DATASCI 3ML3 Final Project – Final Report

Topic: DNN Snake Game Player

Instructor: James Lambert

Due Date: April 20th, 2023

Part 1: Introduction

Snake is a classic action video game where the player navigates a snake inside a rectangular box. Upon keeping the moving snake away from colliding with both boundaries and its body, the snake lengthens when food is eaten. The final goal of the Snake Game is to fill the whole game box with the snake body.

As the Snake Game has developed for hundreds of years, human players reach some boundary on the Snake Game speed run. Some players are curious about whether it is possible for a significant improvement from human's strategies, so they use computer computation/algorithms to prove/disprove such better strategy exist.

We noticed that there exists a naive algorithm to achieve the final goal. If we can find a "circuit of full snake" which let a snake of length m*n fully occupies the m x n game playground and have its head be adjacent to its tail, then our snake can always follow such circuit to avoid collision. Moreover, the snake would eat food and gain unit length at least once every m*n movement, as food is generated in a grid of playground.

But the problem is that the above algorithm is not optimal as it is not efficient enough. It needs O(m*n) movement for a unit lengthen. So, total complexity would be O(m^2n^2) even if we know the "circuit of full snake". To improve efficiency, we think about the problem of the naïve algorithm. By intuition, the snake should not detour away from the food at the beginning of the game ideally, as there exist some simple paths go directly to the food which prevent collision. Getting away from the "circuit of full snake" won't affect achieving the final goal since the snake can be easily recovered to follow the circuit and avoid collision whenever its length much lesser than playground length/width. So, one of the motivations is that we want to find an algorithm which solves snake game problem and balances both performance and efficiency.

Another motivation is that we want to have a deep understanding of different activation functions, such as ReLU, tanh, and sigmoid. We try those activation functions on our deep neural network. The training result will give us an intuitive idea why ReLU is the most popular activation function. We will also talk about when different activation functions should be used.

One more motivation is to unleash potential of AI in the Snake Game. By our test, random move algorithm gets 0.2 scores in average, human players get about 15 scores because human make mistakes when snake moving fast. We expect our model to get at least 30 scores on average to be successful. We play around with hyperparameters of the model to achieve this goal.

As this project is doing reinforcement learning, we will use Q-learning method. Our model will predict the Q score of different actions based on current states. Since we do not know the true Q score of current states, we cannot use the true Q score in loss function. Instead, we compute the current Q value to be the sum of current reward and best Q score of the next state times discount rate. Then, we can use the back propagation to curve the model prediction towards current Q value.

The other method is balance exploration and exploitation. In the beginning of the train step, the model is highly possible just do random move. The probability of random move linearly decreases to zero as game number increases.

By using the above methods, we have a couple of results from this project. First of all, deep neural network is a method to balance efficiency and performance. Second, not all activation functions are suitable for this model to solve Snake Game problem. Third, our deep neural network with Q-learning is capable of performing better than humans to play the Snake Game.

We also found some ways to promote the performance and efficiency of our model. One of the promoting methods is to use a more detailed input layer, e.g., including snake length, snake shape measurement. Then, the model will be more conservative as snake lengthens or its shape become danger. Another promoting method is to build a more complex neural and try dropout.

Part 2: Methods

First, our model is a fully connected neural network. It takes the current state of the snake, including movement direction, food direction, and barrier direction. Each direction is represented by four identical neurons, so upper, right, down, left is linearly independent. After calculating the hidden layers, it outputs three Q scores of three actions, turn left, go forward, and turn right. The action which has the max Q score is the best action predicted by our model. A visualization of our model construction is provided below, with some neurons in hidden layers omitted.

