UNIVERSITY OF LEEDS

Class: Machine Learning

Reinforcement Learning Algorithms

Instructor: Matteo Leonetti

Learning outcomes

UNIVERSITY OF LEEDS

- Update estimates of the value function with:
  - First-visit Monte Carlo
  - Sarsa
  - Q-Learning
- Explain the difference between off and on-policy learning
- Explain the difference between Monte Carlo methods and TD learning.

Dynamic Programming

UNIVERSITY OF LEEDS

We turn the Bellmann optimality equation into an update rule:

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_k(s',a'))$$

We need a model of the dynamics and the reward of the MDP. This is *planning*.

We wanted to do *learning*!

We discussed in the last lecture how we can use the Bellman optimality equation as an update rule, to compute the optimal value function starting with any value function.

However, this requires the model of the environment, that is, the joint distribution of next state and reward conditioned on the current state and action. For this reason, this is a *planning* algorithm, and is called *dynamic programming*.

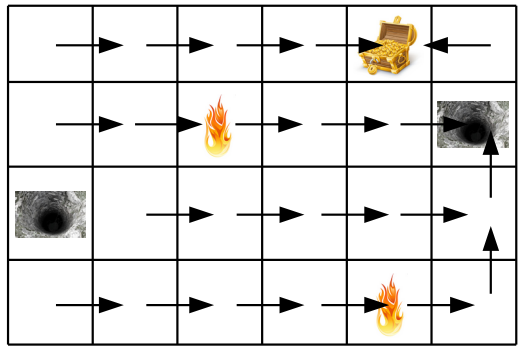
In this lecture we see how to *learn* a behaviour when such a model is not available.

Policy iteration

UNIVERSITY OF LEEDS

Transition and reward function unknown: $MDP\ M = \langle S, A, ?, ? \rangle$

The agent starts with an arbitrary policy (or, equivalently, action-value function), For instance, this policy:



The need for learning arises from the fact that there is something we don't know. In particular we don't know the transition function and the reward function, so we can't apply dynamic programming directly.

The previous method for dynamic programming is also called value iteration, because it iterates over successive value functions, until it converges to the optimal one.

We are now going to look at what is called *policy iteration* which is more suitable for learning.

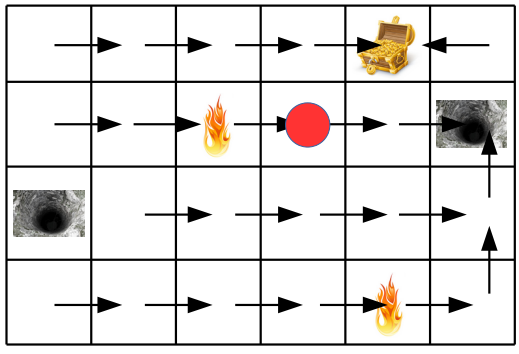
Policy iteration starts by having an initial policy. Any policy will do. The one in the picture is just a possible policy.

Policy iteration

UNIVERSITY OF LEEDS

Step 1: policy evaluation

Compute the action-value function of this policy



For simplicity, let's focus on one state:

$\gamma = 0.5$

$q(\langle 4,2 \rangle, \text{right}) = 0 + \gamma(-100) = -50$

$q(\langle 4,2 \rangle, \text{up}) = 0 + \gamma(50) = 25$

$q(\langle 4,2 \rangle, \text{left}) = -5 + \gamma^3(-100) = -17.5$

$q(\langle 4,2 \rangle, \text{down}) = 0 + \gamma^3(-100) = -12.5$

The first step is a policy evaluation step. We'll see later how to *learn* the value of the current policy, but for now let's just evaluate the current policy by looking at the transition function and the reward.

This must be done for every state, but for brevity we focus on a single state.

The current policy goes right in this state. Going right causes the agent to fall into the pit in the second step, so it's value is 0 for the first step, and -100 for the second step. The value of the second step is discounted by γ.

The expected value of going up and then following the current policy is 0 for the first step, and then 50 for the second. So the return, that is the cumulative discounted reward, is 25.

The expected value of going left is an immediate -5 for the fire, and then falling into the pit in three steps, so γ is elevated to the power of 3. This value is a little higher than going right directly into the pit because the agent lives a little longer, but only marginally better.

