Training a Bridge Bidding Agent using Minimal Feature Engineering and Deep Reinforcement Learning

by

Ulaş Sert

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

September 17, 2020

Training a Bridge Bidding Agent using Minimal Feature Engineering and Deep Reinforcement Learning

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Ulaş Sert

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Deniz Yuret (Advisor)

_____

Assist. Prof. Barış Akgün

_____

Assoc. Prof. Nazım Kemal Üre

Date: _____

ABSTRACT

Training a Bridge Bidding Agent using Minimal Feature Engineering and Deep Reinforcement Learning

Ulaş Sert

Master of Science in Computer Science and Engineering

September 17, 2020

The game of contract bridge, or just bridge, is a four-player imperfect information card game where two partnerships of two players compete against each other. It has two main phases: bidding and play. While the computer players have approached human-level performance two decades ago in the playing phase, bidding is still a very challenging problem. This makes bridge one of the last popular games where computers still lag behind the expert human-level performance. During bidding, players only know their own cards while participating in a public auction. Performing well in this phase requires the players to figure out how to communicate with their partners using the limited vocabulary of bids to decide on a joint contract. This communication is restricted by the strict ordering of legal bids and can be negatively interfered by bids made by the opponent partnership. In this thesis, we experiment with several novel architectures with minimal feature engineering and evaluate them by using supervised training over a data set of expert-level human games. After that, we further study different forms of deep reinforcement learning to refine the resulting model by simulated gameplay. Lastly, we propose an oracle evaluation metric that can measure the quality of any bidding sequence with respect to the game-theoretical optimum.

ÖZETÇE

Minimal Öznitelik Mühendisliği ve Derin Pekiştirmeli Öğrenme kullanarak Briç Deklarasyon Oyuncusu Eğitimi

Ulaş Sert

Bilgisayar Mühendisliği, Yüksek Lisans

17 Eylül 2020

Kontrakt briç, veya sadece briç, her biri iki oyuncudan oluşan iki ortaklığının birbiriyle rekabet ettiği kusurlu bilgili bir kart oyunudur. Oyun iki aşamadan oluşur: deklarasyon ve kart oyunu. Bilgisayar oyuncuları yirmi yıl önce oyun aşamasında insan seviyesinde performanslara ulaşmış olsalar da, deklarasyon aşaması hala zorlayıcı bir problem. Bu aşamada oyuncular, bir açık arttırmaya katılırken yalnızca kendi kartlarının bilgisine sahipler. Burada iyi performans göstermek, oyuncuların aksiyonları ile nasıl iletişim kuracaklarını belirleyip ortak bir kontrakta karar verebilmeyi gerektirir. Bu iletişim tekliflerin sürekli artma zorunluluğu ile limitli olup, rakip ortaklık da kendi teklifleri ile iletişimin arasına karışabilmektedirler. Bu tezde, minimum öznitelik mühendisliğine sahip birkaç yeni mimari ile deney yapıyoruz ve bunları uzman düzeyindeki insan oyunlarından oluşan bir veri kümesi ile gözetimli eğitim kullanarak değerlendiriyoruz. Bundan sonra, elde edilen modeli gerçek oyun oynanışları ile iyileştirmek için farklı derin pekiştirmeli öğrenme biçimlerini inceliyoruz. Son olarak, rakipli briç oyuncuları için rekabet etmek için ayrı bir oyuncu gerektirmeyen bir değerlendirme ölçütü öneriyoruz.

ACKNOWLEDGMENTS

I would first like to thank my advisor, Professor Deniz Yuret, whose expertise and feedback was invaluable in structuring this thesis. His guidance and persistence were also vital whenever my personal drive was ebbed away. Without his help, this thesis would not have been realized.

I am grateful to Assistant Professor Barış Akgün and Associate Professor Nazım Kemal Üre for participating in my thesis committee.

I wish to show my gratitude to my colleagues at the Artificial Intelligence Laboratory, especially Cemil Cengiz and Berkay Furkan Önder, who provided countless hours of discussions that sharpened my mind as well as great distractions when I needed to get away from research.

I acknowledge the financial support for my master's studies provided by Koç University.

Also, I would like to thank my family, for their unwavering love and support, and all the people whose assistance was a milestone in the completion of this project.

TABLE OF CONTENTS

List of Tables viii

List of Figures ix

Abbreviations xiii

Chapter 1: Introduction 1

Chapter 2: Contract Bridge 4

Chapter 3: Methods 8

3.1 Game Representation 9

3.2 Supervised Learning 9

3.2.1 Architecture Experiments 10

3.2.2 Further Training 13

3.3 Reinforcement Learning 14

3.3.1 Policy Gradient Algorithms 16

3.3.2 Game Format 19

3.3.3 Training Regimens 19

3.3.4 Exploration using Entropy Loss 20

3.3.5 All-Pass Reward 21

3.4 Evaluation Metrics 23

3.4.1 Relative to Supervised Learning Agent 23

3.4.2 Relative to Other Agents 23

3.4.3 Absolute Difference to Oracle 24

Chapter 4: Results **25**

4.1 Supervised Learning 25

4.2 Reinforcement Learning from Scratch 27

4.2.1 Algorithm Experiments 27

4.2.2 Regimen Experiments 30

4.2.3 Entropy Loss Experiments 32

4.2.4 All-Pass Rewards 32

4.3 Reinforcement Learning from Supervised Model 35

4.3.1 Game Format Experiments 35

4.4 Performance Against Wbridge5 37

Chapter 5: Related Work **39**

5.1 Previous Work on Non-Bridge Games 39

5.2 Previous Work on Bridge Bidding 41

Chapter 6: Conclusion and Future Work **43**

Bibliography **44**

Appendix A: Extra Results **47**

A.1 Reinforcement Learning from Scratch 47

A.2 Reinforcement Learning from Supervised Model 47

LIST OF TABLES

2.1 Conversion table from point differences into IMPs. 6

4.1 Average IMP Difference to Oracle for both expert human games and test games run with the best supervised model. 26

4.2 The average results, standard deviations of the game results and the standard deviations of the average results for each model evaluated against *Wbridge5*. 37

LIST OF FIGURES

3.1 Naive Recurrent Architecture. All inputs are embedded separately. Bids are then processed by the LSTM, concatenated with the rest of the information before being fed to the MLP. 11

3.2 Fully Connected Architecture. Bids are extended with special empty tokens to reach a fixed size. All inputs are embedded separately. All embeddings are concatenated before being fed to the MLP. 12

3.3 Hidden Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before using them to initialize the hidden state of the LSTM. Bids are then processed by the said LSTM and the final hidden is fed to the MLP. 12

3.4 Token Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before using them to extend the bidding sequence as the first input. This extended sequence is then processed by the LSTM and the final hidden is fed to the MLP. 13

3.5 Concatenated Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before concatenating them with all the embedded bids separately. This concatenated sequence is then processed by the LSTM and the final hidden is fed to the MLP. 13

4.1 Comparison of the training set accuracy over epochs for the best performing model for each candidate architecture. 25

4.2 Absolute Difference to Oracle metric applied to the supervised data set containing expert-level human games. The games are grouped per event basis and averaged within their groups. 26
4.3 Absolute IMP Difference to Oracle for different reinforcement learning algorithms during training from scratch. Lower scores are better. . . . 28
4.4 Relative IMP Difference to Supervised model for different reinforcement learning algorithms during training from scratch. Higher scores are better. 28
4.5 Absolute IMP Difference to Oracle for different opponent regimens during training from scratch. Lower scores are better. 29
4.6 Relative IMP Difference to Supervised model for different opponent regimens during training from scratch. Higher scores are better. . . . 29
4.7 Absolute IMP Difference to Oracle for different opponent update periods during training from scratch. Lower scores are better. 31
4.8 Relative IMP Difference to Supervised model for different opponent update periods during training from scratch. Higher scores are better. 31
4.9 Absolute IMP Difference to Oracle for different entropy loss weights during training from scratch. Lower scores are better. 33
4.10 Relative IMP Difference to Supervised model for different entropy loss weights during training from scratch. Higher scores are better. . . 33
4.11 Absolute IMP Difference to Oracle for different all-pass rewards during training from scratch. Lower scores are better. 34
4.12 Relative IMP Difference to Supervised model for different all-pass rewards during training from scratch. Higher scores are better. 34
4.13 Absolute IMP Difference to Oracle for different all-pass rewards during training from the supervised model start. Lower scores are better. 36
4.14 Relative IMP Difference to Supervised model for different all-pass rewards during training from the supervised model start. Higher scores are better. 36

4.15 Absolute IMP Difference to Oracle for the final reinforcement learning models versus the number of deals they have played. Lower scores are better. 38
4.16 Relative IMP Difference to Supervised model for the final reinforcement learning models versus the number of deals they have played. Higher scores are better. 38

A.1 Absolute IMP Difference to Oracle for different game formats when starting from scratch. Lower scores are better. 48
A.2 Relative IMP Difference to Supervised model for different game formats when starting from scratch. Higher scores are better. 48
A.3 Absolute IMP Difference to Oracle for different learning rates when starting from scratch. Lower scores are better. 49
A.4 Relative IMP Difference to Supervised model for different learning rates when starting from scratch. Higher scores are better. 49
A.5 Absolute IMP Difference to Oracle for different algorithms when starting from the supervised model. Lower scores are better. 50
A.6 Relative IMP Difference to Supervised model for different algorithms when starting from the supervised model. Higher scores are better. . 50
A.7 Absolute IMP Difference to Oracle for different training regimens when starting from the supervised model. Lower scores are better. . . 51
A.8 Relative IMP Difference to Supervised model for different training regimens when starting from the supervised model. Higher scores are better. 51
A.9 Absolute IMP Difference to Oracle for different entropy loss weights when starting from the supervised model. Lower scores are better. . . 52
A.10 Relative IMP Difference to Supervised model for different entropy loss weights when starting from the supervised model. Higher scores are better. 52

A.11 Absolute IMP Difference to Oracle for different all-pass rewards when starting from the supervised model. Lower scores are better. 53
A.12 Relative IMP Difference to Supervised model for different all-pass rewards when starting from the supervised model. Higher scores are better. 53
A.13 Absolute IMP Difference to Oracle for different learning rates when starting from the supervised model. Lower scores are better. 54
A.14 Relative IMP Difference to Supervised model for different learning rates when starting from the supervised model. Higher scores are better. 54

ABBREVIATIONS

DDA	Double Dummy Analysis
IMP	International Match Point
LSTM	Long-Short Term Memory
MLP	Multi Layer Perceptron
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Network

Chapter 1

INTRODUCTION

Games have been a topic of great interest in artificial intelligence (AI) researchers and a proving ground for reinforcement learning. Recent advancements of technology focusing on full-information competitive games like Chess [Campbell et al., 2002, Silver et al., 2018] and Go [Silver et al., 2016, Silver et al., 2017, Silver et al., 2018] outperforms even the top-level human players. Lately, researchers have also worked on imperfect information games like Poker [Moravčík et al., 2017, Brown and Sandholm, 2018]. Computer agents like DeepStack [Moravčík et al., 2017] and Libratus [Brown and Sandholm, 2018] have displayed superhuman performance for no-limit Texas hold 'em, a competitive imperfect information game while being limited to the two-player situation. Bayesian Action Decoder [Foerster et al., 2019] achieves near-optimal play in collaborative games like Hanabi.

Contract Bridge, or simply Bridge, is a trick-taking game played with a standard deck of cards. There are 2 teams of 2 players, each having been dealt 13 cards at the start of the game. The game is played in two main phases: bidding and playing.

In the bidding phase, each player can propose a contract while only being able to see their own cards. This phase is played in turns and players can override the last contract by proposing a higher one for their partnership. The final contract plays a big role in the scoring and alters the playing phase. Reaching higher contracts can offer higher rewards but falling behind a contract also imposes a penalty on the partnership. Since this is the only communication channel in this phase, it is not uncommon to see human players use the bids themselves to communicate and reason about the cards they cannot see.

In the playing phase, each player plays a single card in one turn and the best

card wins that round or wins a trick. The goal is to win as many tricks during this 13 turn process, either to make the contract the partnership proposed or prevent the opponent from doing that. The scoring then depends on the contract and how many tricks the declarer of the contract managed to win.

The last phase, playing, is historically handled well by AI agents. In 1998, the GIB program managed to reach 12th place among 34 expert humans in a contest that only involved the playing phase [Ginsberg, 1999]. Recent agents like Jack [Hans Kuijf, 2020] and Wbridge5 [Yves Costel, 2020] also demonstrated strong performances when put against professional-level human players. Lastly, when comparing top-level professional human performances, the skill level of the human players in the playing phase shows negligible variance [Amit and Markovitch, 2006]. Therefore the bidding is the more decisive phase of the game when comparing skill levels.

Unsurprisingly, the bidding is also the harder of the phases to master. During it, the only information a player has is their own hand and the publicly known bidding history. This two sources of information are in huge state-spaces, 6.35×10^{11} for the hands and 10^{47} for the bidding histories [Amit and Markovitch, 2006]. As the single public information pool, bids themselves represent the only possibility of communication. However, this channel is restricted by the requirement of bidding higher than the last bid, serves as the only interaction with the environment, and can also have interfering opponents. Overall the bidding phase represents various hard challenges in a simple ruleset.

Over the history of the game, human players have designed different methods and rules to bid more effectively. These bidding systems provide heuristics about what to bid with what you have, and in turn what to expect when a bid is made as well. The bidding programs earning awards in the World Computer-Bridge Championship often implement a version of human-crafted bidding systems. Recently, deep reinforcement learning has been used to learn a bidding system through bridge gameplay. Some of these studies focus on the collaborative setting, where the opponent partnership is assumed to always pass their turns [Tian et al., 2018, Yeh et al., 2018]. Others considered the normal competitive case with their hand-designed feature representations [Rong et al., 2019, Gong et al., 2019].

In this thesis, we propose a novel architecture for competitive bidding. It requires minimal feature engineering and instead relies on the early parts of the network to represent the game state. We show that when using expert-level human games with supervised learning, this architecture converges faster than a naive implementation. Furthermore, we explore and propose different ways of how reinforcement learning can fine-tune the model learned by the supervised training. Finally, we propose a novel metric to evaluate and compare competitive bridge bidding agents.

The structure of this thesis is organized as follows:

Chapter 2 explains the details of how the contract bridge is played.

Chapter 3 contains the models and the algorithms we have used to tackle the bidding problem.

Chapter 4 describes the results of the experiments we have conducted.

Chapter 5 outlines the related work in the literature.

Chapter 6 concludes this thesis and presents future research directions.

Chapter 2

CONTRACT BRIDGE

Contract bridge is played by four players, defined as North, South, East, and West. These four players are divided into two competing partnerships: North-South against East-West. The game is played with a standard deck of 52 cards, comprised of 13 ranks (from Ace to 2) in each of the four suits (club ♣, diamond ♦, heart ♥, and spade ♠). The heart and the spade are known as major suits and the two others as minor suits. A player is designated as the dealer and that player deals each player 13 cards from a shuffled deck.

At the start of the bidding phase, or the auction, the dealer proposes the first bid. Then the phase proceeds around the table in a clockwise manner. Each player, in turn, chooses to call a bid from the 38 total possible bids:

- A contract bid higher than the last contract bid declared. A contract bid is a tuple of a level (1-7) and a trump suit (A suit or no trump denoted as NT). A bid is higher than another if its level is higher or if the levels are tied its trump is higher. The ordered trump list is as such: ♣ < ♦ < ♥ < ♠ < NT

- Double a contract bid if the last bid was a contract bid by the opponents. Increasing the rewards or penalties for the last declared contract.

- Redouble a contract bid if the last bid was a double by the opponents. Increasing the rewards or penalties further for the last declared contract.

- Pass the turn.

The auction ends when a bid is followed by three consecutive passes. If all the bids were passes, then the game is declared a draw. Otherwise, the contract is the last contract bid made by any of the players. The partnership that declared the last

bid is called the contractor and the other partnerships is called the defender. The trump of the contract sets which suit, if any, is valued above all else for the playing phase. The first player from the contractors who called the trump of the winning bid becomes the declarer and their partner becomes the dummy.

Playing phase starts after the auction is over, and goes for 13 rounds. In the first round the opening lead, the player that is left to the declarer, plays a card then the dummy reveals their hands. During the entire playing phase, declarer plays the dummy's card as well as theirs. After the first card in any turn, the players must play the suit that matches the suit of the first card (lead suit). If a player does not have any such card, then they can play any suit. The round continues clockwise until every player has played a card. When a round ends the player who played the best card wins a trick. This is the highest-ranked card with the trump suit if there is a trump card or the highest-ranked card with the lead suit otherwise. The winning player leads the next round until 13 rounds and all cards are played.

