MLMI7: Reinforcement Learning and Decision Making

Deep reinforcement learning

Presented by
José Miguel Hernández–Lobato, Carl Edward Rasmussen
& Robert Pinsler

Slides prepared by Milica Gašić
Edited by Adrià Garriga-Alonso

Lent Term

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.
Advantages:

- Approximate the function as a non-linear function
- Pre-define architecture instead of features

Problems:

- No interpretation for the approximation
- the estimate is a local optimum
- Learning deep models requires a lot of data.

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 x into an output y:

$$
\begin{aligned}
\mathrm{x} &= \mathrm{h}_0 & \mathrm{y} &= \mathrm{h}_m \\
\mathrm{h}_i &= g_i(W_i \mathrm{h}_{i-1}^{\mathsf{T}} + b_i), & 0 < i &\le m
\end{aligned}
$$

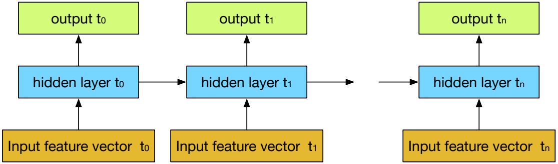
where

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

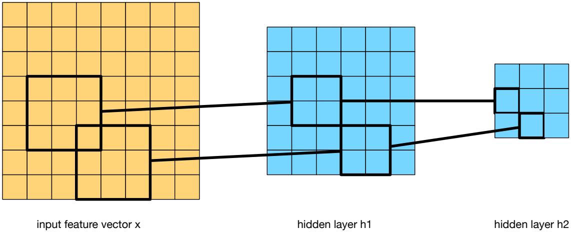
It is trained to minimise the loss function $L = |\mathrm{y}^* - \mathrm{y}|^2$ with stochastic gradient descent in the regression case. In the classification case, it minimises the cross entropy $-\sum_i y_i^* \log y_i$.

Weight sharing

- Recurrent neural network shares weights between time-steps

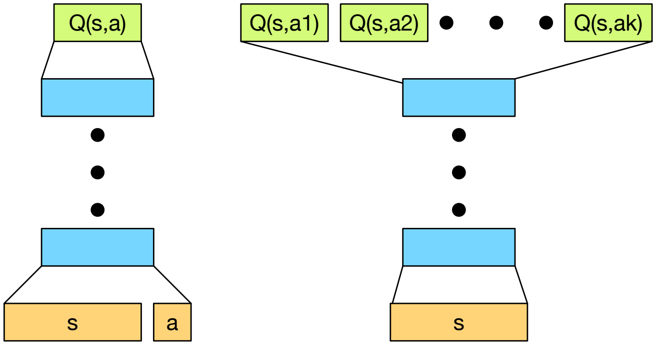


- Convolutional neural network shares weights between local regions



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. Mean squared value error:

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

- Q-learning algorithm where Q-function estimate is a neural network
- Provides a biased gradient estimate (semi-gradient)

This algorithm diverges because

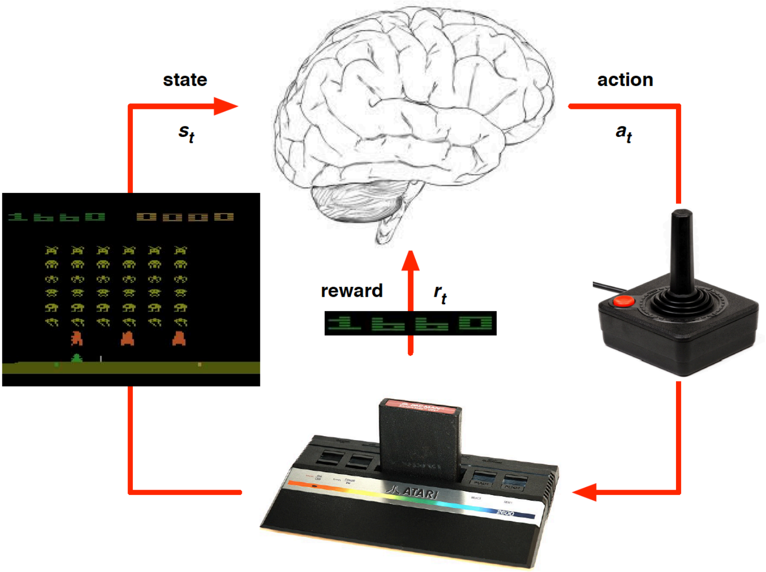
- States are correlated
- Target constantly changes ($\boldsymbol{\theta}$ in target)

DQN - Experience replay

- **Experience replay:** deal with the correlated states.
  - Store a dataset of experience and take random samples from the dataset.
- **Target network:** deal with non-stationary targets.
  - Fix the parameters $\boldsymbol{\theta}^-$ and with some frequency update them.