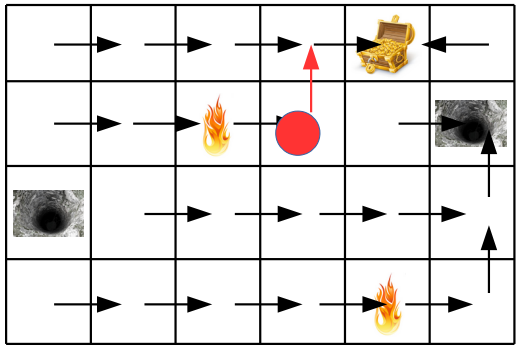
Lastly, the expected value of going down is 0 immediately, and then -100 in 3 steps. This is a little better than going straight into the fire, but again results in the agent jumping into the pit, so still not great.

Policy iteration

UNIVERSITY OF LEEDS

Step 2: policy improvement

Having the value for the current policy, find a state in which a different action would be better than the current one.



$$q(\langle 4,2\rangle, \text{right}) = 0 + \gamma(-100) = -50$$

$$q(\langle 4,2\rangle, \text{up}) = 0 + \gamma(50) = 25$$

$$q(\langle 4,2\rangle, \text{left}) = -5 + \gamma^3(-100) = -17.5$$

$$q(\langle 4,2\rangle, \text{down}) = 0 + \gamma^3(-100) = -12.5$$

The current action is RIGHT, a better action is UP

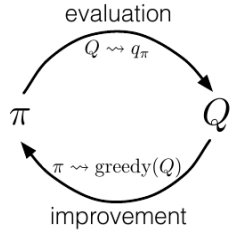
The next step is to look at the evaluation of every action in every state (in our example we only look at one state), and change the policy to any action that achieves a higher expected return.

Now that the policy has changed, we can evaluate the new policy, and keep this loop until the policy does not change any more.

This loop is called: policy iteration.

Generalized Policy iteration

UNIVERSITY OF LEEDS



No need to fully evaluate a policy before making an improvement → *Generalized* PI

On MDPs this is a *hill-climbing* procedure, every solution is strictly better than previous ones.

How do we evaluate a policy without a model?

In practice, we don't have to evaluate the policy in every state before we can make an improvement, but as soon as we find one state in which a change of the action would improve the policy we can make the change, and start evaluating the new policy.

This loop is called *generalized policy iteration*, and is what we are going to use.

So far we computed the action-value function by hand, looking at the grid. This means we have access to the transition function (we know that taking action RIGHT moves the agent one step to the right), and the reward function (we know that the fire is -5, the pit -100, and the gold +50).

Estimating the value function

UNIVERSITY OF LEEDS

We are going to estimate the action-value function through successive approximations obtained by sampling from the environment.

The general learning rule will be the following:

$$q_{k+1}(s,a) = (1-\alpha)\, q_k(s,a) + \alpha\, \text{target}$$

$$= q_k(s,a) + \alpha(\text{target} - q_k(s,a))$$

Similarly to dynamic programming, we want to estimate the action value function through a sequence of converging approximations.

However, in this case we don't use the transition function or the reward function (because we don't know them...) but we sample from them, by actually taking actions in the environment.

We compute the value function as a linear interpolation between the previous estimate and the new sampled values.

Different targets for the interpolation define different algorithms.

Bellman Equation

UNIVERSITY OF LEEDS

$$q_\pi(s,a)=\sum_{s',r} p(s',r|s,a)\left(r+\gamma \sum_{a'} \pi(a'|s)\, q_\pi(s',a')\right)$$

Target policy: the policy I want to learn about

$$q^*(s,a)=\sum_{s',r} p(s',r|s,a)\left(r+\gamma \max_{a'} q^*(s',a')\right)$$

Here the target is the optimal policy

The Bellman Equation defines the value of a state-action pair under some policy π.

In the first of the two equations, the policy is used to determine the value of the next action. This equation shows how the value of the next state-action pair depends on both the environment and the agent. The model of the environment is represented here by a joint distribution p of next state and reward given the current state and action. The policy of the agent is a distribution over actions given a state, and is used to determine the probability of each action in the next state.

