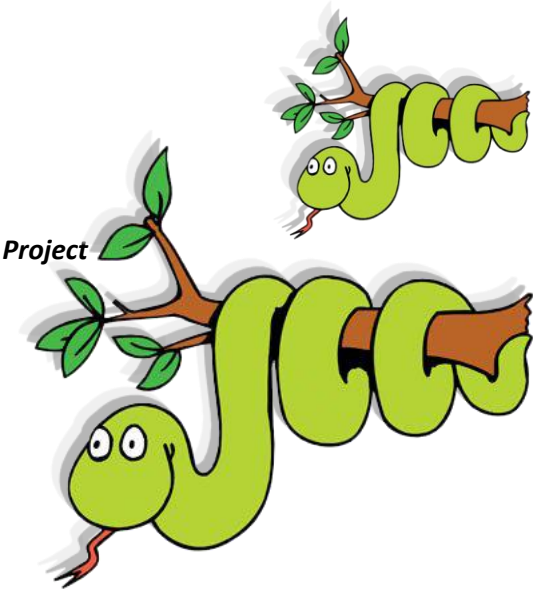
Snake



Advanced Practical Course in Machine Learning: Final Project

Yuval Reif 308490234, Gal Patel 308575836

Welcome to our wonderful project story! We wish you a magical reading journey.

Testing Methods

Before we begin, we would like to have a few words about our testing method since each future explanation is accompanied by plots.

For each parameter test, we ran games against 4 Avoids with epsilons in the range [0, 0.15]. Board size is (40,60). For each test, we:

- Fixed all of the other parameters to a reasonable value
- Iterated over a range of values, and ran 3 games with each value.
    - **Score Plots:** For each game we saved the scores (i.e., the rewards of all of the rounds) then we averaged the scores across 3 games and applied a moving average with n=1000, which in this case is also the score scope.
    - **Loss Plots:** For each game we saved the training loss, then we averaged the losses across 3 games, and applied a moving average with n=100. We did this because the loss was very noisy and was difficult to visually identify its learning curve.

We *always* zeroed the epsilon (exploration probability) when we enter the score scope.

Session length (number of rounds) are different between tests. We sometimes just wanted to see the behavior on a sample of a game.

All visualizations presented here are just a sample of numerous tests we performed.

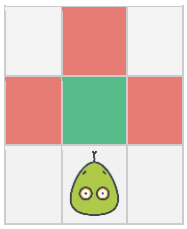
Note: axis x of rounds is not always the same values between loss and score plots, but only because of the difference in when we call learn and act, but please trust us that when we present them side by side they refer to the same test sessions.

Linear Policy

State Representation

We implemented our linear policy state representation to be as simple as possible. We started simple and it worked well.

Given a state and an action, we look 2 steps ahead: what would be in the next location (if we move according to the given action) and what would be another step from there (according to the 3 possible locations). Here is a figure explaining our lookout, if the snake is looking north and is about to take a forward step:

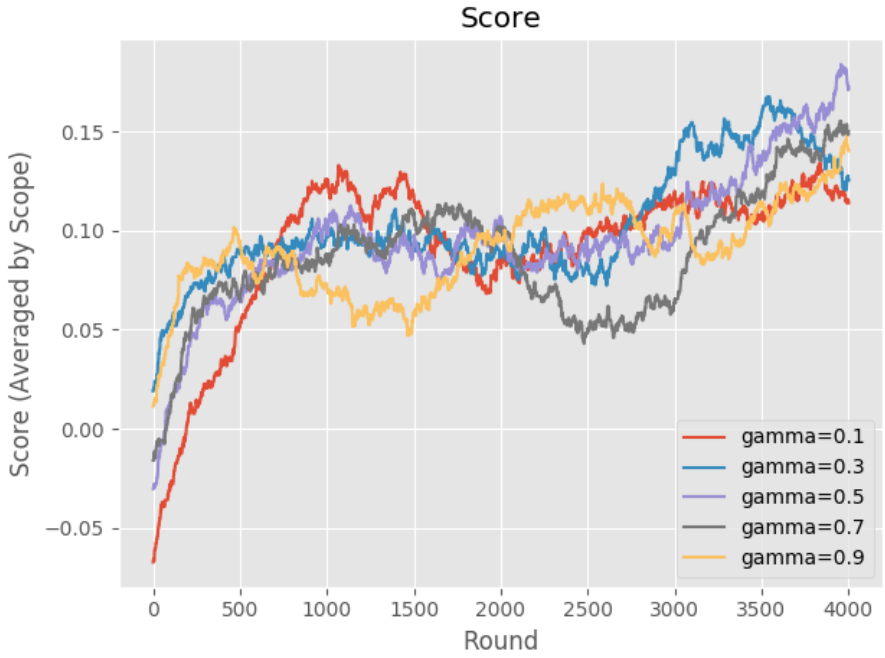


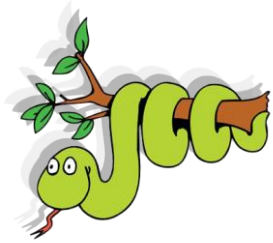
Every one of those 4 board cells is represented in the feature vector as a one-hot-vector of size 11 (the size of the number of different symbols of the board, [-1,9]), so we have a feature vector of size 44 (we added an extra entry which is always 1 to implement bias).

