Lecture 7: Deep reinforcement learning

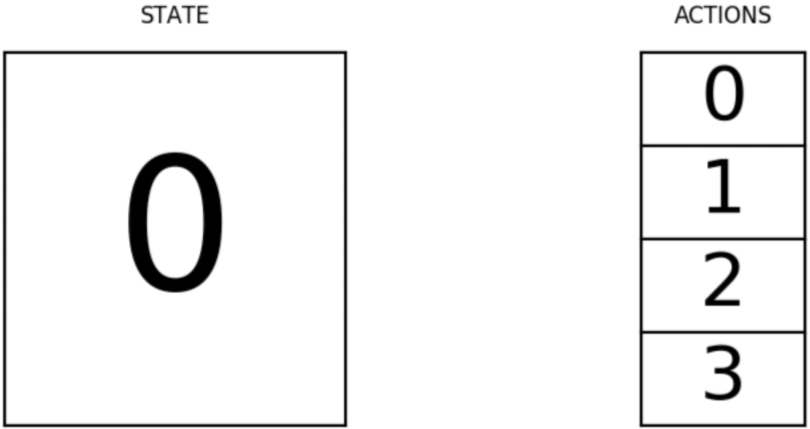
COSC470

Lech Szymanski

Department of Computer Science, University of Otago

September 11, 2018

“Shall we play a game?”



Framework for RL [1]

Sequential decision problems are a framework for problems that consist of the following:

- an environment discretised into states (field of play);

Framework for RL [1]

Sequential decision problems are a framework for problems that consist of the following:

- an environment discretised into states (field of play);
- an agent with some perception of the current state (player);

Framework for RL [1]

Sequential decision problems are a framework for problems that consist of the following:

- an environment discretised into states (field of play);
- an agent with some perception of the current state (player);
- a set of possible actions to choose from (player's actions);

Framework for RL [1]

Sequential decision problems are a framework for problems that consist of the following:

- an environment discretised into states (field of play);
- an agent with some perception of the current state (player);
- a set of possible actions to choose from (player's actions);
- a Markovian transition model where $p(s'|s,a)$ is the probability of ending up in state s' when action a was taken from state s (rules of the game - often unknown to the player)

Framework for RL [1]

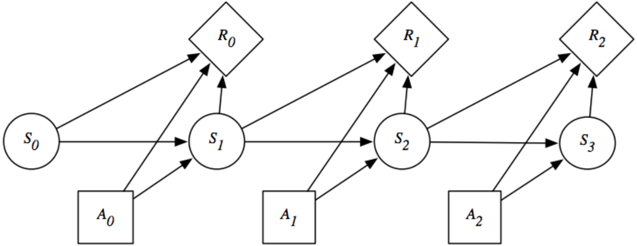
Sequential decision problems are a framework for problems that consist of the following:

- an environment discretised into states (field of play);
- an agent with some perception of the current state (player);
- a set of possible actions to choose from (player's actions);
- a Markovian transition model where $p(s'|s,a)$ is the probability of ending up in state s' when action a was taken from state s (rules of the game - often unknown to the player)
- a reward associated with each state (the score);

Uncertain world

A fully observable stochastic environment with a Markovian transition model is called a **Markov Decision Process** (MDP)

$$p(s'|s,a) = \Pr\{S_{t+1} = s'|S_t = s, A_t = a\}$$



`https://artint.info/html/ArtInt_224.html`

Agent: what to do?

Policy is a function of current state s_t that recommends action a to take. It is formalised as a probability distribution $\pi(a|s)$.
For example:

$$\pi(A_t = \mathsf{N}|S_t = ☎) = 0.3$$
$$\pi(A_t = \mathsf{E}|S_t = ☎) = 0.2$$
$$\pi(A_t = \mathsf{S}|S_t = ☎) = 0.4$$
$$\pi(A_t = \mathsf{W}|S_t = ☎) = 0.1$$

The objective of the agent is to develop a policy to maximise its rewards.

Agent: how well am I doing...?

The agent needs to maximise the total sum of future **rewards**.

$$
\begin{aligned}
G_t &= r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \ldots \\
&= r(s_t) + \gamma G_{t+1}
\end{aligned}
$$

where:

- s_t, $r(s_t)$ denote the current state and reward,
- s_{t+1}, $r(s_{t+1})$ denote the next state and the next reward,
- $0 < \gamma \leq 1$ can be chosen to discount future rewards.

Agent: how well am I doing...in an uncertain world?

There is an additional complication in that the world is assumed to be stochastic - that picking action a while in state s_t may not lead always to the same outcome (recall $p(s'|s_t, a)$).

