YALE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Using Deep Q-Learning to Compare Strategy Ladders of Yahtzee

Philip Vasseur

Advised by James Glenn

December 12, 2019

CONTENTS

***Abstract*—"Bots" playing games is not a new concept, likely going back to the first video games. However, there has been a new wave recently using machine learning to learn to play games at a near optimal level - essentially using neural networks to "solve" games. Depending on the game, this can be relatively straight forward using supervised learning. However, this requires having data for optimal play, which is often not possible due to the sheer complexity of many games. For example, solitaire Yahtzee has this data available, but two player Yahtzee does not due to the massive state space. A recent trend in response to this started with Google Deep Mind in 2013, who used Deep Reinforcement Learning to play various Atari games [4].**

This project will apply Deep Reinforcement Learning (specifically Deep Q-Learning) and measure how an agent learns to play Yahtzee in the form of a strategy ladder. A strategy ladder is a way of looking at how the performance of an AI varies with the computational resources it uses. Different sets of rules changes how the the AI learns which varies the strategy ladder itself. This project will vary the upper bonus threshold and then attempt to measure how "good" the various strategy ladders are - in essence attempting to find the set of rules which creates the "best" version of Yahtzee. We assume/expect that there is some correlation between strategy ladders for AI and strategy ladders for human, meaning that a game with a "good" strategy ladder for an AI indicates that game is interesting and challenging for humans.

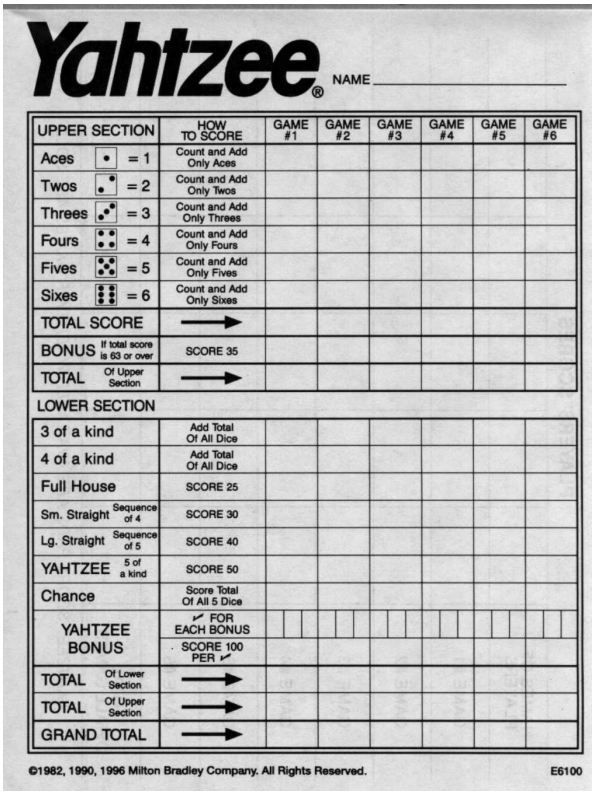
1 Introduction

This project aims to use Deep Q-learning (DQL) to measure the strategy ladders of various versions of single-player Yahtzee, a dice game invented in the 1940s, as well as implementing DQL self-play for two player Yahtzee. A strategy ladder is a way of looking at how the performance of an AI varies with the computational resources it uses - in this project computational resources will be simply defined as training time for the reinforcement learning. This project will compare the various strategy ladders to determine which set of rules gives the "best" strategy ladder, where a "good" strategy ladder would be one that is not too vertical or too horizontal, and rather gives a good combination of learning over time.

This project specifically compares the strategy ladders of solitaire Yahtzee when varying the upper bonus threshold from 53 to 75.

2 Yahtzee Gameplay

Yahtzee is a board game originating in the 1940s which has an an incredibly large random element, though is still largely dependant on skill. The game consists of 13 rounds where the player is given some amount of points based on the combination of dice they have and category they choose. In each of the 13 rounds, the player starts off by rolling 5 dice. The first two "moves" in each round is to pick some subroll of the dice to keep and then reroll the rest. The third and final "move" in each round is for the player to pick one of the 13 categories in Yahtzee to score their moves on. At the end of the 13 rounds, the scores from each category are summed up to give the player's final score. In two player Yahtzee, the player with the highest score is the winner. There is clearly an incredibly large chance element in Yahtzee consisting of rolling the dice. Even if the player chooses optimally at each state, they could get a subpar score.