If we substitute the current policy with the *optimal* policy, that is the policy that always chooses the best possible next action, we have the Bellman Optimality Equation.

The first equation is the basis for on-policy learning, and is what the algorithm Sarsa converges to.

The second equation is the basis for off-policy learning, and is what Q-learning converges to.

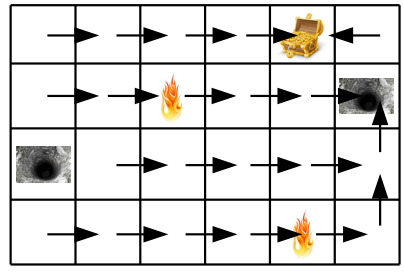
States, actions, rewards

UNIVERSITY OF LEEDS

All we have is a sequence of state, actions and rewards:

$$\langle s_{0,} a_{0,} r_{1,} s_1 \rangle, \langle s_{1,} a_{1,} r_{2,} s_2 \rangle, \langle s_{2,} a_{2,} r_{3,} s_3 \rangle, \ldots, \langle s_{n-1}, a_{n-1}, r_n, s_n \rangle$$

Example sequence:



$$\langle\langle 0,0\rangle, \mathrm{r}, 0, \langle 1,0\rangle\rangle \ \langle\langle 1,0\rangle, \mathrm{r}, 0, \langle 2,0\rangle\rangle \ \langle\langle 2,0\rangle, \mathrm{r}, 0, \langle 3,0\rangle\rangle \ \langle\langle 3,0\rangle, \mathrm{r}, -5, \langle 4,0\rangle\rangle$$

$$\langle\langle 4,0\rangle, \mathrm{r}, 0, \langle 5,0\rangle\rangle \ \langle\langle 5,0\rangle, \mathrm{u}, 0, \langle 5,1\rangle\rangle \ \langle\langle 5,1\rangle, \mathrm{u}, -100, \langle 5,2\rangle\rangle$$

In learning, we don't have the model of the environment, but we can get data by acting in it.

This is an example of a possible trajectory, where each tuple contains the initial state, the action, the reward, and the next state.

This is the data an RL algorithm learns from. Differently from typical supervised learning methods, in this case the agent also controls how to generate the data, by taking actions in the environment.

Therefore we have two distinct problems: prediction and control.

The prediction problem consists in estimating the long-term consequences of actions, through the value function.

The control problem consists in choosing which action to take, given the predictions made.

We predict the weather all the time, but whether or not to take an umbrella is still your choice. So the weather forecast gives you a prediction, but then the agent has to choose an action, and that's the control part.

First-visit Monte Carlo

UNIVERSITY OF LEEDS

A Monte-Carlo update uses a sample of the return as the target:

The return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$\langle\langle 0,0\rangle, \mathrm{r}, 0, \langle 1,0\rangle\rangle \; \langle\langle 1,0\rangle, \mathrm{r}, 0, \langle 2,0\rangle\rangle \; \langle\langle 2,0\rangle, \mathrm{r}, 0, \langle 3,0\rangle\rangle \; \langle\langle 3,0\rangle, \mathrm{r}, -5, \langle 4,0\rangle\rangle$

$\langle\langle 4,0\rangle, \mathrm{r}, 0, \langle 5,0\rangle\rangle \; \langle\langle 5,0\rangle, \mathrm{u}, 0, \langle 5,1\rangle\rangle \; \langle\langle 5,1\rangle, \mathrm{u}, -100, \langle 5,2\rangle\rangle$

$$q^{k+1}(\langle 0,0\rangle, \mathrm{r}) = q^k(\langle 0,0\rangle, \mathrm{r}) + \alpha\left(0 + \gamma\cdot 0 + \gamma^2\cdot 0 + \gamma^3\cdot -5 + \dots - q^k(\langle 0,0\rangle, \mathrm{r})\right)$$

...

target

$$q^{k+1}(\langle 5,0\rangle, \mathrm{u}) = q^k(\langle 5,0\rangle, \mathrm{u}) + \alpha\left(0 + \gamma\cdot(-100) - q^k(\langle 5,0\rangle, \mathrm{u})\right)$$

$$q^{k+1}(\langle 5,1\rangle, \mathrm{u}) = q^k(\langle 5,1\rangle, \mathrm{u}) + \alpha\left(-100 - q^k(\langle 5,1\rangle, \mathrm{u})\right)$$

The first algorithm we consider is called Monte-Carlo, and uses directly the return as a target.

