CSCCORE2H001AZ2021 AI COURSEWORK

REINFORCEMENT LEARNING

MAISHA SADIA

ID:18009632

Table of Contents

[INTRODUCTION: 2](#_Toc103912381)

[Diagram, engineering drawing

Description automatically generated 2](#_Toc103912382)

[Markov decision process(MDP) 2](#_Toc103912383)

[transition probability 3](#_Toc103912384)

[expected return and discounted return 3](#_Toc103912385)

[policy and value functions 3](#_Toc103912386)

[optimization: 4](#_Toc103912387)

[EXPERIMENT AND DISCUSSION: 4](#_Toc103912388)

[Bibliography 9](#_Toc103912389)

# INTRODUCTION:

The new era of Artificial Intelligence claims to mimic and outcompete the human brain. Artificial Intelligence (AI) is the use of a computer model intelligent behaviour where there is minimal human intervention (Hamet and Tremblay, 2017). Reinforcement learning is an area of Machine Learning where it learns from the experiences in an environment in order to maximise some given reward. Reinforcement learning algorithm studies the behaviour of subjects in such environments to optimise the behaviour. Traces of reinforcement learning is found in the early days of cybernetics, statistics, psychology, neuroscience, and computer science (Kaelbling, Littman and Moore, 1996).

Reinforcement Learning is a scenario faced by the agent, where the agent must learn the behaviour through exploring and exploiting the environment. According to the researcher Kaelbling, Littman and Moore (1996), there are two ways of solving reinforcement learning problems:

1. Searching in the space of behaviours in order to find the most optimised performance by the agent in the given environment. This approach has been adapted by genetic algorithms as well as some of the novel search techniques.
2. Using statistical techniques and dynamic programming to estimate the outcome of taking actions in states of the world.

# Diagram, engineering drawing Description automatically generated

Figure The standard reinforcement-learning model

The aim of this report is to deliver the mechanism of the process to achieve a reinforcement-learning model. Starting from Markov Decision Processes to Deep Reinforcement Learning.

# Markov decision process(MDP)

MDP is the process of selecting an action for the agent from the state they are in and then propagating to a new state where they will get a reward for taking the action. This happens simultaneously and results in a trajectory which demonstrates a sequence of states, actions, and rewards. Throughout the process of gaining experiences, the main aim of the agent is to maximise both the immediate and cumulative reward.

MDP consists of sets of states, actions, and rewards, where after the agent takes an action in a given state it receives a reward for taking the action.

Here is the current state, is the action taken and is the reward attained from the state-action pairs.

## transition probability

Transition probability is calculated to identify the possible values of the preceding state-action pairs from the previous state-action taken.

## expected return and discounted return

In order to calculate the expected return to get the maximum cumulative reward the concept of expected return is introduced:

Expected return is necessary to allow the agent to take decisions to gain rewards. Conversely, if the agent is in a continuous task, then this expected return might be for an infinite time limit, this brings the topic of discounted return:

Discounted return changes the aim of the agent from getting the most expected return to achieving the maximum expected discounted return, here the discount rate ,is kept between 0 and 1, which determines the present value of future rewards. This results in the agent concentrating on more immediate rewards as the future rewards will be heavily discounted.

## policy and value functions

Policy function allows to map a state and get the probability of the agent taking the possible action from that state in that time.

Value functions allows to identify the performance of the agent in the given state. The performance of the agent depends on the policy it is following, this allows to calculate the value function for that respective policy.

**STATE-VALUE FUNCTION:**

**Q-FUNCTION:**

# optimization:

Optimization allows to identify the policy which provides more rewards to the agents’ compared to other policies.

**Bellman Optimality Equation:**

Bellman’s’ optimality equation allows to identify the most optimum q-value for given state-action pairs, which is calculated by taking the expected return and the maximum expected discounted return for the next state-action pairs.

Q-learning allows to store the q-values in a q-table. Q-table store all the q-values calculated in the given state by taking the action within each episode. Nevertheless, the first step taken by the agent is taken by introducing a trade-off between exploration and exploitation, as the q-value at the start of the episode is 0, therefore the agent will not be able to discover the action to take in the given state. This will be further discussed in this report by experimenting with the concepts of RL in an environment.

## EXPERIMENT AND DISCUSSION:

The experiment conducted for researching Reinforcement Learning was done using an environment provided by OpenAI called CartPole. In this environment a pole is attached to the cart that is swinging left and right simultaneously. The goal is to teach the agent to balance in the cartpole using Deep Reinforcement Learning.

The environment of the CartPole was set up with all the methods necessary in order to conduct and observe the agent’s performance; this includes getting the height and width of the cartPole image along with the number of actions available for the agent to take etc. In this case, the agent only had two actions which will be elaborated on later in this report.

Chart, box and whisker chart

Description automatically generated

Figure Environement of cartpole

For this experiment the number of episodes taken for the agent to discover the environment was kept to 1000; then the decision of exploring or exploiting the environment is determined by epsilon greedy strategy.

Epsilon greedy strategy allows to gain balance between exploration and exploitation. In this strategy the exploration rate, epsilon, is determined and a decay rate is set up for this exploration rate to reduce the likelihood for exploring the environment instead of exploiting it. As shown in the snippet Figure 3 exploration and exploitation, the epsilon greedy strategy is evaluated by passing the starting exploration rate, which is 1, the ending rate 0.01 and the decay 0.001, respectively; later these values are passed into the get\_exploration\_rate function which in return gives the current exploration rate for the given state.