Yahtzee® NAME _____

UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces = 1	Count and Add Only Aces						
Twos = 2	Count and Add Only Twos						
Threes = 3	Count and Add Only Threes						
Fours = 4	Count and Add Only Fours						
Fives = 5	Count and Add Only Fives						
Sixes = 6	Count and Add Only Sixes						
TOTAL SCORE	→						
BONUS If total score is 63 or over	SCORE 35						
TOTAL Of Upper Section	→						
LOWER SECTION							
3 of a kind	Add Total Of All Dice						
4 of a kind	Add Total Of All Dice						
Full House	SCORE 25						
Sm. Straight Sequence of 4	SCORE 30						
Lg. Straight Sequence of 5	SCORE 40						
YAHTZEE 5 of a kind	SCORE 50						
Chance	Score Total Of All 5 Dice						
YAHTZEE BONUS	✓ FOR EACH BONUS						
	SCORE 100 PER ✓						
TOTAL Of Lower Section	→						
TOTAL Of Upper Section	→						
GRAND TOTAL	→						

©1982, 1990, 1996 Milton Bradley Company. All Rights Reserved. E6100

Fig. 1. A Yahtzee scoresheet [1]

2.1 Categories

The number of points awarded depends on the category chosen and the current dice rolled. Each of the 13 categories can only be used once in the game (though the Yahtzee category has special rules, which will be explained layer on). The categories in Yahtzee are split between the upper and lower sections.

2.1.1 Upper: The first six categories are in the upper section, which are just simply aces, two, threes, fours, fives, and sixes. The number of points the player earns from choosing an upper section category is just the sum of the die faces with that number. If, for example, one chooses sixes without any six faces in their current lineup of dice, they would be given zero points and no longer be able to use the sixes category.

2.1.2 Lower: The latter seven categories are in the lower section. These are three of a kind, four of a kind, full house, small straight, large straight, chance, and Yahtzee. The important thing to note is that the points are only awarded if the category is satisfied. So for three/four of a kind the player gets points equivalent to the sum of all the dice if they have three or four of a kind, respectively. This is the same for all of the rest - full house gives 25 points, small/large straights give 30 and 40 points flat respectively, and Yahtzee gives 50 points flat (again assuming your 5 dice satisfy the category conditions). The unique category in this respect is chance, which gives points equivalent to the sum of all the dice regardless of which dice.

2.1.3 Bonuses: There are two main bonuses in Yahtzee. The first is the upper section bonus, which awards 35 points to the player if the player scores hits a threshold amount of points in the upper section (63 in the official rules, which is achievable with three die in each of the categories). The second is the Yahtzee bonus. If the Yahtzee category has already been successfully filled, then a subsequent Yahtzee used to fill any other category also gives an additional 100 points. The final rule is the joker rule. This project used the free choice joker rule, which allows a subsequent Yahtzee to satisfy full-house and small/large straight conditions if the Yahtzee bonus is satisfied and the corresponding upper section category is filled.

The upper section bonus is a heavily weighted bonus in optimal Yahtzee play (see optimal statistics image below) due to the high amount of points it can give and the relative ease to achieve it.While the Yahtzee bonus gives many more points, it is much more difficult to obtain. In this project, we will be adjusting the upper bonus threshold to values between 53 and 75 and comparing the measured strategy ladders.

