

Reinforcement Learning

Akhil Jain

September 1, 2017

Version: 1.0

University of Bielefeld

Faculty of Technology
Neuroinformatics Group

Report

Reinforcement Learning

Akhil Jain

Supervisors Prof. Dr. Helge Ritter
 Dr. Sascha Flier

September 1, 2017

Contents

1	Introduction	2
1.1	Structure	2
2	Theoretical Part	3
2.1	Basics	3
2.1.1	Markov decision process	3
2.1.2	Value function	4
2.1.3	Bellman equation	5
2.1.4	Value iteration	5
2.2	Q Learning	5
2.2.1	Random Q learning	6
2.2.2	Greedy Q learning	6
2.3	Z Learning	6
2.3.1	Z iteration	8
2.3.2	Random Z Learning	8
2.3.3	Greedy Z Learning	9
3	Implementation	10
3.1	Overview	10
3.2	Algorithms	11
4	Result	16
4.1	Small grid (3X4)	16
4.2	Big grid (11X11)	17
5	Conclusion	24
5.1	Future Work	24

Introduction

The importance of Reinforcement learning in our lives today cannot be overstated. They are used virtually everywhere, from financial sector to manufacturing. Still the scope for innovation is truly vast. With the advancement in computing, this algorithm is certainly paving the way for the next artificial intelligence revolution.

Reinforcement learning works in a cycle of sense-action-goals i.e, how to map situations to actions and actions to rewards, so as to maximize the final reward of the goal state. Learners discover the action that yields most reward by exploring the possible actions. Actions may affect not only the immediate reward but also all the subsequent rewards. The two most important distinguishing features of Reinforcement learning are first trial and error search and second delayed reward.

1.1 Structure

To shed more light on this subject, we will discuss about two Reinforcement learning strategies namely Z learning and Q learning and how effective they are in solving the discrete markov decision problems. It also describes different ways of exploring the actions possible. In the end we will summarize our results and compare these two strategies.

Theoretical Part

Our objectives in this section is to describe the Reinforcement learning problem in a broad sense and the key elements of the problem's mathematical structure, such as value functions and Bellman equations. Focus will be on how Markov Decision problem where minimization over the control space is convex and analytically tractable can be solved i.e, How Bellman equation can be exactly transformed into a linear equation. [Torodov]

2.1 Basics

2.1.1 Markov decision process

A Markov decision process (abbreviated as MDP) is a discrete-time state transition system. It describes the environment of Reinforcement learning. Framework of MDP is given as :

- S , a set of states of the environment (where $s \in S$)
- A , a set of actions (where $a \in A$)
- $P : S \times S \times A \rightarrow [0, 1]$, which specifies the passive dynamics. Transition from one state to another under the influence of the action possible in that state and is written as $P(s' | s, a)$, where

$$\forall s \in S, \forall a \in A \sum_{\forall s' \in S} P(s' | s, a) = 1 \quad (2.1)$$

- $R : S \times A \times S \rightarrow \mathbb{R}$, where $R(s, a, s')$ gives the expected reward when transitioning from state s to s'

Furthermore, the learner and decisionmaker is called an agent. Similarly, the thing it interacts with, comprising everything outside the agent, is called the environment which in our case is the MDP. The goal of the agent is to maximize the cumulative

reward. Selection of optimal action in each particular state lets us achieve our objective. For finding the best action, we develop a strategy (also known as "Policy") Π_t where t is the time at which the agent has policy Π . The agent improves the policy over time and also adjusts it after each action.

2.1.2 Value function

If the sequence of rewards received after time step t is denoted as $R_{t+1}, R_{t+2}, R_{t+3}..$, the agent then seeks to maximize the expected return (or minimize if we take the cost when transitioning from one state to another), where the return R_t is defined as a specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

$$R_t = R_{t+1} + R_{t+2} + R_{t+3} + .. + R_T \quad (2.2)$$

Discounting is an additional concept that needs to be discussed. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized.

$$R_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (2.3)$$

Discount rate is bounded to be smaller than 1, a mathematical trick to make an infinite sum finite and further helps in proving the convergence.

Value function estimates the importance of the agent to be in a given state. Its the expected reward when the agent is in the state s and strategy Π is pursued. This is denoted by $v_{\Pi}(s)$

$$v_{\Pi}(s) = E[R_t | S_t = s] = E_{\Pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \quad (2.4)$$

where $E_{\Pi}[\cdot]$ denotes the expected value of a random variable given that the agent follows policy Π , and t is any time step.

2.1.3 Bellman equation

For any policy Π and any state s , the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v_{\Pi} &= E[R_t | S_t = s] \\ &= E[R_t + \gamma R_{t+1} | S_t = s] \\ &= \sum_a \Pi(a | s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma E_{\pi}[R_{t+1} | S_{t+1} = s']] \quad (2.5) \\ &= \sum_a \Pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\Pi}(s')] \quad \forall s \in S \end{aligned}$$

It expresses a relationship between the value of a state and the values of its successor states.