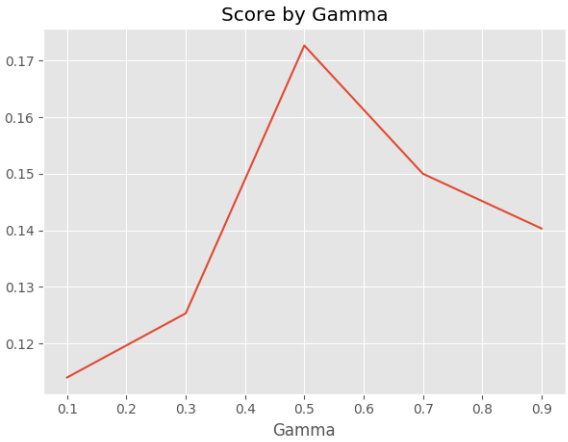
Parameter Selection

According to the following tests' results, we settled on $\gamma = 0.5, lr =, \epsilon = 0.5$ (not the best final epsilon since we only care to overcome the Avoids and 0.5 is positive in earlier stages as well, and we chose the most monotone consistent learning rate).

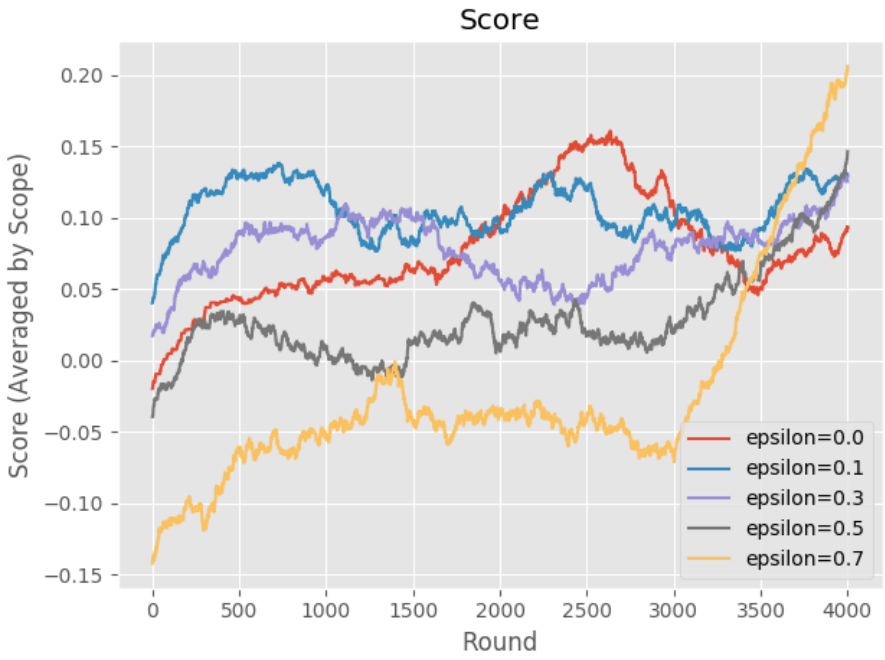
Gamma:

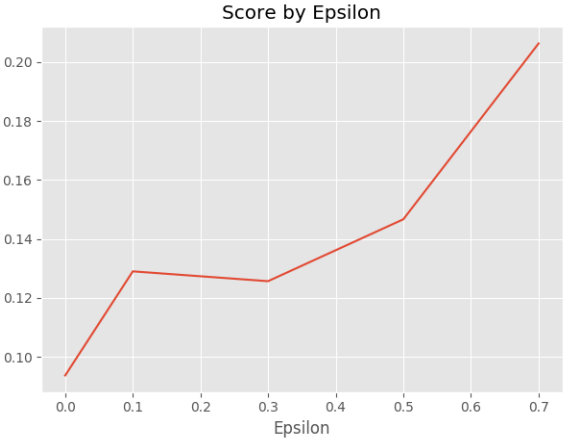


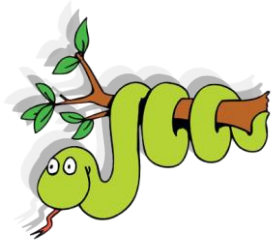



Score by Gamma
0.17
0.16
0.15
0.14
0.13
0.12
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Gamma