2.2 Optimal Play

Solitaire (single-player) Yahtzee has a small enough state space that it can be effectively solved and played

Category	Expectation	Variance	Std. dev.	% 0 scores
Aces	1.88	1.48	1.22	10.84
Twos	5.28	3.99	2.00	1.80
Threes	8.57	7.36	2.71	0.95
Fours	12.16	10.80	3.29	0.60
Fives	15.69	14.83	3.85	0.50
Sixes	19.19	21.56	4.64	0.53
Upper Section Bonus	23.84	266.04	16.31	31.88
Three of a Kind	21.66	31.56	5.62	3.26
Four of a Kind	13.10	122.63	11.07	36.34
Full House	22.59	54.41	7.38	9.63
Small Straight	29.46	15.87	3.98	1.80
Large Straight	32.71	238.42	15.44	18.22
Yahtzee	16.87	558.88	23.64	66.26
Chance	22.01	6.45	2.54	0.00
Extra Yahtzee Bonus	9.58	1161.19	34.08	91.76
GRAND TOTAL	254.59	3553.52	59.61	0.00
Yahtzees Rolled	0.46	0.47	0.69	63.24
Jokers Applied	0.04	0.04	0.19	96.30

Fig. 2. Statistics on the points scored in each category by the optimal Yahtzee player [7]

optimally. The optimal average score in solitaire Yahtzee including the bonuses and jokers mentioned above is approximately 254.59 with a standard deviation of 59.61 [2]. The variance is quite high as mentioned earlier due to the inherent randomness in rolling the dice and simply the mechanics of the game. The worst possible score that can be earned is 5, while the best is 1575. While both of these are extremely unlikely (and the former you would seemingly have to try to do terribly), it wouldn't be unusual for a skilled player to get scores of less than 150 when down on their luck.

A simple example shows this: the trivial random Yahtzee player (who chooses a valid category at the start of each turn) averages around 191 points with a standard deviation of 40.37 points. Assuming normal distribution, let $R \sim N(191.16, 40.37)$ and $O \sim N(254.59, 59.61)$. We then have $P(R > O) = P(R - O > 0) = P(D > 0) \approx 0.189$, where $D \sim N(-63.43, 71.99)$. This straight forward calculation shows this extremely naive bot gets a higher score than the optimal solitaire Yahtzee player slightly under 20% of the time.

2.3 Single vs Two Player Yahtzee

While this project mainly focuses on single player, Deep Q-Learning with self-play was implemented for two player Yahtzee. While the difference is subtle, the optimal strategy for single player is different than the optimal strategy for two player Yahtzee. This difference stems from the fact that the goal of two player Yahtzee is not to get the most points possible, but to simply get more points than your opponent. These are obviously correlated, but not perfectly. Moments of distinction can easily be thought of, such as when a player is down by slightly under 100 points on their final turn. The Yahtzee bonus might be their only chance at winning and thus is obviously the best move to go for. However, going for a Yahtzee bonus probably has a lower expected total score (depending on the current roll) because of how rare it is to get a Yahtzee. In this case, the optimal single player strategy and the optimal two player strategy would differ.

The other reason why two player is interesting to look at is because of how large the state space becomes. While single player Yahtzee is solvable, two player Yahtzee nearly squares the state space, making it much too large to solve with other typical methods. There is no labelled data possible to use supervised learning on, but an option like Deep Reinforcement Learning could work quite well theoretically.

3 Reinforcement Learning

3.1 Deep Q-Network

This project uses Deep Reinforcement Learning (a Deep Q-Network specifically) to allow the agent to learn as it plays. Deep Reinforcement learning works similar to normal reinforcement learning, but rather than using a State-Action table or another approximator for Q-value, it uses a neural network. The neural

network takes in an input the dimensions of the state, and produces an output the dimensions of the action space - with the output vector representing the Q-values for each action. From regular reinforcement learning, the Bellman equation is

$$Q(s_t, a_t) \mathrel{+}= \alpha\big(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\big)$$

Where α is the learning rate and γ is the decay rate. The Q function is iteratively updated, in an attempt to have it converge to a point where $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ (the target) equals $Q(s_t, a_t)$ (the current value of the network). To achieve this in the neural network world, the loss function is simply a masked mean-squared error of the Q-value outputs and the target. All output nodes except a_t are masked to 0, and gradient descent is then performed to minimize $r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$.

