UTRECHT UNIVERSITY

Artificial Intelligence Master Thesis

Predicting the outcomes of Automated Negotiations

First supervisor:
Tim Baarslag

Author:
Mick Camiel Tijdeman
Student number: 9486232

Second supervisor:
Shihan Wang

In cooperation with:
Centrum Wiskunde & Informatica

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science
in the

Department of Information and Computing Science

June 14, 2024

Abstract

Knowing the outcome of an automated negotiation before it is terminated offers many advantages. It allows one to change their strategy to obtain a better result, it gives (multi-issue) negotiators insight into which negotiations are worth the effort and resources and which are better to terminate, and it can aid in the development of negotiating assistants. Despite these advantages, no research has yet been done on predicting negotiation outcomes. This thesis tackles the challenge of predicting the outcome of automated negotiations. We limit the scope of these predictions to bilateral alternating-offer negotiations between two time-dependent agents.

Our method divides the prediction into two parts. First, the bids made by the negotiating agents are converted into utility time series. The future trajectory is then forecast by six time series forecasting methods. Second, these forecasts are used to find a distribution of the most likely intersection points and, with that, a distribution of the most likely negotiation outcomes. In addition, the network gives the probability of ending the negotiation with an agreement.

In general, neural networks performed best in both time series forecasting and outcome prediction, achieving an F1-score of 0.876. We found that negotiation time series are difficult to predict for classic statistical models, as the series have a low level of predictability. The predictability of time series can be improved by applying a behavior-based strategy, which we model by introducing a learning rate.

Our work shows that neural networks are a promising direction for automated negotiation outcome prediction. We believe these results are only the beginning of their capabilities and can further be improved by experimenting with more network architectures and negotiation settings. Therefore, we encourage other researchers to take the next steps to improve our results.

Contents

1. Introduction

Throughout history, negotiating has always been a staple of human decision making. Of all our modes of conflict management, it has proven to be one of the most flexible and effective tools for managing social and economic expectations [1]. As society evolved and human social circles grew bigger, the need to detect and protect oneself from potential threats and deceptions grew. At the same time, the need to be equally strategic in one's own dealings grew with it [2]. In fact, not all negotiations are quite as obvious as they seem at first glance. Even so, they still play an essential role in our daily lives [3]. Some applications are obvious, such as salary negotiations or marketplace haggling, and some are less obvious, such as deciding on a vacation destination with peers. The widespread prevalence of negotiating is not exclusive to humans either. Over the last decade, the field of *Automated Negotiating* has become an active area of research within computer science [4], [5]. Automated agents can alleviate much of the cognitive load associated with human negotiations, making them valuable assistants for human negotiators [6]. An added benefit of automated negotiations is that they are not influenced by psychological factors, as is the case with regular 'human' negotiations. Two of the same negotiating agents arguing over the same matters should come to the same outcome consistently (disregarding some programmed in randomness and agents purposely made to be erratic). In addition, computer systems often must negotiate with each other when dividing resources or coming to a mutual decision. Take, for example, energy contract negotiations [7], logistics and transportation [8], or even the communication of Mars rovers [9].

1.1 Predicting the outcome

While much effort has been put into implementing and designing negotiating agents and environments [10], as well as strategies [11]–[13] it is still largely unclear what tactics in an automated negotiation lead to which outcome and how an ongoing negotiation can still be influenced to steer towards different outcomes.

Gaining a stronger understanding of how to reach a certain negotiating result may prove paramount in both automated and normal negotiations. That is, being able to influence these outcomes gives an obvious tactical advantage to the party using it, steering the negotiation in a direction they deem preferable. Similarly, the negotiation could be directed towards a Pareto-efficient outcome, providing a mutual benefit for all involved parties. In addition, finding the optimal tactics or negotiation parameters may just be the required step to ensure that the negotiation does not terminate without an agreement at all.

Outcome prediction can also aid negotiators in one-to-many negotiations. Knowing where one of these negotiations is headed can inform decisions made in the other negotiations. Say, for example, a party is negotiating to buy resources in bulk from two different suppliers. If they already know that one negotiation is heading towards a good outcome, they may attempt to reach an even better outcome in the other negotiation and try their luck with a riskier strategy. Conversely, if they already know that one negotiation is heading south, they can attempt to ensure that the other negotiation at least ends in an agreement. An agent having insight on a negotiation's direction can also prove to be a good assistant for human negotiators, advising them on what offers result in which outcomes. These tactics may be applicable in any field that requires some form of negotiating such as politics, business, autonomous systems, and much more.

The mechanics underlying automated agents, algorithmic computations, are deterministic in nature. Given the same input, these should consistently give the same output. This is one way in which automated agents differ-

entiate themselves from human negotiators. Statisticians and computer scientists have already developed a wide range of tools for the prediction of various types of similar data, to great success [14]. These methods, when applied to automated negotiation, could prove worthwhile in improving the understanding of automated negotiations and add to the body of instruments available to improve agent behavior.

1.1.1 Key Concepts

An automated negotiation is any negotiation involving automated **agents**. These agents negotiate with each other in a setting that defines the context of the negotiation, a so-called **domain**. This domain defines a set of **issues** over which agents can reach a mutual decision, each issue having a set of **values**. Say, for example, two agents are negotiating over the details of a wedding. This would make the wedding the domain in which the agents negotiate. The issues in this domain may be the wedding location, the decorations, or the music. For the wedding location issue, some values could be a church, a forest, or a castle. All possible combinations of values over these issues together define the **outcome space** of the negotiation. When an agent proposes one of these combinations from the outcome space as a resolution to the negotiation, this is called a **bid**.

Each agent has a set of preferences over the possible outcomes. For every option within an issue, the agent has an associated numerical value indicating how much they prefer this option. This is the **utility function** of an agent, which defines the total **utility** the agent gets from a negotiation outcome.

Like real negotiations, automated negotiations can also take place in various contexts. The first important distinction is the number of agents that have to decide on the issues. In this thesis, we will focus on **bilateral negotiations**, which are negotiations between two agents. Another important distinction is the protocol used to guide the negotiation, which decides the rules about making and accepting offers. This research will focus on the **stacked alternating offers** protocol. This protocol has its agents alternate in sending bids

back and forth until one agent uses its turn to accept the bid of the other.

1.1.2 Time

Negotiations cannot continue indefinitely. Every negotiation has a deadline associated with it, either defined temporally (usually in seconds) or by means of a round limit. If time runs out before the end of the negotiation, both agents receive their reservation value. The reservation value is the minimum utility score obtained in every negotiation, which can be zero. Additionally, some negotiations apply a discount factor that penalizes negotiations that end late.

Under the threat of these penalties, time greatly influences the strategies of negotiating agents. Early in the negotiation, agents enjoy the space to try and influence their opponent, as well as gauge their strategy. Usually, this stage has agents reluctant to concede, while nearer to the end of the negotiation, an agent may be more inclined to settle for less. In addition, the choice of time unit can also affect the strategy employed by agents. Where a set round limit gives the agent a clear view of how much space for bids is remaining, a time limit leaves more uncertainty as every round may vary in duration.

1.1.3 Technical challenge

An agent in an automated negotiation strives to get the outcome that rewards it with the highest possible utility. However, since all agents must agree on the outcome, agents (usually) cannot simply get the result they most desire. Rather, they attempt to find an outcome that can satisfy all participants. Throughout the negotiation, agents (ideally) concede some of their utility in hopes of appealing to the utility of the other agent. A bid that cannot be further improved in utility for either agent without the other agent losing utility is called **Pareto efficient**. Multiple bids in the outcome space can be Pareto efficient, and together define the **Pareto frontier**. From an outside view of the negotiation with full knowledge of all utility functions, these concessions may look as in figure 1.1.

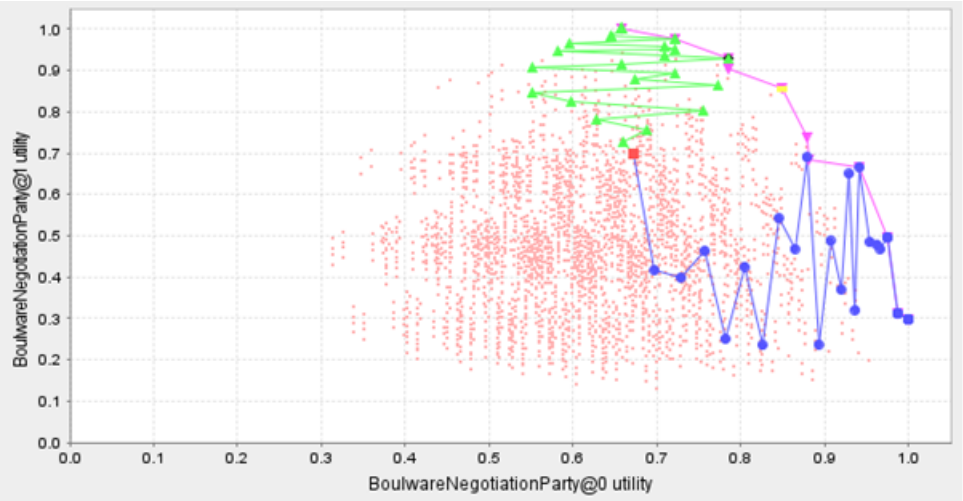


Figure 1.1: Outcome space as shown in the genius environment after a negotiation. Both axes show the utility of 1 agent. The blue and green lines show the bids made by the agents. In purple, the Pareto frontier.

As can be seen here, both agents may start at their best option (maximum utility) and then slowly explore other options while conceding small parts of their utility until an agreement is reached. However, an agent does not have this complete view, as they are only given their own utility function. Based on the opponent's bids, they may estimate their opponent's utility function using an opponent model. An opponent model is an approximation of the opponent's utility function. However, this opponent model is not guaranteed to provide an accurate estimation of the opponent's utility function. When taking into account the utilities belonging to a single agent within the negotiation, the negotiation can be modeled from the perspective of that agent only. This is done by modeling the utility received from their own concessions and the utility received from their opponents' bids over time. An example can be seen in figure 1.2.

In this example, the agent has a clear downward concession curve, while the opponent is seemingly trying a wider palette of offers, increasing both the minimum and maximum received utility for the offers and in turn widening the scope of the function. In essence, the task of predicting the negotiation outcome is about finding the point where the utility received from the opponent and the agent's own concession strategy intersect. In many cases, the agent tries to adjust their concessions based on their opponent by means of

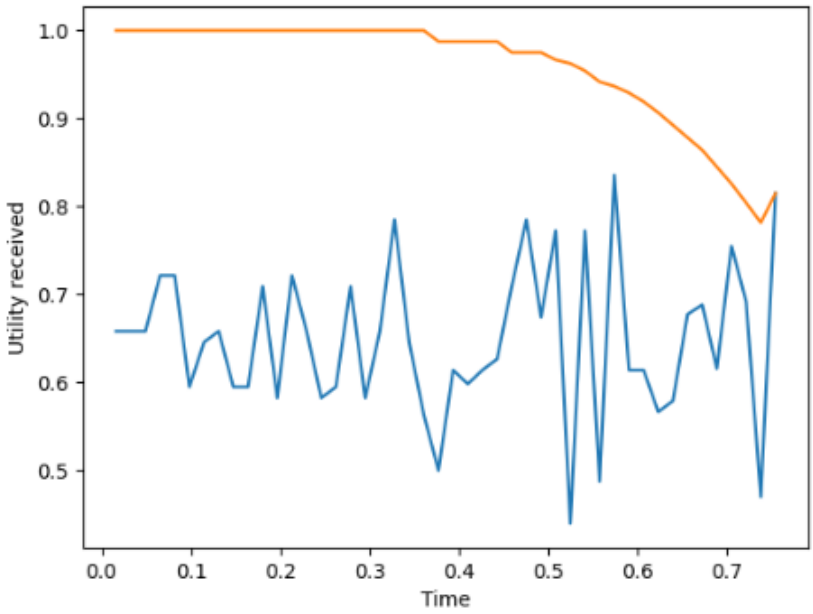


Figure 1.2: Utility curves of a single agent during a negotiation. In orange, the agent's own bids, in blue the opponent's bids.

an opponent model. If done effectively, this would cause the received utility from the opponent to trend upwards, such as in figure 1.3.

When the different factors of an automated negotiation are unpacked like this, what is left are utility graphs of the agents' bids over time. These are time series. As such, these series may be possible to predict by applying time series forecasting (TSF). A great deal of statistical research has already been dedicated to the forecasting of various types of time series [15]. We will build upon this knowledge and apply these techniques to the field of automated negotiations. If we can find the intersection of these forecasts, this will serve as a prediction of the final outcome of the negotiation. Taking into account the perspective of a single agent, this comes down to:

1. Using the utility concession of our own agent as a single time series, forecast the future trajectory of this series.
2. Using the utility received from the bids made by the opponent as the other time series, forecast the future trajectory of this series as well.
3. Predict the outcome by finding the intersection between both time series with an intersection method. This outcome consists of a utility prediction and an agreement probability prediction.

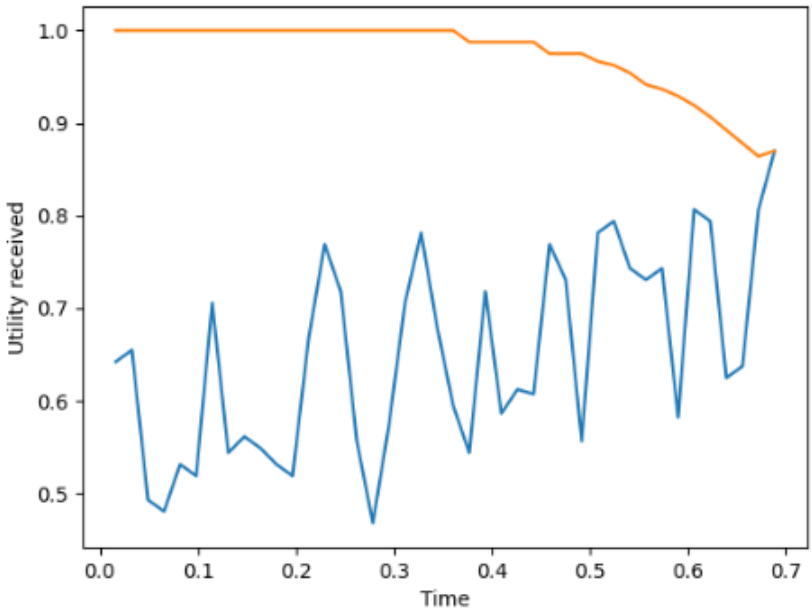


Figure 1.3: Utility curves of a single agent during a negotiation. In orange, the agent's own bids, in blue the opponent's bids. The opponent's bids trend upward, indicating a working opponent model from the other agent.

1.2 Existing Approaches

Predicting future bids in an automated negotiation has been attempted before. Thus far, similar research has primarily focused on predicting the opponent's next bid, usually with the aim of increasing performance for the agent. Williams et al. do this using Gaussian processes [11]. Chen et al. combined Gaussian processes with transfer learning to predict bids [12]. Yesevi et al. employ an LSTM and transformer network to predict future bids from the opponent [16]. Another direction is taken by researchers who attempt to predict the opponent's strategy. Hou uses nonlinear regression analysis to predict the behavior of negotiation agents based on the Boulware-Conceder model[17] as introduced by Faratin et al. [4]. Similarly, Sengupta et al. use strategy templates with learnable parameters to estimate opponent tactics [18]. On top of this, they also use deep reinforcement learning to estimate the opponent's threshold utility. Li et al. opt for using a deep learning-based approach to predict the opponent's strategy by feeding time series data into an RNN [19]. Although not predicting the outcome itself, Ilany and Gal predict the performance of negotiation algorithms for different do-

mains so that the best strategy can be chosen [20]. Evidently, there have already been many different types of research regarding the prediction of (opponent) agent behavior. However, this research has not yet focused on predicting the outcome of the negotiation itself, most of it rather opting for short-term predictions. Therefore, tackling this issue is a novel endeavor within the field of automated negotiations and is a worthwhile addition to the body of research. These approaches will be discussed in more detail in chapter 2.

1.3 Research Question

The research question at the core of this thesis is:

Research Question

How can we best predict the outcome of an automated negotiation?

We address this problem by developing a framework for the prediction of automated negotiations. This framework consists of two smaller steps that together form a solution to our problem. The first step here is to forecast the agent's utility graphs. Each agent has their own utility graph that contains the utility of their bids thus far. This information can be used to forecast the future bids both agents will make by using time series forecasting techniques. Generating these forecasts is the first component of our framework. As such, the first sub-question is the following:

Research Question 1.1

How can we best predict the utility time series of the agents participating in a negotiation?

When these individual utility graphs can be confidently predicted, the next step in our framework is to use the forecasts to estimate where the graphs will intersect. Presumably, the intersection of the graphs gives us the final outcome of the negotiation, since both agents agree that this utility is deemed an acceptable outcome. When given a probability distribution with the prediction of the utility graphs, multiple plausible outcomes of the negotiation can be predicted. In this case, the negotiation outcome predictions can also be given a confidence measure to indicate how likely this outcome is, as well as an agreement probability score. This makes the second sub-question:

Research Question 1.2

How can we best predict the outcome of an automated negotiation based on time series forecasts of utility graphs?

This thesis will investigate to what extent such a framework can be used for the prediction of automated negotiation outcomes. This will be done by applying multiple state-of-the-art forecasting methods to the framework and evaluating their performance in outcome prediction.

1.4 Thesis Outline

Chapter 2 gives an overview of the relevant literature and places the current research in the context of the wider field of (automated) negotiation. It also introduces and explains key concepts used within the Thesis. Chapter 3 goes into detail about our proposed solution and gives a technical overview of the solutions. Chapter 4 discusses the main experiments carried out in the thesis, as well as some preliminary experiments. Chapter 5 discusses the results of these experiments in depth. Chapter 6 places the results in a larger scope and discusses the limitations of the research. Chapter 7 gives our final thoughts on the research, as well as suggestions for future research directions that may prove worthwhile for automated negotiation prediction.

2. Literature Review

This chapter introduces the literature from which this project draws inspiration. It also gives an overview of research on related topics. The relevant literature for the project includes other research into making negotiation predictions. This includes predictions of various aspects of negotiations, but especially research on bid prediction for automated negotiations is relevant. This is because it faces similar challenges and uses similar solutions. The technical literature important for our methodology will also be introduced.

2.1 Predicting Classical Negotiations

Attempting to predict the outcome of a negotiation before it is terminated is not a novel endeavor in itself. Negotiations are a game of strategy where information is key, so naturally any additional source of information is highly valued. Knowing the potential outcome of a negotiation gives a tactical advantage to the party using it, and thus has been thoroughly studied for the classical "human" case. Van Poucke et al. point out several 'reference points', factors that influence the final outcome of a negotiation [21]. The research makes a distinction between two types of reference points. Reference points that are outside the control of the negotiators, such as market value, are called external reference points. Internal reference points, on the other hand, are those set by the negotiators themselves. They include the reservation & aspiration prices and the opening offer. The reservation price is the lowest offer that a negotiator is willing to accept. Here, the negotiator is indifferent about the offer, as it neither improves nor decreases its utility. The aspiration price is the highest outcome for which a negotiator aims that has a non-negligible probability of being accepted by the other negotiator. Thus, it is the best outcome that can reasonably be expected. Van Poucke finds that, in human negotiations, opening offers account for 57% of the

variance in the outcome of negotiations. Since this research deals with automated negotiations in a simulated environment, external reference points are ignored since everything takes place within the confines of a simulation program. Therefore, the negotiating agents have full control over the outcome. However, simulations do contain parallels for the internal reference points. Many negotiating agents set a reservation value themselves in the form of a utility that they will not go under. The opening offer is predefined by the domain used in automated negotiations, as agents often start with whichever bid gives them the highest utility. Despite these parallels, it is questionable at best whether the same theory about negotiation outcome prediction can be applied to automated negotiations. Classical negotiations rely heavily on human psychological processes for their outcome [22], [23], while fully automated negotiations strip the human aspects of a negotiation. An automated agent will not be as influenced by the psychological effects of anchoring or perspective taking as humans often are [24], [25].

Moosmayer et al. tried their hand at using a neural network to predict classical negotiation outcomes and study the relationship between reference points and outcomes of business-to-business price negotiations [26]. They found that neural networks provide a better tool for these predictions than linear regression models, as neural networks were able to capture the nonlinear relations between reference points and the outcome of a negotiation. This shows that there is potential for machine learning-based methods in negotiation prediction, largely because of their ability to find patterns and nonlinear relations in large data sets. Although again hard to compare to our case, it is plausible a similar approach can extract reference points or other values from an automated negotiation to predict the outcome.

Connecting the realms of automated and classical negotiations, there is a growing body of research on automated agents that negotiate with humans in natural language. For example, to study the effects of COVID-19-related stress in negotiations [27]. In an attempt to improve such natural language bots, Chawla et al. analyzed the language used in bilateral buyer-seller negotiations and used this data to train a prediction model (BERT), which attempts to predict the outcome of these classical negotiations [28]. Future

negotiation systems may be able to combine the research of Chawla et al. and this thesis to make accurate predictions for agent-assisted classical negotiations.

In general, classical and automated negotiations are difficult to compare, and thus the methods that work for the prediction of classical negotiations cannot be applied to automated negotiations as such. However, they can inspire prediction methods for automated negotiations.

2.2 Predicting Automated Negotiations

As is the case for classical negotiations, automated agents also benefit from a surplus of information. Because data can be extracted from automated negotiations relatively easily, there are various studies which use this information to make predictions about automated negotiations. A smaller step in predicting a complete negotiation is predicting the opponent's offers. Since being able to predict the coming offers of an opponent can prove a worthwhile tactical advantage in a negotiation, multiple methods have been developed for offer prediction. The methods used here may be useful in our research as well, as a successful offer prediction can be a stepping stone to predicting the negotiation outcome.

One of the older attempts at predicting bids was done by Carbonneau et al., who used pairwise issue modeling to predict the counteroffers made by opponents. This was done by training a network to use negotiations similar to the current to predict the next negotiation offer [29]. Williams et al. used a Gaussian process to estimate the concession rate of an opponent [11]. This was done by recording the best utility, according to the agent's own utility function, offered by the opposing agent within a set window of time (e.g., the best offer between t=0 to t=5, t=5 to t=10 etc.). These best offers should ideally continue to get better with the opponent's concession. Williams argued that only the best offers have to be recorded instead of every offer, as the best offer indicates the highest utility the opponent is willing to settle for, and thus is guaranteed to be achievable. This data is then used to model future concessions with a Gaussian process, which gives

both a prediction and a measure of confidence in this prediction. Williams does this for a discounted negotiation setting, as the concession rate is of higher importance here than in a regular negotiation. An agent integrating this method into their opponent model managed to achieve a good score in a tournament against other agents. Although not exactly applicable to predicting a negotiation outcome, Williams' intuition to only consider the best offers made by the opponent and to integrate a confidence interval in their analysis proved useful for their case. Chen et al. also apply Gaussian processes, but use them in combination with transfer learning to improve agent learning behavior [12].

Another method for predicting opponent bids is explored by Yesevi et al., who employ two deep learning-based approaches [16]. Namely, a long short-term memory (LSTM) network and a Transformer network. They use a sequence of utilities to feed into these networks using a sliding window approach, where the size of the window determines the number of past bids included in the input. For each method, a secondary network is trained which considers an opponent model in its input as well. This is done to investigate whether its utility predictions improve when auxiliary information is taken into account. Another network, with inputs similar to the latter, is trained for the purpose of predicting what utility the opponent gets from their own bids. The networks are trained separately for different-sized domains (small, medium, and large), plus one network trained on all sizes with 600 different negotiations. An evaluation was done on the corresponding domain sizes for 60 negotiations. Notably, the general network performed better than the domain-specific networks, achieving an RMSE of 0.08. Moreover, using additional inputs slightly improved performance, indicating that an opponent model can be beneficial for such predictive networks. There was no significant difference between the performance of the transformer and the LSTM network. These methods give a good indication of the possibilities in feeding machine learning algorithms the traces of bids made to predict future bids. An important distinction, however, is that these methods only aim to predict one bid ahead, whereas this research aims to predict the complete negotiation.

It is not just bids that can be predicted. Many researchers attempt to predict the opponents' strategy and adapt accordingly. One of the first attempts to recognize strategies in automated negotiations was made by Hou [17]. Hou applied a nonlinear regression analysis to discriminate between three families of negotiation tactics (also see 2.2.1), and predict the tactic used by the agent. For the time-based tactic, for example, the model differentiates between conceder, boulware, and linear models. This work is similar to that of Brzostowski et al., who also use a nonlinear regression analysis to predict the behavior of opposing agents [30]. They, however, use four models from which to choose when predicting the opponent's strategy. As early examples of such behavior prediction, these articles already show an improvement in performance for agents using a regression analysis.

In a more recent paper, Li et al. use a recurrent neural network trained on negotiation time series [19] to predict strategies, quite similar to the approach of Yesevi et al. They compiled a large data set of negotiation time series by having their own agent play with a Tit-for-Tat strategy against opponent agents who may play any strategy. The time series belonging to the opponent is saved together with the strategy applied by the opponent. These time series-strategy combinations are then used to train a recurrent neural network with an LSTM cell of 64 extracted features. The network can be used to recognize the strategy that belongs to a time series from a total of ten distinct strategies. The choice for a recurrent neural network is made as they are well suited for handling sequential data. The trained network achieves promising results overall, although still scores low accuracy for certain strategies. Even more important than the opponent's strategy perhaps is the strategy choice of your own agent. To improve strategy selection, Ilany and Gal devised a meta-agent that predicts the performance of different negotiation strategies while the negotiation is running [20]. Their agent uses the multi-armed bandit algorithm to test different negotiation strategies at runtime and choose the best strategy for different opponents. The meta-agent was able to pick the best strategy against many of the finalists in the ANAC negotiation competition.

In general, a clear trend can be seen for the prediction techniques that make

Challenge	Research
Predicting Classical Negotiation Outcomes	Van Poucke et al, Moosmayer et al,
Predicting Hybrid Negotiation Outcomes	Mell et al, Chawla et al.
Predicting Automated Negotiation Offers	Carbonneau et al, Williams et al, Yesevi et al.
Predicting Automated Negotiation Strategies	Hou et al, Brzostowski et al, Li et al.
Predicting Automated Negotiation Outcomes	This Research

up the related work. Most of them analyze the time series consisting of the bids made by the negotiating agent, and use these to train a prediction algorithm. The more recent research often chooses to use a sequential deep network such as an RNN or LSTM, as these types of networks have booked recent successes in many domains, especially those dependent on time series data. Therefore, sequential deep neural networks will also be chosen as our primary method for predicting negotiation outcomes.

2.2.1 Negotiating Strategies

Faratin et al. defined 2 general tactics in automated negotiations [4]. *Time-based strategies* and *behavior-based strategies*. In a time-based strategy, the agent will base its next bid on the amount of time remaining for the negotiation, often opting to explore more options when time runs out. In a behavior-based strategy, the agent will base its bid on the opponent's bids. A well-known example of this is Tit-for-Tat, where the agent will behave cooperatively against a cooperative opponent and vice versa. Keskin et al. propose a third class of strategies, namely one that combines both time-based and behavior-based strategies [31]. Although these three categories are the highest-level classification of strategy types, more strategy types can theoretically be defined depending on the criteria used. Some strategy types are easier to predict than others; thus it is important to consider which types to include in this research.

2.3 Time Series Forecasting

A good candidate for predicting negotiation outcomes is time series forecasting. This is because the bids made by the agents can be modeled as a series of data points ordered in time, as in [11], [16]. These data points can then be used to train an algorithm to forecast the agents' future utility graph. Various methods exist with which to perform TSF. Generally, most methods can be categorized as either classical statistical models or machine/deep learning models. In this research, both types of methods will be tested.

2.3.1 Characteristics of Time Series

To properly analyze a time series, its characteristics must be carefully considered, as these strongly influence the optimal forecasting method. There are several common traits for a time series that are often considered in quantitatively describing a time series. Some of these terms can be somewhat counterintuitive, so it is good to get comfortable with these terms.

First, **trend**. Arguably the most straightforward property, trend describes the directionality of the time series. Usually, a trend in the data can be noticed with simple visual inspection but is also detectable by analyzing the long-term behavior of the data. If the mean shows a clear change over a long period, this implies a trend in the data that may be upward or downward. Trend can be linear, but it does not have to.

Second, **seasonality**. A time series is seasonal if it shows a regular change in behavior with a set frequency. For example, something happens every year, week, or 10 minutes that affects the value of the time series. Important here is that this frequency is constant. If not, the time series may instead be cyclic. A **cyclic** time series is one that shows regular changes in an oscillating pattern that is not related to any frequency.

Third, **stationarity**. A stationary time series is one whose statistical properties, usually its mean and variance, do not depend on the time at which the series is observed [32] and thus do not have any trend or seasonality. Note that a cyclic time series can in fact be stationary, as the cycles appear

at (seemingly) random moments, and thus before we observe the cycles we cannot be sure when they will appear. For a number of applications in time series analysis, a time series must be made stationary to be analyzed. This can be done by differencing the time series. A time series can be differenced by computing the differences between consecutive observations. These differences form the new time series. It is possible that this newly created time series is still not stationary. In this case, the time series is differenced until it is no longer stationary. To test whether a time series is stationary and how often it must be differenced to be made stationary, a *Unit Root Test* can be used, as well as analyzing an *Autocorrelation* (ACF) plot [32], [33].

2.3.1.1 Predictability of Time Series

To be able to apply any time series forecasting method to a time series, the time series must be predictable. Whether a time series can be predicted depends on some key characteristics and is not always clear from a simple visual inspection, nor is it a binary condition. Rather, the predictability of a time series exists on a continuous spectrum. Due to the diversity in both types of time series and methods to predict them, the latter still increasing with the development of neural models, most attempts to quantify predictability have been highly specific to certain fields. Take, for example, the macroeconomic approach of Diebold et al. [34] or the ecological approach of Pennekamp et al. [35]. Xu et al. recognize the difficulty in accurately gauging the predictability of (highly random) time series and pose an open challenge for future studies to tackle this question [36]. While the question of time series predictability (for automated negotiations or otherwise) will not be solved within the pages of this thesis, it is good to have some indication of what does or does not constitute a 'predictable' time series.

A basic yet robust test of the predictability of a time series is whether it can be described with a 'random walk' model [37]. Most famously used as an argument against trying to predict the stock market [38], a random walk model describes a time series that can move in multiple directions (in our case up and down) with some probability p. This probability is often modeled as a series of white noise $w(t)$. Unlike a simple series of random num-

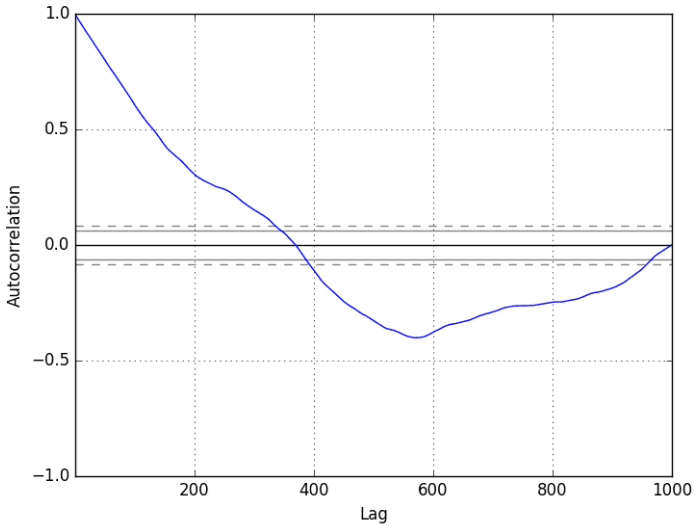


Figure 2.1: Autocorrelation plot of a random walk time series with 1000 steps. Image adapted from [39]

bers, a random walk depends on its previous value $y(t-1)$to determine the next. Thus, a random walk can be described by: $y(t) = y(t-1) * w(t)$.

