CS6482 Deep RL

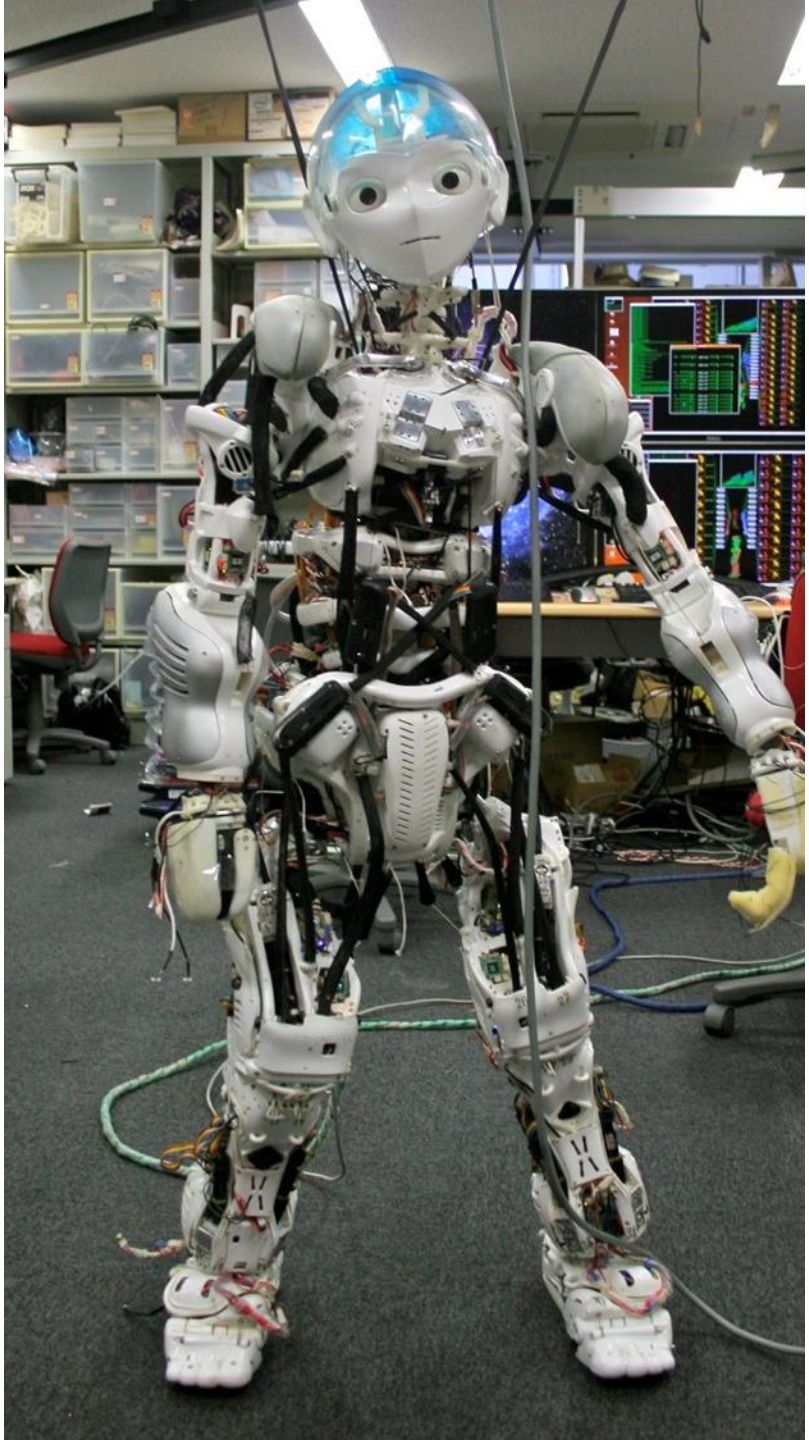
J: Policy Gradient Part2

J.J. Collins

Dept. of CSIS

University of Limerick

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH



Objectives

- Exploring Policy Gradient

Outline

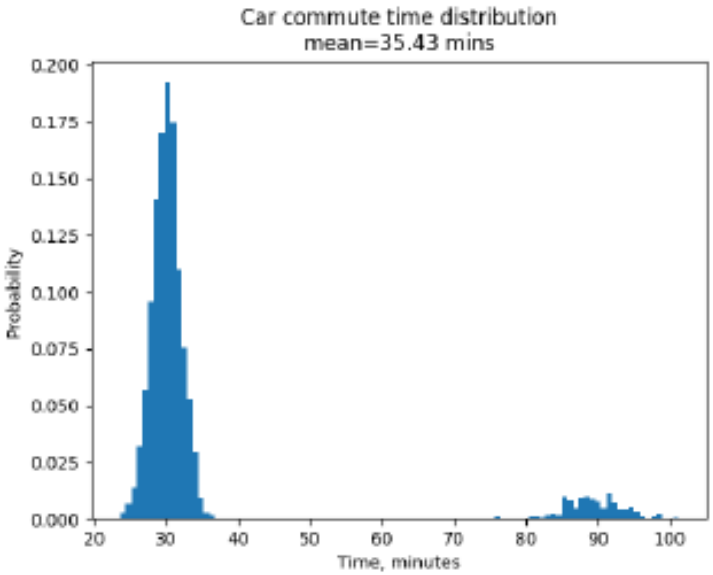
Summary of

Chapter 18 in Gueron. Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2nd Edition, O'Reilly. 2019.

Chapter 13 in Sutton and Barto. Reinforcement Learning: an Introduction, 2nd Edition. The MIT Press. 2018.

Values v Distributions

- Bellemare, Dabney and Munos. A Distributional Perspective on Reinforcement (2017).
  - Replace Q values with more generic Q-value probability distribution.
- Both the Q-learning and value iteration methods work with the values of states (V) or actions (Q) represented as numbers
  - show how much total reward we can achieve from state or action.
- Can you squeeze all future possible reward into one number?
- In complicated environments, the future could be stochastic, giving us different values with different probabilities.
- The commuter scenario:
  - Most of the time, the traffic isn't that heavy and it takes 30 minutes approx. to reach your destination.
  - It's not exactly 30 minutes.
  - From time to time, something happens ref traffic volumes/flow, it takes you three-times longer to get to work.
- The probability of your commute time can be represented as distribution of the 'commute time' random variable and is shown in the following chart.



UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Why Policy Methods?

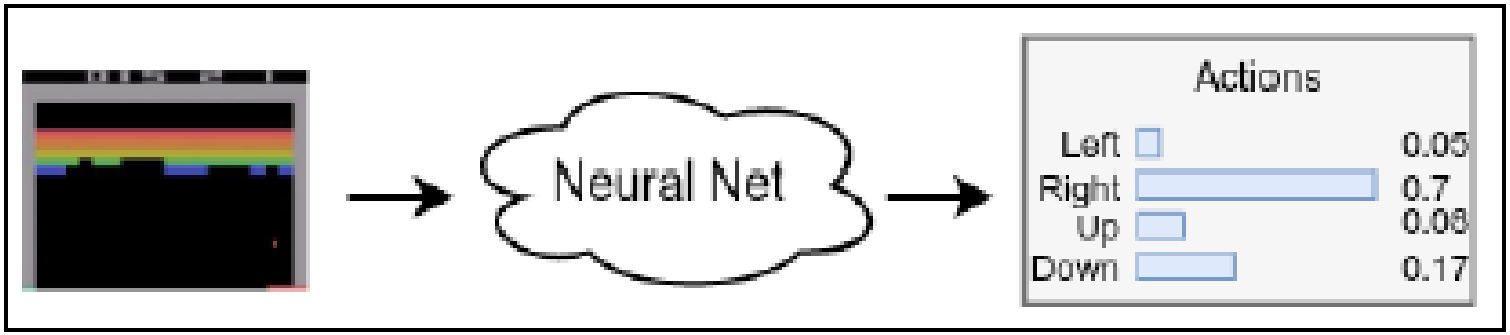
- In the case of Atari, the action space was discrete
- What about continuous action space?
- And estimating a distribution rather than a scalar!
- In such cases, better to work with policies directly
- And particularly for environments with stochasticity

Learning a Policy

- Brute force Search directly in Policy space
  - Hill Climbing
  - Simulated Annealing
    - At each step, the simulated annealing heuristic considers some neighboring state *s** of the current state *s*, and probabilistically decides between moving the system to state *s** or staying in-state *s*.
    - Determined by energies of the two states and a global time varying parameter called Temperature
      - Annealing schedule dictates a gradual reduction in T
    - These probabilities ultimately lead the system to move to states of lower energy.
    - Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.
- Genetic Algorithms
  - Where each candidate solution is a policy
  - A mapping from states to actions

Learning a Policy

- Or use a function approximator which returns
- A discrete numeric output that is an action to be executed, or
  - Small adjustment of weights can lead to selection of different actions
- A probability distribution for the mutually exclusive actions
  - A small adjustment of weights will usually lead to a small change in the distribution