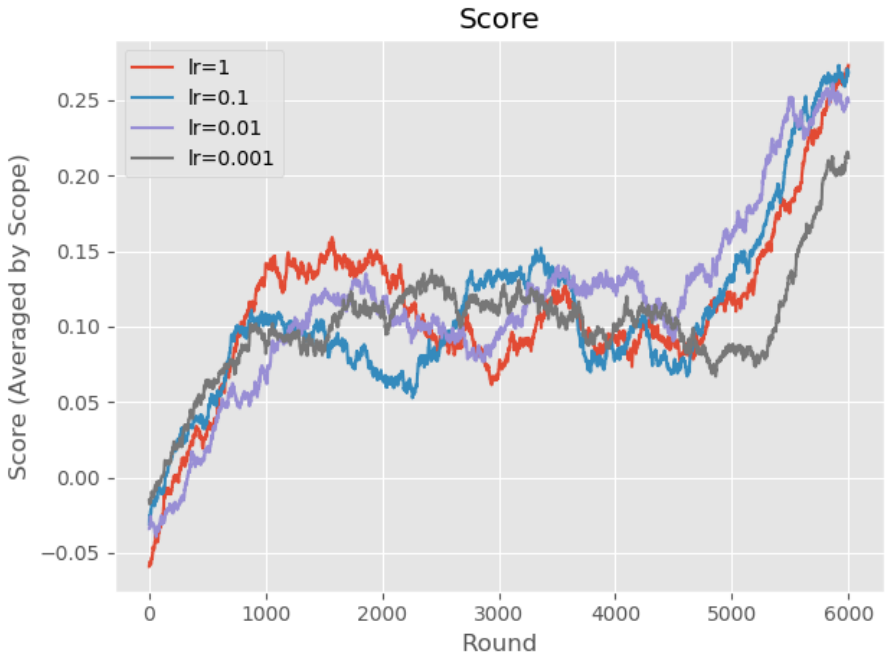
Epsilon:

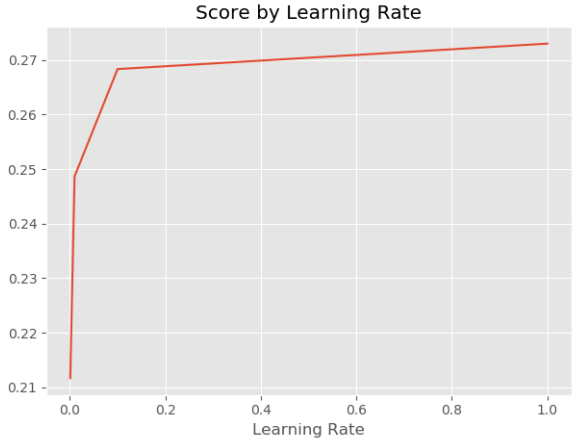

Score
Score (Averaged by Scope)
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
500
1000
1500
2000
2500
3000
3500
4000
Round
epsilon=0.0
epsilon=0.1
epsilon=0.3
epsilon=0.5
epsilon=0.7

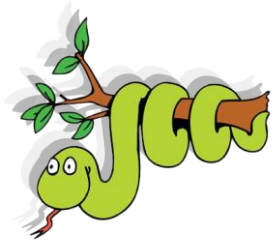

Score by Epsilon
0.20
0.18
0.16
0.14
0.12
0.10
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Epsilon



Learning Rate:







Custom Model

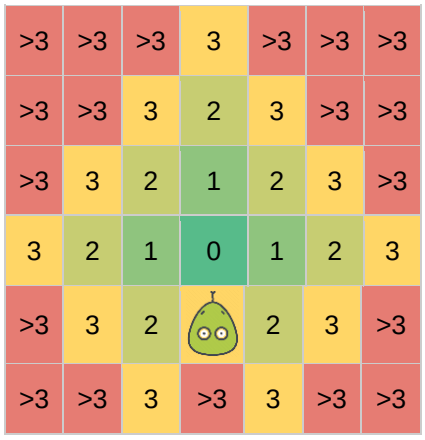
State Representation

In our custom model (model description written in the next section) we expanded our state representation to include 3 different sections of features: distance features, spatial features, snake length. Let's explain each:

Distance Features

Given a distance scope d, we look at what is on the board in x steps from the current location, when $x \in \{0,1,..,d\}$ and current location is the location we were in as if we have taken the given action. We also look at the board in any location bigger then d as a single section. For each such section we have a vector of size 11 (= number of different symbols) representing the percentage each symbol takes from this section of the board, then we concatenate everything together.

An example might explain it better. Let's say we decided on a distance scope $d = 3$. The board is divided to 5 different sections (imagine the snake is looking north and has taken the forward step, so it is stepping on $x = 0$):

