67750 | Advanced Practical Course in Machine Learning | Ex 4

Guy Lutsker 207029448

December 27, 2020

1 Implementation of Reinforcement Learning

1.1 Q-Learning

To explore this algorithm I decided to construct a linear neural network - it is able to see the 9 cells around the snakes head (including the head) and has 2 linear layers with the same width as the input, with Relu activation. using this network I am able to evaluate algorithm.

A note of graphs in this submission - I ran a lot of analysis and to fit all the graphs I had to make them smaller. I think that if you zoom into them in the resulting PDF you can still see everything you need, but in case it doesn't work I added to each graph's description a link to a web full resolution representation.

Also, this report is 5 pages long, im sorry for that, but I got your approvement for this :)

1.1.1 γ

The first thing I wanted to evaluate is the effect of the γ hyper-parameter has on the reward. γ is the discount factor for future reward, the higher γ is, the more we take the future reward in account when calculating our optimal next action. In order to test this effect I wanted to run different γ values on the Q leaning framework. I ran $\gamma \in [0.1, ..., 0.8]$ and for robustness sake I wanted to run each value for 4 iteration to get a better picture of the effect (Why 4? because its a nice even number, and mostly because it takes ALOT of time to run so many models and I had to have a bound). In addition I decided to run all the models for 200 thousand steps for a good measure of how they did. Results are shown in Figure.1 :

Figure 1: Rewards in Q-Learning with a simple linear neural network on different γ values (full res: https://bit.ly/3ru76Pf)

As can be seen. the robustness argument was important since the run can differ by a certain value and we should take into account the variance. After thinking about for a while I decided that a γ factor of 0.6 is optimal and decided to continue for this value.

1.1.2 ϵ

Another important hyper-parameter is ε, this value accounts for the Exploration/Exploitation ratio, essentially it is the probability of choosing to explore, our world randomly. In order to choose the optimal ε I wanted to run several value if it. Again running for $\varepsilon \in [0.1, ..., 0.9]$ for 4 iteration each, with 100 thousand steps(lower here since we have a base measure for γ that should ensure positive convergence for the rewards, and if we don't converge within 100k steps this means we chose a bad hyper-parameter). Graph of rewards for different ε values:

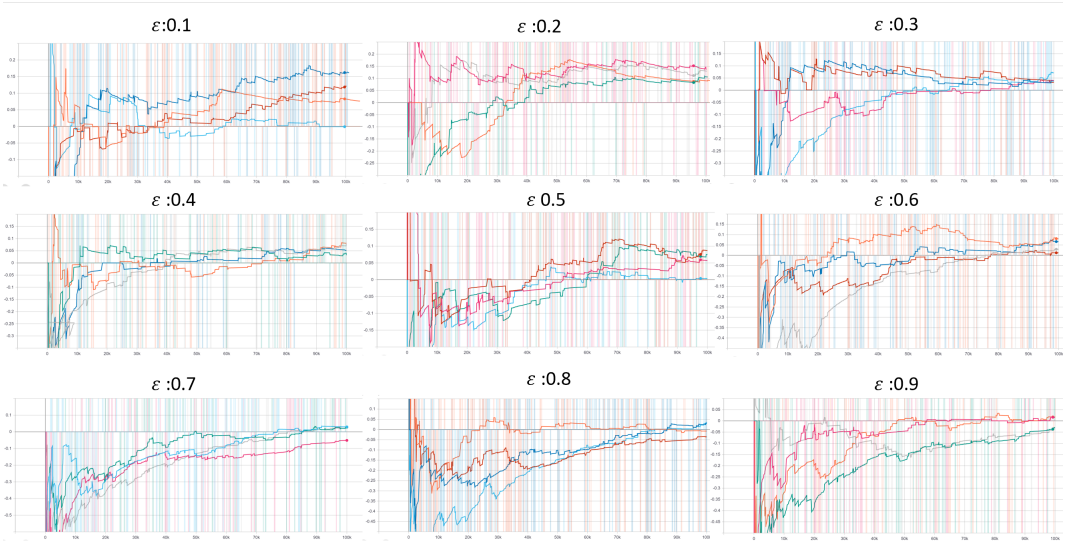

Figure 2: Rewards in Q-Learning with a simple linear neural network on different ε values on $\gamma = 0.6$ (full res: https://bit.ly/38JOOBc)

From these graphs, I decided to continue with $\varepsilon = 0.2$ as it seemed as though it might bring optimal results.

