Lecture Outline

- Problems with experience replay
- Asynchronous methods for deep RL
- Deep actor-critic methods
- Deep deterministic policy gradient
- Debugging deep RL
Recap: DQN
Recap: Deep Q-Network (DQN)

Deep Q-Network:
- Approximate state-action values using a neural network
- Stabilise training by:
  - Sampling batches from experience replay buffer
  - Using separate network to compute target values
- Further optimisation by:
  - Double DQN to reduce overestimation of Q-values
  - Prioritised replay to increase likelihood of sampling valuable experience
Problems of DQN

- Requires large storage for replay buffer (e.g. Atari game requires ≈56GB, cannot fit in a modern PC)
- Use of replay buffer requires off-policy method (why?)
- Not straightforward handling of multi-step returns (why?)
Problems of DQN

- Requires large storage for replay buffer (e.g. Atari game requires \( \approx 56 \) GB, cannot fit in a modern PC)
- Use of replay buffer requires off-policy method (why?)
- Not straightforward handling of multi-step returns (why?)

Is there an alternative approach to break correlations of consecutive experience?
Asynchronous Training
Asynchronous Framework

Create

$n$ parallel "worker" threads with own environment copies and shared global network

Each worker interacts independently with its environment

Asynchronous updates: Periodically, each worker updates the global network parameters based on its local experiences.
• Create $n$ parallel “worker” threads with own environment copies and shared global network
Asynchronous Framework

- Create \( n \) parallel “worker” threads with own environment copies and shared global network
- Each worker interacts independently with its environment
Asynchronous Framework

- Create $n$ parallel “worker” threads with own environment copies and shared global network
- Each worker interacts independently with its environment
- **Asynchronous updates:** Periodically, each worker updates the global network parameters based on its local experiences
Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples
  ⇒ Means we don’t need replay buffer!
Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples → Means we don’t need replay buffer!
- Better handling of sequential data: can use on-policy and multi-step returns
Asynchronous updating is another way of breaking correlation in samples
⇒ Means we don’t need replay buffer!

Better handling of sequential data: can use on-policy and multi-step returns

_runs on normal multi-threaded CPUs_
Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples
  \( \Rightarrow \) Means we don’t need replay buffer!

- Better handling of sequential data: can use on-policy and multi-step returns

- Runs on normal multi-threaded CPUs

- Alternative: parallel, vectorised environments
Asynchronous 1-Step Q-Learning [Mnih et al., 2016]

\[ \text{repeat} \]

Take action \( a \) with \( \epsilon \)-greedy policy based on \( Q(s, a; \theta) \)

Receive new state \( s' \) and reward \( r \)

\[
y = \begin{cases} 
  r & \text{for terminal } s' \\
  r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s'
\end{cases}
\]

Accumulate gradients wrt \( \theta \): \( d\theta \leftarrow d\theta + \frac{\partial (y-Q(s,a;\theta))^2}{\partial \theta} \)

\( s = s' \)

\( T \leftarrow T + 1 \) and \( t \leftarrow t + 1 \)

\[ \text{if } T \mod I_{\text{target}} == 0 \text{ then} \]

\[ \text{Update the target network } \theta^- \leftarrow \theta \]

\[ \text{end if} \]

\[ \text{if } t \mod I_{\text{AsyncUpdate}} == 0 \text{ or } s \text{ is terminal then} \]

\[ \text{Perform asynchronous update of } \theta \text{ using } d\theta. \]

\[ \text{Clear gradients } d\theta \leftarrow 0. \]

\[ \text{end if} \]

\[ \text{until } T > T_{\text{max}} \]
More workers (parallel threads) lead to faster learning
More workers (parallel threads) lead to faster learning

- Workers explore different parts of the environment
- Workers can use different exploration policies (e.g. $\epsilon$-values)
Deep Actor-Critic
Recap: Actor-Critic Algorithm

Objective: Find parameters $\theta$ which maximise $J = V^{\pi_\theta}(s)$
Recap: Actor-Critic Algorithm

Objective: Find parameters $\theta$ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla \theta J$ using the policy gradient theorem:

$$\nabla \theta J = \mathbb{E}_{s \sim d(s), a \sim \pi}[R \nabla \theta \log \pi_\theta(a|s)]$$
Recap: Actor-Critic Algorithm