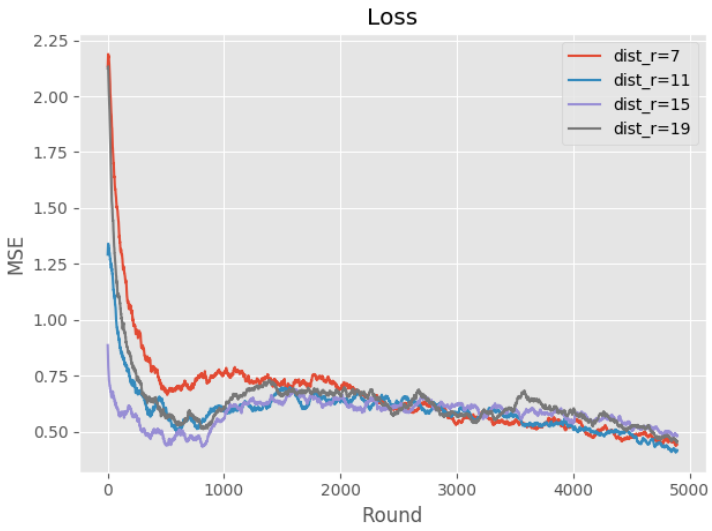


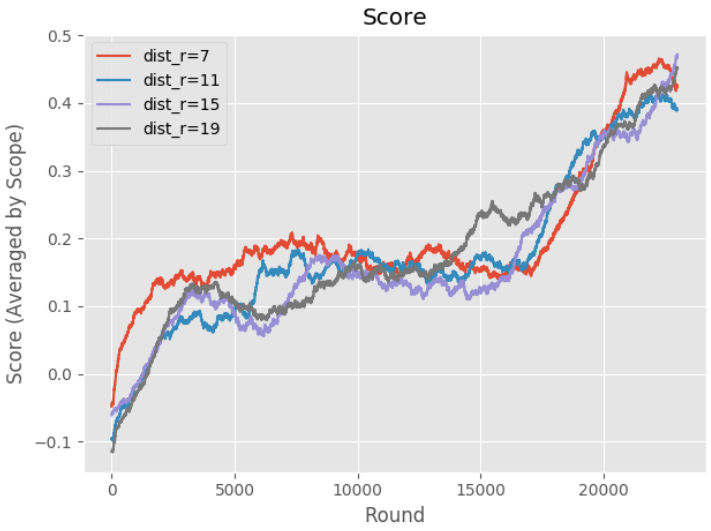
Now, for every colored section of the board, we have a feature vector of size 11. Entry i of this vector is the number of appearances the i'th symbol has on this section, dividing by the size of the section.

The distance features are of size $(d+1) * num\_symbols$. In our example, it's 55.

How did we decide on such features? We were thinking about what is important if we were playing – to understand what would be in the snake's reach, depending on the action it takes. Using this features, we disregard exact spatial organization and look at what the perimeter of future locations would look like (do we have yummy fruits waiting for us in this direction? Do we have enemies there?). Also, this view utilizes symmetry aspects.

Here we can see our comparison between different distance scopes (only additional feature on these tests is the snake length). We decided on a distant scope of 15 (contributing 187 features):

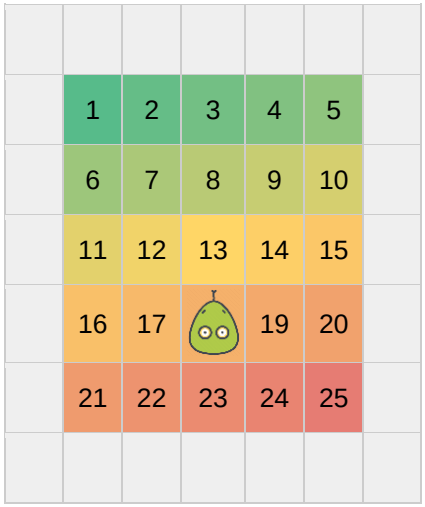




We theorize that the reason that larger distance scopes are not necessarily better is because spots that are very far from us don't matter much because until we'll reach them the board would change greatly (fruits may appear in new places, other snakes would move, etc.).

Spatial Features

We realized our "Distance Features" might not be enough. Let's say that if we take a "right" action, we get 50% fruits at distance 4, but those are blocked by obstacles which are aligned exactly between them and our snake, we can't identify such situation! Therefore, we add more features which are basically looking a small cropped board around the **future** snake's head location (rotated so the snakes looks to the north and about to take a forward action), and for each cell of this cropped section we have a one-hot-vector to indicate what symbol is in that cell. For this feature we defined a radius r representing the size of the cropped section as radius from the snake location (after it takes the given action). For example, for $r = 2$ we have an vector of size eleven for each colored cell in the following board (total feature vector of size $25 * 11 = 275$):



(The snake is looking north and is about to take a forward action).

Of course, this feature is much more expensive then the "distance" feature – it takes much more memory to cover similar areas.

Here we can see our comparison between different spatial radiuses (only additional feature on these tests is the snake length). We decided on a distant scope of 5 (contributing 275 features):

Snake Length (not included in final version since we realized it takes time to compute we need to save when the board size is large).

Another feature is only a scalar: the size of the snake dividing by the size of the board. We think it might be useful since in advance rounds of the game the snake might get very big and should behave differently – moving around more carefully, being aware its own body is an obstacle.

Model Choice

We chose to implement of model of deep q-learning. We opted for deep learning since we believe it's very versatile and powerfully expressive (and, come on, it was the obvious choice ☺). In contrast to our linear agent, who presume the Q function is linear, with deep q-learning we have a wider range of families of functions (including non-linear ones).

