Reinforcement Learning Applied to Cribbage

Danielle Cloutier

December 15, 2022

1 Introduction

Cribbage is a card game, normally played between two players which is split into multiple phases of play[1]. In this project, I attempted to create a neural network model that uses Deep Q-Learning to learn how to play the discard phase of the game. This decision is because the game's different phases essentially require completely different decision processes, which would likely require two separate agents to be trained somewhat in tandem and would require far more computational power than I possess to train. This network was trained in a multitude of different ways to verify certain methods that have been used by others for other games.

1.1 Reinforcement Learning and Deep Q-Learning

Reinforcement learning is a paradigm of machine learning which deals with learning in sequential decision making problems where there is limited feedback. This is done by imitating a Markov Decision Process. This is a process where an agent chooses actions on a state space based on a policy to optimize the amount of reward that the agent receives[2]. This policy is often depicted using a table, often called a Q-Table, and is then continually updated so the agent can make better decisions, using a Bellman Equation [3]:

$Q(s,a) = (1-\alpha)Q(s,a) + \alpha * (r_t + \gamma * max_a(Q(s',a)))$

where α is the learning rate and γ is a discount rate, so the agent learns to more heavily emphasize current rewards over future rewards. We also consider s and s' as the current and later states, with a being the action on that state. We also encode a reward, r_t, in this equation, which is the reward that the agent will receive from choosing that action. This equation is then recursively iterated, until the agent achieves an optimal policy. This process where we store these values in a Q-Table which is initially randomized is called Q-Learning[2].

One issue with this method is that as the state and action spaces enlarge, it requires significantly more memory to store these significantly more massive tables. This is especially an issue in my specific case, where a player is dealt 6 cards from a 52 card deck, meaning there are $\binom{52}{6}$ different hands, or more specifically, 20,358,520 possible hand states and 15 actions, which will be defined in more detail later. When it comes to applying the Bellman Equation over this massive table, it would take an incredible amount of time, as every single hand would have to be progressively iterated in order to optimize the decision process.

One way that has been attempted to circumvent this is to implement these Q-Tables as a neural network with inputs representing the state space and ouputs representing the different Q-Values of the function in an attempt to create a proper function approximator instead of using a table. This has however historically been known to be unstable using a nonlinear function approximator, which is precisely what a neural network attempts to do[4]. This shortcoming was then addressed by Google DeepMind in 2015, where they attempted successfully to fix this with two ideas, namely what they call experience replay, as well as a mechanism that adjusts the outputs of the network towards the target value only periodically[5]. In experience replay, the data is randomized and replayed, instead of iteratively applying this network over the environment as is standard with Q-Learning. These two ideas are quintessential to the concept of a Deep Q-Network[6].

1.2 Cribbage

The form of cribbage that I'll be implementing and creating this network for is a two player version of cribbage, using a standard 52 card deck. The way the game is played as follows:

1. Dealer shuffles the cards
2. Non-dealer cuts the cards
3. Dealer deals 6 cards to each player one at a time
4. Both players discard two cards from their hand. These four cards discarded by the players go into what's called the "crib"
5. The non-dealer then cuts the deck, and the top card is turned face up. This card counts for when the pegging phase is done towards all hands, including the crib. If the card turned up is a Jack, the dealer scores 2 points. This card is called the "start card"
6. Players take turns playing during what's often referred as the "pegging" phase
7. Once the pegging phase is done, the hands are scored in order non-dealer, then dealer, then the crib is scored for the dealer

This is repeated, alternating the dealer between both players until one player reaches 121 points or more. If at any point during scoring a player reaches 121 points or more, play is stopped and that player is declared the winner. This holds true even if it's during the pegging phase, meaning a player can lose even before their hand is scored [1][7]. One thing to note is that every card uses the value on its face for all purposes where adding is necessary except for face cards; the Ace is worth 1, and all other face cards are worth the value 10.

