Effects of Memory Capacity on Performance of DQN in Blackjack

William Weng (williamweng917@g.ucla.edu)
Department of Psychology, 502 Portola Plaza
Los Angeles, CA 90095 USA

Kexin Wang (ykx1998@g.ucla.edu)
Department of Psychology, 502 Portola Plaza
Los Angeles, CA 90095 USA

Abstract

With advanced reinforcement learning algorithms, neural networks have become better at learning more complex games. While most networks focus on learning to play a game, in this paper, we investigated if our network could learn similarly like us. Specially, we focused on whether additional memory capacity would help network performance in the game of blackjack. We find that only through meaningful network architecture, combining a belief network with Deep Q-Learning network, our network was able to extract meaningful information from additional memory.

Keywords: Blackjack; Deep Q-Learning; Belief network; Reinforcement Learning

1. Introduction

1.1 Blackjack

Blackjack is a popular casino game. The first documented record of blackjack is in Miguel de Cervantes's novel (written in 1601) when the Spanish writer described his main character cheating in the game. The player's goal of the game is to acquire a hand of cards with a higher sum than the dealer but smaller than 21. The game is simple enough for most people to understand but the underlying mechanics are far more complicated that no clear optimal strategy is clear.

1.2 Deep Q-Learning

Deep Q-Learning is a type of reinforcement learning stemmed from Q-Learning. Deep Q-Learning applies neural network and replay buffer to help the model better understand the game when there are too many unique states the game could be in (Mnih, etc., 2013), which is the case in the game of blackjack.

1.3 Deep Q-Learning of Blackjack

In this paper, we implemented two approaches to explore the effects of increasing memory on the performance of the model. The first approach we used is inspired by Wu's network. We modified on top of his model by adding new input nodes. For the second approach, we proposed a unique combination of two models, a belief network and a Deep Q Network (DQN). The belief network produces a belief distribution of the probability landscape of the next card; and the belief was inputted into the DQN for training.

1.4 Related Work

Deep Q-Learning is a well-known reinforcement learning method in the field and had already been used in Go, Chess, and other sequential-decision games since its inception. However, in the game of Blackjack, where an optimal strategy exists, reinforcement learning still could not reach the optimal (Wu, 2018).

We studied past research into blackjack and found most researcher focused on the optimization of performance. In this paper, we are more interested in the relationship between the amount of input information and expected payout, or in simpler terms, memory and performance. Although our paper's focus is different from others, their implementation could generalize to our model.

2. Model 1 – A Standalone DQN

We proposed two approaches; and although they share a lot of similarities in the actual code, the meaning behind each one is quite different. In this paper, we will describe each model separately and provide results associated with each one.

2.1 General Setup (Game Engine)

In both of our model, we set up the game in the same way. There is a game engine which 1) initiates the game, 2) processes input from the model, 3) produce output to the model, and 4) signal the model the reward for each round.

We implemented the following rules for our game engine. First, we defined the total number of cards for this game is a total of four decks, 208 cards. The game will automatically restart (reshuffle) after 170 cards had been played. There are only two players involved, the dealer and the agent. The goal of the agent is to beat the dealer by 1) having a larger sum or 2) the dealer having a sum exceeding 21 (exploding). The agent will always start the round and has two decision, hit or stand, meaning draw and not draw. The dealer has a drawing rule that he must draw if the sum is smaller than 17 and he cannot draw if the sum is larger or equal to 17.

In each round, game engine will give the agent (our model) two cards, both facing up, and give the dealer two cards, one facing up and one facing down. Now, the agent will decide to hit or stand. After the agent decides, the game engine will process the agent's action and continue the game in two ways.

If the agent chooses to hit, the game engine will give the agent another card facing up, if the agent's sum is smaller than 21, the game engine would request another decision from the agent. If the sum of the agent now is larger than 21, the agent loses, and another round is started.

If the agent chooses to stand, the engine will follow the dealer's drawing rule and compute the sum for each player and declare the winner, loser, or a draw. The game repeats itself in this fashion.