1.1.3 Clipped Gradient

A common problem one might encounter in these kind of leaning algorithms is a problem of "exploding gradients". This term refers to a phenomena in which the gradients we get from the model are too large and they might throw our model out of order. To combat this phenomena we employ a method of "clipping" the gradient - meaning taking $\nabla' = min\{\nabla, bound\}$ where ∇ denoted our returned "real" gradient, and ∇' denoted our clipped gradient. Here I trued to evaluate my model with runs of clipped v.s unclipped gradients, with $bound = 1$. To reach a more educated conclusion, since there were only two options, I decided to up my number of runs per option to 8, and to calculate the mean and std deviation of each option:

Figure 3: Rewards in Q-Learning with a simple linear neural network, $\gamma = 0.6, \varepsilon = 0.2$ on clipped v.s unclipped gradients, with $bound = 1$ (full res: https://bit.ly/2WLIIKH)

As with most of the choices in this exercise, the results are very inconclusive, but the clipped rewards were slightly better, and with lower variance, and so I decided to go with them from now on.

1.1.4 Optimizers & Learning Rate

The choice of optimizers and a good leaning rate is also very important. Here I want to ovulate the Adam and the RMSProp optimizers on different learning rates to choose the optimal option:

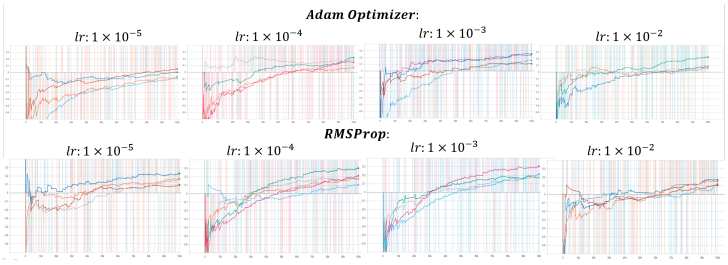

Figure 4: Rewards in Q-Learning with a simple linear neural network,$\gamma = 0.6, \varepsilon = 0.2$, clipped gradient, on Adam and the RMSProp optimizers on different learning rates (full res: https://bit.ly/3pub0WK)

Here I decided to go with RMSProp with $lr = 1 \times 10^{-3}$.

1.1.5 Normalization

Normalization of the reward can contribute to large changes to the gradients of the model, and so are another excellent candidate to try and optimize. Results for this method for Q-Learning is at section 1.2.5(because of space constrains).

1.2 Vanilla REINFORCE

This is considered a more advanced algorithm and we learned its pseudo code in the recitation. To explore this algorithm I decided to construct both a linear neural network and a convolutional one - as before it is able to see the 9 cells around the snakes head (including the head) and has 2 linear/convolutional layers.

1.2.1 γ

As mentioned before I would like to evaluate different γ values for best performance:

