A simplification of the Mixture of Theories *

Felix Scheepens 545351

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis (FEB63007)

Supervisor: A. Baillon, Second Assessor: M. Zhelonkin

Date final version: 3 July 2022

Abstract

The Mixture of Theories (MoT) model is a model that is used for predicting human decision making. It is revolutionary because it performs better than other models, while it is a simplification of these models. However, it does still contain a very large amount of parameters. The goal of this paper is to see whether the MoT model can be made more parsimonious. I do that by imposing a simplified version of the model, taking out the probability weighting functions from the MoT model. I investigate whether this can predict human decision making with similar performance as the MoT model. The findings of this paper are that the gain in simplicity comes at the cost of a drop in performance. However, the utility function that my simplified model generates is of similar shape as the one in MoT.

*The views stated in this thesis are those of the author and not necessarily those of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam.

Contents

1 Introduction

A long time desire in the field of economics, has been to predict human decision making (Bernoulli, 1954). Since the 1970s, the search for prediction models for human decision making has intensified (Dawes, 1971; Levison & Tanner, 1972). Initially, it was assumed that people made their decisions to maximize their expected utility (EU) (Von Neumann & Morgenstern, 1947), but this approach has some major limitations (Davis & Jensen, 1994; Hodgson et al., 2012). Because of that, new models were introduced, like the prospect theory (PT) (Tversky & Kahneman, 1992, 1974). Briefly, this is an approach based on the assumption that gains and losses are valued differently by individuals (Levy, 1992). More recently, research focused on complementing this theory by performing data-driven research. To be more precise: machine learning models were used to forecast human decision making and to verify existing theories (Noti et al., 2016; Rosenfeld & Kraus, 2018). An issue in machine learning is the large amount of data needed for good results (Peterson et al., 2021). Peterson et al. (2021) is the first to use a properly large dataset to analyze people's choices. They do that by using machine learning methods to derive different types of models that can describe the way people make decisions.

One of these models is the "context-dependent" model. A special case of this model is the so-called Mixture of Theories (MoT) model. Here they combine the principles of PT and EU into one model, which turns out to perform extremely well. However, it does contain a lot of parameters, which makes it hard to interpret. For that reason it might be desirable to have less parameters. That is where my research comes in.

In this paper, I will investigate whether I can simplify the Mixture of Theories (MoT) model, without losing too much model performance. In other words:

"Is it possible to make the MoT model more parsimonious?"

I will answer the research question by making use of the choices13k dataset, which is a dataset that contains over 13000 choice problems. In each problem, people get the choice between two gambles: A and B. An example of a choice problem could be the following: A: 50% chance on 10 payoff, 50% on 0. B: 100% certainty of 4 payoff. For each problem there is data available about what fraction of people chose gamble B. I will use that as a benchmark for the model.

I will answer the research question by introducing a simplified version of the MoT model, and go through the same steps as MoT to determine $P(A)$, which is the probability of choosing gamble A over gamble B. This requires the usage and training of a neural network, a custom loss function, and a grid search to optimize the parameters in function V^s, which determines the value of a gamble. After finding the best performing simplified model, I compare the performance of my model with that of MoT, by means of the MSE. In addition, I compare the characteristics of the utility functions in both models.

This contributes to existing literature, since MoT is a new concept which is not well known yet. If I can find a way to simplify it without losing performance, that could be very useful for companies that have their customers make choices, like sport betting websites. If these choices can be be predicted more easily, it is easier to adapt to for these companies to maximize their profits.

The results of this paper are that after finding the best version of our simplified model, it did not come near the performance of the MoT model. The MSE Peterson et al. (2021) found for MoT was 0.0113, and in my model it was 0.0502. The gain in simplicity in my simplified model thus comes at a cost of worse performance. However, the models have similarities, as the utility function from the simplified model had the same shape as the one from the MoT model.

This paper is structured in the following way. First, I will review existing literature. After that, I further elaborate on the dataset I used. In Section 4, I describe the used methods in detail. In Section 5, the results are displayed and discussed, and lastly in Section 6 I draw a conclusion.

2 Literature review

Since the Peterson et al. (2021) paper is such a crucial paper for this research, I will will summarize what they did in this section.