2.1.4 Value iteration

The key idea of Dynamic Programming, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies. We can find optimal policy once we find the optimal value function (2.5) Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. The update rule after each iteration converges towards finding the optimal state value.

2.2 Q Learning

Q Learning is a model free reinforcement learning technique. The parameters of the environment are generally unknown to us and we must approximate the optimal action-value function by interacting with the environment. Typically, monte carlo or temporal difference learning are used to solve this problem. Monte carlo explores each action in every state and compares the reward in the goal state. The one with the highest reward (or minimum cost) will be the optimal action-state value. After large number of iteration, the agent can approximate the action-value function. In temporal difference whole set of actions are not considered, value function is re-evaluated after each action is taken. It forms a union of Dynamic Programming and Monte Carlo methods by learning from individual steps online. The core of Q learning is a simple value iteration update. In this scenario, we use One-step Q learning that has the update rule which is similar to temporal difference learning update rule. We can also use Q function to define the policy, because an action can be chosen just by taking the one with the maximum Q value for the current state.

The Q-learning rule is

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_{t+1}(s, a) + \gamma \min_{a \in A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.6)$$

where α_t = learning rate

$r_{t+1}(s, a)$ = Immediate reward

$Q(s_{t+1}, a)$ = Q value for next state after action a

An essential part of Q-learning in finding an optimal policy is the selection of action a in state s. In Q-learning there exists a tradeoff between selecting the currently expected optimal action, or selecting a different action in the hope that it will yield a higher cumulative reward in the future. To investigate the goal-finding abilities of an agent using Q-learning, we investigate ϵ -greedy and random exploration strategies.

2.2.1 Random Q learning

This exploration strategy is the slowest among all other strategies. We randomly select the action and generally the agent takes a long time to converge. Selection of action can be done by generating a random number in between the total number of possible actions in that state. If the maze or the gridworld is large, to reach to the goal state, it takes a long time but this strategy is guaranteed to converge.

2.2.2 Greedy Q learning

ϵ -greedy exploration is one of the most used exploration strategies. It uses $0 \leq \epsilon \leq 1$ as parameter of exploration to decide performing action using $Q_t(s, a)$. The agent chooses the action with the highest Q-value in the current state with probability $1 - \epsilon$, or a random action otherwise. A larger value for ϵ means more exploration actions are selected by the agent. Randomness is necessary for an agent navigating through all the possible policies to learn the optimal policy. With the experiments, we will find out which value of ϵ is optimal for our setup and how ϵ -greedy performs compare to the random strategy.

2.3 Z Learning

Transforming the discrete optimization problem over actions to a continuous optimization problem over transition probabilities is the main idea of Z learning. The

original formulation of MDP had predefined uncontrolled transition probability which controlled the dynamics but Torodov came up with an alternate formulation of the MDP i.e, to view the controlled probability distribution as a stochastic policy over deterministic actions. This calculates the controlled probability based on the effect of action on passive dynamics of the environment. Framework of MDP in Z learning case:

- S , a set of states of the environment
- $A(i)$ is a set of admissible actions at state $i \in S$
- $r(i,a) \geq 0$ is a cost for being in state i and choosing action $a \in A(i)$
- $P(a)$ is a stochastic matrix whose element $p_{ij}(a)$ is the transition probability from state i to state j under action a

Next we define the controlled transition probabilities as

$$p_{ij}(a) = \bar{p}_{ij} \exp(a_j) \quad (2.7)$$

This guarantees positive probabilities and absolute continuity of the controlled dynamics with respect to the passive dynamics. We initialize our stochastic controlled transition matrix with passive dynamics of the model, i.e, $P(0) = \bar{P}$. $P(a)$ has two constraint, $p_{ij} = 0$ where $\bar{p}_{ij} = 0$ and sum of all the probabilities from one state equals 1 ($\sum_j \bar{p}_{ij} \exp(a_j) = 1$). Since action effects directly on transition probability, the action cost term ($r(i,a)$) can be directly calculated by measuring the deviation between controlled and transition probability and adding it to the immediate reward.

$$r(i, a) = q(i) + KL(p_i(a)||p_i(0)) = q(i) + \sum_{j: \bar{p}_{ij} \neq 0} p_{ij}(a) \log \frac{p_{ij}(a)}{p_{ij}(0)} \quad (2.8)$$

Substituting this equation with optimal value function 2.5, we get

$$v(i) = \min_{u \in U(i)} q(i) + \sum_j \bar{p}_{ij} \exp(a_j) (a_j + v(j)) \quad (2.9)$$