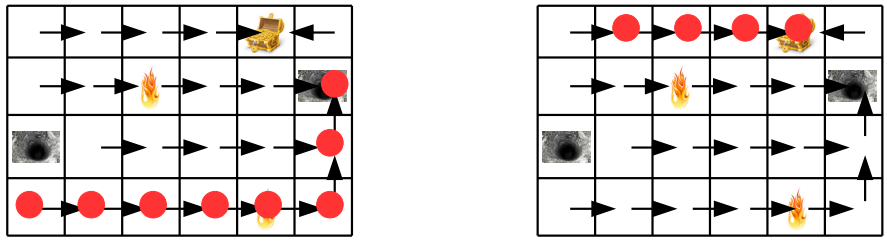
Given the sequence for an entire episode, the sum of the rewards from each state along the sequence is computed, and the value function updated accordingly.

Exploration

UNIVERSITY OF LEEDS

The agent needs to cover every state action pair.

One option (when possible) is to do random restarts:



How can the agent estimate the value of the actions that the policy does not take?

The other option is to *explore*: every once in a while take an action different from the policy under evaluation.

12

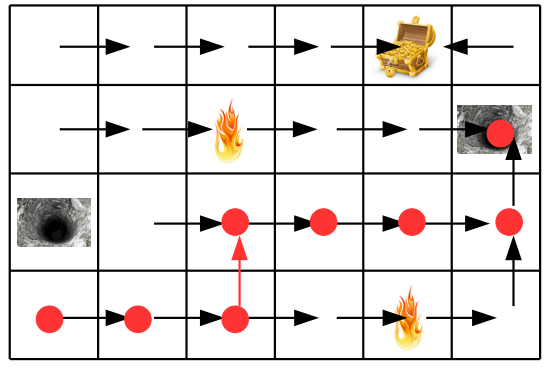
Monte-Carlo gives us a prediction, now we need to consider the control problem. If we just follow a given trajectory we can only learn about that trajectory, and we can't know if there is a better action.

To find out about better actions we need exploration, that is, we need to try some other, often suboptimal, action.

One possible way to explore is through random restarts, getting the agent to start every time with a different state and action. This isn't too efficient, and often not even possible (in physical systems, for instance, one usually can't start from any state they like).

Exploration vs. exploitation

UNIVERSITY OF LEEDS



Exploratory step (in red)

Exploration-exploitation tradeoff

Another way to explore is to stop following what's currently the best policy, and take a different action.

Clearly, there is a chance that the new action will be worse than the previous one, but we have to run this risk to be able to learn, otherwise we'll never try actions that may indeed be better. This is the exploration-exploitation tradeoff, sometimes referred to more dramatically as dilemma.

You experience it when you have to choose a restaurant to go to. Do you go to a restaurant you know you like, or try a new one?

If you never try new restaurants, there may be one you'd love but you'd never find out about it. However, you are likely to go to a restaurant you like less than your current favourite one.

ε-greedy

UNIVERSITY OF LEEDS

For ease of implementation, we will use a very simple rule for exploration called ε-greedy.

At every time step:

Follow the current optimal policy with probability $(1-\epsilon)$

Take a random action with probability: ϵ

A simple exploration strategy is called ε-greedy: take the optimal action with probability 1-ε, and a random action with probability ε.

On Monte Carlo updates

UNIVERSITY OF LEEDS

Monte Carlo updates can only be done after a whole sequence has been collected.

Problem #1: valid only for episodic tasks

Problem #2: it cannot learn **during** an episode

TD methods

UNIVERSITY OF LEEDS

We address those two problems with a new class of algorithms called *temporal difference* methods.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = R_{t+1} + \gamma G_{t+1}$$

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a] \leftarrow \text{Monte-Carlo target}$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$$

$$= E_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, A_t = a]$$

$$= E_\pi[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \leftarrow \text{TD target}$$

To overcome the limitations of Monte-Carlo predictions, temporal difference methods have been developed.