In their research, they had the goal to "progress in understanding how people make decisions by using large datasets to power machine learning algorithms, constrained to produce interpretable psychological theories." Peterson did that by focusing on risky choice problems: decision makers have two options, called a "gamble", which differ in

payoffs and probabilities. For example: gamble A could be recieving 100 with a probability of 0.5 or 0 with a probability of 0.5, and gamble B receiving 200 with probability 0.5 or -100 with probability 0.5. The variable of interest is $P(A)$, the probability that gamble A gets chosen over gamble B. The goal is to find a function that can describe as many choice problems as possible. But, this is a large challenge. Since the value and probability for each possible outcome of each gamble define the amount of dimensions of the space of possible outcomes, the space of possible choice problems is extremely big. That is, because each pair of gambles could in theory require any amount of dimensions. In addition, because the function we aspire to obtain has to map choice problems from a gamble to a probability, the space of possible functions is even bigger.

That is where Peterson et al. (2021) will use machine learning methods like 'deep neural networks'. These methods are very useful for approximating functions, which is of course desirable in this case. But, as said before, for machine learning to be useful, a large dataset is needed. Also, the functions discovered are often hard to interpret, or sometimes it is even impossible. For that reason, these are mostly poor scientific models.

With the goal of overcoming these challenges, the 'choices13k' dataset (Peterson et al., 2021; Bourgin et al., 2019) was used for research. This is a dataset consisting of a little over 13000 risky choice problems. For each problem, there are a few variables for which the values are in the dataset as well. The contents of these will be discussed in greater detail in the Data section.

In their goal for finding a function that is useful for as many choice problems as possible, they use the following methods. As said, these are all based on machine learning applications. What they do, is that they define a hierarchy of decision theories. In this hierarchy, an increasing number of constraints is being reflected. They make a distinction between classes, each of which are explained in detail in the supplementary materials of Peterson et al. (2021). These are the following, ranked from most to least constraints:

1. Neural EU
2. Neural PT
3. Neural CPT
4. Value-Based
5. Context-dependent

On the "context-dependent" model, there is only one constraint, and that is that the function needs to be differentiable. Hence, this is a fully unconstrained neural network, that takes all information about the gambles as input. Peterson et al. (2021) comes to the conclusion that the "context-dependent" model comes to the best results, which implies that it results in the best predictions to view payoffs and probabilities of a certain gamble in a way that is dependent on the context of the other gamble. However, the big limitation is that this method provides limited psychological insight.

The next step Peterson et al. (2021) takes, is to analyze another restricted class of context-dependent models; Contextual Multiplicative Models. These models are used to show the key properties of the best performing context-dependent model. Peterson et al. (2021) drew the following conclusion: "Outcomes and probabilities are largely combined to form an average but are subjectively transformed in ways that depend on information across both gambles, especially outcomes." Researchers hope to find a theory that has these characteristics, while also able to be extrapolated to other datasets.

To do this, Peterson et al. (2021) compared the best performing context-dependent model with expected utility theory, and determined patterns in the shortcomings of prospect theory. This lead to the hypothesis that people use different strategies for different choice problems, which was tested by using a Mixture of Theories model. This is the model I will try to simplify in this paper. In short, it is a model that combines a convex combination of two utility functions with a convex combination of two probability functions to determine the gamble values. The weights for both convex combinations are obtained by separate neural networks. How this is done exactly, is explained in great detail in Section 4.1.

3 Data

In this section I will explain what data I used, how I transformed it to be able to work with it and how I split it in a training set and a test set.

In this research I am going to make use of the choices13k dataset (Peterson et al., 2021; Bourgin et al., 2019). This is a set that consists of 13006 risky choice problems. A risky choice problem constist of two gambles: A&B. In this dataset, gamble A is constrained

to have only two possible outcomes. In contrast, gamble B can have an unconstrained amount of possible outcomes. Out of all gambles B in this dataset, the one with the most has 9 possible outcomes.

The dataset contains a lot of variables, but not all are used in this research. First, I cleaned the data based on two variables Feedback and Ambiguity: Feedback is a boolean that indicates whether participants were informed about the actual outcome of the gamble. Because people made a choice 5 times in the same problem, this could influence their decision in the later tries. Ambiguity is a boolean that indicates whether the probabilities of outcomes in gamble B were made invisible or not. If it is false, a player had complete information, and thus the probabilities were known by the participant.