This randomness makes such a time series difficult to predict as the next value the series will take cannot be inferred from any of the available data. As such, it is important to verify that our series are not random walks. Although easy to describe, determining whether a time series is a random walk can be quite challenging. Several tests can be run to determine whether a time series can be classified as such. The primary test is that of autocorrelation. An autocorrelation plot describes the correlation between an observation at a time step and the previous observations at earlier time steps. It describes how well an observation can be inferred on the basis of other observations. Since the value of an observation in a random walk is dependent on its previous value, an autocorrelation plot will show a high autocorrelation with time step $t-1$ and show a linear fall from that point on. A typical autocorrelation plot for a random walk may look like figure 2.1.

Additionally, all random walk processes are non-stationary. This is, once again, because the current observation is a random step from the previous observation. A non-stationary time series does not have a consistent mean over time, which also serves as an indication of a random time series.

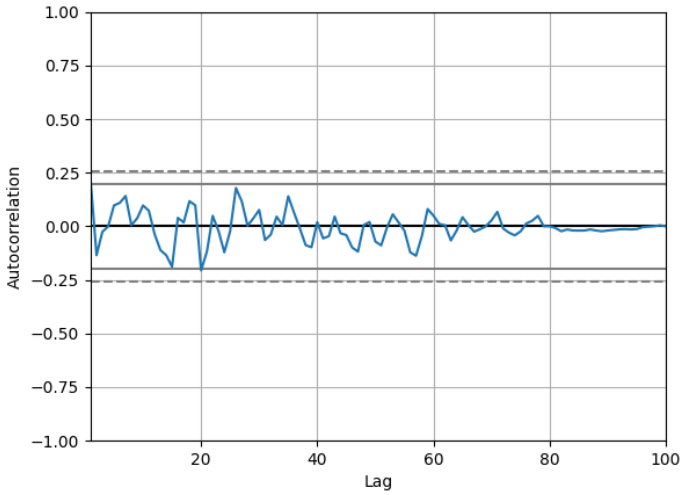


Figure 2.2: Example of a white noise autocorrelation plot. Notice how all autocorrelation are underneath the significance line at +- 0.2.

However, that is not the only thing an autocorrelation plot can detect. Another important property to look out for when assessing the predictability of a time series is whether it shows characteristics of white noise. A white noise time series draws every step from a random distribution. It has no correlation between time steps at all, and thus the autocorrelation plot will show no strong correlation for any range. As such, an autocorrelation plot for this type of time series will show that most autocorrelations are close to zero [32]. See image 2.2 for an example of a white noise autocorrelation plot.

A time series showing random walk or white noise characteristics is not unpredictable per definition. However, it is difficult to do this with classical autoregressive methods, as these methods only consider other observations within the same plot. However, there may be other indicators outside of the plot itself that can assist in predicting future values of the series. A common method to test this is Granger causality. A signal "Granger-causes" a different signal if past values of one can inform future values of the other [40]. Granger causality is, however, still highly dependent on the time dimension (only observations earlier can predict observations later), and lacks in its ability to find nonlinear relations. Makridakis et al. showed in their M-competition paper that deep learning models scored best for noisy, nonlinear, and trended data [41]. So, for data that seems to have low predictability,

deep learning may provide a solution.

2.4 Deep Learning

Deep learning is a subsection of machine learning techniques that has gained much popularity in many different fields over the last decades. From the recent boom in generative AI [42], [43], to image and speech recognition [44] or even genomics [45], deep learning appears to be embedded in most technology in some way or another. There are many types of deep learning models and architectures, but they all share the same basic building blocks. The most basic deep neural network is the (feedforward) multi-layer perceptron (MLP) [46]. An MLP consists of many small units called neurons. A neuron receives input, makes some calculation (usually a summation) with the input, and passes the result of this calculation through some activation function to generate an output.

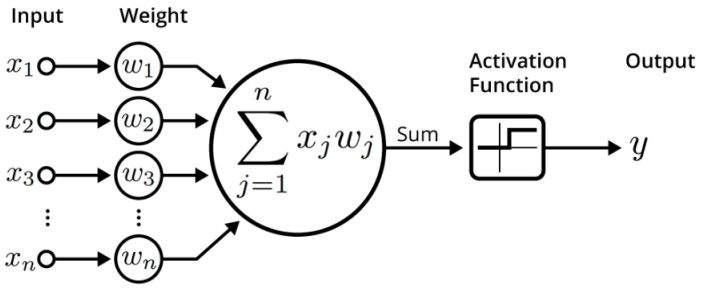


Figure 2.3: Schematic view of a Neuron. Image adapted from [47].

In a DNN, these neurons are connected in layers to perform complex calculations. The structure of a typical MLP has a series of input neurons through which the data is fed first. These input neurons connect to neurons in several (usually fully connected) hidden layers after which they reach the output layer. This output layer makes the final prediction based on the data that is fed into the network and computed by the individual neurons. A large neural network may have up to millions of neurons in the network. The connections between neurons all have an associated weight. This weight is a multiplication factor that decides how strongly one neuron that is fed into another influences this neuron. Next to the weights are the biases, an addi-

tive factor that is added to the neuron's calculation. The weights and biases are the part of the network that is updated when the network is "learning". It is learning the optimal values for the weights and biases to generate a correct output.

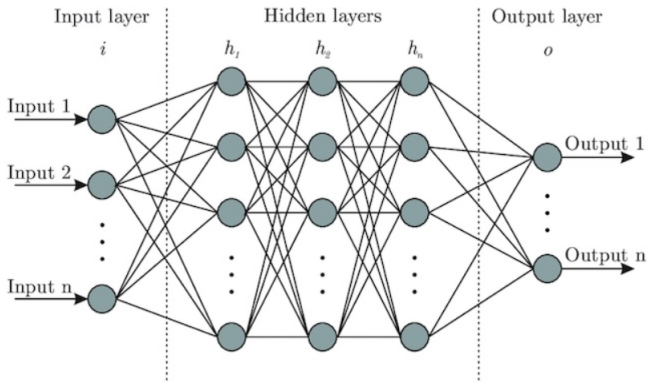


Figure 2.4: Schematic view of a Multi Layer Perceptron. Image adapted from [48]

The activation function was already briefly mentioned. It is a mathematical function that transforms the output of a neuron. There are many different ways to do this, the most important feature is that it introduces a nonlinearity in the network, as the simple summation function with weights and biases is still only a linear function. Due to this, the network can generate nonlinear outputs, which are required for most classification tasks.

A supervised DNN trains by comparing the output to the true label by means of a loss function. This loss function represents the error between the generated and true outputs. It is this error value which must be minimized by the learning process. There are many loss functions, and the one used is mostly dependent on the type of output generated by the network. The error value is minimized, and, in turn, the weights and biases are updated by a process called backpropagation. This process will be explained further in section 3.2.4.

2.4.1 Deep Learning for TSF

Time series forecasting is no exception to deep learning's newfound glory. The most recent M-competition, a competition set up by Makridakis et al.

that demonstrates the latest advances in TSF, saw that on average deep learning models outperform statistical models [41]. These results still are highly dependent on the content of the data. However, in general, the advancement of deep learning models is undeniable. Although classical statistical methods are driven by domain expertise, modern deep learning methods can find patterns in time series and predict them based on a purely data-driven method [49]. Because of this, deep learning methods are less dependent on the precise structure of data, being able to adapt to various data structures and characterize this data. This, combined with the abundant availability of libraries and software packages for deep learning and the steadily increasing amount of data available in nearly all domains [50], [51] makes deep learning a viable contender for many data prediction endeavors. This also applies to our research. A desirable quality of deep learning is its ability to learn from previous prediction results and apply this to future predictions, whereas statistical methods can only take into account the time series they are currently predicting. Classical TSF methods rely on the careful tuning of a model, taking into account statistical parameters like trend and seasonality to design the perfect model for the time series at hand. This is not only labor-intensive, but must be done for each individual time series. When there is one long time series that must be predicted, such as the sales of a certain product over many years, this is not a problem. However, one may imagine many cases in which it is desirable to predict many related time series, rather than one big one. This is also the case for this project, where a large amount of relatively small but related time series must be predicted. These time series can behave quite erratic, seemingly lacking a clear pattern or trend within the data at first grasp. This is why learning is an important feature, as it is only with the experience of many negotiations that the behavior of the agents and the shape of utility graphs can be understood better. Ideally, a deep learning predictor should start to understand these patterns throughout different negotiations and apply its gained knowledge to predict novel ones. However, these data-driven methods are also not without drawbacks. The complexity of deep learning networks makes them more involved in setup and training, and flaws in the

network are considerably harder to detect and solve.

2.4.2 Long Short-Term Memory

In domains dealing with temporal data, recurrent neural networks (RNNs) have become the de facto deep learning tool for prediction problems [52]. RNNs set themselves apart from regular deep learning networks by their cyclic architecture. Where a classical neural network feeds information forward through its layers, an RNN introduces lateral and backward connections between neurons. Because of these connections, the flow of information within an RNN is multidirectional. This makes a trained RNN especially well-suited for sequential data, such as time series. The field of natural language processing has also seen improvements after the adaptation of RNNs, particularly the improved-upon version of an RNN, the long short-term memory or LSTM network [53], [54].

LSTMs are an improvement on RNNs, as they solve the vanishing gradients problem. When training a neural network using backpropagation, the gradients that are used to calculate the derivative and update the weights of the networks can become vanishingly small. Therefore, the network cannot be trained anymore. LSTMs solve this by introducing memory gates in their architecture. For brevity's sake, we will not go over their functionality in detail. Importantly, the last decade has proved fruitful for LSTM research, achieving results in various domains [55]. Their gated architecture makes them able to capture far more nuanced properties of the data on which it is trained. However, they are not wholly without problems. Like most deep learning methods, LSTMs take up a lot more memory than simple statistical methods. This is especially true here, as LSTMs have a more involved architecture, with more parameters to train. This in turn causes the training process to also use more computational power and makes the model more difficult to interpret than a simpler RNN. On top of this, the large number of parameters also makes the network prone to overfitting, making it perform very well on the training set but poorly on any data outside the training set. Moreover, LSTMs, while solving the vanishing gradient problem, can

still suffer from the complementary 'exploding gradient' problem, where the values for the gradients get too large. This results in the model being unable to improve its parameters and learn.

2.4.3 GluonTS

GluonTS is an open source library for deep learning-based time series modeling developed by Amazon Web Services [56]. GluonTS itself is not a deep learning model, but rather a tool for using other deep learning models and developing your own. It comes with a large library of pre-made models based on architectures from various deep-learning papers, including RNN, LSTM and Multi-Layer Perceptron (MLP) models. These models can be trained on custom data sets to adjust the associated weights. Because the tool is made specifically for TSF, it greatly simplifies the creation and testing of models. Its models are specifically made for probabilistic forecasting, meaning that they will give a distribution as output for its prediction, rather than just a single point in time. The different distributions available include common parametric distributions, such as Student's T, Gaussian, gamma, and negative binomial. To learn a distribution, the model needs to learn the associated parameters of the distribution. For example, if a Gaussian distribution is generated, the mean and variance of this distribution must be learned. To achieve this, every distribution comes with a special class that maps the output of the model to the parameters of the distribution, comparable to introducing an extra layer on top of the network. The distribution parameters are then optimized along with the rest of the model, and thus it learns the distribution parameters for each time step in the series.

2.4.4 DeepAR

Of the many models with which GluonTS comes, DeepAR is the best candidate for the problem at hand [57]. DeepAR is a recurrent neural network with some LSTM functionalities that builds on previous work in probabilistic forecasting, and was found to be the best performing deep learning model in the M-competition [41]. It is most applicable for large data sets of

related time series. For such cases, it learns by training on these related time series jointly and as such generalizes to be able to predict similar time series accurately. Salinas et al. describe four advantages that separate DeepAR from other (deep learning) forecasting methods:

1. The model learns dependencies and seasonal behavior for data with multiple variables, such that no manual interference is necessary to capture group behavior.
2. The forecasts are made in the form of large sets of Monte Carlo samples. These samples are in turn used to compute quantiles and probabilities.
3. As DeepAR is trained on big data sets of time series, it can make proper forecasts for time series that have little own data available, whereas single-item forecasts would fail at such a task.
4. The model can incorporate a wide array of likelihood functions, allowing the user to choose whichever is most fitting for the data.

As this project does not deal with multivariate data, the first advantage does not apply here. The second point is advantageous for the project since the Monte Carlo samples can be used to determine potential intersection points of the graph, also see section 2.5. The third point is what sets this method apart from classical forecasting methods, as early in a negotiation there is little data to base a prediction on. A method that is still capable of making accurate predictions despite this limitation is desirable. Lastly, the fourth point allows us to experiment with the optimal likelihood function to best fit the negotiation data. On top of these four advantages, DeepAR also shows that it performs better than other deep learning forecasting methods, has an open source repository available for inspection, and enjoys regular updates from its developers.

2.5 Monte Carlo Methods

Monte Carlo methods (MCM) are a broad class of computational algorithms that use repeated random sampling to obtain numerical results, often used

when there is no clear analytical method available. MCM are commonly used in collision prediction, where trajectories of particles or vehicles are simulated to find the probability of collision with their environment [58], [59]. In the problem at hand, this would entail sampling from the forecast distributions a large number of times, and finding the intersection points between the samples belonging to the two agents. MCM are a good option for simulating the negotiation when statistical methods cannot suffice. They have already seen great success, especially now that computational power is increasing across the board [60]. MCM does, however, have the obvious disadvantage of being potentially computationally heavy, as running many simulations takes a lot of computing power. Thus, MCM can only be as effective as computational resources and efficient programming allow.

3. Predicting the Negotiation Outcome

In this section, the challenge of predicting automated negotiations and our solution will be further expanded upon. In particular, the workings of the full prediction algorithm are highlighted, showing the entire process from predicting time series to generating a utility prediction.

From input to output, generating a time series prediction consists of several steps. First, the negotiation bids are converted to time series. Then, the future trajectory of these time series will be predicted using a TSF method. For both time series, it generates a utility prediction for all remaining time steps in the form of a probability distribution. Monte Carlo methods are then used to generate samples from these distributions. An intersection method is applied over the resulting Monte Carlo samples, which finds both a probability distribution for the final negotiation outcome and a binary probability describing the chance that the negotiation ends in an agreement or a breakoff. See figure 3.1 for an overview of this process.

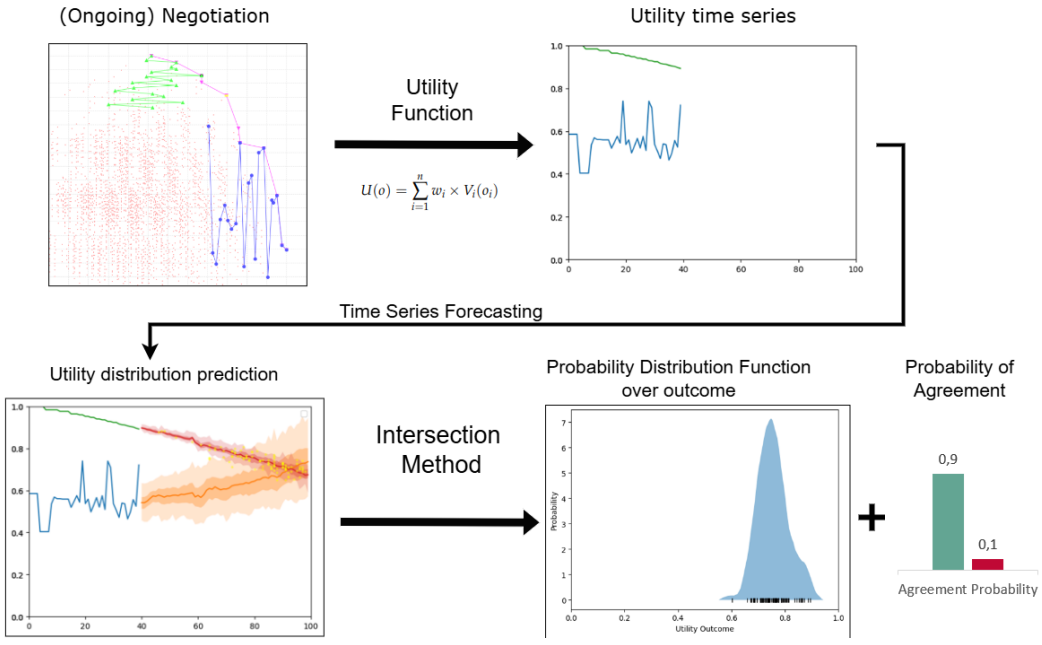


Figure 3.1: Schematic overview of how the outcome of a negotiation is predicted via our framework.

3.1 Negotiation Outcomes

We will now take a closer look at how automated negotiations are formally described and in which way they are used in this research. Recall that an outcome in an automated negotiation is a complete assignment of values to all issues within a domain. The set of all possible outcomes within a domain is called the outcome space, from hereon denoted with Ω, where $o \in \Omega$ is an outcome of the negotiation. Say we are organizing a wedding, for which the issues are the *location*, *music* and *food*. A possible outcome of this negotiation, which assigns a value to each issue, could be a *church*, *band* and *cake*, such that $o = (church, band, cake)$. Although describing the outcome in this form is insightful for the material result of the negotiation, it does not tell us how much the agents prefer this selection of values. To gain this insight, the outcome must be converted to a utility score via the agents' utility functions, which makes a valuation for all values selected in a bid. In our case, this is an additive utility function, which makes the calculation as such:

$$U(o) = \sum_{i=1}^{n} w_i \cdot V_i(o_i)$$

With w_i the weight of the selected issue and $V_i(o_i)$ the valuation of the offer selected for the issue. In this research, utility scores are always between 0 and 1. Say, we plug in the wedding example ($o = (church, band, cake)$) once more for two utility functions U_1 and U_2. As these utility functions have different valuations and weights to model their distinct preferences, they will compute different utility values for the same offer. In this case, U_1 is moderately enthusiastic about o, while U_2 is happy with o. This can lead to $U_1(o) = 0.4$ and $U_2(o) = 0.7$, where the outcomes represent how content both agents are with this outcome. In essence, the utility functions provide the perspective of the different participating agents in the negotiation.

This numerical perspective is essential when predicting negotiation outcomes, as it provides a clear, quantitative measure from which future values

can be induced. We can look at a complete negotiation as a vector of bids, made by two agents in an alternating offers protocol, as such:

$$\mathbf{o} = (o_1^1, o_1^2, o_2^1, o_2^2 \dots o_r^1, o_r^2)$$

Where o_r^a is a bid, r is the round in which the bid is made, and a is the agent who made the bid. This vector can be split into two vectors based on which agent made the bid, so that:

$$\mathbf{o^1} = (o_1^1, o_2^1, o_3^1 \dots o_r^1) \text{ and } \mathbf{o^2} = (o_1^2, o_2^2, o_3^2 \dots o_r^2)$$

These two outcome vectors can then be converted to four utility vectors by using both agents' utility functions as such:

$$\begin{aligned}
\mathbf{u}_1^1 &= (U_1(o_1^1), U_1(o_2^1), U_1(o_3^1) \dots U_1(o_r^1)) \\
\mathbf{u}_1^2 &= (U_1(o_1^2), U_1(o_2^2), U_1(o_3^2) \dots U_1(o_r^2)) \\
\mathbf{u}_2^1 &= (U_2(o_1^1), U_2(o_2^1), U_2(o_3^1) \dots U_2(o_r^1)) \\
\mathbf{u}_2^2 &= (U_2(o_1^2), U_2(o_2^2), U_2(o_3^2) \dots U_2(o_r^2))
\end{aligned}$$

This gives us four vectors of utilities in the form u_f^a, where f refers to the agent whose utility function is used, while a indicates which agents' vector of bids is used as input.

A negotiation can end in two different outcomes, an agreement or a breakoff. Looking at a complete negotiation $\mathbf{o}$, an agreement has been reached when both agents make the same offer consecutively. This can happen within the same round or the next. More formally, an agreement is reached when $o_{r_1}^1 = o_{r_2}^2$ and $|r_1 - r_2| < 2$. If this is not the case, the negotiation either ended

in a breakoff or has not finished yet. If the number of bids is equal to twice the round limit, this indicates that the negotiation has ended in a breakoff. This is because that means that both agents have made all their available bids. In general, we will only look at completed negotiations in this thesis, so ongoing negotiations will not be evaluated. Of course, the final algorithm will be able to predict ongoing negotiations. However, to develop and test the application, we will need negotiations for which the outcomes can be compared to our predictions.

In addition, we make predictions from the perspective of one agent. This aligns with how an agent in a negotiation perceives the negotiation and composes strategies for it. Moreover, the singular perspective makes the final application ideal for integrating into real negotiations done by automated agents, providing the most realistic perspective. Because of this, only one utility function will be used in the training, prediction, and analysis of negotiations. We will call this perspective, and the associated utilities "our" agent, since we assume this agent as our own perspective, while the other perspective and utility vector is referred to by the "opponent". This leaves us with two utility vectors from our own perspective, $\mathbf{u}_1^1$ and $\mathbf{u}_1^2$.

The first facet of our solution to negotiation outcome prediction is based on techniques from the domain of time series forecasting, which is concerned with the forecasting of time series. As such, TSF methods require time series as their input, which they then attempt to extend as accurately as possible. This raises the question of how these utility vectors are converted to time series. In our case, the chosen negotiation settings make this quite trivial, since the negotiation has a set number of rounds until the deadline. For this reason, every utility, corresponding to one round, can be interpreted as one value per "time unit". Additionally, the remaining rounds correspond to the amount of time units left. Therefore, the utility vector can be used in TSF unchanged, as time is an implicit part of its structure. Of course, there are scenarios imaginable for which this does not hold, and one round does not always perfectly correspond to the same amount of time. In these scenarios, additional steps must be taken to adapt the utility vector to a time series. In this thesis however, we will not consider such time series.

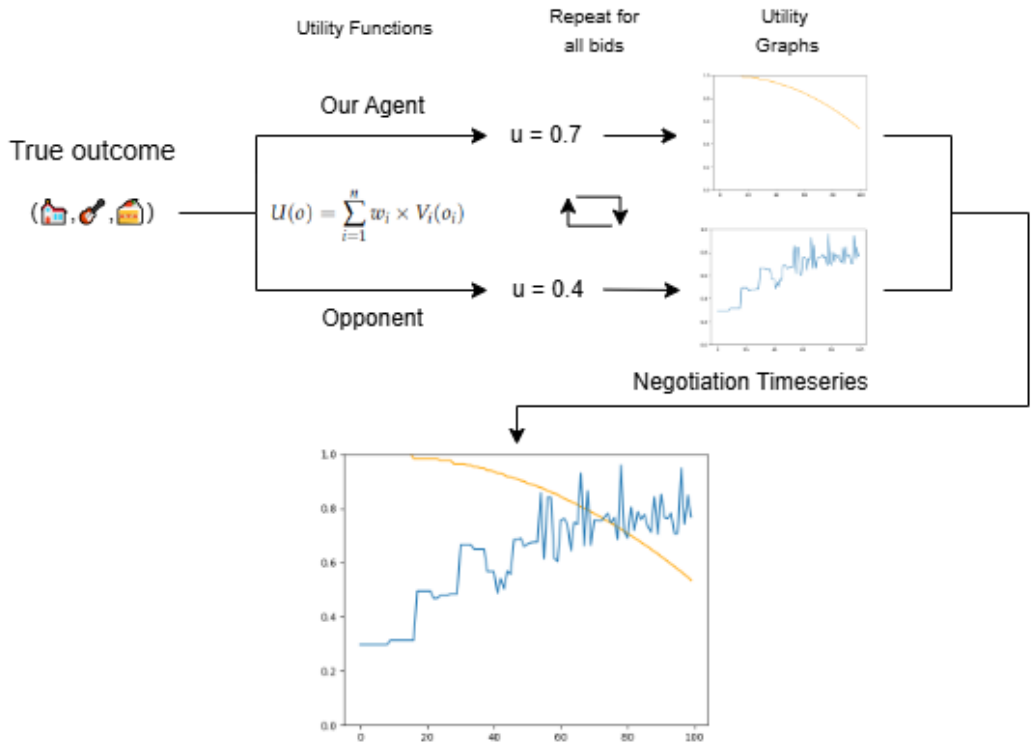


Figure 3.2: The process of extracting utility time series from bids.

3.2 TSF Solution: Neural Networks

3.2.1 Problem Formalization

Now that we have formalized the time series, we can make our first problem more exact. Since only one agent's utility function is used, we will now use u_r^a to refer to $U_a(o_r^a)$. The time series over a specific interval is indicated with $\mathbf{u}_{\mathbf{s:r}}^{\mathbf{a}}$, where s is the start of the interval and r the end.

The problem can then be framed as follows:

Given a time series observed at time t with a maximum length of r, find a function f to estimate a distributional forecast $\hat{\theta}$ for every time step in the interval $[t : r]$. Since this is done for both agents' time series, we will use an identifier a to indicate which time series is forecast. This gives:

$$f(\mathbf{u}_{1:t}^a) = (\hat{\theta}_{t+1}^a, \ldots, \hat{\theta}_r^a) = \hat{\boldsymbol{\theta}}^a$$

For which $f : S^* \longrightarrow S^{**}$

Where S* is a set of values for which each value $s \in I$, where I is the unit interval $[0, 1]$. S** is a secondary set where each value is a distribution θ. Every distribution θ consists of n values where for each value $p \in I$. n is a hyperparameter of the neural network that decides how many simulations are run to calculate the distribution $\hat{\theta}$.

Recall the first Research Question:

Research Question 1.1

How can we best predict the utility time series of the agents participating in a negotiation?

We propose to tackle this problem by using a neural network optimized for TSF. Automated negotiation time series are difficult to predict with classical statistical methods that only consider the time series they are currently predicting. This is because automated negotiation time series have a relatively low order of predictability (also see 4.3.2). Therefore, linear autoregressive models (such as ARIMA) will have a difficult time making accurate forecasts. While computationally heavier and vastly more work to set up, we believe using a neural network will pay off by its ability to detect nonlinear relations across time series.

We will show that this is best done by using a recurrent neural network to extend the two vectors of bids belonging to the agents participating in the negotiation. The RNN generates probability distributions that are evaluated using the negative log likelihood.

3.2.2 The network

Again, we want to find a method that generates a series of distributions based on past inputs. Recall the general form equation: $f(\mathbf{u}_{1:t}) = \hat{\boldsymbol{\theta}}$

Our solution must have the ability to learn from previously seen negotiations and use the inherent temporal structure of time series to find patterns in its input, which it can then use to generate the output. In section 2.4 we

already discussed the advantages of RNNs over regular neural networks. RNNs have been designed for the analysis of temporal data, and as such they were found to be the most effective network for predicting time series. To illustrate how an RNN is formed, we will build one up from a basic TSF prediction algorithm.

Let us start with a simple TSF predictor. Lim et al. define time series forecasting in its simplest (one-step ahead) case as follows [49]:

$$\hat{y}_{t+1} = f(\mathbf{y_{t-n:t}})$$

Where $\hat{y}_{t+1}$ is the forecast generated for the next time step, $\mathbf{y_{t-n:t}}$ the previous n observations of the value to be predicted (in our case, bids). Since we are using utilities as input for the model, from now on u is used instead of the common y notation. Moreover, our model predicts up to the round limit r, rather than predicting one step into the future, and takes into account the full bid history, rather than any n number of past bids. So, the appropriate definition becomes:

$$\mathbf{\hat{u}_{t+1:r}} = f(\mathbf{u_{1:t}})$$

This function provides a high level of mathematical abstraction and as such defines what we need from the network we are constructing. To make the step towards an actual neural network, we must zoom in on the contents of the function f. Recall from section 2.4 that a neural network consists of many units called neurons. A neuron is a mathematical function that makes a computation over all its input neurons using its weights, biases, and the activation function, as such:

$$a\left(b + \sum_{j=1}^{n} x_j w_j\right)$$

Where a is the activation function, x the input from its n input neurons, and w and b the weight and biases associated with this specific neuron. The activation function introduces a nonlinearity over its input, which is essential for any neural network to function as this enables it to find nonlinear relations in the data. The weights scale the input by some factor, whereas the biases add a value directly to the output. The values for both the weights and biases are optimized during the training phase. One such neuron makes a simple calculation, but a collection of many neurons together quickly increases the complexity and aggregates into a neural network. The content of our function f is such a collection.

The primary way in which RNNs are tailored for temporal data is that their network structure has lateral connections between neurons, rather than only feedforward connections. See figure 3.3 for an illustration of the data flow in an RNN. These lateral connections ensure data is 'saved' within the network and thus act as an internal memory state. We will refer to this memory state at time step t as z_t, which can be defined as:

$$z_t = v(z_{t-1}, u_t)$$

Where v is the function to update the internal memory state, z_{t-1} the previous memory state and u_t the bid made at time step t. z_t here represents the content of all neurons in the network that together decide how the RNN reacts to input, and is the result of the interpretation of its previous inputs. This is what distinguishes RNNs from regular neural networks, as it does not only use its weights and biases to compute output. As such, previous input, which determines the memory state, also influences the output. The

same input can thus result in different outputs, depending on what the network has seen earlier.

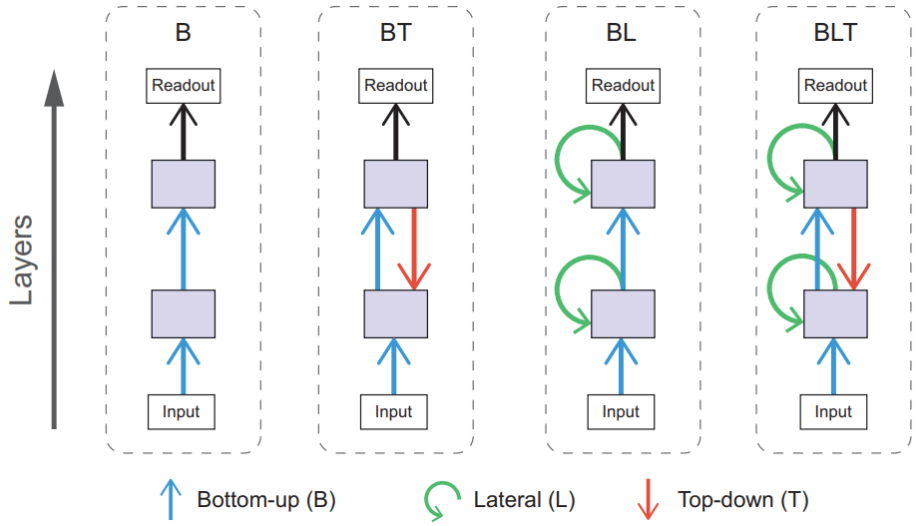


Figure 3.3: Schematic overview of the dataflow inside an RNN. While a regular NN only has bottom-up connections, an RNN also has top-down and lateral connections between its neurons. Image adapted from [61].

Since all parameters in this computation influence the memory state, we must also consider additional parameters in our definition. We add the earlier defined weights (**W**), biases (**b**) and activation function. In our case, the activation function used is softplus, which is commonly indicated by γ. The new equation becomes:

$$z_t = \gamma_z(\mathbf{W}_z z_{t-1} + \mathbf{W}_u u_t + \mathbf{b}_z)$$

Here, $\mathbf{W}_z$ are the weights for the hidden units, $\mathbf{W}_u$ the weights for the input and $\mathbf{b}_z$ the biases belonging to the hidden units.