Cross Entropy

- Cross entropy originated from information theory.
- Suppose you want to efficiently transmit information about the weather.
- If there are eight options (sunny, rainy, etc.), you could encode each option using three binary bits
  - $2^3 = 8$.
- However, if it will be sunny almost every day, it would be much more efficient to code “sunny” on just one bit (0) and the other seven options on four bits (starting with a 1).
- Cross entropy measures the average number of bits you actually send per option.
- If your assumption about the weather is perfect, cross entropy will be equal to the entropy of the weather itself (i.e., its intrinsic unpredictability).
- But if your assumptions are wrong (e.g., if it rains often), cross entropy will be greater by an amount called the *Kullback–Leibler (KL) divergence*.

Policy Gradient

- Define Policy Gradient PG – a loss function, as
- $\mathcal{L} = -Q(s,a)\log\pi(a|s)$
- PG defines the direction in which we need to change our network's parameters to improve the policy in terms of the accumulated total reward.
- The scale of the gradient is proportional to the value of the action taken, which is Q(s,a)
- Intuitively, increase the probability of actions that give "good" total reward, and decrease probability of actions with bad outcomes.
- How are gradient scales Q(s,a) calculated?

The Cross Entropy Method

- The formula of PG that we've just seen is used by most of the policy-based methods.
- The Cross-Entropy Method
  - play several episodes
  - calculated the total reward for each of them,
  - train on transitions from episodes with a better-than-average reward.
- This training procedure is the PG method with Q(s, a) = 1 for actions from good episodes (with large total reward) and Q(s, a) = 0 for actions from worse episodes.
- Cross-entropy method is model-free, policy-based, and on-policy:
  - It doesn't build any model of the environment; it just says to the agent what to do at every step
  - It approximates the policy of the agent
  - It requires fresh data obtained from the environment

Cross Entropy Method

1. Play *N* number of episodes using our current model and environment.
2. Calculate the total reward for every episode and decide on a reward boundary. Usually, we use some percentile of all rewards, such as 50th or 70th.
3. Throw away all episodes with a reward below the boundary.
4. Train on the remaining "elite" episodes using observations as the input and issued actions as the desired output.
5. Repeat from step 1 until we become satisfied with the result.

Cross Entropy Method

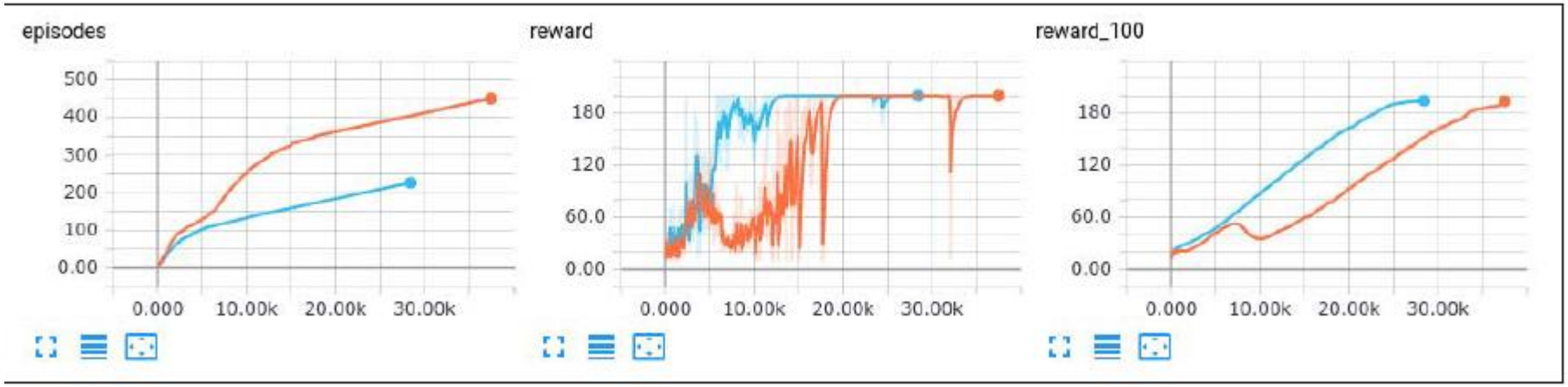
- The cross-entropy method works even with those simple assumptions, but the obvious improvement will be to use Q(s, a) for training instead of just 0 and 1.
- So why should it help? The answer is a more fine-grained separation of episodes.
- For example, transitions of the episode with the total reward = 10 should contribute to the gradient more than transitions from the episode with the reward = 1.
- The second reason to use Q(s, a) instead of just 0 or 1 constants is to increase probabilities of good actions in the beginning of the episode and decrease the actions closer to the end of episode.

