Deep reinforcement learning

Milica Gašić

Dialogue Systems Group, Cambridge University Engineering Department
In this lecture...

- Introduction to deep reinforcement learning
- Value-based Deep RL
- Deep Q-network
- Policy-based Deep RL
- Advantage actor-critic
- Model-based Deep RL
Deep reinforcement learning

Reinforcement learning where

- the value function,
- the policy, or
- the model

is approximated via a neural network is deep reinforcement learning. Neural network approximates a function as a non-linear function which is preferred in reinforcement learning. However, the approximation does not give any interpretation and the estimate is a local optimum which is not always desirable.
Deep representations

- A deep representation is a composition of many functions
- Its gradient can be backpropagated by the chain rule
Deep neural networks

Neural network transforms input vector $\mathbf{x}$ into an output $\mathbf{y}$:

$$
\begin{align*}
\mathbf{h}_0 &= g_0(W_0\mathbf{x}^T + b_0) \\
\mathbf{h}_i &= g_i(W_i\mathbf{h}_{i-1}^T + b_i), 0 < i < m \\
\mathbf{y} &= g_m(W_m\mathbf{h}_{m-1}^T + b_m)
\end{align*}
$$

where

$g_i$ (differentiable) activation functions hyperbolic tangent $\tanh$ or sigmoid $\sigma$, $0 \leq i \leq m$

$W_i, b_i$ parameters to be estimated, $0 \leq i \leq m$

It is trained to minimise the loss function $L = |\mathbf{y}^* - \mathbf{y}|^2$ with stochastic gradient descent in the regression case. In the classification case, it minimises the cross entropy $-\sum_i y_i^* \log y_i$. 
Weight sharing

- Recurrent neural network shares weights between time-steps

- Convolutional neural network shares weights between local regions
Q-networks

- Q-networks approximate the Q-function as a neural network.
- There are two architectures:
  1. Q-network takes an input $s, a$ and produces $Q(s, a)$
  2. Q-network takes an input $s$ and produces a vector $Q(s, a_1), \cdots, Q(s, a_k)$

![Diagram of Q-network architectures](image)
Deep Q-network

Q(s, a, \theta) is a neural network.

\[
MSVE = \left( r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2
\]

- Q-learning algorithm where Q-function estimate is a neural network
- This algorithm provides a biased estimate

This algorithm diverges because
- States are correlated
- Targets are non-stationary
DQN - Experience replay

- In order to deal with the correlated states, the agent builds a dataset of experience and then makes random samples from the dataset.
- In order to deal with non-stationary targets, the agent fixes the parameters $\theta^-$ and then with some frequency updates them

$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta)\right)^2$$
Atari

Diagram showing a cycle with state $s_t$, action $a_t$, and reward $r_t$. The Atari game is depicted with a joystick and a brain, illustrating the interaction between the environment and the agent.
DQN for Atari [Mnih et al., 2015]

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- State $s$ is stack of raw pixels from last 4 frames
- Action $a$ is one of 18 joystick/button positions
- Reward $r$ is change in score for that step
Results - Atari

[Image of a bar chart showing the results of various Atari games, with percentages indicating the level of difficulty for each game.]
Prioritised replay [Schaul et al., 2015]

- Related to prioritised sweeping in Dyna-Q framework
- Instead of randomly selecting experience order the experience by some measure of priority
- The priority is typically proportional to the TD-error

$$\delta = |r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta)|$$
Double DQN [van Hasselt et al., 2015]

- Remove upward bias caused by $\max_{a'} Q(s', a', \theta^-)$
- The idea is to produce two Q-networks
  1. Current Q-network $\theta$ is used to select actions
  2. Older Q-network $\theta^-$ is used to evaluate actions

$$MSVE = \left( r + \gamma Q(s', \text{arg max}_{a'} Q(s', a', \theta), \theta^-) - Q(s, a, \theta) \right)^2$$
Dueling Q-network [Wang et al., 2015]

- Dueling Q-network combined two streams to produce Q-function:
  1. one for state values
  2. another for advantage function

- The network learns state values for which actions have no effect

- Dueling architecture can more quickly identify correct action in the case of redundancy
Dueling Q-network