Lastly, the network does not give a point prediction, but rather a conditional distribution for each step in range [t:r]. This is done by predicting all parameters that correspond to the distribution θ. In our case, this is a Student's t-distribution. As such, it estimates the degrees of freedom v, location μ and scale τ parameters. A Student's t-distribution was chosen for its relatively heavy tail, which resembles the heavy spread in the bids made

by automated agents, as also found in 4.3.2. Therefore, the output will be indicated by the earlier defined $\hat{\boldsymbol{\theta}} = \hat{\theta}_{t+1} \dots \hat{\theta}_r$.

Combining this output with the internal memory update gives:

$$\hat{\boldsymbol{\theta}} = \gamma_z(\mathbf{W}_z z_{t-1} + \mathbf{W}_u u_t + \mathbf{b}_z)$$

As the final RNN function.

The architecture of our DeepAR network will consist of 1 input layer, followed by 4 fully connected hidden layers that all contain 50 nodes, and ends with an output layer.

3.2.3 Loss function

To ensure that the network generates a prediction of the probability distribution similar to the actual bids, the weights and biases will be adjusted in the training phase. This is done by minimizing the loss function. For DeepAR, the loss function is the negative log-likelihood (NLL). The NLL is defined as [62]:

$$-log(L(\hat{\boldsymbol{\theta}}, \mathbf{u}))$$

The NLL is a common loss function in deep learning that takes the negative logarithm of the likelihood function, L. The input of the likelihood function consists of two vectors. One vector contains the predicted distributions of the remaining time steps $\hat{\boldsymbol{\theta}}$. The other contains the true utility values of the remaining time steps, indicated with $\mathbf{u}$. For every time step, the likelihood function gives us the probability of observing the true value u when the data is extracted from the probability distribution $\hat{\theta}$.

Since the input for the logarithm is the likelihood of generating a correct probability, this input is to be maximized. However, since it is common

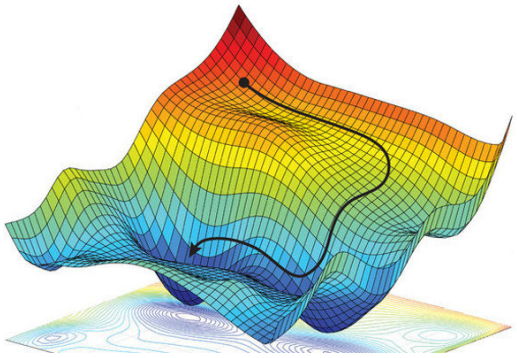


Figure 3.4: Model of backpropagation via gradient descent. Every location on the plane represents a different combination of optimized parameters for the neural network. The height of the location shows the loss associated with this combination of parameter settings. In gradient descent, the location on the plane that has the lowest loss is sought after. Image adapted from [63].

practice in deep learning to minimize a loss function, the negative log-likelihood is taken instead, so that a minimized value is desired. The likelihood function can theoretically output any real-valued positive number. After applying the two remaining operations, the theoretical output can be any real-valued number, both positive and negative.

3.2.4 Training

To optimize the weights and biases so that the network can predict time series, the network must be trained. For this, we make use of supervised training on a data set of completed negotiations. After a first round of predictions made with randomly initialized weights and biases, the network updates these weights with a process called backpropagation. In short, backpropagation calculates the gradient between the loss function and the weights and biases. Then, the weights and biases are updated so that the value of the loss function is minimized, e.g. descending down the calculated gradient. See figure 3.4 for an illustration of the backpropagation process.

This process is repeated many times, each iteration further optimizing the network's parameters and minimizing the loss function, until it is no longer (or only very marginally) improving. This is also why a loss function is essential in deep learning, as it defines the way in which the weights and biases are updated, which in turn dictates the network's behavior.

3.3 Intersection Solution: Monte Carlo Methods

The next step in the process is to connect the two generated distribution forecasts and use them to compute a prediction of the utility outcome of the negotiation and an estimate of the agreement probability.

3.3.1 Problem Formalization

Earlier in this chapter we defined that a negotiation ends in an agreement when both agents make the same offer, since this implies an agreement. However, this definition cannot be used for negotiation prediction with time series. This is because using the utility function to convert a regular negotiation outcome to a utility score makes it a numerical score. Since the forecasting method extends these numerical scores without regard for actual possible outcomes, the outcomes generated by forecasting are continuous. This is in contrast to the true possible negotiation outcomes, which are all in the discrete outcome space. Because of this, the equality of two outcomes can no longer be properly assessed.

For this reason, we use a different definition of an agreement for this part of the research, one that can be used when assessing continuous forecasts made by TSF methods. We will henceforth define an agreement as the moment the two forecast lines intersect each other. This is because the intersection of these forecasts indicates that both agents have conceded toward each other enough to come to a mutual agreement around this utility score. More formally, we say that the forecast method found an agreement when $(u_t^1 <= u_t^2 \mid u_{t-1}^1 > u_{t-1}^2)$, where $u_t^q \in \mathbf{u_r^a}$. We use many of such agreement predictions to estimate the total agreement probability p and the utility probability distribution $\hat{\beta}$.

The problem can then be framed as follows: Given two distribution vectors of the remaining rounds $\hat{\boldsymbol{\theta}}^{\mathbf{1}}$ and $\hat{\boldsymbol{\theta}}^{\mathbf{2}}$, estimate a probability distribution over the utility range $[0 : 1]$, as well as predict the agreement probability p. For this instance, the distribution estimation will be indicated with the $\hat{\beta}$

symbol. The goal is to find an estimator function f such that:

$$f(\hat{\boldsymbol{\theta}}^{\mathbf{1}}, \hat{\boldsymbol{\theta}}^{\mathbf{2}}) = (\hat{\beta}, p)$$

For which $f : S^{**} \times S^{**} \longrightarrow (\beta, I)$. Here, S** refers to the earlier defined set of distributions, β to a continuous distribution, and I to the unit interval [0,1].

Recall the second Research Question:

Research Question 1.2

How can we best predict the outcome of an automated negotiation based on time series forecasts of utility graphs?

We propose employing Monte Carlo sampling as the method of choice to infer a distribution of utility outcomes from the RNN output. In Monte Carlo sampling, random samples are taken from a distribution in an attempt to reconstruct this distribution. This method is an easy choice as our neural network already generates its distribution in a Monte Carlo fashion, namely via many individual predictions that together make up one distribution. These individual predictions make it so that samples can be easily drawn from the forecasts. Monte Carlo also provides a method that is very countable. In other words, every individual sample is a discrete unit that all together make up our total prediction. This is advantageous for calculating the agreement probability, as will be shown later in this section.

3.3.2 Generating Samples

The first step of the function f is to generate a multitude of samples from the distributions $\hat{\boldsymbol{\theta}}^{\mathbf{1}}$ and $\hat{\boldsymbol{\theta}}^{\mathbf{2}}$. Let us define a function g that, given a distribution, returns a vector of utilities of the same length, sampled from the distribution, such that $\hat{\mathbf{u}} \sim \hat{\boldsymbol{\theta}}$. Since these utilities are an estimation, and to avoid confusion with true negotiation bid histories, they will be denoted with $\hat{\mathbf{u}}$. This gives: $g(\hat{\boldsymbol{\theta}}, n) = \hat{\mathbf{u}}$, where n is the desired number of samples.

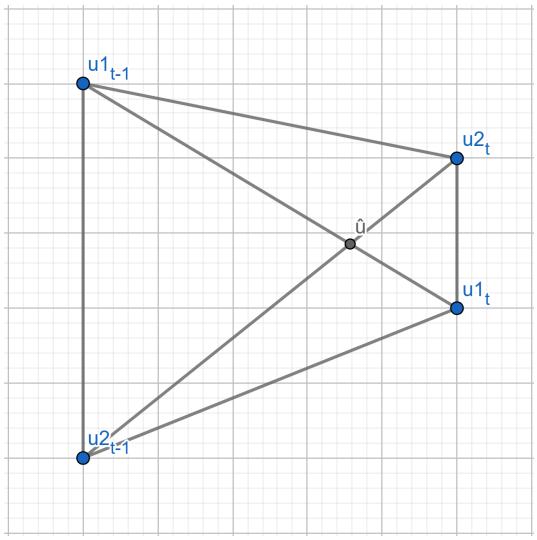


Figure 3.5: Trapzium formed to find the intersection of two utility forecasts. $\hat{u}$ is the intersection point that we aim to find.

These utilities describe a continuation of the utility time series path already observed by the RNN.

The next step is to use the utility vectors generated from both distributions to find the intersection of these paths. Function h takes two utility paths and finds this intersection, a scalar utility value, such that $h(\hat{\mathbf{u}}^{\mathbf{1}}, \hat{\mathbf{u}}^{\mathbf{2}}) = \hat{u}$. It does this using the earlier defined agreement definition $(u_t^1 <= u_t^2 \mid u_{t-1}^1 > u_{t-1}^2)$. As an estimate of the final utility, this method takes the intersection of the lines going from u_{t-1}^1 to u_t^1 and u_{t-1}^2 to u_t^2, which form a trapezium. As such, the utility is the y-coordinate of the intersection of the diagonals of this trapezium, which can be found using high school math. Also see figure 3.5.

If the condition for an intersection is never reached (the lines do not cross), no utility is calculated. Instead, the negotiation is said to have ended without an agreement. This process is repeated n times, where all generated utilities are added together to yet another vector, $\mathbf{v}$. This vector contains the different point estimations of the utility outcomes found using the intersection method h, so that $\mathbf{v} = [\hat{u}_1, \hat{u}_2 \dots \hat{u}_n]$. This leaves us with a collection of utility predictions for the outcome of the negotiation.

In addition to utility prediction, we are also interested in the probability that the negotiation will end in an agreement, the agreement probability p.

When the desired number of Monte Carlo simulations has been run, the ratio of simulations that ended in an agreement is returned, and is used as the network's prediction of the likelihood of the true negotiation ending in an agreement.

3.3.3 Forming a distribution

Now that we have a vector of utilities, we must convert them into a probability distribution. A common method to convert a large number of points into a distribution is via a Kernel Density Estimator (KDE). A KDE applies a smoothing function over the utilities to convert the finite data samples to a continuous probability function. The density f of this probability function at a given point x, is found by:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

Here, K is the kernel function and h is a smoothing parameter, called the bandwidth. The kernel function can be one of many different functions that influence the shape of the final distribution (e.g. a certain kernel may always generate one which approximates a guassian distribution).

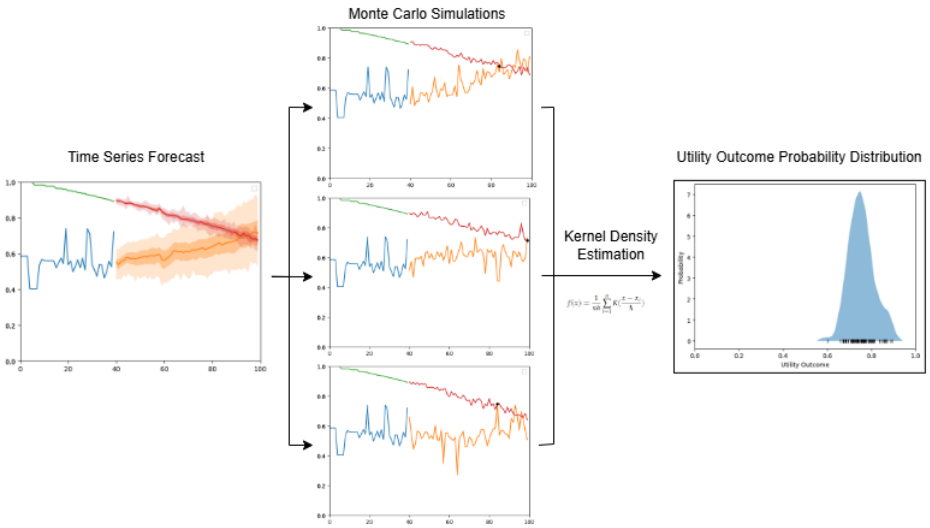


Figure 3.6: Monte Carlo method of generating a probability distribution outcome from utility graph distributions.

3.4 CRPS

During the experiment, different TSF techniques will be used as benchmarks for the deep learning method. The predictions of these different forecasting methods must be compared to be able to draw a conclusion about their relative performances. For this purpose, a metric must be used that measures the similarity between the forecasts generated by the TSF methods and the actual results of the negotiation. Since predictions are made from completed negotiations, the true values with which the forecast can be compared are known. Given this, the accuracy score used must compare a point (the actual value) to a distribution (the forecast). Hyndman introduces multiple of such distributional forecast accuracy metrics [32]. Although some of these metrics require a specific quantile of the prediction distribution to be selected for comparison, we are interested in the whole forecast distribution, rather than particular quantiles or prediction intervals. For this purpose, the continuous ranked probability score (CRPS) is a good metric. First introduced by Matheson and Winkler [64], CRPS quantifies the difference between the perfect distribution of data and the predicted distribution [65]. It does this by comparing the cumulative distribution functions (CDF). The formula for CRPS is:

$$CRPS = \int_{-\infty}^{\infty} (F(x) - \mathbb{1}\{y \leq x\})^2 dx$$

Where $F(x)$ is the CDF of the predicted distribution and y the realized value. The CDF can be found by integrating the generated probability density function. $\mathbb{1}\{y \leq x\}$ is a function that returns 1 when inequality $y \leq x$ is true and zero otherwise [66]. As such, CRPS values will always be between 0 and 1 and approximately indicate the distance of the distribution mass to the true value. For example, imagine that the generated distribution lays around 0.6, while the true value is 0.7. CRPS will return an error score of approximately 0.1. Additionally, CRPS has the advantage of being able to compare both pointwise and distributive predictions to the ground-truth

(pointwise) value. This is ideal, as the majority of benchmark methods generate a pointwise prediction as well. When computing a pointwise comparison, CRPS simplifies to the mean average error (MAE), a score that simply gives the distance between the true value and the prediction. For the evaluation of TSF performance, this metric can be used for each generated time step, after which the average CRPS over the time steps can be computed for a final evaluation of the prediction. Furthermore, when evaluating the negotiation outcome prediction, the method can simply be used to compare the prediction to the actual outcome. Note that earlier in this chapter, we described the use of the negative log-likelihood as also comparing distributions with pointwise outcomes, only there for calculating the loss function. CRPS is used here because it is easy to interpret and easy to compare to pointwise prediction evaluations via MAE. NLL on the other hand is better suited as a loss function since it does not have a lower bound, which is a desirable quality for a loss function.

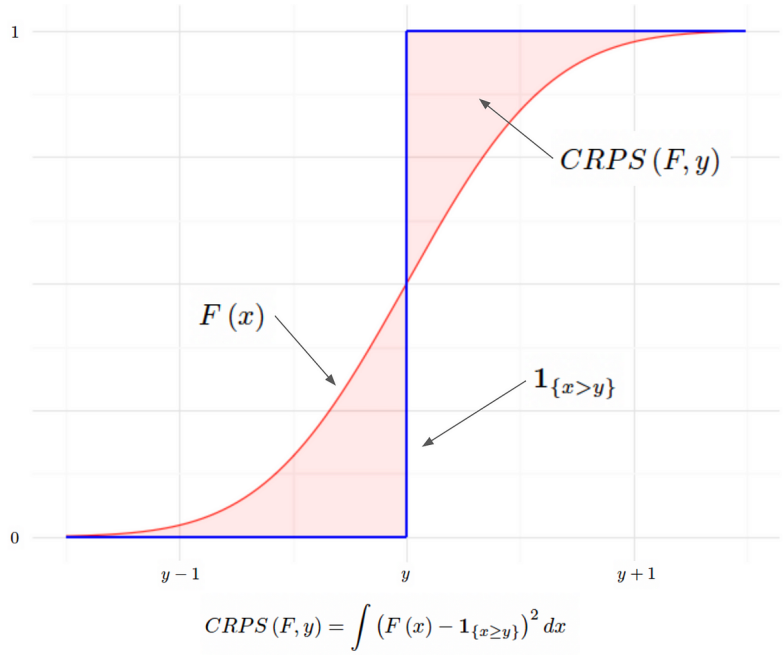


Figure 3.7: Visualization of CRPS. In red: The CDF beloning to the generated distribution. In blue: The CDF beloning to the ground truth. The CRPS is the area between these two CDF's. Image adapted from [67].

4. Experiments

4.1 Experiment Overview

We describe two main experiments to answer our research questions. This chapter will go into detail about these experiments, explaining their setup. In addition, some preliminary experiments have been run to arrive at the exact parameter settings used in the main experiment. These will also be explained after the main experiments.

The first experiment evaluates the performance of DeepAR in its ability to forecast time series. This is done by having DeepAR and five benchmark methods forecast the utility graphs of 1100 negotiation time series. These 1100 time series consist of 100 series for 11 different learning rates. Every time series will be predicted from four different starting points, to evaluate the influence the amount of input data has.

The setup for the second experiment is very similar to that of the first. Once again, DeepAR and five benchmark methods are used, this time to predict the outcome of negotiations. The data set used for experiment 1 will also be reused for this experiment. For experiment 2, we expect a similar relative ordering of performance between the six tested methods as in experiment 1. This is because we expect the ability to predict time series to be indicative of these methods' ability to predict the negotiation outcomes.

4.2 Experiment Setup

Before running these experiments, we must first carefully consider which negotiations to use in evaluating the methods' performances. There are various types of negotiating agents that all use their own distinct strategy. These strategies in turn also react to different strategies in diverse ways. On top of

this, the setting in which the agents negotiate and the rules that they must adhere to influence the negotiation as well.

Together, these factors result in negotiations being affected by many interacting parameters. For this research, it is beneficial that these interacting factors are kept to the essential ones only. This is because if there are many factors influencing a negotiation, it will be harder to assess which of these factors determine both our predictions and final experimental results. If we have a small number of parameters that we can adjust ourselves, this means that we can test the effects of these parameters deliberately in a controlled environment. This in turn increases the robustness and applicability of this research.

In short, the experiment will be set up as follows:

A data set of negotiations is made by running 100 negotiations for 11 learning rates, 0 to 1.0 in steps of 0.1. These negotiations are run between time-dependant agents whose e-value is randomized between 0.1 and 1.3, and whose m-value is randomized between 0.5 and 0.9. All negotiations are run between profiles belonging to the party domain. Six different methods are used to forecast the associated time series: DeepAR and five benchmark methods. Every method makes a forecast from four different points within every negotiation, from 20%, 40%, 60%, and 80% (Experiment 1). The forecasts are then used to predict the final outcome of the negotiation via Monte Carlo sampling, as well as predict the likelihood of ending in a breakoff (Experiment 2). The results of both experiments are evaluated using the CRPS metric to compare the different benchmarks.

4.2.1 Time-Dependent Agents

We want the agents used in the experiments of this research to be influenced by a limited number of factors. Those factors that it is influenced by must be within our control, as touched upon in the above section. However, this restriction must not decrease the possible variety of shapes the utility graph can take too extensively, as we want our final prediction method to be rigorous in its ability to predict many types of negotiations. Recall

the three broad categories of automated negotiation strategies (see section 2.2.1): time-based, behavior-based and mixed. The latter two change their bids based on their opponents' actions. To mitigate the complexity such strategies would add, we will use an agent using a time-based strategy. Specifically, we opt to use the time-dependant agent (TDA).

The TDA makes concessions by following a precise concession curve, defined by the formula:

$$u(t) = 1 - (t^{1/e} \cdot (1 - m))$$

Where t is the fraction of the total negotiation time that has passed. e is the concession rate and m the target utility. The values of e and m are set before the negotiation, while t goes from 0 to 1 throughout it. The TDA finds the bid in the outcome space Ω that is closest to this value u.

Since the TDA always follows this curve, it is a deterministic strategy and therefore uninfluenced by other factors. On top of this, via parameters e, m and the learning rate, the TDA can be made to mimic the shapes of various other negotiation strategies. Therefore, despite being only a single strategy in theory, it still ensures that our prediction method is capable of recognizing the bidding patterns of many diverse strategies.

4.2.1.1 Parameters

Numerous parameters dictate the exact behavior of the TDA, all of which have to be carefully set before the research is run. We will go over all parameter settings and briefly explain their use as well as their settings.

Recall the TDA formula defined earlier. The first crucial setting is the concession value, e. The TDA slowly concedes over time by making bids of an increasingly lower utility, based on the remaining time. e determines the speed of this concession. A TDA with a concession speed of 0 will not concede at all and instead play hardline. The higher the concession speed, the

faster the agent concedes its utility. A TDA with a concession speed of 1 will concede linearly. When the agent has a concession speed below 1, it will start the negotiation playing hardline for some time, after which it will start to concede. See figure 4.1 to see what effect different e-values have on the utility graph of the TDA. For the experiment, e will be set between 0.1 and 1.3. See section 4.3.1 for an explanation of the chosen values.

The next parameter influencing the TDA's concession curve is m. We call this parameter the 'target utility', as it determines the final utility TDA concedes towards. This can be shown by entering t = 1 (the end of the negotiation) in the formula, which gives: $u(1) = 1 - (1 - m) = m$. A higher m value will make the agent aim for a higher final utility and, as such, will give it less space to concede. This can be compared to a reservation value. The target utility m is highly influential in the negotiation utility graph, particularly in whether the negotiation ends in an agreement or a breakoff. See figure 4.2 to see what effect different e-values have on the utility graph of the TDA. Also see Appenidix B.3 for a more elaborate overview of the effects these parameters have on the concession curve.

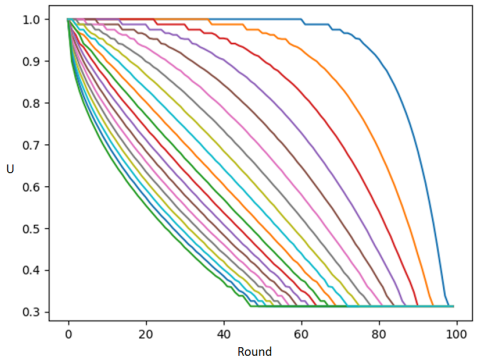


Figure 4.1: Effect concession rates 0.1 - 2.0 have on the concession curve.

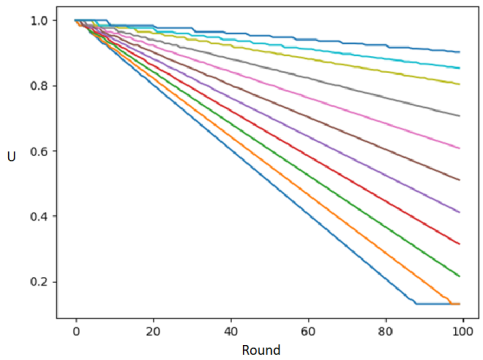


Figure 4.2: Effect target utilities 0 - 0.9 have on the concession curve.

4.2.1.2 Learning

To be able to test an even wider array of negotiation shapes, we devised a method to simulate learning behavior in TDAs. We suspect that an agent that learns will produce a bidding curve that is easier to predict by our forecasting methods, as it will increase the trend and lower the noise.

To model the effects that a behavior-based strategy has on the predictions, we introduce a "learning rate" (l) parameter to the agents. The learning rate is intended to model an agent using an opponent model. Such an agent will have some knowledge of their opponent's utility function and can use this in making bids. Recall the utility calculation made by the TDA:

$$u = 1 - (t^{1/e} \cdot (1 - m))$$

A regular TDA will choose the bid within the outcome space Ω that is closest to u. The learning rate limits this outcome space so that it no longer includes all possible outcomes. This is done by removing all outcomes for which the *opponent* has a utility below $P(t) \cdot l$, where P(t) is the utility value corresponding to the Pareto frontier of the current time step and l the learning rate. The learning rate can be set between 0 and 1. For $l = 0$, the TDA behaves as normal without a learning function, while $l = 1$ makes the TDA follow the Pareto frontier exactly, implying maximal learning. Also see figure 4.3.

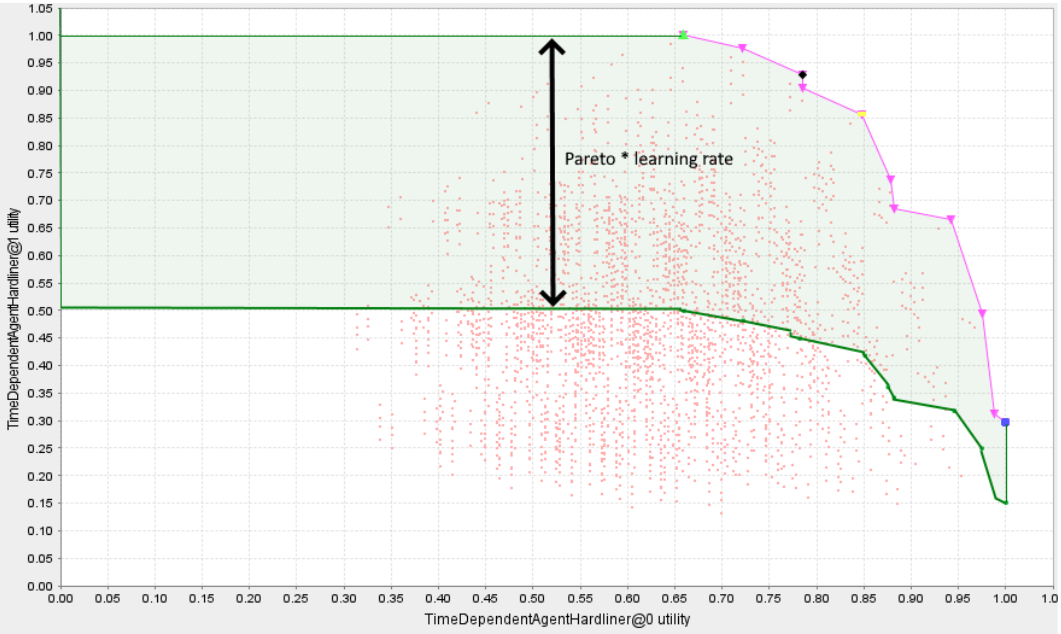


Figure 4.3: Outcome space of a negotiation. In green, the remaining outcome space for a learning rate of 0.5. The pink line is the Pareto frontier, which the green area is based on.

Parameter	Description	Valid Range
Concession Rate (e)	Determines the speed at which the TDA concedes its utility.	[0 - inf]
Target Utility (m)	Determines the closing utility value the TDA concedes towards.	[0 - 1]
Learning Rate (l)	Determines the relative minimum utility value.	[0 - 1]

Table 4.1: Table showing the different parameters relevant to the TDA.

Learning as we implemented it in the TDA will install a lower bound in the bids the agent makes from the perspective of its opponent's utility function. As such, the opponent will always receive some minimal amount of utility, scaled from the Pareto frontier.

Going by the step model conceived by Hindriks et al. [68], a regular TDA can only make concessions and unfortunate moves, since it decreases its own utility while moving along the outcome space. In the context of TDA, our learning method makes as many moves as possible concessions, rather than unfortunate moves. Such concessions are appealing to the opponent, providing high utility for (ideally) both parties.

4.2.2 Negotiation Parameters

The negotiations used for training and forecasting follow a stacked alternating offer protocol, with a 100-turn round limit. This means that the agents will go back and forth making bids until one of them accepts the bid of the other agent. If both agents have made 100 bids without coming to a resolution, the negotiation is ended without an agreement.

A negotiation prediction can take place at any point during the negotiation. To study the effect of making a prediction starting at different moments, the experiment is run for four different starting points. The starting points are from 20%, 40%, 60% and 80%, of the negotiation. This is done for all methods and will provide insight into how strongly extra observations improve predictions.

The profiles used for the negotiations are those in the party domain, one of the domains available in Genius. This domain is chosen for its diversity in

profiles, as it provides 8 unique ones. On top of this, the outcome spaces belonging to the different combinations of profiles capture a wide portion of the utility outcome plane, with a high density of outcomes. The party domain has 23 options divided over 6 issues, making a total of 3072 possible outcomes.

4.2.3 Benchmarks

The main focus of this research is on deep learning methods for predicting negotiations. However, since this is a novel endeavor in the field, there has been no comparable research done. To place the research in the field of automated negotiations, a number of different methods will be used to predict negotiations as well. This collection is composed of state-of-the-art statistical methods and simple benchmark methods. The following methods will be used for comparison:

- ARIMA: Stands for Autoregressive Integrated Moving Average, a classic statistical model that fits a model for the data based on three parameters, p, d and q.
- Exponential Smoothing: Uses a weighted average of past observations to predict future values, where older observations' weights decrease exponentially.
- Linear Regression: Uses a regression algorithm to fit a linear function over past observations.
- Linear Extrapolation: Fits a linear function based on the first and latest bid in the past observations.
- Naive: Repeats the latest bid indefinitely.

Of these methods, only ARIMA is capable of generating distributive predictions as well. The other methods will generate a point prediction for each future round. For a more thorough explanation of these benchmarks, see Appendix A.1.

4.2.4 Metrics

The predictions made by the evaluated methods must be assessed. Recall that every method makes two predictions. One time series forecast over the utility graph and one prediction of the negotiation outcome. Assessment of the utility graph forecasts will be done using the CRPS metric, as it is well suited for comparing distributions to points, while also being usable for simple point predictions, in which case it simplifies to the MAE score. In experiment 1, the average CRPS/MAE will be taken over all rounds for which a prediction was made.

CRPS will also be used to compare the predicted utility (distribution) with the true negotiation outcome in experiment 2. The metric used to assess the performance in predicting agreement probabilities is the average agreement probability. For distributive methods, this is the average of true probabilities, while for pointwise methods, it is the average of boolean (0 or 1) predictions. This average is computed for the two types of negotiations, agreement and breakoff, separately. A good performing method will have this average close to 1 for the former, and close to 0 for the latter.

4.3 Preliminary Experiments

To be able to effectively run the two main experiments, a lot of preparatory experimentation had to be done relating to the setting in which the experiments were ran. Those that influenced the main experiment in a meaningful way will be described here.

4.3.1 Parameter settings

The parameters selected for the main experiments were chosen based on tests designed to create an optimal environment to evaluate the different prediction methods. Let us go over the choice for parameters e and m. Recall that e dictates the concession rate and m the end value of the TDA. These values strongly influence both the length of the negotiation and the breakoff rate by changing the shape of the utility graph. For the main experiment,

there are two requirements that the final data set must meet.

1. **Negotiation Length** In the experiments, negotiations are predicted from a minimum cutoff length of 20. Therefore, negotiations that terminate before this point have no use in being evaluated. So, the parameters must be chosen so that there is a minimal number of early terminations in the data set.

2. **Agreement Rate** One of the two outputs generated by the prediction method is the agreement probability. To properly test the methods performance in this category, there must be enough negotiations ending in both agreements and breakoffs. However, we still want the vast majority of negotiations to end in an agreement, since that is required to have a sizeable data set with which to calculate the CRPS for the utility outcome predictions. Therefore, we aim for approximately 20% negotiations in the data set to end in a break-off.

Within the bounds of these requirements, we will try to find a wide range of values for both parameters. All agents used to create the data set for the main experiment will be initialized with their parameters set within this range. To determine what values are best suited to meet these requirements, a large data set is built consisting of 100 negotiations for every combination of e and m. The range of e is 0.1 to 2.0, in steps of one ($e \in \{0.1, 0.2, ..., 2.0\}$), and the range of m is 0 to 0.9 in steps of 0.1 ($m \in \{0, 0.1, ..., 0.9\}$). For each of these negotiations, the agent has a random learning rate l. Its opponent is initialized randomly for all parameters e, m and l. We can plot the lengths of these negotiations, the results of which can be found in figure 4.4.