3.2 Double and Dueling DQN

A problem with vanilla DQNs is that they often overestimate action values due to the inherent maximization bias. Q-Learning at its core is learning to estimate from its own estimates, and therefore it is somewhat clear how this along with the fact that the target Q-value is maximized can lead to substantially overestimated Q-values and thus substantially worse performances. The solution to this is Double DQNs [6]. This modification creates an entirely separate target network Q' separate from Q. Q is still used to evaluate the Q-values of the actions, but rather than using a target of $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ we use $r_t + \gamma Q\big(s_{t+1}, \text{argmax}_{a'} Q'(s_{t+1}, a')\big)$, using the target network Q' to select the action and updating $Q' = Q$ every so often. This then disentangles the overestimate bias.

Another improvement is the Dueling DQN [8]. This seemed to have less of an affect in practice, but was still useful. The Dueling DQN deals with the issue that not all states are valuable and it is often "wasteful" in a sense to learn the value of an action at every state, if the state itself is invaluable. For example, there might be situations in Yahtzee where which action is taken does not truly affect things, and the state itself is the important factor. The Dueling DQN solves this by separating the $Q(s, a)$ into $A(s, a) + V(s)$ - a function to calculate the advantage of selecting a specific action plus the value of the state itself. The output though must be the same, so they are combined back in an identifiable way, to still output $Q(s, a)$ and train in the same manner (though now with the networks split earlier on). This may seem unimpactful, but in fact allows the network to learn the value of a state without having to learn which actions are or are not valuable.

4 Implementation Details

4.1 Yahtzee Architecture

The implementation of single and two player Yahtzee was largely an extension of previously written work by my advisor, Professor James Glenn (for which I am extremely grateful). The state of a game of Yahtzee simply consists of a scoresheet, a roll, and a number of rerolls left, which were already written. To simplify the code, these things were combined into a class `YahtzeeGame`, which upon initialization allows for a single player game of Yahtzee to be played by repeatedly calling `game.make_move`.

In order to simplify debugging, this was then wrapped in an Open AI Gym environment `YahtzeeEnv`. This extends `gym.Env`, which is a common API for reinforcement learning problems. The API consists of `init` to get to the starting state, `reset` to get back to the starting state and return this state, and most importantly `step` which takes in an action and returns the next state, reward, and whether the game is done. This allows Yahtzee to be easily swapped out when reinforcement learning

algorithm with any other simple Open AI Gym environment. The `YahtzeeEnv step` function takes in a meta action (essentially representing which category the player is aiming for - one to six, n-kind, full house, straights, or chance) instead of a roll or a category, and uses a conversion utility to change the meta action into an action which can be passed into `YahtzeeGame`. The environment wrapper also took in a number of players, allowing for two player Yahtzee, which initialized multiple Yahtzee games and kept track of turns. As the environment was also built for reinforcement learning problems, it had the functionality to calculate and return the state of the game in terms of an input vector, which varied in size depending on how many players were in the game.

4.2 Training Architecture

The main workhorse of the project is a class `Trainer`, which is given an environment (such as Yahtzee), an agent (such as the Huskarl DQN agent discussed below), and a number of players - after which one can simply call `trainer.train(num_steps)`. The trainer iterates `num_steps` times, getting an action from the agent based on the current state, pushing the State-Action-Reward-NextState (SARS) tuple into the agents memory, and then having the agent train by sampling from its memory. While iterating it also checks when the current game finishes, where it then resets the environment and calls the `update` function. This simply keeps track of the progress of the agent as it learns, for example validating it's current total score in single player every 1000 games of training to keep track of the current best model.

The lowest level parts of the reinforcement learning for single and two player uses a forked version of a deep reinforcement learning library `Huskarl`. The vanilla version of Huskarl provides an implementation of a Deep Q-Network (DQN) which then rely only on a few basic calls to use - namely `act`, `push`, and `train`, to calculate the action from the current state, to add to the agents memory, and to train based off the memory respectively. While this seemed fantastic, in practice the package was quite problematic, which was why for this project we forked and made some small adjustments. Other than simply tuning various parameters, the implementation of the memory was simplified (as the Huskarl implementation of prioritized experience replay was quickly became the bottleneck of training), the save functionality was rewritten (as it was poorly written, used outdated tensorflow functionality, and didn't save the full state of the network), and the load function added (as it somehow did not exist beforehand). The original version of Huskarl worked somewhat well - out of the box it was able to have the single player agent learn from averaging 70 points to averaging around 130 - but with tuning and massively speeding up the runtime in a matter of two hours or so, the single player agent was able to learn from averaging 70 to averaging 232.

