SIT215 – Artificial and Computational Intelligence Project
Investigating Reinforcement Learning

Reinforcement learning (RL) is a subfield within machine learning focused on teaching agents how to behave in their environment through the use of rewards and penalties. Environments generally contain complex decision problems, commonly represented as Markov Decision Processes (MDP). Reinforcement learning attempts to create solutions for problems by teaching agents to take actions that will maximise their long term reward, estimating a function over the MDP.

At the most abstract level, RL operates by an agent *A* taking some action *n* in an environment *E*, and adjusting behaviour according to some reward(s) *r* provided by that environment. Punishment for undesired behaviour is achieved through negative reward. The agents understanding of expected reward, given state, can be represented in a policy *P.* The agent can take an action it expects to lead to the highest reward given the current state *S* of the environment, and allow its behaviour to be changed through environmental feedback. An agent acting based on its policy and current state, then updating policy based on reward can be modelled:

$$A_{P(S_1)} \rightarrow N_1 \;;\; P'_{S1} \rightarrow E(R \mid S_1, N_1)$$

This report explores the application of several types of RL agent in 3 different environments provided by OpenAI gym. A project was built in Python 3 that implements different types of learning agents in an engine applies them to the gym environments. Agents are trained, evaluated, tracked and results are charted in a time-series. The code is available here: https://github.com/PhilipCastiglione/SIT215_Project

Taxi Problem

The taxi problem involves an environment designed for training an autonomous agent to operate a taxi service. A static map contains roads, pick-up/drop-off locations and barriers. At the beginning of each scenario, a passenger waits at one of the 4 locations with a goal destination. The agent must travel to the pick-up location, pick up the passenger, travel to the drop-off location and drop them off, avoiding pick-up or drop-off actions elsewhere or unnecessary movement.

The problem is suitable for modelling hierarchical reinforcement learning because it is episodic and has clear subproblems (go pick up the passenger, travel to the destination and drop them off).

The OpenAI gym environment for the taxi problem provides an observation of the environment that is a number representing a state which fully describes the current state of the environment. There are 500 states, which we derive from 5 possible locations for the passenger (4 locations + in the taxi) x 4 locations for the destination x 25 (5 x 5 grid) locations for the taxi. The environment provides the following rewards/penalties for actions:

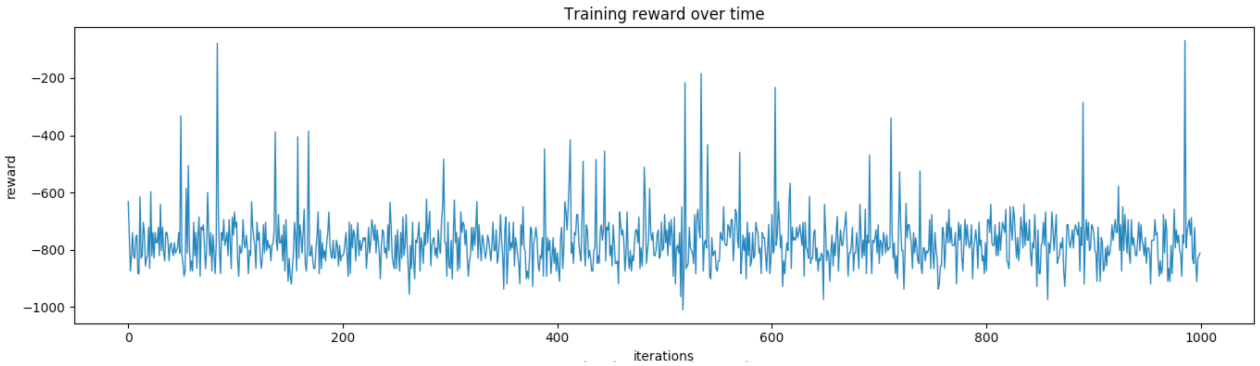
Action	Reward (-Penalty)
Move (N/E/S/W)	-1
Valid Pick-up	-1
Invalid Pick-up	-10
Valid Drop-off	20
Invalid Drop-off	-10

Two agents were applied to the taxi environment: a random agent and a q-learner agent.