Because this project only deals with the scoring phase, I'll not be describing the scoring rules for the pegging phase. In my modified version of the game, everything remains the same, however there will be no pegging phase, and no start card will be used as this introduces even more unnecessary randomness that is completely irrelevant to the decision making process. During scoring the rules are as follows:

1. Any combination of cards adding up to 15 scores 2 points. For example a Jack and Five count as fifteen, but so do a Four, Five and Six. As a more specific example, a hand of Four, Five(1), Five(2), Six would contain 4 points worth of 15s. These are the combinations Four, Five(1), Six as well as the second trio of Four, Five(2), Six.

2. Any pair of cards of the same face scores 2 points. This effect can stack as well, for example a hand Four(1), Four(2), Four(3) will contain 6 points worth of pairs, for each combination of pairs possible. This effect runs the same for a four of a kind as well, scoring 12 points. To note is that two different face cards, such as a Jack and King, despite holding the same value of 10 for counting do not count as a pair.

3. Any three cards or more of consecutive rank scores one point per card in the run, such as Four-Five-Six, which would count 3 points. This effect can happen multiple times for a hand such as Four-Five-Five-Six, which contains two runs of three cards, totalling 6 points in runs, but for a hand such as Five-Six-Seven-Eight, this only counts 4 points for the run of four cards, and **not** as two runs of three (Five-Six-Seven and Six-Seven-Eight).

4. If all four cards in a player's hand are of the same suit, 4 points are scored. If the start card also matches suit, it instead scores 5 points. Note that you only score this if all the player's hand is the same suit; if the start card matches suit with three of the player's hand, this does not count as a 4 point flush.

5. The last rule is called "one for his nob". If the hand contains the Jack card that matches the suit of the start card, that player scores an extra point.

One thing to note during all of this scoring is that the same card can be used as part of several of these different scoring rules. For example, if you have a hand Seven-Eight-Eight-Nine, you can score 4 points of 15s (Seven-Eight twice), 2 points for a pair (Eight-Eight), as well as 6 points for runs (Seven-Eight-Nine twice). This hand would therefore score a total of 12 points. These same scoring rules also apply to the crib, and those points go to the player who is currently acting as the dealer.

Because of these very clear cut rules for scoring, and some of the emergent strategy behind the crib, where you want to minimize the amount of points you give away via your discards to the opponent if they're dealing, and you want to maximize the number of points split between your hand and the crib when you're dealing, this makes it a perfect candidate for reinforcement learning. As described, reinforcement learning tries to maximize reward and these rules give us a very clear definition for rewarding our agent.

2 Related Works

Previous works on the game of cribbage are incredibly limited, likely due to to it being significantly less popular than games like poker, backgammon or blackjack and possibly due to it's incredibly large and random environment.

Namely, Russel O'Connor applied temporal difference learning on a multi-layer perceptron to the game of cribbage in 2000[8]. However, their approach was applied to the game as a whole and their application of temporal difference learning was with a single hidden layer, where the weights of each layer was updated according using the policy and reward similar to the formula described above for Q-Learning. It also does not specify exactly how the network was used, it merely explains the structure of the network.

There is also a genetic algorithm that was applied to cribbage by Graham Kendall and Stephen Shaw in 2003 [9]. This paper does also only deal with the scoring phase and not the pegging phase, however they do also completely cut out suits from the game, reducing the number of possible six card hands down to 18,395. This is mostly fine seeing as suits play an incredibly small part in the game, but I wanted to be as accurate as possible to the entire discard and scoring phase as a whole. They do mention some interesting ideas, however most of these are not applicable to my situation and are specific to a genetic adversarial approach, as they did in their paper.

3 Network Structure

The input for the network is relatively straightfoward as it contains a one-hot encoding of the six cards in the player's hand consisting of 318 nodes, followed by 3 more inputs, one for the player's score, their opponents score, and a bit for determining whether the player is dealer or not. The hope with encoding the players scores and the dealer as information is to make the network consider the opponent's score with their decision. This is especially important later in the game, as you want to especially make sure you minimize the opponent's score if they're on the verge of winning and the dealer, as this is when you can best minimize their score if they're on the verge of winning. The dealer bit also helps the agent know whether it's fine to discard cards that give points to the crib, as if they are the dealer, they'll get to score those points regardless. The cards in the input will always be sorted to introduce some consistency in the way the output is represented.

