Deep reinforcement learning for AI chess player

Thomas Davies

[a]*The University of Sheffield,*

1. Introduction

This report will explore the use of deep reinforcement learning to train an AI chess player to deliver a check-mate. The game of chess will be reduced to a 4x4 board, containing 3 pieces in total - Player 1's King and Queen, and Player 2's King. Deep reinforcement learning will be used to train player 1's King and Queen to deliver a check-mate on Player 2's King, which will make random, legal moves after a move from Player 1.

Two different deep reinforcement learning algorithms will be implemented and an in-depth analysis of the results will be provided, along with the results of varying hyperparameters within the models.

2. Deep reinforcement learning

Deep reinforcement learning refers to the combination of deep neural networks and Temporal Difference learning (TD) algorithms being used to train an agent to make optimal decisions based on it's environment.

2.1. Use of Deep Neural Network

In deep reinforcement learning, a deep neural network is used as a function approximator for the Q values. The deep neural network will have a structure specific to the environment it is being used for, but the number of inputs nodes will always be equal to the number of possible states, and the number of output nodes will be equal to the number of possible actions.

To approximate a Q value for a state-action pair, a forward pass of the network is executed after receiving a state as an input. The output of this forward pass is all of the estimated Q values.

2.2. TD learning

Temporal Difference (TD) learning is used to update the Q value for a state-action pair, based on the difference between the estimated Q value and actual Q value. TD learning assumes the system is Markovian, meaning the future state is only decided from the current state and unaffected by previous states.

2.2.1. The Bellman Equation

In TD learning, the Bellman equation specifically is used to update the Q-value estimates. It uses the sum of the expected future rewards from the current state to update the Q value.

Equation 1 shows an application of the Bellman Equation to calculate the temporal difference between actual Q values and estimated Q values in reinforcement learning. η shown in the equation is the learning rate, which directly determines the magnitude to which the Q values are updated.

$$\Delta Q_n(s,a) = \eta \left(r - Q_n(s,a) + \gamma \max_{a'} Q_n(s',a') \right) \quad (1)$$

More generally, Equation 2 shows how the Bellman equation is used to update the weights between each layer in the back-propagation of the neural network.

$$\Delta w_{ij} = \eta \left(R + \gamma Q(s',i') - Q(s,i) \right) H(h(n)) x(n-1) \quad (2)$$

where H is the Heaviside function.

2.3. Exploitation vs exploration and the Epsilon Greedy policy

In reinforcement learning, the agent must take an action to interact with the environment. To select an action, the agent must either take the action it currently predicts to have the highest reward (exploitation) or explore new actions it has not explored yet from the current state (exploration). This proposes a trade-off between exploitation and exploration that must be considered when implementing reinforcement learning algorithms.

An epsilon-greedy policy is used to determine when the agent exploits or explores. With probability (1 - ϵ) (epsilon), the agent will choose the next action with the highest Q value - exploitation. With probability ϵ the agent will choose a random action - exploration. A common method to optimise the trade-off between exploitation and exploration is to decay the value of epsilon throughout training - initially the agent will explore more random actions and as training progress and more knowledge is gained, the agent will exploit the learned information by selecting actions with higher estimated values.

2.4. Deep Q learning and SARSA

Q-learning and SARSA are both TD learning algorithms, and when a neural network is used as an approximator for the Q values become deep reinforcement learning algorithms. The difference between the two is subtle, but will produce noticeably different results.

2.4.1. Deep Q learning

Deep Q learning is an off-policy method of deep reinforcement learning that works in the following way. The training of the Deep Q agent is done over a specified number of episodes - at the start of each episode the state and environment are reset. For each episode:

1. A random state, S is initialised
2. An action, A is chosen. The action is chosen by obtaining Q values for the current state using forward propagation of the neural network. The epsilon-greedy policy then either chooses the action with the highest Q value, or a random action.
3. The action, A, is executed and the reward is observed. The state has also updated to S'.
   If S' is terminal, the current episode is ended and the next episode begins. Otherwise, the following steps are executed.
4. An action A' is now selected using a greedy policy (as opposed to epsilon greedy). This again means using forward propagation of the neural network to calculate Q values , however only choosing the action with the highest Q value this time.
5. This is where the temporal difference is calculated. The temporal difference is then used to update the weights and biases of the neural network, using backpropagation.
6. The state S is set to S' and steps 2-6 are repeated until a terminal state has been reached.

These stages are repeated for the specified number of training episodes. After sufficient training, and assuming suitable hyperparameters are provided, the neural network will be updated so that after forward propagation it will output Q values that that accurately reflect the expected cumulative rewards for each state-action pair.