In this research I only used the choice problems which had TRUE as their Feedback value, and FALSE for Ambiguity, because that is what Peterson et al. (2021) did as well. Effictively, this means that I only consider the choice problems that are fully observable. This left me with 9831 choice problems to perform the research on.

I split the dataset in 5, using Excel:

- A 9831 x 2 matrix X_A, which contains all payoffs of all gambles A. Elements $x_{A_j,i}$ represent payoff i for gamble A of the j'th choice problem.
- A 9831 x 2 matrix P_A , which contains all probabilities for the payoffs in X_A. Elements $p_{A_j,i}$ represent the probability off payoff i for gamble A of the j'th choice problem.
- A 9831 x 9 matrix X_B, which contains all payoffs of all gambles B. Elements $x_{B_j,i}$ represent payoff i for gamble B of the j'th choice problem.
- A 9831 x 9 matrix P_B, which contains all probabilities for the payoffs in X_B. Elements $p_{B_j,i}$ represent probability off payoff i for gamble B of the j'th choice problem.
- A 9831 x 1 vector y_{true}, which contains the bRate, so the fraction of people who chose for gamble B, for all choice problems. Elements $y_{true,j}$ represent the fraction of people who chose gamble B in the j'th choice problem.

Because not all gambles B have the same amount of possible outcomes, there are empty spots in X_B and P_B, which I filled with zeros.

3.1 Training/Test split

An important part of managing the data, is to split it in a training and test set. The train set is used to base our model on, and the test set is used to validate our model. In this research, I used a 90/10 train/test split. Risky choice problems were assigned randomly to the train and test set, by making use of the train_test_split function from the sklearn.model_selection library in Python. That now gave a set of 8847 problems to base the training on, and 984 in the test set. In the Results section, I will present the outcomes of our optimization based on the train set, and after that compare it with the results of the test set.

4 Methodology

As said, I am going to investigate whether I can make the Mixture of Theories model in Peterson et al. (2021) more parsimonious. To be able to do this, we first need a deeper understanding of the Mixture of Theories model itself. That will be explained in the first subsection. In the second subsection I will explain the simplified model. In this model I take out the probability weighting functions and restrict one of the utility functions to be the identity function. In the third subsection I will discuss how I decide on the (hyper)parameters of the model.

4.1 Mixture of Theories

The Mixture of Theories (MoT) model is, as the name suggests, a mixture between two existing models: the Expected Utility approach, which bases values of gambles on the expected utility of their payoffs, and the Prospect Theory, which states that people tend to overestimate small probabilities, and underestimate large ones.

To be more precise, MoT relies on two separate convex combinations, one of two utility functions, and one of two probability weighting functions:

$$V(G_j) = \sum_i [\omega_j u_1(x_{G_j,i})) + (1-\omega_j)u_2(x_{G_j,i})][\omega_j' \pi_1(p_{G_j,i}) + (1-\omega_j')\pi_2(p_{G_j,i})] \qquad (1)$$

In (1), it holds that $G_j \in \{A_j, B_j\}$, where A_j and B_j are the two gambles of choice problem j, which is thus an element of $\{1, 2, ..., 8847\}$. In addition, ω_j and ω_j' are restricted

to be between 0 and 1. The i is an index whithin the gambles. For gambles A it ranges from 1 to 2, for gambles B from 1 to 9.

For the utility functions, we use the following representation, in the supplementary materials of Peterson et al. (2021) this is referred to as the "General Power":

$$u_m(x_{G_j,i}) = \begin{cases} \beta_m * x_{G_j,i}^{\alpha_m} & if \quad x_{G_j,i} \geq 0 \\ -\lambda_m(-\delta_m x_{G_j,i})^{\gamma_m} & if \quad x_{G_j,i} < 0 \end{cases} \tag{2}$$

In (2), it holds that $m \in \{1, 2\}$ and for i and G_j the same holds as in (1).

For the probability functions, we use the "Kahneman-Tversky" representation:

$$\pi_k(p_{G_j,i}) = \frac{p_{G_j,i}^{\alpha_k}}{(p_{G_j,i}^{\alpha_k} + (1 - p_{G_j,i})^{\alpha_k})^{\frac{1}{\alpha_k}}} \tag{3}$$

In (3), it holds that $k \in \{3, 4\}$ and for i and G_j the same holds as in (1). Because of data-fitting, one of the two functions will always be estimated as the identity function. For that reason, Peterson et al. (2021) restricts one to be the identity function in advance, to reduce model parameters.