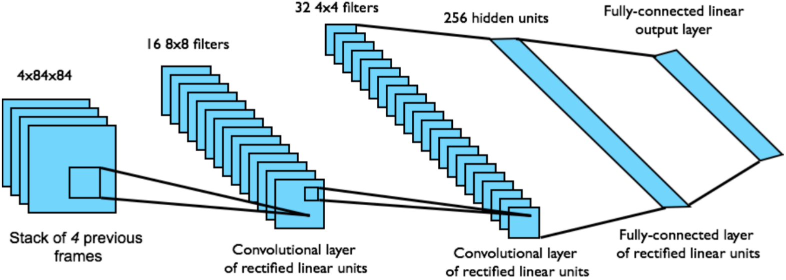
$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta}) \right)^2$$

Atari

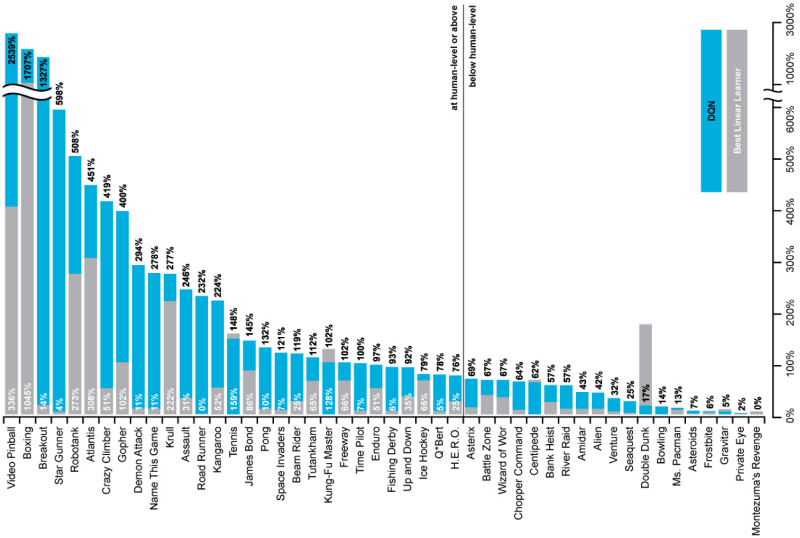


DQN for Atari [Mnih et al., 2015]

- 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 uniformly selecting experience weigh the experience by some measure of priority
- The selection probability is typically proportional to the TD-error (analogous to surprise)

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

- Need importance sampling to remove bias caused by priorities

Double DQN [van Hasselt et al., 2015]

- DQN overestimates Q-function
- Remove upward bias caused by using $\max_{a'} Q(s', a', \boldsymbol{\theta}^-)$ as a target.
  - Solution for tabular Q-learning: use two estimates of the Q-function.
- The idea is to produce two Q-networks
  1. Current Q-network $\boldsymbol{\theta}$ is used to select actions
  2. Older (target) 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

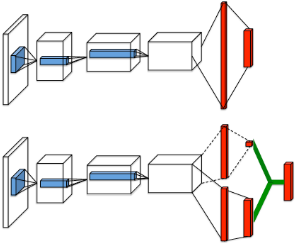
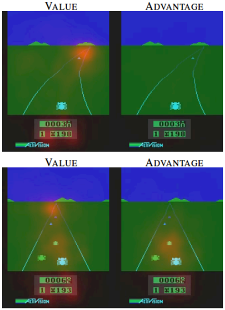


Figure 1: Top: traditional DQN. Bottom: dueling DQN.

- Recall $Q(s, a) = V(s) + A(s, a)$
- Need additional constraints on V, A to make it work.



- 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
  - Can use on-policy learning, all data is from the current policy.

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

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

- 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

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

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

- Same objective
- Update $\boldsymbol{\omega}$ with $\nabla J(\boldsymbol{\omega})$
  $\nabla J(\boldsymbol{\omega}) = E_\pi \left[\gamma^t (R_t - V(s_t, \boldsymbol{\theta})) \nabla_{\boldsymbol{\omega}} \log \pi(a_t, s_t, \boldsymbol{\omega})\right]$
- R_t is an estimate of the Q-function, the *n*-step return.

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 $\nabla L(\boldsymbol{\theta})$

It's a semi-gradient update again.

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 $\boldsymbol{\theta}, \boldsymbol{\omega}, V(terminal, \boldsymbol{\theta}) = 0$
    Initialise s_0
    Obtain an episode $s_0, a_0, r_1, \cdots, r_T, s_T$ according to $\pi(\boldsymbol{\omega})$
    $R \leftarrow 0$
    for $t = T$ downto 0 **do**
        $R \leftarrow r_t + \gamma R$
        $\nabla J \leftarrow \nabla J + (R - V(s_t, \boldsymbol{\theta}))\nabla_{\boldsymbol{\omega}} \log \pi(a_t, s_t, \boldsymbol{\omega})$
        $\nabla L \leftarrow \nabla L + \nabla_{\boldsymbol{\theta}}(R - V(s_t, \boldsymbol{\theta}))^2$
    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 transition 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
- In the last couple of years, this has started to work (using uncertainty in deep learning and other approaches)

Summary

- Neural networks can be used to approximate the value function, the policy or the model in reinforcement learning.
- Any algorithm 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

References I

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016).
Asynchronous methods for deep reinforcement learning.
CoRR, abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015).
Prioritized experience replay.
CoRR, abs/1511.05952.

References II

van Hasselt, H., Guez, A., and Silver, D. (2015).
Deep reinforcement learning with double q-learning.
CoRR, abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. (2015).
Dueling network architectures for deep reinforcement learning.
CoRR, abs/1511.06581.