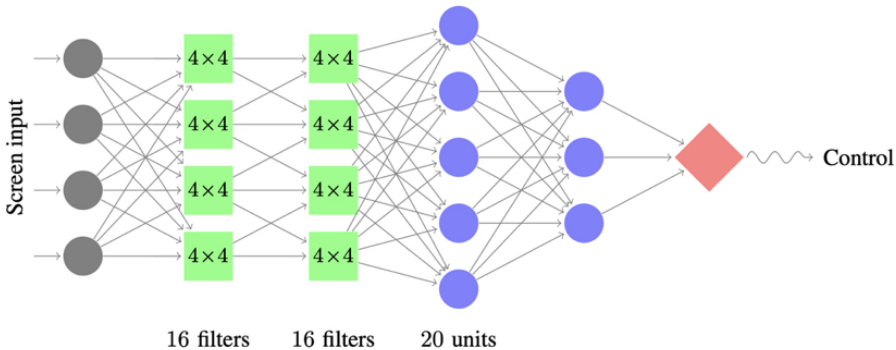
We have

$$\nabla_w \log \pi_w(s|a) \quad = \quad \nabla_w f_w(s,a) - \mathbb{E}_{a\sim\pi_w(\cdot|s)}[\nabla_w f_w(s,a)]$$

Therefore we can sample from $a \sim \pi_w(\cdot|s)$ to get estimates of $\nabla_w \log \pi_w(s|a)$

Large $\mathcal{A}$. We still need to evaluate $f_w(s,a)$ for every a! (Not suitable for large action spaces)

Example: Softmax Policy for Atari Games



Example: Softmax Policy for Atari Games

Video link

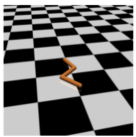
MUJOCO Tasks: Dealing with Infinite Action Spaces

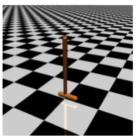
Setting. 3D physical environment (simulator).

Tasks. Controlling agents ("robots") to walk as far as possible...

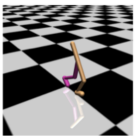
Note. both state and action spaces are "infinite"!!

- **States:**
  Joints
- **Actions:**
  Torques


Swimmer


Hopper


Half Cheetah


Walker


Ant


Simplified Humanoid


Full Humanoid

Example: Gaussian Policy

What if we parametrize $\pi_w(\cdot|s)$ to belong to a family of well known distributions over $\mathcal{A}$?

Gaussian Policy. Suppose $\mathcal{A} = \mathbb{R}^m$. We consider $\pi_w(\cdot|s) = \mathcal{N}(\mu_w(s), \sigma^2 I)$

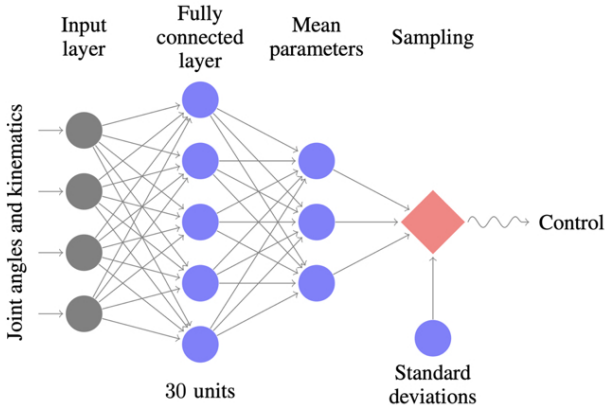
- $\mu_w : \mathcal{S} \to \mathcal{A}$ a "mean" function parametrized by w (e.g. a NN)
- σ the (typically fixed) standard deviation. I the $m \times m$ identity matrix.

Sampling. We can easily sample actions a without searching the whole $\mathcal{A}$!

Log Gradient. Even easier to compute than softmax!

$$\nabla_w \log \pi_w(a|s) \quad = \quad \nabla_w \log \frac{1}{\sqrt{(2\pi\sigma^2)^m}} e^{-\frac{\|a-\mu_w(s)\|^2}{\sigma^2}} \quad = \quad \frac{2}{\sigma^2}(a - \mu_w(s))\, \nabla_w \mu_w(s)$$

Example: Gaussian Policy for MUJOCO



Example: Gaussian Policy for MUJOCO

Video link

Issues with Policy Gradient

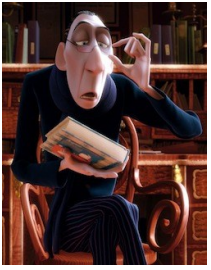
Two main issues with policy gradient:

- Choosing the step size α is critical:
  - **Supervised Learning:** for not optimal α a step could lead to some mistake but then the next step can fix it (no dynamic in training data)
  - **Reinforcement Learning:** a wrong step leads to a bad policy from which bad new data is sampled, which leads to even worse policies, etc...
- **High variance:** typically REINFORCE with Monte Carlo sampling takes millions of iterations to converge to a stable policy (even for very small problems!).

Reducing the Variance by Introducing a Critic

How can we reduce the variance?

With a function "evaluating" the agents' choices: a critic!



Go back to the previous class: **Function Approximation**

$Q(s, a|\theta)$ could be a more stable estimate for $Q_\pi(s, a)$ than the samples g_t.

Actor-Critic

We have two sets of parameters

- **Actor:** the agent policy π_w parametrized by w,
- **Critic:** the critic $Q(s, a|\theta)$ parametrized by θ.

An Actor-Critic paradigm proceeds as follows:

- Follow the policy update in the direction suggested by the critic

$$\nabla_w J(w) \approx \mathbb{E}_{s\sim\mu_w}\mathbb{E}_{a\sim\pi_w(\cdot|s)}\left[\nabla_w \log \pi_w(a|s)\ Q(s, a|\theta)\right]$$

- Estimate the $Q(s, a|\theta)$ to approximate Q_{π_w} (e.g. by **Time Differences**)

Action-Value Actor Critic (A2C)

Initialize. The actor w and the critic θ.

Sample: $s \sim \mu_0$ and $a \sim \pi_w(\cdot|s)$.

For every step

- **Sample** the reward and new state $(s', r) \sim P(s', r|s, a)$,
- **Sample** a new action $a' \sim \pi_w(\cdot|s')$,
- **Set** $\Delta_{TD} = r + \gamma Q(s', a'|\theta) - Q(s, a|\theta)$
- **Update** $w = w + \alpha_1 \nabla_w \log \pi_w(a|s)\ Q(s, a|\theta)$
- **Update** $\theta = \theta + \alpha_2\ \Delta_{TD} \nabla_\theta Q(s, a|\theta)$
- **Update** $s = s'$ and $a = a'$

Summary

This class. We have discussed:

- The advantages/disadvantages of policy approximation methods,
- How to approximate the gradient of the expected return w.r.t. a policy π_w,
- Leveraging function approximation to obtain a critic,
- How to reduce the variance of policy gradient methods.

Next class. we will conclude our tour of DL by addressing hyperparameter optimization via:

- Meta-learning (a.k.a. Learning to Learn).

. . . But the amazing world of Reinforcement Learning does not end here!

Video link