Minimization in this equation subject to constraint and is solved using Lagrange multipliers which gives the optimal control law as

$$a_j^* = -v(j) - \log(\sum_k \bar{p}_{ik} \exp(-v(k))) \quad (2.10)$$

So, if we have the optimal value function we can easily calculate the optimal control law from the equation above. Thus finding the optimal control law indicates the

next best state the agent should go to. Optimally controlled transition probability is given as

$$p_{ij}(a^*(i)) = \frac{\bar{p}_{ij} \exp(-v(j))}{\sum_k \bar{p}_{ik} \exp(-v(k))} \quad (2.11)$$

Substituting the optimal control in equation 2.9 and dropping the min operator, we get

$$v(i) = q(i) + \sum_j \bar{p}_{ij}(a^*(i))(a_j^* + v(j)) = q(i) - \log\left(\sum_k \bar{p}_{ik} \exp(-v(k))\right) \quad (2.12)$$

Taking exponential on both sides and rearranging the terms we get

$$z(i) = \exp(-q(i)) \sum_j \bar{p}_{ij} z(j) \quad (2.13)$$

where $z(i) = \exp(-v(i))$

2.3.1 Z iteration

In [2.13], the Z-learning algorithm is presented as a way to solve the linear eigen vector problem $z = G\bar{P}z$ where G is the diagonal matrix with $\exp(-q(i))$ as its diagonal element, and $\sum_j \bar{p}_{ij}$ is nothing but passive dynamics. One way to solve this equation is by iterating it until the z value converges. The reason it converges is because the stochastic matrix \bar{P} has spectral radius 1 and multiplication with G boils it down to a value ≤ 1 . Iteration is equivalent to the power method (without the rescaling) so it converges to a largest eigen vector with eigenvalue 1.

2.3.2 Random Z Learning

Random Z-learning samples transitions from the passive dynamics P , which essentially amounts to a random walk. Since LMDPs have no explicit actions, each transition (s_t, r_t, s_{t+1}) consists of a state s_t , a next state s_{t+1} and a reward r_t recorded during the transition. Z-learning maintains an estimate Z of the optimal Z value, and this estimate is updated after each transition. The update rule is given as:

$$z(s_t) = (1 - \alpha)z(s_t) + \alpha \exp(r_t)z(s_{t+1}) \quad (2.14)$$

2.3.3 Greedy Z Learning

In Z learning, the expected reward of the next state is dependent on the passive dynamics of the model. In greedy Z learning, we sample according to the optimal control probability (To sample according to more informed transition probability) 2.11 and thus the states with larger estimated Z-values are more frequent which speeds up the approximation and thus converges faster. States with higher Z value will have control probability higher than or equal to passive transition probability. Samples are thus weighted with

$$\frac{\bar{p}_{ij}}{p_{ij}(a^*(i))} \quad (2.15)$$

Thus the update rule becomes

$$z(s_t) = (1 - \alpha)z(s_t) + \alpha \exp(r_t)z(s_{t+1}) \frac{\bar{p}_{ij}}{p_{ij}(a^*(i))} \quad (2.16)$$

Note that the importance sampling is done using the control probability.

Implementation

3.1 Overview

We start by evaluating the proposed algorithm on a finite state and action spaces. The state space is given by a $N \times N$ grid with some obstacles. The agent can move the state to any adjacent ones not occupied by an obstacle and the move succeeds with a transition probability. Additionally, a target state was defined and the agent incurs a cost of 1 at all states and obstacles has cost of 5 other than the target where it incurs no cost, i.e, 0. In the figure given below we have a 3×4 grid state. The letter s indicates the starting state, x is the obstacle and * indicates the final or goal state of the grid.

There are a number of different ways to organize the experience of the agent. We use the episodic formulation. The agent's experience is grouped into episodes or trials. At the beginning of each trial, the environment is reset (agent moves back to starting state). The agent interacts with the environment until the end of the episode or until it reaches the goal state, then the environment is reset again and the cycle continues. The agent is aware of the episode resets. We are using fixed-length episodes and each episode has fixed number of steps the agent can take. A step is equivalent to moving from one state to another. Learning rate (α) is dynamic and decay at each step.

	o		o		o		o	
	o		x		x		o	
	s		x		*		o	

Fig. 3.1: Grid state overview

In the grid world the transition probability of each state has been calculated according to the number of states it can go to and each row indicates the probability of each state. As mentioned in the equation (2.1) it sums up to 1. For example, the state 1 can go right or go down. So the state agent can go to i.e, 2 and 5 has the probability

of 0.5 and rest has 0 probability. The transition probability matrix for the above grid is shown below

Transition probabilities of the grid

0.00 0.50 0.00 0.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.50 0.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.50 0.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.50 0.00 0.00 0.00 0.00 0.50 0.00 0.00 0.00 0.00
0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.50 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.50
0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.50 0.00 0.00 0.50 0.00

Fig. 3.2: Transition probability matrix

3.2 Algorithms

Algorithm 2: Value iteration

Initialize $V(s) = 0$.

while not converged **do**

for all $s \in S$ **do**

 get all possible action in state s

 Update value function: $V(s) = \min_a R(s, a) + \sum_{s'} P(s'|s, a) \gamma V(s')$