After the playing phase, the contractors compare the number of tricks they've won with the contract. A contract is made if the contractors won a number of tricks greater than or equal to *contract level* + 6. Any tricks won over this number is called over-tricks and the number of missing tricks for the contract is called under-tricks. If the contract is made, the contractors win a contract score as a reward and some more for each over-trick. Otherwise, they will get a negative score for each under-trick. Contracts below 4 ♥ (except 3NT) are called *part score contracts* and rewards a low contract score. Contracts above 4 ♦ (and 3NT) are called *game contracts* which grant a larger contract score. Contracts with level 6 and 7 are called *small slams* and *grand slams* respectively and earn a very large bonus score if made.

The scoring can be amplified by two factors: doubling and vulnerability. If a bid was either doubled or redoubled, then the declarer partnership gets more points if they make the bid or loses more points if they fall short. Vulnerability, unlike doubling, is a constant state on a partnership even before the bidding starts. In each game, a partnership can be either vulnerable or not. Being vulnerable also increases the reward and the penalty a partnership can get if they declare a contract. [ACBL, 2020] contains more details on how the score is exactly calculated.

Raw Points	**IMPs**	**Raw Points**	**IMPs**	**Raw Points**	**IMPs**
20 - 40	1	370 - 420	9	1500 - 1740	17
50 - 80	2	430 - 490	10	1750 - 1990	18
90 - 120	3	500 - 590	11	2000 - 2240	19
130 - 160	4	600 - 740	12	2250 - 2490	20
170 - 210	5	750 - 890	13	2500 - 2990	21
220 - 260	6	900 - 1090	14	3000 - 3490	22
270 - 310	7	1100 - 1290	15	3500 - 3990	23
320 - 360	8	1300 - 1490	16	4000 and up	24

Table 2.1: Conversion table from point differences into IMPs.

For this study, we have used the standard format of tournament scoring in high-level professional play. Duplicate bridge is a common variation of bridge that involves two sets of games being played. Players from two teams play the same game, same deal and same dealer, twice. A team that plays North-South in one game plays East-West in the duplicate game and vice versa. For human teams, different players play different games in order to not leak card information. The scores of teams are then calculated as the relative performances between the games. The goal of this variant is to reduce the effects of randomness over performance. As an example, if a deal favours the North-South partnership then the expectation is that that partnership will do better overall. But if two teams both get to be North-South in that same deal, then skill becomes more important as the advantage is shared. After this relative score is calculated, we convert it to the International Match Point (IMP) scale, which approximates a rough square root of the raw score and discretizes it between 0 and 24. Table 2.1 shows how the IMPs are calculated.

To reduce the variance introduced by the playing phase of the game, we eschew the phase entirely and use a tool called Double Dummy Analysis (DDA) to emulate it. DDA is a method that assumes perfect players with oracle access to every card in every hand and calculates how many tricks each partnership can take. Aside from the deal, there are only 2 factors that go into consideration: the leader of the

first round and the trump suit. With 4 possible leads and 5 possible trumps, there are 20 cases to be considered for each deal which is not that expensive of analysis and can be pre-calculated before the bidding phase. An analysis done on the DDA evaluation finds 55% of the games where the experts take the same amount of tricks DDA predicts and in 90% of the games the difference between the actual play and the analysis tricks were equal or less than one, concluding that DDA is a reasonable approximation to expert play [Rong et al., 2019].

After the DDA, the oracle approximation can go a step further into the bidding phase. Par scores and contracts are the results of perfect oracle agents playing on a deal starting from the bidding phase. Having access to all cards and even the DDA results, each agent is assumed to make the perfect bid that maximizes their score. This can result in contracts that cannot be made, as the agents will prefer getting penalties if they can prevent opponents from getting a larger score compared to the penalty.

Chapter 3

METHODS

For this work, we aim to construct a neural network designed to take the current observable state of the game and provide a probability distribution on the possible bids. We train this network using various deep learning techniques so that the bridge agent that would bid according to these probabilities is a good bridge player.

To achieve this task, our strategy is two-fold. First, we use a data set of expert-level human games and define a classification task of attempting to predict the bids the humans make from the states they observed. Such a task fits perfectly with what we wanted to achieve as both results in a probability distribution over the moves. We use supervised learning to learn this task. The resulting model will not be an ideal player, as expert humans will use different strategies and our model will learn a mix of those strategies. Instead, our aim in this is to both measure the performances of different neural network architectures and to create a strong starting point for the next part of our strategy.

Then, we start to use this network in a reinforcement learning environment. By using the model inside of an actual game environment, we can measure the scores our model gets. These scores are used as signals guiding the network to prefer acting in a manner that results in the most score received. We consider various choices for this process and compare them to determine the best approach to our problem.

We also consider the impact of starting from the model trained with supervised learning by repeating reinforcement learning on randomly initialized models. These resulting models are completely isolated from experiences generated by human bidding systems and thus free of both weaknesses and strengths of said systems. Starting randomly also allow us to better differentiate between our training choices, as some of the experiments done on supervised start yield inconclusive results primarily because of the strong starting point.

Lastly, we consider various ways of measuring and evaluating our network. While the first part had the benefit of having a concrete set of bids we want to replicate, we lack such resources in the reinforcement learning part. If we had the best bids for all possibilities, then our task would be trivial. We propose several ways to quantify the displayed performance of the trained neural network, each with their own strengths and weaknesses.

3.1 Game Representation

To give out a probability vector, or even to compute anything, the neural network needs to have the current observed game state in a way it can process. Neural networks, ultimately, work with numbers and not with a card of "Queen of Clubs" or a bid of "2 no-trump" directly.

Previous studies on bridge bidding have come up with various ways to format the game state into something that can be fed into neural networks. Usually a form of binary string [Yeh et al., 2018, Rong et al., 2019, Gong et al., 2019] or including how long ago certain bids were made [Tian et al., 2018], but always a fixed-length format that can be processed by the most simple neural units.

Our representation in this work takes inspiration from Natural Language Processing paradigms. All information that can be gathered from the state is represented as tokens. The hand is a set of 13 unique tokens out of 52 cards. The vulnerability state is a token out of 4 states. The bidding history is a variable-length sequence of tokens out of 38 bids. All of these tokens are processed by their own embedding layers, tables that convert tokens into numeric vectors, and then passed into the proper network. This allows the training processes to also optimize how to represent each card, each vulnerability, and each bid.

3.2 Supervised Learning

The first task for our networks is a Supervised Classification Problem. Given a set of inputs and expected outputs, we train the network to maximize the probability of predicted outputs matching the expected outputs. In our problem, the inputs are

the game states and the outputs are the bids themselves.

The data set we've used for supervised training consists of over 29 thousand professional level games and over 319 thousand bids from more than 50 tournaments. The original data contains a record of each hand, vulnerability, the dealer, each bid, the final contract, the cards played in the playing phase, number of tricks won by the contractors, and the final achieved score. We took all this data and distilled it into simple pairs of observed state, and each bid.

We have split this data set into two. 90% of the set is assigned for training the networks, while the remainder is reserved for evaluating the accuracy of the networks.

Since our data set involves a significant amount of duplicate bridge games, separate game instances inevitably share deals. This sometimes results in the same observed state resulting in different bids. Since different human bidding systems consider different bids ideal, this is not that surprising. We analysed the data set and observed that even an oracle, which will predict the most frequent bid for any state, can at most achieve 94.8% accuracy on the data set. This places a theoretical cap on the network performance, as even with arbitrary access to all the data we cannot perfectly predict all the bids.

3.2.1 Architecture Experiments

Using this dataset, we have performed experiments on candidate architectures on the bid prediction task. Since we represent the bidding history as a variable-length sequence, we used Recurrent Neural Networks (RNNs) to process the history. RNNs are neural networks which are designed to process data without a fixed length limitation. We used a long-short term memory (LSTM) unit [Hochreiter and Schmidhuber, 1997] to encode the bidding history to a fixed hidden vector. This is then combined with the embeddings of other information like the hand and the vulnerability. Together the entire summary of the state is given to a final layer, a multi-layer perceptron (MLP) with a softmax head, which outputs 38 probabilities for all possible bids. We will call this "Naive Recurrent Architecture". A visual overview of

the architecture can be seen in Figure 3.1.

After this, we considered a non-recurrent architecture named "Fully Connected Architecture" which requires to process the history as a fixed-length input. Our data set does not contain any state with infinite history. Even in theory bridge cannot ever have infinite histories due to the ever-increasing nature of bids. With an upper limit on the sequence length, we can define our history with a length equal to the longest existing history. For the sequences that don't make it to the limit, a vast majority of them, a special token is used to pad the sequences to that limit. This process also introduces a high-level decision about how exactly to pad the sequence. We considered keeping the first bid in a fixed position and keeping the last bid in a fixed position. With this fixed-length representation, we can directly feed it to a final MLP layer that outputs the bid predictions. This candidate can be seen in Figure 3.2.

Lastly, we turn our attention back to the RNNs. One problem of that architecture was the separation of the information. The three parts of the state are all processed separately. We hypothesized that if the LSTM had access to the hand and vulnerability information while processing the bids, we could perform better. Therefore, we also experimented with various ways to incorporate that information to the LSTM, before the final MLP.

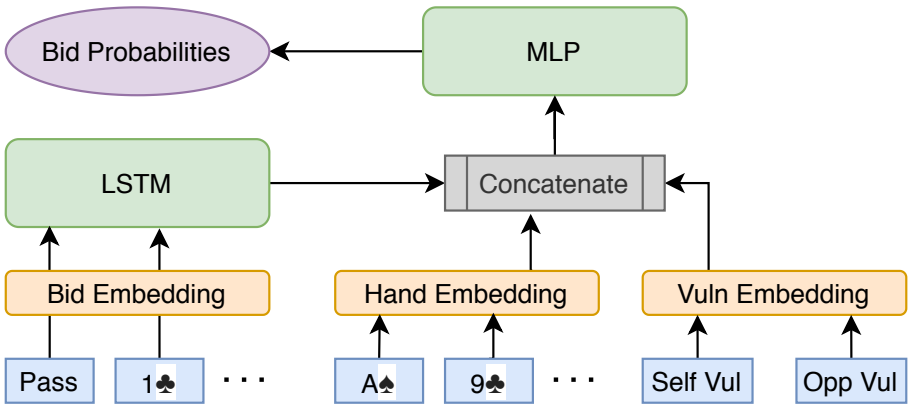

Figure 3.1: Naive Recurrent Architecture. All inputs are embedded separately. Bids are then processed by the LSTM, concatenated with the rest of the information before being fed to the MLP.

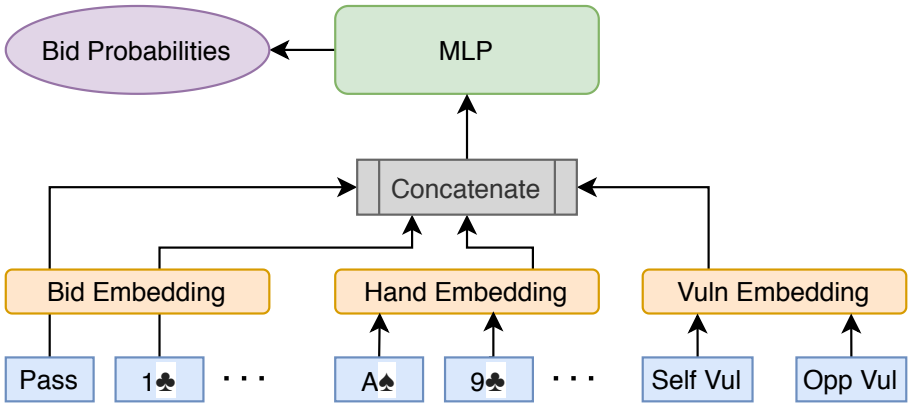

Figure 3.2: Fully Connected Architecture. Bids are extended with special empty tokens to reach a fixed size. All inputs are embedded separately. All embeddings are concatenated before being fed to the MLP.

We used three different ways to integrate the non-bid information to the recurrent network. First was to set the initial state of the network with said information, named "Hidden Recurrent Architecture". Next, we considered giving the information as the first token to the LSTM before the bids, dubbed "Token Recurrent Architecture". Lastly, we merged the information to each of the bid representations before feeding it to the RNN as "Concatenated Recurrent Architecture". These architectures can be found in Figures 3.3, 3.4, and 3.5 respectively.

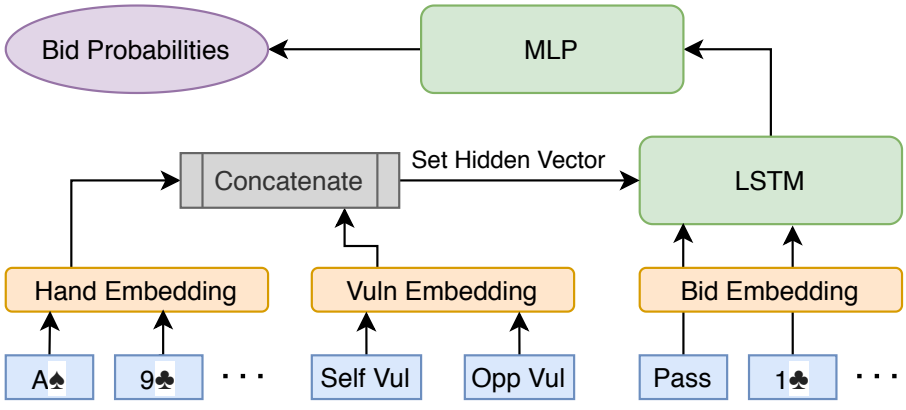

Figure 3.3: Hidden Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before using them to initialize the hidden state of the LSTM. Bids are then processed by the said LSTM and the final hidden is fed to the MLP.

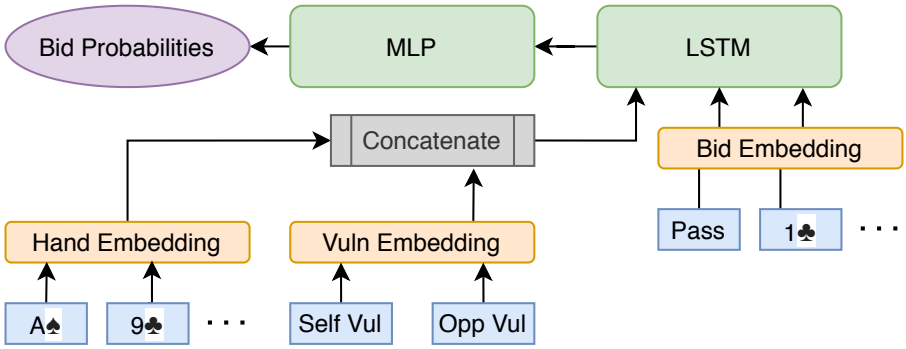

Figure 3.4: Token Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before using them to extend the bidding sequence as the first input. This extended sequence is then processed by the LSTM and the final hidden is fed to the MLP.

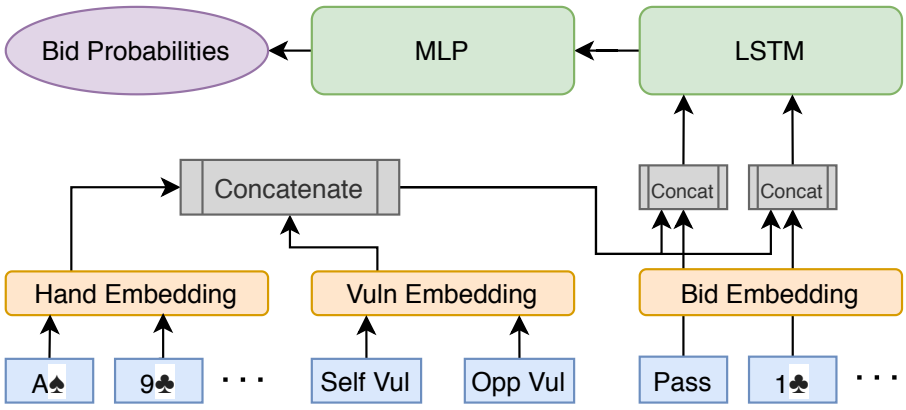

Figure 3.5: Concatenated Recurrent Architecture. All inputs are embedded separately. Hand and vulnerability state are concatenated before concatenating them with all the embedded bids separately. This concatenated sequence is then processed by the LSTM and the final hidden is fed to the MLP.

For all of these architectures, we evaluated them on two metrics. The converged accuracy, and the speed of said convergence. We expected that with given enough complexity and epochs, most models would converge to a similar accuracy. Within those networks, we looked for the ones whose performance improved most rapidly.

3.2.2 Further Training

After the experiments with supervised training, we found that the Concatenated Recurrent Architecture performs the best. However, some reinforcement learning methods need networks that do not act but instead reason about the game.