The output for the network will be fifteen nodes, each corresponding to the assumed reward for a certain pair of cards being discarded, as in the $\binom{6}{2}$ possible combinations of cards that can be discarded. These outputs are where we'll apply the Bellman Equation, however because in our case there are no effects of the current decision on future rewards due to the way the game is played, I completely ignore computing the estimated reward of future states and instead use the output nodes as simply the estimated reward, or score, of that specific hand state and action combination. For the preliminary tests that I will run, this target score will be the score of the hand, however for the full game, this score will be the score of the agent, minus the score of the opponent, with the crib adding or subracting score depending on whether the agent is the dealer or not.

The hidden layers were chosen somewhat arbitrarily and were not able to be modified during tests due to time constraints, but were chosen to be four dense layers of 56, 56, 25 and 20 nodes each.

4 Benchmarks

Before moving forward, I wanted to assess some benchmarks that I could aim towards. Namely, I wanted to answer some questions about how good naive players are at the game, so I could better assess the performance of my model later on. It's interesting to create a model that can learn to play a game, but it's not very useful to evaluate it's metrics unless there's something to compare it to.

Unfortunately I don't have access to any known algorithms that can play the game of cribbage, but what I do have is two naive approaches to metrics, namely:

1. How an agent that plays completely randomly performs
2. How an agent that optimizes the hand score, ignoring the crib, can perform

This means I now have two other agents to compare the model to. From here, we can now evaluate some benchmarks, and set some goals for the model. Of course, before this, we need to know what metrics exactly we'll be evaluating. For this, I decided that I'd compare the average scores of hands between the agents. I also had the agents play matches against each other, and compared their scores. These results can be seen in Tables 1, 2 and 3.

Table 1: Player Average Scores Without Crib (1000 hands)

	Average Score
Random Player	2.682
Network Player	2.575
Naive Player	6.147

As seen in the results, a completely randomized network seems to perform about as well as a player that picks cards completely at random, which is expected. What's more important is to see that the naive player achieves an

Table 2: Player Score Differences With Crib (1000 hands)

	Random Player	Network Player	Naive Player
Random Player	■	-0.055	-3.423
Network Player	0.055	■	-3.716
Naive Player	3.423	3.716	■

Table 3: Player Match Scores (1000 games)

	Random Player	Network Player	Naive Player
Random Player	■	529-471	27-973
Network Player	471-529	■	20-980
Naive Player	973-27	980-20	■

average of about 6 points per hand without including a crib. This is a good benchmark for the network, as we now know what it looks like for the network to pick the optimal scoring hand without considering the crib.

One thing to note is that the average hand scores were done without a crib. This was just to avoid any external randomness from affecting the benchmark.

5 Experiments and Methodology

Because of the learning nature of this project, I made a few different attempts at this network structure using different learning methods to see the effects. One thing of note is that because of the incredibly stochastic nature of the game, accuracy and loss were not measured during training and loss was only used for the sake of backpropagation, as it doesn't offer any particularly useful information compared to other metrics like average hand score, the difference between the chosen hand score and the maximum score, or the ratio between the maximum score and the chosen hand score. The first two exploratory methods were not tracked during training and only had their metrics scored at the end of training by comparing the score ratio and score differences between the network and the naive approach to scoring for identical hands.

5.1 Naive Method

The first attempt I made was naive training without applying exprience replay or network saving from Mnih et al[5]. As expected, no difference was observed between the untrained network and the network after five hundred thousand training samples.

5.2 Including Exploration

The second attempt I made was also naive training, however with exploration included, where the network while training will occasionally pick a random value from it's outputs instead of choosing it's known optimum[6]. Again, this saw no observed change between the untrained network after five hundred thousand training samples.

5.3 Full Training Without Crib

As a primary naive attempt, the first legitimate training was done by implementing experience replay by generating one thousand different hands and selecting randomly thirty two hands, then replaying the training on those hands fifteen times, replacing the network weights on each replay. The network also had a random chance of choosing a random output, starting at 0.8 and decaying by multiplying by 0.95 on each batch of thirty two randomly picked experiences. One thousand batches were run. This would mean that at the start of training, the network would be highly encouraged to explore different possibilities, but as training nears the end of the one thousand batches, the network should start to become more deterministic in choosing it's known optimum. Each experience replay had it's average score compared to the naive approach as explained above. To note is that this initial training run was done without an adversary, so without a crib. Therefore this was mostly a test to ensure the network could learn the game in it's most deterministic way; where you simply choose the highest scoring possible discards. The target for the output was the score for the particular hand combination resulting from that choice.