2.2 Model Specifications

In this first model, we utilized a DQN to act as the agent through iterations of game. The model has one input layer, two hidden layers and one output layer.

Input Layer

Our input layer has 36 input nodes. For the first 30 nodes, the first 10 represents the current hand of the agent, the next 10 represents the hand of the dealer, and the last 10 represents the memory of the agent of past cards. For each node, the position represents the card value it represents, and the input represents how many are there. For example, if a player remembers the past three cards of ten, six, and six, the current agent hand is four and four, and the dealer is showing a ace. The corresponding representation on the input layer is 2 on node #4, 1 on node #11, 1 on node #30 and 2 on node #26. The last six nodes represents additional information about the player's sum, the dealer's sum, whether player has an ace, is this the first turn, how far is the agent from 21, and how far is the dealer from 21.

Hidden Layers

We have two fully connected hidden layers, with 64 nodes and 32 nodes with ReLU activation function.

Output Layer

The output layer has two nodes with a softmax function to simulate the probability of hitting and standing in any given situation. The sum of the two nodes is 1.

Training and Updating

Since the game of blackjack has many unique states, we do not want the model to always correct itself based on every single iteration but we want it to have a more generalizable sense about the game. In Deep Q-Learning, we achieve this through having a Priming Q Network (PQN), a Target Q-Network (TQN), and a replay buffer.

The PQN produces the action of the agent; the TQN calculates the loss, the error; and the replay buffer stores all the game samples. At the end of each round, the game sample is stored in the replay buffer. After some iterations, a batch of samples from the replay buffer would be drawn to update the Q network through calculating the error with respect to the TQN. Then after a predetermined number of iterations, the PQN's value is copied to TQN. The practice of having another Q network is to prevent the model shifting too much and too frequent or getting stuck into a feedback loop.

The loss function, or how we generated our error, is through the Huber Loss Function, which calculated how far our prediction is from the target values.

$$Loss = \sum (Q_{Target} - Q_{Predicted})^2$$

The loss is very similar to the error term in supervised learning. However, since we do not have the correct label, we created our own label, Q target value.

$$Q_{Target} = r^t + \gamma * maxQ_{Target}(s^{t+1})$$

Due to the complexity of the game, we encouraged our model to explore untraditional options through ε-greedy exploration. This exploration process encourages to deviate from the Q values in early training rounds and the intensity of encouragement decreases as rounds increase. The process is implemented as followed: if a random number is generated and is smaller than ε, the player will play randomly, regardless of the Q values. If the number is larger than ε, the player plays the action based on the Q values. The value of ε decreases as the following:

$$\varepsilon = \varepsilon_{start} + \frac{\varepsilon_{start} - \varepsilon_{end}}{k_{decay}}$$

$$k_{decay} = \varepsilon^{\frac{-k}{\varepsilon_{decay}}} \; ; \; k = current\ iteration\ mod\ "switch"$$

Hyperparameters

The model is trained 2e5 iterations. The learning rate is set at 1e-5, and decays to 5e-6 to the later half. The discount factor when calculating target Q value is 0.9. The replay buffer has maximum capacity of 10000 samples, and continuously replaces old ones with new one. The batch rate is for each training sample is at 20. For exploration, the initial ε is set at 0.95, final at 0.01, with decay rate of switch / 3 and switch at 5e4.

2.3 Results

After training, we found no effect between increasing memory size and model performance as seen on Figure 1. The average performance and maximum performance of models across different memory sizes are similar. The performance fluctuates as memory size increases. However, our model's performance is stable across different memory sizes, as seen on Figure 2. The performance generally stabilizes after 50 thousand iterations.

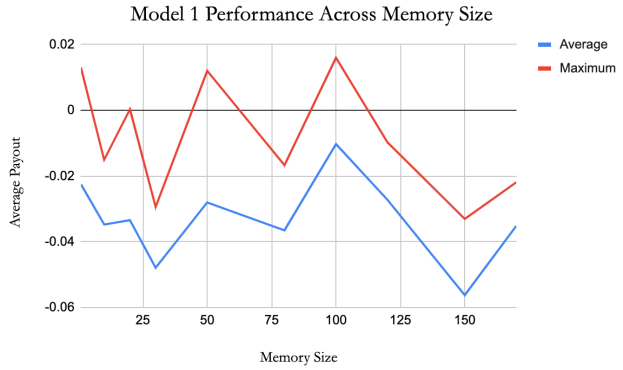

Figure 1: Model performance does not depend on memory sizes.

