University of Sheffield

Collective Report B

Jake Quinn Sturgeon

Lecturer: Prof. Eleni Vasilaki

Module Code: COM3240

A report submitted in partial fulfilment of the requirements
for the degree of Computer Science with a Year in Industry MComp

in the

Department of Computer Science

May 13, 2019

Introduction

The report explores two forms of Reinforcement Learning (RL): Q-Learning (QL) and State–action–reward–state–action (SARSA) to play a simplified version of Chess. This version consists of a 4 by 4 grid with only three pieces (two kings and a Queen). The goal of player one is to checkmate player two, and player two's goal is to move so it is not threatened by player one.

Question 1

This section will describe the algorithms Q-learning and SARSA and will describe the advantages and disadvantages between them.

Firstly, this report uses a Deep Neural Network (DNN) to model the state and action, where the input layer represents all the current states and the output layer are all the possible actions. The weights between neurons represent the State/action pair or Q-value, which is updated by a form of Hebbian Learning rule (Eq. 1). This form of Deep RL allows the use of back propagation, a supervised algorithm, where the target values are the estimated Q-values of the system.

$$\Delta Q(s,a) = \eta(r - Q(s,a)) \tag{1}$$

Q-Learning

Q-Learning (QL) is an example of a model-free RL algorithm, with the task of developing its own policy (can be called off-policy) i.e. the agent will develop its own strategy for a given situation. QL will find the optimal policy which maximises its total expected reward. This is calculated by adding its current state reward to its maximum rewards from future states. This allows the possibility of future rewards to influence the action to be taken.

$$\Delta Q(s_t,a_t) = \eta[r_t - (Q(s_t,a_t) - \gamma \max_a(s_{t+1},a))] \tag{2}$$

SARSA

SARSA, which stands for State–action–reward –state–action, is an example of an on-policy RL algorithm. This is due to the fact that SARSA uses it's control policy to update its Q-values, unlike QL, which only assumes the optimal policy. This slight alteration allows the agent to explicitly know its future reward, rather than expecting the optimal choice to result in the optimal outcome.

$$\Delta Q(s_t,a_t) = \eta[r_t - (Q(s_t,a_t) - \gamma(s_{t+1},a_{t+1}))] \tag{3}$$

In conclusion, QL has the advantage of finding the the optimal policy where as SARSA only learns the near-optimal policy. However, this may result QL to get stuck in a local minima but, SARSA, with an ϵ-greedy policy, can avoid this by 'jumping' from a local minima and explore other actions.

Question 2

This section will describe the implementation of QL and will visually explain the performance the system.

The ANN Neural activations where calculated using the snippet 1 and back propagation is calculated via snippet 2.

```
# Neural activation: input layer -> hidden layer
act1 = np.dot(W1, x) + bias_W1
# Apply the RELU function
out1 = np.maximum(act1, 0)

# Neural activation: hidden layer -> output layer
act2 = np.dot(W2, out1) + bias_W2
# Apply the RELU function
Q = np.maximum(act2, 0)

return Q, out1
```

Listing 1: Neural activations

```
out2delta = (t - Q[a_agent]) * np.heaviside(Q, 0)
W2[a_agent] += (eta * np.outer(out2delta, out1))[a_agent]
bias_W2[a_agent] += (eta * out2delta)[a_agent]

# Backpropagation: hidden layer -> input layer
out1delta = np.dot(out2delta, W2).dot(np.heaviside(out1, 0))
W1 += eta * np.outer(out1delta, x)
bias_W1 += eta * out1delta
```

Listing 2: Backprop example

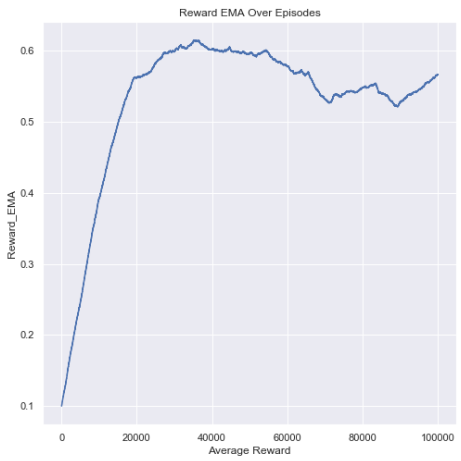


Figure 1: EMA of the Reward per Game

The figure 1 shows that the reward of QL initially increases to 0.6 but then begins to decline. This could be explained that the "optimal" policy that QL adapts is a high-risk policy which leads to many draws. When compared to figure 2, which shows the average number of moves, shows that during the same period that the reward decreases the number of moves also decreases. Suggesting that the agent is pursuing games that require the least amount of moves but runs the risk of drawing.