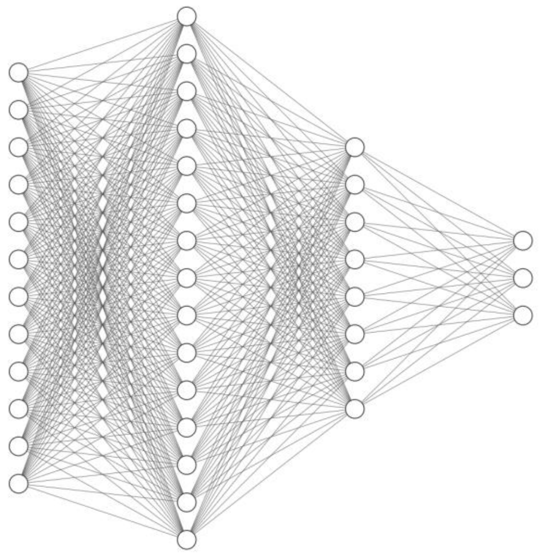

FIGURE 1

Here are some advantages of this model. One advantage is that direction is linearly independent, then the reward of different action could be properly separated. Another advantage is that the size of hidden layers is large, so I expected it could perform well even if the dead ReLU problem exists, and it provides potential for applying the drop out. The third advantage is that this model is easily simplified to an O(1) decision algorithm. Since the input is just 12 neurons of either zero or one value, it can be reduced to an array of size 2^{12} . The action predicted by the model is recorded in this array and can get in O(1) time.

A disadvantage of this model is that this model does not take snake length and danger level into consideration. As discussed in the introduction part, the decision of snake movement is relative to the snake length. The longer the snake is, the danger it could be collided with its body or boundary. So, ideally our model should take other input neurons which reflect measurements of snake length and danger level, such as existence of 'C' shape in snake body. For example, the snake head moving into a 'C' shape of its body is danger as shown in graph below. Then, the model should be more likely to choose conservative movements to prevent game over.

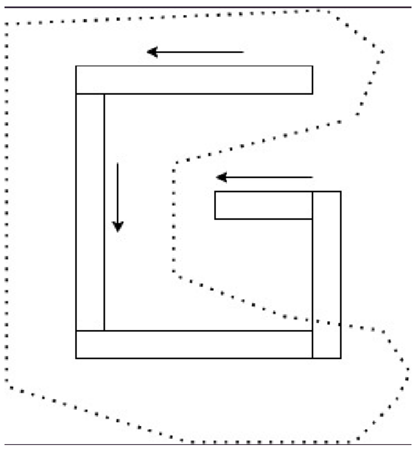

FIGURE 2

Second, we use Q-learning method to measure scores of each movement. At the beginning, we just know that game over is bad and eat food is good. We set the reward of game over movement to be -10 and reward of eat food movement to be 10. We don't know the quality of intermediate movement when the snake neither collides nor eats food. So, we use the Q-learning formula:

$$Q(current\ state)\ =\ reward\ +\ \gamma\ *\ Max(Q(next\ state))$$

while Q(next state) is approximated by model(next state), and γ be the discount rate. Then, the current prediction is:

$$model(current\ state)$$

and the ‘true label’/better prediction of Q score is:

$$Q(current\ state)$$

By using criteria function, we compute the loss as:

$$Loss = criteria(Q(current\ state), model(current\ state))$$

Next, by applying back propagation, we train the base model to predict Q scores on intermediate movements.

The advantage is that this method is easy to implement. As long as the Q(current state) is better than the model(current state), our model will keep on optimizing. By choosing proper hyperparameters, the model will converge to a proper prediction of Q score of each step. Moreover, the best model is changed only if the new model played 10 games and got a higher average score. This efficiently prevents overfitting. The disadvantage is that the training progress is sensitive to hyperparameters. If the choice of hyperparameters is not good, it takes longer for the model to converge, or the performance would not be good.

The other method is to train both short-term and long-term memory. The short memory is trained every single movement of the snake, updating QNet by Q-learning method we described before.

The long-term memory is trained when a game is over, it reads a batch of data from the memory and trains the model.

The advantage of this method is that it reuses the data that was generated before, which increases efficiency. Another advantage is enforcing long-term memory. Since the effect from data trained long ago would be covered by later training, randomly reading from memory dataset, and training a batch size after game over could help enforce long-term memory.