Two such networks, value and action-value networks, are about predicting the score gained at the end of the game. The value network does this by only looking at the current state, while the action-value network also considers the action to be taken at the current state. To be able to properly engage with methods requiring these networks, we can also use supervised training for them.

Our original data set contains the scores obtained by professional players includes their playing phase. Since our game environment eschews the playing phase with the results obtained by DDA, this will create differences with the scores in the supervised data set. So, to keep things consistent, we have re-calculated the score from the number of tricks predicted by the double dummy analysis.

After the architecture experiments are concluded, we take the most promising architecture in terms of convergence speed and stability and use it for training for these other networks as well. These networks share the same architecture but differ in the output layer. They are trained for the task of value prediction, action-value prediction, and a multi-headed model that does all three tasks at once. All four of these models are saved for the reinforcement learning part as strong starting points.

3.3 Reinforcement Learning

After we have pre-trained multiple networks for different tasks related to bridge bidding, we start the process of reinforcement learning. The process is simple in essence. Let the network guide an agent playing the game, and use the experiences of said game to modify the agent to increase the overall rewards it gets.

However, the process is not as stable as supervised learning. Because the network also guides the agent that is experiencing the game, the data used to refine the model is highly correlated with the model itself. Also, achieving high-level play requires the balance of deciding when to explore different strategies and when to exploit a known strategy.

Our first major choice is how to start this process. We can start from a randomly initialized model and train from that state. This would completely isolate the model from the strengths and weaknesses of human bidding systems. Alternatively, we can

start with the model we trained using supervised learning. It would let us skip the process of making a bad agent into a decent one and focus on making a decent agent a good one. After the initialization, reinforcement learning can proceed without any difference between the two options.

For the learning process, we focused on the policy network, the network that gives the probability vector over the bids, as our main network. We use policy gradient methods to refine our network, which will encourage bids that result in positive scores and discourage bids that result in negative scores.

The game environment we have used is only the bidding phase of contract bridge. As mentioned in Chapter 2, we use double dummy analysis to substitute expert level playing phase. Since the only reward in the bridge comes at the end of the episode, we keep the discount rate at 1. This means that any future rewards are considered as having the same value as having the reward immediately. Lower discount rates quite literally discount future rewards so that they aren't considered the same as an immediate reward, which can be useful if the agent could get rewards during the play. Instead of considering the raw score as the reward, however, we compare that score to the par score and convert it to the IMPs scale and normalize it between $[-1, 1]$ range to calculate the reward.

For the deals used in the games, we have used a data set of over 5 million randomly generated games and their pre-computed DDA results. We have set one thousand games aside for evaluation purposes.

To speed up the game playing, we have played them in a batched manner. For each batch of games, the observations of the games of a particular time step are processed together in the to minimize the time spent per game. This helps with the speed as the hardware we use can process combined inputs faster than providing them separately. As the individual episodes terminate, they are removed from the batch and their results are fixed. This process repeats until all the games are terminated.

To perform the reinforcement learning we have experimented with five different decisions that are mostly orthogonal to each other.

3.3.1 Policy Gradient Algorithms

As previously stated, we use a policy network to guide our agent and use the game experiences to refine this network directly. Policy gradient algorithms are the processes that allow us to do the second part. Simply put these methods use the gradient of the policy, which calculates how to change our network to encourage or discourage a certain bid in a certain state, in different ways to train the network.

For our baseline, we have used one of the simplest policy gradient methods, the REINFORCE algorithm [Sutton and Barto, 2018]. For each experience, that is the observed state and the action we took, we calculate the gradient for that particular move, then multiply it by the final reward received in that game. This causes moves that earn more reward to get encouraged more than the moves that earn less reward, and it even discourages the moves that earn negative rewards. But we will encounter moves with higher probabilities more and thus update them more. To counteract that, we divide this with the probability of encountering that move so that more frequent moves get smaller updates. This method leverages the fact that over the distribution of possible games, the expected gradients are equal to the actual gradients for the Policy Gradient Theorem [Sutton and Barto, 2018].

$$\nabla Loss(state, bid) = -Reward \times \frac{\nabla Policy(state, bid)}{Policy(state, bid)} \tag{3.1}$$

Equation 3.1 gives how to calculate the gradient of the *Loss* function for the REINFORCE algorithm. $Policy(state, bid)$ refers to the probability of the bid the network assigns to a particular bid when seeing a state. *Reward* refers to the final reward the agent gets. We can directly use it since our discount factor is 1. Do note that the loss gradient is the negative of reward times normalized policy gradient. Policy gradient methods work with gradient ascent, i.e attempting to increase the function value. Loss functions are defined with gradient descent, i.e the opposite of ascent. That negative will turn a gradient descent on the loss function into a gradient ascent for our policy network. With some math, we can directly define the *Loss* function, which is given at Equation 3.2

$$Loss(state, bid) = -Reward \times log(Policy(state, bid)) \tag{3.2}$$

Next, we considered an Actor-Critic approach [Sutton and Barto, 2018]. Instead of using the reward directly, we use a separate value network as a baseline for the actual reward. By using the difference between them for our updates, we calculate how much better or worse we did then our expectations. Performing the gradient ascent using this metric allows us to better differentiate between actions that we expect to result in large rewards so that we can focus on the small differences between them. We utilize the pre-trained network predicting the value of a state to initialize the network at a strong point that is already consistent with the policy network

$$Advantage(state) = Reward - Value(state) \tag{3.3}$$

$$Loss(state, bid) = -Advantage(state) \times log(Policy(state, bid)) \tag{3.4}$$

Equations 3.3 and 3.4 defines the *Advantage* function and the *Loss* function for our policy network. The $Value$ refers to the value network predicting the reward from the state. Since the expected reward should change with the changing strategy, we should also train this network too. Equation 3.5 defines the loss function used to train the value network, which gets to its lowest point if $Value$ exactly matches the *Reward* the agent earns.

$$Loss(state) = Advantage(state)^2 \tag{3.5}$$

Lastly, we have looked into Proximal Policy Optimization (PPO) [Schulman et al., 2017]. The previous two algorithms required the updates to be applied directly after playing the game with the given policy network. This is called on-policy training. Another option is off-policy training, in which the algorithm can apply experiences from one agent to train another agent. PPO is one such algorithm.

$$Ratio(state, state, bid) = \frac{TargetPolicy(state, bid)}{SourcePolicy(state, bid)} \tag{3.6}$$

We define the ratio of probabilities for a given *state* and *bid* for a pair of policy networks as the *Ratio* function. The $TargetPolicy$ is the policy network we want to update, and the $SourcePolicy$ is the policy in which the bid was taken. We will only take the gradients with respect to the $TargetPolicy$ and the probability from

the *SourcePolicy* will remain constant. Consider a naive off-policy loss defined in the equation 3.7

$$Loss(state, bid) = -Advantage(state) \times Ratio(state, bid) \tag{3.7}$$

$$\nabla Loss(state, bid) = -Advantage(state) \times \frac{\nabla TargetPolicy(state, bid)}{SourcePolicy(state, bid)} \tag{3.8}$$

Intuitively, it looks reasonable. The gradient of it will work on the target network while keeping in mind that it was coming from the source network and using that probability to counteract the frequency of updating that combination. If both networks happen to be the same, then the gradient is equal to the gradient in the previous algorithm.

But if the bid probabilities are very different from one another, it can lead to instability in the updates in the form of extremely large gradients. To counteract this, PPO proposes a limit to probability difference. Using a new hyperparameter ϵ, the new loss function forces the policy ratio to stay within bounds.

$$Loss(state, bid) = -\min \begin{pmatrix} Advantage(state) \times Ratio(state, bid), \\ g(\epsilon, Advantage(state)) \end{pmatrix} \tag{3.9}$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & \text{if } A \geq 0 \\ (1-\epsilon)A, & \text{otherwise} \end{cases} \tag{3.10}$$

The proposed loss, defined in Equation 3.9, puts a limit on how much the ratio can get away from 1. If the advantage is positive, increasing the ratio beyond $1+\epsilon$ does not cause the loss function to improve. Likewise, if the advantage is negative, decreasing the ratio beyond $1-\epsilon$ does not cause the loss function to improve. In all cases, the improvement is throttled to ensure that the new policy does not diverge too much from the source policy. The authors of PPO suggest 0.2 as a reasonable hyper-parameter, so we used that in our experiments.

Using an off-policy method will also require a way to use off-policy data. The previous two algorithms have used on-policy experience. Since for on-policy data the ratio is 1 by definition, no throttling happens. That means as stated above, using on-policy experience with PPO will essentially be the same as our previous

method. To leverage the off-policy properties, PPO trains on the experiences of a game batch repeatedly for multiple epochs before moving on to another game batch.

3.3.2 Game Format

While the bridge environment is very clearly defined, the existence of the duplicate game structure gives us a choice on how exactly do we play the game in training. Whether to play in a single or duplicate format can be an important factor in our training performance. Therefore, we explored three ways of actually playing the game.

The first one is to just play a single table and use the results directly. This has the benefit of rewards being more directly related to the actions of the agent, but it does so at the cost of being slightly different than our final goal. It would also mean the random deals would impact the game more significantly.

The second option we experimented on is a compromise between the single and duplicate formats that we call Single Par. In it, we play the game in a single format and then compare it against a hypothetical table in which the same deal is played by the oracle agent. This reduces the impact of random deals, is calculated the same way as a full duplicate game, and the rewards aren't related to an entirely different game. However, the oracle play doesn't always make sense in a non-oracle setting, which can skew the rewards by comparing them to an illogical standard.

Lastly, we consider the duplicate format directly. If we want the model to work in a duplicate bridge setting, then using it seems like the natural choice. However, the reward received being tied to another game that the agents had no influence on may seem counter-intuitive.

3.3.3 Training Regimens

One factor that can be important in multi-agent reinforcement learning is the selection of agents. Complete self-play is the popular choice, but we also experimented with integrating other fixed models into the playing phase. For all algorithms, we only use the experiences generated from a single set of partnership. All other expe-

riences that are generated by the other partnership are discarded.

The first option was to integrate the supervised agent as an opponent. We experimented with playing exclusively versus this agent and playing each game versus randomly against either self or the supervised model.

A different approach, as seen in [Gong et al., 2019], is to use a staggered opponent. In this version, the opponent starts as a model's copy, but it doesn't immediately get updated. Instead, the opponent model is updated every once in a while. This gives the training process a bit of time to adapt their strategy to beat their previous self without focusing on fully exploiting a static agent.

Another approach was to build a model zoo. Starting with playing against the initial agent frozen, every so often we copy and freeze the current training model. Then for all future games, the training agent randomly plays against one of the frozen agents. This process can repeat arbitrarily as the training model starts to accumulate experiences against a diverse set of agents. As an addition, the model can also keep the self-playing aspect while building this model zoo.

3.3.4 Exploration using Entropy Loss

One of the central challenges in reinforcement learning is the balancing of exploration versus exploitation. An agent has to both try out new strategies, exploration and then use the well-known strategies with high reward, exploitation, to maximize the rewards it gets. An agent which will only explore will never get the highest rewards, due to never sticking to a good strategy. Likewise, an agent which will only exploit will get stuck to the first strategy it discovers and will never try new and possibly better strategies.

In order to also experiment with this balancing act, we have also added an entropy parameter to our policy loss. We encourage policies with higher entropy by adding negative entropy to our loss function. This discourages policies that heavily prefers a single move. This also adds another hyper-parameter to be tested about how heavily we should weight this in our updates.

Entropy loss also raises an interesting dynamic between the two algorithms we

described that utilizes the advantage function. As the model starts to exploit a discovered strategy, the value function gets more accurate and starts to lower the received advantage value. As this value lowers, the policy loss gets lowered and the entropy loss starts to dominate the gradient and encourage exploration. As the model starts to explore, the strategy starts to change and in turn, the value function loses its accuracy. Which allows the model to discover something new with higher than expected reward and exploit that instead. Intuitively these two factors together seem to generate phases of alternating exploration and exploitation, which can work into our advantage.

3.3.5 All-Pass Reward

When training from scratch, we encountered a particular problem which we call the All-Pass Problem. During training, our models would very rapidly converge into a policy of almost always passing and not improve for a significant amount of training.

Normally, if all of the four players pass then the game is aborted and no one gets any points. This helps making always passing is an extremely defensive strategy. With bad hands making a contract can be sub-optimal, as the chance of fulfilling it is low, and even with good hands it won't get punished too hard as the opponent needs to somehow get good results from bad hands.

Getting out of it is also quite tricky. Normal gameplay still involves a large amount of passing, so the agents need to figure out when to bid without completely disregarding passing. This balance isn't trivial to learn, and the exploration is doubly hostile against any mistake. If only a single random non-pass bid is made, the chances are it was a wrong bid. If so, the bid gets a negative result while the opposing partnership gets a positive result for passing. While leaving this policy is not impossible, it still takes a significant amount of trial and error.

We have tried several possible solutions to this problem, from increasing the entropy loss for more exploration to amplifying learning on non-pass bids. Ultimately, we found that modifying the reward given for the all-pass state allows our networks to quickly leave the all-pass policy.

The primary question here is, what is the new reward for the all-pass state. The general approach we took here is to reward the partnership with bad hands, for passing is superficially the optimal play, and punish the partnership with good hands, for not they could have got some rewards from bidding.

Note that in high-level human games all-pass is very uncommon and in oracle games it is extremely rare. So while we are deviating from the rules of the game, the alteration should not be that significant at high-level games.

Our first option is to give the negative par score to each partnership. If the partnership's par is positive then their hands were good and therefore they are punished, and vice versa. We call this Game Pass Reward.

This is good for single-game formats or in duplicate format when only one table all-passes. However, the 0 reward remains unchanged if both tables all-pass in duplicate bridge. Therefore we experimented with more alterations specifically tailored for the duplicate game. This requires us to break the reward symmetry of duplicate game when double all-pass occurs. Otherwise, when the model has seen all hands and passed in all of them, a joint reward cannot both encourage the good passes while discouraging the bad passes.

Duplicate Pass Reward only changes the reward when both tables all-pass in a single duplicate game. It rewards the negative par to the players without comparing with the other table. A small alteration to it, called the Dual Pass Reward, combines it directly with Game Pass Reward as they alter the rewards of different conditions.

Alternatively, we also tried extending the non-comparison of Duplicate Pass Reward to when a single table all-passes. This method, Extended Duplicate Pass Reward, gives all all-passes direct rewards without comparing with the other table. When only one table all-passes however, we also need to decide on what reward the table that didn't pass gets. We tried two approaches where we either gave it's reward directly or relative to the oracle play for the deal.

3.4 Evaluation Metrics

In supervised learning, we could evaluate the network performance by comparing it to the bids in the data set. However, we cannot do the same thing in the reinforcement learning context. Learning about the best bids in each possible state is what we are trying to achieve. If we had a cheat sheet for that, our problem would already be supervised.

To evaluate the performances of various methods of training described in the previous section, we use a small set of metrics. All of the following metrics is about playing games, and comparing the scores like in duplicate bridge and converting it to IMPs. Each of them also has their own strengths and weaknesses.

3.4.1 Relative to Supervised Learning Agent

We can evaluate the performance of an agent by making it play against a common opponent. Since the supervised pre-trained model is easily accessible for all training processes, we can play games versus it and measure our performance that way.

However, a better agent might not be necessarily better against the supervised agent. The supervised agent will not be perfect, it could be exploited. This evaluation favours the models that can exploit it better, and training regimens involving playing against it are at an advantage when it comes to exploiting its weaknesses.

3.4.2 Relative to Other Agents

If we want to ground our performances in relation to other agents (such as expert humans and hand-crafted programs), we can put our models directly against them. We have managed to interface with the award-winning hand-crafted bridge program *Wbridge5* [Yves Costel, 2020] over a TCP bridge playing protocol for this purpose. But, this metric is very costly to establish. Such an evaluation is more than a thousand times slower than our model playing against itself. This means we cannot use them arbitrarily during training and therefore it remains reserved for only the most promising of models.

3.4.3 Absolute Difference to Oracle

Finally, We propose a fourth metric. We can evaluate the performance of an agent by making it play games with itself, and record how close it gets to the perfect oracle agents. We expect better models to be closer to the performance of the oracle agents than others. This is the only evaluation we perform that does not rely on other agents.

A problem of this approach is that the oracle play is not always logical in the normal partially observable game. If a play relies heavily on a normally hidden information that wouldn't easily be exposed, then we wouldn't expect perfect agents to perform that play under normal conditions. Also since the evaluation is done on a self-play basis, this metric might have a bias for the approaches where the agent gathers experience using self-play.

