Deep Reinforcement Learning

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 $\mathbf{y}$:

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

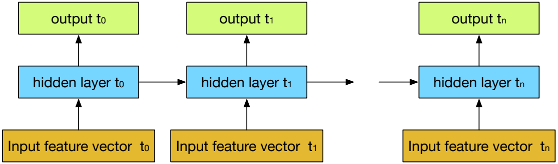
where

- g_i (differentiable) activation functions hyperbolic tangent tanh or sigmoid σ, $0 \leq i \leq m$
- W_i, b_i parameters to be estimated, $0 \leq i \leq m$

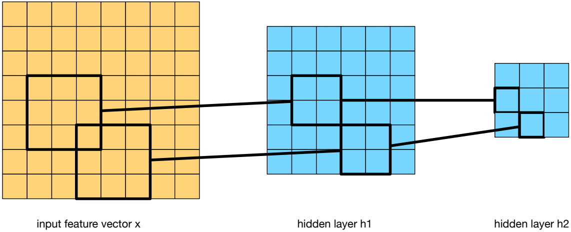
It is trained to minimise the loss function $L = |\mathbf{y}^* - \mathbf{y}|^2$ with stochastic gradient descent in the regression case. In the classification case, it minimises the cross entropy $-\sum_i y_i^* \log y_i$.

Weight sharing

- Recurrent neural network shares weights between time-steps

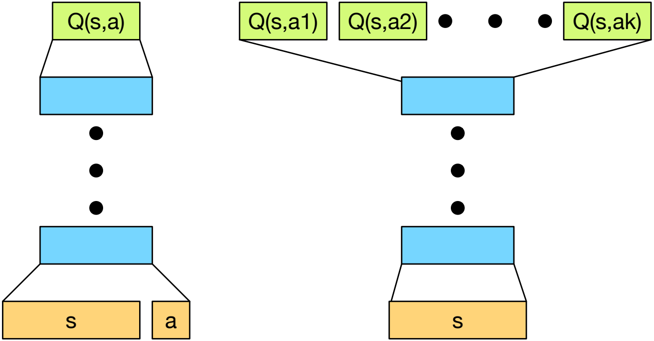


- Convolutional neural network shares weights between local regions



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 $Q(s, a_1), \cdots, Q(s, a_k)$



Deep Q-network

$Q(s, a, \boldsymbol{\theta})$ is a neural network.

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

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

This algorithm diverges because

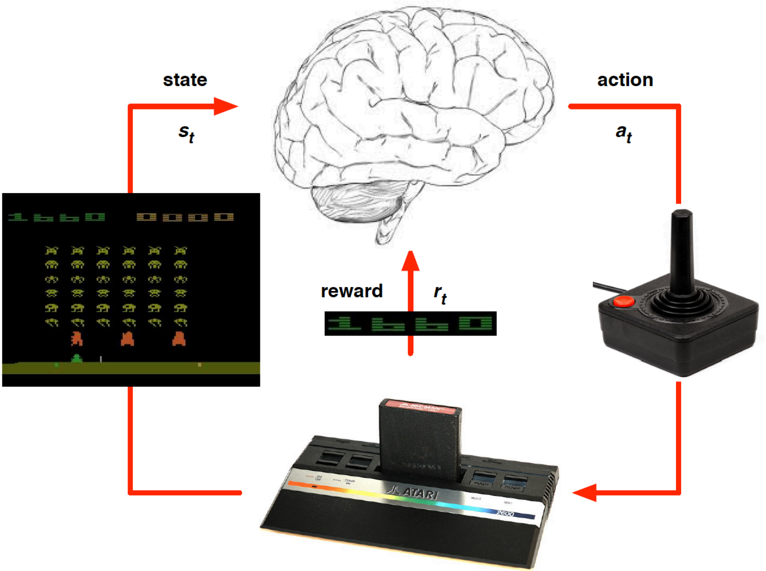
- States are correlated
- Targets are non-stationary

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 $\boldsymbol{\theta}^-$ and then with some frequency updates them

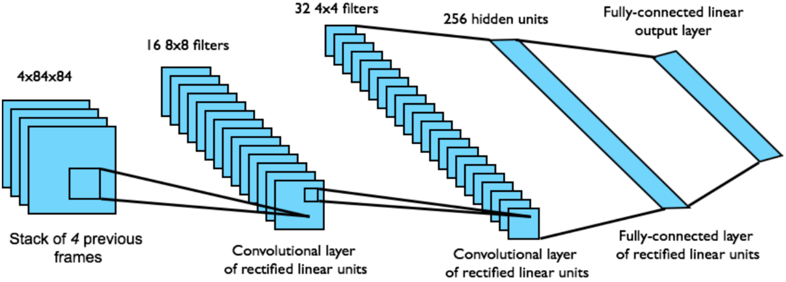
$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta}) \right)^2$$