Next method is balance exploration and exploitation. In the beginning of the train step, we have to get some data from random decider. The reason is, though weights of neurons of our model are randomly selected, our model output is deterministic given fixed inputs. So, if we do not use random decider, the data would be imbalanced, and the model would not do exploration. (I.e., the model is less trained or never trained in some states.) Then as game numbers increase, the probability of random move linearly decreases to zero. From then on, the movement would never depend on random decider anymore. The model can focus on exploitation then.

Part 3: Results

The history of training is shown below. The blue polygonal line shows the score of a separated game during training. The red polygonal line shows model evolution process. When we suspect a model is better than the old model (the model before some back propagation steps), we play the snake game with the new model a couple of times and compare average scores. The curve up of red line means a model with higher average score was found. Otherwise, the curve is horizon if new model has lower or equal average score.

As a result, the ReLU version DNN performs well. Its score and average score significantly increased when about 170 games have been trained. After about 220 games, the average score reaches 50, which is enough for our motivation.

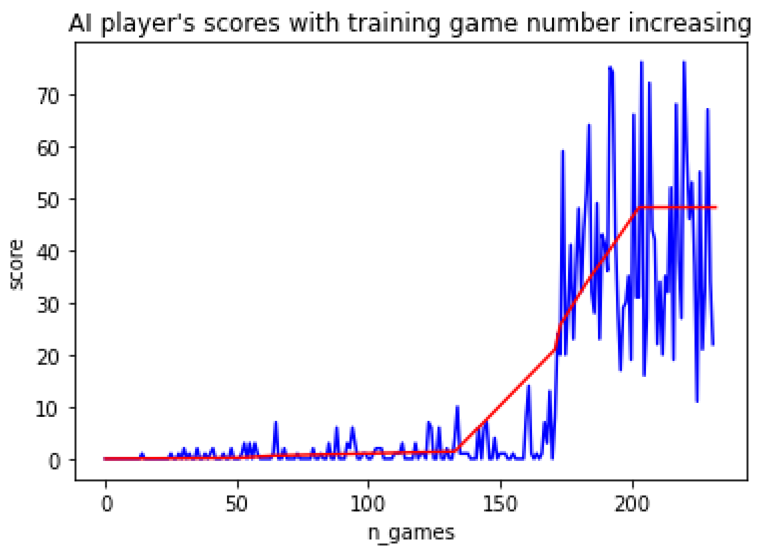

FIGURE 3

However, the Tanh version DNN does not work well. By using the same parameters and same setting (except the learning rate), the training process is like below:

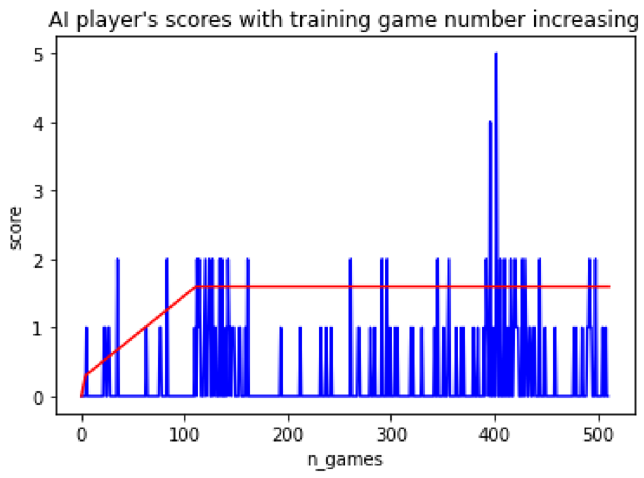

FIGURE 4

It shows that the maximal score is just five and our model gets zero scores in about half the games. The best model achieves about 2 scores on average. It is much lower than the ReLU version of our model. We made a little change to our model to improve its performance. We remove one hidden layer and reduce the learning rate. We add penalty of movement when snake neither eat food nor go dead as we found the trained Tanh DNN trend to go in a closed cycle to prevent death. Then, we get a little improvement:

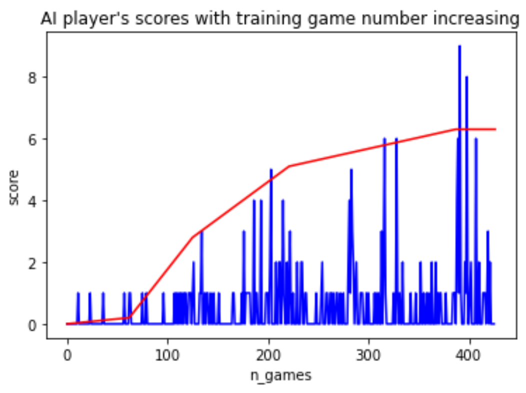

FIGURE 5

By Figure 5, the Tanh DNN gets zero or one score in most games during training. The model is properly trained as the scores get higher. Finally, the best Tanh DNN with highest average score get 6 scores in average.

Now we compare its performance with the performance of random decider. In the below graph, we let a snake move randomly play 500 games and record its scores:

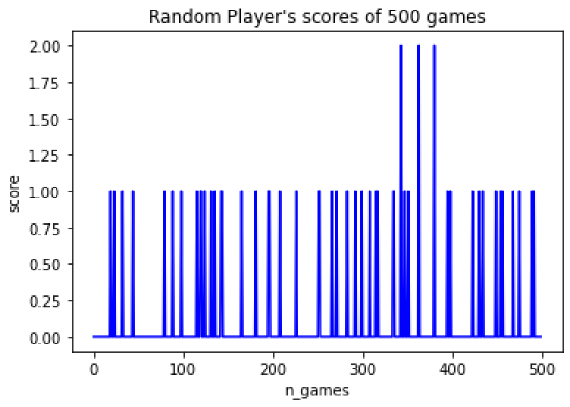

FIGURE 6

It shows that Tanh DNN with proper parameters is much better than random move decider, but far worse than ReLU DNN.

Then, we focus on ReLU DNN as it works the best for our goal. We played the Snake Game by trained AI a couple of times. We expect the final shape of snake could show some weakness of our model. Here are four final graphs of the snake when game over:

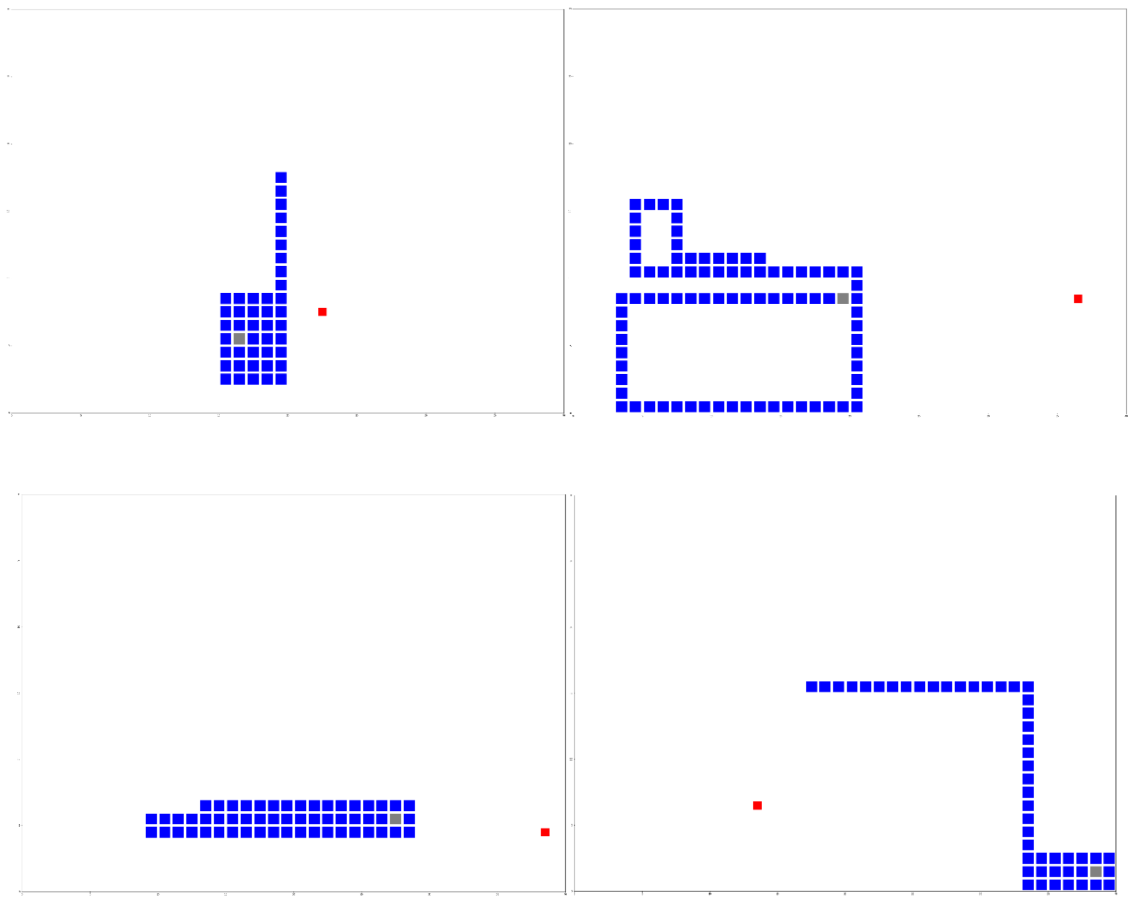

FIGURE 7

Those images reveal some problems of our ReLU DNN, which we will discuss in the next part.

Part 4: Conclusion/Discussion

By Figure 3, the success of the ReLU DNN indicates our methods, including hidden layers setting, Q-learning, long/short term memory, and balancing exploration and exploitation, work well. As there are no individual measurements of these methods, we can only conclude that they work well as a whole.

The success of this model may be attributed to properly choosing parameters and using the famous optimizer, ‘Adam’, and activation function, ‘ReLU’.

By comparation of Figure 3,4,5,6, the ReLU version of DNN works the best. It achieves 50 average scores which satisfies our expectation. The Tanh version of DNN did just a little better than random decider, both Tanh DNN and random decider are far away from satisfying our motivation.

The reason why ‘Tanh’ activation function performs worse than ‘ReLU’ may be because the ReLU function allows the network to learn more discriminative features by selectively activating neurons that are sensitive to particular patterns in the input.

From Figure 7 we also find some problems with our ReLU DNN:

In the first, third, fourth graph of Figure 7, the snake head is totally surrounded by its body, which indicates that there exists at least one wrong action the snake did. It leads the snake to a dead situation where the snake would dead in some movements no matter what action it did after the wrong action. This is due to the defect in the model as we discussed before. The model does not take snake length and danger level into consideration. Though the snake could act to prevent coming into danger area because it has trained before by a game when it come into danger and dead, such training data may not enough. Some possible solutions would be measuring danger levels which affect conservative level and training more on specific training data.

In the second graph of Figure 7, the snake collides even if there is action can be chosen to prevent collide immediately, which indicates training data is not enough and training have best to continue.

But as it is a DNN, neurons weights of some neurons are shared. Training and increasing performance in one state of the snake could result in decreasing performance for the other state of the snake. The above ideas may not lead to actual improvement of the DNN model.

The true problem is the model should describe the snake's body and danger level with more detail as we discussed in part 1. Moreover, as ReLU has dying ReLU problem, we can use substitution of ReLU such as ELU to try to train better DNN model.

In conclusion, AI has potential to achieve 30 scores in the Snake Game on average. The ReLU DNN satisfied our expectation of more than 30 scores on average, using Adam optimizer. Tanh DNN performs worse than ReLU DNN, as it has some disadvantages.

Though there are tricks can use in training step to slightly improve the model performance, the best way to improve our model performance on the Snake Game is to modify the input layer and provide the model with detail information of the snake body.

Part 5: Reference

https://www.youtube.com/playlist?list=PLqnslRFeH2UrDh7vUmJ60YrmWd64mTTKV

https://towardsdatascience.com/today-im-going-to-talk-about-a-small-practical-example-of-using-neural-networks-training-one-to-6b2cbd6efdb3

https://www.freecodecamp.org/news/train-an-ai-to-play-a-snake-game-using-python/