NIHR | Applied Research Collaboration South West Peninsula

@penCHORD_UoE
@peninsula_ARC

Module 7 : Machine Learning
Session 7F : Reinforcement Learning
Dr Daniel Chalk

"Treasure Hunt"

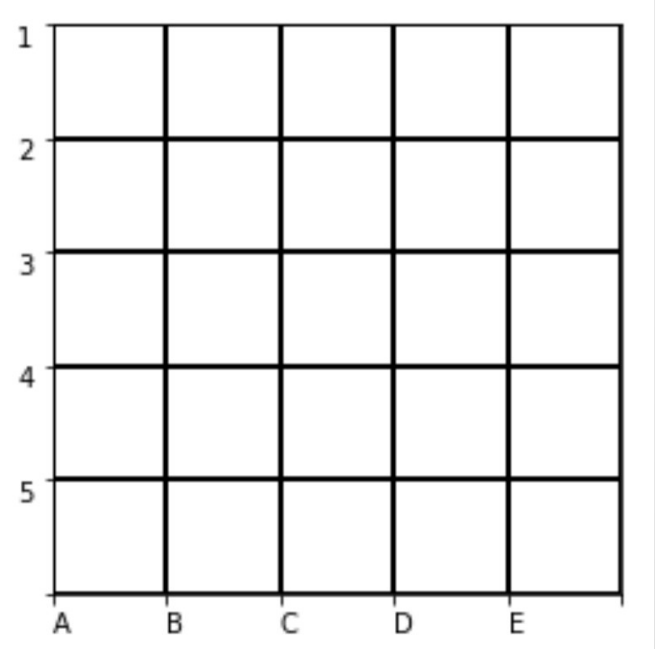
HSMA 5

#hsma5isalive

Let’s Play Treasure Hunt!

It’s time to play Dan Chalk’s Treasure Hunt – a slightly (almost completely) different version of the classic (old) 80s TV series. But it does feature Anneka Rice in a helicopter.

You are presented with a grid of 5 x 5 cells, each of which represents a location that contains treasure. Each location is denoted using a letter (column) and number (row) combination (e.g. C3).



Your task, working as a cohort, is to guide Anneka in her helicopter to various locations on the grid, and dig for treasure. You need to find as much treasure as possible within the time limit.

Let’s Play Treasure Hunt!

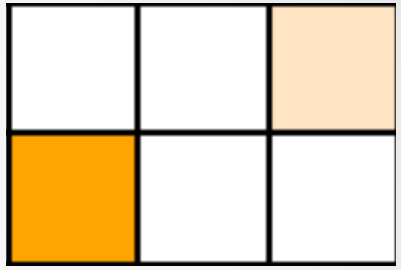
Each location contains treasure (and you can assume there is sufficient treasure in any location so that you won’t run out). BUT the probability of digging up treasure on any turn varies across the landscape.

Each group will play in turn (with a volunteer from the group speaking on behalf of the group). Each group will get 20 turns (so a total of 200 turns for the cohort as a whole – remember you’re all working as an entire cohort to get as much treasure as possible).

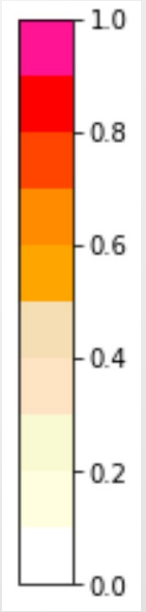
On each turn, the group in play can choose to do one of two things :
1. Dig for treasure in the current location (consumes 1 turn)
2. Move to another location and then dig (consumes 2 turns – 1 to travel to the location, and 1 to dig there. All locations are assumed to be reachable within 1 turn). Note a – if you move on turn 200, you will move but not be able to dig. Note b – the very first turn (for the first group only) does not incur a travel cost – any location can be selected as the starting point.

Let’s Play Treasure Hunt!

After digging in a location, one of two things will happen :
- you will find 1 unit of treasure
- you will not find any treasure



After each dig, you will be presented with an updated grid which will recolour cells based on the success rate you (as a cohort) have had in each cell (ie successful digs / total digs). At the start, the success rate for all cells is 0.



A scale next to the grid shows the colour scheme used.

You will also get an update on your current total treasure every time you dig up some treasure.

You can then choose whether to dig again in the same location, or move and dig somewhere else. If you’re on the final turn for your group, you can only dig (not move and dig).

Treasure Hunt High Score Table

Just for a bit of added competitiveness, I got Elliott to play the game recently. He scored :

54 units of treasure.

Can you beat his score?

Let’s Play Treasure Hunt!

Any questions?