2.4.2. SARSA

SARSA is an on-policy reinforcement learning algorithm that can be applied to deep reinforcement learning. SARSA is similar to Q learning, however there is a subtle yet import distinction which can lead to varied results when comparing the performance of both algorithms.

The steps of SARSA in deep reinforcement learning are largely similar to those described of Deep Q learning, however the distinction is in step 4. When choosing action A' to be used in the temporal difference calculation, the policy, epsilon-greedy, is used again.

3. Deep Q Learning implementation

To implement Deep Q learning considering the environment described in the introduction, the neural network must have a shape of 32 input nodes and 58 output nodes. This is due to the 32 possible states of the environment, and 58 potential actions that may be made. It is important to consider that due to the rules of chess, not all 58 actions can be legally made, so the environment must also provide a function to determine which actions are legally allowed.

In this implementation of Deep Q learning, a neural network with the 32 input nodes, 58 output nodes and a hidden layer of 200 nodes was used. The RELU activation function was used in the forward and backward propagation of the neural network.

The following shows the results from the implementation of the Deep Q learning algorithm. The exponential moving average of the average reward and the average number of steps per episode, with following starting hyperparameters:

Table 1: Hyperparameters

Hyperparameter	Value
Discount factor (γ)	0.85
Starting epsilon (ε)	0.2
Epsilon decay rate (β)	0.00005
Learning rate	0.0035

Figure 1 shows the exponential moving average of the average reward per episode. As shown, after approximately 40,000 training episodes the average reward per episode begins to converge, meaning the agent has learned a relatively stable policy and consistently achieves a similar average reward per episode.

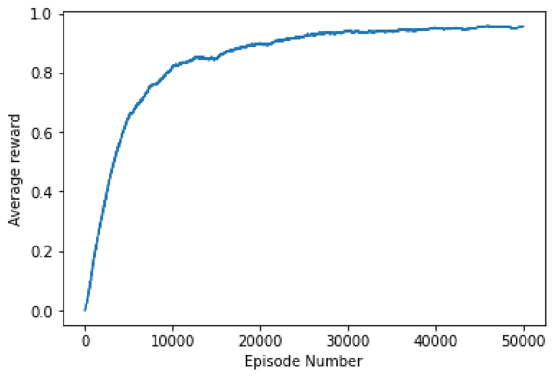

Figure 1: Average reward per episode

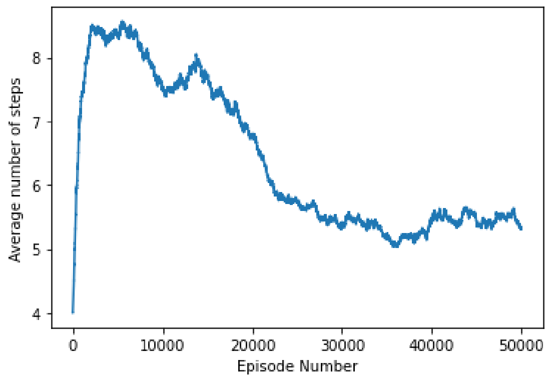

Figure 2: Average number of steps per episode

Figure 2 shows the average number of moves converging after approximately 40,000 moves too.

4. Effects of hyperparameters

The following shows the results of varying discount factor and and epsilon decay rate. In each training phase, the hyper-parameters shown in Table 1 were used, unless they were intentionally varied.

4.1. Discount Factor (γ):

The discount factor, γ, determines the importance the agent places on future rewards when updating the Q values using the Bellman Equation. As shown in equation 1, the discount factor multiplies the estimated future rewards. It can be observed from equation 1 that a higher discount factor increases importance of estimated future rewards when updating the Q values.

Figure 3 shows the effect of the discount factor whilst training the agent. It shows that in this environment, a discount factor lower than 0.85 places too much importance on immediate rewards, resulting in lower average reward per episode.

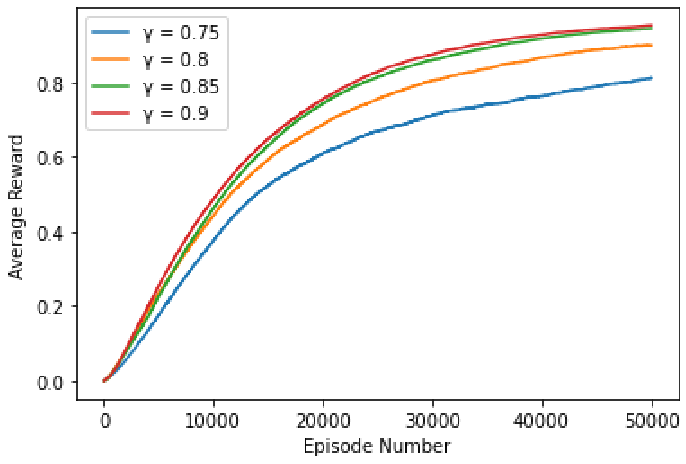

Figure 3: Varying γ

Table 2 also shows that a discount factor too low effects the agents efficiency in delivering a checkmate. From the results, it could be interpolated that the optimal value for the discount factor would lie somewhere between 0.8 and 0.9.