Random Agent

The random agent takes an action sampled from the available action space, acting with no intelligence.

Below is a chart generated by the training engine. The x-axis represents training epochs and the y-axis represents cumulative reward across a complete epoch. If an agent effectively learns, the cumulative reward over time should trend upwards asymptotically. We use reward as the ultimate measure of learning and agent success, as reward is comparable across agents and other outcomes are essentially only proxies for reward, assuming it is effectively modelled.



TAXI RANDOM AGENT: REWARD OVER TIME

The random agent reward is consistently negative as random actions will very rarely result in a successful pick-up and drop off of the passenger. The random agent does not learn.

Q-learning Agent

Q-learning is a reinforcement learning approach that aims to learn a policy, represented in a table, for optimal choices to actions to maximise reward, given a state.

Implemented as an algorithm, q-learning will observe the environment state (*s*), then based on some arbitrary rate of epsilon (ϵ) the agent will choose to either "explore" the environment or "exploit" the existing policy. If exploring, the agent will select a random next action (*n*) in the same way as the random agent. If exploiting, the agent will refer to its current policy, or "q table" (*Q*), and choose the best next action to maximise reward.

Next, to learn, the agent will update its q table using the existing policy, some learning rate alpha (α), the reward (*r*) provided by the environment from the action, and the predicted reward associated with the next best action (*n'*) and the new observed state (*s'*) using the current policy, with some future discount factor gamma (γ). The update method is given by:

$$Q'sn = (1 - \alpha) * Qsn + \alpha * (r + \gamma * Qs'n')$$

Note that α, γ and ϵ are hyperparameters to the algorithm.

Theoretically, as the algorithm explores the environment and applies its policy, the policy will continually be updated to improve the accuracy of reward values associated with each choice of action for a given state.

An agent employing Q-learning was applied to the taxi environment.

TAXI Q-LEARNING AGENT: REWARD OVER TIME

We can see that after around 2,000 epochs, the agent has learned how to pick up and drop off the passenger to achieve the maximal reward. Clearly, this agent substantially outperforms the random agent, consistently achieving positive or near positive reward, as it selects actions that fit the state of the problem.

Q-learning is a good fit for this problem because there are discrete state and action spaces of modest size and deterministic outcomes from actions. More complex problems may be difficult or impossible to model entirely in a table.

Cart-Pole Problem

The cart-pole problem is a continuous control problem where a cart moves along a frictionless track, with a pole connected by an un-actuated joint balanced on top. The goal in the environment is to move the cart so that the pole does not fall over.

The cart-pole problem state space is represented by four floating point numbers, the cart position on the track, the carts velocity, the pole angle and the pole velocity (at the tip). The only actions available are to exert force to move the cart right, or left. The environment terminates if the pole angle reaches ±12 degrees, or if the cart position is ±2.4 units from centre. As with all gym environments, the episode also terminates after 200 time steps.

The most immediate difference between the cart-pole problem and the taxi problem is the nature of the state space. While the taxi problem has a discrete state space, the cart-pole problem state is continuous across 4 variables. Second, while the taxi problem has a clear set of tasks and goals that are broken into sub-problems, the cart-pole environment has only one goal, but a less clear relationship between current and future actions.

Reward is modelled simply; any time step in which the episode continues accrues a reward of 1. If the episode ends for any of the previously mentioned reasons, rewards cease. With t representing a time step, rewards are therefore calculated:

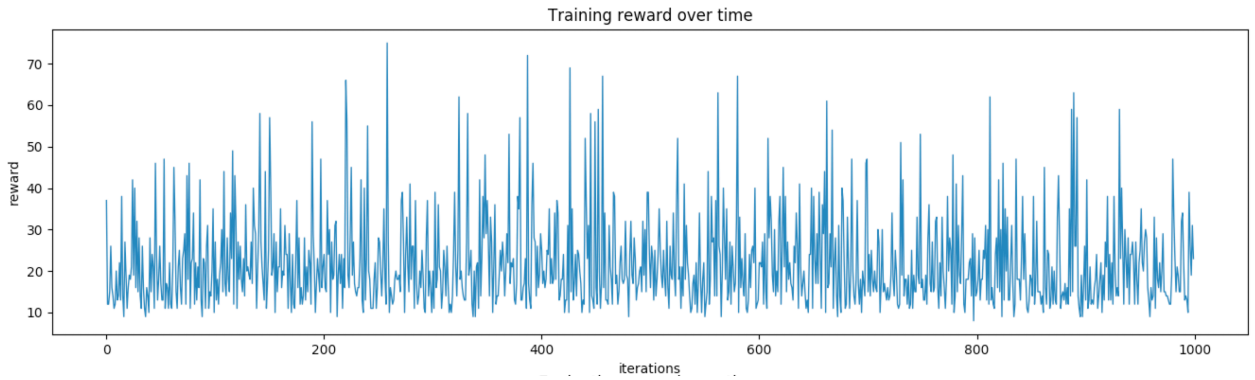
$$R_{t1} = R_{t0} + 1$$

A reinforcement learning agent will learn to take actions that avoid cessation of the scenario by balancing the pole and staying within cart boundaries.

Initially, two agents were applied to the cart-pole environment: a random agent and a q-learner agent.

Random Agent

The random agent was implemented as a baselining exercise and performed as expected. The agent often fails to balance the pole rapidly and receives low reward scores, occasionally randomly achieving slightly better results. As expected, no learning occurs.