Our neural network is basically an MLP (input is a feature vector and output is a scalar). We explored and tested many variants of the network architecture (number of layers, width of each layer, dropouts, activations) against various board sizes. Since architectures with more parameters didn't drastically improved the learning and they take more time, we settled on the following (no plots this time since we played with it manually):

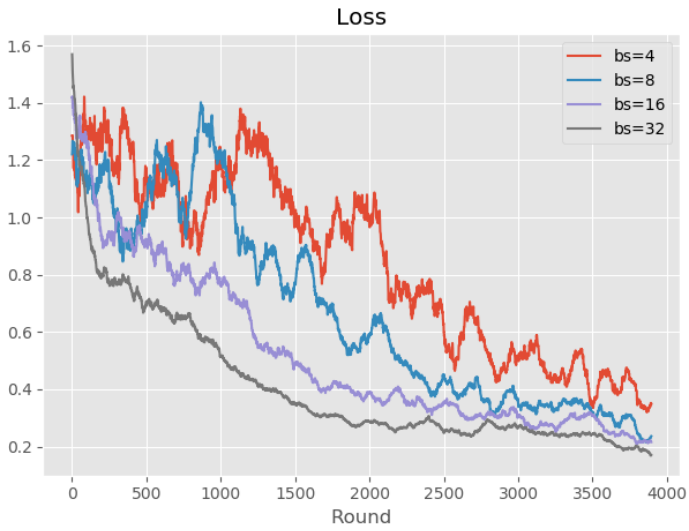
Layer Name	Size	Activation	Dropout
Input	# features		
Layer 1	256	ReLU	0.3
Layer 2	64	ReLU	0.3
Output	1		

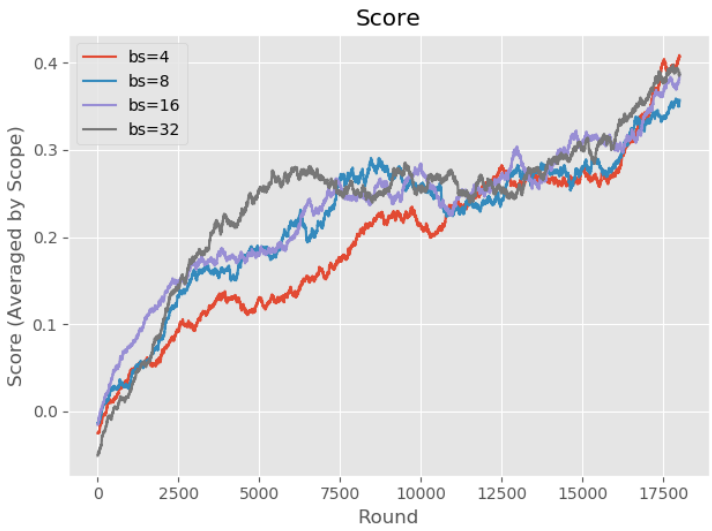
In the following tests we used spatial radius of 2 and distance scope of 5 so the number of features is $(25 * 11) + (7 * 11) + 1 = 353$

Learning Algorithm

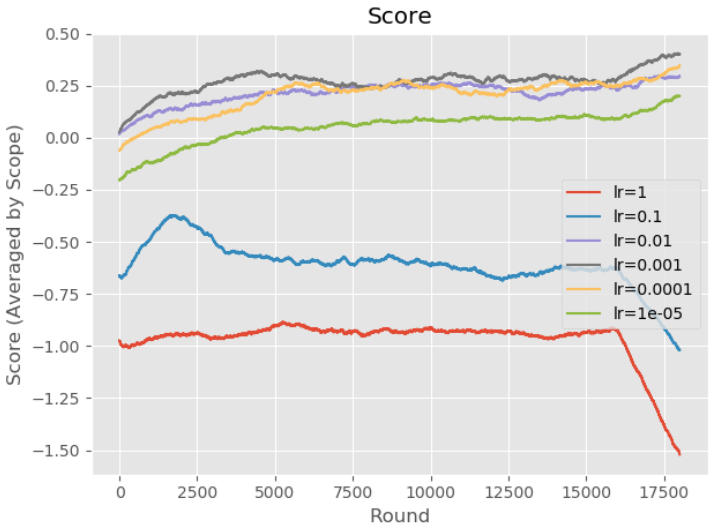
We manually tested the SGD optimizer against Adam optimizer. We didn't see a very drastic difference between them but decided on Adam because of our knowledge – it takes the best of two worlds: AdaGrad and RMSProp. Essentially, we believe it is better to maintain individual learning rates for different parameters and to utilize momentum.