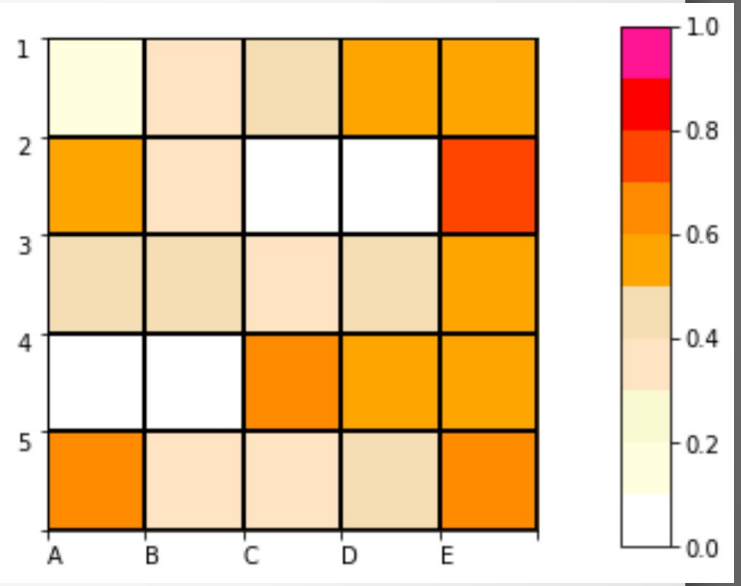
Let’s Play Treasure Hunt!

Let’s Play!

The Treasure Hunt Landscape

Let’s see what the actual treasure “payout” was across the landscape :

	Standard	Standard	Standard	Standard	Standard
1	0.2	0.3	0.4	0.5	0.5
2	0.5	0.3	0.05	0.01	0.7
3	0.4	0.4	0.3	0.4	0.5
4	0.02	0.03	0.6	0.5	0.5
5	0.6	0.3	0.3	0.4	0.6



Let’s Play Treasure Hunt!

Let’s discuss what how you approached this – strategies etc

Reinforcement Learning

Reinforcement Learning is a branch of AI in which an *Agent* takes actions within an *Environment* so as to maximise the amount of *Reward* they accumulate.

Each action that an agent takes will result in either *reward* or *punishment* (which is often simply the absence of reward).

The Agent takes actions that are in part “trial and error”, whilst updating its perception of the likely reward of taking various actions.

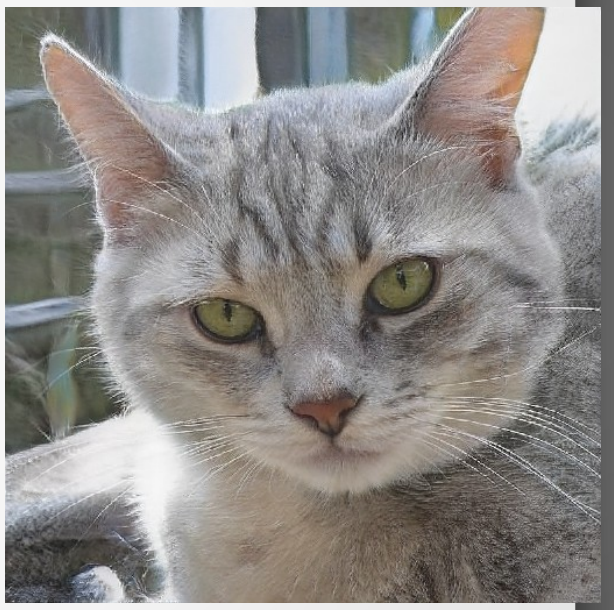
This means that the agent gradually learns to take actions that are more likely to lead to rewards, and avoid actions that are more likely to lead to punishments (or no reward).

Reinforcement Learning

In the animal world, animals typically learn via Reinforcement Learning, especially when foraging for food. For example, they learn to forage in locations that are more likely to yield food.

If we train a dog, we might reward them with treats for good behaviour. This is Reinforcement Learning.

Similarly, cats train us to do things (like give them treats) and reward us by rubbing around our legs, or not relentlessly meowing at us for a short period of time. This is also Reinforcement Learning (indeed, our cat is an expert at it).



Skinner Box

In the world of animal behaviour, there is a famous experiment known as the *Skinner Box Experiment* (also known as an *operant conditioning chamber*) developed by the American psychologist B.F. Skinner.

In the experiment, an animal is placed in a box, where it learns to respond to various stimuli (e.g. pulling levers etc, depending on the animal) to receive a reward (such as food).

The same principle was applied in 2015 to teach pigeons how to recognise images of cancerous cells (by being rewarded by a food pellet when they pecked the button correctly when shown an image of cancerous cells).

They found that, within hours, a pigeon could do better than random, and within a month, a single pigeon's accuracy could rise to around 80%. But showing the images to a flock of pigeons and taking the most popular answer yielded 99% accuracy – equivalent to trained human experts.