CART-POLE RANDOM AGENT: REWARD OVER TIME

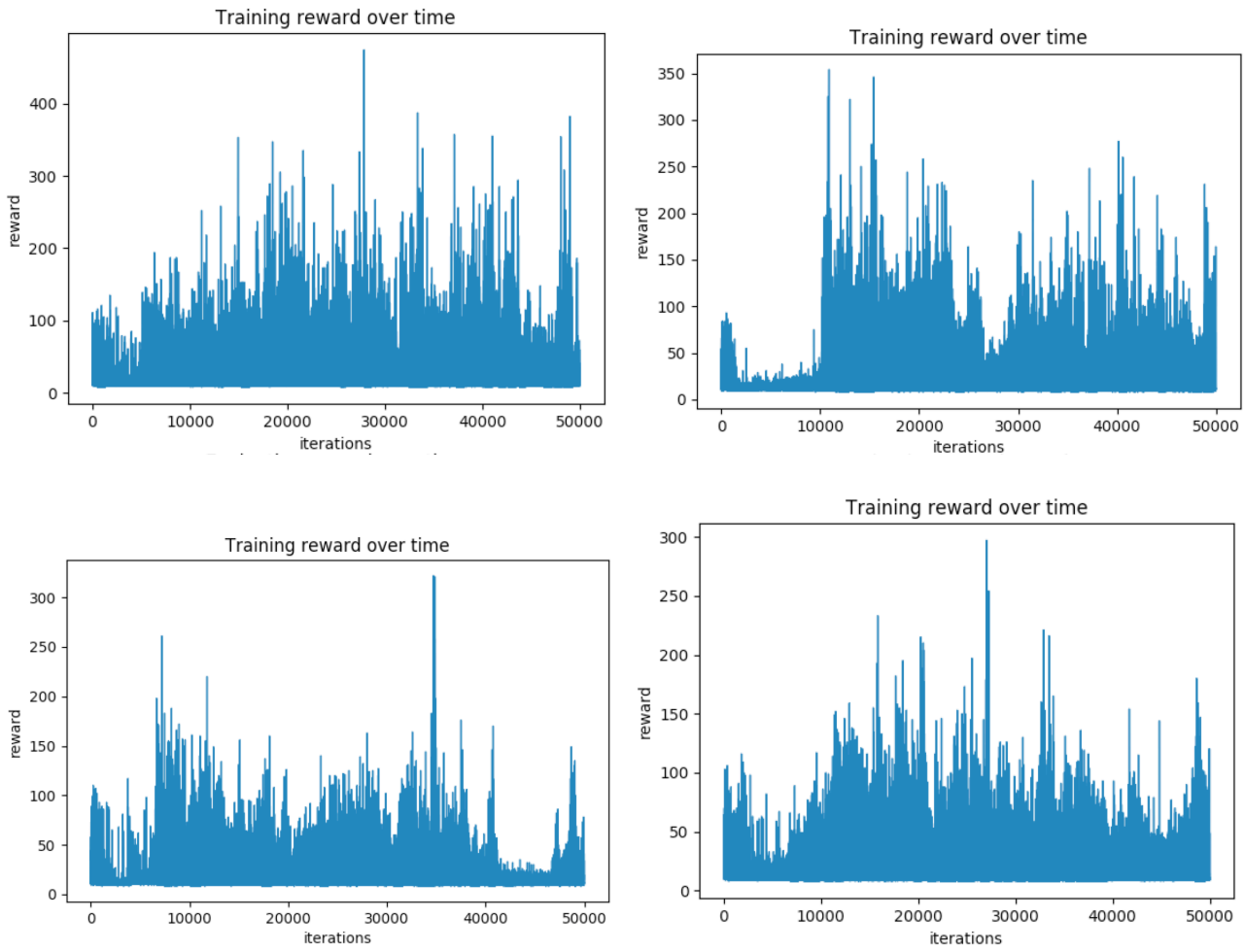
Q-learning Agent

Applying a q-learning algorithm to the cart-pole problem presents significant challenges. First, to represent continuous variables in a discrete space as required for a q table, those variables must be quantised. Particularly given that two of the four variables have infinite range, upper and lower bounds must be determined. A number of discrete buckets must also be chosen for the representation of each variable

For each of the four state space variables, the upper and lower bounds, and number of buckets become new hyperparameters to the algorithm.

After spending many hours attempting to tweak the original hyperparameters to the q-learning agent, and find appropriate values for the quantising process, it was concluded that q-learning is not a very effective approach to solving the cart-pole environment.

While the average reward is still much better than the random agent, from the training results it is visible that q-learning is inconsistent with respect to its ability to learn the optimal policy for maximising reward.



CART-POLE Q-LEARNING AGENT 1-4: REWARD OVER TIME

There are periods when learning appears to occur, and periods when it appears as though the policy actually becomes mislead, perhaps due to overweighting some action based on greedy reward that temporarily leads to higher rewards, but at the cost of developing better long term policy. An example of this would be where the cart balances the pole but edges too far from the centre and ends the episode.

It is possible that with substantial time or computational resources, hyperparameter optimisation (ie. using a genetic algorithm or grid-search) could result in the magic set of numbers that would produce an effective q table for the cart-pole environment.

In summary, q-learning does not easily achieve the optimal policy for the cart-pole environment in the same way that it did for the taxi problem. A discrete, episodic problem with a granular reward structure and modest state space is ideal for this technique.

Frozen Lake Problem

The frozen lake problem is another environment provided by the OpenAI gym. The agent must guide a character through a static map towards a goal tile, while avoiding traps.

The environment has discrete observation and action state spaces and is episodic. In these ways it is superficially like the taxi problem. However, two key differences exist. First, the environment is stochastic; taking an action does not always lead to the same outcome as the ice is slippery and you may move somewhere you did not intend. Second, rewards are not distributed across action outcomes in a granular way to encourage shaping of agent behaviour. If you succeed you get 1 reward, if you fail in any way you get 0. These two differences substantially increase the difficulty of training an agent.