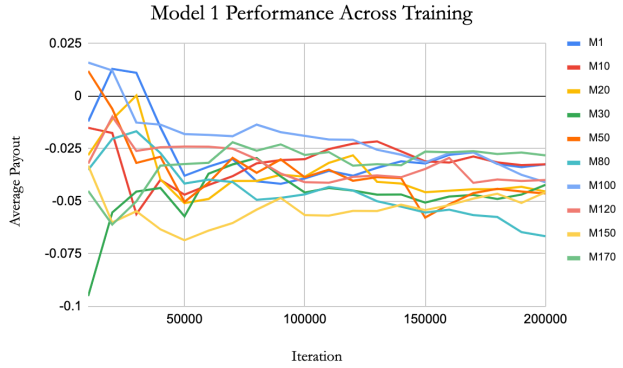

Figure 2: Model performance stabilizes after 50 thousand iterations.

We think the reason for our model to converge but not produce our expected result because the model could not comprehend the additional memory information in a constructive manner. Simply inputting memory information alongside with current cards on the table may cause confusion for the model.

3. Model 2 – Belief Network + DQN

In this model, the general setup is the same as the previous and the DQN's structure and training are relatively similar as well, only with few minor tweaks in the input layer and some hyperparameters. However, the main difference between model 1 and model 2 is the addition of the belief network and how it impacts the model performance in response to memory size.

After seeing model 1 failed to establish any relationship between memory size and performance, we questioned this phenomenon. We know, as human, we could perform better if we have perfect memory of past cards. So, with this additional information, the model should, but did not, perform better. We believe that the model couldn't understand the input in a meaningful way; and therefore, we constructed meaning into the additional input through a belief network.

3.1 Belief Network

The belief network is a fully connected neural network with one input layer, two hidden layers, and one output layer. The input layer takes in information about the current and past cards. The output layer depicts the probability landscape for the rest of the cards left in the deck.

Input Layer

The input layer has 21 nodes. Node 1 to 20 follows similar representation with the input layer from model 1. The first ten represents the cards on the table at given state. The second ten represents the agent's memory of past cards. The last number is the certainty of the agent, calculated by the total number of cards played minus the agent's memory size.

Hidden Layer

There are fully connected two hidden layers with 64 and 32 nodes and a random initializer.

Output Layer

The output layer has ten nodes with a softmax function. Each node represents the probability of that card appearing in the next draw. Eg. Node 1 represents the probability ace appearing in the next draw.

Training and Updating

The belief network follows a separate training scheme from the DQN but similar. In the belief network, the agent is not exposed to the true distribution at any given state. So, we need to construct a way to define error. In our network's training setting, if the belief network has a probability of the next card being four is 10% and the next draw is a four, it would calculate as the true distribution of the card is 100% for four and 0% for all others. Although the label we provided is not correct, in the long term, it would "push" our belief network to have an accurate understand of the landscape.

$$Error = (D_{BN} - D_{nextDraw})^2$$

3.3 Modified DQN

The modified DQN keeps the same updating structure, hidden layers, and output layer. We changed the input layer to 18 nodes. The first ten nodes come from the outputs of the belief network. The other eight represents the game information we computed (the first 6 nodes are the same as the last 6 nodes for the first model and added 2 more nodes about the memory size of this player and the total number of cards being played in the previous rounds.)

3.3 Hyperparameters

We trained the model 1e5 iterations. The learning rate for the DQN is the same at 1e-5 and for the belief network is 1e-3, with decaying to half after half of the iterations passes. The discount factor and the replay buffer were kept the same, at 0.9 and 10000 respectively. The batch size was reduced to 1 and ε was kept the same.

3.4 Results

Our model this time converges much faster and shows a positive relationship between memory size and performance.