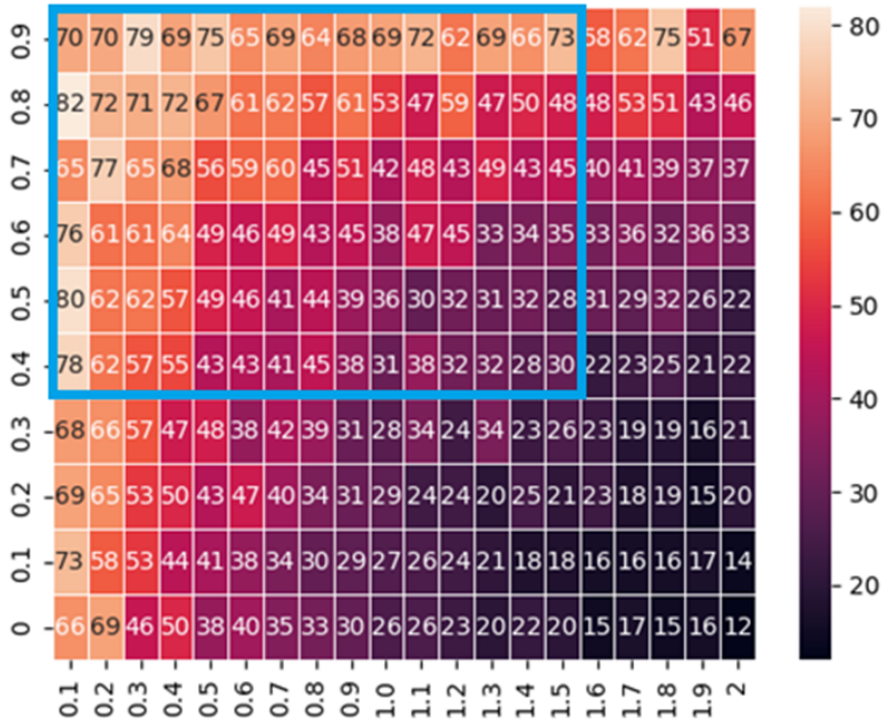


Figure 4.4: Heatmap showing the average negotiation lengths for negotiations of different parameter combinations. In blue outline the range of parameter combinations to be considered.

As we want all negotiations in the true data set to end after the 20th round, we will only consider parameter combinations that yielded an average length of 25 and up. Therefore, our first bound is that we will only consider agents with an e value of 1.5 and lower, and a m value of 4.0 and higher, as indicated with the blue border in figure 4.4.

Within this range, we investigate the breakoff ratio for different combinations of parameters. The breakoff ratio addresses the entire collection of negotiations within a data set, rather than individual negotiations. Therefore, we will not consider just the combination of two parameters. Rather, multiple data sets will be created in which every agent is randomly initialized within the values of a range. The outermost points of this range will be $m = 0.9$ and $e = 0.1$, which is the highest leftmost cel in the heatmap. The parameter settings on the other end of this range are the values that

we will experiment with. For m, this will be {0.4,0.5,0.6} and for e this will be {1.0,...,1.5}. These values were chosen to ensure that the final ranges still occupy a large part of the initial heatmap, as we would like the data set for our main experiment to be as diverse as possible within our requirements. Looking at figure 4.5, we can see that the combination of 0.5 and 1.3 as the (other) edges of the range provides a breakoff percentage of 20%. Therefore, these values will be used in the main experiment.

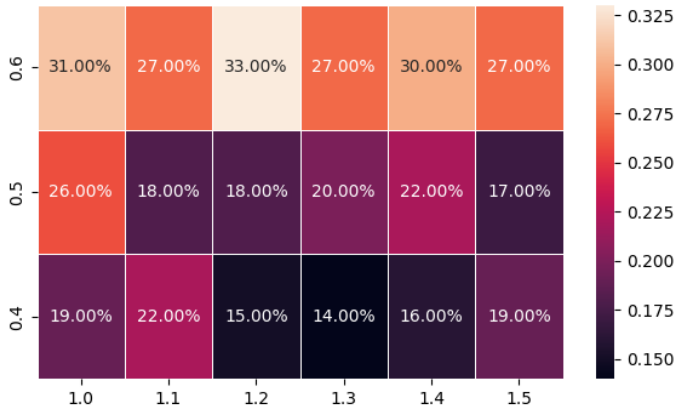


Figure 4.5: Heatmap showing the percentage of negotiations which ended in a breakoff, calculated over 100 agents per cell. The agents were intialized with a random value for t and e within a preset range. One end of the range is m = 0.9 and e = 0.1, and the other end the values on the heatmap axes.

4.3.2 Predictability

The decision to have a deep learning method as our main focus of this thesis came after careful analysis of the time series data derived from automated negotiations. In this section, we will present our findings related to the predictability of automated negotiation time series data, which led us to choose deep learning over statistical models as the basis for this research.

To determine the predictability of the time series at hand, we will use several measures that can serve as indicators of the predictability of the time series. Recall that predictability by itself cannot be plainly assessed like speed or volume, but by using some analysis we can get an idea of the general predictability of the time series.

1. **Trend** To start, the trend. The range of this value is between -1 and 1, where any extreme of this scale indicates a strong trend downward or upward, respectively. A trend of 0 indicates that there is no trend at all, which means that the graph moves downward as much as it moves upward. The value indicates the ratio with which it moved in some direction more than the other. It is calculated as such:

$$(u - d)/(u + d)$$

   Where u is the number of past observations smaller than the current observation (indicating an upward trend) and d the number of past observations bigger than the current (indicating a downward trend). Figure 4.6 shows that time series with a low learning rate are as good as trendless, but as the learning rate increases, so does the amount of trend in the data.

2. **White noise** Testing for white noise requires analyzing the ACF plots of the time series. These results can not be averaged together, which means each individual time series must be analyzed individually. A time series shows signs of white noise if less than 5% of ACF correlations lie outside $\pm 2/\sqrt{T}$, where T is the length of the time series (100 in our case). Figure 4.7 shows what percentage of time series adheres to this definition across learning rates. We can see that for learning rates 0.0 - 0.4, the percentage of white noise series stays somewhat consistent around 0.7. After this point, the percentage quickly goes down, until it reaches 0% at learning rate 0.8. These results suggest that, for low learning rates, the majority of time series resembles white noise. A type of time series which is very difficult to predict with classical prediction methods.

3. **Random Walk** In section 2.3.1.1 we discussed the characteristics of a random walk. A time series might be a random walk if it has two properties:

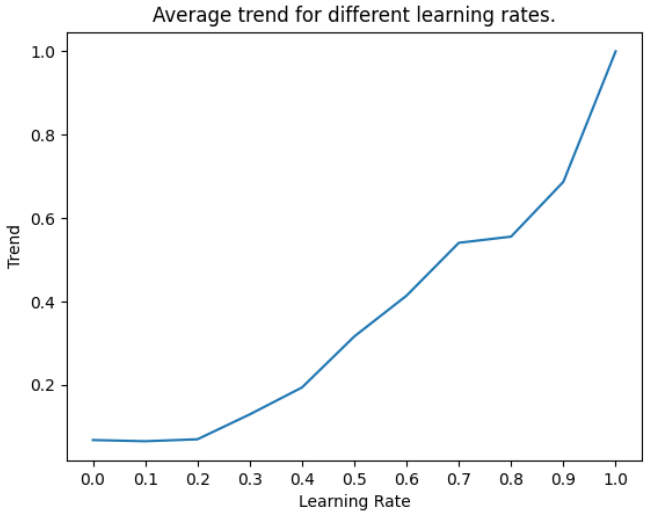


Figure 4.6: Average trend for TDA time series of different learning rates. Each average computed with 100 different time series.

(a) It is non-stationary

(b) Its autocorrelation plot shows a high dependence on the previous observation, after which it steeply drops.

As with white noise, we cannot definitively prove that a time series is a random walk, but if we can observe these two properties, this gives a strong indication in that direction. Once again, every time series in the data set was analyzed for indicative behavior. The results can be found in figure 4.8. Here we can see that the higher the learning rate, the more graphs resemble a random walk (with upward drift). This is likely due to the graphs getting more directionality at higher learning rates, which translates into a high autocorrelation with the previous time steps. Moreover, the random walk graph appears to approximate the inverse of the white noise graph. This may suggest that those time series that are not white noise are random walks.

Taking all these observations together indicates that TDAs without learning produce time series that are difficult to predict for classical statistical methods, as the series frequently resemble white noise series and do not have any trend. It also shows that the learning rate has an evident effect on the TDA time series, improving the trend and removing the white noise.

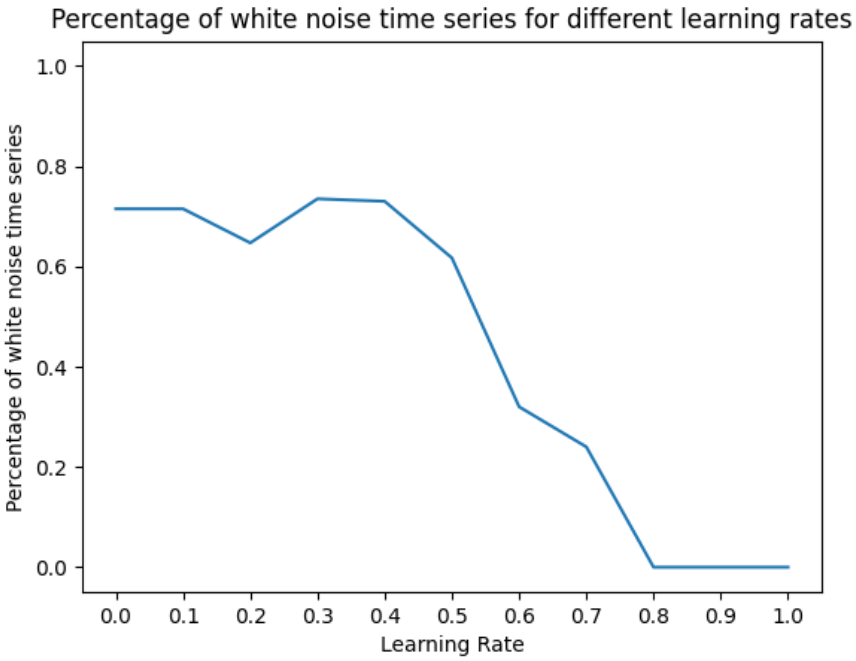


Figure 4.7: Ratio of TDA time series that show 'white noise' behavior for different learning rates.

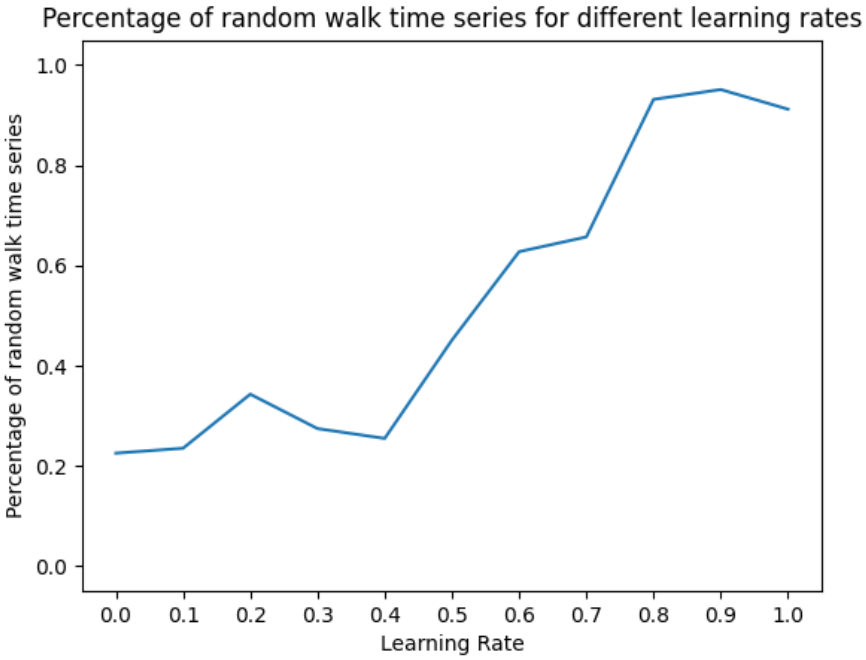


Figure 4.8: Ratio of TDA time series that show 'random walk' behavior for different learning rates.

5. Results

This chapter will describe the results achieved by the six prediction methods and evaluate their results for both time series forecasting and negotiation outcome prediction. The latter will be divided into outcome utility and outcome agreement prediction.

5.1 Time Series Forecasting

In general, DeepAR outperforms the other five methods in TSF accuracy. It achieves the lowest overall CRPS scores for the different cutoff points and learning rates. Figure 5.1 shows the spread of all CRPS scores realized by the TSF methods. This is for predictions made across all cutoff points and learning rates, and thus encompasses many different types of negotiations.

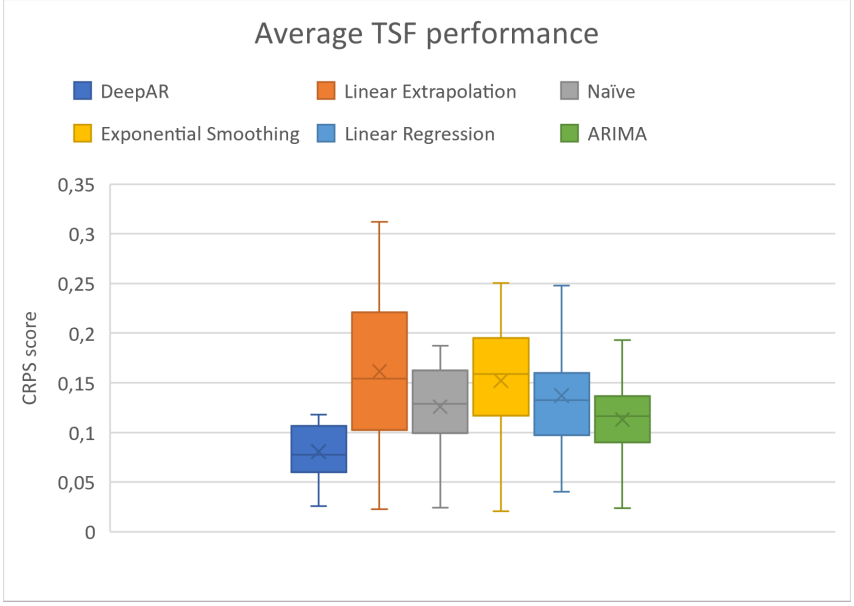


Figure 5.1: Box and Whiskers plot showing the average performance of all TSF methods and their spread.

Method	DeepAR	ES	ARIMA	LE	LR	Naive
Average	0.081	0.152	0.113	0.161	0.137	0.126

Table 5.1: Table showing the average CRPS score achieved by all 6 methods in time series forecasting

Figures 5.2 - 5.5 give us a closer look at how the six methods performed for the different parameter settings.

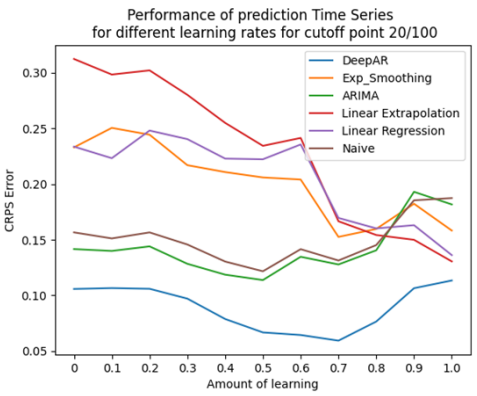


Figure 5.2: CRPS score achieved by six different forecasting methods, across eleven learning rates, for cutoff point 20.

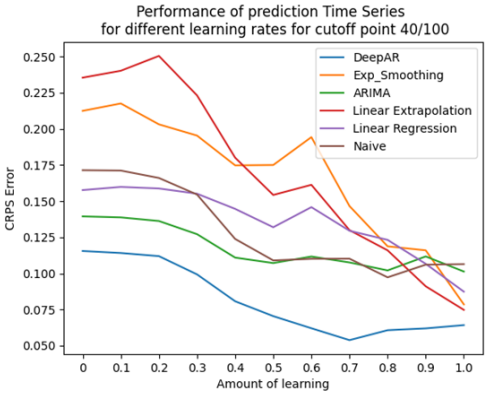


Figure 5.3: CRPS score achieved by six different forecasting methods, across eleven learning rates, for cutoff point 40.

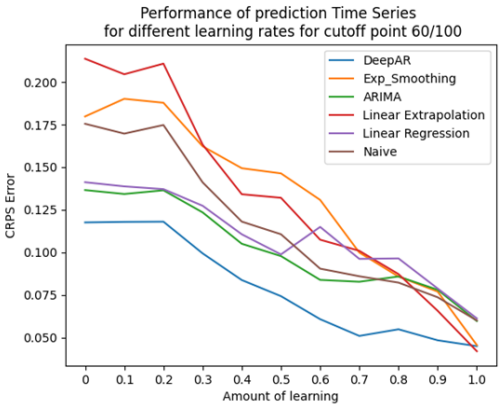


Figure 5.4: CRPS score achieved by six different forecasting methods, across eleven learning rates, for cutoff point 60.

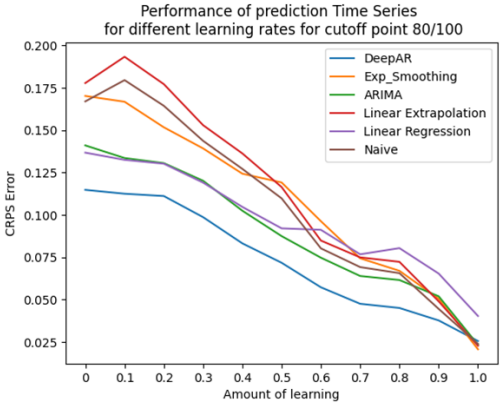


Figure 5.5: CRPS score achieved by six different forecasting methods, across eleven learning rates, for cutoff point 80.

We can see here that DeepAR is only marginally surpassed for learning rate 1.0 when the cutoff point is at 60 or 80. However, all methods achieved high

scores for these parameter settings. The average CRPS score DeepAR realized for TSF is 0.081. This is twice as good as the worst benchmark, Linear Extrapolation, which got an average CRPS score of 0.16. Figure 5.1 gives a good insight into the stark difference between DeepAR and the other 5 methods, which shows that the bulk of DeepAR's predictions lie well below the biggest mass of all benchmark methods.

5.1.1 DeepAR

Although this makes it clear that DeepAR made the best forecasts compared to these benchmarks, it does not give us insight into whether this constitutes a strong result by itself. Therefore, we will now consider the performance of DeepAR in more depth. See figure 5.6 and table 5.2 for the TSF results achieved by DeepAR.

5.1.1.1 Cutoff points

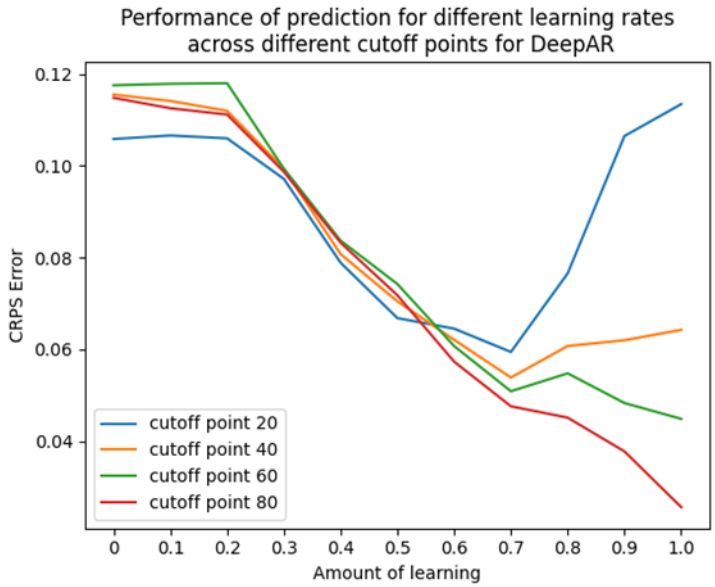


Figure 5.6: CRPS score that DeepAR achieved for the forecasting of negotiation time series for eleven learning rates, across four cutoff points.

Taking a closer look at figure 5.6, several observations can be made. First, the results hardly change between learning rates 0 - 0.2. This is because this increase in learning rate has a negligible real effect on the shape of the utility

graph. From 0.3 on, all curves show a downward slope. The different cutoff points only start diverging in error for a learning rate of 0.7 and higher. Before this point, all networks achieve approximately the same error score. The endpoints, for a learning rate of 1, are ordered from lowest to highest cutoff point in CRPS score. This makes intuitive sense, as it should be easier to predict a graph from which more information is known. Finally, what is likely the biggest anomaly in the results, the CRPS score for cutoff point 20 increases again from learning rate 0.7 onward. While the other cutoff points also show a stagnation from this point on, it is not as steep as cutoff point 20, which trends back towards its starting CRPS error value.

To explain this behavior, we ran some additional tests. These show that if DeepAR has too little information to make an informed forecast, it will make a safe bet and return the most average time series it has trained on. In our data set, these are negotiations with a learning rate of 0.7. So, a forecast made from cutoff point 20 will often result in a distribution that has a high resemblance to such a negotiation. Therefore, cutoff point 20 performs best at learning rate 0.7, after which it decreases in performance again. Also see figure 5.7. This problem disappears for later cutoff points as the network has more data on which to base its forecasts. We can also conclude from this that a later cutoff point increases the predictability of the time series.

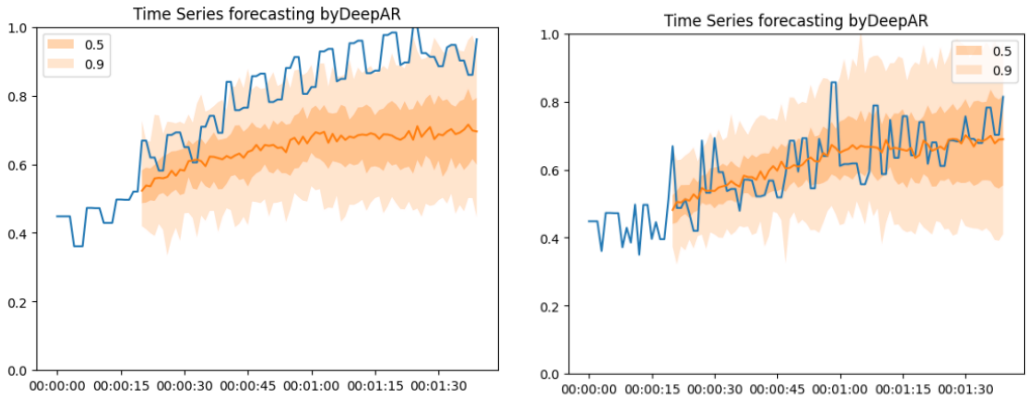


Figure 5.7: TSF of a negotiation with a high learning rate on the left, and an intermediate learning rate on the right. The forecast is made from an early cutoff point. Here, we can see that when the network does not have enough data to make an informed forecast, it makes an average prediction that corresponds to a learning rate of 0.7 best.

5.1.1.2 Learning Rates

Recall that the learning rate models behavior-based strategies, strategies in which the agent attempts to estimate the preference profile of their opponent and consider this in its bids. As such, a high learning rate causes the agent to make bids that are favorable to the opponent. In our results, it is clearly visible that the learning rate has a strong influence on forecast accuracy. CRPS error decreased for every cutoff point until a learning rate of 0.7, at which point cutoff points 20 and 40 increased again. The remaining cutoff points had their lowest error for a maximum learning rate. The reason for this is that the learning rate increases the predictability of the graph. Where we have shown a regular TDA to exhibit white noise characteristics, a higher learning rate introduces a strong trend to the trajectory. The algorithm learns to recognize such a trend and include this in its predictions. Negotiations with a high learning rate also have a decreased variance, resulting in a tighter spread around their mean value. This works in the advantage of the model, allowing it to make a more precise prediction with a tighter distribution. Such distributions are able to reach a low error score. Low learning rates on the other hand cause graphs to have a high amount of variance. The network responds to this by also making a more spread out distribution forecast, as this is the most realistic prediction it can make. Therefore, the lowest possible error rate is higher for lower learning rates.

This increased performance for high learning rates pulls into question whether it is still wise to use such strategies if it gives the opponent more opportunity to gauge your next moves, and turn this into their tactical advantage. Unpredictability may be a valuable weapon in negotiations; however, it can also result in a net loss for both parties. Using a strategy that helps both parties achieve a better result is ultimately the best outcome, and a predictable strategy may help achieve this goal.

5.1.1.3 Numerical Results

	Cutoff Point			
LR	20	40	60	80
0.0	0.106	0.116	0.118	0.115
0.1	0.107	0.114	0.118	0.113
0.2	0.106	0.112	0.118	0.111
0.3	0.097	0.099	0.099	0.099
0.4	0.079	0.081	0.084	0.083
0.5	0.067	0.070	0.074	0.072
0.6	0.064	0.062	0.061	0.057
0.7	0.059	0.054	0.051	0.048
0.8	0.077	0.061	0.055	0.045
0.9	0.106	0.062	0.048	0.038
1.0	0.113	0.064	0.045	0.026

Table 5.2: Table showing the different CRPS scores achieved by DeepAR across Learning Rates 0.0 - 1.0, and Cutoff Points 20 - 80. Performance for the forecasts of time series.

Since DeepAR made predictions for many negotiations in varying settings, the results are multi-faceted. It reached CRPS scores between 0.118 and 0.026, see table 5.2 for the exact scores. Recall the intuition behind CRPS. A CRPS score can be imagined as a distributive version of the Mean Average Error. Basically, it describes the distance between the distribution mass and the true outcome. So, a CRPS score of 0.1 points to the distribution being removed 0.1 "utility" away from the true outcome.

On average, the DeepAR forecasts were 0.081 utility removed from the true values of the bids made by the opponent. Is this an acceptable margin of error for the forecasts to be useful? It depends on the application. Let us consider the domain on which this network was trained, the party domain. Figure 5.8 shows how many real outcomes are within this range of 0.081 utility for different utility scores, for the first party domain profile.

At its peak at 0.41, this range can encompass up to 1185 unique outcomes. A negotiation that ends around the Nash equilibrium, usually somewhere between 0.7 and 0.9, can expect between 433 and 134 outcomes within this range. This may seem like much, but does not consider that there is a second agent also influencing the negotiation. This agent reduces the number

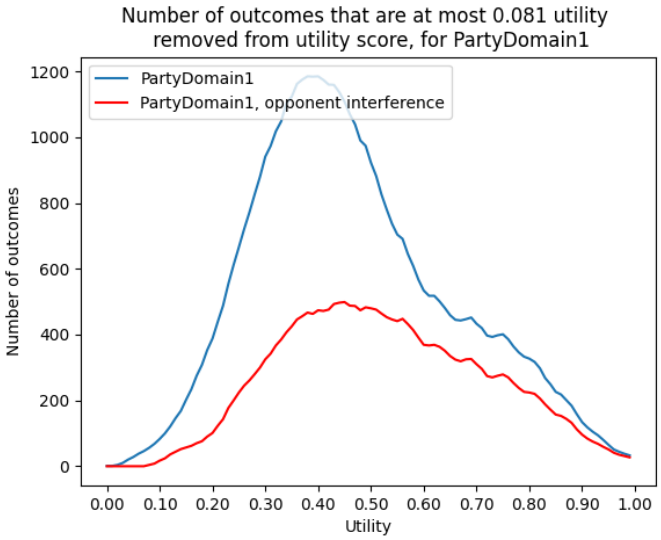


Figure 5.8: Graph showing how many outcomes are at most 0.081 utility removed from a certain utility score, for party domain 1. Blue is the regular number of outcomes, red the number of outcomes for which the opponent (PartyDomain2) also gets over 0.5 utility.

of possible outcomes as it will (usually) not accept outcomes that give it low utility. The exact effect of this differs per agent, depending on their concession rate. If we, however, assume the opponent agent will only accept outcomes that give it 0.5 or more utility (still a low utility value) this graph already decreases substantially, as indicated with the red line. Now, the peak has 499 possible outcomes, and the Nash equilibrium range between 310 and 96 outcomes. Since the entire party domain consists of 3072 outcomes, this is between 10% and 3% of the outcome space. Note that these values can still differ hugely between profile combinations, but serve to give the reader some better idea of possible outcomes. We see that even for the best TSF method, its error compromises a significant part of the outcome space. This is most detrimental in cases where the exact outcome of the negotiation is desired. Our network is not optimized for these situations, as exact outcomes are not integrated into its architecture. If the user, however, just wants an estimation of its opponent's future utility graph, these results can certainly help. Most importantly, this first experiment is not the end point of this research. Its main function is to use these forecasts in outcome prediction, the results of which will be evaluated next.

5.2 Outcome Prediction

The second experiment used the forecasts created by our TSF methods to make outcome predictions and calculate the agreement probabilities. It does this by finding the intersection between two forecasts using Monte Carlo Methods, as described in 3.3.

5.2.1 Utility Prediction

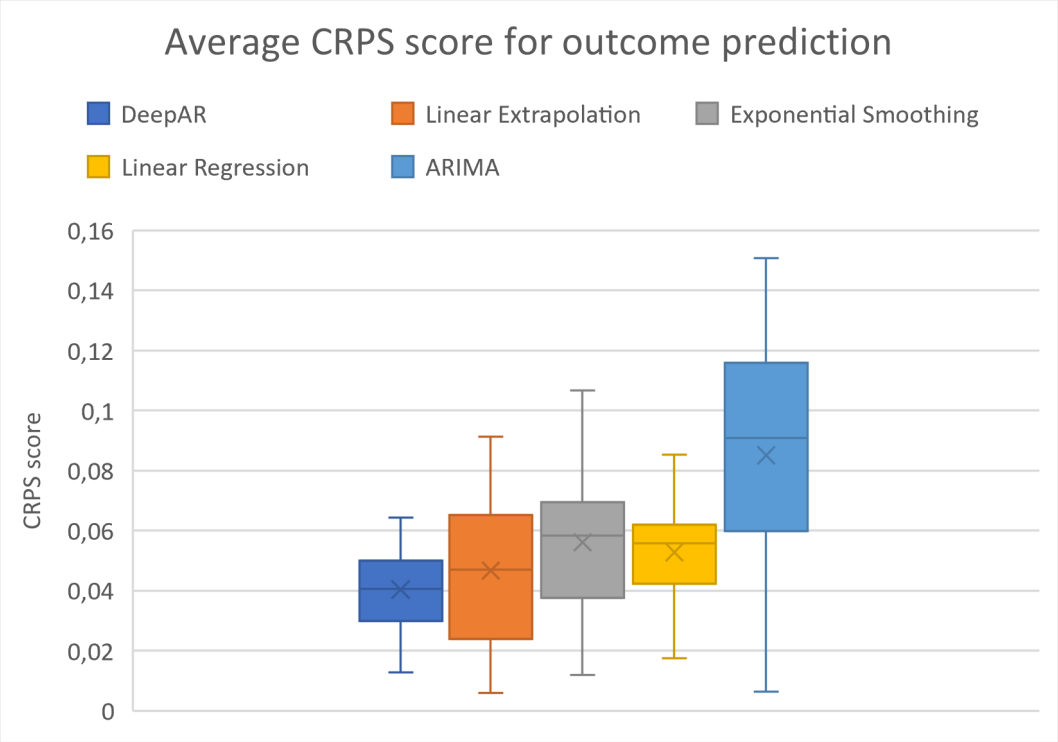


Figure 5.9: Box and Whiskers plot showing the average CRPS scores achieved in outcome prediction by the five methods

Method	DeepAR	ES	ARIMA	LE	LR
Average	0.040	0.056	0.085	0.047	0.052

Table 5.3: Table showing the average CRPS score achieved by 5 outcome prediction methods.