Chapter 4

RESULTS

4.1 Supervised Learning

We have conducted the first set of experiments on the supervised bid classification task. For the candidate architectures, multiple runs with different hidden, embedding sizes, and layer counts were tested. Figure 4.1 shows the accuracy of the best models for each possible architecture. Most candidate models can achieve performances close to our theoretical limit of 94.8%. We observe that using recurrent units by feeding bids and all non-bid information together for each time step pro-

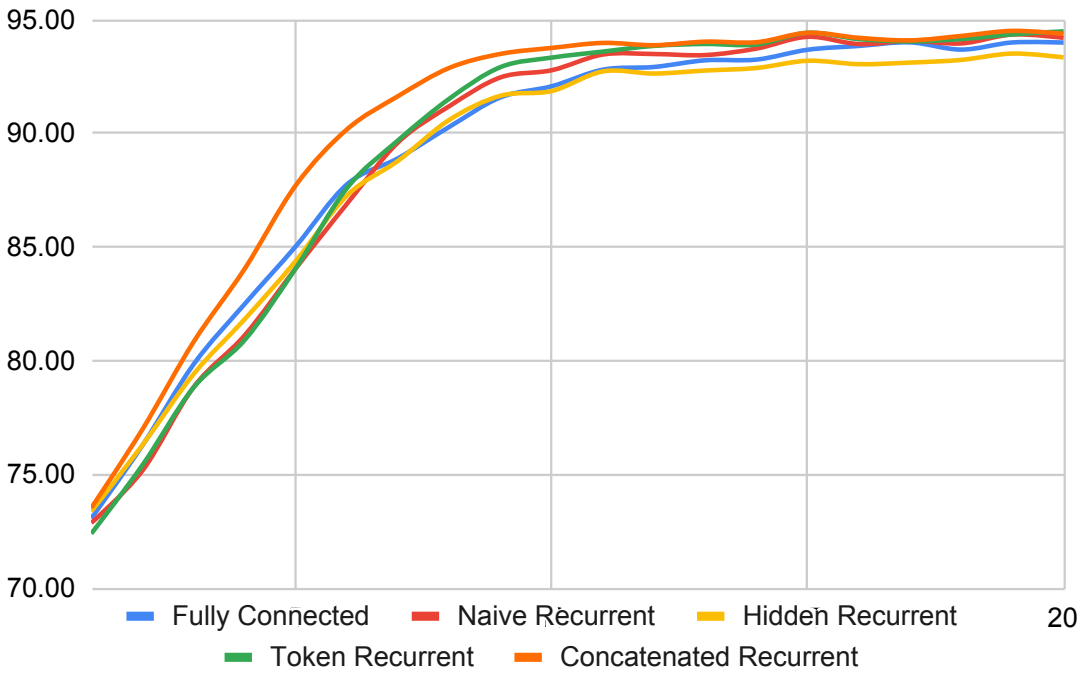

Figure 4.1: Comparison of the training set accuracy over epochs for the best performing model for each candidate architecture.

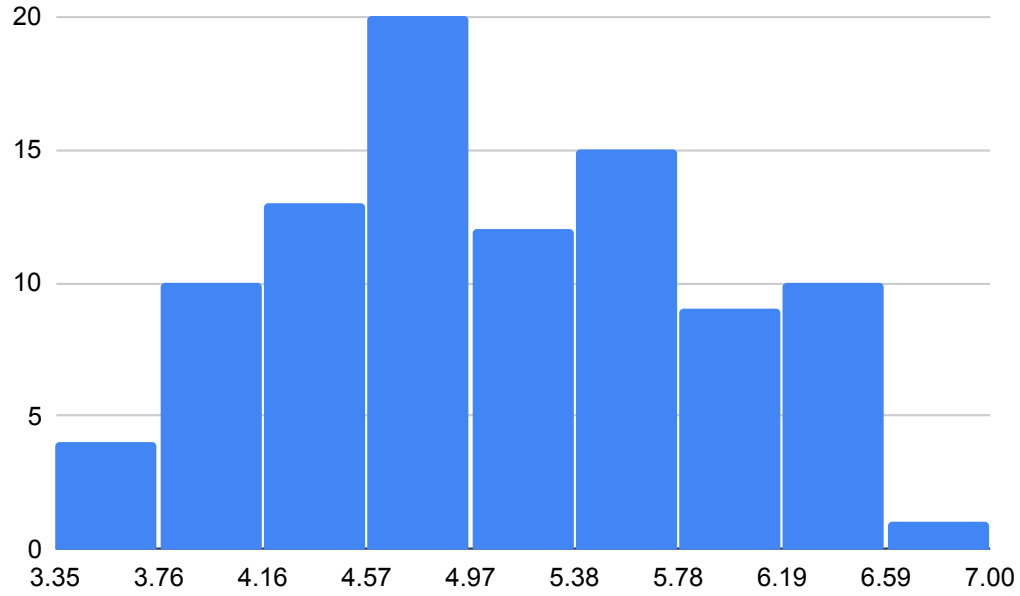

Figure 4.2: Absolute Difference to Oracle metric applied to the supervised data set containing expert-level human games. The games are grouped per event basis and averaged within their groups.

vides a noticeable boost to convergence speed. As an additional advantage the best performing model, "Concatenated Recurrent Architecture", is approximately half the size of "Hidden Recurrent Architecture" and near quarter the size of the other models.

Next, we wanted to evaluate the resulting network in an actual bridge bidding environment. By using the thousand games we reserved for testing, we can calculate an absolute difference to the oracle for each game. But just a number would be meaningless without context. To provide one, we also applied this evaluation metric

Player	**Average Difference**
Expert Level Humans	4.934
Best Supervised Model	4.897

Table 4.1: Average IMP Difference to Oracle for both expert human games and test games run with the best supervised model.

to each of the games in our supervised data set. Figure 4.2 shows a histogram of this metric when grouped by the event and the stage. Table 4.1 shows the mean difference for both the professional games and our supervised model.

While our supervised model performs better in this metric than the expert humans, it doesn't necessarily mean that it is better than them. This metric was intended for self-play, in which all players are the same. By applying this to expert games, which all have different players, we allow it to be divergent from the oracle play by any weak side. As the histogram shows, there are quite an amount of event and stage combinations that perform better than our model. However, by this comparison, we can certainly say that it bids like a decent player and would be a strong starting model for reinforcement learning.

4.2 Reinforcement Learning from Scratch

After we determine the best architecture, we start the experiments on reinforcement learning. All training done here was initialized randomly with the architecture of the best supervised model. For all experiments, we recorded a relative performance against the supervised agent and absolute difference to the oracle during the training. The training is capped at 6 hours or approximately 2.5 million deals, whichever comes first. Appendix A.1 contains additional experiments done starting from scratch.

4.2.1 Algorithm Experiments

First, we have compared the candidate algorithms. For proximal policy optimization, we have experimented with the game experiences repeating for 1, 2, 3, 5 and 10 epochs. Figures 4.3 and 4.4 display the results of oracle comparison and supervised model comparison respectively, averaged over 5 runs.

The most basic algorithm, REINFORCE, perform significantly worse than the others. It spends almost a million deals in the all-pass policy. This shows us that even with our reward engineering on the all-pass state it is not a foolproof method to avoid the all-pass policy.

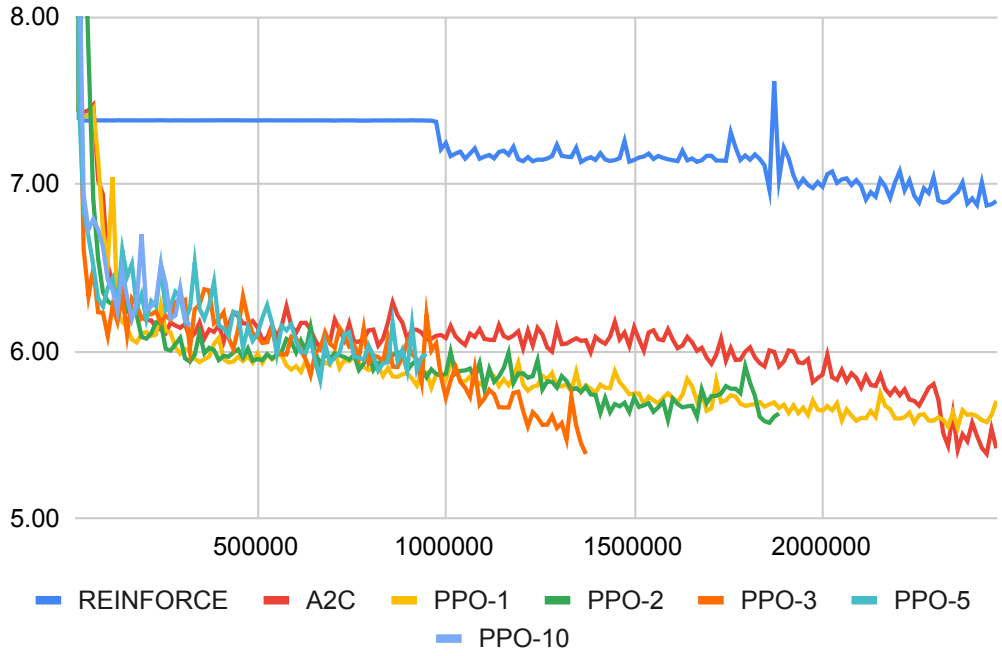

Figure 4.3: Absolute IMP Difference to Oracle for different reinforcement learning algorithms during training from scratch. Lower scores are better.

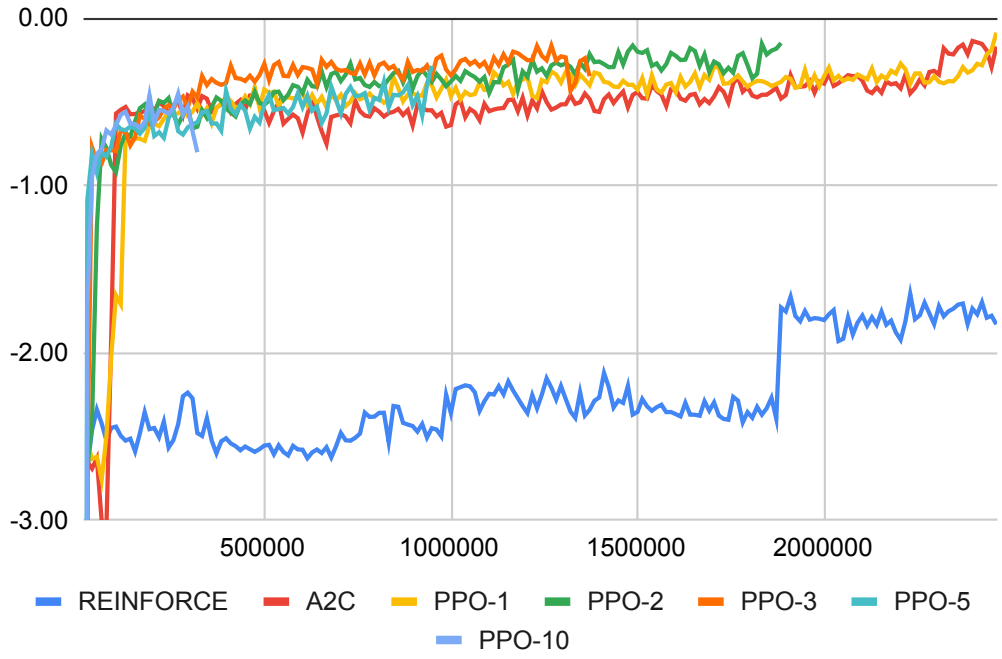

Figure 4.4: Relative IMP Difference to Supervised model for different reinforcement learning algorithms during training from scratch. Higher scores are better.

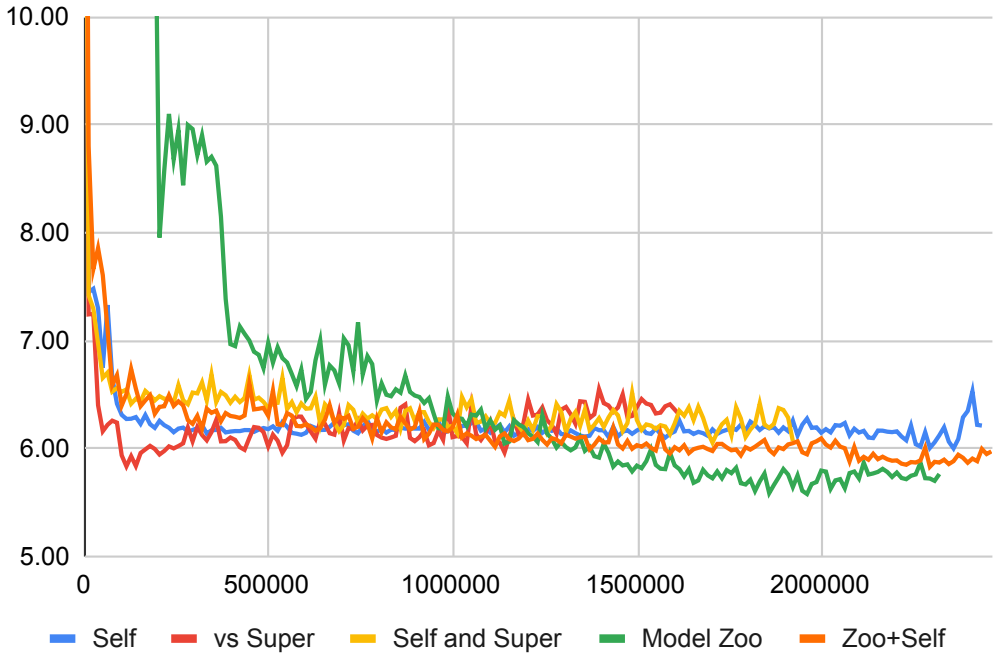

Figure 4.5: Absolute IMP Difference to Oracle for different opponent regimens during training from scratch. Lower scores are better.

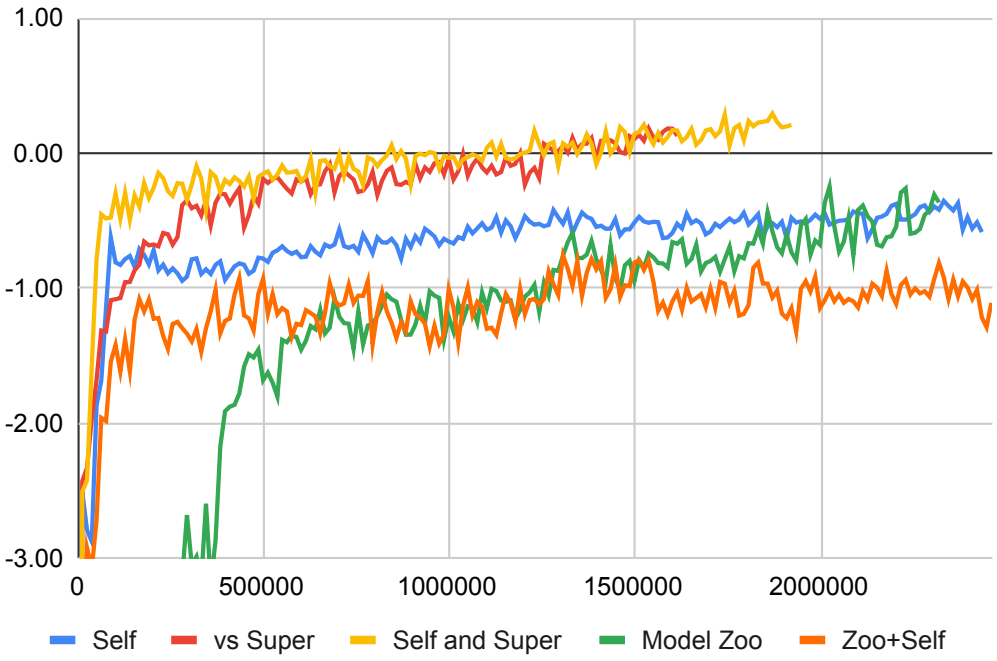

Figure 4.6: Relative IMP Difference to Supervised model for different opponent regimens during training from scratch. Higher scores are better.

We can see that PPO with 3 epochs, while not the fastest algorithm, distinguishes itself in absolute difference, and manages to be slightly above others in performance against the supervised model.

4.2.2 Regimen Experiments

Next, we take a look at the possible different training regimens. Like the previous set of experiments, the Figures 4.5 and 4.6 shows the result of oracle and supervised comparison during the training process as the average of 5 different training runs.

The first observation of note is that the regimens involving the supervised agents, "vs Super" and "Self and Super", perform much better against the supervised model. Their performances in the oracle metric, however, aren't significantly better than other regimens.

"Self" performs the most stable among them all and seems the best at quickly converging into a decent position. While "Model Zoo" approaches the performance of "Self" much later, it remains significantly less stable. We think that since model zoo keeps the very early versions of the model, the training will always include playing against very bad players and may cause the instability. This also manifests in "Zoo and Self", who while starting to improve faster than without the self-play ultimately ends up the worst against the supervised agent.