Table 2: Varying γ

γ	Average number of moves per episode
0.75	9.46522
0.8	8.30584
0.85	6.06796
0.9	6.65420

4.2. Epsilon decay rate (β):

The epsilon decay rate, β, effects the agent's tendency to exploit/explore over time. Initially, when the epsilon value is at it's highest, the agent has a higher tendency to explore new actions, and not exploit action with the highest associated reward. As the value of epsilon decreases, the agent tendency to exploit increases.

The results show that in this environment a larger value for β optimal. This means that the sooner the agent begins to favour exploiting over exploring, the sooner it trains to successfully deliver checkmates. In figure 4 and table 3 it can be interpolated that the optimal value for β in this environment lies somewher around 0.00005.

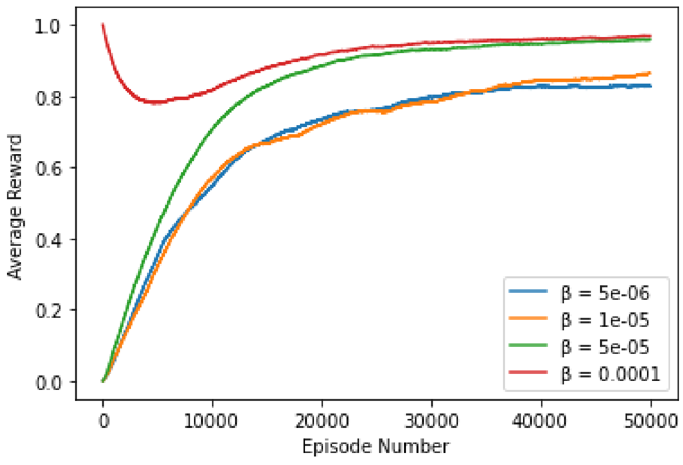

Figure 4: Varying β

Table 3: Varying β

β	Average number of moves per episode
0.000005	7.24004
0.00001	7.75486
0.00005	5.28384
0.0001	6.10346

5. SARSA implementation

A SARSA implementation was also completed. This section will discuss the expected results and present comparisons from the results of training Deep Q learning and SARSA models.

5.1. Expected results

It is expected that SARSA will take longer to converge than Deep Q learning. This is due to SARSA being on-policy when choosing an action to update it's Q values with. As previously mentioned, SARSA will choose it's update action with an epsilon-greedy policy, whereas Deep Q learning will use a greedy policy. This means that sometimes SARSA will update it's Q values after sometimes not selecting the optimal action.

5.2. Results

As expected, SARSA takes slightly more training episodes to converge than Deep Q learning. When evaluating the performance of both agents after 50,00 training episodes, the last 2000 episodes were used to determine an average reward per episode, as by this stage the model has converged. Interestingly, SARSA had a higher final average reward of 0.946, compared to 0.9175 of Deep Q learning.

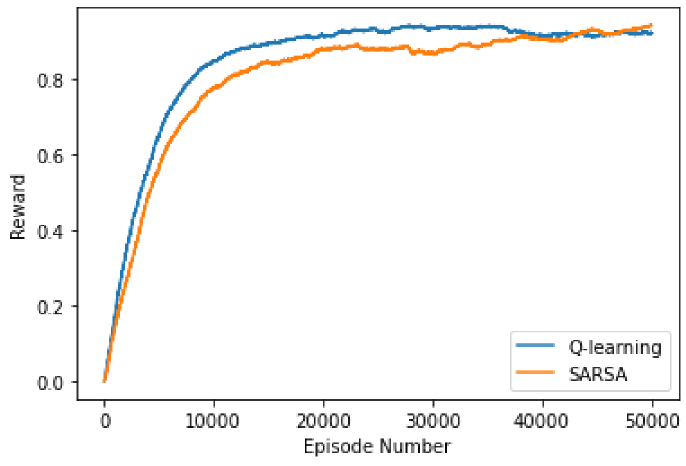

Figure 5: SARSA and Deep Q learning

6. Reproducing results

To reproduce the results shown in the report, please clone the repistory found at https://github.com/thomasdavies2000/adaptive/tree/main.

Assignment.ipynb comments will indicate which cells to execute. Some of the numerical values may vary slightly, but the trends shown in the plots will be the same.