 Note that we considered the KL divergence value in calculating the reward to have the same optimal value function as in Linear MDP case

 ($R(s) = R(s) + \lambda KL(a^*(\cdot|s)||P(\cdot|s))$ where $\lambda = 1$ in our case)

end

end

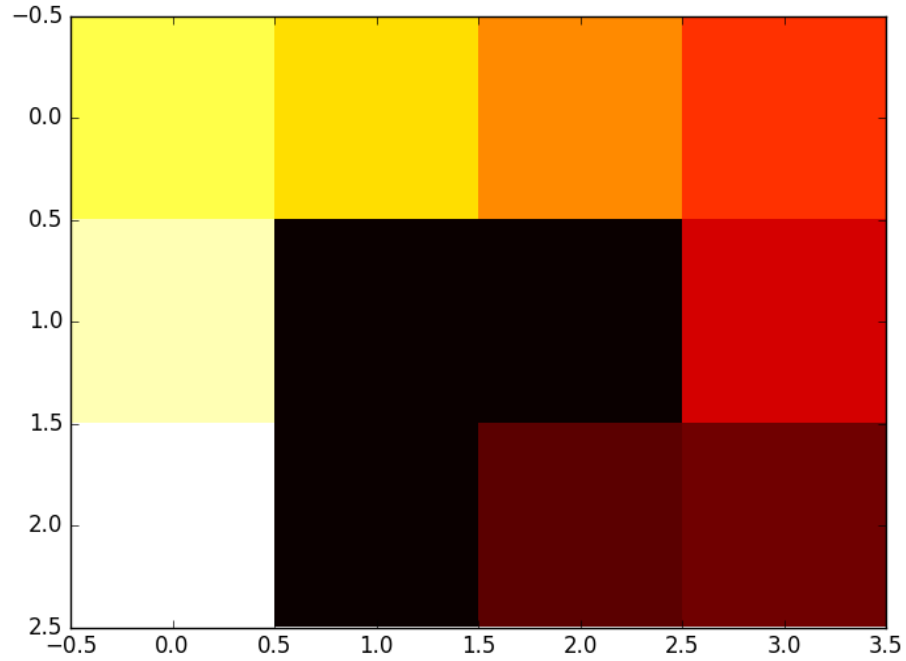


Fig. 3.3: Heatmap of a 3X4 grid obtained from value iteration

Algorithm 4: Z iteration

Initialize $Z(s) = 1$ and $G = \exp(-q(s))$ where s represent the state.

while not converged **do**

for all $s \in S$ **do**

 Update Z value: $Z_{t+1}(s) = G(s) * P(\cdot|s) * Z_t(s)$

end

end

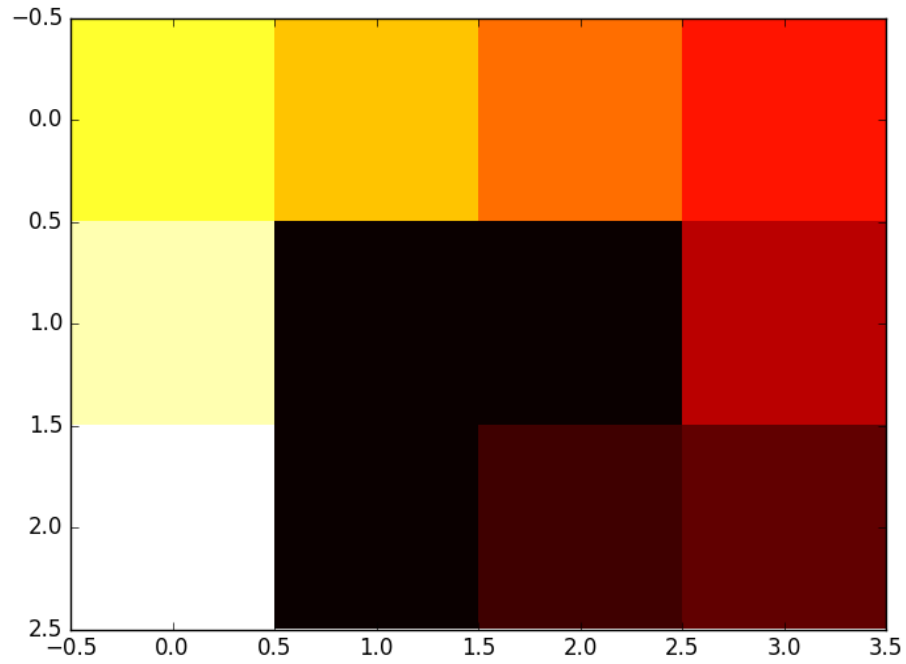


Fig. 3.4: Heatmap obtained from Z iteration

Algorithm 5: Random Z learning

Initialize $Z(s) = 1$

for each episode **do**

 Move the agent to starting state

while not equal to total no. of steps possible in each episode or agent reached the goal state **do**