REINFORCE

1. Initialize the network with random weights
2. Play *N* full episodes, saving their *(s, a, r, s')* transitions
3. For every step t of every episode k, calculate the discounted total reward for subsequent steps $Q_{k,t} = \sum_{i=0} \gamma^2 r_i$
4. Calculate the loss function for all transitions $\mathcal{L} = -\sum_{k,t} Q_{k,t} \log \pi(s_{k,t}, a_{k,t})$
5. Perform SGD update of weights minimizing the loss
6. Repeat from step 2 until converged

The Cart Pole in PyTorch

- *The Cross-Entropy Method*, the cross-entropy method required about 40 batches of 16 episodes each to solve the CartPole environment, which is 640 episodes in total.
- The REINFORCE method is able to do the same in less than 300 episodes, which is a nice improvement.
- Compared to DQN (orange plots)



UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Issue 1: Full episodes

First of all, we still need to wait for the full episode to complete before we can start training.

Even worse, both REINFORCE and cross-entropy behave better with more episodes used for training (just from the fact that more episodes mean more training data, which means more accurate PG).

This situation is fine for short episodes in the CartPole, when in the beginning, we can barely handle the bar for more than 10 steps, but in Atari, it is completely different: every episode can lasts hundreds or even thousands of frames.

Issue 1: Full episodes

- First of all, we still need to wait for the full episode to complete before we can start training.
- Even worse, both REINFORCE and cross-entropy behave better with more episodes used for training (just from the fact that more episodes mean more training data, which means more accurate PG).
- This situation is fine for short episodes in the CartPole, when in the beginning, we can barely handle the bar for more than 10 steps, but in Atari, it is completely different: every episode can lasts hundreds or even thousands of frames.

Issue 1: Full episodes

- It’s equally bad from the training perspective, as our training batch becomes very large and from sample efficiency, when we need to communicate with the environment a lot just to perform a single training step.
- To overcome this, ask our network to estimate V(s) and use this estimation to obtain Q. This approach is the basis for Actor-Critic methods, which is the most popular method from the PG family.

Issue 2: Variance

- The PG formula $\nabla J \approx \mathrm{E}\left[Q_{s,a} \nabla \pi \, log(a|s)\right]$
- → gradient proportional to the discounted reward from the given state.
- However, the range of this reward is heavily environment-dependent.
- In the CartPole environment we're getting the reward of 1 for every timestamp we're holding the pole vertically. If we can do this for five steps, we'll get total (undiscounted) reward of 5.
- If our agent is smart and can hold the pole for, say, 100 steps, the total reward will be 100.
- The difference in value between those two scenarios is 20 times, which means that the scale between gradients of unsuccessful samples will be 20 times lower than that for more successful ones.
- Such a large difference can seriously affect our training dynamics, as one lucky episode will dominate in the final gradient.

Issue 2: Variance

- In mathematical terms, our PGs have high variance and we need to do something about this in complex environments, otherwise, the training process can become unstable.
- The usual approach to handle this is subtracting a value called **baseline** from the Q.
- The possible choices of the baseline are as follows:

1. Some constant value, which normally is the mean of the discounted rewards
2. The moving average of the discounted rewards
3. Value of the state V(s)