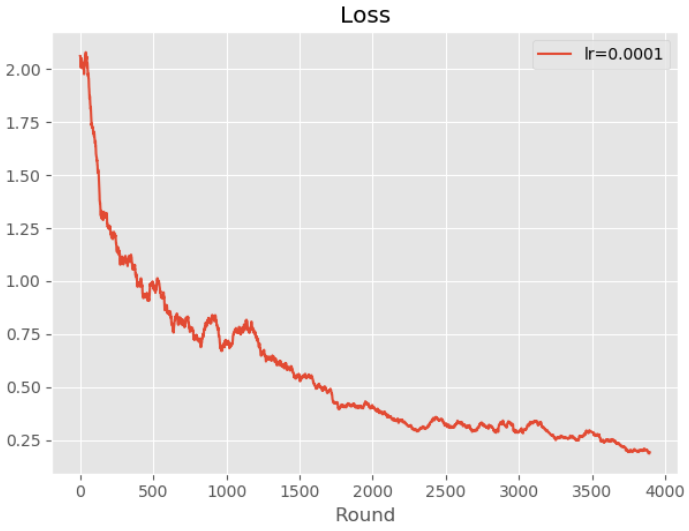
Batch Size choice was both according to learning curve *and* the constraints of the learn function time. The final choice is 16 (32 performed quit well but for large board size, combining with the time it took to compute the feature vectors, it was too much. Its performance is quite similar to the 16 batch size so we didn't even feel a need to use the "too_slow" parameter).

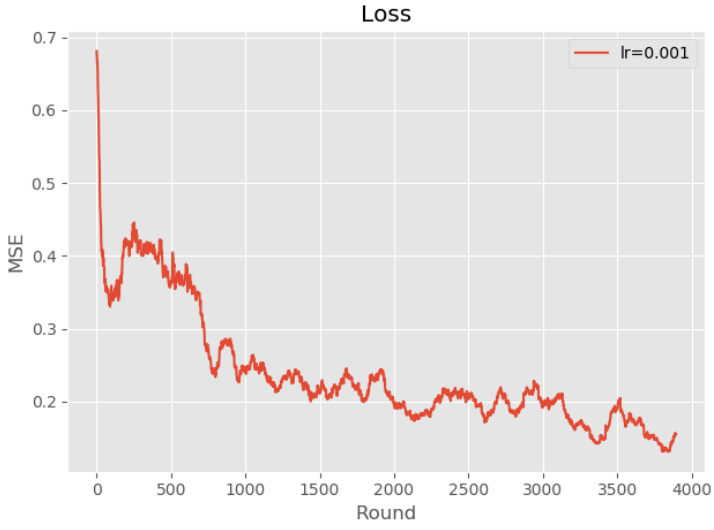


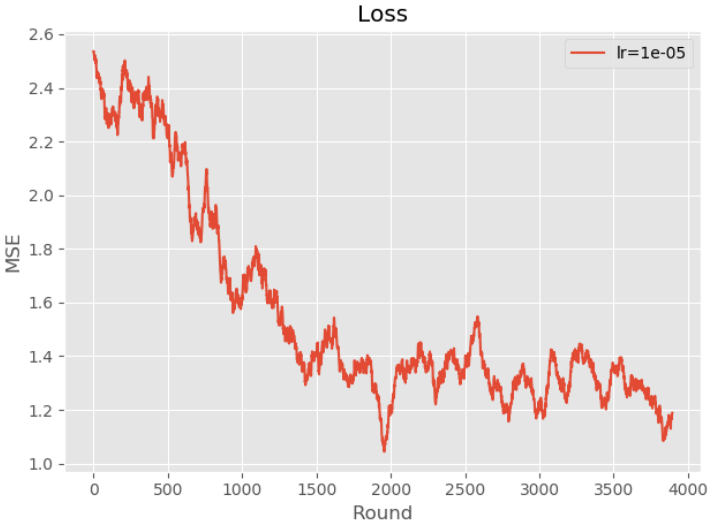


Learning Rate was a bit trickier to choose. We separate here the plots of the loss because they are not on the same scale and was difficult to visualize together. We chose learning rate of 0.0001 because it is has the most monotone consistent loss and its score is among the highest.