 Find out all the next possible states

 Randomly select the next state

$$\alpha = \frac{\text{calpha}}{\text{calpha} + \text{step}}$$

$$z[\text{currentstate}] = (1 - \alpha) * z[\text{currentstate}] + \alpha * G[\text{currentstate}][\text{currentstate}] * z[\text{nextstate}]$$

 Move agent to the next state

 Increase the step by 1

end

 Increase the episode by 1

end

Algorithm 6: Greedy Z learning

Initialize $Z(s) = 1$

for each episode **do**

 Move the agent to starting state

while not equal to total no. of steps possible in each episode or agent reached the goal state **do**

 Calculate the control probability of the state (see 2.11)

 Find out next state by sampling the control probability

$$\alpha = \frac{\text{calpha}}{\text{calpha} + \text{step}}$$

$$z[\text{currentstate}] = (1 - \alpha) * z[\text{currentstate}] + \alpha *$$

$$G[\text{currentstate}][\text{currentstate}] * z[\text{nextstate}] * \frac{P(\text{nextstate}|\text{currentstate})}{a*(\text{nextstate}|\text{currentstate})}$$

 Move agent to the next state

 Increase the step by 1

end

 Increase the episode by 1

end

Algorithm 7: Random Q learning

Initialize $Q(\text{states}, \text{actions}) = 0$

for each episode **do**

 Move the agent to starting state

while not equal to total no. of steps possible in each episode or agent reached the goal state **do**

 Get all the action possible on the state

 Get the next state by taking the random action

$$\alpha = \frac{\text{calpha}}{\text{calpha} + \text{step}}$$

 Update Q value:

$$Q(\text{current state}, \text{action}) = (1 - \alpha)Q(\text{current state}, \text{action}) + \alpha * (R(\text{current state}) + \gamma * \min(Q(\text{next state}, \text{all actions possible})))$$

 where γ is discount rate

 Move agent to the next state

 Increase the step by 1

end

 Increase the episode by 1

end

Algorithm 8: Greedy Q learning

Initialize $Q(\text{states}, \text{actions}) = 0$

for each episode **do**

 Move the agent to starting state

while not equal to total no. of steps possible in each episode or agent reached the goal state **do**

 Generate a random number between 0 and 1

if random number $\leq \epsilon$ **then**

 find out the minimum q value of next states possible

 Based on the state found, find out the action

end

else

 select the random action possible on the current state

end

 Get the next state by taking the action

$$\alpha = \frac{\text{calpha}}{\text{calpha} + \text{step}}$$

 Update Q value:

$Q(\text{current state}, \text{action}) = (1-\alpha)Q(\text{current state}, \text{action}) + \alpha * (R(\text{current state}) + \gamma * \min(Q(\text{next state}, \text{all actions possible})))$

 where γ is discount rate

 Move agent to the next state

 Increase the step by 1

end

 Increase the episode by 1

end

Result

For comparison, we use two different size of grid. Accordingly, we have changed the parameters.

4.1 Small grid (3X4)

The heatmap generated is shown above (Figure 3.4). The total number of episodes were 10 and in each episode we had 100 steps. We use learning rate of 100 for both the algorithms. Q learning requires exploration. Thus, we implement an ϵ -greedy policy with $\epsilon = 0.1$. Z learning implicitly contains exploration so we directly use the greedy policy. Entire simulation was repeated 5 times. Approximation error for each step is calculated as:

$$\frac{\max_i |v(i) - \hat{v}(i)|}{\max_i \hat{v}(i)} \quad (4.1)$$

where \hat{v} is the optimal value of state obtained from the value iteration and v is the value of the state at each step.