Figure 5.9 and table 5.3 show how the five methods performed on average in outcome prediction. Once again, DeepAR performs best overall, but with a smaller margin than for experiment 1. The worst benchmark this time

is ARIMA, which got an average CRPS of 0.085, barely improving its experiment 1 score. Linear extrapolation, on the other hand, shows much improvement, enjoying a larger spread of outcomes and, at its best, surpassing DeepAR for some parameter settings. However, we cannot compare these methods as easily as we did in the previous experiment. It is important to note that there is a difference in how this error is calculated between distributive methods (DeepAR, ARIMA) and pointwise methods (exponential smoothing, linear regression, linear extrapolation). This is because pointwise methods give either a single outcome prediction or no prediction at all, if they predict a breakoff. This means that if a pointwise method were to predict a breakoff, this prediction can not be considered in calculating the average CRPS error, as there is nothing to compare against. This difference is only relevant for negotiations that ended in an *agreement* but for which a *breakoff* was predicted, or a "false negative"[1]. Distributive methods, on the other hand, will still predict some utility outcome, even if the method gives a breakoff a much higher probability. In these cases, even if only 2% of Monte Carlo samples find an intersection, this 2% is still taken into account when calculating the average CRPS score. Also see table 5.4. Because of this, the error scores for the pointwise methods in figure 5.9 can make their performance seem better than it was. However, this also largely depends on how well they did at agreement prediction, which we will get to later.

	TP	FN	FP	TN
Distributive Method		Usually Considered		
Pointwise Method				

Table 5.4: Table showing whether predictions are taken into account when calculating CRPS score for outcome prediction. Green = taken into account, red = not taken into account. TP = true positive, FN = false negative, FP = false positive, TN = true negative.

[1]Note that the terminology such as "false negative" does not strictly apply to distributive methods, since they produce an agreement probability rather than a binary answer. Here, the terminology refers to what the method primarily predicts.

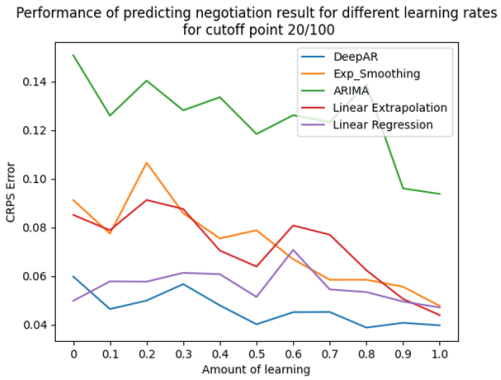


Figure 5.10: CRPS score achieved by five different forecasting methods in outcome prediction, across eleven learning rates, for cutoff point 20.

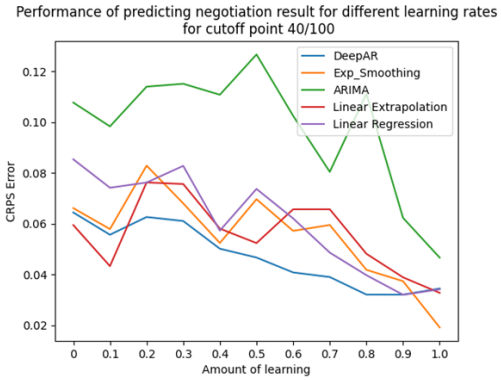


Figure 5.11: CRPS score achieved by five different forecasting methods in outcome prediction, across eleven learning rates, for cutoff point 40.

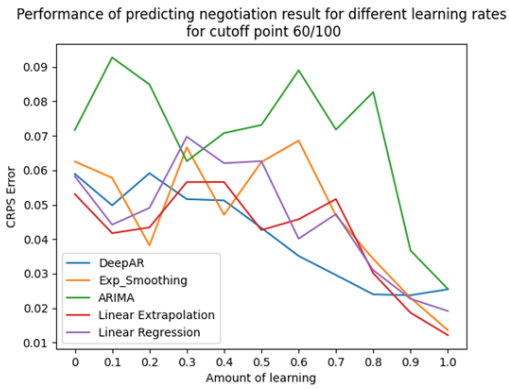


Figure 5.12: CRPS score achieved by five different forecasting methods in outcome prediction, across eleven learning rates, for cutoff point 60.

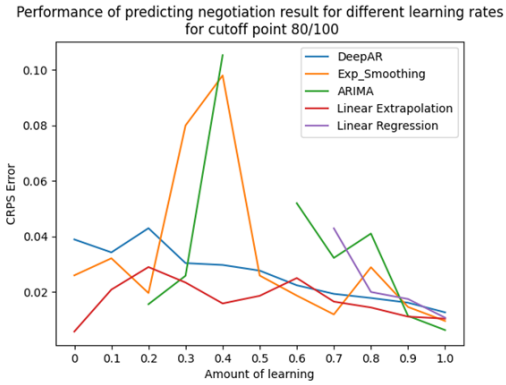


Figure 5.13: CRPS score achieved by five different forecasting methods in outcome prediction, across eleven learning rates, for cutoff point 80.

Figures 5.10 - 5.13 show how all five[2] prediction methods performed for different parameter settings. Here we see that DeepAR still performs best overall, but is outperformed for some specific parameter settings. Especially for high cutoff points (80) linear extrapolation achieves a better CRPS score. For lower cut-off points, however, DeepAR makes the most accurate predictions. We can see that all methods improved their CRPS scores for the second experiment, compared to the first. Not only that, the shapes of the graphs look quite different compared to the first experiment. Take

[2]Note that the Naive method is not included in these figures as it only predicts constant lines that can never cross, and thus never predicts an agreement.

DeepAR's graphs, for instance, seen in figure 5.14. It previously had a "u-shape" for cutoff point 20, as well as milder bends for cutoff points 40 and 60. In experiment 2, these graphs show a downward trajectory, decreasing their CRPS error until learning rate 0.8, after which the scores stabilize.

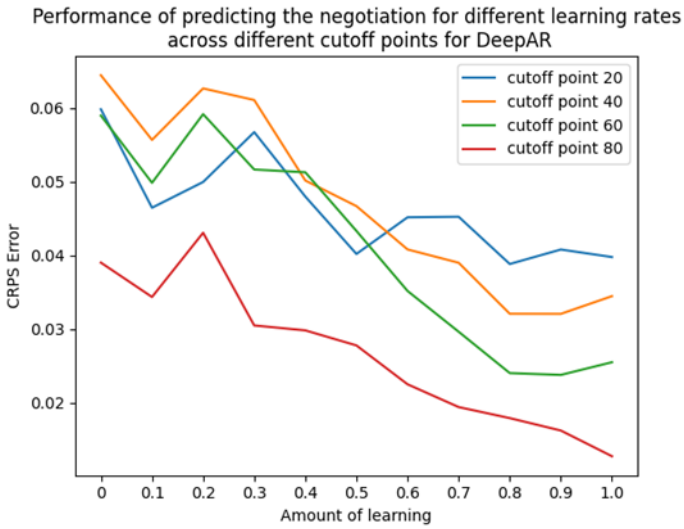


Figure 5.14: CRPS score for the prediction of negotiation outcomes by DeepAR for varying cutoff points and learning rates.

If the results of the first experiment simply translated to the results of the second experiment, we would expect these graphs to show a higher degree of similarity. Instead, the results are improving overall. A comparable change can be seen for the benchmark methods, which show an even larger relative decrease in error score. Why is this? We believe that this decrease can be explained by two factors:

Firstly, the distributions obtained from forecasting the agent's own concessions. Recall that experiment 1 tests only the forecasting of the opponent's bids, whereas the predictions made in experiment 2 result from combining the forecasts of the agent's own and its opponent's concessions. The own concession has a higher degree of predictability, as it has a strong trend and does not resemble white noise nor a random walk. As a result of this, the forecasts made for an agents own concession curve have a lower error score, and thus increase the overall performance. See figures 5.15 and 5.16 for examples of such forecasts.

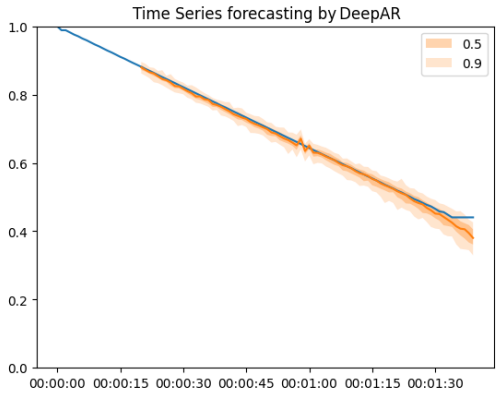


Figure 5.15: Time series forecast of own concession curve by DeepAR.

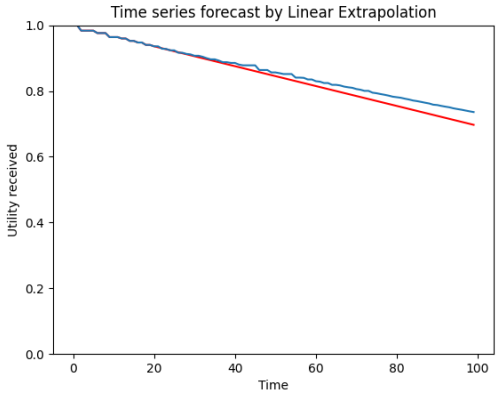


Figure 5.16: Time series forecast of own concession curve by LE.

The second reason is that outcome utility prediction for pointwise methods has a bias toward including accurate forecasts in its evaluation. This is because these methods only consider correctly predicted agreement negotiations when calculating the CRPS score. So, those inaccurate forecasts that decreased the overall error score in experiment 1 are not included in the CRPS calculation in experiment 2.

5.2.1.1 DeepAR

	Cutoff Point			
LR	20	40	60	80
0.0	0.060	0.064	0.059	0.039
0.1	0.046	0.056	0.050	0.034
0.2	0.050	0.063	0.059	0.043
0.3	0.057	0.061	0.052	0.030
0.4	0.048	0.050	0.051	0.030
0.5	0.040	0.047	0.043	0.028
0.6	0.045	0.041	0.035	0.022
0.7	0.045	0.039	0.030	0.019
0.8	0.039	0.032	0.024	0.018
0.9	0.041	0.032	0.024	0.016
1.0	0.040	0.034	0.025	0.013

Table 5.5: Table showing the different CRPS scores achieved by DeepAR across Learning Rates 0.0 - 1.0, and Cutoff Points 20 - 80. Performance for the prediction of utility outcomes of negotiations.

Table 5.5 shows the exact CRPS scores DeepAR achieved for different parameter settings. These values range from 0.064 to 0.013, with a total average score of 0.040. If we run the same analysis as in section 5.1.1.3, we can see that this makes for a steady decrease in outcomes within the utility range. Now, the peak is at 611 outcomes without opponent interference and 264 with. Utilities around the Nash equilibrium now have between and 223 and 77 outcomes without interference, and between 154 and 52 with interference, or between 5.0% and 1.7% of the outcome space. Also see figure 5.17. Another advantageous result is the worst-case prediction of DeepAR, which is still an amendable 0.064 utility removed from the true outcome. There is still room for improvement, and there are many opportunities to do so, but this first result is promising for neural network applications in outcome prediction.

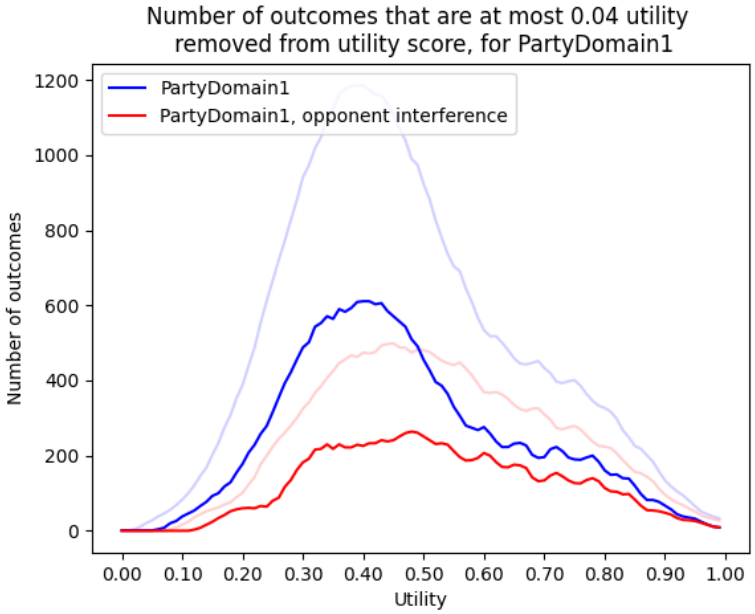


Figure 5.17: Graph showing how many outcomes are at most 0.04 utility removed from a certain utility score, for party domain 1. Blue is the regular number of outcomes, red the number of outcomes for which the opponent (PartyDomain2) also gets over 0.5 utility. The transparent lines show the results for 0.081 utility, as seen in the previous experiment.

5.2.2 Agreement Prediction

However, utilities form only half of the outcome. The other important aspect is the agreement probability prediction. This category is split up into two sections: predictions made for negotiations that ended in an agreement (agreement negotiations), and predictions made for negotiations that ended in a breakoff (breakoff negotiations). Figures 5.18 and 5.19 show how all the methods performed on average for predicting both agreement and breakoff negotiations. Important to note here is the difference in how results are calculated between distributive and pointwise methods. For distributive methods, these averages simply represent the average agreement probability that was predicted by the method. Pointwise methods, on the other hand, do not produce an agreement probability but rather make a binary prediction between an agreement or a breakoff. As such, for the pointwise methods, these averages represent the ratio of negotiations for which the method predicted an agreement.

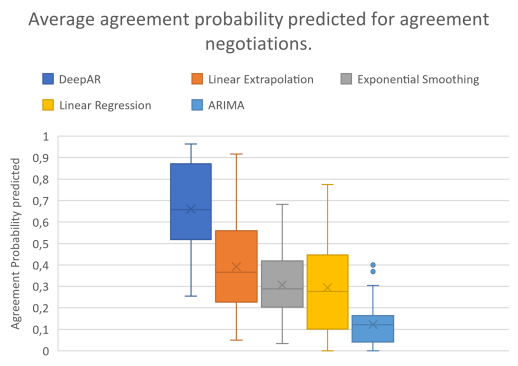


Figure 5.18: Box and Whisker plot showing the average agreement probability predicted by several methods for agreement negotiations. If a method did well, it is closer to the top (full agreement predicted) of the plot.

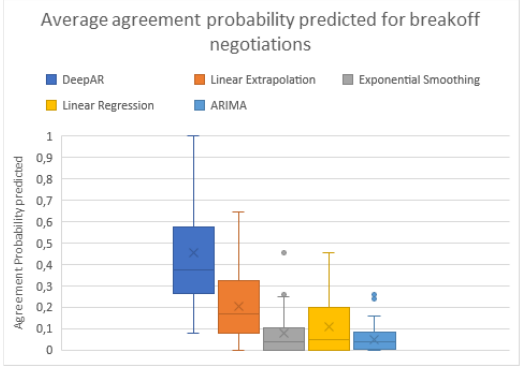


Figure 5.19: Box and Whisker plot showing the average agreement probability predicted by several methods for breakoff negotiations. If a method did well, it is closer to the bottom (full breakoff predicted) of the plot.

For figure 5.18 a 1.0 represents true positives, while a 0.0 represents false negatives. For figure 5.19, a 1.0 represents false positives, while 0.0 represents true negatives. These plots show that the benchmark methods have a bias towards predicting breakoffs, while DeepAR stays closer to the cen-

ter. As such, DeepAR performed far better in predicting agreement negotiations, whereas it did the worst in predicting breakoff negotiations. Note that in the entire data set, agreement negotiations constitute 77% of negotiations, which means that a method that performs better for this type also increases performance in the whole data set. Despite its subpar ability to predict breakoffs, DeepAR still performs best overall in discriminating between the two types of negotiations.

5.2.2.1 DeepAR

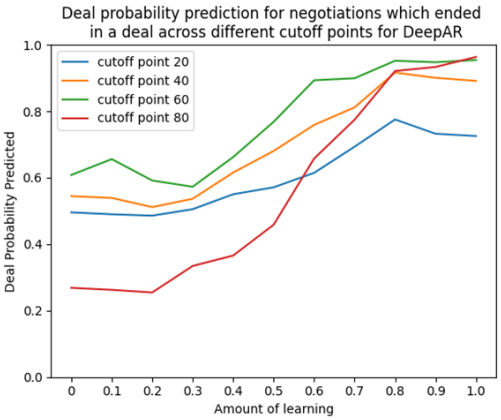


Figure 5.20: Proportion of agreements predicted by DeepAR for agreement negotiations, for four cutoff points and eleven learning rates.

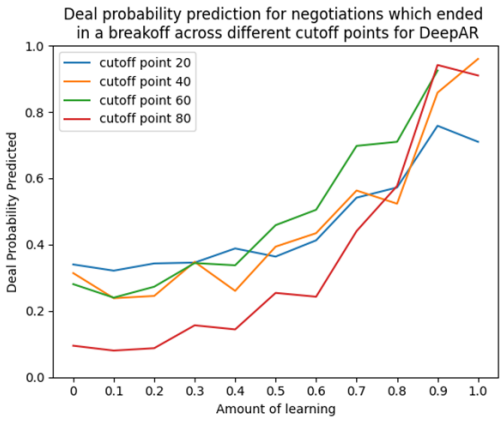


Figure 5.21: Proportion of agreements predicted by DeepAR for breakoff negotiations, for four cutoff points and eleven learning rates.

Figures 5.20 and 5.21 show how DeepAR performed for different parameter settings. It performs decently in agreement prediction, but there is still room for improvement. Looking at these graphs, some things catch the attention. First, the network generally attributes a higher agreement probability to agreement negotiations than to breakoff negotiations. This is desirable as it demonstrates that the network is capable of making a distinction between the two negotiation types. Something else to note is that both graphs show an increase in agreement probability for higher learning rates. We believe this is because the method mirrors the data set in its output, as is logical for a deep learning network. If we look at the ratio of agreement and breakoff negotiations for high learning rates, this does indeed include very few breakoffs; see figure 5.23. So, for those negotiations, it may be "logical"

to be biased towards predicting agreements. A similar observation can be made for cutoff point 80. Here, we see DeepAR showing a bias towards predicting breakoffs for both graphs. Once again, this can be explained by looking at figure 5.22. Here, we see that cutoff point 80 had a relatively low number of negotiations that ended in an agreement, and the network adapts to this by predicting a lower agreement probability. The reason why this cutoff point saw fewer agreements is because many agreement negotiations already reached an agreement before this cutoff point, and thus were not included for analysis.

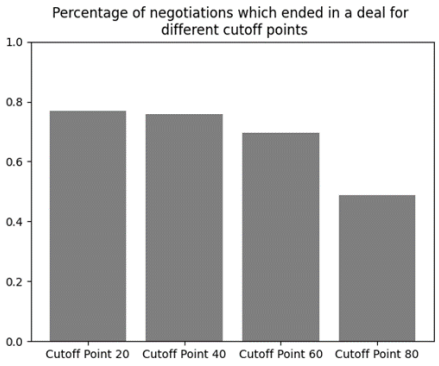


Figure 5.22: Proportion of negotiation which ended in an agreement for the four different cutoff points

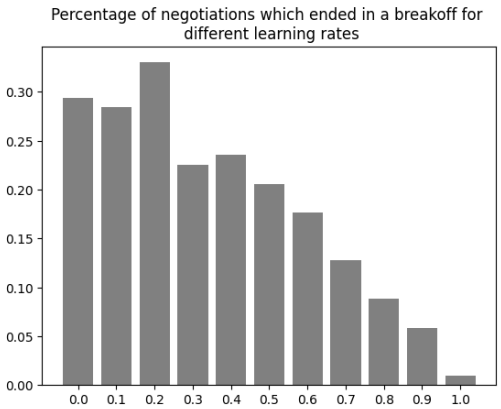


Figure 5.23: Proportion of negotiations which ended in a breakoff for the eleven different learning rates.

It is also possible to look at DeepAR's results via a simpler metric. This is done by considering every negotiation for which it predicts an agreement with over 50% confidence as predicting a full agreement, and less than 50 % as a breakoff prediction. In this way, we treat DeepAR like a binary classifier. This is somewhat reductive, but may give an easier frame of analysis and allow us to make calculations with precision and recall. Table 5.6 shows how DeepAR performed on the entire data set, when applying this definition. Interestingly, this data show that DeepAR managed a prediction accuracy for both agreement and breakoff negotiations of 82%.

True Positives	2794	True Negatives	826
False Negatives	604	False Positives	176

Table 5.6: Table showing how DeepAR classified the different negotiations.

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \qquad F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In machine learning, it is common to evaluate classifiers (especially those dealing with imbalanced data sets) using precision and recall. These can then be used to calculate the F1-score. Precision can be seen as the fraction of agreement predictions which the program correctly classified, while recall indicates what fraction of agreement negotiations were correctly predicted as such. DeepAR achieved a precision of 0.94 and a recall of 0.82. Its F1-score is 0.876, which is generally considered a good score. The highest possible F1-score which can be achieved is 1, if the network gets perfect precision and recall.

When we look at the classifier from this lens, the results look very favorable. Part of the reason for this apparent "difference" is that the plain accuracy reported earlier did not account for the number of negotiations each category represented. Take, for example, a breakoff negotiation with a high learning rate. Such a negotiation is difficult for the network to predict correctly, but also represents a very small part of the data set.

6. Discussion and Limitations

This chapter will explore some topics that have not been adequately discussed in previous chapters. In addition, we will discuss the limitations of this thesis. This will serve as a starting point for future research suggestions in the field of automated negotiation prediction.

6.1 Discussion

6.1.1 Metrics

In the previous chapter, the difficulty of comparing distributive and pointwise prediction methods was highlighted. Here, the comparison of utility predictions by themselves is not an issue, as CRPS is able to handle both types of data in a way that allows them to be compared quite well. The issue in comparing their outputs lies in the different way they handle agreement predictions. As already touched upon, distributive methods give every negotiation a probability that the negotiation ends in an agreement, whereas pointwise method simply either predicts the negotiation ends in an agreement or does not. From this difference in prediction type comes a difference in how the data is analyzed as well, as was discussed in 5.2.1

We already saw that reporting only the accuracy scores of the included methods made for a difficult comparison because of their different calculations. Moreover, such an analysis does not account for the proportion of negotiation types included in the data set. For this reason, we opted to evaluate DeepAR using precision, recall, and F1-score as well. These metrics are common in data science because they provide a complementary look at the evaluated network and account for imbalanced data. In many situations, an increase in precision can mean a loss in recall and vice versa. Because of its widespread usage, using this metric also provides us with an easy

communication tool towards others in the field.

That is not to say that these metrics are without problems. For DeepAR, using a metric designed for binary classification ignores its distributive nature, which is an important feature of our designed solution. Binary classification regards a confidence level of 51% equally to one of 99%. Moreover, an integral assumption of precision and recall is that one class is the "relevant" one. As such, the calculations made rely on this assumption of having one important and one unimportant class. In our case, however, an agreement is not more or less desirable than a breakoff, but rather both are plausible negotiation outcomes. We can also calculate the F1-score with breakoffs as our relevant class, which gives 0.68. Still a decent score, but lacking compared to our ability to classify agreements.

6.1.1.1 Different Metric

The reason the metrics were chosen as such is that they offer a way to compare the results for two different types of methods, while also being relatively easy to understand. Nevertheless, the experiments may have benefitted from a more extensive metric that takes into account all these factors at once, so that the different methods can be compared one on one.

A suggestion for such a metric could be one that, for the distributive methods, adds a weight factor to the utility score outcome when calculating the average. This weight factor would be scaled so that highly confident predictions contribute more to the total average and vice versa. One can argue that pointwise predictions already implement a version of this, as their included predictions are all 100% confident. Therefore, the distributive methods and pointwise methods should be easier to compare with this metric. On the other hand, the distributive metrics would suffer in their interpretability, making it harder to gauge what the total experimental results mean.

An additional approach that can improve how well all methods' agreement probabilities can be compared is to design one metric encompassing both agreement and breakoff negotiations, which takes the average difference between the ground truth (either 100% agreement or 0% agreement) and the

predicted probability. For a pointwise method, this predicted probability would always be either 100% agreement or 0% agreement as well. Like the previous suggested metric, this may improve how well the different methods can be compared, but also decreases interpretability on an individual basis. As this research is first about DeepAR and second about the benchmark methods, we opted to use metrics that are easier to interpret.

6.1.2 Wrong Predictions

As with most classification tasks, negotiation outcome prediction also suffers from incorrect classifications. Taking a closer look at the results, it seems that most incorrect predictions were made for negotiations that did not behave similar to negotiations with the same parameters. Take, for example, negotiations with a high learning rate. This category consists mainly of agreement negotiations. As such, DeepAR tends to predict a high agreement probability for these negotiations. However, there are still cases where such negotiations end in a breakoff. These cases often get predicted wrongly. The same behavior can be observed across the entire data set for different parameters. On the one hand, this does show that our network learns to distinguish different types of negotiations based on the shape of the time series and make predictions based on it.

There are also cases in which DeepAR gives a very low agreement probability to negotiations that do end in an agreement. To some extent, this is also due to the unpredictable nature of negotiations. The difficulty of correctly predicting an agreement for negotiations on an individual basis is exemplified by figure 6.1, which shows how the erratic behavior of these time series can cause negotiations to still end in an agreement, even if the trend so far indicated otherwise. Negotiations of this type can put a limit on the optimal performance of any method that attempts to predict them. However, across the board, methods should be able to extract general trends from data and use this to make informed predictions. It just so happens that sometimes the true outcome is not what appears to be the most likely one. The advantage of DeepAR in these situations is that it still gives the agent a utility

prediction, which the agent can use in whichever way it seems fit.

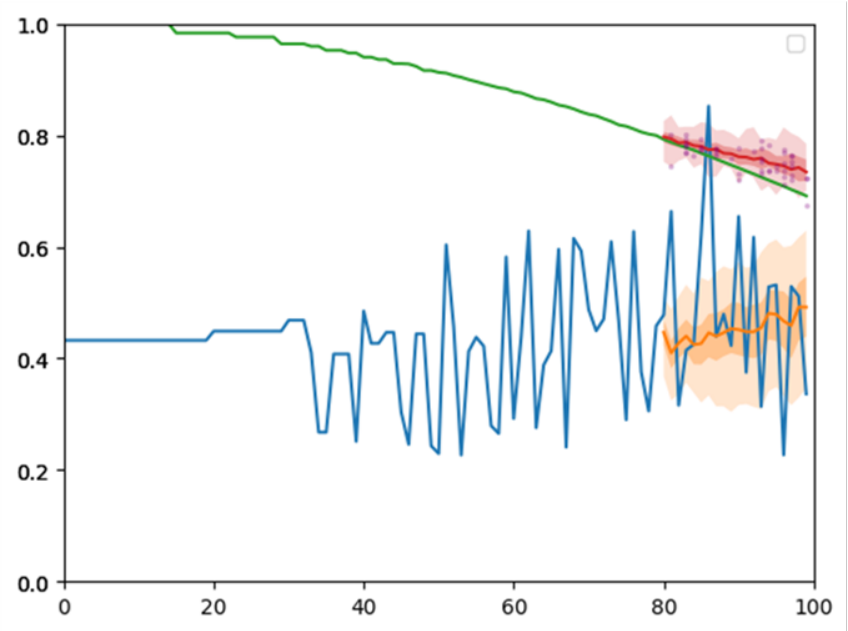


Figure 6.1: Example of a negotiation that unexpectedly ended in an agreement

From this, one may conclude that the most effective way to manage an opponent that uses an outcome predictor is to radically change the strategy used in the final stage of the negotiation. If there is no correlation between the bids made, it will be impossible to accurately make a TSF. In these situations, the network may benefit from an incorporated opponent model to gauge the plausible outcomes.

6.1.3 Advantages and Disadvantages of DeepAR

Bringing us to another discussion point: is DeepAR still the preferable solution if we take the setup costs into account? The benchmark methods used in this research are all "plug and play", meaning that they can be applied to any negotiation from any moment, and require no training[1]. Deep learning, on the other hand, gains in performance at the cost of flexibility, which it lacks compared to all benchmark methods. During training, DeepAR requires a data set representative of the negotiations it will be predicting, and

[1]Note that for ARIMA, the parameters must still be set. Although this requires no training, they do still differ per time series and can also take much time.

is also limited to a preset input and output length. On top of this, training can take multiple hours depending on the size of the data set, convergence rate, and processing capacity. If someone wanted a method to just quickly predict a negotiation, deep learning is, at the current state of research, not their best bet.

Nevertheless, we believe that it is by far the most worthwhile method in automated negotiating research, for a number of reasons:

1. **Performance** The most straightforward reason, the performance of deep learning. This research has shown that DeepAR outperforms every benchmark, and we believe these results will only improve with more research. Deep learning networks exist in many possible permutations far outside just RNN and LSTM networks, with even more possibilities regarding hyperparameters and training settings. As this research's primary focus was not to design the optimal deep learning network, but rather investigate whether such a network could be feasible at all, we believe research dedicated to this endeavor could result in a network outperforming the one described in this thesis. The benchmark methods, on the other hand, are quite simple in their workings and will likely gain little from additional research. We also believe additional research may remove some of the earlier described barriers by making a network that can work for many different outputs, or is trained on a broad selection of negotiation, and thus can be used for many negotiation contexts.
2. **Insight** DeepAR returns two outputs to the agent, a distribution of outcomes and an agreement probability. Additionally, DeepAR can provide the forecast distributions that it used in generating the outcome prediction. This collection of outputs gives the agent insight into the negotiation, which it can then use to decide its next move(s). Compare that with pointwise methods that give only the predicted outcome. Although valuable, they provide no indication of the confidence in this prediction nor of how likely other predictions are. Once more, further research may be able to improve this category by adding

additional outputs to the network, for instance, having the network also produce material outcomes of the negotiation.

3. **Customizability** Another advantage of deep learning that has been useful in many areas is its customizability. The networks can be tailor-made for the specific use case at hand. If you know in what setting and against what type of opponents you will negotiate beforehand, this gives deep learning a large advantage over other methods. Moreover, the network can be made lighter for small usecases, or apply a new likelihood function for different data distributions. This customizability argument can be made for ARIMA as well to some degree, since it uses three parameters to fit various types of data. However, ARIMA has performed far worse than DeepAR in this research, and deep learning is far more customizable.

Overall, choice of method still depends heavily on the context. Generally, the more time one is willing to invest in a solution, the more worthwhile deep learning becomes. For a quick prediction, a different method might be better for now, at least until a more flexible deep learning solution is developed for automated negotiations.

6.2 Limitations

This section will dive into the limitations of this study and provide insight into the constraints encountered during the research. We hope these lessons are informative and inspire future researchers in this field, helping them achieve ambitious results.

6.2.1 Agents

After some consideration, we decided to limit the scope of this research to involve only time-dependent agents. The TDA uses a purely time-based strategy and is therefore deterministic, with a methodology that is dependent on only a few parameters. These parameters allowed much control over the shape of the agents' utility graph, which meant the effects of dif-

ferent shapes could be investigated well, and the boundaries of the research well defined.

Nevertheless, this decision was not without drawbacks. One of the major obstacles we encountered regarding TDA is its shape for low learning rates. Although technically a deterministic strategy, in practice, the opponent's utility is chosen at random for every new time step. On the one hand, this formed an obstacle in terms of the predictability of the TDA, where the network had a hard time making sense of the time series. This led us to introduce new variables (learning rate, target utility) that allowed more control over the TDA in hopes of reducing this problem. On the other hand, this did allow us to show the effect an opponent model (in the form of learning) can have in reducing randomness from the made bids and thus the predictability of the negotiation. One may wonder what effect a different choice of agent would have had in this matter. If this hypothetical agent had adhered to a strategy that is easier to predict, the results could have been greatly affected.