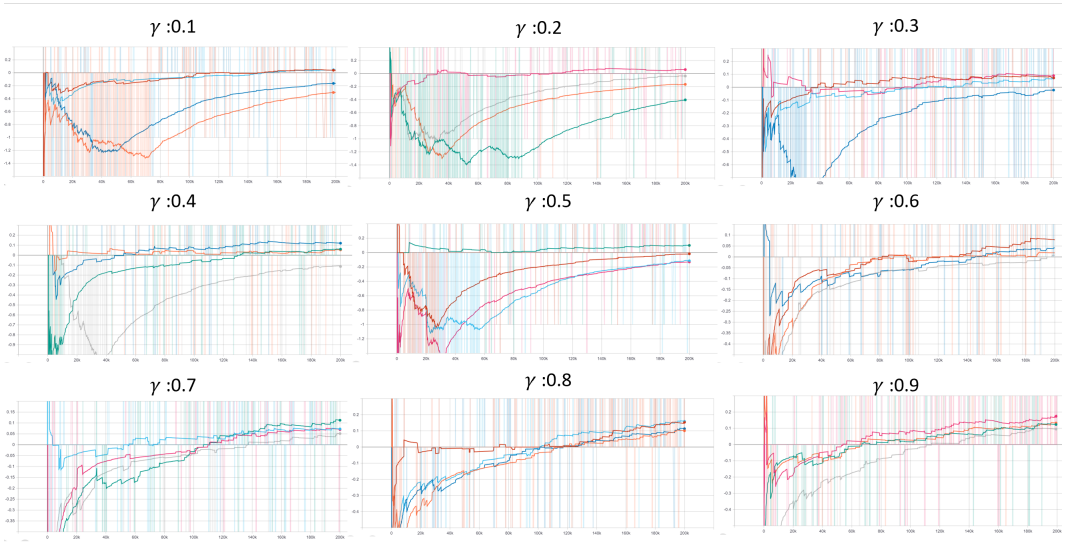

Figure 5: Rewards in Vanilla REINFORCE with a simple linear neural network on different γ values (full res: https://bit.ly/2KvqDhO)

It seems a though best results are obtained by $\gamma = 0.8$.
A graph from the convolution model:

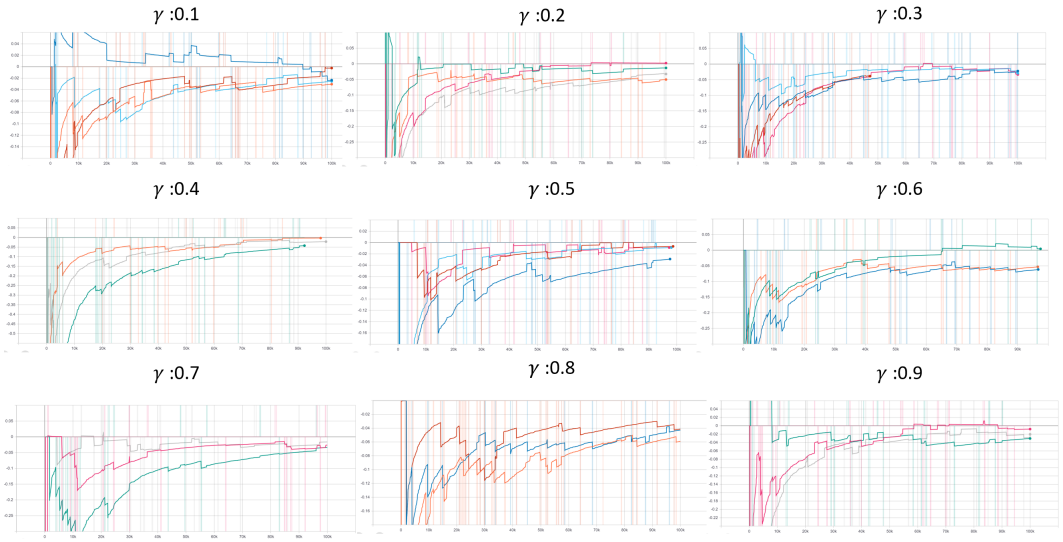

Figure 6: Rewards in Vanilla with a convolutional neural network on different γ values (full res: https://bit.ly/2WNhnYD)

Unfortunately I was not able to make this model converge to a positive reward :(This was very surprising to me since I actually expected this model to do better than a simple linear model for several reasons: 1. for entirely naive reasons I hoped that a more complex model will yield better results. 2. our input is a window into the snakes world and I expected that a convolutional model will be better at analyzing such 2D data. But I was wrong, and so I decided to move on with the linear model.

1.2.2 α

The vanilla framework required $\varepsilon = 0$ but it introduced a new hyper-parameter α. In this model we are able to "inject" entropy to the equation to try and optimize our rewards- α denoted the weight we give to the entropy in the optimization problem. Here I wanted to plot the effect of α on our linear model with $\gamma = 0.8$:

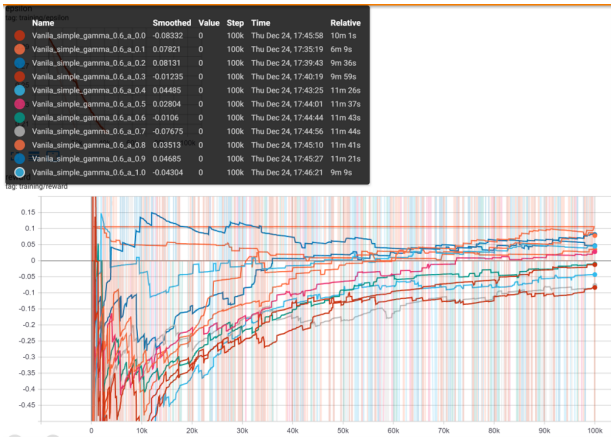

Figure 7: Rewards in Vanilla with a simple linear neural network $\gamma = 0.8$ (not 0.6) with different α values (full res: https://bit.ly/37RNlJO)

Decided to continue with $\alpha = 0.1$ for optimal results.

1.2.3 Clipped Gradient

As mentioned in Q-Learning, I wanted to see how a clipped gradient will effect our results. Again I decided to go with the clipped gradient for the same reasons:

Figure 8: Rewards in Q-Learning with a simple linear neural network $\gamma = 0.8, \alpha = 0.1$ for clipped/unclipped gradients (full res: https://bit.ly/3rGLFe2)

1.2.4 Optimizers & Learning Rate

And once more, deciding for an optimizer and a learning rate. I deiced to go with Adam on $lr = 1 \times 10^{-4}$:

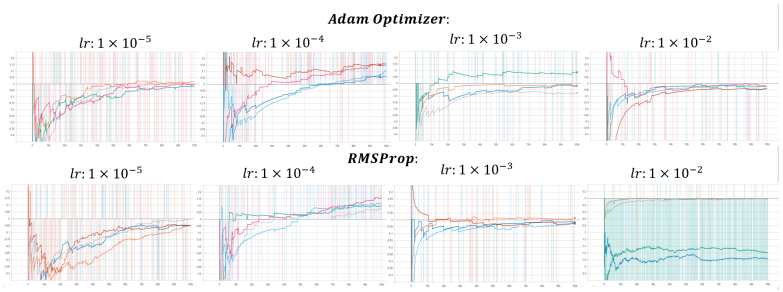

Figure 9: Rewards in Vanilla with a simple linear neural network $\gamma = 0.8, \alpha = 0.1$, clipped gradient different lr (full res: https://bit.ly/3nRir9O)

1.2.5 Normalization

As mentioned in section 1.1.5 normalization of the reward can contribute to large changes to the gradients of the model. Here I review the changes different normalization methods have on the reward where we have 1. No normalization. 2. Dividing by 5. 3. applying $Reward_{new} = \frac{Reward - \mu_{Reward}}{\sigma_{Reward}}$. 4. applying $Reward_{new} = \frac{Reward - min\{Reward\}}{max\{Reward\} - min\{Reward\}}$.

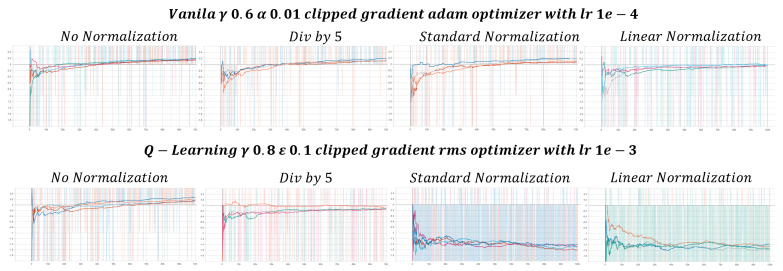

Figure 10: Rewards in Q-Learning/Vanilla on final parameters in each (full res: https://bit.ly/3o55Yzi)

Q-Learning: Interestingly enough normalization here only made thing worst, way worst, since all rewards converged to a negative value for some reason. The logic behind the reward normalization is that it stabilizes training, high magnitude rewards might cause instability because of exploding gradients. and here diving by 5 almost achieves a positive value(although still worst than no normalization at all), all all other normalization methods fail miserably...

Vanilla REINFORCE: Here the situation is more complex, while Linear normalization seemed to fail (converged to 0 on all iterations), and still no normalization seems to beat all other normalization methods, the difference here is not as clear. It looks as though dividing by 5 and standard normalization achieve similar results. Its hard for me to explain exactly what is going on here.