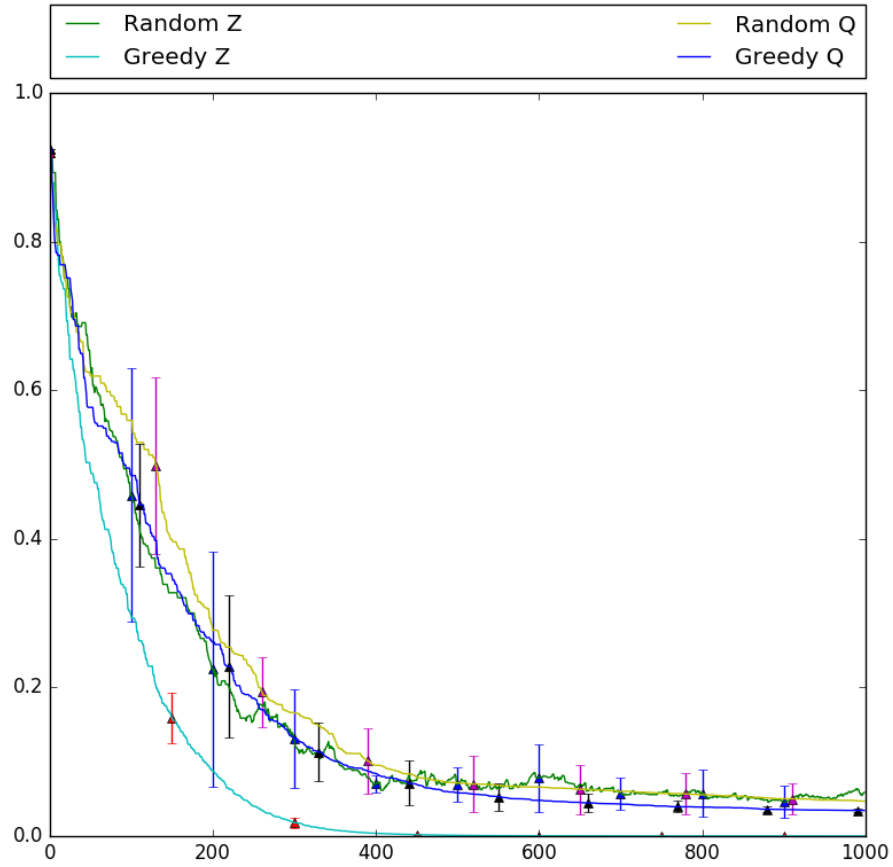


Fig. 4.1: Comparison in small grid case

Both algorithms converge quickly against the optimal state-value function. Importance sampling speeds up convergence slightly in the small Gridworld. Z learning performs slightly better but overall they all converge fastly, which can be contributed to the fact that the exploration in small grids are limited and thus agent in all the cases behave almost the same way, i.e, they follow more or less the same pattern.

4.2 Big grid (11X11)

Heatmap for the grid obtained from Z iteration is shown below. The total number of episodes were 50 and in each episode we had 1000 steps. The α value for calculating the learning rate was 5000 for Z learning and 10000 for Q learning. We use an ϵ -greedy policy with $\epsilon = 0.2$. Finding the optimal parameters is of critical importance. Trying out different values for parameter may give some idea about it but it takes a long time. For example, complexity of the grid in our case determined the learning rate, the higher the complexity the higher the learning rate. Learning rate plays a

very crucial role in deciding the convergence and thus should be chosen wisely. The entire simulation was repeated 5 times and the difference in the behaviour of the agent each time was plotted by the error bar.

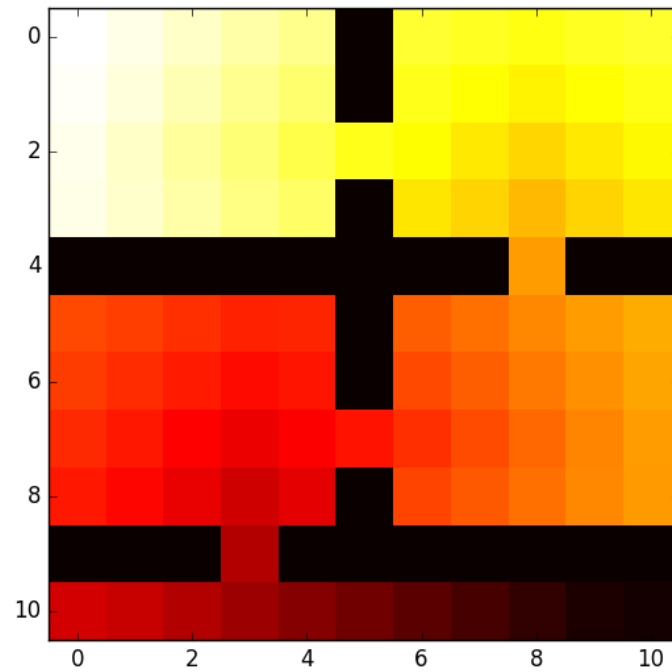


Fig. 4.2: Heatmap of 11X11 grid (generated from the Z iteration obtained state values)