4.3 Self-Play Implementation

Many aspects of the codebase did not need to be changed in order to accommodate training for two player Yahtzee. As the game environment was built with multiplayer in mind, the main adjustments were with the `Trainer` class.

The first step was to create another opponent copy of the agent. This opponent would be updated periodically as the main agent improved. A threshold used in Alpha Go Zero [5] is if the current agent wins 55% of games, which is what was used as inspiration for many aspects of the self-play implementation. The two agents went back and forth every turn acting on the environment, though instead of pushing each SARS tuple into the main agent's memory, it was

stored in a temporary buffer. This is because there is no current reward, and rather also inspired by AlphaGo Zero, the reward is simply whether the agent has won the game or not. Finally, after the game finishes, the buffer of SARS is updated with the proper rewards, which are then pushed into the main agents memory to be trained on later.

5 Results

5.1 Strategy Ladder Comparison

In order to reduce the inherent variance of Yahtzee, the strategy ladder was measured by taking the average score of the current model, but rather the average score of the best model so far. While generally training improved the model, slight changes to the model could result in dramatically worse scores, which made the results much less interpretable. Further, rather than measuring the raw scores themselves, the fraction of the optimal score for the respective set of rules was recorded. This was done to normalize the strategy ladders, as changing the upper bonus threshold would clearly affect the optimal score. Using code previously written by my advisor, the optimal values for Yahtzee with an upper bonus from 53 to 75 were calculated.

In the end, strategy ladders were obtained by testing the average score of the model over 1000 games every 1000 games of training for 160,000 games. If the model scored better than the current best model, than the best model was replaced with the current (see Figure 3).

5.1.1 Quantifiers: In order to quantitatively compare the strategy ladders, I'll be using three different measurements of "goodness". As mentioned earlier, a "good" strategy ladder is one that does not improve too quickly, but also not too slowly. So we are looking for one which takes many small steps over a few big steps, but also one which has a high horizontal asymptote (which represents how much it was able to learn in 160,000 games - the final performance "FP"). The first measure of "goodness" is one described in *Depth in Strategic Games* [3], which picks a constant step size for x and y and counts how many times the target y (which is the previous y + the y step size) is reached when taking the x step size. This effectively measures how many decently sizeable steps are taken. We will call this "P1". The y step size was chosen arbitrarily as $\frac{1}{12}$ of what was left to learn - when calculating the metric with sizes from $\frac{1}{10}$ to $\frac{1}{30}$ and averaging, the top three thresholds were still 55, 57, and 53 (averaging 18.1, 17.7, and 17.65 for P1 respectively). The second measure of "goodness" is simply `np.sum(np.log(1 + np.diff(data)))`. This is essentially just summing up how much is learned each step, but taking the log makes it so taking multiple small steps is weighted much heavier than taking one large step. We will call this "P2". The third measure of "goodness" is `(P1/np.max(P1) + P2/np.max(P2))/2`, which is simply a scaled combination of P1 and P2. We will call this "P3". Calculating these metrics for each set of rules gives the table in Figure 4.

Threshold	P1	P2	P3	FP
57.0	14.0	0.5992	0.9929	0.8956
55.0	14.0	0.5979	0.9918	0.8925
53.0	13.0	0.6079	0.9643	0.9002
63.0	13.0	0.6017	0.9592	0.9121
65.0	13.0	0.577	0.9389	0.8908
67.0	13.0	0.5591	0.9242	0.8721
59.0	12.0	0.5806	0.9062	0.882
73.0	12.0	0.5741	0.9008	0.9011
71.0	12.0	0.5685	0.8962	0.8931
61.0	11.0	0.577	0.8675	0.8829
75.0	11.0	0.56	0.8535	0.888
69.0	11.0	0.5497	0.845	0.8681