Objective: Find parameters $\theta$ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla J$ using the policy gradient theorem:
  \[
  \nabla J = \mathbb{E}_{s \sim d(s), a \sim \pi} [R \nabla \log \pi_\theta(a|s)]
  \]

- Approximate the return $R$ using a critic $\hat{V}_w$ with parameters $w$
  \[
  \nabla J = \mathbb{E}_{s \sim d(s), a \sim \pi} [(r + \hat{V}_w(s')) \nabla \log \pi_\theta(a|s)]
  \]

Train the critic by minimising the TD-error
Recap: Actor-Critic Algorithm

Objective: Find parameters $\theta$ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla J$ using the policy gradient theorem:
  \[ \nabla J = E_{s \sim d(s), a \sim \pi}[R \nabla \log \pi_\theta(a|s)] \]

- Approximate the return $R$ using a critic $\hat{V}_w$ with parameters $w$
  \[ \nabla J = E_{s \sim d(s), a \sim \pi}[(r + \hat{V}_w(s'))\nabla \log \pi_\theta(a|s)] \]

Train the critic by minimising the TD-error

- Subtract a baseline function in order to reduce the variance of the estimation
  \[ \nabla J = E_{s \sim d(s), a \sim \pi}[(r + \hat{V}_w(s') - \hat{V}_w(s))\nabla \log \pi_\theta(a|s)] \]
Asynchronous Advantage Actor-Critic (A3C) [Mnih et al., 2016]

repeat
   Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
   Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
   $t_{start} = t$
   Get state $s_t$
   repeat
      Perform $a_t$ according to policy $\pi(a_t|s_t; \theta')$
      Receive reward $r_t$ and new state $s_{t+1}$
      $t \leftarrow t + 1$
      $T \leftarrow T + 1$
   until terminal $s_t$ or $t - t_{start} \geq t_{max}$

   $R = \begin{cases} 
   0 & \text{for terminal } s_t \\
   V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases}$

   for $i \in \{t - 1, \ldots, t_{start}\}$ do
      $R \leftarrow r_i + \gamma R$
      Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$
      Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$
   end for
   Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
Entropy Regularisation

- **Entropy** of a stochastic policy

\[ H[\pi(a|s)] = \mathbb{E}_{a \sim \pi(a|s)}[- \log \pi(a|s)] = - \sum_a \pi(a|s) \log \pi(a|s) \]

The entropy is maximised when the policy distribution is uniform.
Entropy Regularisation

• **Entropy** of a stochastic policy

\[ H[\pi(a|s)] = E_{a \sim \pi(a|s)}[-\log \pi(a|s)] = -\sum_a \pi(a|s) \log \pi(a|s) \]

The entropy is maximised when the policy distribution is uniform

• Add an entropy regularisation in A3C

\[ L_{actor} = -(R - V(s)) \log \pi(a|s) - \beta H[\pi(a|s)] \]

Encourage exploration by maximising entropy while minimising policy loss
Deep Deterministic Policy Gradient
• Can we use A3C?
  - *How?*
• Can we use A3C?
  - *How?*

• Can we use DQN and discretize the action spaces?
  - *What is the disadvantage?*
• Can we use A3C?
  - How?

• Can we use DQN and discretize the action spaces?
  - What is the disadvantage?

• How do we compute $\text{argmax}_a Q(s, a)$ in continuous action spaces?
Deterministic Policy Gradient

- Extension of policy gradient to *deterministic* policies $\mu : S \to \mathbb{R}^{|A|}$

$$\nabla_{\theta\mu} V(s_0) = \mathbb{E}_{s \sim d(s)} \left[ \nabla_a Q(s, \mu(s|\theta^\mu)|\theta^Q) \nabla_{\theta\mu} \mu(s) \right]$$
Deterministic Policy Gradient

- Extension of policy gradient to deterministic policies $\mu : S \rightarrow \mathbb{R}^{|A|}$
  \[
  \nabla_{\theta^\mu} V(s_0) = \mathbb{E}_{s \sim d(s)} \left[ \nabla_a Q(s, \mu(s|\theta^\mu)|\theta^Q) \nabla_{\theta^\mu} \mu(s) \right]
  \]
- It assumes continuous actions. The actor loss is:
  \[
  L_a = -Q(s, \mu(s|\theta^\mu))
  \]
Deterministic Policy Gradient