The state space contains 16 states, for the location of the character on the map, and 4 actions (up, left, down, right, though those actions do not lead deterministically to the next state. After taking any number of actions (less than the 200 limit imposed by the gym) the problem can be abstracted to the following outcome where S_H is a state in which the character ends in a hole and S_G is a state in which the character ends at the goal, R_0 is a reward of 0 and R_1 is a reward of 1:

$S_H \rightarrow R_0$

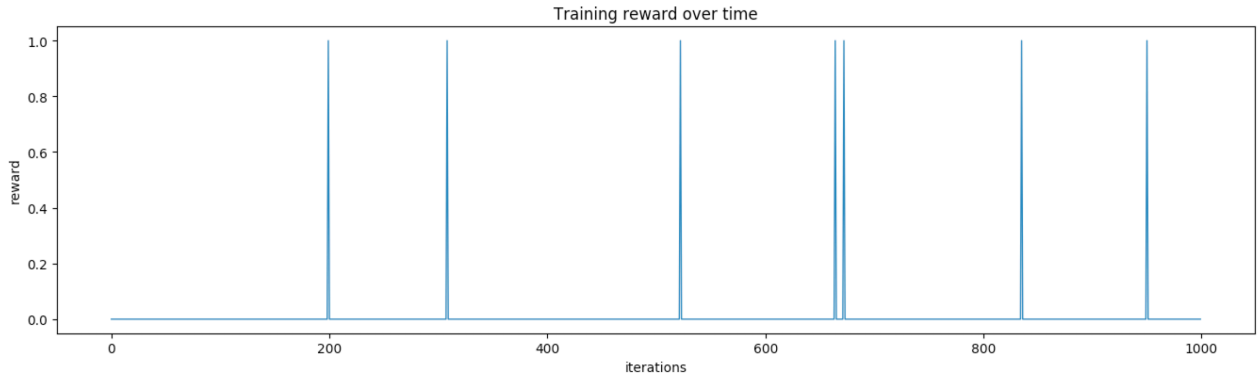
$S_G \rightarrow R_1$

A reinforcement learning agent will therefore learn to avoid the holes and to seek out the goal.

Initially, two agents were applied to the frozen lake environment: a random agent and a q-learner agent.

Random Agent

The random agent was implemented as a baselining exercise and performed as expected. The agent often falls into a hold and receives 0 reward, occasionally randomly reaching the goal instead.



FROZEN LAKE RANDOM AGENT: REWARD OVER TIME

Note that interpretation of this chart may not be intuitive; as reward is always zero or one, the density of the 1 scores represents success, and therefore learning. As expected, no learning occurs for the random agent, and the vast majority of episodes result in 0 reward.

Q-learning Agent

Structurally, application of the q-learning agent to the frozen lake problem takes the same form as application to the taxi problem. Discrete observations of the environments state are mapped to a the set of actions available to the agent with associated expected rewards. As the agent explores and exploits, the q table containing this mapping becomes more accurate.

What we see empirically when applying the q-learning agent to frozen lake is that the agent rapidly learns a set of moves that leads to success, with some probability of failure due to the randomness in the environment. Then, it essentially repeats that set of moves with minor variation, as the deviation caused by exploration doesn't find a set of moves with greater reward.

FROZEN LAKE Q-LEARNING AGENT: REWARD OVER TIME

Perfect performance is not possible in this environment, yet the q-learning agent still achieves optimal behaviour despite stochasticity and limited reward structure.

Temporal Difference learning

Temporal Difference learning (TD learning) is a category of reinforcement learning techniques that use knowledge of previous observations from the environment to update a policy for predictions of the future. Q-learning is an algorithm within the TD learning category. Comparing Q-learning to TD learning is slightly unclear given TD learning is the greater category to which Q-learning belongs, so this investigation considers a specific implementation of TD learning called SARSA.