Fig. 4. Performance metrics for each of the thresholds

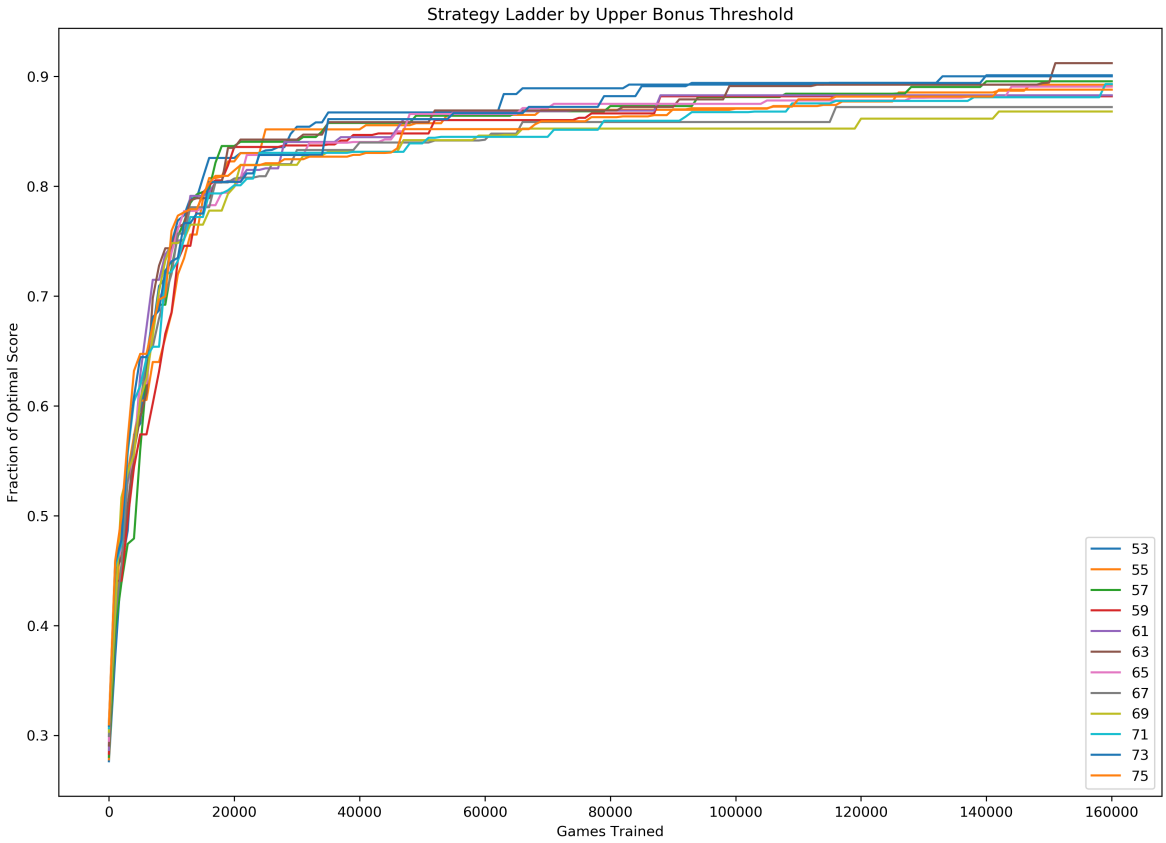


Fig. 3. Plot of the normalized strategy ladders for each variation on the official set of rules

Plotting these three metrics versus the final performance gives the scatter plots in Figure 5.

5.1.2 Analysis: The first thing to note is that the learning parameters were not tuned for each threshold value. Rather, they were tuned for a threshold value of 63 and then the same parameters were reused for the rest of the thresholds. This was largely because tuning the parameters was very time consuming and as such, tuning the parameters for 12 different models would have become overly tedious. This makes it so the final performance result of threshold 63 must be taken with a grain of salt when comparing.

P1 and P2 show relatively similar results in which rules create the "best" strategy ladder - they have a correlation coefficient of 0.6985. Thresholds of 53 to 57 seem to have the highest performance measurements, and relatively high final performance (with only 63 easily beating them - potentially due to the reason listed above).