This brings us to another point, the narrow scope of the agents in this research. While making the research more focused, the decision to include only one agent also made it so this research's applicability to the broader world of (automated) negotiation is still uncertain. There is a plethora of additional possible strategies and negotiation settings that may be quite different from a bilateral negotiation between TDAs in the party domain. This research confirmed that there is potential for predicting automated negotiations, but the case for a broader selection of agents (and other parameters) remains to be made.

6.2.2 Learning

The learning in this research means to simulate an agent that uses a behavior-based strategy. That is, a strategy in which the agent uses some opponent model to estimate the preference profile of its opponent. This can then be used in an attempt to make bids that are profitable for both negotiating parties. However, it is questionable whether the learning model we developed

fully captured this behavior. A regular agent using an opponent model may start off having no idea about its opponents' preferences, and slowly gain this knowledge throughout the negotiation. As such, you could say that its learning rate is increasing with time. Our model, on the other hand, assumes some amount of knowledge from the beginning, which stays consistent throughout the negotiation. So, in this sense, the model does not fully model learning, but rather insight into the opponent. This design was chosen because we opted for the learning rate to be simple and understandable, rather than highly realistic. In this way, it is clear how the opponent's learning influences their predictability, rather than a method that is more realistic but for which it is hard to assess what facet causes a certain effect. A model is, by definition, an abstraction and can never be the same as that which it models. We believe that this model strikes a good balance between realism and transparency. However, it would be interesting to see how a true learning agent would influence the research.

6.2.3 DL network

As already touched on, the current deep learning network is quite rigid in its settings. It can only give predictions for rounds 20, 40, 60 and 80 and will always predict until exactly round 100. If one wanted to apply the network in a different context, this would require the training of a new network with the desired specifications. This limits the broad applicability of the network. Fortunately, such rigid settings are not a hard requirement for deep learning networks in general, meaning that this could be mitigated in the future.

Additionally, the amount of resources it takes to train and test a network means that the current network settings (hyperparameters) were not fully tested against other settings. Rather, due to constraints in time and resources, when a network was developed that performed good enough, this network was kept. However, it is plausible that some difference in hyperparameters, be it in the network structure, loss function, or training settings, could have improved the performance even more. The goal of this research was not to create an optimal network, but rather to test whether such an

approach can prove fruitful at all.

6.2.4 Own utility graph prediction

In the first experiment, we opted to only test the performance of the algorithms in forecasting their opponent's utility graph, for this was seen as the main challenge in TSF. Forecasting the own utility graph was seen as an arbitrary task, based on earlier experiments. This still holds for the most part, as predicting the opponent's utility graph is the harder challenge of the two (also see section 2.3.1.1). However, it may still have been insightful to test whether the forecast error for the own utility graph is sufficiently low, as well as whether there is a large difference in forecast accuracy between the benchmarks. The clearest advantage is that this would give additional insight in the process of going from time series forecasts to outcome predictions. Currently, the results show some discrepancy between experiment 1 and experiment 2. Where the first shows DeepAR as the clear preferred method in TSF, the second is more ambiguous. Part of this puzzle may lie in the ability of these methods to predict their own utility graph.

However, one can argue that predicting the own utility graph is, in this case, irrelevant. A TDA knows from the start what trajectory it will follow and thus should not have the need to still predict it. Instead, it could just use its own future utility graph to find the intersection with the opponent's prediction. Although this line of reasoning holds for TDA, the same cannot be said for all types of agents. Many agents may themselves be unsure about their future utility graph, and be dependent on the opponent's moves, as well as some inherent randomness. Still, one can argue that these agents will know better what direction they will take themselves. This is true for many agents, but removing the own prediction would also decrease the wide applicability of the research even further, as the user would have to insert their (possible) future utility graph themselves, as opposed to it being included in the framework.

Nevertheless, a small additional test was run to gain some insight into the above question. Here, the second experiment was repeated for a minor sub-

set of negotiations, but with the true utility graph of the own agent used to find the intersection, instead of the forecast. We found that doing this improved the performance for all methods, but still maintains the same relative distribution of the performances between the methods. So, no single method improved disproportionately much. This suggests that the in- or exclusion of a prediction of your own utility graph is not responsible for the relative ordering in prediction method performance.

6.2.5 Intersection Method

To be able to make a TSF, the true outcomes are converted to utility time series. The forecasts made based on these time series do not take into account what can or cannot be an actual negotiation outcome. Rather, it returns utilities that it believes are logical continuations of past observations. For this reason, it is possible for the forecasting method to make a forecast that does not have any real negotiation outcomes surrounding it. This results in the intersection method finding outcomes that equally do not have any associated real outcome. This was not a problem in this research because the party domain is densely spread out over the outcome plane. Therefore, there was always an outcome close to the predicted utility. However, for sparse outcome spaces, this may present an issue. For such outcome spaces, the chance to return a utility that does not have an associated outcome increases. In these situations, it may be wise to adjust the model or intersection method so that it takes into account the outcome space when making predictions.

Another issue in the current intersection method is that the intersection is always made from the perspective of our own agent. When the forecasts cross each other, that is what we count as having reached an agreement. The analysis was chosen like this as it is the only way to determine that an agreement is reached from time series forecasts. In reality, however, this is only true when our own agent ends the negotiation with an agreement, since it is viewed from their perspective. As such, the method does not consider that the opposing agent may not view the utilities as having crossed yet at

all. Fortunately, this only presents an issue in situations where the opposing agent would have terminated the negotiation earlier. This is because if they wanted to end it later, our agent already beat them to the punch, meaning that our agent "determines" the outcome. Moreover, for these negotiations terminated by the opponent, our own agent receives a higher utility since the negotiation is ended earlier in its concession curve.

6.2.6 Applicability

Many algorithms and techniques in the field of automated negotiation (so far) have primarily seen usage in simulated environments, such as Genius, and in the annual ANAC negotiation competitions. In this context, the primary 'goal' is improving the outcomes achieved by the agent in various negotiation formats. The idea is that if these agents are made to be good in simulated environments, this will translate to real negotiation environments. This idea is also implicitly present in this research. For this purpose, the predicting method is valid. One can imagine putting different concession trajectories for their own agent in the program and evaluating the different outcome predictions. These can help the agent decide whether they want to change their trajectory for the possibility of gaining utility. Additionally, they can evaluate the associated risk of doing so by analyzing the distribution's spread and agreement probability. Agents involved in multiple simultaneous negotiations can decide which negotiations are most worthwhile to spend computational resources on. However, for both scenarios, to be able to fully reap the benefits of this research, more work is needed. See 7.2.2 for a discussion of this.

Then on to non-simulated negotiations. Utilities should give us an indicative idea of how content a party is with a certain negotiation outcome (e.g., two different outcomes with the same utility should be evaluated equally). If this indication does a good job at describing the negotiation context, this research can be used effectively. Think of scenarios such as commerce or salary negotiation, where the concepts that are up for negotiation already have a clear numerical dimension. In these settings, the tactics described

above for simulated environments can be applied just as well[2].

However, there are scenarios imaginable where such a numerical translation of a party's preferences fails to fully capture the broader context of a negotiation. After all, it is hard to quantize abstract concepts such as "want", "need", or "desire" into numbers. You would have to transform the issues into utilities and the negotiation dynamics into a bilateral round-based setting. These computational negotiation mechanics are great for artificial agents to understand and do analysis on but do not capture the social mechanisms of a real (automated) negotiation. Even for fully automated negotiations, it is questionable how easily the settings can be translated to one that our model can interpret. In these situations, the research as it stands now falls short. Nevertheless, this research can be a step in the right direction to be able to do this in the future. For this reason, we have developed a modular application based on this research to predict negotiation outcomes. More information on this can be found in 7.2.3.

[2]This does assume the negotiation is still automated. However, the reader is free to bring an AI to a non-automated salary negotiation if they so desire, we would be very interested in their findings.

7. Conclusion and Future Work

7.1 Conclusion

In this thesis, we made the first steps in predicting the outcomes of automated negotiations. The main question we sought to answer was:

Research Question

How can we best predict the outcome of an automated negotiation?

We developed a framework for predicting automated negotiation outcomes as our solution to our research problem. This framework treats negotiations as time series and divides the problem into two subprocesses. The first of these is a time series forecasting process and the second an intersection one. Splitting the framework into these two distinct steps improves its modularity. We believe that this makes our solution comprehensible and suitable for further experimentation by other researchers, as individual parts can be easily adjusted.

When applying state-of-the-art TSF methodology to this framework, our solution can predict the utility outcomes of negotiations within 0.04 utility of the real outcome, on average. It also correctly predicts whether a negotiation ends in an agreement for 82% of negotiations, achieving an F1-score of 0.876.

The first subquestion we answered was:

Research Question 1.1

How can we best predict the utility time series of the agents participating in a negotiation?

We tested six different TSF methods in forecasting negotiation time series.

We showed that automated negotiation utility time series are difficult to predict with classic statistical models. Deep learning models were found to be the most effective method for TSF, due to their ability to learn from data across different negotiations. These networks outperformed the other five methods in forecasting accuracy, as measured with CRPS error. The best method, DeepAR, achieved an average CRPS error score of 0.081. Additionally, we showed the predictability of negotiation time series could be improved by introducing a behavior-based strategy component. In this thesis, this was modeled with the learning rate.

The second subquestion we answered was:

Research Question 1.2

How can we best predict the outcome of an automated negotiation based on time series forecasts of utility graphs?

We found that Monte Carlo methods are the most effective method for converting utility forecasts to outcome predictions. Notably, performance in time series forecasting did not translate directly into outcome predictions for all TSF methods. In general, deep learning methods were once again the most effective tool for accurately predicting the outcome of negotiations, both for utility outcome and agreement probability prediction. DeepAR showed the best performance across the board, especially for agreement negotiations. In contrast, it has the most room to improve when classifying breakoff negotiations, perhaps because these formed a smaller part of our data set. DeepAR achieved an average CRPS of 0.04 in utility outcome prediction, and classified 82% of negotiations correctly for both agreements and breakoffs. It achieved a recall of 0.82 and a precision of 0.94, resulting in an F1-score of 0.876.

We also found that the main drawback of deep learning networks is their inflexibility, as setup takes longer and the network had to be tailored to the specific negotiation setting at hand. When choosing between classical statistic models only, we found linear extrapolation to perform best. All methods improved in performance when given more negotiation data ("later cutoff

points") and when the opponent behaved more predictable ("learning rate").

In this thesis, we have made the first steps in investigating the possibilities of predicting the outcome of automated negotiations. In conclusion, automated negotiation outcomes are still challenging to predict, but we believe our approach is a valuable first step in their prediction.

7.2 Future Work

As this is the first research of this kind, we believe that the drawbacks described here can be mitigated with future research on (deep learning) prediction methods for automated negotiation. In this section, we will discuss some suggestions for future research directions that we believe may prove promising for the development of automated negotiation predictions.

7.2.1 Improving the predictions

The most straightforward direction for future research is improving time series forecasts and outcome predictions. This is best done by further developing the prediction network. Naturally, a network more equipped for the task at hand should lead to better predictions. Perhaps a more advanced type of temporal network can be tested for its ability to make accurate predictions. Another possibility is to take more parameters into account when making the prediction. In deep learning, this can make a big difference in prediction accuracy. We propose to integrate an opponent model into the network which can assist in the prediction. Additionally, the network can also be made to directly predict the outcome, as opposed to splitting the time series forecasting and intersecting in two separate steps. If this is done, an additional improvement could be to give the network an extra dimension by integrating the "true" negotiation outcomes into its input and output, instead of just utility scores.

7.2.2 Widening the scope

To reduce the number of variables that can influence the results, the scope of this research was kept to one negotiation setting with one type of negotiating agent. With the first steps made, a logical development of this research is to widen the scope and see how well it holds up. One option is to include more agents and strategy types in the research. Perhaps a general predictor can be developed that works for all or a wide subset of agents. Likewise, the research can be extended outside of bilateral, alternating-offer negotiations to include more negotiation settings.

A different direction that is of paramount importance is the network's applicability to non-simulated negotiations. Improving the network so that it can be tested in real use cases should be a next step in its development. Many algorithms and agents in automated negotiations are developed and tested in a simulated environment, and this research is no exception. If research should ever want to have an impact on stakeholders outside of just academia, the network must also be made to work in this context. Examples of relevant settings can include, but are not limited to salary negotiations, commerce, automated systems communication, or political negotiations.

7.2.3 Application

To give our work back to the scientific community and encourage further research building on this, we have developed an application for the Python-based negotiation simulator NegMAS [69]. It comes with the DeepAR models trained for this research, as well as the possibility to train a custom model, and all five benchmarks. Additionally, the application is fully modular, meaning the user can choose which parts of the research they would like to experiment with, and customize their networks with all options GluonTS offers for their DeepAR network. This means they can experiment with different model architectures, likelihood-, or loss functions. The intersection method is included as well, with the possibility to input any combination of forecasts and true concession curves.

A. Additional Information

A.1 Other TSF methods

A.1.1 ARIMA

ARIMA stands for AutoRegressive Integrated Moving Average. It is one of the most widely used approaches to Time Series Forecasting and aims to describe autocorrelations in data [32]. Autocorrelations in data are the similarities between observations made at different points in the time series. ARIMA uses these earlier value as predictors for future values in the series, so called "lagged values". An ARIMA model can not easily be used and ran "as is". It needs to be fine-tuned precisely in accordance with the type of data which it attempts to predict, and the data must be prepared for it to be used with ARIMA similarly as well. This is most commonly done using the Box-Jenkins method [15], a process in which the data is analyzed and differenced to be made usable for the specific time series. To my knowledge, ARIMA has not been used for the analysis of negotiations before like this, probably because of the tedious "tuning" which must be done for each time series.

An ARIMA model is tuned using 3 parameters, corresponding to the 3 different submodels contained within ARIMA [70], [71]. These are:

p: The lag order, or number of lag observations in the model. It corresponds to the AutoRegressive submodel (AR). This is the time series regressed with its previous value i.e. y(t-1), y(t-2) and so on. The formula for the AR prediction model is:

$$\hat{y}(t) = c + \Phi 1 \cdot y(t-1) + \Phi 2 \cdot y(t-2) + \cdots + \Phi p \cdot y(t-p) + \epsilon t$$

Where y(t) at any time is the value for the timeseries. c is a constant, and Φ the coefficient of the AR term y(t).

d: The degree of differencing, or the number of times the time series must be differenced before it becomes stationary. It corresponds to the Integrated submodel (I). In general, a time series of a *d*th order difference can be written as:

$$y'(t) = (1 - B)^d y(t)$$

Where y(t) is the regular time series and B the 'backshift operator'.

q: The order of moving average, or the size of the moving average window. It corresponds to the Moving Average submodel (MA), which uses a regression model on past forecasting errors.

$$\hat{y}(t) = c + \theta 1 \cdot \epsilon(t-1) + \theta 2 \cdot \epsilon(t-2) + \cdots + \theta q \cdot \epsilon(t-q)$$

This is similar to the AR term, again with y(t) the value for the timeseries, c a constant, and ϵ the coefficient of the MA term y(t).

Combining these, the full ARIMA model is as follows:

$$y'(t) = c + \Phi 1 \cdot y'(t-1) + \cdots + \Phi p \cdot y'(t-p) + \theta 1 \cdot \epsilon(t-1) + \cdots + \theta q \cdot \epsilon(t-q) + \epsilon t$$

These parameters must be carefully determined to create the optimal model for every time series. Each parameter has its own method by which its optimal value can be determined, which we will go over.

Firstly, the degree of differencing *d*. As mentioned, the time series must be made stationary. A stationary time series is one whose statistical properties do not depend on the time at which the series is observed [32] and thus does not have any trend or seasonality. To test whether a time series is stationary

and how often it must be differenced to be made stationary, a *Unit Root Test* can be used as well as analysing an *Autocorrelation* (ACF) plot [32], [33].

After the degree of differencing has been determined, the p and q values still must be judged. This is done by analyzing the ACF and Partial Autocorrelation (PACF) plots, respectively. An ACF is a bar chart of the coefficients of the correlation between a time series and lags of itself. A PACF plot is a bar chart of *partial* correlation coefficients between the series and lags of itself. For the sake of brevity, I will not go into detail about the exact analysis for the above-mentioned methods.

A.1.2 Exponential Smoothing

Exponential Smoothing is another statistical time series forecasting method. Exponential Smoothing and ARIMA together serve as the most commonly used approaches within time series forecasting, and can be seen as complementary approaches [32]. Where ARIMA looks at autocorrelations, Exponential Smoothing describes the trend and seasonality of data. It does this by using the weighted average of past observations, where the weights of older observations decay exponentially [72]. Thus, the more recent observations account for a heavier part of the prediction. The most simple version was introduced by Brown & Goodell [73], which was then improved by Holt et al. by making the method better equipped at handing data with a trend [74]. As our data will likely have trends in it, Holt's Exponential Smoothing is the preferred version to use. The formula for Holt's Exponential Smoothing is:

$$\hat{y}_{t+h|t} = l_t + hb_t$$

Here, l_t indicates an estimation of the level of the series at time t and b_t indicates the estimation of the trend of the time series at time t. The equations for l_t and b_t are:

$$l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$$

$$b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_t$$

Where $\alpha$is the smoothing parameter for the level, and β is the smoothing parameter for the trend $0 \leq \alpha, \beta \geq 1$. There are more versions of exponential smoothing, the most prevalent of which introduce smoothing and seasonality, but those are not relevant to the problem at hand, as our data does not require damping, nor is it seasonal. Since exponential smoothing is a relatively easy-to-implement method which does not require much tuning, it can serve as a baseline time series forecasting method.

A.1.3 Linear Extrapolation

Linear Extrapolation is a TSF method which extrapolates a line between the first and final observation in a time series.

A.1.4 Linear Regression

Linear Regression takes all observations in a time series and applies a regression over said points. The outcome is always a straight line, hence the name. The regression used in this research is SKLearn's Linear Regression module for python [75].

A.1.5 Naive

The Naive approach simply extends the last observation in the time series until the end of the range. This approach is often chosen for random walk models when no other prediction method is shown to work well.

A.2 Statistical Intersection Methods

A numerical way to calculate the intersection point probabilities is by utilizing statistical methods. The negotiations ran have discrete timesteps, upper-bounded by the maximum number of bids allowed to be made by an agent, which is defined in the Genius environment. So, there is always a maximum number of bids left to be made, equal to this upper bound minus the

number of bids already made. As established earlier, a prediction of the utility trajectories gives a probability density function (PDF) for every one of these (potential) future time steps for all trajectories. For each time step, these PDFs can be used to find the probability the points will intersect, as well as which point they are most likely to meet on. Here, it should be taken into account that earlier time steps influence the probabilities of later time steps as well, as the chance the trajectories intersect earlier decreases their chances of intersecting at a later timestep.

When looking at a single timestep. We have two probability density functions $X \sim N(\mu, \sigma^2)$ and $Y \sim N(\mu, \sigma^2)$, where μ represents the mean and σ^2 the variance. X is the utility given to us by the opponents' bids, and Y is the utility we receive for the bids made ourselves. We assume that X will start lower than Y, and will trend upwards by the opponent conceding, and Y will trend downwards by us making concessions. So, $Y(\mu) > X(\mu)$. The trajectories meet when they cross each other, and thus X > Y. In other words, the probability that the two utilities intersect is the chance that a random sample from X is larger than a random sample from Y. Given that this is the case, a new probability density function must be made by combining the two other functions which gives the most likely point of overlap. Thus, we are interested in the joint probability function under the condition that X > Y. If we let Z be the joint probability function, then $f_{Z|X>Y}(t)$.

Having found the individual timestep distributions, the next step is adding them. Calculating this joint probability function for each timestep gives a multitude of distributions over the utility space [0,1]. Combining these gives the mixture distribution $f(x) = \sum_{i=1}^{n} w_i p_i(x)$ where $p_1(x), \ldots, p_n(x)$ corresponds to the probability density functions for all timesteps and $w_1(x)$, $\ldots, w_n(x)$ the weights belonging to the functions. These weights are computed relative to the chance a negotiation ends at the given timestep. The chance a negotiation ends at t is found by multiplying the chance the probabilities overlap for timestep t, $(P(X_t > Y_t))$ by the chance the probabilities do not overlap at an earlier timestep. $j_t = \prod_{i=1}^{t-1}(1 - p(i)) \times p(t)$ where p(t) is the probability a negotiation ends at timestep t. Conveniently, the complement of the sum of these probabilities gives the chance the negotiation does

not end at any timestep, and thus no deal is made, $P(nodeal) = 1 - \sum_{i=1}^{n} j_i$. The weights are then computed by $w_t(x) = \frac{j_t}{\sum_{i=1}^{n} j_i}$, and can be used to calculate the final mixture distribution, which gives a probability density function describing the expected utility for our agent.

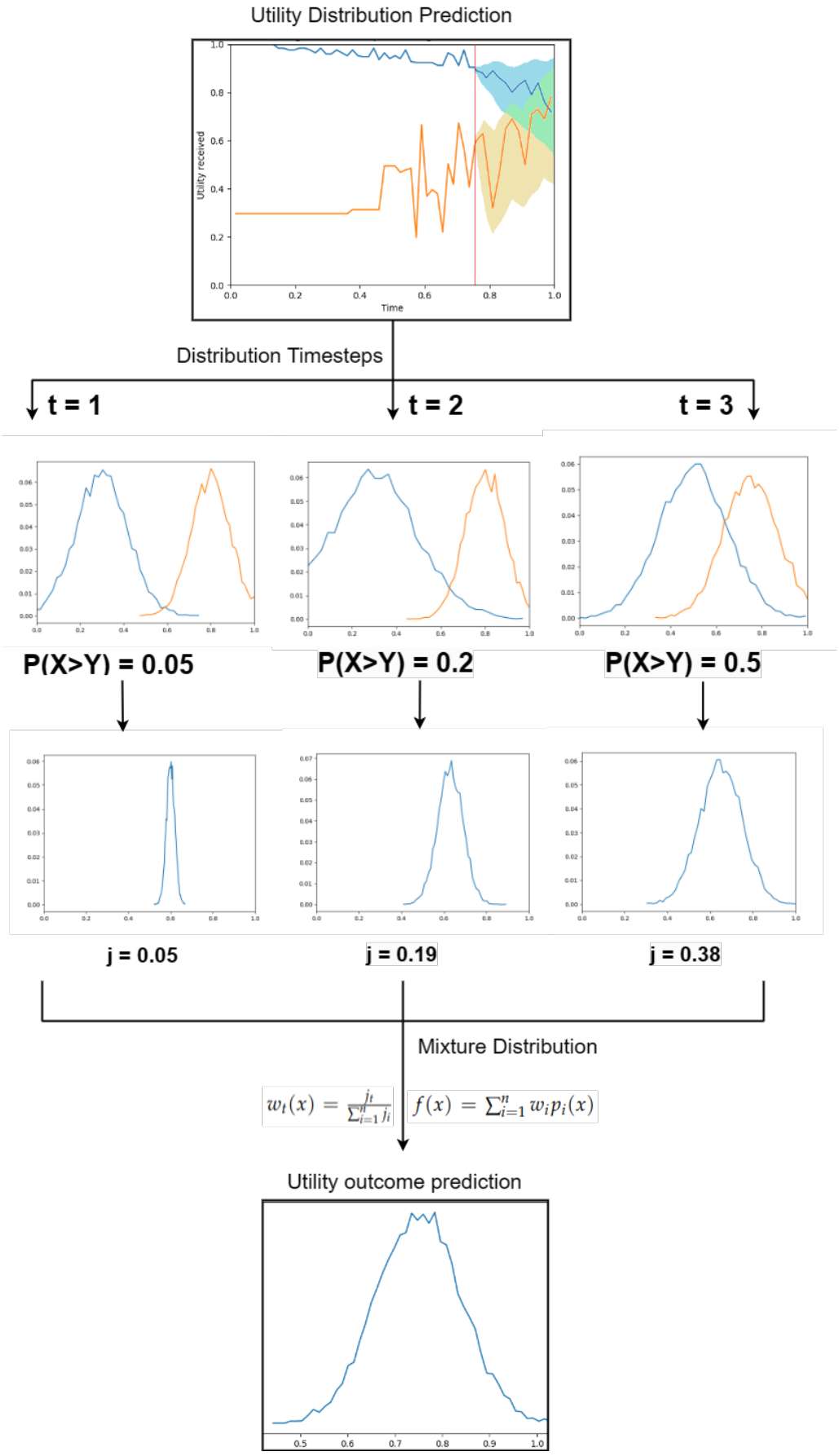


Figure A.1: Statistical process of generating a probability distribution outcome from utility graph distributions

A.3 Genius

The application used in this research to run and analyze negotiations is Genius [10]. Genius is the most commonly used automated negotiation simulator, sporting a robust community of researchers working on the development of automated negotiations. Most other research referenced here regarding automated negotiation is developed, ran and tested at least partially in the program. In Genius, the outcome space of a negotiation and the associated utilities are represented as seen in figure A.2

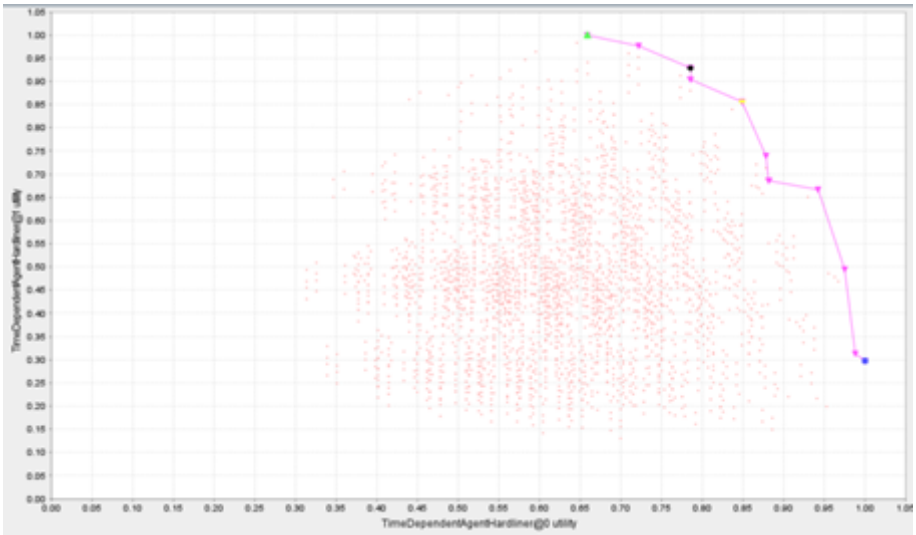


Figure A.2: Outcome space as shown in the genius environment, empty

In this image, red points on the plane indicate all possible outcomes of the negotiation, the outcome space. The blue and green points show the starting points for both agents, being the outcomes for which they receive maximum utility. Finally, the purple line indicates the Pareto frontier, which contains all Pareto optional outcomes for the negotiation. Ideally, a negotiation should result in an outcome which is **Pareto optimal**. An outcome is Pareto optimal when there exists no outcome which improves the utility for one agent while keeping the utility for the other at least the same (e.g. this outcome would be flatly better than the other one, as no agent has to compromise anything).

During a negotiation, Genius will also track the offers made by both agents, which will show how much the agents concede. An example of a full nego-

tiation can be found in A.3:

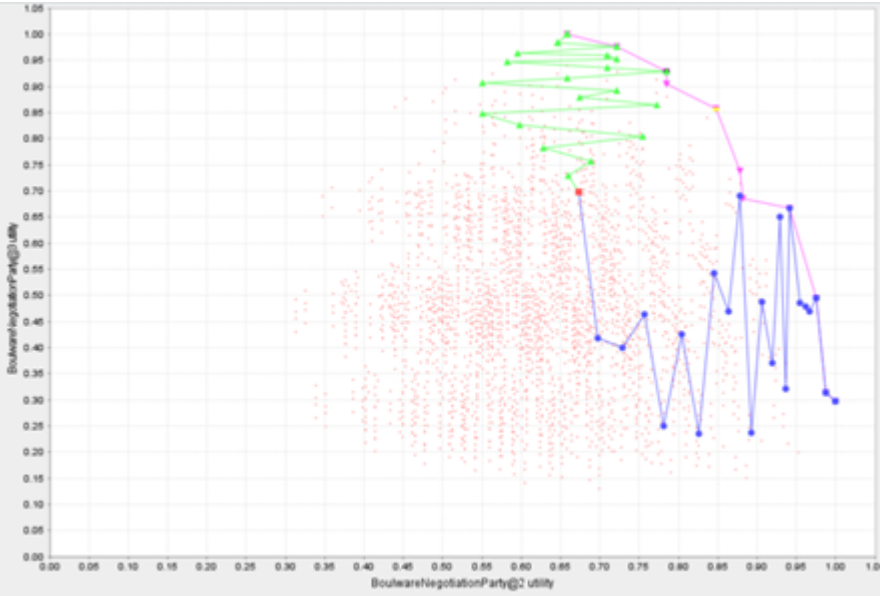
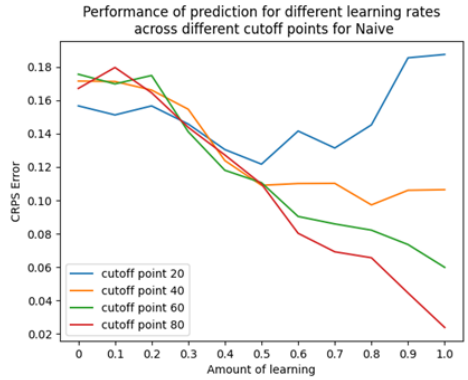
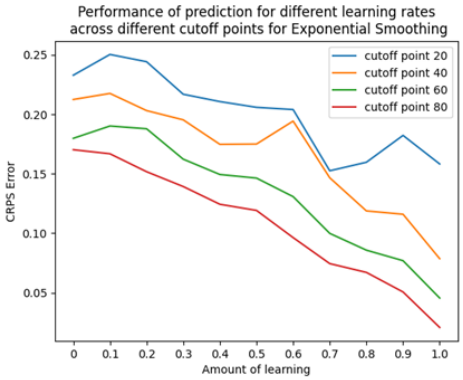


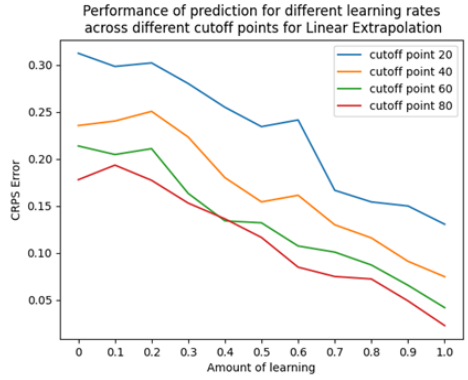
Figure A.3: Outcome space as shown in the genius environment after a negotiation

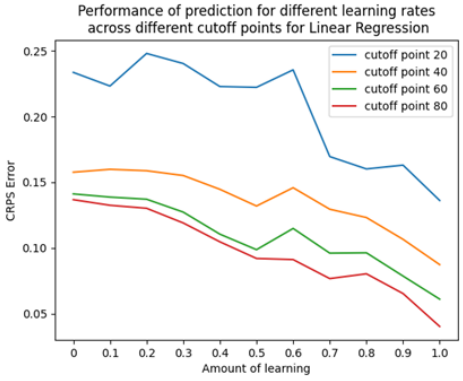
B. Additional Results

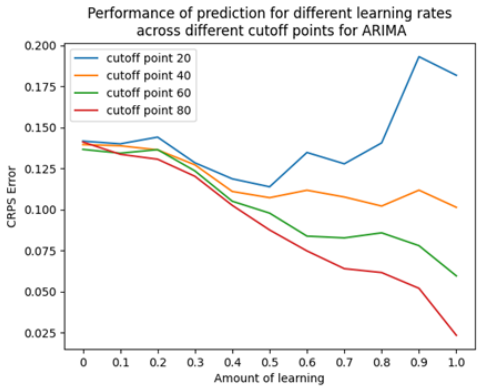
B.1 Results Experiment 1

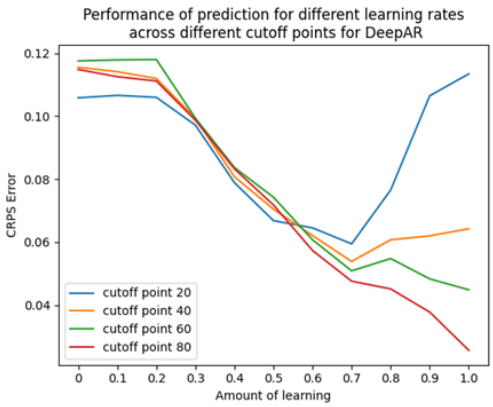












B.2 Results Experiment 2

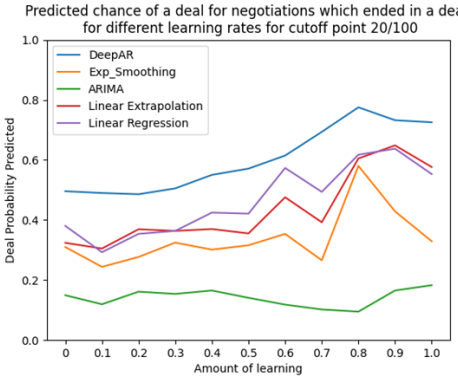


Figure B.1: Proportion of deals predicted by different methods for negotiations which ended in a deal, for cutoff point 20.