Reinforcement Learning

The game you just played was a Reinforcement Learning problem. In fact, it's a particular type of RL problem known as a *Multi-armed bandit problem*.

In a multi-armed bandit problem, an agent chooses between actions that may or may not lead to reward where each action has an underlying reward probability.

A multi-armed bandit problem can be thought of as someone walking into a casino and being presented with a number of *one-armed bandits* (slot machines) that they can choose to play. The player is trying to choose machines such that they maximise their payout (their reward).

Reinforcement Learning

In the Treasure Hunt game, your environment is a grid of cells representing locations. Each of the grid locations are effectively one-armed bandits with different levels of reward pay-out (probability of providing treasure if a dig is made).

You played the game as the Agent. You were making decisions to try to maximise the amount of reward you accumulated. Each action you took (dig or move and dig) led to either a reward (1 unit of treasure being accumulated) or a punishment (no treasure and therefore a wasted turn / two turns).

The strategies you took likely (either consciously or subconsciously) matched a fundamental dilemma in Reinforcement Learning...

The Exploration-Exploitation Dilemma

In Reinforcement Learning, there is something known as the *Exploration-Exploitation Dilemma.*

When a Reinforcement Learning agent choosing how to act, there are two fundamental types of decision it can make :

1. to *exploit* – to take an action that, according to current perceptions / estimates, will yield the highest reward

2. to *explore* – to deliberately take an action that might be sub-optimal according to current perceptions / estimates, but which helps to explore the environment, recognising that the current estimates of how good things are might be wrong.

The Exploration-Exploitation Dilemma

In the Treasure Hunt game, sometimes you exploited (chose to dig / move and dig in a location that seemed to be yielding good levels of reward), and sometimes you explored (chose to move to another location to see if things might be better there).

Balancing exploration and exploitation behaviours in a Reinforcement Learning can mean the difference between poor and strong performance. There are various algorithms that can be used to try to optimise this balance.

Comfort Break

We'll now take a 15 minute break. When we come back, we're going to play the Treasure Hunt game again, only slightly differently.

Treasure Hunt Round 2

We’re now going to play the Treasure Hunt game again but a bit differently.

This time, each group is going to be playing **against** each other. In addition, you won’t be making choices yourself. Instead, I have coded a (very) simple Reinforcement Learning agent that will play for you.

Treasure Hunt Round 2

Here's how the Agent (who is going to play on behalf of each team) works.

The Agent maintains an estimate of the "payout" probability of each location in the grid. At the start, these estimates are generated completely randomly (below numbers illustrative only).

0.5	0.2	0.3	0.1	0.7
0.6	0.99	0.2	0.1	0.2
0.05	0.15	0.8	0.99	0.85
0.3	0.7	0.2	0.1	0.2
0.4	0.6	0.1	0.2	0.7

Treasure Hunt Round 2

On each turn, the agent will either exploit or explore. Whether the agent exploits or explores on each turn is determined using a simple probability.

If the agent exploits, then it will choose to dig (or move and dig) at the location that it currently estimates to have the highest reward payout (if there is a tie, then it will dig at the first location with the tied estimate that it encounters reading left-to-right, top-to-bottom).

0.5	0.2	0.3	0.1	0.7
0.6	0.99	0.2	0.1	0.2
0.05	0.15	0.8	0.99	0.85
0.3	0.7	0.2	0.1	0.2
0.4	0.6	0.1	0.2	0.7

If exploiting with these estimates, the agent would dig (or move and dig) here

Treasure Hunt Round 2

If the agent explores, then it will simply choose any location at random, regardless of the current estimate it holds about the location's expected payout.

0.5	0.2	0.3	0.1	0.7
0.6	0.99	0.2	0.1	0.2
0.05	0.15	0.8	0.99	0.85
0.3	0.7	0.2	0.1	0.2
0.4	0.6	0.1	0.2	0.7

Treasure Hunt Round 2

After each dig, the agent will update the estimate exactly as we did before – by calculating the number of successful digs in that location / total number of digs in that location.

If the agent dug here and got treasure, and it's the first time they dug here, this value will become 1.0

0.5	0.2	0.3	0.1	0.7
0.6	1.0	0.2	0.1	0.2
0.05	0.15	0.8	0.99	0.85
0.3	0.7	0.2	0.1	0.2
0.4	0.6	0.1	0.2	0.7

Treasure Hunt Round 2

The agent will play using the same rules as you did. It has 200 turns. Moving (other than the first move) costs a turn. And, crucially, it does not know the real payouts of each location.