This is also supported by the cumulative metric P3. In the plot of P3 we also interestingly see that the very high thresholds seem to provide poor strategy ladders. When qualitatively analyzing the category statistics from the models, this is supported - after a threshold of 67 the emphasis on the upper bonus threshold becomes dramatically lower. With thresholds of 71, 73, and 75 the average upper bonus points were 0.770, 0.245, and 0.070 respectively, while with the official threshold of 63 our model averaged 15.753 upper bonus points. In the variations of the game with a high threshold, the upper bonus becomes much less emphasized, as one would expect. This seemingly makes the game simpler and less "interesting".

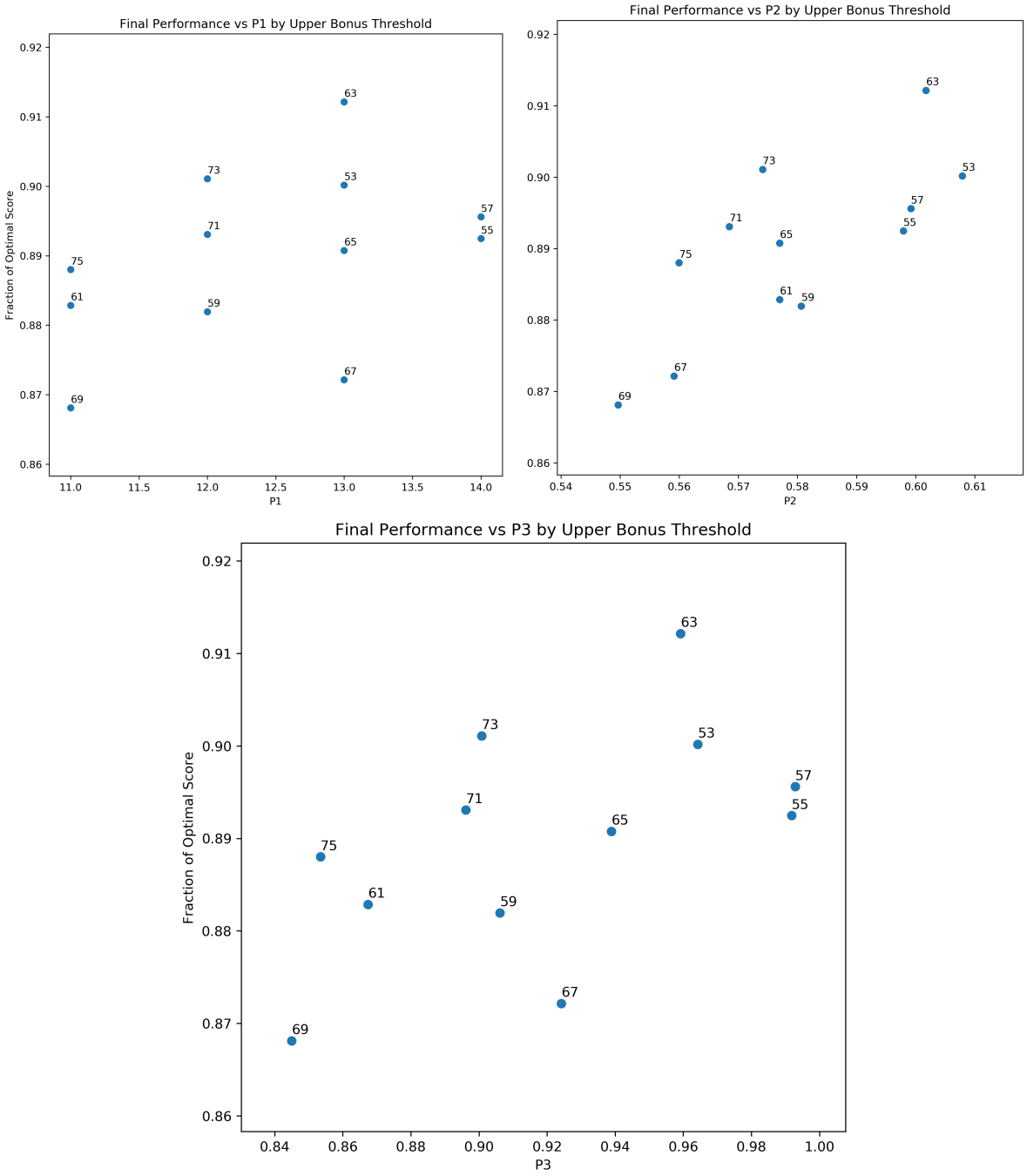


Fig. 5. Results scatter plots of P1, P2, and P3 versus the final performance of the model

Overall, it seems like decreasing the upper bonus threshold in Yahtzee could lead to a "better" strategy ladder - especially when taking into account that if thresholds 53-57 had their parameters tuned, they would have had a higher final performance.