▶ Traditional DQN and dueling DQN architecture

▶ The value stream learns to pay attention to the road.
▶ The advantage stream learns to pay attention only when there are cars immediately in front.
Asynchronous deep reinforcement learning

- Exploits multithreading of standard CPU
- Execute many instances of agent in parallel
- Network parameters shared between threads
- Parallelism decorrelates data
- Viable alternative to experience replay
Policy approximation

- Policy $\pi$ is a neural network parametrised with $\omega \in \mathbb{R}^n$, $\pi(a, s, \omega)$
- Performance measure $J(\omega)$ is the value of the initial state $V_{\pi(\omega)}(s_0) = E_{\pi(\omega)}[r_0 + \gamma r_1 + \gamma^2 r_2, + \cdots]$
- The update of the parameters is
  $$\omega_{t+1} = \omega_t + \alpha \nabla J(\omega_t)$$
- And the gradient is given by the policy gradient theorem
  $$\nabla J(\omega) = E_{\pi} \left[ \gamma^t R_t \nabla \log \pi(a|s_t, \omega) \right]$$
- This gives REINFORCE algorithm for a neural network policy
Natural actor-critic with neural network approximations

- Approximate the advantage function as a neural network $\gamma^t A(s, a, \theta)$
- Approximate the policy as a neural network $\pi(a, s, \omega)$

**Critic evaluation** Choose $\theta$ and $J$ to minimise

$$(\sum_t \gamma^t A(s_t, a_t, \theta) + J - R)^2$$

**Actor update** $\omega \leftarrow \omega + \alpha \theta$ using compatible function approximation, where $\theta$ is natural gradient of $J(\omega)$
Advantage actor-critic [Mnih et al., 2016]

Approximate the policy as a neural network $\pi(a, s, \omega)$

- Define the objective
  \[ J(\omega) = V_{\pi(\omega)}(s_0) = E_{\pi(\omega)}[r_0 + \gamma r_1 + \gamma^2 r_2, + \cdots] \]
- Update $\omega$ with $\nabla J(\omega)$
  \[ \nabla J(\omega) = E_\pi [\gamma^t (R_t - V(s_t, \theta)) \nabla \omega \log \pi(a_t, s_t, \omega)] \]

Approximate the value function as a neural network $V(s, \theta)$

- Define the loss $L(\theta) = \gamma^t (R_t - V(s_t, \theta))^2$
- Update $\theta$ with $\nabla L(\theta)$

Compatible function approximation: $\nabla J(\omega)$ depends on the current estimate of $V(s, \theta)$
Advantage actor-critic

**Algorithm 1** Advantage actor-critic

1: Input: neural network parametrisation of $\pi(\omega)$
2: Input: neural network parametrisation of $V(\theta)$
3: repeat
4:     Initialise $\theta, \omega, V(\text{terminal}, \theta) = 0$
5:     Initialise $s_0$
6:     Obtain an episode $s_0, a_0, r_1, \cdots, r_T, s_T$ according to $\pi(\omega)$
7:     $R_T = 0$
8:     for $t = T$ downto 0 do
9:         $R_{t-1} = r_t + \gamma V(s_t, \theta)$
10:        $\nabla J = \nabla J + \gamma^t (R_t - V(s_t, \theta)) \nabla \omega \log \pi(a_t, s_t, \omega)$
11:        $\nabla L = \nabla L + \gamma^t \nabla \theta (R_t - V(s_t, \theta))^2$
12:     end for
13:     $\omega = \omega + \alpha \nabla J$
14:     $\theta = \theta + \beta \nabla L$
15: until convergence
Model-based Deep RL

- Dyna-Q framework can be used where transitions probabilities, rewards and the Q-function are all approximated by a neural network.
- Challenging to plan due to compounding errors
- Errors in the transition model compound over the trajectory
- Planning trajectories differ from executed trajectories
- At end of long, unusual trajectory, rewards are totally wrong
Summary

- Neural networks can be used to approximate the value function, the policy or the model in reinforcement learning.
- Any algorithms that assumes a parametric approximation can be applied with neural networks.
- However, vanilla versions might not always converge due to biased estimates and correlated samples.
- With methods such as prioritised replay, double Q-network or duelling networks the stability can be achieved.
- Neural networks can also be applied to actor-critic methods.
- Using them for model-based method does not always work well due to compounding errors.