In TD methods, the target is not the return directly, but the immediate reward + the current estimate of the value from the next state.

This means that TD methods base a prediction on another prediction, therefore they *bootstrap*.

Targets

UNIVERSITY OF LEEDS

TD target

$$E[R_t + \gamma q(S_{t+1}, A_{t+1}) | S_t, A_t] = \sum_{s'} p(s'|S_t, A_t)(R_{t+1} + \gamma \sum_{a'} \pi(a'|s') q(s', a'))$$

Two distributions to sample from:

Sarsa On-policy

$$q_{k+1}(s,a) = q_k(s,a) + \alpha(r_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) - q_k(s,a))$$

Expected Sarsa On or off-policy

$$q_{k+1}(s,a) = q_k(s,a) + \alpha(r_{t+1} + \gamma \sum_{a'} \pi(a'|s') q(s', a') - q_k(s,a))$$

Q-learning Off-policy

$$q_{k+1}(s,a) = q_k(s,a) + \alpha(r_{t+1} + \gamma max_{a'} q(s', a') - q_k(s,a))$$

These are three TD algorithms, with slightly different targets.

All of them sample the next state and the reward from the environment, therefore they are learning algorithms (as opposed to planning).

Sarsa also samples the next action from the current policy. In Sarsa, prediction and control and linked: the action used in the update of the prediction has to be the one actually executed next. Therefore, Sarsa is an *on-policy* algorithm.

Expected Sarsa does not sample the action, but uses the entire distribution of *a* policy to determine the value of the next state. In expected Sarsa prediction and control are decoupled: the policy used to make the update may not be the one actually followed by the agent. So, there is a *control* policy and a *target* policy, and the value function will converge to the value of the *target* policy, even if the control policy is used to choose actions. This works as long as the control policy assigns a non-zero probability to every action that has a non-zero probability in the target policy (remember that we can only learn from the actions we try!). Expected Sarsa can be either on- or off- policy depending on whether the target and control policy are the same.

Q-learning also decouples prediction and control, and in this case the target policy is the *optimal* policy, that is the one that always chooses the best action. Therefore, Q-learning is off-policy, because it learns a prediction of the optimal policy, regardless of the policy that the agent is actually following.

Sarsa

UNIVERSITY OF LEEDS

$$\text{target} = R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})$$

$$q_\pi^{k+1}(s,a) = q_\pi^k(s,a) + \alpha\left(r_{t+1} + \gamma q_\pi^k(s', \pi(s')) - q_\pi^k(s,a)\right)$$

Given only one step from the sequence from the example in slide 13

$$\langle\langle 2,0\rangle, \mathrm{r}, 0, \langle 3,0\rangle\rangle \quad \langle\langle 3,0\rangle, \mathrm{r}, -5, \langle 4,0\rangle\rangle$$

$$q_\pi^{k+1}(\langle 2,0\rangle, \mathrm{r}) = q_\pi^k(\langle 2,0\rangle, \mathrm{r}) + \alpha\left(0 + \gamma q_\pi^k(\langle 3,0\rangle, \mathrm{r}) - q_\pi^k(\langle 2,0\rangle, \mathrm{r})\right)$$

TD methods allow the agent to update the value function after each step, and not only at the end of an episode like Monte-Carlo.

However, the prediction is based on the prediction of the next state-action pair, which for a certain time will be affected by the initial value, and will be wrong.

Bootstrapping

UNIVERSITY OF LEEDS

Monte Carlo

$$q_\pi^{k+1}(s,a) = q_\pi^k(s,a) + \alpha\left(g_t - q_\pi^k(s,a)\right)$$ g_t is a sample of the return G_t

Sarsa

$$q_\pi^{k+1}(s,a) = q_\pi^k(s,a) + \alpha\left(r_{t+1} + \gamma q_\pi^k(s',\pi(s')) - q_\pi^k(s,a)\right)$$

another prediction

TD methods *bootstrap:* the new prediction of the expected return is based not only on data (as in Monte Carlo) but also on another prediction.