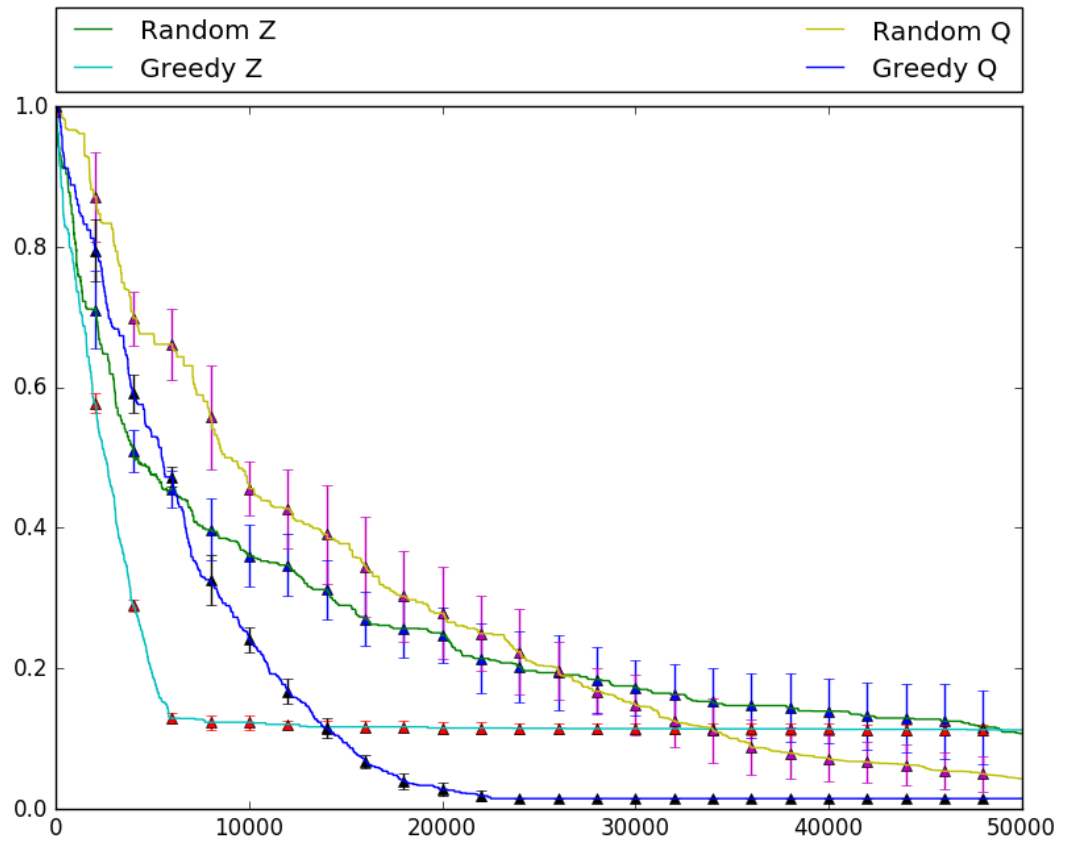


Fig. 4.3: Comparison in big grid case

Number of steps after training 30

Q learning trained path

	*		*		*		*		o		x		o		o		o		o		o	
	o		o		o		*		o		x		o		o		o		o		o	
	o		o		o		*		*		*		*		*		*		o		o	
	o		o		o		o		o		x		o		o		*		o		o	
	x		x		x		x		x		x		x		x		*		x		x	
	o		o		o		o		o		x		o		*		*		o		o	
	o		o		o		o		o		x		o		*		o		o		o	
	o		o		o		*		*		*		*		*		o		o		o	
	o		o		o		*		o		x		o		o		o		o		o	
	x		x		x		*		x		x		x		x		x		x		x	
	o		o		o		*		*		*		*		*		*		*		*	

Fig. 4.4: Q learning optimal path

Number of steps after training 30

Z learning trained path

	*		*		*		*		*		x		o		o		o		o		o	
	o		o		o		o		*		x		o		o		o		o		o	
	o		o		o		o		*		*		*		o		o		o		o	
	o		o		o		o		o		x		*		*		*		o		o	
	x		x		x		x		x		x		x		x		*		x		x	
	o		o		o		o		o		x		*		*		*		o		o	
	o		o		o		o		o		x		*		o		o		o		o	
	o		o		o		o		*		*		*		o		o		o		o	
	o		o		o		*		*		x		o		o		o		o		o	
	x		x		x		*		x		x		x		x		x		x		x	
	o		o		o		*		*		*		*		*		*		*		*	

Fig. 4.5: Z learning optimal path

Here, the speed difference becomes clearer. Greedy Z algorithm is the fastest and the Greedy variant is relatively faster than the random variant. Greedy Z learning doesn't quite converge in the big grid, because we sample according to the control probability and few of the unwanted states were not easily reachable in this grid. However, the agent reaches the goal in the minimum number of steps. The path learned by Q learning and Z learning are a bit different in this grid but both have the optimal path and reached the goal state in the same number of steps. Also in the given plot we can see that the error bar in random case is high which clearly indicates the random behaviour of the agent where similarly in greedy algorithms it follows the same path most of the time.

We also checked the algorithm on easy grid having the same size as above, lowering the complexity does not give any difference in the result which indicates the greedy nature of Z learning. As it is implicitly greedy, we can not converge it by adding some randomness as we did in the case of Q learning where we explore and exploit based on the ϵ value.