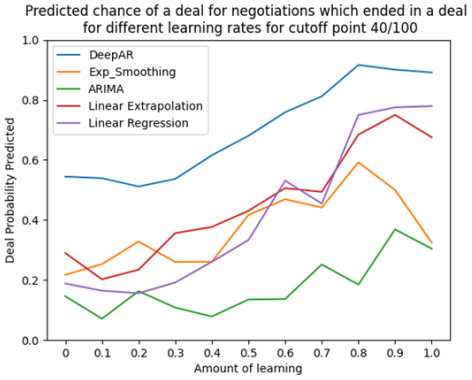


Figure B.2: Proportion of deals predicted by different methods for negotiations which ended in a deal, for cutoff point 40.

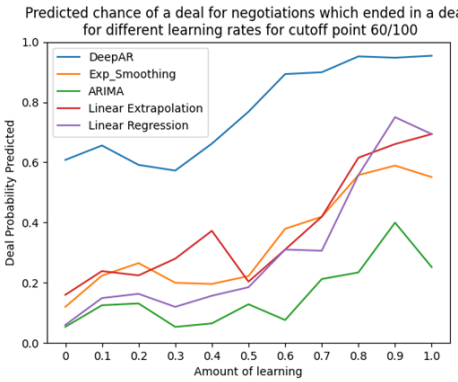


Figure B.3: Proportion of deals predicted by different methods for negotiations which ended in a deal, for cutoff point 60.

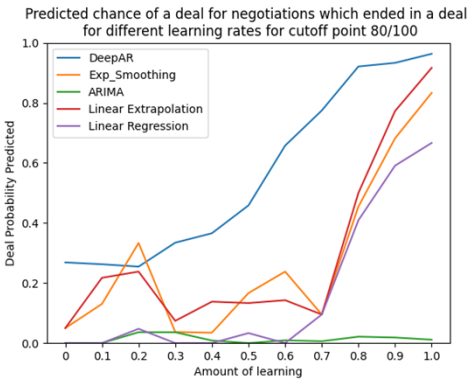


Figure B.4: Proportion of deals predicted by different methods for negotiations which ended in a deal, for cutoff point 80.

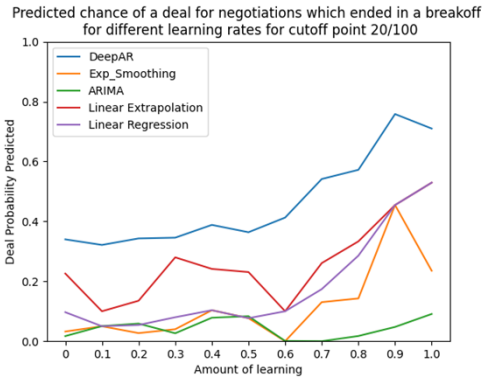


Figure B.5: Proportion of deals predicted by different methods for negotiations which ended in a breakoff, for cutoff point 20.

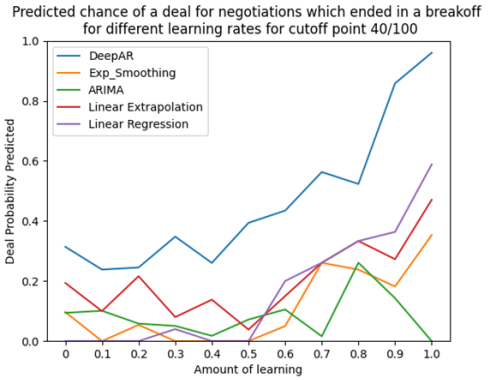


Figure B.6: Proportion of deals predicted by different methods for negotiations which ended in a breakoff, for cutoff point 40.

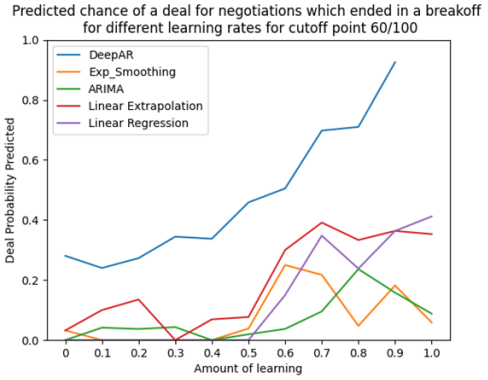


Figure B.7: Proportion of deals predicted by different methods for negotiations which ended in a breakoff, for cutoff point 60.

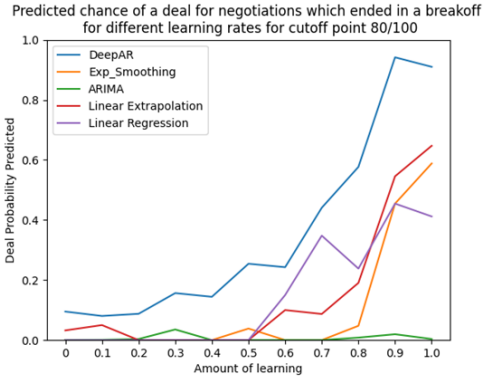
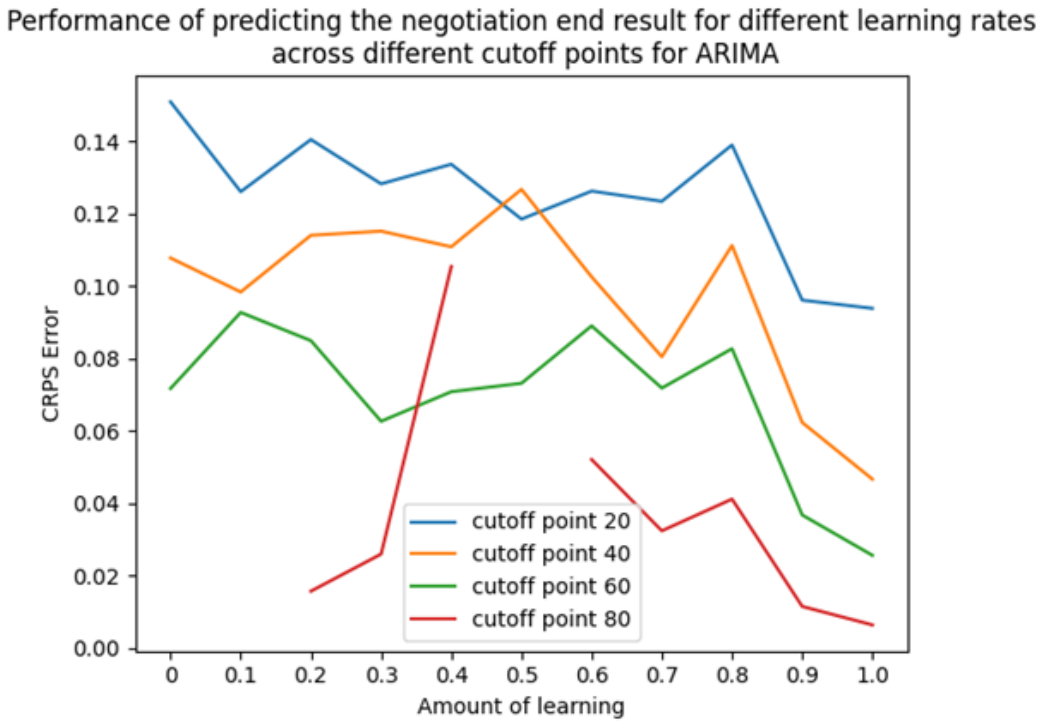
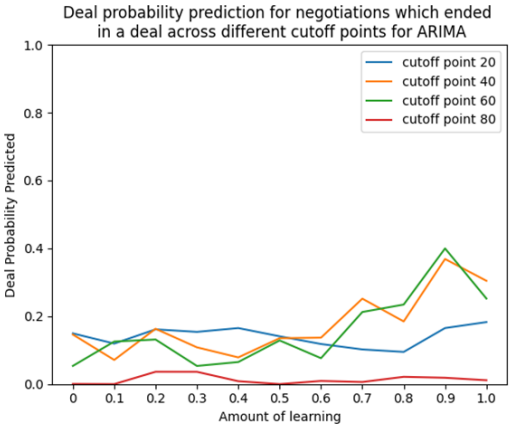
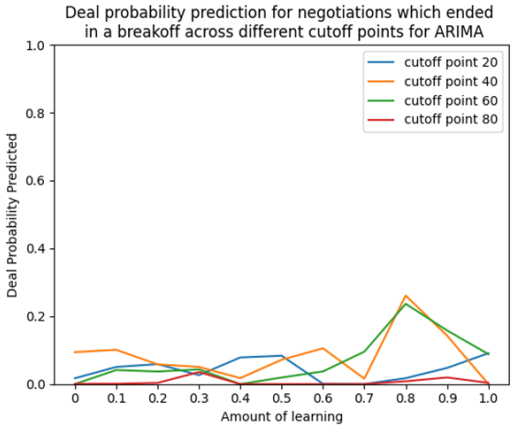


Figure B.8: Proportion of deals predicted by different methods for negotiations which ended in a breakoff, for cutoff point 80.

B.2.1 ARIMA

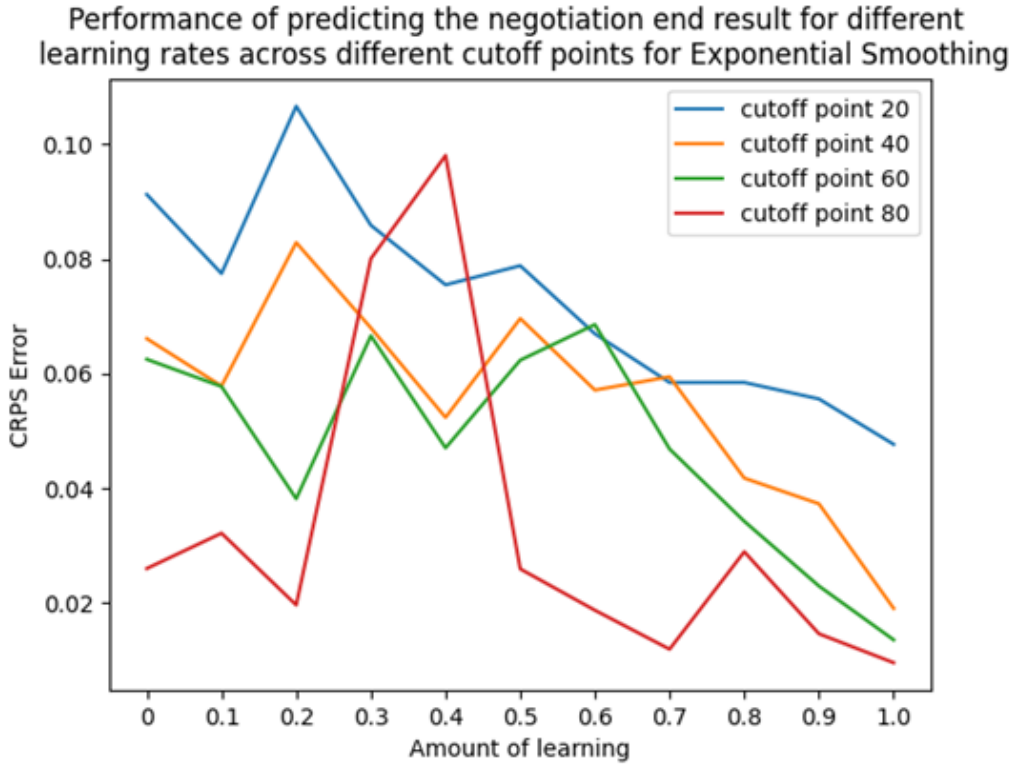


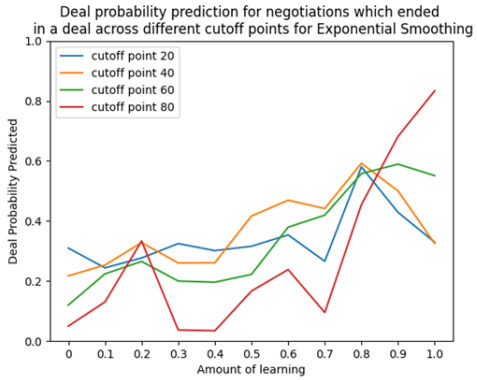


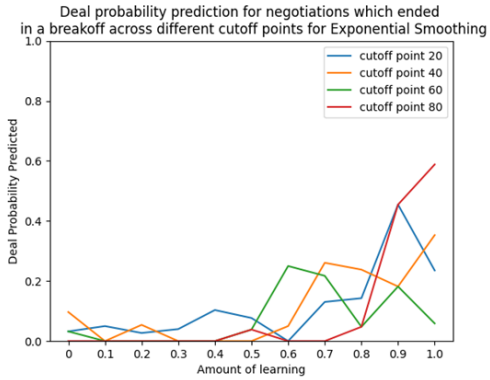


Here, it becomes clear that ARIMA, while scoring decently at TSF, nearly always predicts that a negotiation ends in a breakoff. The gaps in the prediction graph are present because ARIMA did not predict a deal for any negotiation with those settings.

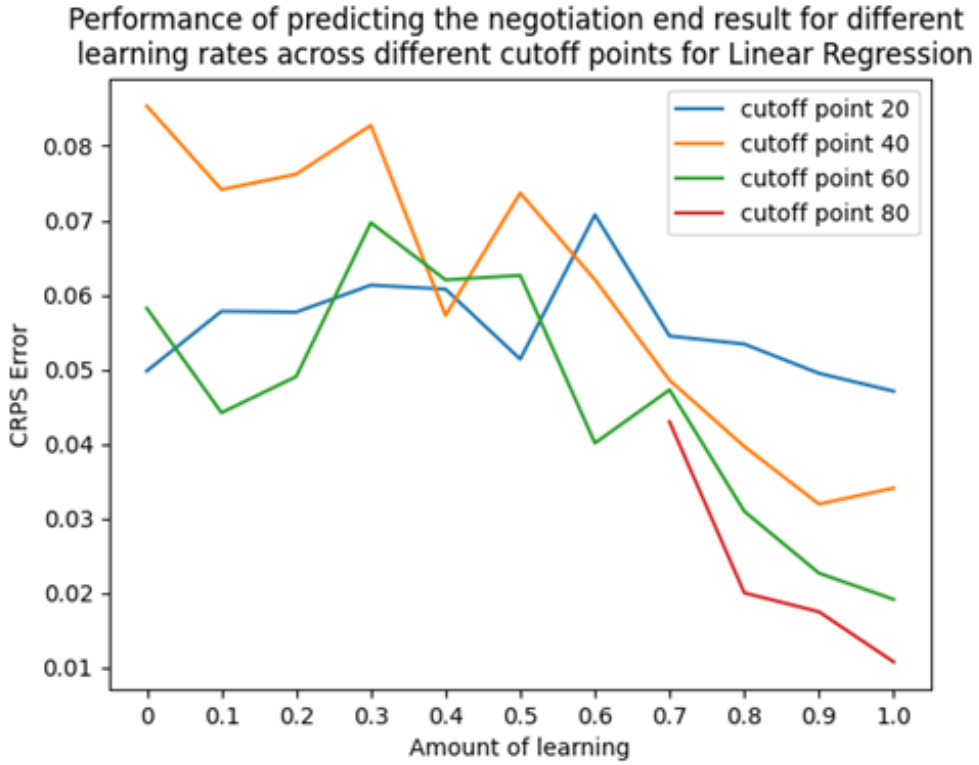
B.2.2 Exponential Smoothing

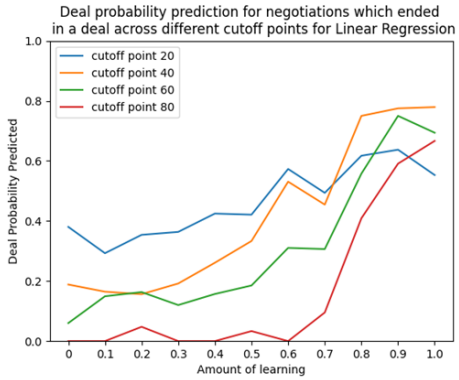


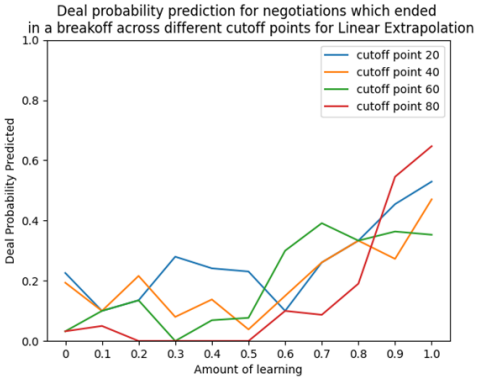




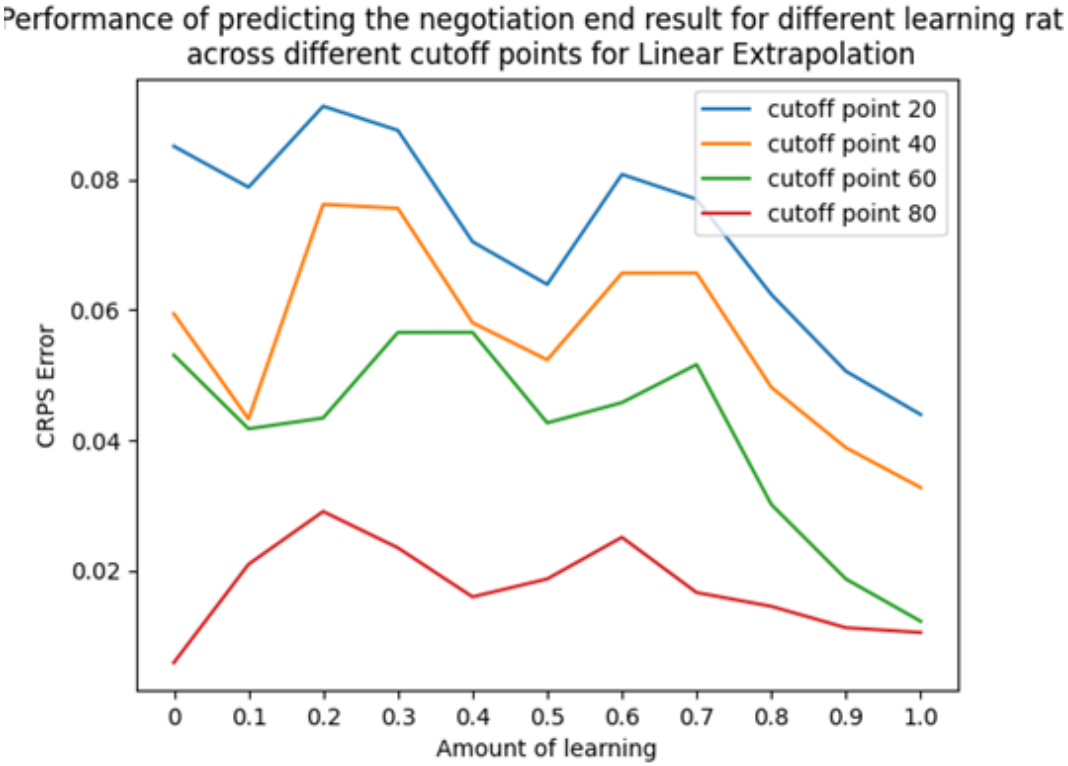
B.2.3 Linear Regression

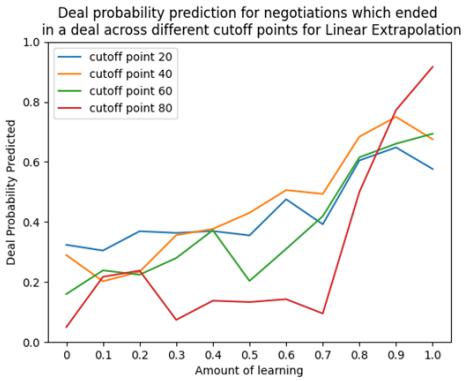


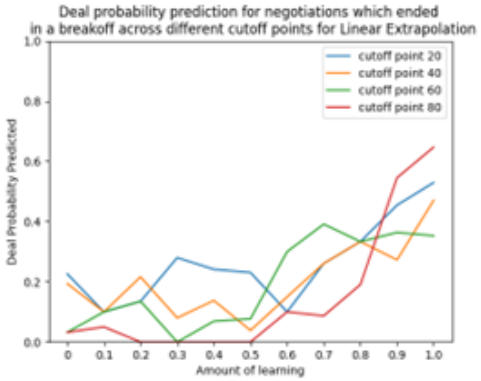




B.2.4 Linear Extrapolation

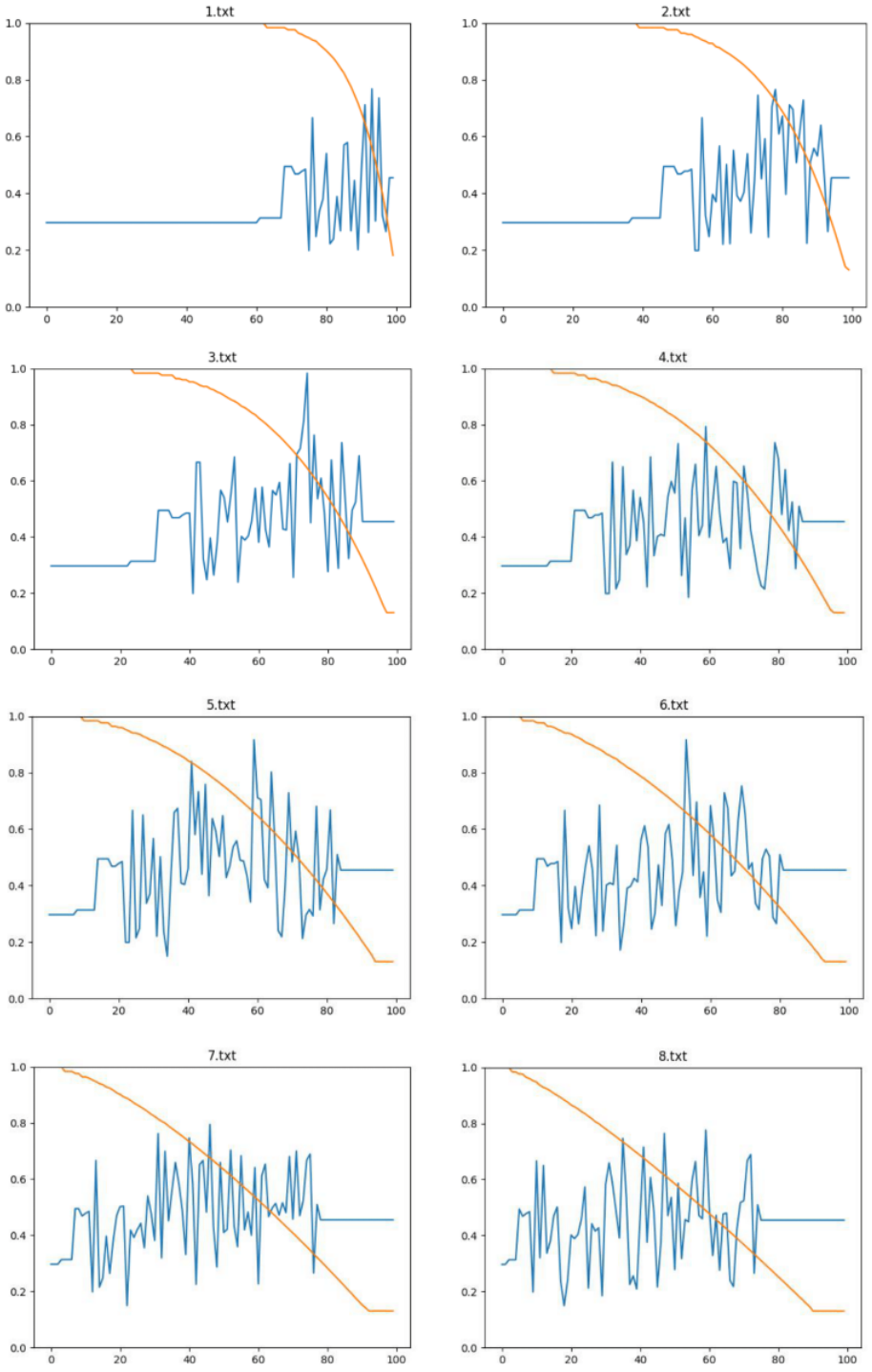


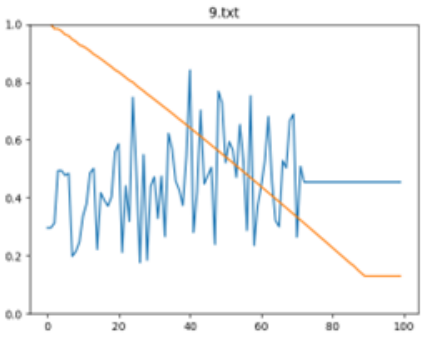


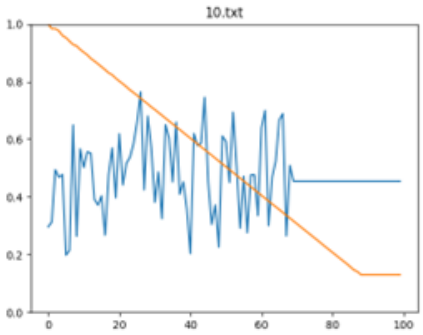


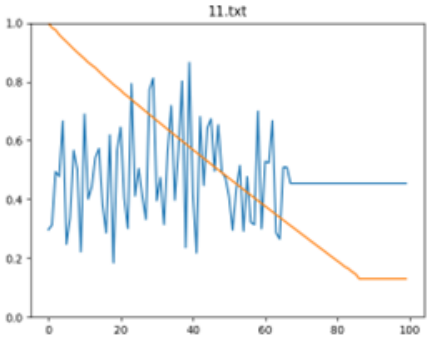
B.3 Parameter Effect

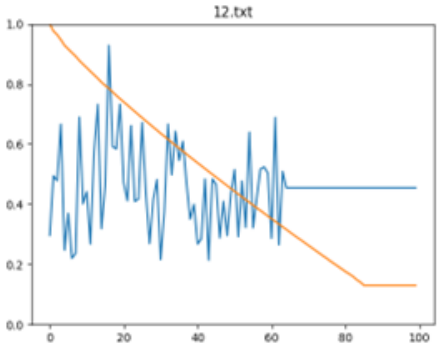
B.3.1 Concession rate

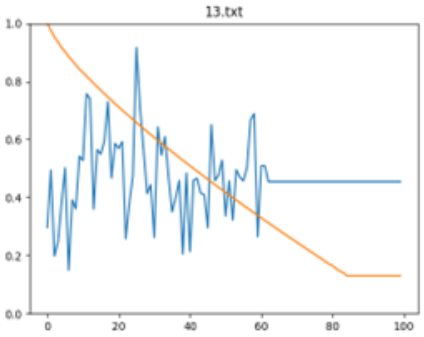


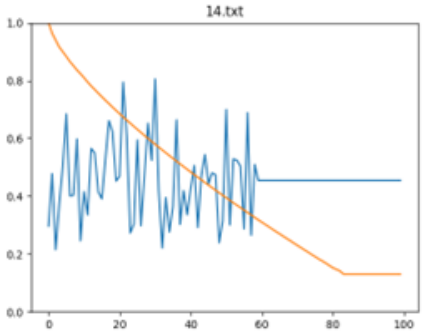

9.txt

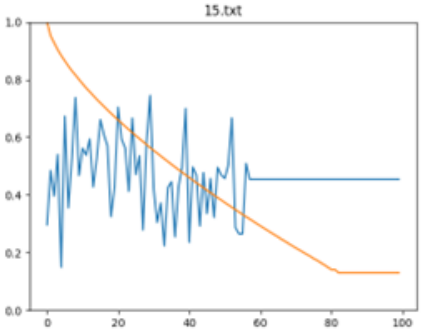

10.txt

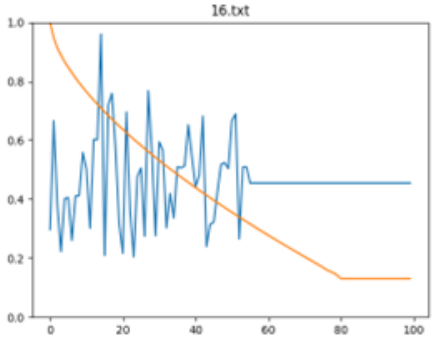

11.txt

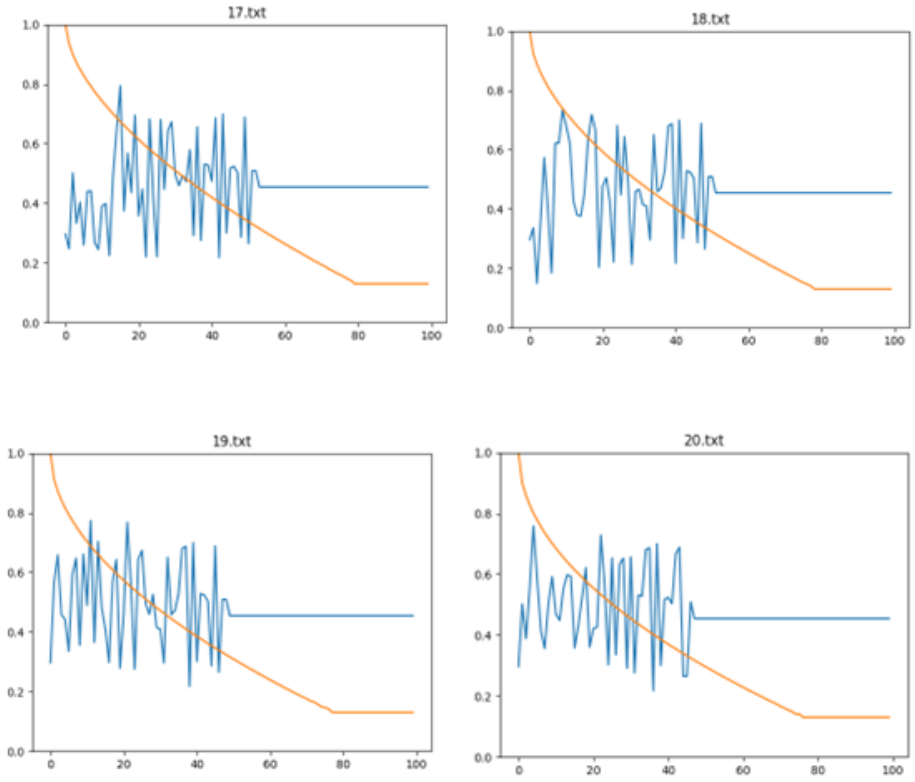

12.txt


13.txt


14.txt


15.txt


16.txt



The figures show the time series of two TDAs for 20 different concession rates, between 0.1 and 2.0. Parameters m and l are both 0.