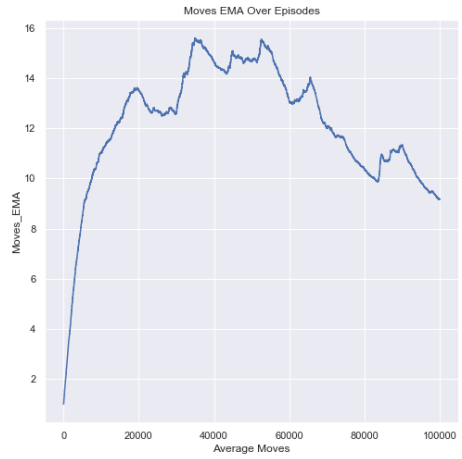
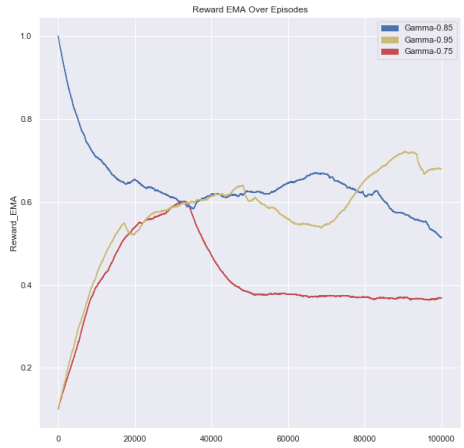
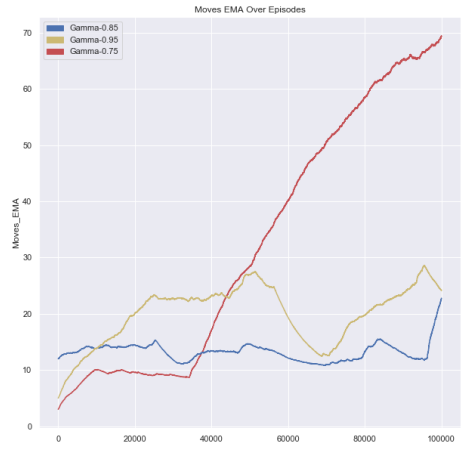


Figure 2: EMA of the Reward per Game



(a) EMA of Reward over Number of Episodes



(b) EMA of Moves over Number of Episodes

Figure 3: Gamma Results

Question 3

This section will explore the use of two hyperparameters to understand how they effect the system as a whole.

Gamma

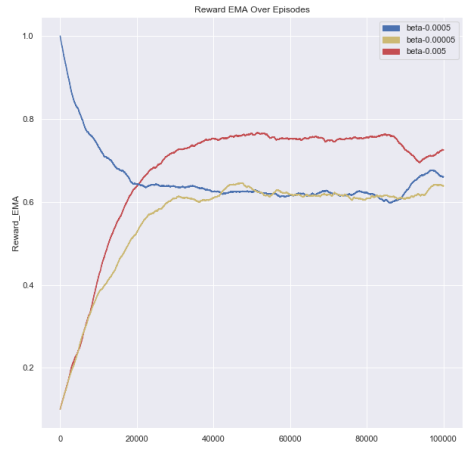
Gamma (γ) is the discount factor. The discount factor affects the importance of future rewards in decision making. A γ value that is close to 0 will make the agent only consider current reward, essentially making the agent very short-sighted. On the other hand, a γ value that is close to 1 will result in an agent that will look for high future reward. Figure 3a shows that the higher the γ value, the better the system performs. This shows that the system benefits from a long-term outlook on the game. However, when $\gamma = 0.75$, the system results in a local minimum that rarely wins. This can be seen by the sharp increase of moves in figure 3b

Beta

Beta (β) is the discount factor for the ϵ-Greedy policy. A larger β means that the system will only explore early during learning and a lower β means that exploration will occur throughout learning. It is shown in figure 4a that a higher beta benefits the system. This can be explained as the system doesn't have to explore any further after a certain point as it has already learnt the best policy to win. However, this could lead to a situation where the agent gets stuck in a local minimum. Secondly, figure 4b shows that when $\beta = 0.005$ the QL agent reaches the optimal amount of moves to get the optimal reward. This can be seen by the almost straight line. As the system now remains in the minima and does not leave.