Issue 3: Exploration

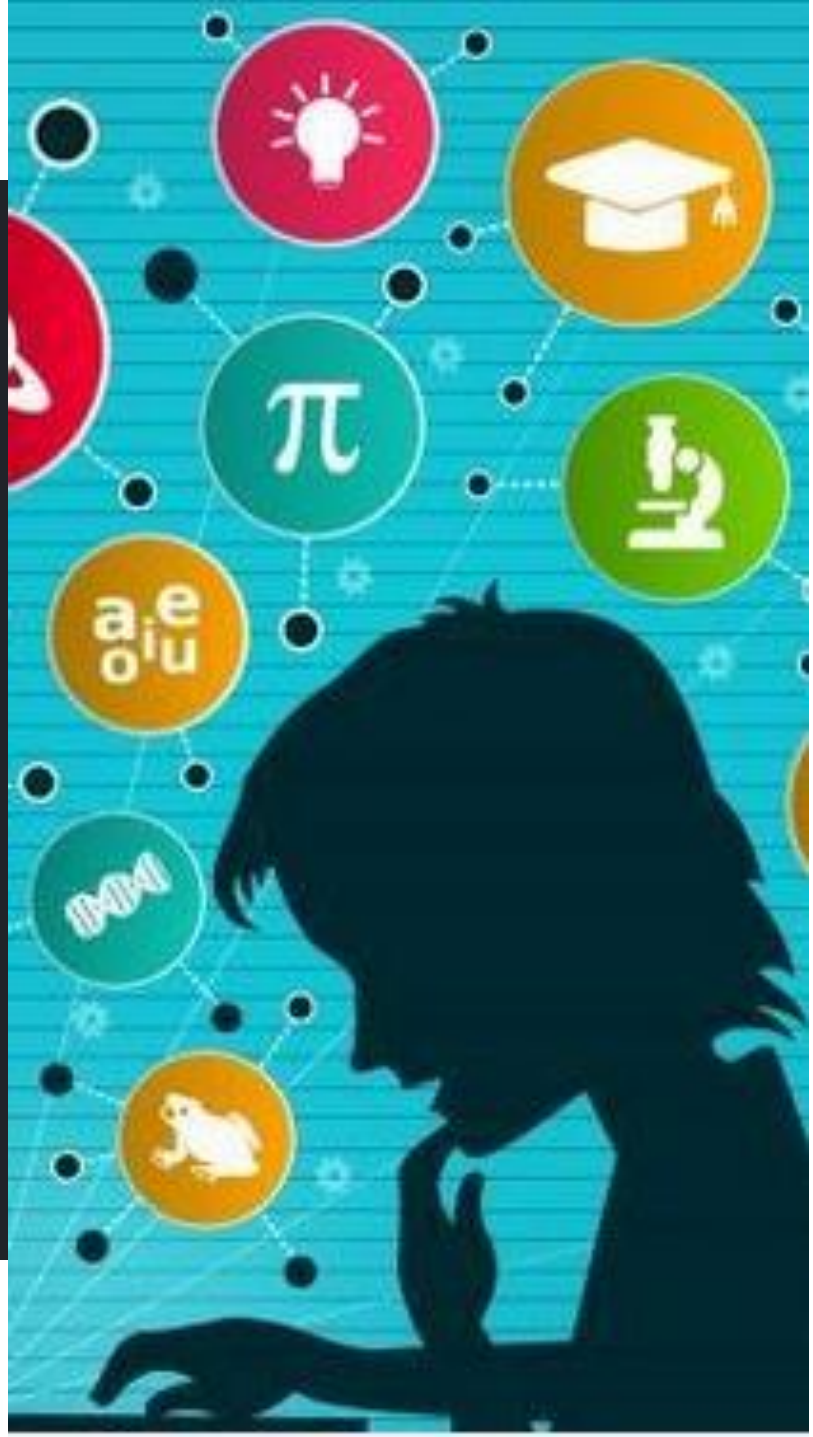
- Even with the policy represented as probability distribution, there is a high chance that the agent will converge to some locally-optimal policy and stop exploring the environment.
- In DQN, we solved this using epsilon-greedy action selection with probability epsilon
  - the agent took some random action instead of the action dictated by the current policy.
- PG uses entropy bonus.
- The entropy is a measure of uncertainty in some system
- $H(\pi) = -\sum \pi(a|s) \log \pi\,(a|s)$
- Entropy becomes minimal when our policy has 1 for some action and 0 for all others, which means that the agent is absolutely sure what to do.
- To prevent our agent from being stuck in the local minimum, subtract the entropy from the loss function, punishing the agent for being too certain about the action to take.

Issue 4: Correlation Between Samples

- training samples in one single episode are usually heavily correlated, which is bad for SGD training.
- In the case of DQN, we solved this issue by having a large replay buffer with 100k-1M observations that we sampled our training batch from.
- This solution is not applicable to the PG family anymore, due to the fact that those methods belong to the on-policy class.
- To solve this, parallel environments are normally used.
- The idea is simple: instead of communicating with one environment, we use several and use their transitions as training data.

Summary

- Policy methods directly optimise what we care about: behavior.
  - The value methods such as DQN are doing the same indirectly, learning the value first and providing to us policy based on this value.
- Policy methods are on-policy and require fresh samples from the environment.
  - The value methods can benefit from old data, obtained from the old policy, human demonstration, and other sources.