On top of this, we also performed shorter experiments on different staggered self-play regimens. Figures 4.7 and 4.8 shows the result of oracle and supervised comparison during the training process as the average of 5 different training runs. In these results we can clearly see that update frequency of 1, meaning direct self-play, performs the best compared to updating the opponent at a lower frequency.

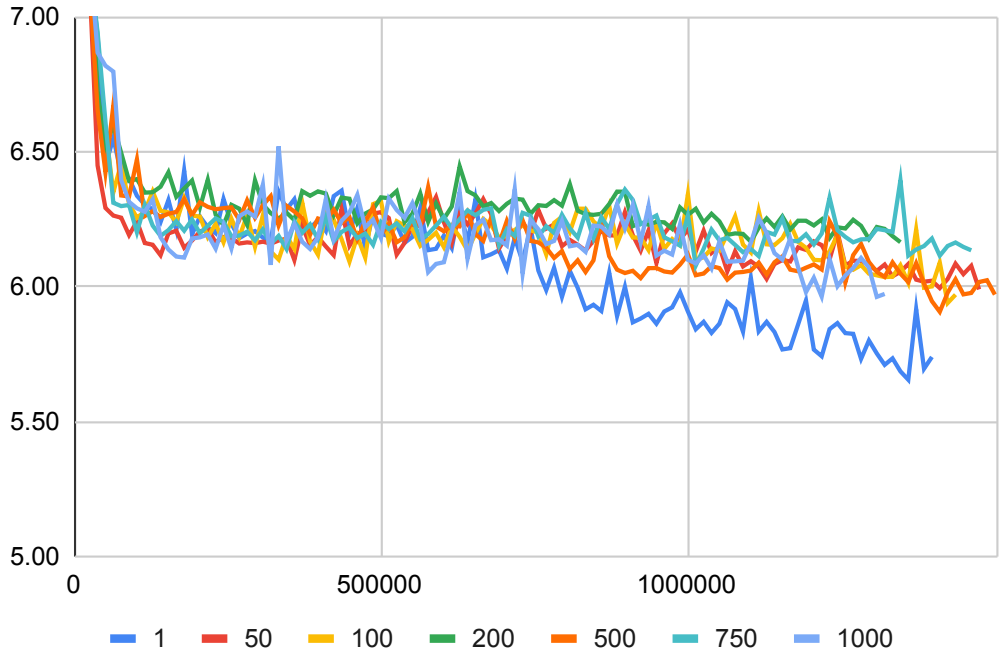

Figure 4.7: Absolute IMP Difference to Oracle for different opponent update periods during training from scratch. Lower scores are better.

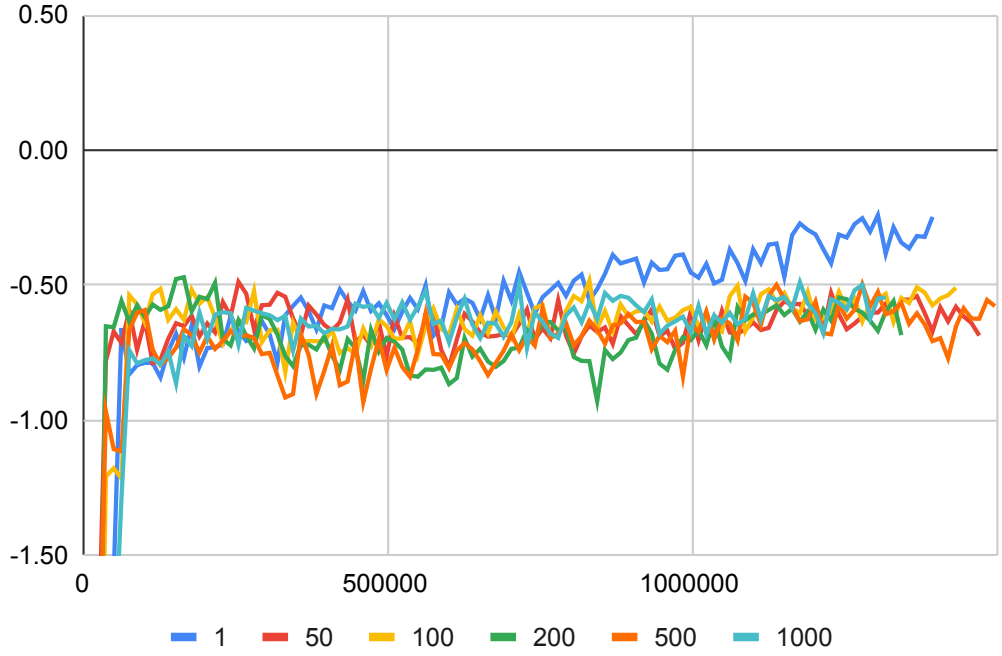

Figure 4.8: Relative IMP Difference to Supervised model for different opponent update periods during training from scratch. Higher scores are better.

4.2.3 Entropy Loss Experiments

We also performed experiments for the weight of the entropy loss to encourage exploration by the agent. As before, the oracle and supervised results are averaged over 5 runs. These results, which can be seen in Figures 4.9 and 4.10, seem to favour a bit of exploration around 10^{-4} and 10^{-3}. These values are marginally better performing than 0 entropy and they do not seem to worsen the performance like 10^{-2} does.

4.2.4 All-Pass Rewards

Lastly, we experimented on different ways to alter the rewards of an all-pass state. The oracle and supervised results, averaged over 5 runs, can be seen in Figures 4.11 and 4.12. In both metrics, we can see that "Game Pass Reward" performing the best, followed by the "Dual Pass Reward" which includes the previous reward. "Duplicate Pass Reward" trails behind, while "Extended Duplicate Pass Rewards" seem to be overall worse than having no modification at all.

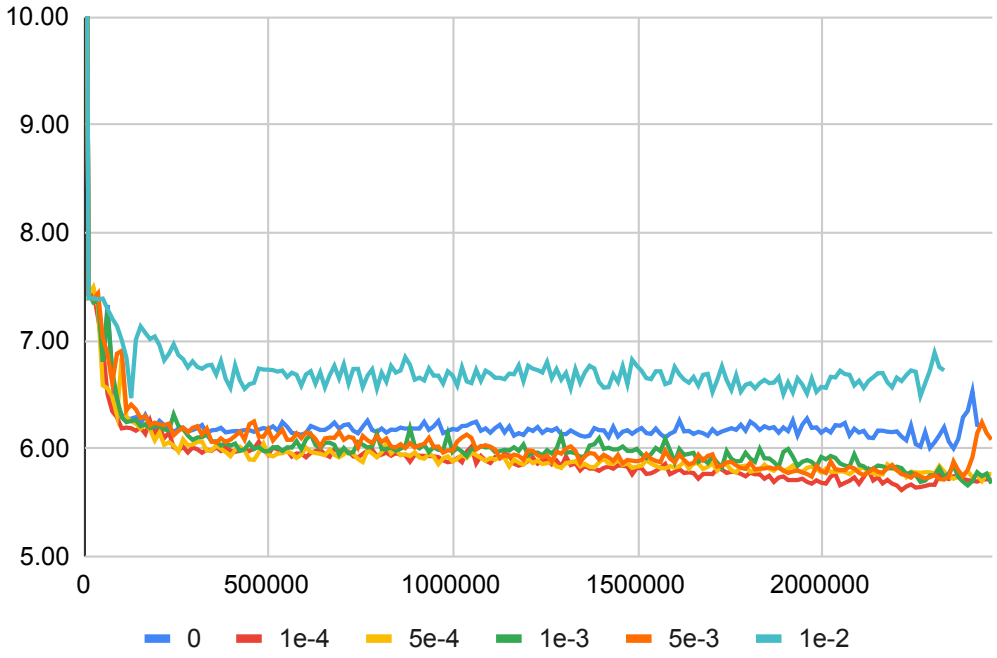

Figure 4.9: Absolute IMP Difference to Oracle for different entropy loss weights during training from scratch. Lower scores are better.

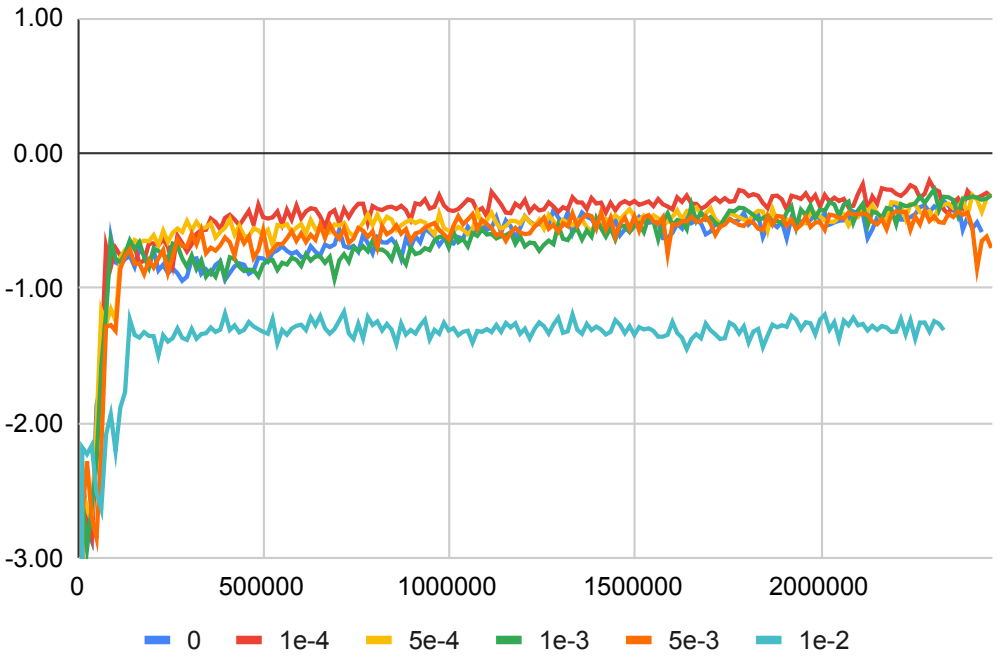

Figure 4.10: Relative IMP Difference to Supervised model for different entropy loss weights during training from scratch. Higher scores are better.

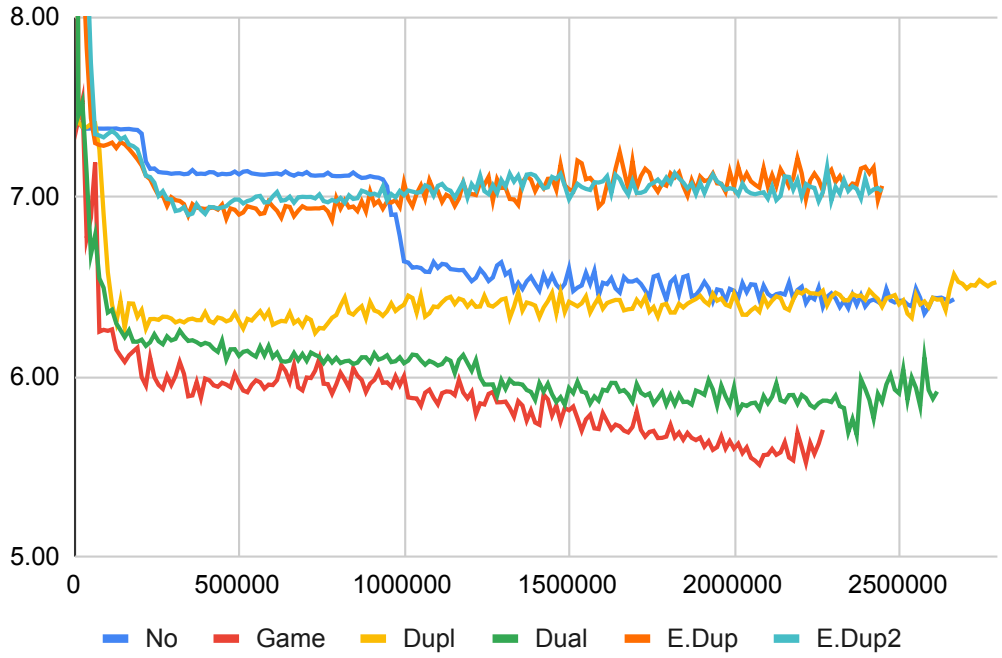

Figure 4.11: Absolute IMP Difference to Oracle for different all-pass rewards during training from scratch. Lower scores are better.

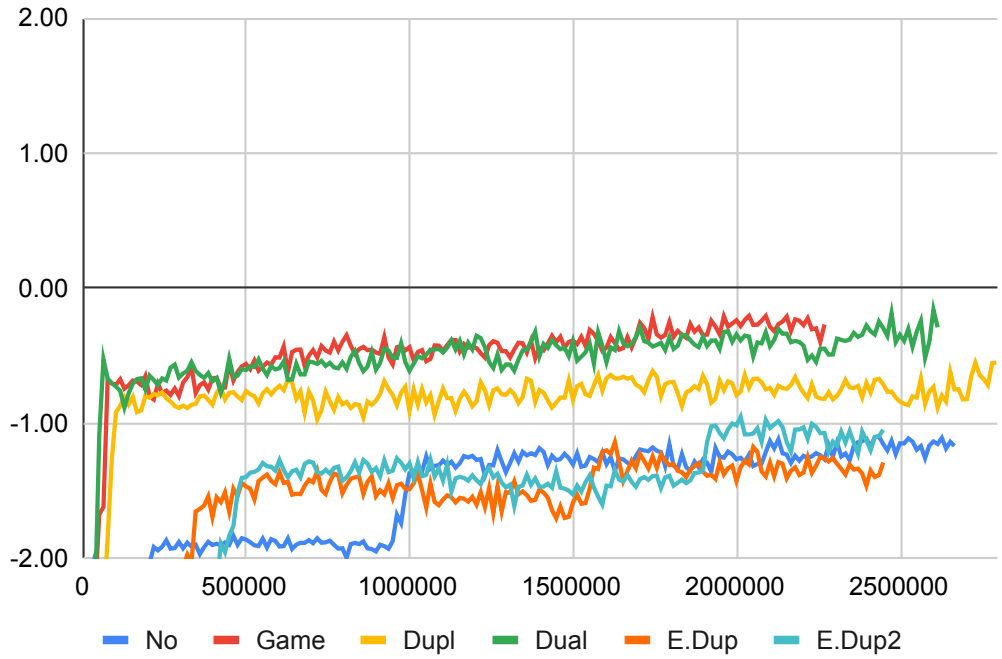

Figure 4.12: Relative IMP Difference to Supervised model for different all-pass rewards during training from scratch. Higher scores are better.

4.3 Reinforcement Learning from Supervised Model

All experiments for reinforcement learning is also done with the initial model starting from the supervised model. For all experiments, we recorded a relative performance against the supervised agent and absolute difference to the oracle during the training. The training is capped at 6 hours or approximately 2.5 million deals like the previous experiments.

We will only discuss a single set of experiments here. The remainder of the experiments, which you can find in Appendix A.2, generally result in too similar results unless an explored option is significantly worse for the training.

4.3.1 Game Format Experiments

We have experimented with different game formats and how it would affect our training process. As before, the oracle and supervised results are averaged over 5 runs, which can be seen in Figures 4.13 and 4.13. The resulting graphs indicate that training in duplicate format, while slower, achieve a better result in significantly less amount of deals processed.

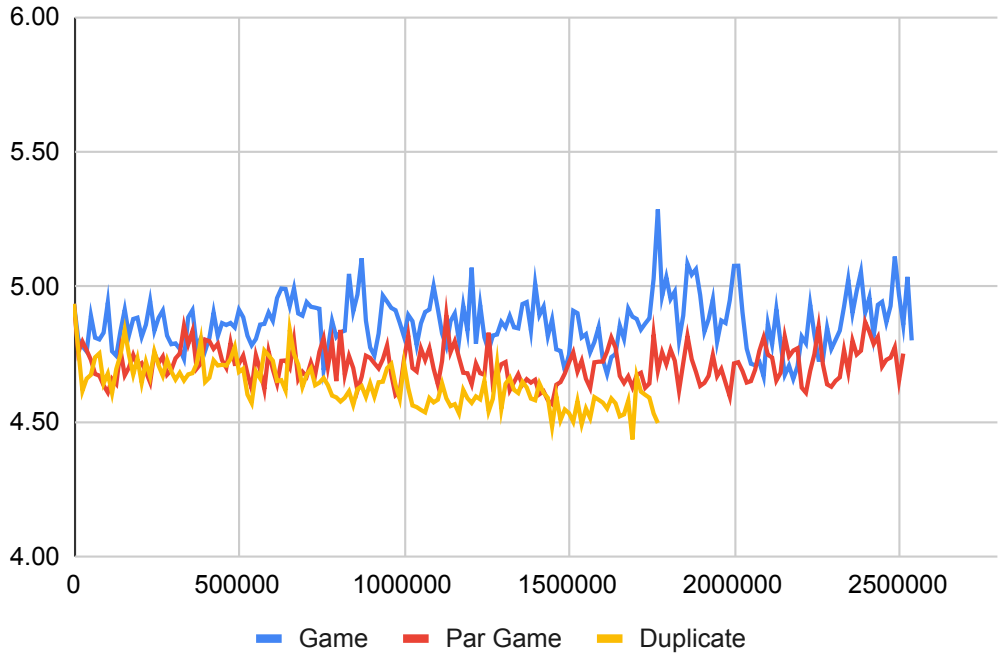

Figure 4.13: Absolute IMP Difference to Oracle for different all-pass rewards during training from the supervised model start. Lower scores are better.