Atari

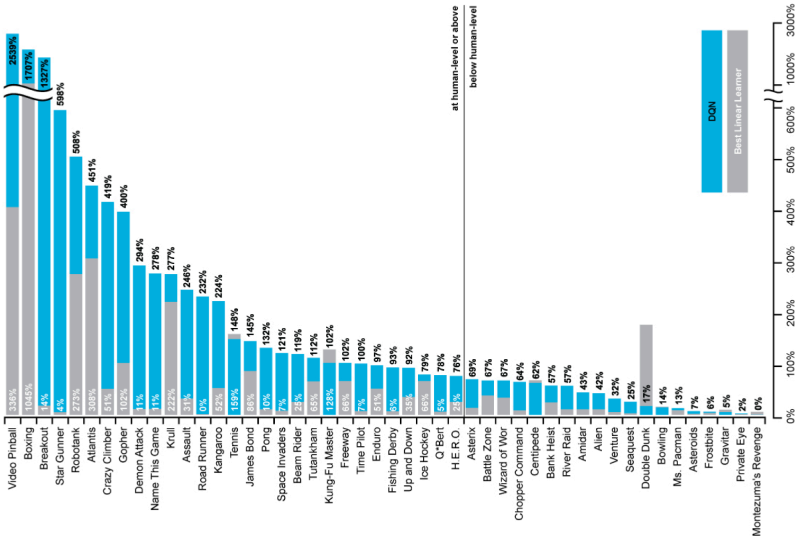


DQN for Atari [Mnih et al., 2015]

- 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



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

$$\delta = |r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta})|$$

Double DQN [van Hasselt et al., 2015]

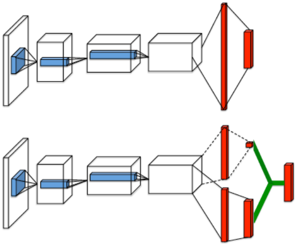
- Remove upward bias caused by $\max_{a'} Q(s', a', \boldsymbol{\theta}^-)$
- The idea is to produce two Q-networks
  1. Current Q-network $\boldsymbol{\theta}$ is used to select actions
  2. Older Q-network $\boldsymbol{\theta}^-$ is used to evaluate actions

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

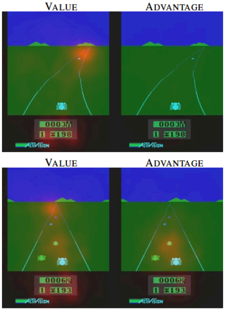
Dueling Q-network [Wang et al., 2015]

- 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 π is a neural network parametrised with $\boldsymbol{\omega} \in \mathbb{R}^n$, $\pi(a, s, \boldsymbol{\omega})$
- Performance measure $J(\boldsymbol{\omega})$ is the value of the initial state $V_{\pi(\boldsymbol{\omega})}(s_0) = E_{\pi(\boldsymbol{\omega})}[r_0 + \gamma r_1 + \gamma^2 r_2, + \cdots]$
- The update of the parameters is

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \nabla J(\boldsymbol{\omega}_{\boldsymbol{t}})$$

- And the gradient is given by the policy gradient theorem

$$\nabla J(\boldsymbol{\omega}) = E_\pi \left[\gamma^t R_t \nabla_{\boldsymbol{\omega}} \log \pi(a|s_t, \boldsymbol{\omega}) \right]$$

- This gives REINFORCE algorithm for a neural network policy

Natural actor-critic with neural network approximations

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

Critic evaluation
: Choose $\boldsymbol{\theta}$ and J to minimise $\left(\sum_t \gamma^t A(s_t, a_t, \boldsymbol{\theta}) + J - R\right)^2$

Actor update
: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\boldsymbol{\theta}$ using compatible function approximation, where $\boldsymbol{\theta}$ is natural gradient of $J(\boldsymbol{\omega})$

Advantage actor-critic [Mnih et al., 2016]

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

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

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

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

Compatible function approximation: $\nabla J(\boldsymbol{\omega})$ depends on the current estimate of $V(s, \boldsymbol{\theta})$

Advantage actor-critic

Algorithm 1 Advantage actor-critic

Input: neural network parametrisation of $\pi(\boldsymbol{\omega})$
Input: neural network parametrisation of $V(\boldsymbol{\theta})$
repeat
    Initialise $\boldsymbol{\theta}, \boldsymbol{\omega}, V(\textit{terminal}, \boldsymbol{\theta}) = 0$
    Initialise s_0
    Obtain an episode $s_0, a_0, r_1, \cdots, r_T, s_T$ according to $\pi(\boldsymbol{\omega})$
    $R_T = 0$
    for $t = T$ downto 0 **do**
        $R_{t-1} = r_t + \gamma V(s_t, \boldsymbol{\theta})$
        $\nabla J = \nabla J + \gamma^t (R_t - V(s_t, \boldsymbol{\theta})) \nabla_{\boldsymbol{\omega}} \log \pi(a_t, s_t, \boldsymbol{\omega})$
        $\nabla L = \nabla L + \gamma^t \nabla_{\boldsymbol{\theta}} (R_t - V(s_t, \boldsymbol{\theta}))^2$
    end for
    $\boldsymbol{\omega} = \boldsymbol{\omega} + \alpha \nabla J$
    $\boldsymbol{\theta} = \boldsymbol{\theta} + \beta \nabla L$
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