The value (or utility) of a given state is the expected reward from the current state:

$$
\begin{aligned}
v(s) &= E\{G_t | S_t = s\} \\
&= \sum_a \pi(a|s) \sum_{s'} p(s'|a, s)\big[r(s') + \gamma v(s')\big] \\
&= \sum_a \pi(a|s) q(a, s),
\end{aligned}
$$

where $q(a, s) = \sum_{s'} p(s'|a, s) v(s')$ is the expected value of action a when in state s.

Value iteration algorithm

Given $p(s'|a, s)$ and the recursive nature of the formula for $v(s)$ (Bellman equation), the policy can be found using Dynamic Programming (DP) methods.

Algorithm 1 Pseudocode for the value iteration algorithm

Initialise $v(s)$ to 0 for all $s \in S$
repeat
    for each non-terminal $s \in S$ **do**
        $v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a)\big[r(s') + \gamma v(s')\big]$
    end for
until converged

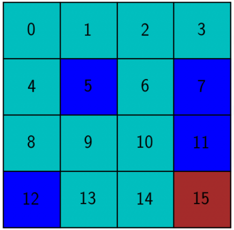
The state value is the same as the best action value $v(s) = \max_a q(s, a)$ and $q(a, s) = \sum_{s'} p(s'|a, s)v(s')$.

Once the algorithm converges, the policy is given by $\arg\max_a q(s, a)$.

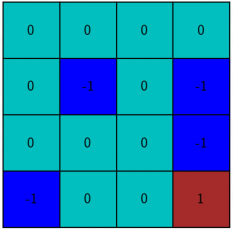
Example: Frozen lake

Environment:

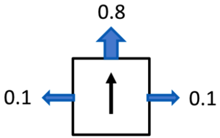
- $S = \{0, \ldots, 15\}$
- Start state $s = 0$
- $r(s) = 0$ for non-terminal states; $r(s) = \pm 1$ for terminal states
- $\gamma = 0.8$
- 4 actions - go N, E, S or W
- 80% chance of action's success, 20% chance of slipping sideways
- Remain in the same state when walking into the wall



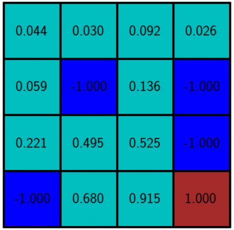
States $s \in \{0, \ldots, 15\}$



Rewards $r \in \{-1, 0, 1\}$



$p(s'|a, s)$

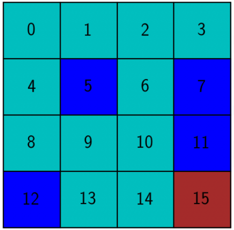


State values
$v(s) = \max_a q(s, a)$

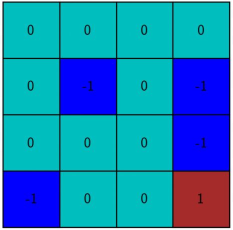
Example: Frozen lake

Q-table:

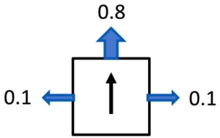
s	S	E	N	W
0	0.044	0.027	0.034	0.036
1	-0.790	-0.030	0.030	-0.070
2	0.092	0.035	0.063	0.037
3	-0.791	-0.081	0.026	-0.039
4	0.046	-0.779	-0.067	0.059
5	0	0	0	0
6	0.136	-0.751	-0.141	-0.751
7	0	0	0	0
8	-0.743	0.221	0.095	0.046
9	0.495	0.291	-0.740	0.096
10	0.525	-0.716	0.027	0.401
11	0	0	0	0
12	0	0	0	0
13	0.408	0.680	0.290	-0.706
14	0.740	0.915	0.491	0.550
15	0	0	0	0



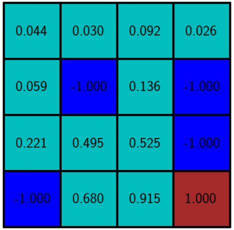
States $s \in \{0, \ldots, 15\}$



Rewards $r(s) \in \{-1, 0, 1\}$



$p(s'|a, s)$



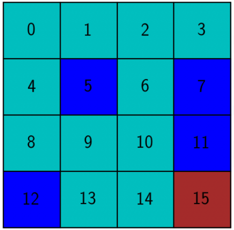
State values

$v(s) = \max_a q(s, a)$

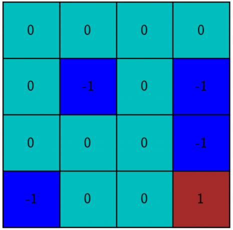
Example: Frozen lake

Policy from the Q-table:

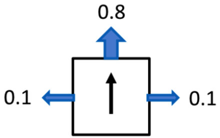
s	$\pi(\mathsf{S}\|s)$	$\pi(\mathsf{E}\|s)$	$\pi(\mathsf{N}\|s)$	$\pi(\mathsf{W}\|s)$
0	0.252	0.248	0.250	0.250
1	0.134	0.285	0.305	0.276
2	0.259	0.245	0.252	0.245
3	0.139	0.274	0.305	0.286
4	0.230	0.131	0.267	0.303
5	0.25	0.25	0.25	0.25
6	0.387	0.160	0.293	0.169
7	0.25	0.25	0.25	0.25
8	0.123	0.322	0.284	0.270
9	0.360	0.294	0.105	0.242
10	0.360	0.104	0.219	0.318
11	0.25	0.25	0.25	0.25
12	0.25	0.25	0.25	0.25
13	0.283	0.372	0.252	0.093
14	0.263	0.314	0.205	0.218
15	0.25	0.25	0.25	0.25



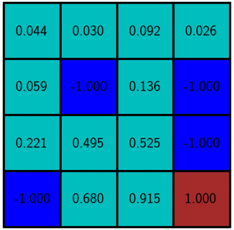
States $s \in \{0, \ldots, 15\}$



Rewards $r \in \{-1, 0, 1\}$



$p(s'|a, s)$



State values
$v(s) = \max_a q(s, a)$

Reinforcement learning (RL)

Reinforcement learning (RL) solves a sequential decision problem without being given the rules of the game, $p(s'|a, s)$:

- It figures out what to do by acting in the world and discovering the rewards through experience
- **Exploitation** - choosing the best options according to policy $\pi(a|s)$
- **Exploration** - choosing non-optimal actions just to see what happens

Temporal difference learning

RL agent samples sequences of states actions and rewards by interacting with the environment, which lends itself to Monte Carlo (MC) methods. Temporal difference learning (TD) is a combination of DP and MC. Given action a_t was chosen in state s_t and the result was state s_{t+1} with reward $r(s_{t+1})$:

$$v(s_t) \leftarrow v(s_t) + \alpha\big[v^* - v(s_t)\big],$$

where $v^* = r(s_{t+1}) + \gamma v(s_{t+1})$ is the target state value.

For Q-learning the TD update is:

$$q(s, a_t) \leftarrow q(s, a_t) + \alpha\big[q^* - q(s, a_t)\big],$$

where $q^* = r(s_{t+1}) + \gamma \max_a q(s_{t+1}, a)$ is the target action value.

Q-learning

Algorithm 2 Pseudocode for the q-learning algorithm

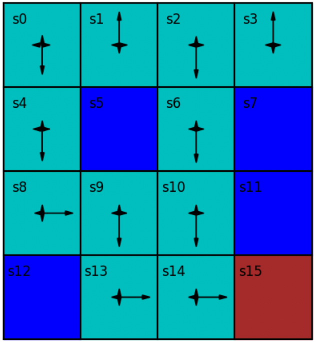
```
Initialise $q(s, a)$ to 0 for all $a \in A$ in each $s \in S$
repeat
    Reset for new episode $s_0$,$t \leftarrow 0$
    while $s_t$ is non-terminal do
        Choose $a_t$ according to $\pi(a_t|s_t) \leftarrow \frac{e^{q(a_t,s_t)}}{\sum_a e^{q(a,s_t)}}$
        Perform action $a_t$, get $s_{t+1}$ and $r(s_{t+1})$
        $q^* \leftarrow r(s_{t+1}) + \gamma \max_a q(s_{t+1}, a)$
        $q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha\left[q^* - q(s_t, a_t)\right]$
        $t \leftarrow t + 1$
    end while
until converged
```

Example: Frozen lake

Environment:

- $S = \{0, \ldots, 15\}$
- Start state $s = 0$
- $r(s) = -0.04$ for non-terminal states; $r(s) = \pm 1$ for terminal states
- $\gamma = 0.8$
- $a \in \{\mathsf{S}, \mathsf{E}, \mathsf{N}, \mathsf{W}\}$

Episode 50000



Q table

	S	E	N	W
s0	-0.08	-0.13	-0.11	-0.10
s1	-0.83	-0.17	-0.13	-0.20
s2	-0.01	-0.12	-0.06	-0.11
s3	-0.82	-0.21	-0.13	-0.18
s4	-0.04	-0.83	-0.19	-0.06
s5	0.00	0.00	0.00	0.00
s6	0.07	-0.80	-0.28	-0.78
s7	0.00	0.00	0.00	0.00
s8	-0.76	0.14	-0.02	-0.00
s9	0.39	0.21	-0.75	-0.02
s10	0.45	-0.71	-0.06	0.28
s11	0.00	0.00	0.00	0.00
s12	0.00	0.00	0.00	0.00
s13	0.34	0.58	0.17	-0.70
s14	0.66	0.87	0.40	0.45
s15	0.00	0.00	0.00	0.00