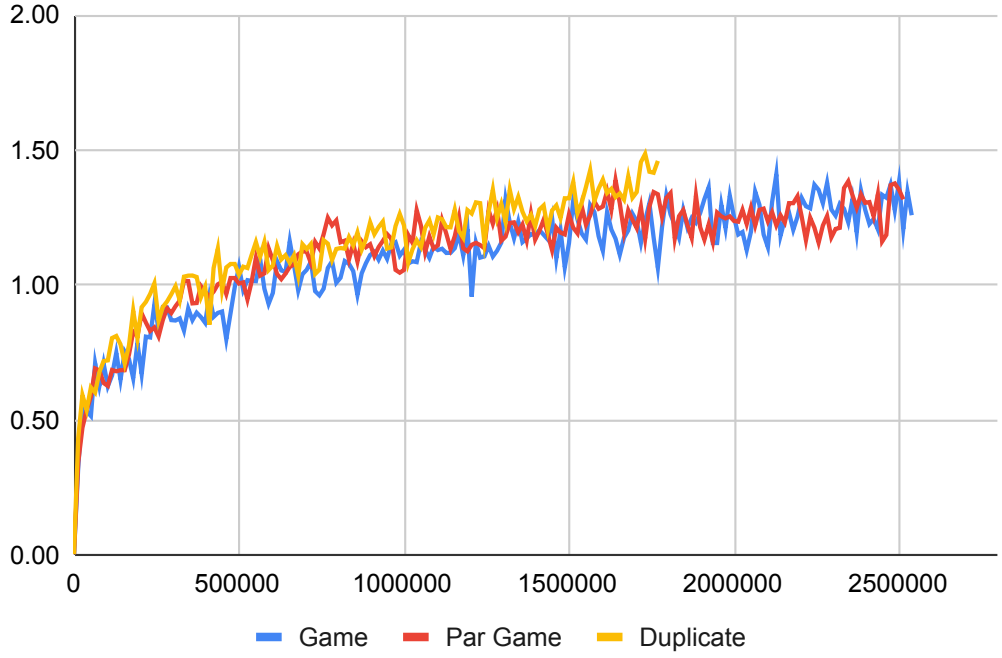

Figure 4.14: Relative IMP Difference to Supervised model for different all-pass rewards during training from the supervised model start. Higher scores are better.

4.4 Performance Against Wbridge5

To evaluate the performance of our models in a more grounded manner, we directly competed against the *Wbridge5* [Yves Costel, 2020], an award-winning handcrafted bridge software. We have pitted our best models in three categories against it, using a separate set of testing deals that were not used before.

Our first model is the best supervised agent we have. It performed with an average of -1.71 IMPs against *Wbridge5*.

The second model is the best model trained from scratch with reinforcement learning, playing over 19 million games over a period of 3 days. It's training consisted of Proximal Policy Optimization with 3 epochs using the duplicate format with direct self-play. It had $5 * 10^{-4}$ entropy weight and Rmsprop with $7 * 10^{-4}$ learning rate as it's optimizer. When played against *Wbridge5*, the model got an average of -3.20 IMPs.

The last model is the best model trained from the supervised model with reinforcement learning, trained over 32 million games in 5 days. It's used A2C with the duplicate format and direct self-play. It had 0 entropy weight and Rmsprop with $5 * 10^{-4}$ learning rate as it's optimizer. The final model achieved an average of -0.35 IMPs versus *Wbridge5*.

Figures 4.15 and 4.16 display the observed performance on our other metrics over the duration of the training for our final models.

Model Category	**Average Result**	**Result Std**	**Average Std**
Supervised	-1.71	5.465	0.195
From Scratch	-3.20	6.431	0.234
Pre-trained	-0.35	5.286	0.195

Table 4.2: The average results, standard deviations of the game results and the standard deviations of the average results for each model evaluated against *Wbridge5*.

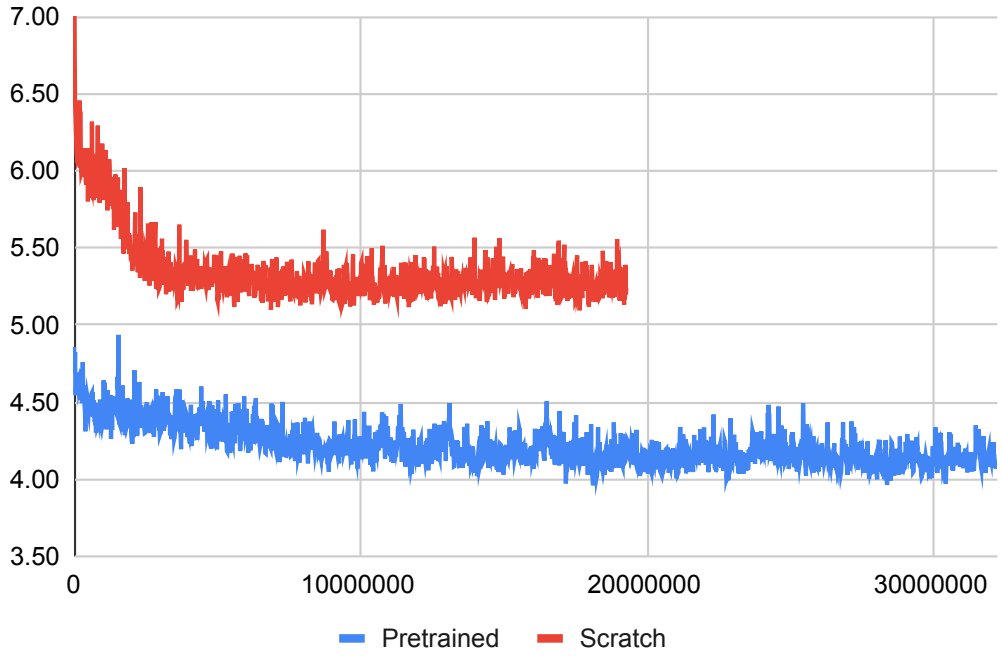

Figure 4.15: Absolute IMP Difference to Oracle for the final reinforcement learning models versus the number of deals they have played. Lower scores are better.

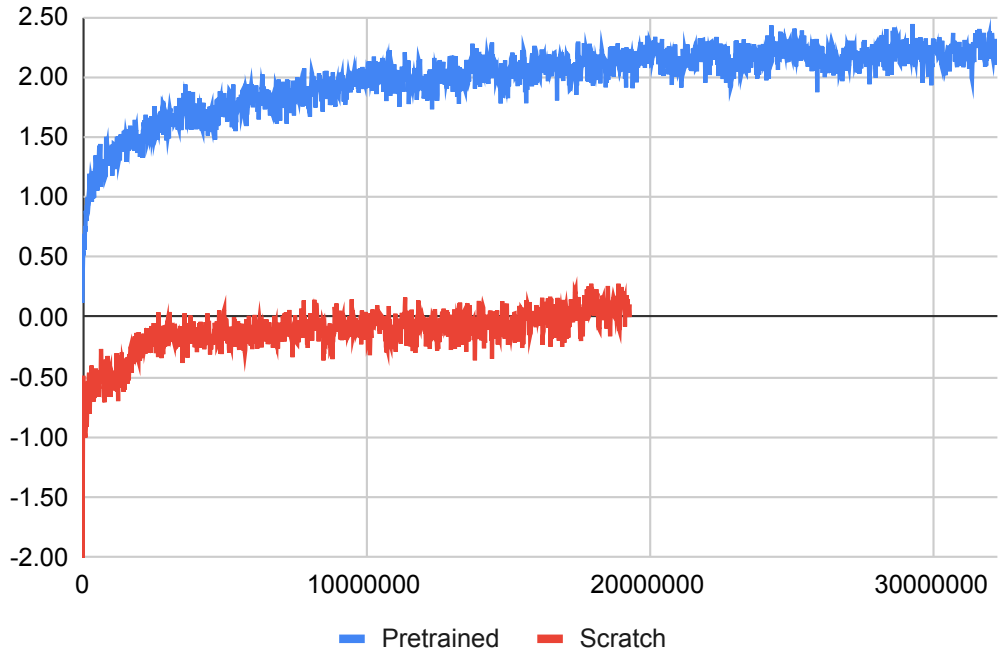

Figure 4.16: Relative IMP Difference to Supervised model for the final reinforcement learning models versus the number of deals they have played. Higher scores are better.

Chapter 5

RELATED WORK

In this chapter, we present and describe the related work referenced throughout the thesis in order to compare and contrast between our and their approaches. First, we explore the studies done on games in general. Then we consider the research done on the bridge in particular.

5.1 Previous Work on Non-Bridge Games

Research on perfect information games has used a potent tool that is not available to the imperfect information game agents. Perfect information allows players to simulate the games for multiple time steps at a time to refine their estimates. Deep Blue [Campbell et al., 2002] has used this technique by performing the minimax algorithm on the game tree to refine the values it obtained from a hand-designed value function. More recently the performances of AlphaGo, AlphaGoZero and AlphaZero [Silver et al., 2016, Silver et al., 2017, Silver et al., 2018] have benefited Monte Carlo Tree Search from their current state with guidance from their value functions to refine their policy functions. While it would be useful, performing these simulations in an imperfect information setting is not a feasible approach for us.

Another complication imperfect information games face is a phenomenon called subgame dependency [Brown and Sandholm, 2018]. In perfect information games, a strategy for a game state only depends expected rewards for states players can reach from that state. In chess, a what-if situation in the early game doesn't affect the moves for mid-game. But, such a dependency exists in imperfect information games. The intuition behind this is that a player can reason about private information from the states the opponent ignored in favour of the current state.

Two previous work on the game of poker has addressed this problem. Libratus

[Brown and Sandholm, 2018] has used game-theoretic tools to entirely solve an easier version of Poker to build a blueprint strategy beforehand. When playing they construct an abstraction for the current state and solve it in real-time. This simplified version significantly lowers the action space, leveraging the fact that in no-limit poker similar raises have similar meanings. It also lowers the card possibilities by grouping deals together algorithmically. While these reductions are discarded in actual play, they are what allow an entire game solution in the form of the blueprint strategy.

Such a game-theoretic approach is quite different than our deep learning approach, but it can be useful for us. A blueprint strategy for a bridge variant with massively simplified hand space can be valuable. Unfortunately, we are not sure about how to reduce our game complexity low enough to solve it directly while keeping it high enough so that the blueprint strategy is useful in the bridge game. Also, we can try deep reinforcement learning to learn a strategy and use their refinement techniques to elevate our final strategy.

DeepStack [Moravčík et al., 2017], on the other hand, models belief about its own hand and the counterfactual values for the opponent's hand to solve the subgames. This requires it to assign values for each possible hands for both themselves and their opponent. This is not a problem in the game of Hold'em Poker. With each player having 2 cards, the number of possible hands in it is 1326. However, to perform this in bridge, we would need to reduce the state space of possible hands from 6.35×10^{11} to keep it feasible.

Bayesian Action Decoder [Foerster et al., 2019] models both public and private belief over possible hidden information states to decide on a deterministic communication protocol. They suggest formulating belief on card games to predict individual cards independently to reduce the state space. Also, their belief formulation on the Hanabi game involves predicting hand-crafted heuristics. Both of these ideas can be used to reduce our state-space when we need a distribution of values over it.

5.2 Previous Work on Bridge Bidding

The research done on bridge bidding is divided into two categories: competitive and cooperative. The first category involves bridge bidding as is. The co-operative category involves a variant of the bridge with only a single partnership whose opposition is assumed to always pass their turn. This situation can happen frequently in actual games and it is a popular way to evaluate the bidding level of partnerships without any opponent interference.

[Yeh et al., 2018] use reinforcement learning to train a co-operative bidding agent. They propose Penetrative Bellman's Equation to train an action-value network more efficiently. However, their system trains a separate network for each time step, and thus can only handle a fixed amount of time steps. While they train up to five different networks, approximately equating ten time-steps in the competitive bridge, this limitation is not realistic in a regular bridge game. Their network takes a binary string representing their hand and which bids are previous made, non-ambiguous in the co-operative bridge, and use a fully connected network to predict a value for each action. Reinforcement learning is done using 100 thousand deals.

[Tian et al., 2018] consider co-operative bridge bidding as a case study in communication through actions. They train a policy network with the help of a belief network that tries to model the cards of their partner. This belief is directly used by the policy network. To encourage informative bids, they give auxiliary rewards to bids that help their partner model their cards more accurately. Their belief network processes bids, which is a vector that signifies who made the bid and when, and the resulting belief is combined with the hand before being processed by the policy network. Both networks are fully connected and the policy network is trained using Proximal Policy Optimization. They train their networks using 1.5 million deals in total.

[Rong et al., 2019] train a competitive agent using deep reinforcement learning techniques. Like the previous approach, they also explicitly model their partner's hand. They model the bidding history as a 318-dimensional binary vector and unlike the previous approach, they do not separate which network gets which information.

Both networks process the player's cards, vulnerability, and the bidding history. Policy network also processes the output of the belief network. Both networks are fully connected with skip connections. Before reinforcement learning, they first perform supervised training on a data set of 700 thousand games or more than 8.4 million bids. They use REINFORCE as their policy training algorithm and maintain a model zoo. Each 100 mini-batches current network is saved to the zoo and each batch is played against a random model from the zoo. Training is done using 2 million deals and the final model manages to beat *Wbrige5* by 0.25 IMPs per board over 64 games.

[Gong et al., 2019] also trains a competitive agent with reinforcement learning, but they eschew the supervised training and train from scratch only using reinforcement learning. They encode the entire state as a binary string with 267 bits. Their model is also fully connected with skip connections, and it predicts both the policy and the value but it does not attempt to explicitly model their partner's hand. They use Asynchronous Advantage Actor-Critic [Mnih et al., 2016] on 2.5 million deals and the resulting model manages to beat *Wbrige5* by 0.41 IMPs per board over 64 games.

Chapter 6

CONCLUSION AND FUTURE WORK

This thesis presented a reinforcement learning approach to learn a bridge bidding agent, supported by minimal feature engineering and supervised learning on expert level human bidding data. The experiments we have conducted on supervised training show that using a recurrent unit that processes bids together with the hand and the vulnerability state converges faster on supervised data compared to other candidate architectures. We explored several independent strategies to perform reinforcement learning to train the agent. Our experiments revealed a slight advantage to Proximal Policy Optimization when directly compared to other algorithms, and while using complete self-play for the training regimen was the most stable option without compromising from performance. Experimenting with the entropy loss to encourage exploration offered small advantages when used with small weights. Ultimately we used a duplicate game format with a small modification to the reward function to discourage strategies that mostly passed. When compared against the award-winning bidding software *Wbridge5*, our model has lost only with 0.35 IMPs per board against it.

For future work, we first consider replacing the recurrent unit with a Transformer [Vaswani et al., 2017], as it shows great promise in processing variable-length data. Next, we suggest incorporating other strong agents, whether hand-designed or machine learning agents, into the training process as strong baselines. Lastly, our methods were completely model-free, but explicitly using the bridge rules to either perform simulations to refine bids or explicitly model the private state of the game may be helpful to the agent performance.

BIBLIOGRAPHY

[ACBL, 2020] ACBL ((accessed September 10, 2020)). Duplicate bridge scoring. `https://www.acbl.org/learn_page/how-to-play-bridge/how-to-keep-score/duplicate/`.

[Amit and Markovitch, 2006] Amit, A. and Markovitch, S. (2006). Learning to bid in bridge. *Machine Learning*, 63(3):287–327.

[Brown and Sandholm, 2018] Brown, N. and Sandholm, T. (2018). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424.

[Campbell et al., 2002] Campbell, M., Hoane, A., and hsiung Hsu, F. (2002). Deep blue. *Artificial Intelligence*, 134(1-2):57–83.

[Foerster et al., 2019] Foerster, J., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M., and Bowling, M. (2019). Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951.

[Ginsberg, 1999] Ginsberg, M. L. (1999). Gib: Steps toward an expert-level bridge-playing program. In *IJCAI*, volume 99, pages 584–589. Citeseer.