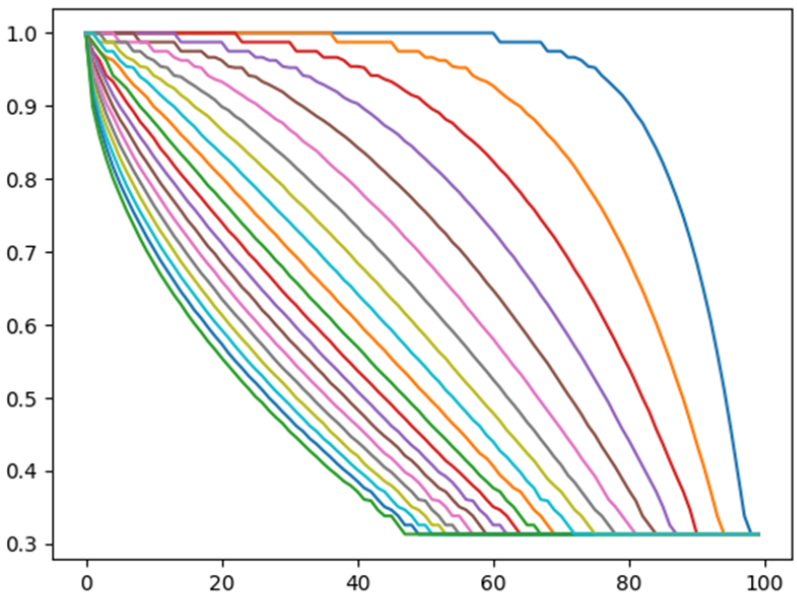
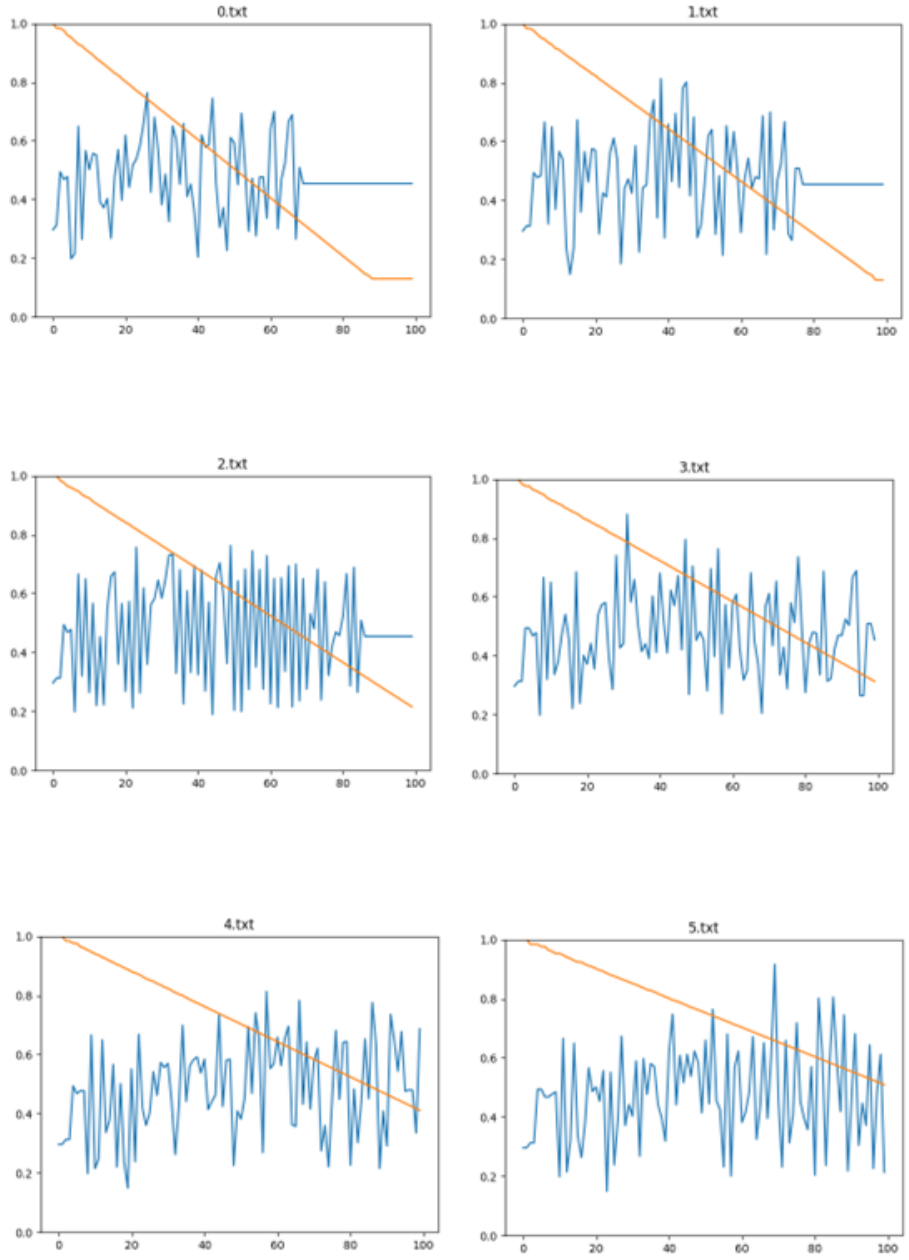


Figure B.9: Effect concession rates 0.1 - 2.0 have on the concession curve.

B.3.2 Target Utility



The figures show the time series of two TDAs for 10 different target utilities, between 0 and 0.9. Parameter e is 1, and l is 0. See figure B.9 for the transition from a 0 to a 0.9 target utility.

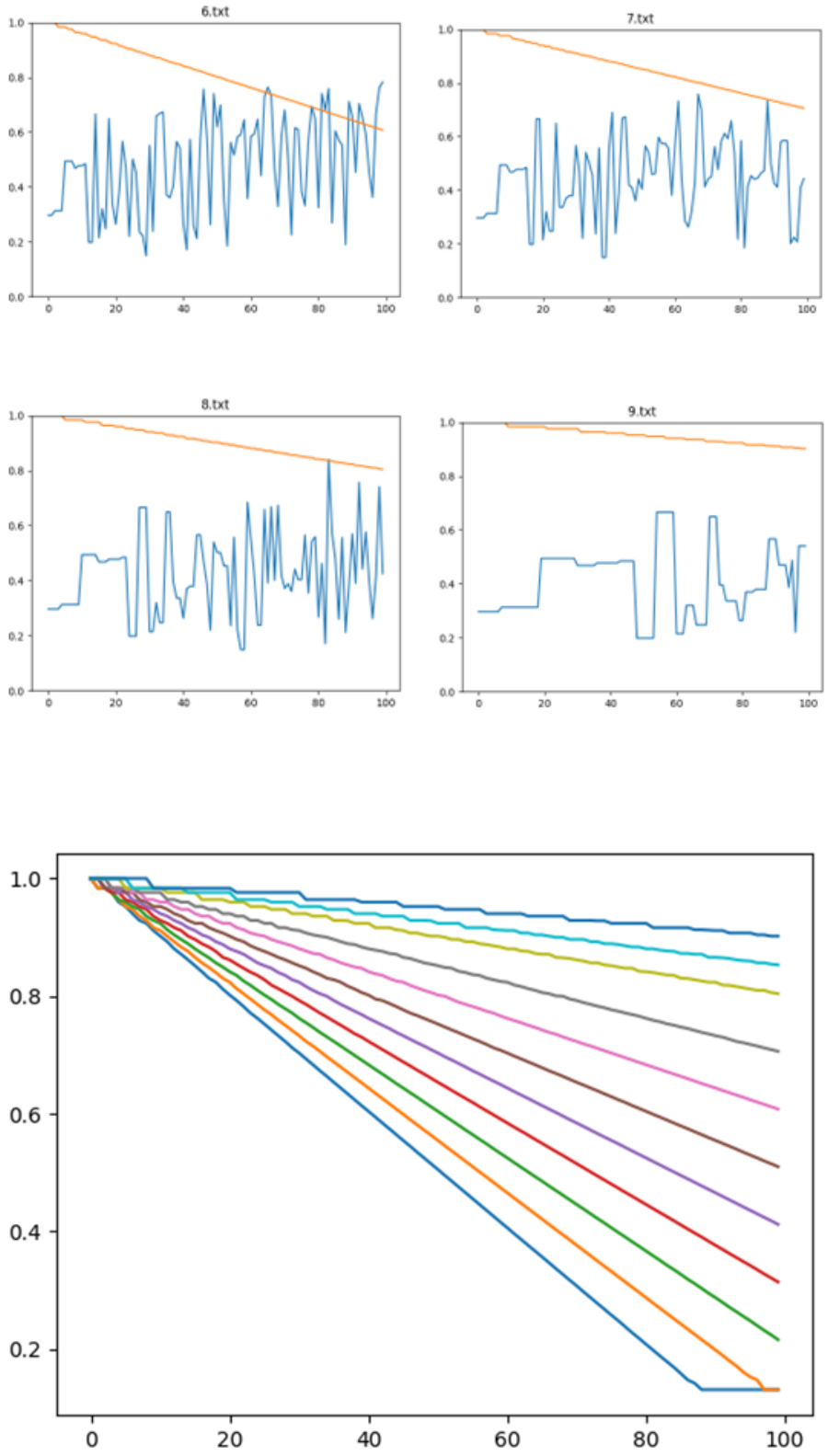
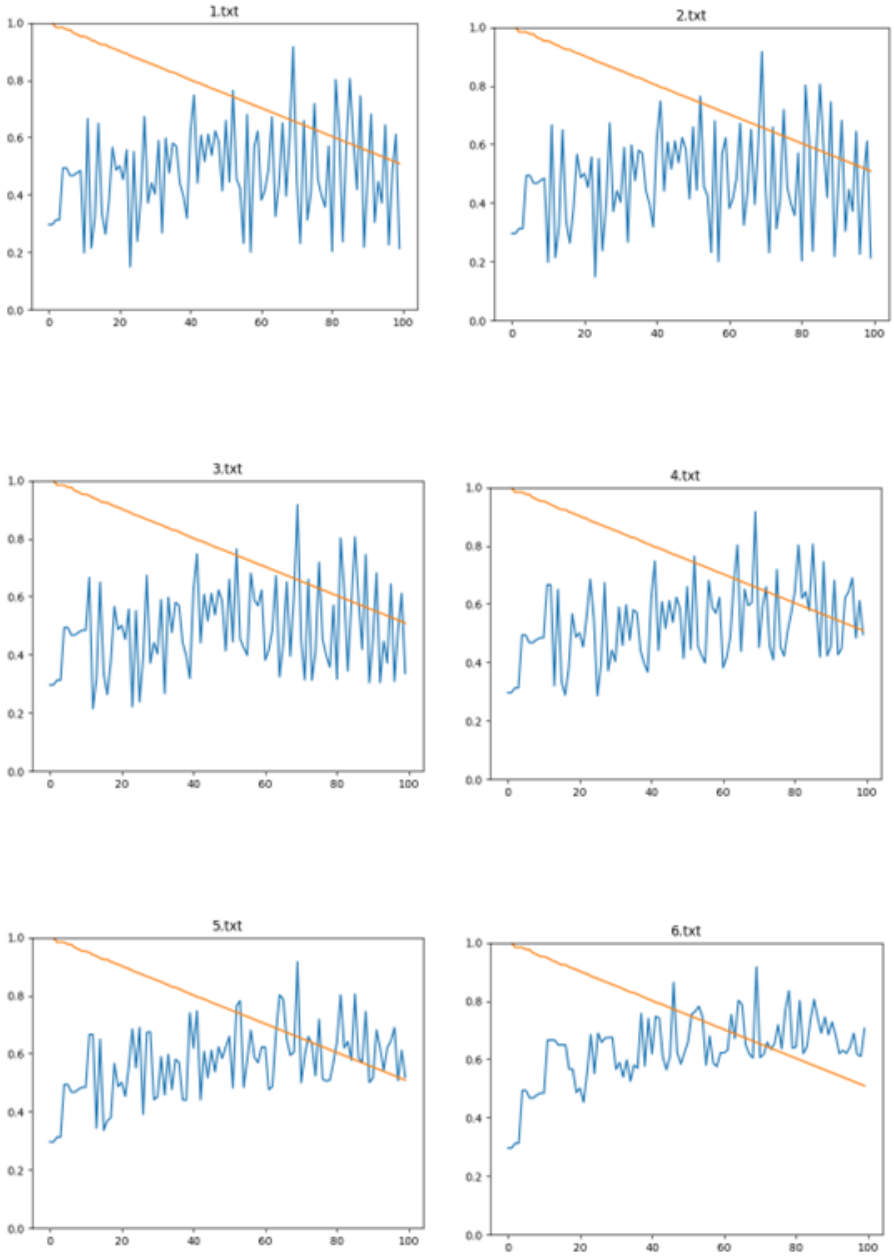
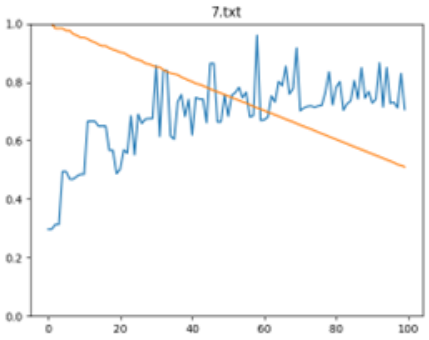


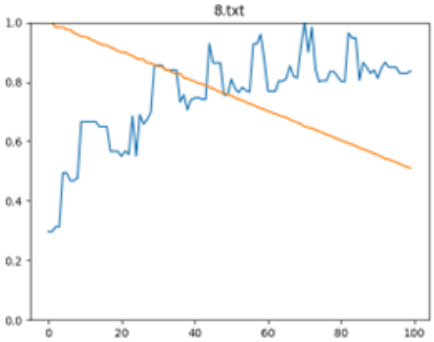
Figure B.10: Effect target utilities 0 - 0.9 have on the concession curve.

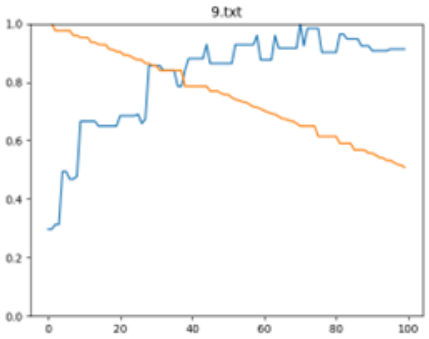
B.3.3 Learning Rate

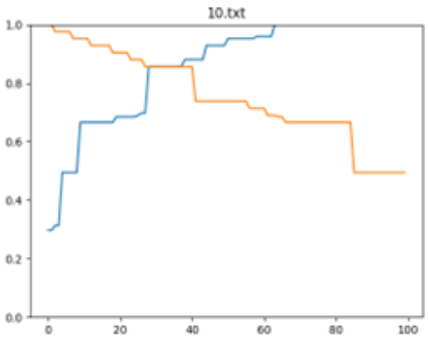


The figures show the time series of two TDAs for 10 different learning rates, between 0.1 and 1.0. Parameter e is 1, and m is 0.


7.txt
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


8.txt
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


9.txt
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


10.txt
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

Bibliography

[1] R. Benjamin, "The natural history of negotiation and mediation: The evolution of negotiative behaviors, rituals, and approaches," *Mediate–Everything Mediation*, 2012.

[2] R. I. M. Dunbar, *Grooming, gossip, and the evolution of language*. Harvard University Press, 1996.

[3] D. A. Lax and J. K. Sebenius, *The Manager as Negotiator*. NY: Free Press, 1986.

[4] P. Faratin, C. Sierra, and N. R. Jennings, "Negotiation decision functions for autonomous agents," *Robotics and Autonomous Systems*, vol. 24, no. 3-4, pp. 159–182, 1998, ISSN: 09218890. DOI: 10.1016/S0921-8890(98)00029-3.

[5] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated negotiation: Prospects, methods and challenges," *International Journal of Group Decision and Negotiation*, vol. 10, no. 2, pp. 199–215, 2001.

[6] M. M. Delaney, A. Foroughi, and W. C. Perkins, "An empirical study of the efficacy of a computerized negotiation support system (nss)," *Decision Support Systems*, vol. 20, no. 3, pp. 185–197, 1997.

[7] S. Chakraborty, T. Baarslag, and M. Kaisers, "Automated peer-to-peer negotiation for energy contract settlements in residential cooperatives," *Applied Energy*, vol. 259, p. 114 173, 2020.

[8] C. Eran, M. O. Keskin, F. Cantürk, and R. Aydoğan, "A decentralized token-based negotiation approach for multi-agent path finding," in *European Conference on Multi-Agent Systems*, Springer, 2021, pp. 264–280.

[9] B. J. Clement and A. C. Barrett, "Continual coordination through shared activities," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003, pp. 57–64.

[10] R. Lin, S. Kraus, T. Baarslag, D. Tykhonov, K. Hindriks, and C. M. Jonker, "Genius: An integrated environment for supporting the design of generic automated negotiators," *Computational Intelligence*, vol. 30, no. 1, pp. 48–70, 2014.

[11] C. R. Williams, V. Robu, E. H. Gerding, and N. R. Jennings, "Using gaussian processes to optimise concession in complex negotiations against unknown opponents," 2011. [Online]. Available: https://eprints.soton.ac.uk/271965/.

[12] S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss, "Transfer learning for bilateral multi-issue negotiation," *Belgian/Netherlands Artificial Intelligence Conference*, 2012, ISSN: 15687805.

[13] K. Hindriks, C. M. Jonker, and D. Tykhonov, "Let's dans! An analytic framework of negotiation dynamics and strategies," *Web Intelligence and Agent Systems*, vol. 9, no. 4, pp. 319–335, 2011, ISSN: 15701263. DOI: 10.3233/WIA-2011-0221.

[14] V. Dhar, "Data science and prediction," *Communications of the ACM*, vol. 56, no. 12, pp. 64–73, 2013.

[15] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 1970.

[16] G. Yesevi, M. O. Keskin, A. Doğru, and R. Aydoğan, "Time Series Predictive Models for Opponent Behavior Modeling in Bilateral Negotiations," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 13753 LNAI, pp. 381–398, 2023, ISSN: 16113349. DOI: 10.1007/978-3-031-21203-1_23. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-21203-1_23.

[17] C. Hou, "Predicting agents tactics in automated negotiation," *Proceedings - IEEE/WIC/ACM International Conference on Intelligent Agent Technology. IAT 2004*, pp. 127–133, 2004. DOI: 10.1109/iat.2004.1342934.

[18] A. Sengupta, Y. Mohammad, and S. Nakadai, "An autonomous negotiating agent framework with reinforcement learning based strategies and adaptive strategy switching mechanism," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 2, pp. 1151–1160, 2021, ISSN: 15582914. DOI: 10.5555/3463952.3464087. arXiv: 2102.03588.

[19] M. Li, P. K. Murukannaiah, and C. M. Jonker, "A data-driven method for recognizing automated negotiation strategies," no. 1, 2021. arXiv: 2107.01496. [Online]. Available: https://arxiv.org/abs/2107.01496v2.

[20] L. Ilany and Y. Gal, "Algorithm selection in bilateral negotiation," *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 4, pp. 697–723, 2016, ISSN: 15737454. DOI: 10.1007/s10458-015-9302-8.

[21] D. Van Poucke and M. Buelens, "Predicting the outcome of a two-party price negotiation: Contribution of reservation price, aspiration price and opening offer," *Journal of Economic Psychology*, vol. 23, no. 1, pp. 67–76, 2002, ISSN: 01674870. DOI: 10.1016/S0167-4870(01)00068-X.

[22] Max H. Bazerman, Jared R. Curhan, Don A. Moore, and Kathleen L. Valley, "Negotiation," *Annual Review of Psychology*, vol. 51, pp. 279–314, 2000.

[23] S. Sharma, W. P. Bottom, and H. A. Elfenbein, "On the role of personality, cognitive ability, and emotional intelligence in predicting negotiation outcomes: A meta-analysis," *journals.sagepub.com*, vol. 3, no. 4, pp. 293–336, 2013. DOI: 10.1177/2041386613505857. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/2041386613505857?casa_token=B5UhB9y8tgwAAAAA:EfZX3Muz-

PA-wr4s9uMYVKMm5by7yNos42kW6gXd7XnxbzGhBbzuMt17BiBScsEku5qnpt8clBo.

[24] H. Kristensen and T. Gärling, "The effects of anchor points and reference points on negotiation process and outcome," *Organizational behavior and human decision processes*, vol. 71, no. 1, pp. 85–94, 1997.

[25] A. D. Galinsky, W. W. Maddux, D. Gilin, and J. B. White, "Why it pays to get inside the head of your opponent: The differential effects of perspective taking and empathy in negotiations," *Psychological science*, vol. 19, no. 4, pp. 378–384, 2008.

[26] D. C. Moosmayer, A. Y. L. Chong, M. J. Liu, and B. Schuppar, "A neural network approach to predicting price negotiation outcomes in business-to-business contexts," *Expert Systems with Applications*, vol. 40, no. 8, pp. 3028–3035, 2013, ISSN: 09574174. DOI: 10.1016/j.eswa.2012.12.018. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2012.12.018.

[27] J. Mell, G. M. Lucas, and J. Gratch, "Pandemic Panic: The Effect of Disaster-Related Stress on Negotiation Outcomes," *Proceedings of the 21st ACM International Conference on Intelligent Virtual Agents, IVA 2021*, pp. 148–155, 2021. DOI: 10.1145/3472306.3478353.

[28] K. Chawla, G. Lucas, J. May, and J. Gratch, "BERT in Negotiations: Early Prediction of Buyer-Seller Negotiation Outcomes," 2020. arXiv: 2004.02363. [Online]. Available: http://arxiv.org/abs/2004.02363.

[29] R. A. Carbonneau, G. E. Kersten, and R. M. Vahidov, "Pairwise issue modeling for negotiation counteroffer prediction using neural networks," *Decision Support Systems*, vol. 50, no. 2, pp. 449–459, 2011, ISSN: 01679236. DOI: 10.1016/j.dss.2010.11.002.

[30] J. Brzostowski and R. Kowalczyk, "Adaptive negotiation with on-line prediction of opponent behaviour in agent-based negotiations," *Proceedings - 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2006 Main Conference Proceedings), IAT'06*, pp. 263–269, 2006. DOI: 10.1109/IAT.2006.26.

[31] M. O. Keskin, U. Çakan, and R. Aydogan, "Solver agent: Towards emotional and opponent-aware agent for human-robot negotiation," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 3, pp. 1545–1547, 2021, ISSN: 15582914.

[32] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[33] R. Nau, *Statistical forecasting: notes on regression and time series analysis*, 2020. [Online]. Available: https://people.duke.edu/$%5Csim$rnau/411home.htm.

[34] F. X. Diebold and L. Kilian, "Measuring predictability: Theory and macroeconomic applications," *Journal of Applied Econometrics*, vol. 16, no. 6, pp. 657–669, 2001, ISSN: 08837252. DOI: 10.1002/jae.619.

[35] F. Pennekamp, A. C. Iles, J. Garland, *et al.*, "The intrinsic predictability of ecological time series and its potential to guide forecasting," *Ecological Monographs*, vol. 89, no. 2, pp. 1–17, 2019, ISSN: 15577015. DOI: 10.1002/ecm.1359.

[36] P. Xu, L. Yin, Z. Yue, and T. Zhou, "On predictability of time series," *Physica A: Statistical Mechanics and its Applications*, vol. 523, pp. 345–351, 2019, ISSN: 03784371. DOI: 10.1016/j.physa.2019.02.006. arXiv: 1806.03876. [Online]. Available: https://doi.org/10.1016/j.physa.2019.02.006.

[37] K. Pearson, "The problem of the random walk," *Nature*, vol. 72, no. 1865, pp. 294–294, 1905.

[38] E. F. Fama, "Random walks in stock market prices," *Financial analysts journal*, vol. 51, no. 1, pp. 75–80, 1995.

[39] J. Brownlee, *A Gentle Introduction to the Random Walk for Times Series Forecasting*, 2020. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-random-walk-times-series-forecasting-python/.

[40] A. Seth, "Granger causality," *Scholarpedia*, vol. 2, no. 7, p. 1667, 2007, revision #127333. DOI: 10.4249/scholarpedia.1667.

[41] S. Makridakis, E. Spiliotis, V. Assimakopoulos, A. A. Semenoglou, G. Mulder, and K. Nikolopoulos, "Statistical, machine learning and deep learning forecasting methods: Comparisons and ways forward," *Journal of the Operational Research Society*, vol. 74, no. 3, pp. 840–859, 2023, ISSN: 14769360. DOI: 10.1080/01605682.2022.2118629. [Online]. Available: https://doi.org/10.1080/01605682.2022.2118629.

[42] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[43] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.

[44] J. Gu, Z. Wang, J. Kuen, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[45] J. Zou, M. Huss, A. Abid, P. Mohammadi, A. Torkamani, and A. Telenti, "A primer on deep learning in genomics," *Nature genetics*, vol. 51, no. 1, pp. 12–18, 2019.

[46] S. Amari, "A theory of adaptive pattern classifiers," *IEEE Transactions on Electronic Computers*, no. 3, pp. 299–307, 1967.

[47] N. McCullum, *Deep Learning Neural Networks Explained in Plain English*, 2020. [Online]. Available: https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/#:$%5Csim$:text=Neurons%20in%20deep%20learning%20models,layer%20of%20the%20neural%20net..

[48] L. Shukla, *Fundamentals of Neural Networks*, 2023. [Online]. Available: https://wandb.ai/wandb_fc/articles/reports/Fundamentals-of-Neural-Networks--Vmlldzo1NDQ0Mzk1.

[49] B. Lim and S. Zohren, "Time-series forecasting with deep learning: A survey," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2194, 2021, ISSN: 1364503X. DOI: 10.1098/rsta.2020.0209. arXiv: 2004.13408.

[50] J. Manyika, M. Chui, B. Brown, *et al.*, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[51] P. Taylor, "Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025," *Statista*, 2022. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/.

[52] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, ISSN: 0899-7667. DOI: 10.1162/NECO_A_01199. [Online]. Available: https://direct-mit-edu.proxy.library.uu.nl/neco/article/31/7/1235/8500/A-Review-of-Recurrent-Neural-Networks-LSTM-Cells.

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[55] J. Schmidhuber, *The 2010s: Our Decade of Deep Learning / Outlook on the 2020s*, 2020. [Online]. Available: https://people.idsia.ch/$%5Csim$juergen/2010s-our-decade-of-deep-learning.html.

[56] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, *et al.*, "GluonTS: Probabilistic Time Series Modeling in Python," *arXiv preprint arXiv:1906.05264*, 2019.

[57] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020, ISSN: 01692070. DOI: 10.1016/j.ijforecast.2019.07.001. arXiv: 1704.04110. [Online]. Available: https://doi.org/10.1016/j.ijforecast.2019.07.001.

[58] A. Lambert, D. Gruyer, and G. S. Pierre, "A Fast Monte Carlo algorithm for collision probability estimation," *2008 10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008*, vol. 1, no. December, pp. 406–411, 2008. DOI: 10.1109/ICARCV.2008.4795553.

[59] V. Vahedi and M. Surendra, "A monte carlo collision model for the particle-in-cell method: Applications to argon and oxygen discharges," *Computer Physics Communications*, vol. 87, no. 1, pp. 179–198, 1995, Particle Simulation Methods, ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(94)00171-W. [Online]. Available: https://www.sciencedirect.com/science/article/pii/001046559400171W.

[60] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev, "Why the monte carlo method is so important today," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, no. 6, pp. 386–392, 2014.

[61] C. J. Spoerer, P. McClure, and N. Kriegeskorte, "Recurrent convolutional neural networks: A better model of biological object recognition," *Frontiers in Psychology*, vol. 8, no. SEP, pp. 1–14, 2017, ISSN: 16641078. DOI: 10.3389/fpsyg.2017.01551.

[62] M. Taboga, *"Log-likelihood", Lectures on probability theory and mathematical statistics.* 2021. [Online]. Available: https://www.statlect.com/glossary/log-likelihood.

[63] A. Amini, A. Amini, S. Karaman, and D. Rus, "Spatial uncertainty sampling for end-to-end control," May 2018.

[64] J. Matheson and R. Winkler, "Scoring rules for continuous probability distributions," *Management Science*, vol. 22, no. 10, pp. 1087–1096, 1976.

[65] L. Tsaprounis, *Metrics for Distributional Forecasts*, 2023. [Online]. Available: https://medium.com/trusted-data-science-haleon/metrics-for-distributional-forecasts-60e156c60177.

[66] T. Gneiting and M. Katzfuss, "Probabilistic forecasting," *Annual Review of Statistics and Its Application*, vol. 1, pp. 125–151, 2014, ISSN: 2326831X. DOI: 10.1146/annurev-statistics-062713-085831.

[67] I. Faran, *CRPS: A scoring function for bayesian machine learning models*, 2023. [Online]. Available: https://towardsdatascience.com/crps-a-scoring-function-for-bayesian-machine-learning-models-dd55a7a337a8.

[68] K. Hindriks, C. M. Jonker, and D. Tykhonov, "Analysis of negotiation dynamics," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4676 LNAI, pp. 27–35, 2007, ISSN: 16113349. DOI: 10.1007/978-3-540-75119-9_3.

[69] Y. Mohammed, *NegMAS*, 2018. [Online]. Available: https://negmas.readthedocs.io/en/latest/#.

[70] N. Bora, *Understanding ARIMA Models for Machine Learning*, 2021. [Online]. Available: https://www.capitalone.com/tech/machine-learning/understanding-arima-models/.

[71] Shweta, *Introduction to Time Series Forecasting — Part 2 (ARIMA Models)*, 2021. [Online]. Available: https://towardsdatascience.com/introduction-to-time-series-forecasting-part-2-arima-models-9f47bf0f476b.

[72] E. S. Gardner Jr, "Exponential smoothing: The state of the art—part ii," *International journal of forecasting*, vol. 22, no. 4, pp. 637–666, 2006.

[73] R. G. Brown, "Exponential smoothing for predicting demand. cambridge, mass., arthur d. little," *Book Exponential Smoothing for Predicting Demand*, 1956.

[74] C. C. Holt, "Forecasting trends and seasonals by exponentially weighted moving averages," *ONR Memorandum*, vol. 52, no. 52, pp. 5–10, 1957.

[75] *Sklearn Linear Regression*, 2023. [Online]. Available: https://www.simplilearn.com/tutorials/scikit-learn-tutorial/sklearn-linear-regression-with-examples (visited on 02/22/2024).