5.4 Full Training with Adversary

The final training was done exactly as above, however the training was instead done with an adversary, where the target values for the network were the difference between the adversary's score and the player's score, with a crib scored. The dealer was chosen was at random, and the adversary was the naive player as described above that had no regard for the crib when evaluating it's discards, with a different hand selected from the same pool of one thousand random hands. Again, the network was tested against a naive player who was given the same hand and had the score of the agent's chosen hand compared to the naive option's score with the crib included or excluded depending on whether they were randomly chosen as the dealer. Training results were not split into "dealer" and "not dealer" results, because the model being trained would be expected to have better scores over time irrespective of whether they were the dealer compared to the naive player over time as long as they're learning.

6 Results

To determine results of the two main training methodologies I supply three different metrics:

1. The average score of hands during training compared to the naive optimum

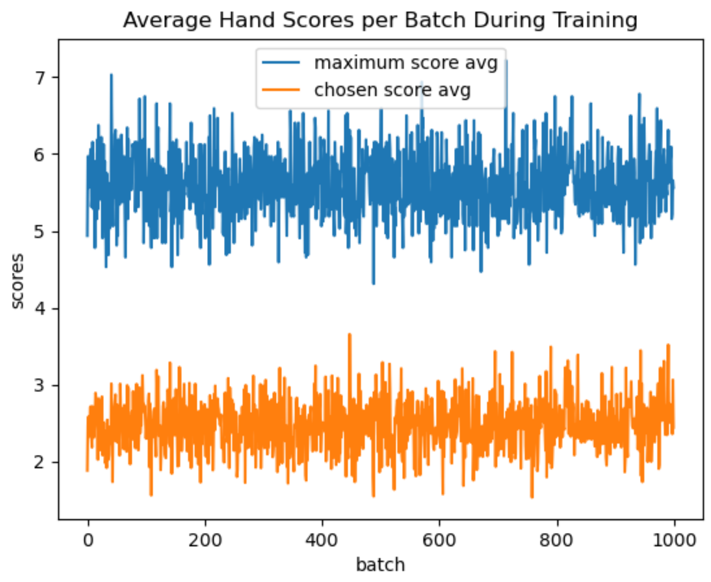
2. The results when playing against a naive player in a full modified game (without the pegging phase)
3. The optimized score of each hand post training compared to the naive optimum and a baseline random network

For the third metric, because one training phase was without a crib and the other was trained including a crib, I decided again to select a random hand from the one thousand training samples, giving it to a naive player and testing the network and tester (another naive player, with the same hand as the network) and comparing their scores. This was only for the results for the network trained with a crib.

6.1 Full Training Without Crib

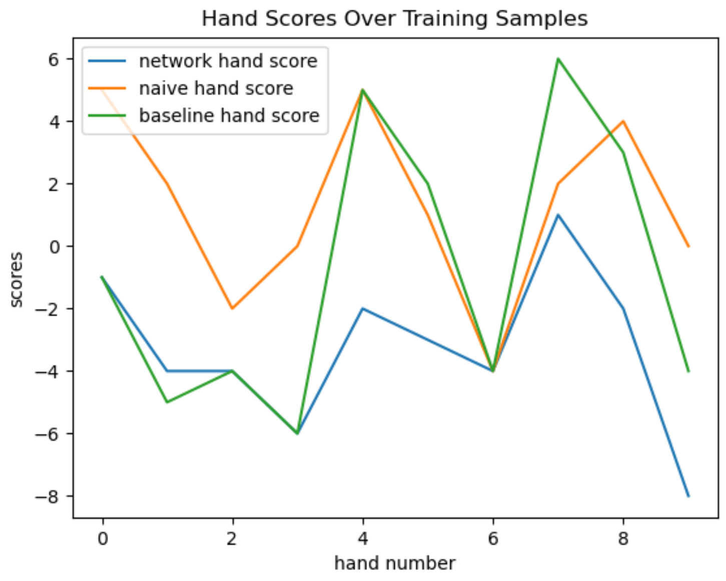
Unfortunately this did not achieve any visibly positive results. Over the one thousand training batches, the network appears to have not learned anything about the game as over the one thousand training batches, there was no significant change in the average score by the network, as shown in Figure 1.