A random number is generated between 1 and 0 to determine whether the agent should explore or exploit the environment, and this is compared with the exploration rate in each episode to take the next action accordingly. For the first episode, the exploration rate is kept which ensures that the agent will explore the environment and update the q-value in the q-table. In each episode the highest Q-value in the Q-table is determined.



Figure exploration and exploitation

﻿

In order to converge the q-value to optimal q-value, loss between the q-value and the optimal q-value is compared each time for the given state-action pair:

Later, updating the q-value for the same action-pair to reduce this loss and increase the learning rate. The new Q-value is calculated by:

Here is the learning rate, to establish optimal policy, optimal q-function is needed to be discovered by the agent. Notwithstanding, Q-learning has limitations with value iteration as the q-table updates simultaneously during the iteration, which ultimately increases the state space and the time in order to traverse those states, finally updating the Q-values. Therefore, to get the optimal Q-function a function approximation method is used. In this case, the function approximation method was Artificial Neutral Networks.

Deep Q-Learning allows to estimate the q-values by using deep neural networks for each state-action pair. The neural network is used to get the predicted q-value from the state passed into the network. Furthermore, it remembers the optimal q-value for each given state and the loss from the network is evaluated by calculating the difference between the q-values that is being outputted by the network and the target q-values as mentioned in the Bellman equation earlier in this report. This was done in order to minimise the loss.

Consecutively, Stochastic Gradient Descent (SGD) and backpropagation is used to get the weights within the hidden layers of the neural networks. This is done repeatedly until the loss is minimised for each given state. As mentioned earlier, Bellman equation was used to calculate the q-values and update the q-table to retrieve the optimal q-value. DQN uses this Bellman equation to predict the q-values and then find the optimal q-values. For this project, the output was the action taken by the agent, in this case, it is either right or left, and two hidden layers were used which were fully connected layers. In the first layer, 24 outputs were passed to the second layer where 32 outputs were being passed, ultimately giving the final two outputs. Later the forward function is implemented to estimate the next action taken by the agent by the given state as shown in Figure 4 DQN values. For this project, pyTorch is used to train the DQN. The images of the cart and pole is passed into the DQN, later a DQN object is implemented by using the height and weight of the image. The layers are defined as fc1, and fc2 respectively,



Figure DQN values

The agent experiences at each episode are stored in replay memory, this stores the state of the environment, action taken in that state and the reward gained by taking this state-action pair along with the upcoming state. This is done to prevent the model being overfitted. To illustrate, storing the replay allows the network to take random samples from the experience and break the correlation between consecutive samples, leading to inefficient learning. Figure 5 Replay memory, showcases how the replay memory is implemented in the project, the capacity of the replay memory is the value that is passed during the episode; here the experience of the agent is stored. Later, a pre-processed random state from the replay memory is passed into the policy network as input, this allows to identify the optimal q-function by estimating the optimal policy. In the policy network, the input state is forward propagated through the network, giving the predicted q-value for each possible action by the given input state. The loss is then evaluated, in order to find the max q-value, by passing the next state. From this policy network the max term is identified, which again enables to calculate the loss between q-value estimated by the policy network and the target q-value for the same state-action pair. Gradient descent is used to update the weights of the network. Yet, there are drawbacks in doing a second pass to the policy network. As the same policy network is used, it establishes instability in the network; as the weights are updated the q-values along with the target q-values will also update, therefore as the q-values will be updated to have a closer value to the target q-values, the target q-values will get updated which will increment the distance between the values. Hence, a separate target network is used, which is a clone of the policy network where both networks have the same weight and get updated at the same time in each episode.



Figure Replay memory

The moving average is calculated by transforming the values in pyTorch tensor, then comparing the length of the values with the period. This is done as the moving average cannot be calculated if the dataset is not as large as the period it is calculated for.

# 

# results

# Chart, line chart Description automatically generated

Figure Results

Figure 6 Results showcases the performance of the agent in 1000 episodes. The actual q-values are plotted as blue and the 100-period moving average across these values is orange. From the graph, it can be demonstrated that the 100-period moving average was 0 from the starting episode till the 100th episode, where in the 100th the first moving average was determined. Between 100 and 200 moving average, the second average of 100 values is received. The growing reward indicates the agent was learning the environment over time.

# conclusion:

Reinforcement learning opened a new horizon of Artificial Intelligence. Starting from using Markov Decision Processes to identify the relationship between the agent and the environment by taking actions in the given state. Exploring and exploiting the environment using the Epsilon greedy method and later implementing these concepts using neural network where the q-values were estimated by passing the state through policy network. Later these estimated states are then used to retrieve the max term by the target network, where it has the same weight as the policy net. All these processes were undertaken in order to minimise the loss and get the optimum q-values from the optimum-policy net.

# Bibliography

1. Hamet, P. and Tremblay, J., 2017. Artificial intelligence in medicine. *Metabolism*, *69*, pp.S36-S40.
2. Kaelbling, L.P., Littman, M.L. and Moore, A.W., 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, pp.237-285.