Peterson et al. (2021) estimates two neural network simultaneously: one with the payoffs of a gamble as input, and weights ω_j and $1 - \omega_j$ for all j as output, and the other with the probabilities corresponding to these payoffs as input, and the weights $\omega_j^{'}$ and $1 - \omega_j^{'}$ for all j as output. In the next step, these weights will be used to calculate the value of each gamble using (1). These values, specific for every gamble in a choice problem, will be used to calculate the probability of choosing A over B in a specific problem ($P(A_j)$), using the following formula:

$$P(A_j) = \frac{e^{\eta * V(A_j)}}{e^{\eta * V(A_j)} + e^{\eta * V(B_j)}} \tag{4}$$

In (4), η is a parameter that captures the level of determinism in a choice. It will be one of the parameters to optimize over.

The last step is to use $P(A)$ to determine the Mean Squared Error (MSE). For this the y_{true} from our data is used. The MSE of this model is calculated as follows:

$$MSE = \frac{1}{N} \sum_{j=1}^{N} (P(B_j) - y_{true,j})^2 \tag{5}$$

where N stands for the total number of problems, 8847 in our case, and j is a problem-specific index. The neural network will be optimized by minimizing the MSE by finding the optimal values for all parameters, so all $\omega_j, \omega_j^{'}$, the η, and the parameters of the two utility and probability weighting functions.

To clarify, I illustrated the model below, where the MSE will be minimized:

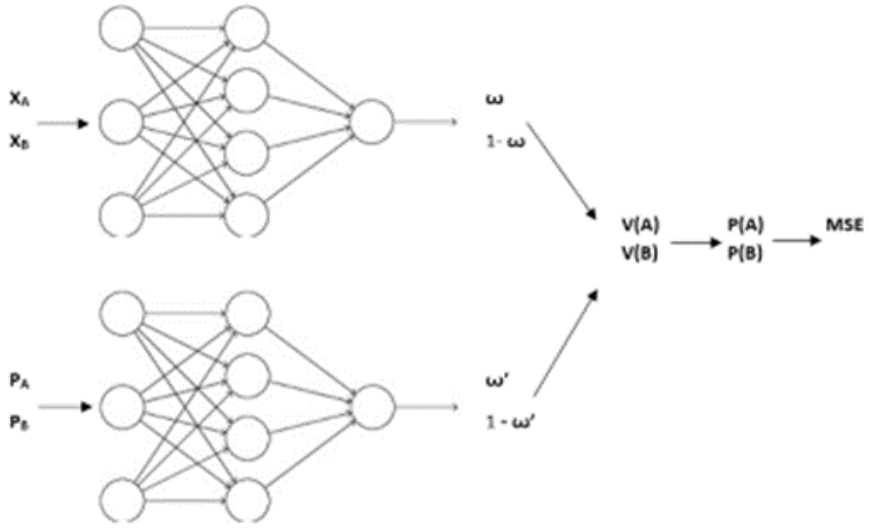

Figure 1: Illustration of the MoT model

In this research, I will see whether I can simplify this idea by reducing the number of parameters, hence try making it more parsimonious. I will do this by constructing and evaluating a different model, which will be explained in the next subsection.

4.2 Simplification of MoT

Because of the many parameters in the MoT model, it is still quite hard to interpret. For that reason, I decided to see whether I could get similar results as the MoT, but with a simplified model.

MoT imposes a function that values every gamble (1). In this function there are two different probability weighting functions ($\pi_k(p_{G_j,i})$), and two different utility functions ($u_m(x_{G_j,i})$). As said before, the weights for the two convex combinations are decided by two neural networks simultaneously. One of the probability functions was already restricted to be the identity function, but I will restrict the other one to be the identity

function as well. Consequence of this is that the principles of the Prospect Theory are no longer in the model. In addition, the weights for both probability weighting functions in (1) ($\omega_j^{'}$ and $(1-\omega_j^{'})$) no longer have to be estimated. That means that there are no longer two Neural Networks to be estimated, but only 1, the one for the weights of the utility functions. Please note that this is still a mixture of existing theories. Namely, the Expected Value theory ($u_1(x_{G_j,i})$ is the identity function) and the Expected Utility theory ($u_2(x_{G_j,i})$ is to be estimated).