Policy (left) according to Q-table (right)

Policy gradient [2]

For problems where it is impossible to maintain a Q-table of all possible states, the policy can be modelled with a a neural network, $\pi(a|s, \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the set of parameters (network weight and biases). After a sequence of states and rewards has been observed, the parameters $\boldsymbol{\theta}$ are updated so as to minimise the following cost function:

$$J_t(\boldsymbol{\theta}) = -G_t \ln \pi(a_t|s_t, \boldsymbol{\theta}),$$

Recall that $G_t = r(s_{t+1}) + \gamma G_{t+1}$ is the reward for each action in the episode. This reward is often offset by a baseline value $\hat{v}(s, \boldsymbol{\omega})$, which is the predicted reward coming from a different model parametrised by $\boldsymbol{\omega}$:

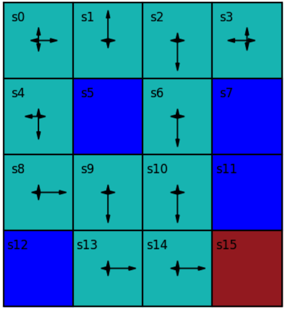
$$J_t(\boldsymbol{\theta}) = \big(G_t - \hat{v}(s, \boldsymbol{\omega})\big) \ln \pi(a_t|s_t, \boldsymbol{\theta})$$
$$J_t(\boldsymbol{\omega}) = \Big(G_t - \hat{v}(s, \boldsymbol{\omega})\Big)^2$$

Example: Frozen lake

Environment:

- $S = \{0, \ldots, 15\}$
- Start state $s = 0$
- $r(s) = -0.04$ for non-terminal states; $r(s) = \pm 1$ for terminal states
- $\gamma = 0.8$
- $a \in \{\mathsf{S}, \mathsf{E}, \mathsf{N}, \mathsf{W}\}$

Episode 5000



NN generated policy

	S	E	N	W
s0	0.14	0.57	0.24	0.05
s1	0.00	0.01	0.99	0.00
s2	0.99	0.00	0.00	0.01
s3	0.11	0.06	0.24	0.60
s4	0.67	0.00	0.01	0.32
s5				
s6	1.00	0.00	0.00	0.00
s7				
s8	0.00	1.00	0.00	0.00
s9	1.00	0.00	0.00	0.00
s10	1.00	0.00	0.00	0.00
s11				
s12				
s13	0.00	1.00	0.00	0.00
s14	0.00	1.00	0.00	0.00
s15				

Policy (left) according to NN output (right)

Actor-Critic

When $\hat{v}(s, \boldsymbol{\omega})$ is updated using its prediction of the next states value (as opposed to G_t) the twin model method is referred to as *actor-critic* method. The policy model $\pi(a|s, \boldsymbol{\theta})$ is the *actor* and the value model $\hat{v}(s, \boldsymbol{\omega})$ is the *critic*. The *actor* is trained by minimising the following cost with respect to $\boldsymbol{\theta}$:

$$J_t(\boldsymbol{\theta}) = \big(R_{t+1} + \gamma\hat{v}(s_{t+1}, \boldsymbol{\omega}) - \hat{v}(s, \boldsymbol{\omega})\big) \ln \pi(a_t|s_t, \boldsymbol{\theta}),$$

whereas the *critic* is trained by minimising the following cost with respect to $\boldsymbol{\omega}$:

$$J_t(\boldsymbol{\omega}) = \Big(R_{t+1} + \gamma\hat{v}(s_{t+1}, \boldsymbol{\omega}) - \hat{v}(s, \boldsymbol{\omega})\Big)^2$$

DQN [3]

A deep Q-network (DQN) agent is a neural network $\hat{q}(a, s, \boldsymbol{\theta})$ which models the Q-table. The cost function optimised is:

$$J_t(\boldsymbol{\theta}) = \big(q^* - \hat{q}(s_t, a, \boldsymbol{\theta})\big)^2,$$

where $q^* = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \boldsymbol{\theta})$.
The agent uses a number of *tricks* in order to stabilise the training:

- Experience replay
- Treating $\max_a \hat{q}(S_{t+1}, a, \boldsymbol{\theta})$ as a constant
- The error term $q^* - \hat{q}(s_t, a, \boldsymbol{\theta})$ was clipped to interval $[-1, 1]$.

References

[1] R. S. Sutton, and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 2ed draft `http://incompleteideas.net/book/bookdraft2017nov5.pdf`, 2017.

[2] R. J. Williams. *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning* Machine Learning, 8(3-4): 229–256, 1992.

[3] V. Mnih, K Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, *Playing Atari with Deep Reinforcement Learning*. CoRR, `http://arxiv.org/abs/1312.5602`, 2013.