Summary

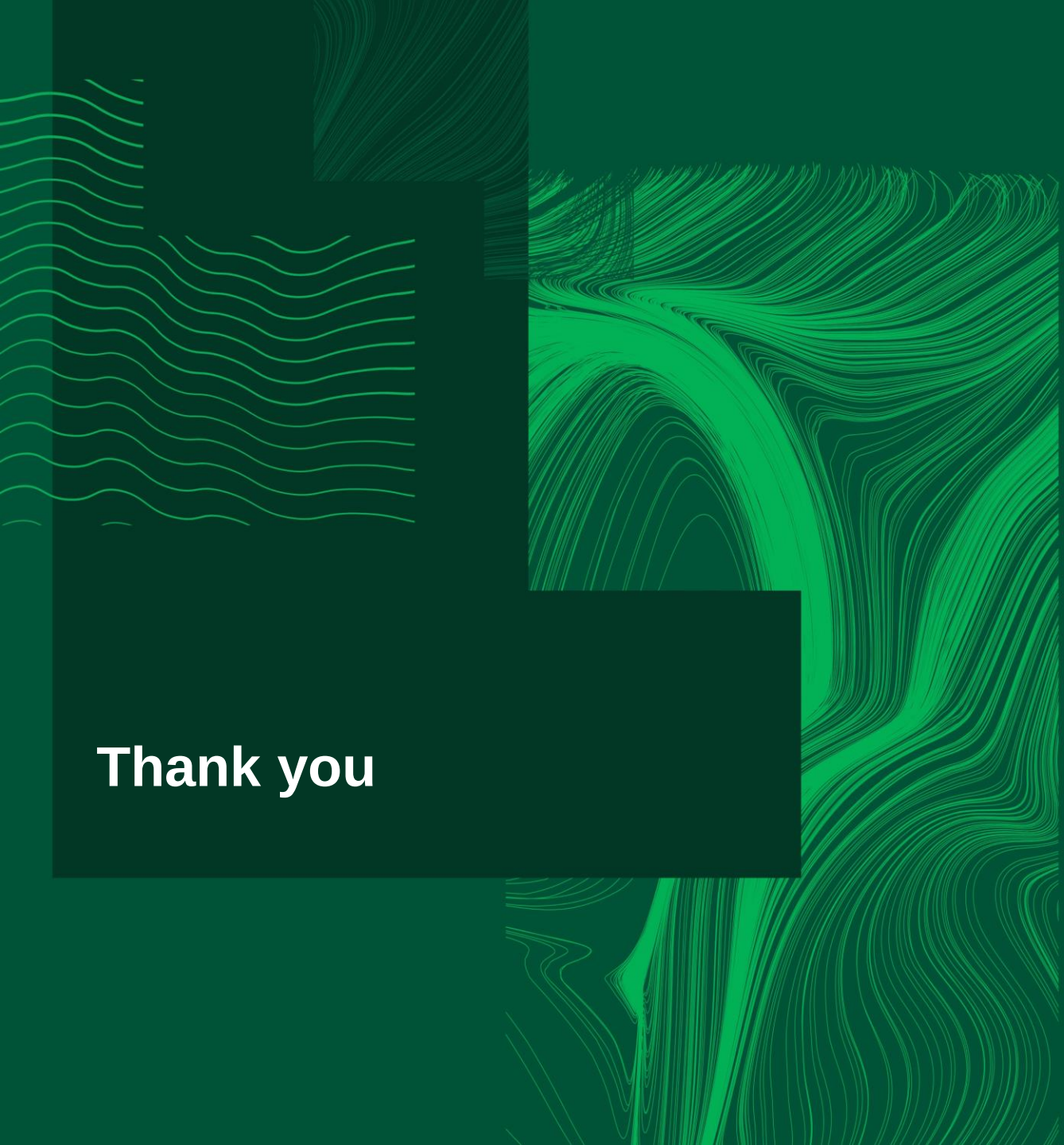
- Policy methods are usually less sample-efficient, which means they require more interaction with the environment.
- The value methods can benefit from the large replay buffers.
- However, sample efficiency doesn't mean that value methods are more computationally efficient and very often it's the opposite.
- In the above example, during the training, we need to access our NN only once, to get the probabilities of actions.
- In DQN, we need to process two batch of states: one for the current state and another for the next state in the Bellman update.

Homework

1. **Work through first half of chapter 18 in Geron**

2. **Implement PG for CartPole**

Thank you



UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 T9PX,
Ireland.

Ollscoil Luimnigh,
Luimneach,
V94 T9PX, Éire.

+353 (0) 61 202020

ul.ie