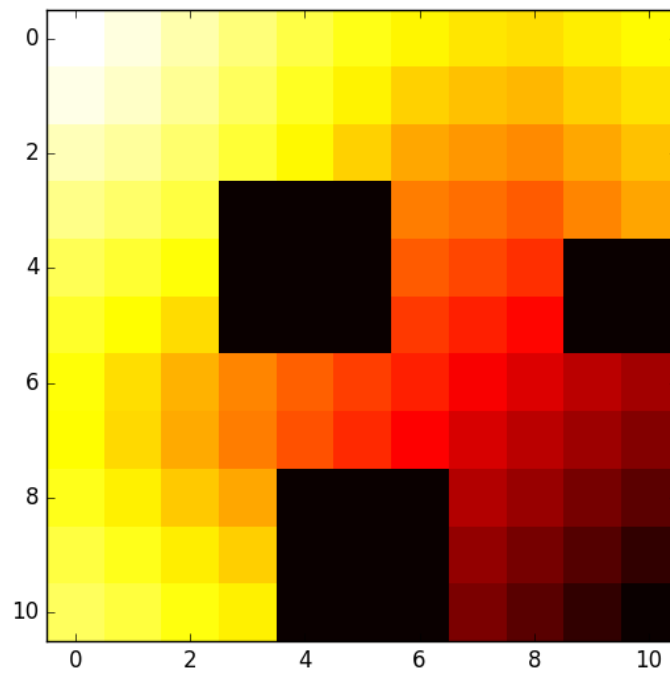


Fig. 4.6: Heatmap of 11X11 grid (Z iteration obtained state values)

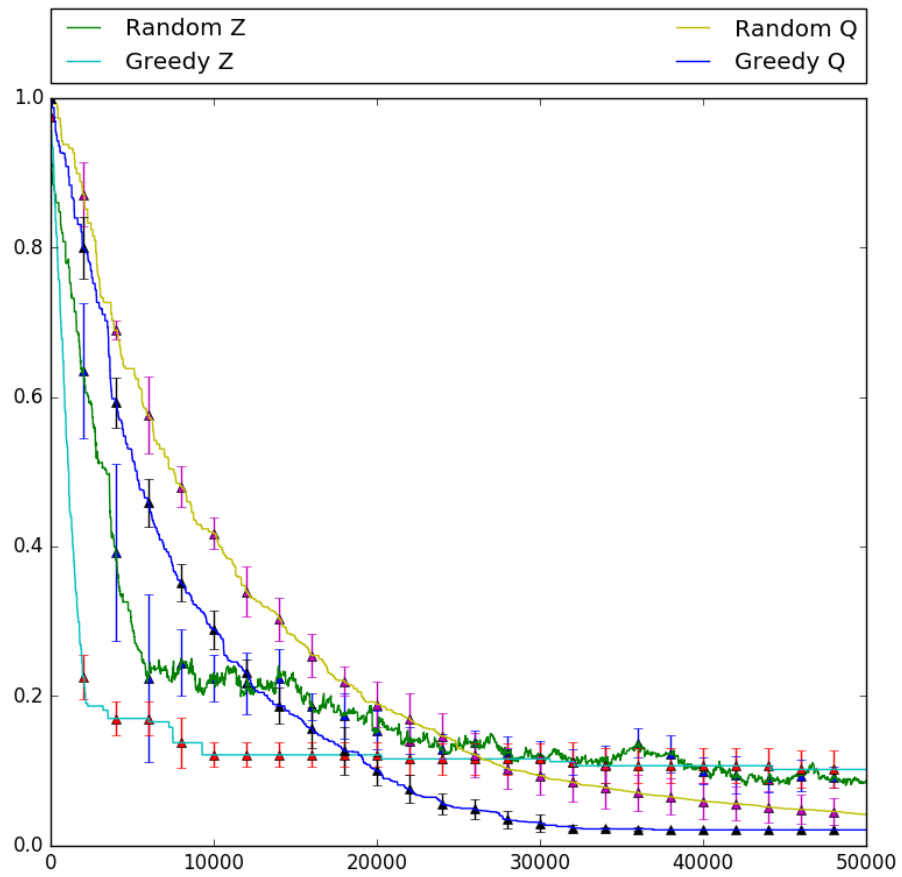


Fig. 4.7: Comparision in less complex grid

Conclusion

In this project, we successfully implemented Q and Z learning algorithms. We did thorough research before implementation and then we finalized them as described in our theoretical section. The proposed algorithms were very stable in finding the optimal path. Similarly, the same was verified by the experiment results we conducted.

5.1 Future Work

Further inquiry could be based on moving from two dimensional world to three dimensional world. It would be really interesting to see how the algorithms behave when we further increase the complexity of the environment.

Papers

- [Torodov] *Linearly-solvable Markov decision problems*. Mar. 2009 (cit. on p. 3).
- [Torodov] *Linearly-solvable Markov decision problems*. Mar. 2009 (cit. on p. 3).
- [SuttonBarto] Sutton Sutton and Barto Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2017, p. 1.
- [Bossert] *Z-Learning auf diskreten Markov-Entscheidungsproblemen mit zustandsabhängigen Aktionen*. Nov. 2013.
- [Froese] *Vergleich von Z-Learning und Q-Learning auf diskreten Markov Decision Problems*. Nov. 2010.
- [Jonsson] *Hierarchical Linearly-Solvable Markov Decision Problems*. Mar. 2016.