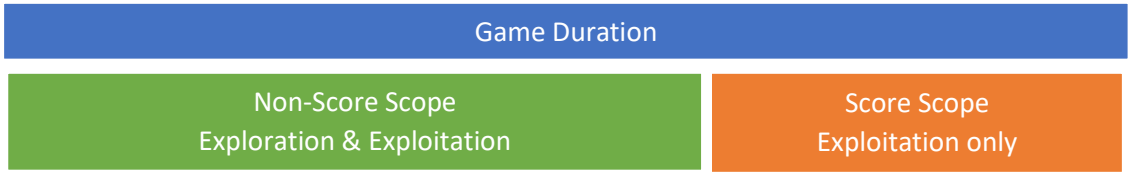




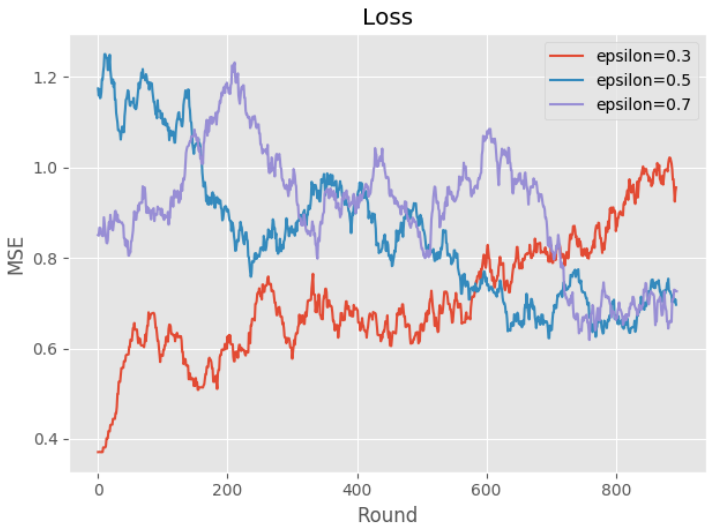
Exploration-Exploitation Trade-off

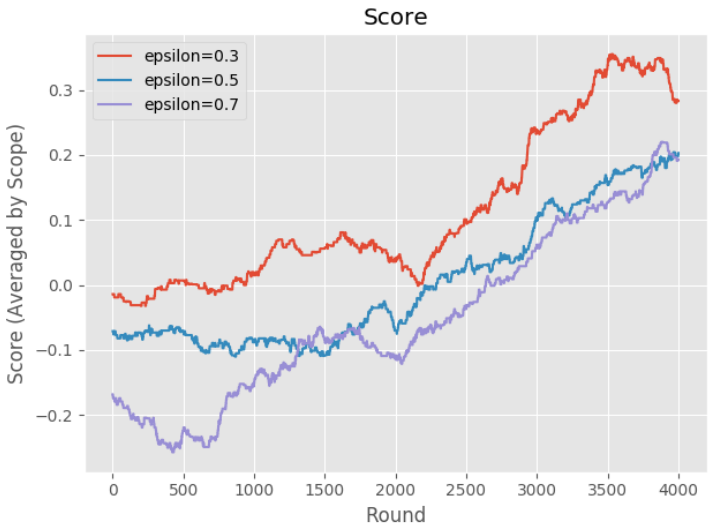
We fist examined which epsilon value would provide to us the best learning curve and after we fixed this parameter we tested linear decay.

Let's look at the following chart:

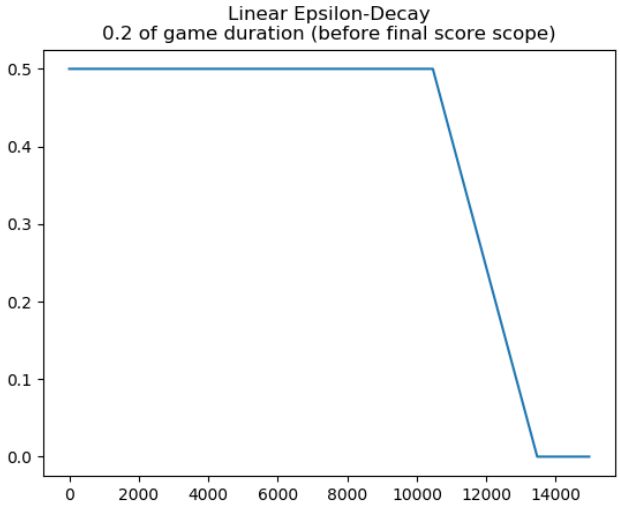


First we tested a fixed epsilon throughout the Non-Score Scope that changes to zero when we start the Score Scope. According to the following tests, we decided on 0.5 as epsilon because the loss was most decreasing and the score was most monotone consistent:

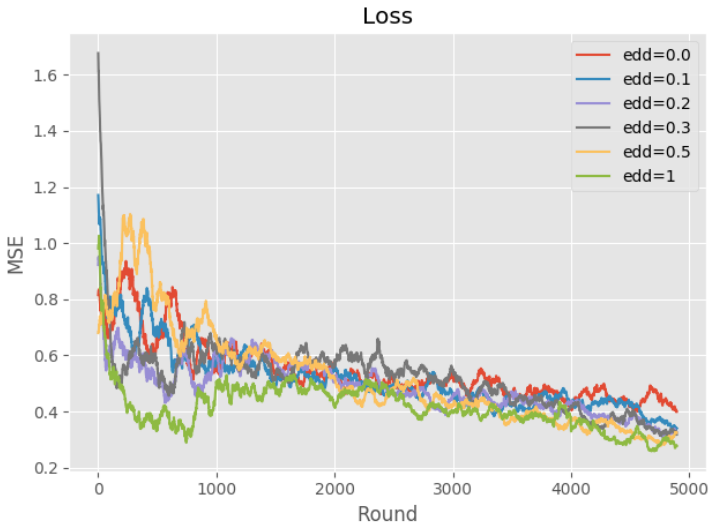


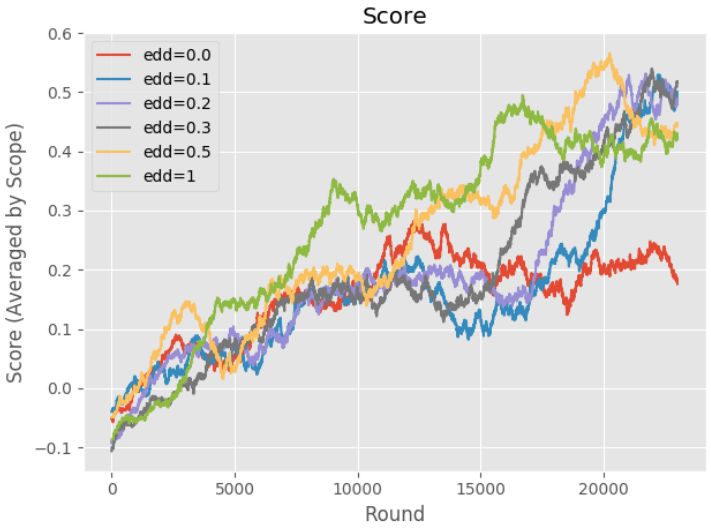


We define the decay range by the percentage of the Non-Score Scope the actual decay happens. For instance, if we don't want a gradual decay, a range of 0% means we have a fixed epsilon during the Non-Score Scope and a sudden change to 0 when entering the Score Scope. The decay itself is linear. Here is an example of the value of epsilon during a session:



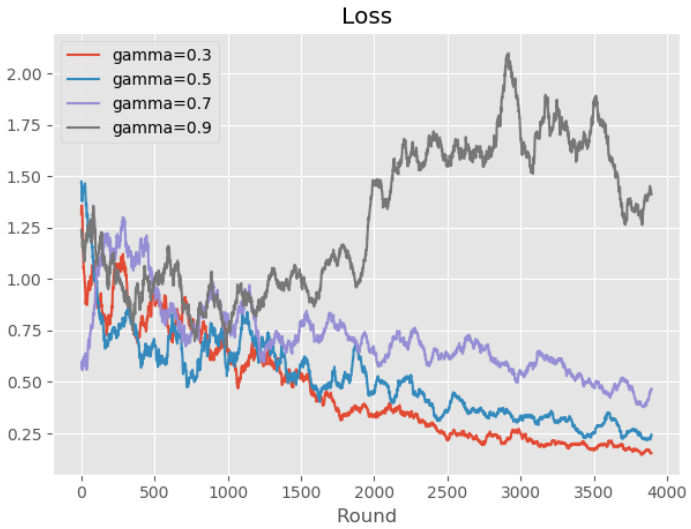
After looking at those results, we decided to have a decay range of 20% (tested on epsilon starting at 0.5):

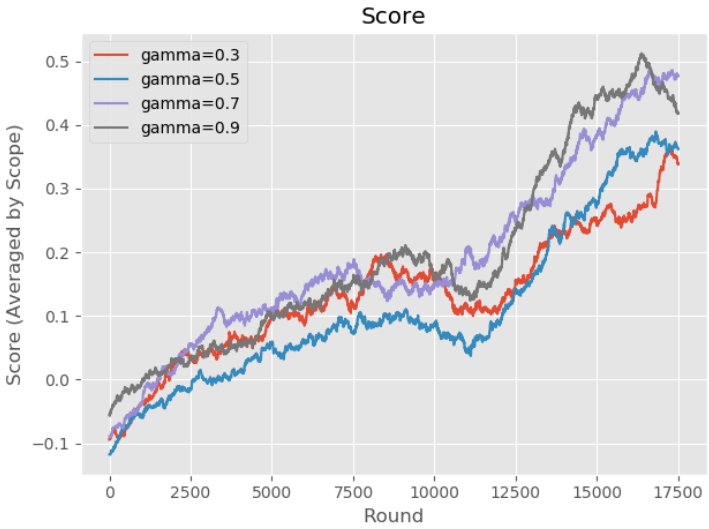




Choosing Gamma

It is obvious that the gamma in our custom model would be higher than the linear model because with our feature vectors we can see farther ahead. After the we decided on gamma=0.7:





Other Solutions We Discarded

Initially we read a lot of information about Reinforcement Learning in games. We discovered that often the Deep Q-Learning is done not on a feature vector but the input is the game board as an image. In those cases, the network's output is a vector with length equals the number of possible actions, so each output entry is the Q value as if the state is the network's input and the action is corresponding to that entry.

We implemented a convolutional neural network that gets as input the board game, with preprocessing variations such as centering and rotating the board such that the snake would look north and positioned in the center (by "rolling" the board). Also, to deal with rotation of non-squared board, we cropped to the minimal dimension. Also we tried normalizing the board values to the range of zero-one or dividing each symbol to a different channel and using only zero and one values.

This solution was not good mainly because the convolutional neural network took too much time to compute and greatly surpassed the required act time and learn time. Additionally, even when we set bigger times, just for testing, we did not manage to easily discover good hyper parameters to allow for meaningful learning.

Another idea we didn't use is to have our custom model include an internal linear agent to act according to it in the first few thousands rounds. Why? Because we see that the linear agent can learn to eat good fruits quite early and we rationalized that if we have an agent that learns slowly against Avoids, when played against quick-learning agents it might never encounter states where it's eating good fruits since they are getting eaten by the other agents. We thought it might help our agent in the tournament against other students' agents but we discarded the idea because the testing was complex and we preferred to focus on other parameters testing in the time we have left.

Final Words

Thank you! We enjoyed working on this project ☺