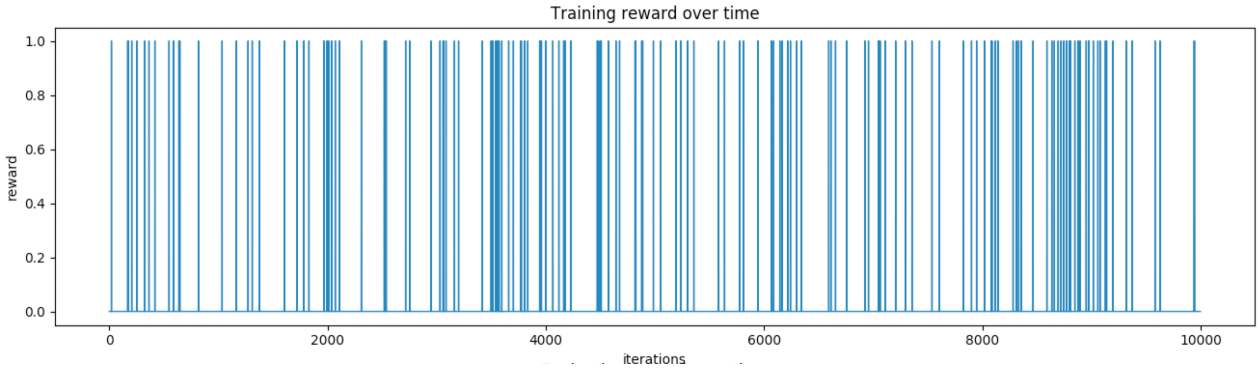
SARSA is an algorithm with a name that stands for its main loop: state–action–reward–state–action. This algorithm follows the same conceptual pattern as q-learning, updating its policy using the reward for an action and expected reward from the next observation. However, it differs from q-learning in that it uses a probability weighted sum to select the next actions reward score, rather than always choosing the action that the current policy expects to lead to the highest reward.

The model for this algorithm differs in that *n** replaces *n'*. *n** represents the action selected from the next states actions using a probability weighted sum.

$$Q'sn = (1 - \alpha) * Qsn + \alpha * (r + \gamma * Qs'n*)$$

Frozen Lake TD learning Agent

The TD learning agent using the SARSA algorithm does not perform as well in the frozen lake environment. Successful reward events are substantially fewer than achieved by the q-learning algorithm.



FROZEN LAKE TD-LEARNING AGENT (SARSA): REWARD OVER TIME

While it's difficult to know the exact reason for the relatively worse performance, it is notable that SARSA does not update policy weights using the best next move for the current state like q-learning does. Given that the frozen lake environment has sparse rewards and requires effective long term planning, it is possible that the SARSA algorithm is less effective for this problem. The

SARSA algorithm may not be able to update the policy towards the theoretical ideal in this environment because it does not lock in on the first successful pathway to a reward and aggressively choose it, like the q-learning algorithm does.

Cart-Pole TD learning Agent

Applying the TD learning agent to the cart-pole problem is similar to applying Q-learning, with some key differences. The use of probabilistic action selection for policy updates means the SARSA algorithm tends to converge to optimal values more slowly, but more predictably. Additionally, it appears to make the agent more resilient to misguided learning events and therefore more stable over time. It is also more resilient to early events and consistently forms the same training pattern.

CART-POLE TD-LEARNING AGENT (SARSA): REWARD OVER TIME

The average reward isn't substantially different to q-learning, but the training results are much less spiky. While this agent isn't meeting the criteria for solving the cart-pole environment, further tuning of hyperparameters - particularly the amount and ranges of buckets - might substantially improve the results.

Finally, we compare empirical results for the q-learning algorithm with the other implemented TD learning algorithm SARSA for the cart-pole environment over a smaller number of epochs to illustrate their different mechanics.

CART-POLE Q-LEARNING AGENT 5K EPOCHS: REWARD OVER TIME

CART-POLE TD-LEARNING AGENT (SARSA) 5K EPOCHS: REWARD OVER TIME

The q-learning algorithm in its initial phase clearly performs well but through learning, weights are adjusted in such a way that it is not consistently learning to effectively exploit the environment and maximise reward. Run for more epochs, some phase may emerge with more effective behaviour, but the algorithm is not ideal for the problem.

By comparison, the SARSA algorithm starts poorly and quickly adjusts its policy through learning to some higher reward state, quite consistently.