Q-learning

UNIVERSITY OF LEEDS

Very similar to Sarsa, indeed it is another TD method

$$\text{target} = R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})$$

$$q_\pi^{k+1}(s,a) = q_\pi^k(s,a) + \alpha\left(r_{t+1} + \gamma \max_{a'} q_\pi^k(s',a') - q_\pi^k(s,a)\right)$$

Next action not from the policy: Q-learning is *off-policy*

On vs Off-policy

UNIVERSITY OF LEEDS

On-policy methods, such as Monte-Carlo and Sarsa, learn the value of the policy that is generating the samples.

In order to converge to the optimal policy, such a policy must be iteratively improved (cf. the policy improvement step).

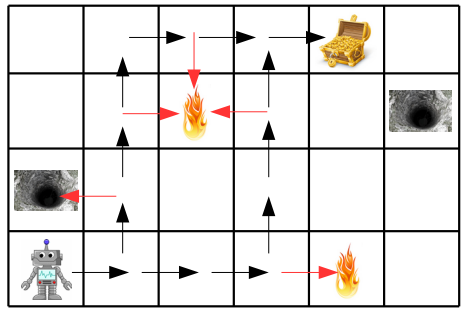
Off-policy methods, such as Q-learning, learn the value function of the optimal policy **directly** (they are the learning version of value iteration in dynamic programming).

Understanding On vs Off-policy

UNIVERSITY OF LEEDS

Agent acting according to an ε-greedy policy.

Do Q-learning and Sarsa learn the same policy?



In this example, the agent is following an ε-greedy policy where the greedy actions are the black ones.

The red actions are possible bad actions taken when ε makes the agent choose a random action.

Since Sarsa estimates the value of the actions that are actually taken, it will assign a lower value to the action of going up from the second state, because there is a slight chance that the agent will fall into the pit and get a very low reward.

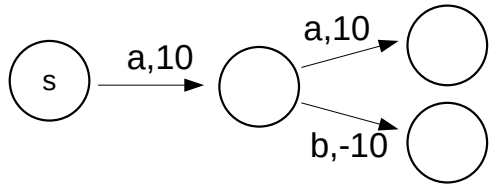
However, to Q-learning both actions in the second state are the same, because falling into the pit does not depend on the environment but on the agent, and Q-learning learns about the optimal policy, which never chooses the action to go into the pit.

A simple numerical example

UNIVERSITY OF LEEDS

Agent acting according to an ε-greedy policy.

$\epsilon = 0.1 \qquad \gamma = 0.5$



Action value learned by Sarsa:

$$q(s,a) = 10 + \gamma(0.9 \cdot 10 + 0.1 \cdot (-10)) = 14$$

Action value learned by Q-learning:

$$q(s,a) = 10 + \gamma(10) = 15$$

Here is another, smaller, MDP that demonstrates the difference between the two algorithms.

History

UNIVERSITY OF LEEDS

- 1957 - Richard Bellman (Optimal Control thread)

- 1911 - Edward Thorndike (Psychology thread)

History

UNIVERSITY OF LEEDS

- (since) 1970s – Andy Barto (UMass Amherst) and Rich Sutton (U of Alberta)

Authors of the main RL textbook: Reinforcement Learning: an Introduction.

For most of us “the book” (http://incompleteideas.net/book/the-book-2nd.html).

- 1989 – Chris Watkins (U of London)

  Q-learning (http://www.cs.rhul.ac.uk/home/chrisw/)

History

UNIVERSITY OF LEEDS

- 1992 – Gerry Tesauro

(… a lot of other stuff…)

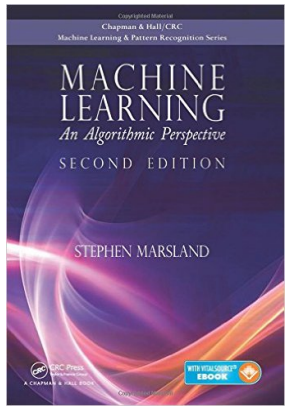
- 2013 – A lot of people…

DeepMind

I'll mention David Silver as you have probably heard of AlphaGo (2015)

Conclusion

UNIVERSITY OF LEEDS



Chapter 11