- Extension of policy gradient to \textit{deterministic} policies $\mu : S \to \mathbb{R}^{|A|}$
  \[
  \nabla_{\theta^\mu} V(s_0) = \mathbb{E}_{s \sim d(s)} \left[ \nabla_a Q(s, \mu(s|\theta^\mu)|\theta^Q) \nabla_{\theta^\mu} \mu(s) \right]
  \]
- It assumes continuous actions. The actor loss is:
  \[
  L_a = -Q(s, \mu(s|\theta^\mu))
  \]
- Can be extended to discrete environments using mechanisms that produce differentiable samples from categorical distribution (e.g. \textit{Gumbel-Softmax})
Deterministic Policy Gradient

- Extension of policy gradient to deterministic policies $\mu : S \to \mathbb{R}^{|A|}$

$$\nabla_{\theta^\mu} V(s_0) = \mathbb{E}_{s \sim d(s)} \left[ \nabla_a Q(s, \mu(s|\theta^\mu)|\theta^Q) \nabla_{\theta^\mu} \mu(s) \right]$$

- It assumes continuous actions. The actor loss is:

$$L_a = -Q(s, \mu(s|\theta^\mu))$$

- Can be extended to discrete environments using mechanisms that produce differentiable samples from categorical distribution (e.g. Gumbel-Softmax)

- Train the critic by minimising the TD-error:

$$L_c = \frac{1}{2} \left( r + \gamma Q_{target}(s', \mu_{target}(s'|\theta^{\mu'})|\theta^{Q'}) - Q(s, a|\theta^Q) \right)^2$$
Deterministic Policy Gradient – Diagram
• Q-learning uses $\epsilon$-greedy

• A3C samples from a Softmax distribution and exploration is encouraged through an entropy-based term in the actor’s loss

• DDPG adds random noise to the output of the actor (e.g. Gaussian noise, Ornstein–Uhlenbeck noise)

$$a = \mu(s|\theta^\mu) + \mathcal{N}$$
for episode = 1, M do
  Initialize a random process \( \mathcal{N} \) for action exploration
  Receive initial observation state \( s_1 \)
  for \( t = 1, T \) do
    Select action \( a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \) according to the current policy and exploration noise
    Execute action \( a_t \) and observe reward \( r_t \) and observe new state \( s_{t+1} \)
    Store transition \((s_t, a_t, r_t, s_{t+1})\) in \( R \)
    Sample a random minibatch of \( N \) transitions \((s_i, a_i, r_i, s_{i+1})\) from \( R \)
    Set \( y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})) \)
    Update critic by minimizing the loss: \( L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \)
    Update the actor policy using the sampled policy gradient:
    \[
    \nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^\mu)|_{s_i}
    \]
    Update the target networks:
    \[
    \theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\
    \theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}
    \]
Sample Efficiency of DDPG [Wang et al., 2017]

DDPG converges in 1M steps, A3C requires 150M steps
Debugging Deep RL
Debugging Deep RL Algorithms

- Start with simple environments that are quick to train on
- Log everything (Frequently)!
  - In particular, keep track of:
    - Performance
    - Exploration hyperparameters
    - Loss function components
    - Gradients (Ensure they do not explode)
- Save your logs in a format that can be used for further processing
- Use tools that automatically displays your logs as Figures, e.g. Tensorboard
Debugging Deep RL Algorithms

- Policy Gradient
  - Policy should not get too close to deterministic policies early on
  - Track the magnitude of the policy gradient loss and entropy loss

- Q-Learning based methods
  - Track learning rate schedules
  - Track exploration schedule
  - Check magnitude of the gradients

- Visualize the policies during evaluation


Going Forward ...

• ~ 3 weeks left for the coursework
• Labs still this week (W7) and next week (W8)
  • Come with questions prepared!
• If you are unfamiliar with PyTorch, check out the provided notebook from the labs and documentation and tutorials on https://pytorch.org
Any Questions?