Here's how we'll play. I'm going to ask each group to play, in turn. Each group must decide only one thing – how do they want to set the probability that the agent **exploits** on any given turn (any value between 0 and 1, with a value of 1 meaning the agent would *never* explore, and a value of 0 meaning they would *always* explore).

The agent will then play the game on your behalf using the exploitation rate you specified. At the end of each game, we will find out how much treasure it accumulated. Each group should have a chat in their Slack group to decide what rate they will use - this may change as they observe other groups' success or failure, so to be fair, we will randomly decide the order of the groups.

Treasure Hunt Round 2

Let’s pick the first group, and then we’ll give them a couple minutes to chat on Slack, given they’re first up.

Then we’ll play!

Keep track of your group’s score – if someone can post it in the module 7 channel on Slack after each turn that would be great.

Treasure Hunt Round 2

We have a winner (hopefully!). What did we observe happening here?

Learning Rates

So far, our agent has updated its estimate of each location based solely on the calculated success rate in that location.

However, in a real RL model, we would tend to use a slightly more sophisticated way of updating estimates that allows us to tweak the rate at which our agents learn. This allows us to strike a balance between the importance an agent places on the latest experience vs its memory of prior experiences.

One simple way we can do this is using something known as a *Linear Operator Learning Rule*.

Linear Operator Learning Rule

Here is an example of a simple Linear Operator Learning Rule that can be used for Reinforcement Learning (indeed, this is the rule I used for my PhD Bumble Bee model) :

$$q(i)_{new} = \beta s(i) + (1 - \beta) q(i)_{old}$$

This might look complicated, but it's actually really straightforward :

$q(i)_{new}$ is the new quality / payout estimate given to an action

$q(i)_{old}$ is the previous quality / payout estimate given to an action

$s(i)$ is the value of the latest reward sample (either 0 or 1 in our problem)

β is the learning rate parameter, which essentially determines how much the latest sample should influence the estimate vs the cumulative memory of perceptions of this action's payout.

Linear Operator Learning Rule

$$q(i)_{new} = \beta s(i) + (1 - \beta) q(i)_{old}$$

If β is 1, then the agent will only ever take the latest sample and forget everything it knew before (the right hand side of the equation will be eliminated). So if it samples a 1 (e.g. digs treasure) then it thinks the location has a perfect payout rate (1.0). If it samples a 0 (e.g. digs nothing) then it thinks the location has no payout (0.0).

If β is 0, then the agent ignores all new information (the left hand side of the equation is eliminated) and never updates its original estimates.

Between these extremes, as β tends towards 1, the agent places more importance on the latest sample, and less on the cumulative memory of its estimate of the payout.

Treasure Hunt Round 3

We're now going to play the Treasure Hunt game once more. This time, the agent will use a Linear Operator Learning Rule to update its (initially random) estimates.

And so, as you have probably guessed, this time each group must decide on two parameter values for the agent :
- the exploitation rate (as before)
- the learning rate β to be used in the Linear Operator Learning Rule.

Remember – the agent is now updating estimates very differently to before, so an exploitation rate that worked well before may or may not work well here.

Again, we will randomise the order of the groups.

Treasure Hunt Round 3

Let’s pick the first group, and then we’ll give them a couple minutes to chat on Slack, given they’re first up.

Then we’ll play!

Keep track of your group’s score – if someone can post it in the module 7 channel on Slack after each turn that would be great.

Treasure Hunt Round 3

We have a winner (hopefully!). What did we observe happening here?

Deep Q Neural Networks

There is a newer branch of Reinforcement Learning known as *Deep Reinforcement Learning* which uses *Deep Q Neural Networks*.

This approach combines principles of Reinforcement Learning with Deep Learning (Neural Networks).

They use a concept known as *Q*, which represents the expected maximum long term reward for a particular action from a given state. This allows the agent to learn that taking certain actions which may result in short-term pain can lead to long-term gain.

We do not currently cover Deep Q Networks in HSMA, but you can find an excellent 2.5 hour introductory tutorial from our colleague Mike Allen here :
https://www.youtube.com/watch?v=A9CFqcyI4v8

Exercise 1

We'll now take a short 5 minute break. Then, you'll work in your groups to come up with some ideas about how Reinforcement Learning approaches could be used in your own organisations.

Before you do this, you should watch these two short videos from Mike to give you some inspiration :
- The Learning Hospital : Combining Deep RL with Hospital Simulation (https://www.youtube.com/watch?v=Rm1bPdBGs00) - 6 minutes
- Using Deep RL Agents to solve the Ambulance Location Problem (https://www.youtube.com/watch?v=UYJtOLYcOU8) - 9 minutes

Here's the timeline :
- Break (5 mins)
- Watch 2 x videos (15 mins)
- Discussion in your groups (30 mins)
- Report back to whole group (10 mins)