Question 4

Figures 5a, 5a, and 5c compare the use of QL (used in the previous experiments) against SARSA. Firstly, figure 5a shows that SARSA begins to outperform QL as SARSA's average reward is higher than QL. However, this sharply falls after around 37000 episodes. This is due to a gradient exposition that led to unstable gradient descent and so SARSA was unable to learn. This explosion can be seen in figure 5c. Around episode 50000 a small blip can be seen in the SARSA line. Figure a5c shows the average error (i.e. the rate of learning) of the system. This is possibly due to the propagation of unstable weight changes that occurred in previous episodes. Overall, SARSA has a smaller average loss which suggests SARSA has performed better at finding its target action. This is due to the fact that SARSA is on-policy, and in this case, uses $\epsilon-$greedy to explicitly explore future rewards.



(a) EMA of Reward over Number of Episodes

(b) EMA of Moves over Number of Episodes

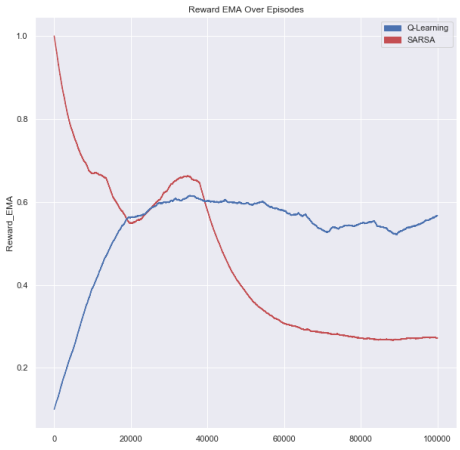
Figure 4: Beta Results

Question 5

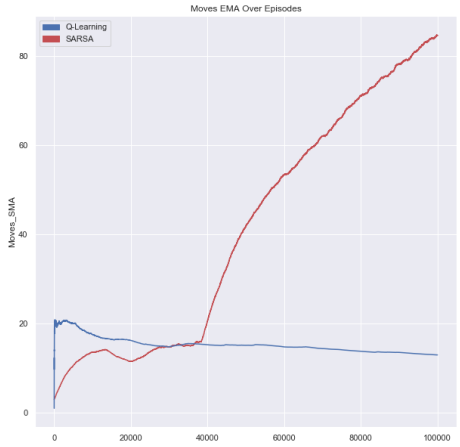
Exploding gradients can result from accumulating error gradients together which generate very large error values. These error values can then result in large updates for the network. In the worse cases, these errors can become extremely large and lead to an "explosion", which can lead to overflow errors and 'Nan' during run time. This makes the network unstable, thus precautions must be made to prevent this issue. An example of an exploding gradient can be seen in the SARSA line around 50000 episodes in the figure 5c. A possible fix for such an issue is called RMSprop. RMSprop (also known as Root Mean Square Propagation) allows for the learning rate to be adapted during runtime for each weight, via a running average of mean squares (Eq. 4) ,where γ is the forgetting factor, and each weight is updated by Eq. 5.

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \quad (4)$$

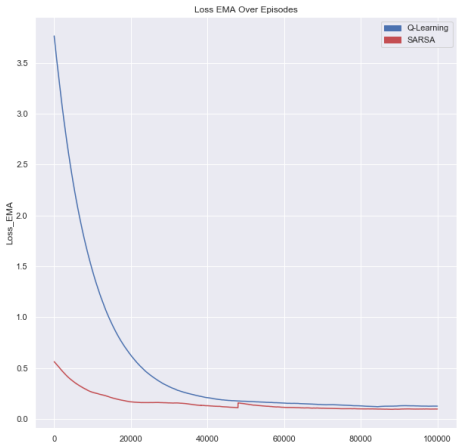
$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w) \quad (5)$$



(a) Comparison of Q-Learning and SARSA Average Reward



(b) Comparison of Q-Learning and SARSA Average Moves



(c) Comparison of Q-Learning and SARSA Average Loss

Figure 5: SARSA Results