In figure 3, after 40000 iterations, the model performance across all memory sizes start to stabilize, meaning our model was capable finding the minimum. In figure 4, we see that model performance increases as the memory size of the model increases. The relationship is between linear and sigmoid. However, based on the resolution of the graph, we could not conclude and would need more models across more memory sizes.

One thing worth noting is that, for memory size under 40, we see that it is nearly identical to the model of memory size of one. We believe such result is cause by the lack of certainty of the model to make accurate results and leading to consistent performance in the lower range of the memory size.

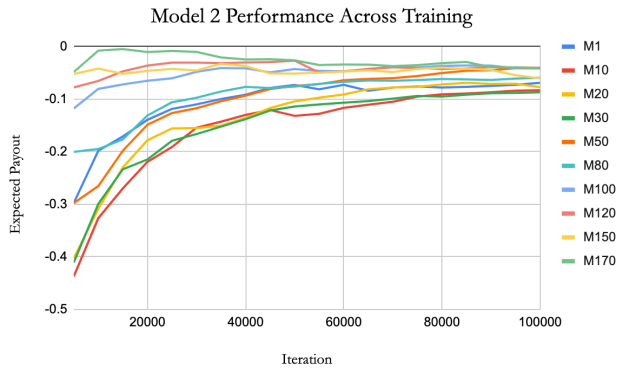

Figure 3: Model 2 performance converges after ~ 40000 iterations.

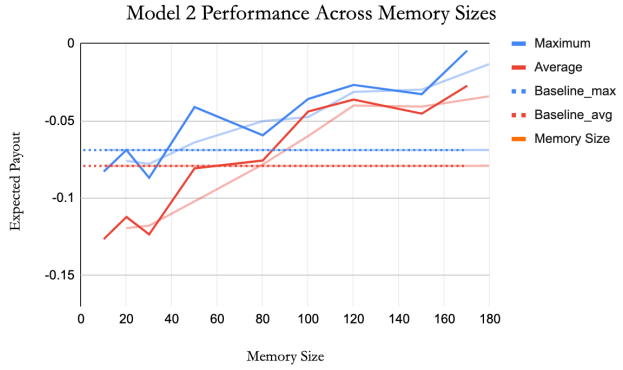

Figure 4: Model 2 performance shows a positive relationship in respect of memory size.

4. Conclusion

In this paper, we discussed the effects of increasing input information on model performance in the game of blackjack. While inputting the information without a meaningful structure, our model was not responding to the additional information. However, as we restructured our model and incorporated the belief network, suddenly additional information input helps our model to perform better.

The reason why we were interested in this topic is simple. As human, we achieve better results with better memory; but is it the case with neural network? And if not, why is it not? Through our two models, we found that additional information was meaningless to the network if it could not make "sense" of it. And only be having a structure that have a clear goal, the model would benefit from the additional information. In plain term, model does not work better if you just add more inputs – you need a better architecture as well.

We also understand that our model still have a lot of questions unanswered. First of all, we did not test for the accuracy of the belief network. By testing the belief network, we could better understand the stability and functionality of the belief network, independently of the DQN. Second, under small memory size (<40) the effect on additional memory size is small, but what caused it? How do we test it? Lastly, the hyperparameters of the belief network could be tweaked to achieve higher outcome.

In the end, although our model focuses on the game of blackjack. This finding could be generalized to other reinforcement learning tasks. While constructing a model for a specific task, we could learn from the human decision process and incorporate different steps human make decision and design an architecture similar to that. In our model, we achieved this through a belief network because we believe the reason why we can become better at blackjack with more memory capacity is because we could use this memory information to gauge future card more accurately; so a belief network mimics our thinking process and proven to be successful.

5. Acknowledgments

We want to thank Professor Zili Liu for his guidance and our TA Lucy Cui for her unconditional support.

6. References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Lillicrap, T. (2017). *Mastering chess and shogi by self-play with a general reinforcement learning algorithm.* arXiv preprint arXiv:1712.01815.

Wu, A. (2018). *Playing Blackjack with Deep Q-Learning.*