Figure 1: Hand Scores During Training Without Crib



When playing against a naive player in a one thousand game match, the results were 942 wins for the naive agent and 58 for the trained network, which is very similar to when it was untrained. Because of the relatively small batch of games played and lack of computational resources to test for any larger batch sizes, it's unclear if this small improvement is due to learning or randomness.

Figure 2: Hand Scores over 10 games chosen from training sample and average scores over all 1000 samples



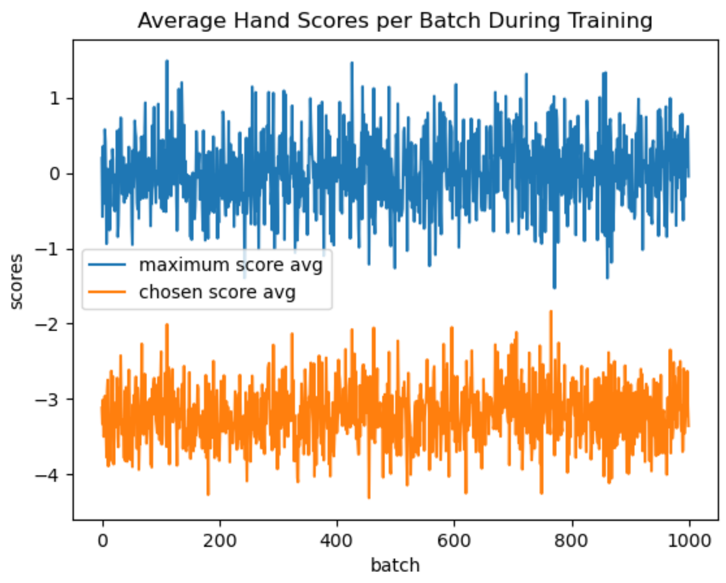
- Trained Network Average Score: 2.284
- Naive Player Average Score: 5.638
- Untrained Network Average Score: 2.457

When testing the network against a baseline randomly weighted network and against a naive approach over a randomly chosen batch of ten hands from the original batch of one thousand used for training, there seems to have been no improvement. When evaluating the average hand scores of each of these over the entire one thousand hand batch, there also appears to have been no improvement over the randomly assigned network and the trained network. In fact, the network seemed to have gotten worse compared to a randomly weighted network on it's own training set, as seen in Figure 2.

6.2 Full Training With Crib

Once again, this did not achieve any visibly positive results. Over the one thousand training batches, the network appears to have not learned anything about the game as over the one thousand training batches, there was no significant change in the average score by the network even when including a crib as seen in Figure 3.

Figure 3: Hand Scores During Training With Crib



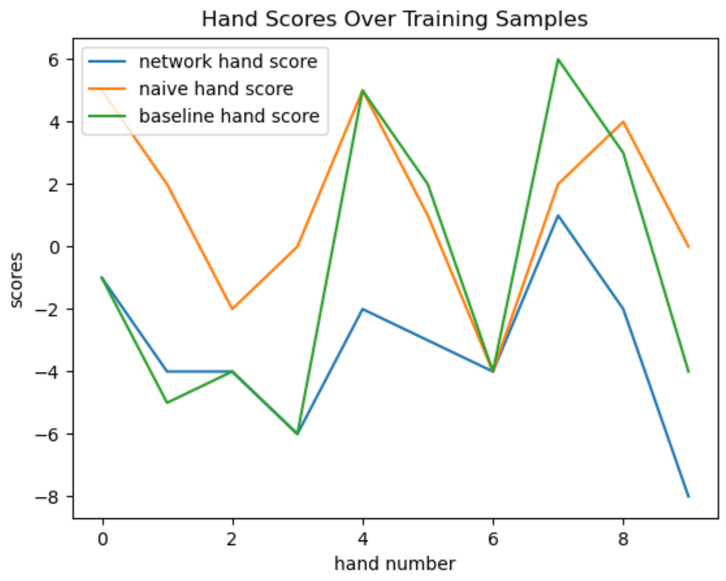
When playing against a naive player in a one thousand game match, the results were 970 wins for the naive agent and 30 for the trained network, which is very similar to when it was untrained. Because of the relatively small batch of games played and lack of computational resources to test for any larger batch sizes, it seems unclear whether the network did get worse or if this is simply due to the random nature of the game.