5.2 Self-Play

Learning and comparisons in solitaire Yahtzee worked well, though this was not the case with two player Yahtzee. Rather than using a trained solitaire Yahtzee network as a place to jump start from, initially a new neural network was used in order to test the waters and see if any learning would occur. While there was a quick jump in the first 1000 games or so from averaging 70 points to averaging 100 points, afterwards no consistent learning occurred. Despite trying various parameters, adjusting what was included in the two player state, including and excluding the non-main agents SARS from the memory buffer, and numerous other various, there seemed to be no substantial change. After jumping up to 100, the network often went back down to 70 or even lower. For the rest of the 80,000 games trained, the main agent rarely ever won more than 55% of the test games, showing no progress whatsoever.

Slower learning was somewhat expected, as the rewards as much less nuanced - being 1 or -1 depending on whether the agent has won or lost. The dimensions of the input being twice the size also would likely have a similar effect. Due to these two things, the difficult of learning likely increased dramatically and in order to obtain any good results a much larger and (or) deeper neural network would be necessary - this would also require much more powerful machines.

6 Future Work

6.1 Solitaire Yahtzee

While this project was only a semester long, with more time it would have been interesting to look at more variations on the solitaire Yahtzee rules. Comparing a large number of variations and combinations of those variations to attempt to come up with a truly "best" version of Yahtzee would be a challenging, but incredibly interesting project. To do this truly properly, one would also need to spend much more time on each variation actually tuning the model in order to get a better idea of top performance. The tuning in this project was done "by hand", though a pipeline to automatically test a large number of parameters for some amount of training would be incredibly useful and help make the above possible.

6.2 Two Player Yahtzee

With more powerful machines and more time to train, building out an optimal two player Yahtzee agent would be an obvious follow up to the work attempted in this project. The question of how different/how much better an optimal two player Yahtzee agent is compared to an optimal solitaire agent was left unanswered but would be an interesting question to look further into. Along with the research of solitaire Yahtzee, once the above is completed it would be possible to also compare the strategy ladders of two player Yahtzee - looking at how often they win against some constant agent as they learn.

7 Conclusion

This project used Deep Q-Learning to measure and compare the strategy ladder of solitaire Yahtzee when varying the upper bonus threshold from 53 to 75. Quantifying the strategy ladders via two performance metrics and a cumulative metric of the two showed that the official rules of Yahtzee might not be the "best" rules, and that lowering the upper bonus threshold to somewhere in the range of 53 to 57 could result in a more "interesting" variation of Yahtzee.

8 Acknowledgements

I would like to thank Professor James Glenn for being my advisor throughout this semester. Professor Glenn helped me countless times through a large variety of problems, giving extremely valuable insight and feedback as I progressed through my work (as well as an implementation of Yahtzee to start this project off with) - thank you! I would also like to thank my friends (especially the non-STEM ones) who listened to me ramble about my project for the past few months. Finally, I would like to thank my brothers and parents.

References

[1] Math Explorers' Club Cornell. Yahtzee: Rules and scoring. http://pi.math.cornell.edu/~mec/2006-2007/Probability/Yahtzee.htm. Accessed: 2019-12-07.

[2] James Glenn. An optimal strategy for yahtzee. *Loyola College Technical Report*, CS-TR-0002, May 2006.

[3] Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andrew Nealen, and Julian Togelius. Depth in strategic games. In *WS-17-01*, volume WS-17-01 - WS-17-15, pages 967–974. AI Access Foundation, 1 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[6] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[7] Tom Verhoeff. Optimal solitaire yahtzee player. http://www-set.win.tue.nl/~wstomv/misc/yahtzee/trivia.html, 1999. Accessed: 2019-12-07.

[8] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.