[Gong et al., 2019] Gong, Q., Jiang, Y., and Tian, Y. (2019). Simple is better: Training an end-to-end contract bridge bidding agent without human knowledge.

[Hans Kuijf, 2020] Hans Kuijf ((accessed September 10, 2020)). Jack bridge. `http://www.jackbridge.com/eindex.htm`.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[Moravčík et al., 2017] Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513.

[Rong et al., 2019] Rong, J., Qin, T., and An, B. (2019). Competitive bridge bidding with deep neural networks. *arXiv preprint arXiv:1903.00900*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[Tian et al., 2018] Tian, Z., Zou, S., Warr, T., Wu, L., and Wang, J. (2018). Learning multi-agent implicit communication through actions: A case study in contract bridge, a collaborative imperfect information game. *arXiv preprint arXiv:1810.04444*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Computing Research Repository*, arXiv:1706.03762.

[Yeh et al., 2018] Yeh, C.-K., Hsieh, C.-Y., and Lin, H.-T. (2018). Automatic bridge bidding using deep reinforcement learning. *IEEE Transactions on Games*, 10(4):365–377.

[Yves Costel, 2020] Yves Costel ((accessed September 10, 2020)). Wbridge5. `http://www.wbridge5.com/`.

Appendix A

EXTRA RESULTS

Some of our experiments have results that are not clear or were not answering questions that are not as interesting as the ones in the main text. They are included here for the sake of completeness.

A.1 Reinforcement Learning from Scratch

For training from scratch, we have the experiments for the game format (Figures A.1 and A.2) and learning rate (Figures A.3 and A.4). The game format has duplicate games performing better than others in the oracle metric, but otherwise, the results are similar. Learning rate experiments, while conclusive, was a standard hyper-parameter search that doesn't teach us much about our problem.

A.2 Reinforcement Learning from Supervised Model

For the application of reinforcement learning on top of supervised learning, we have four experiments with inconclusive results and also the learning rate experiment. The inconclusive experiments are as follows: Algorithm Experiment (Figures A.5 and A.6), Training Regimen Experiment (Figures A.7 and A.8), Entropy Loss Experiment (Figures A.9 and A.10), and All-Pass Reward Experiment (Figures A.11 and A.12). The results of the Learning Rate Experiment can be found in Figures A.13 and A.14.

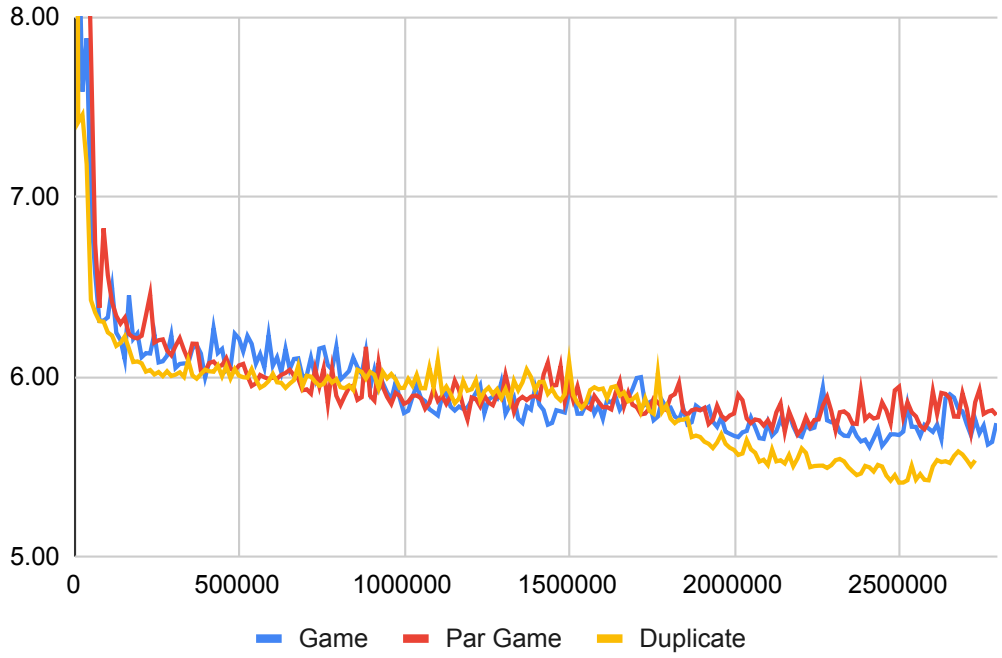

Figure A.1: Absolute IMP Difference to Oracle for different game formats when starting from scratch. Lower scores are better.

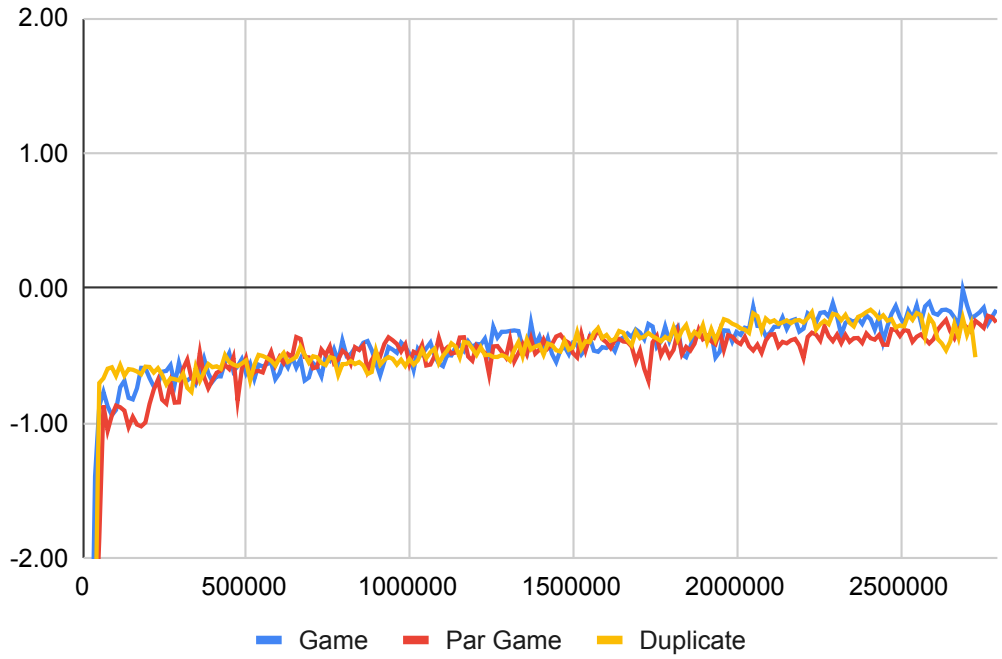

Figure A.2: Relative IMP Difference to Supervised model for different game formats when starting from scratch. Higher scores are better.

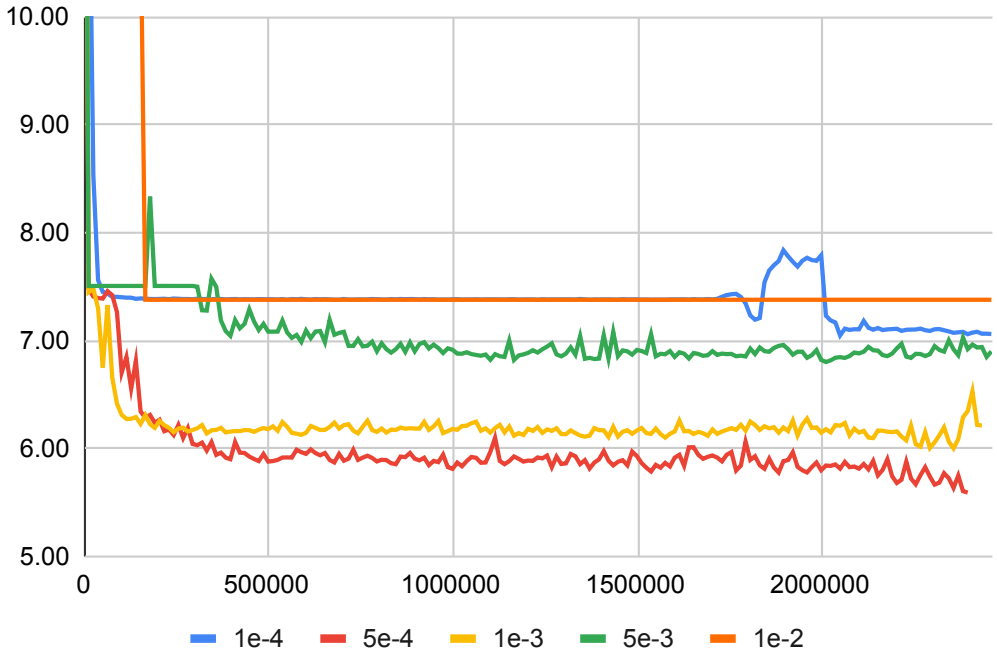

Figure A.3: Absolute IMP Difference to Oracle for different learning rates when starting from scratch. Lower scores are better.

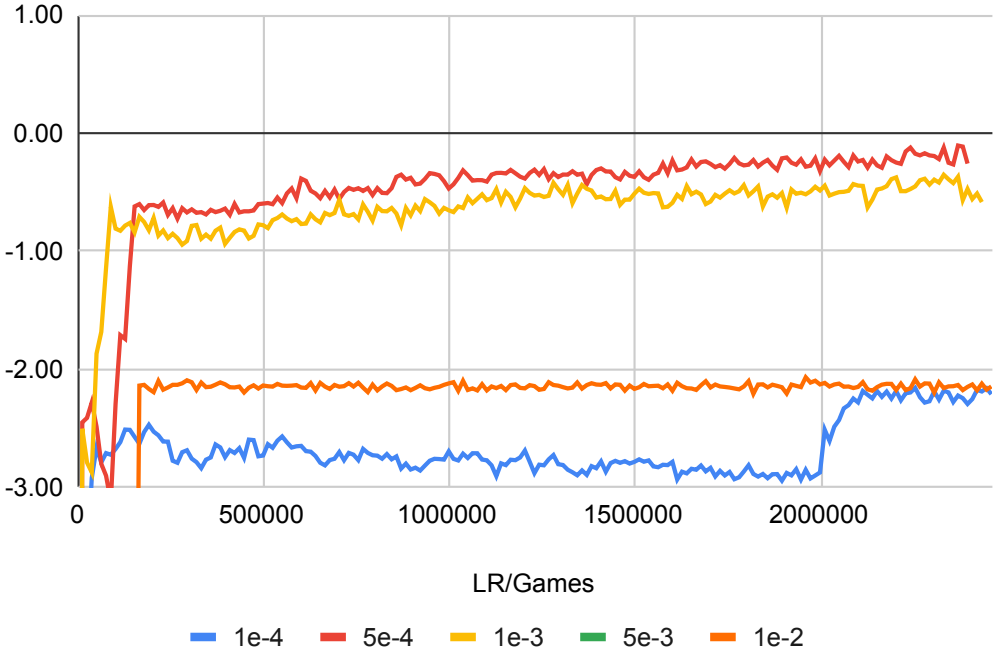

Figure A.4: Relative IMP Difference to Supervised model for different learning rates when starting from scratch. Higher scores are better.

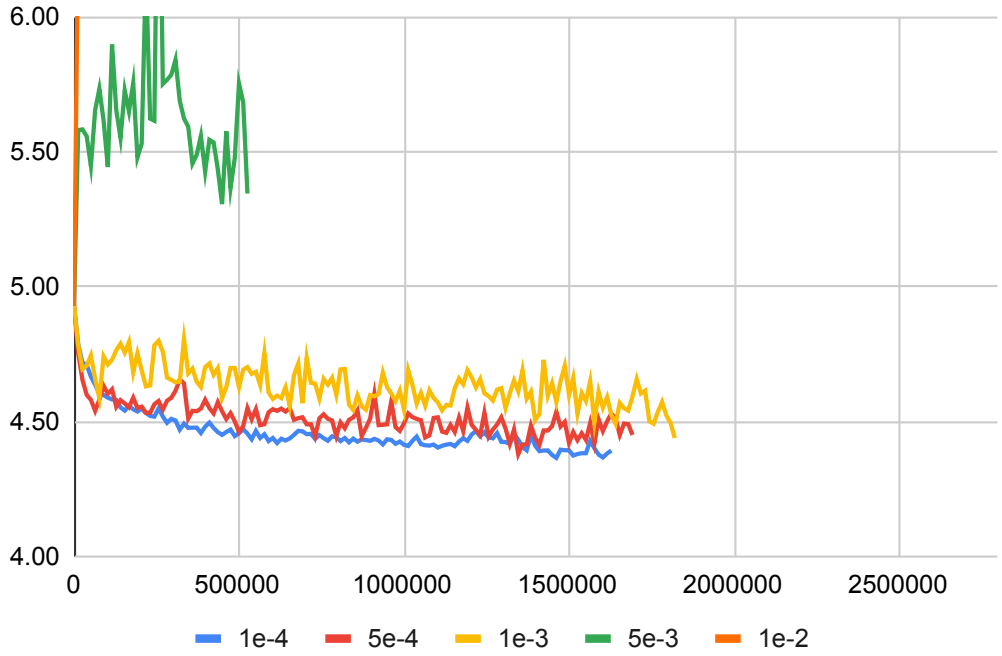

Figure A.5: Absolute IMP Difference to Oracle for different algorithms when starting from the supervised model. Lower scores are better.

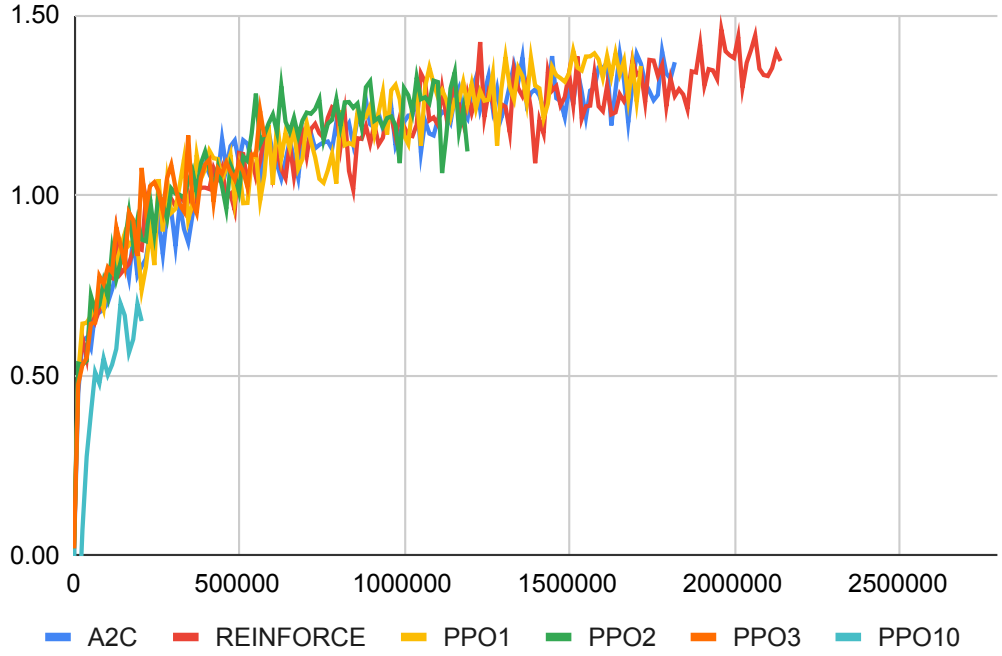

Figure A.6: Relative IMP Difference to Supervised model for different algorithms when starting from the supervised model. Higher scores are better.

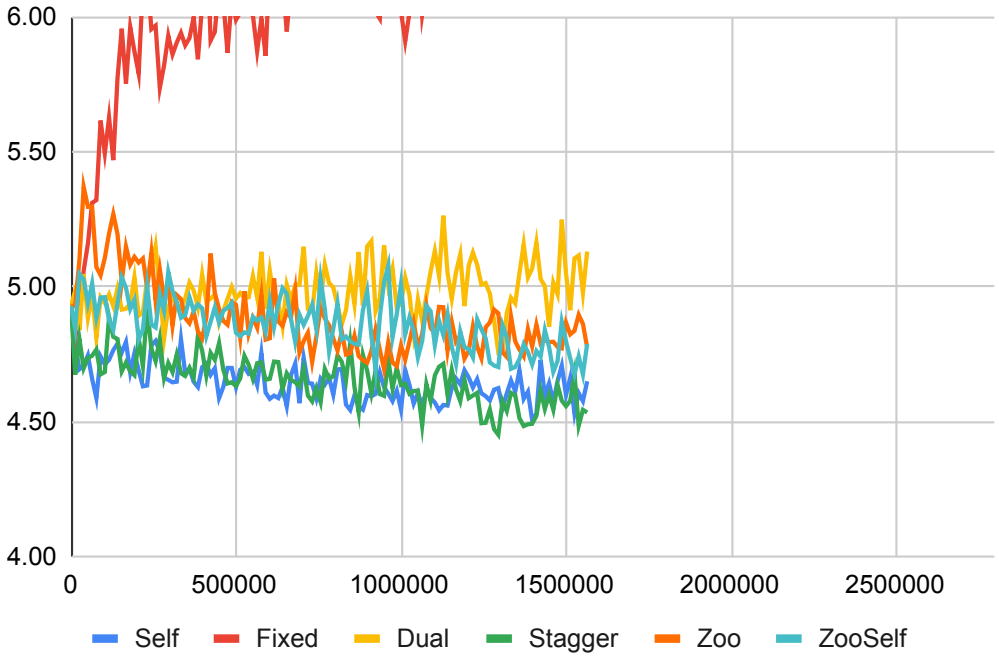

Figure A.7: Absolute IMP Difference to Oracle for different training regimens when starting from the supervised model. Lower scores are better.