However, when testing the network against a baseline randomly weighted network and a naive approach over a randomly chosen batch of ten hands from the original batch of one thousand used for training, the network seems to be underperforming compared to a randomly weighted network on it's own training set. This is even more apparent when looking at the average scores over the entire batch of one thousand hands, where the trained network achieves an average score of -3.3, compared to 1.3 for the naive approach and -0.8 for the untrained network as seen in Figure 4.

7 Thoughts and Future Work

Because of the relatively simple rules of the game that can be learned by an adult human with relative ease, I originally thought that not much training would be needed to achieve a result that was at least better than completely random play. However, it does appear that significantly more training and perhaps a more rigorous training method would be required. I suspect that the

Figure 4: Hand Scores over 10 games chosen from training sample and average scores over all 1000 samples



- Trained Network Average Score: -3.3
- Naive Player Average Score: 1.3
- Untrained Network Average Score: -0.8

extremely random nature of the game of cribbage caused issues during training, as there are going to be inconsistencies in output even with the same input, due to the opponent's hand being hidden. Sometimes an identical hand can give completely different scores, even when both are dealt as the dealer. This phenomenon could very well explain why the network got worse over training, it likely diverged from the intended solution as explained in [4] where nonlinear function approximators for Q-Learning can diverge.

There could also be issues with the actual network structure itself. The network structure is completely static and one dimensional, but perhaps treating the input as two dimensional with convolution and dropout layers could be a better approach. Dropout layers especially may be of use because of the randomness introduced by the game[10]. Because of the network's tendency to overfit and the fact that the random nature of the game makes this incredibly detrimental, dropout layers could help to mitigate this effect.

Maybe even some different approaches to the outputs could be used. For example, an approach using an encoding of the entire deck with rewards per node for being chosen and selecting the top two that are in the input hand could be used. This would of course require a different reward mechanism still tied to the game scores, but perhaps more complex. A similar method where every output node would also have a reward tied to it regardless of choice could also help with improving learning speed, if such an approach could be created that seemed sensible.

An issue with the current model is that only one node's target can be modified per training session, because the reward for other nodes are unknown until they've been chosen. Perhaps the training could be modified to change the target of all of the output nodes at the same time, once the cards in the crib are known in order to speed up training. This would of course mean the network would no longer be playing the game to learn, but instead would be conceived more as the agent getting a hand and viewing the results of every possible choice they could make during training. Again, because of the random nature of the game, it's unclear whether this could even function correctly, as the same input could have multiple different "best" outputs depending on the opponent's hand, which is hidden.

There could also be room for improvement on the current network structure by simply tuning hyperparameters, using different loss functions, trying different exploration incentives, and so on. In the end, although this project did not achieve great performance, there is most definitely room for improvement not only in tuning this methodology's hyperparameters, but also in testing out other network structures.

References

[1] B. Cards, "Learn to play cribbage," 2022.

[2] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*, pp. 3–42. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[3] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[4] B. Van Roy *et al.*, "An analysis of temporal-difference learning with function approximation," *Automatic Control, IEEE Transactions on*, vol. 42, no. 5, pp. 674–690, 1997.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[7] J. Paxton, "Six card cribbage," 1995.

[8] R. O'Connor, "Temporal difference reinforcement learning applied to cribbage," 2000.

[9] G. Kendall and S. Shaw, "Investigation of an adaptive cribbage player," in *Computers and Games* (J. Schaeffer, M. Müller, and Y. Björnsson, eds.), (Berlin, Heidelberg), pp. 29–41, Springer Berlin Heidelberg, 2003.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, jan 2014.