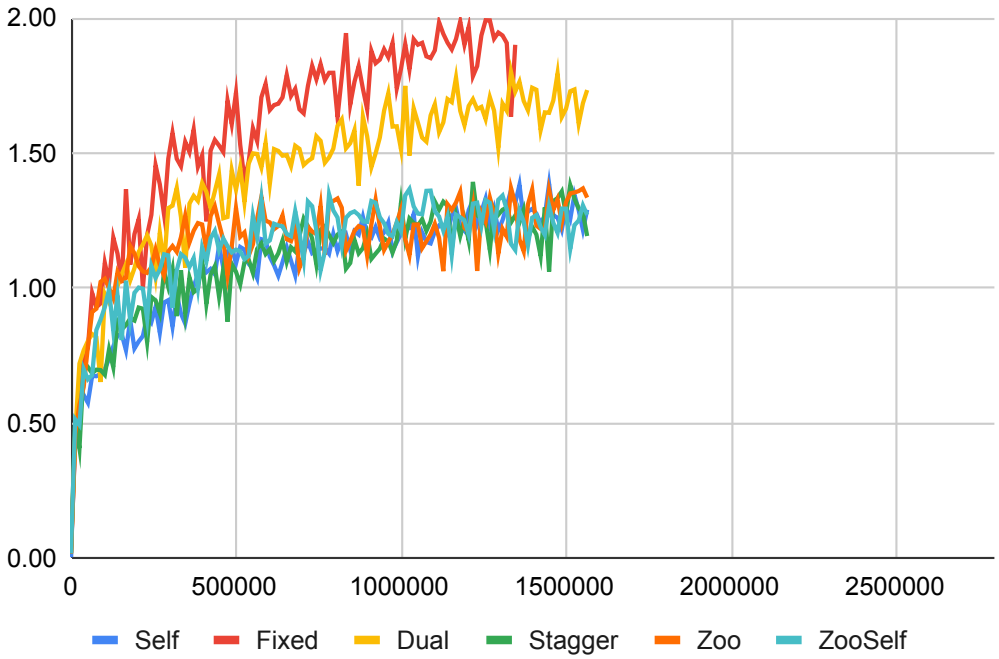

Figure A.8: Relative IMP Difference to Supervised model for different training regimens when starting from the supervised model. Higher scores are better.

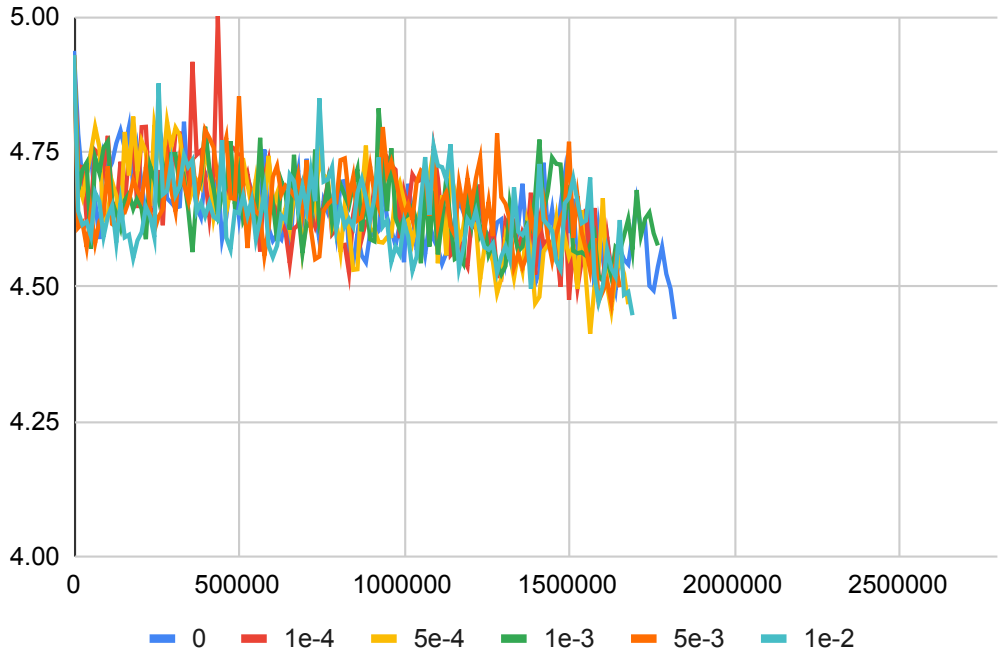

Figure A.9: Absolute IMP Difference to Oracle for different entropy loss weights when starting from the supervised model. Lower scores are better.

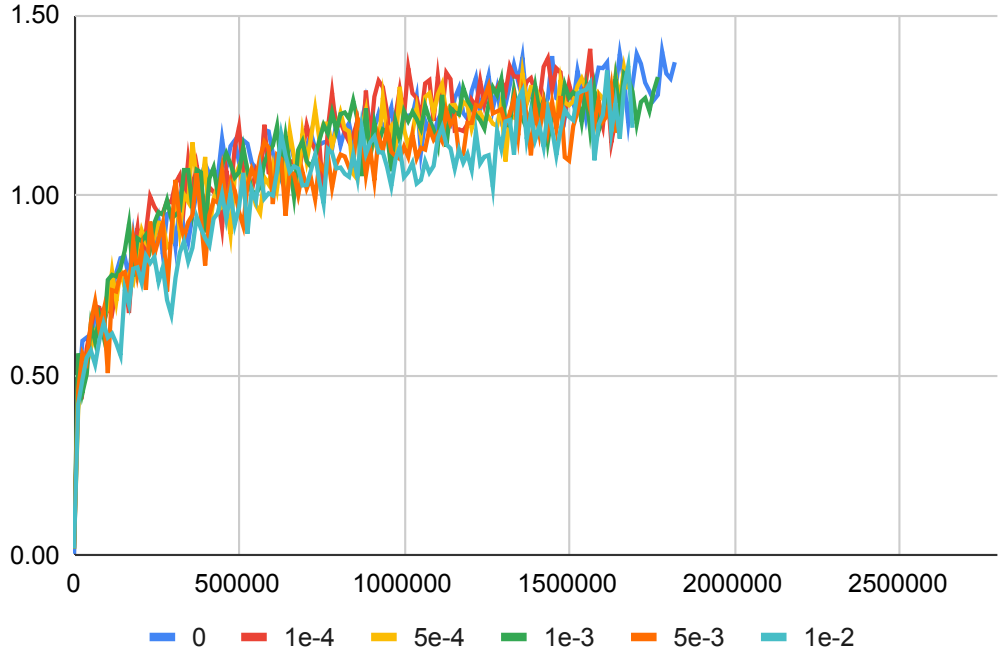

Figure A.10: Relative IMP Difference to Supervised model for different entropy loss weights when starting from the supervised model. Higher scores are better.

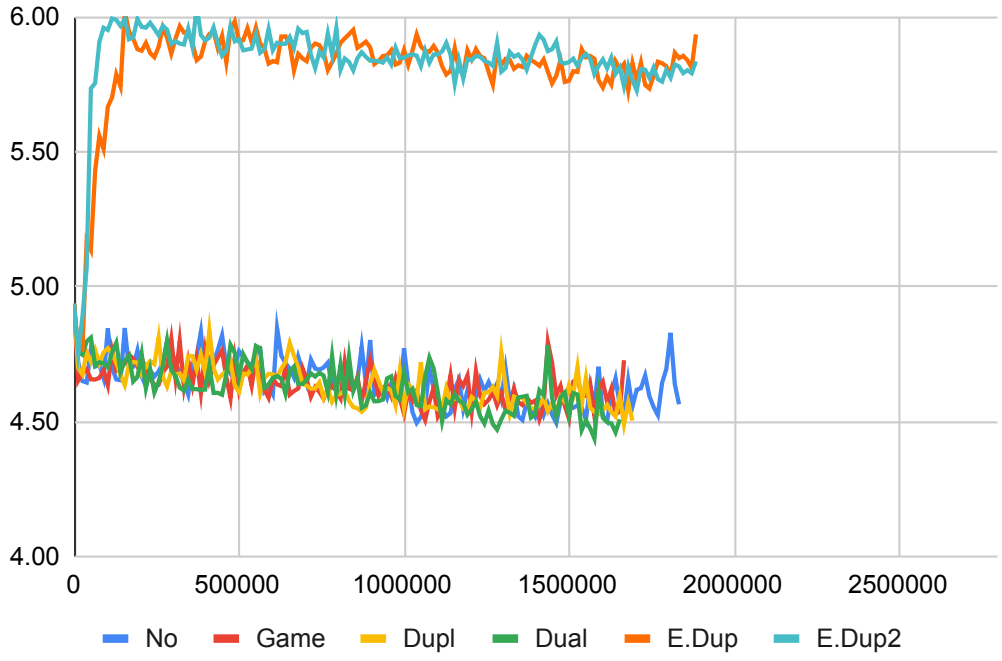

Figure A.11: Absolute IMP Difference to Oracle for different all-pass rewards when starting from the supervised model. Lower scores are better.

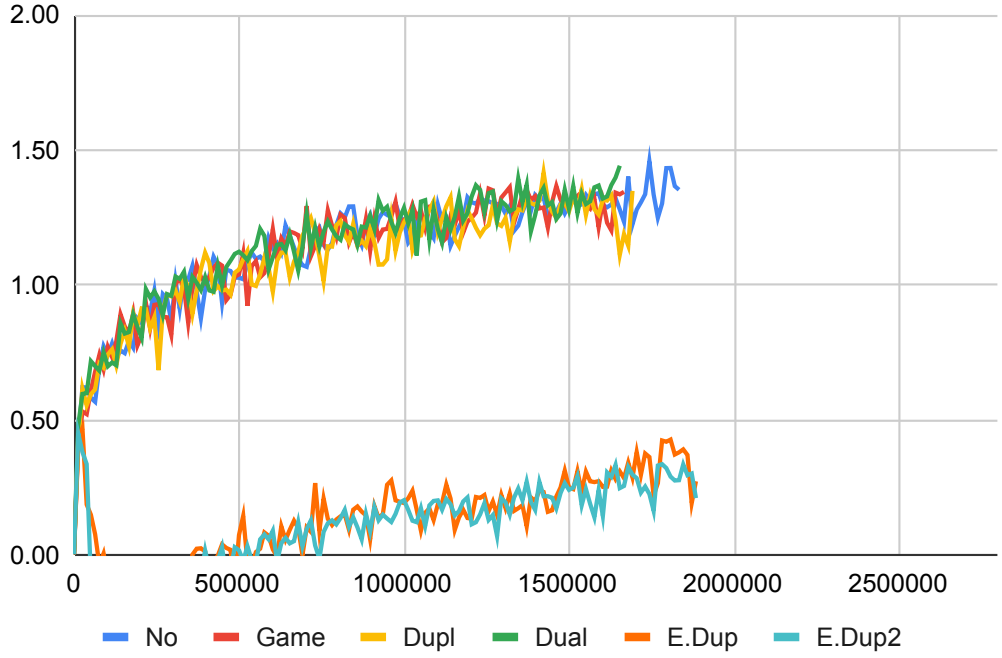

Figure A.12: Relative IMP Difference to Supervised model for different all-pass rewards when starting from the supervised model. Higher scores are better.

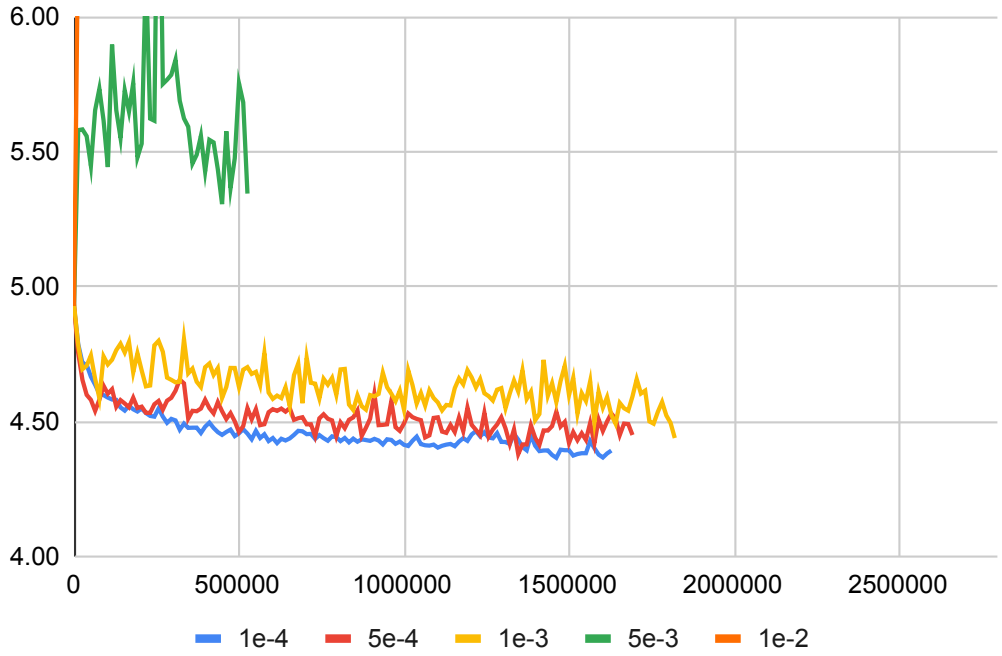

Figure A.13: Absolute IMP Difference to Oracle for different learning rates when starting from the supervised model. Lower scores are better.

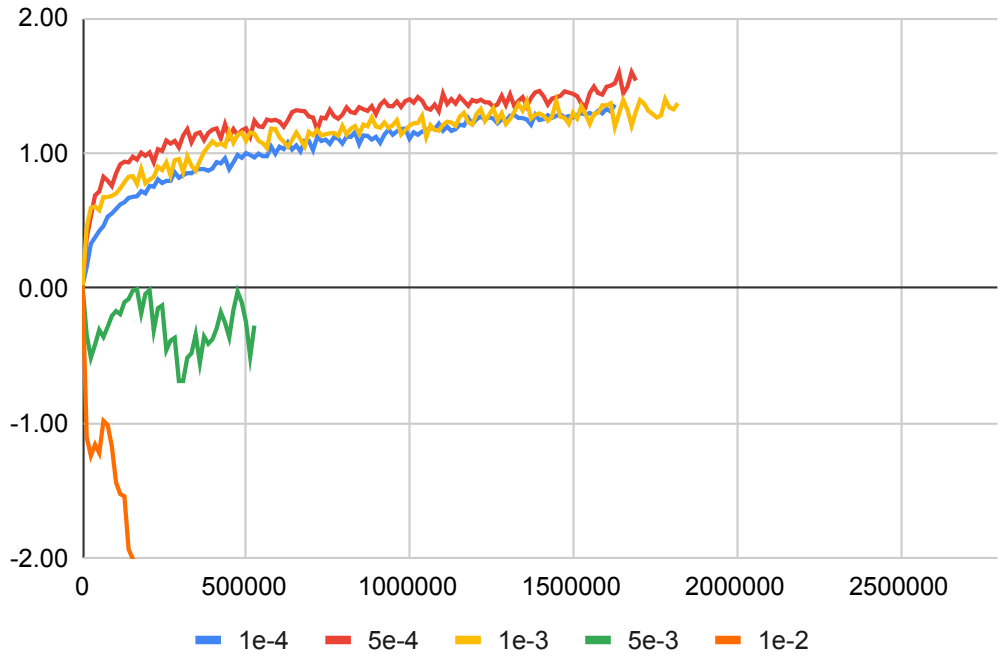

Figure A.14: Relative IMP Difference to Supervised model for different learning rates when starting from the supervised model. Higher scores are better.