I will also make a simplification in the area of the utility functions: one of the two utility functions will be restricted to be the identity function as well. That means that all parameters in (2) are equal to 1. Only the parameters of one utility function, and the η in (4) will now have to be estimated, and the weights ω_j and $(1-\omega_j)$. This will make our life a lot easier in the coding aspect, but it is still a complex problem. Let's make it more formal.

The first step is to estimate ω_j with a Neural Network, with the payoffs of all possible outcomes as input:

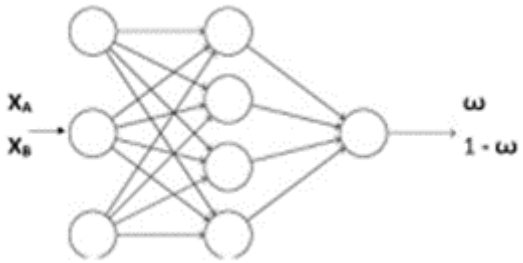

Figure 2: NN for simplified model

Next, I will use the ω_j's in a simplified value function, to calculate the values for every gamble:

$$V^s(G_j) = \sum_i [p_{G_j,i}(\omega_j * x_{G_j,i} + (1-\omega_j) * u(x_{G_j,i}))] \tag{6}$$

After I have obtained the simplified values, I will estimate the probability of choosing option A over option B in the same way as (4), but using the values obtained from (6) instead of (1). The last step is to calculate the MSE and optimize over the ω_j's in such a way that the MSE is minimized.

In my case however, it is not straighforward to minimize the loss function. That is, because a normal loss function takes in the output of the neural network, and immediately compares that with the real value. It optimizes over the weights on the nodes to minimize the loss. However, in this case a couple of steps have to be taken before the output of the neural network can be compared with the 'real value', the y_{true}. These steps also take in the train data. In other words: the loss is dependent on the train data, which I have to account for.

A custom loss function will have to be defined. After programming the neural network, and functions (4), (5) and (6), I can specify the custom loss function. In TensorFlow, which is the library I am using, a loss function is restricted to have two inputs: predicted values and real values. The predicted values are automatically taken as the output of the neural network. Because of this, I need to include the data needed to transform the predicted values, in the real values in some way. So, I decided to define a new numpy array called data, which contains the train set of the following variables: y_{true}, X_A, P_A, X_B and P_B. These will be used to transform the predicted values, thus the output from the neural network, in such a way that we can compare it to y_{true}. Whithin the loss function this array is split, and the subarrays can be used as input for the value (6), probability (4) and MSE (5) functions. Now the steps can be taken whithin the loss function in the right order, and I can compare the transformed output of the neural network to the y_{true} and optimize over this loss.

4.3 Setting the (hyper)parameters

In neural networks there are hyperparameters. These are parameters that control the learning process of the network, but cannot be derived via training. Examples are the batch size, learning rate and the number of epochs. The best values for these are only obtainable via trial and error. To test which work best, I will initialize the parameters in (4) and (2). The first utility function is already set as the identity funcion. The second utility function has a risk averse shape in Peterson et al. (2021), so I will initialize the parameters in such a way that the shape is similar. Hence, I use $\beta_2 = 1, \alpha_2 = 0.8, \gamma_2 = 0.8, \lambda_2 = 2$ and $\delta_2 = 1$. For η I will use an initial value of 0.25.

Peterson et al. (2021) uses 200 epochs and a learning rate of 0.01. To decide on which batch size I will use, I will test different values with the same values as Peterson et al. (2021) for the epochs and learning rate. I will compare the performance of batch sizes 32, 64, 128, 256 and 512, by looking at the value of the loss function of the the train set after training, as well as the shape of the learning curve. It should be descending smoothly until it converges.

Previous research (K. & C., 2020) states that with a large learning rate, a larger batch size performs better. But, I use a limited amount of epochs (200), for two reasons. The first is that Peterson et al. (2021) uses it and I will compare my results to the results of the MoT. Secondly, an increase of the amount of epochs leads to a very long run time, due to the laptop I am using. This limited amount of epochs is critical to the tradeoff of using small and large batch sizes.

A large batch size will lead to poorer generalization and slower convergence. A small batch size converges faster to good solutions, because the model already learns before it has seen all the data. But, it is not guarenteed that it converges to the optima (Keskar et al., 2016). In the Results section I will show which batch size is best for the data used in this research. Because the ADAM optimizer, which is used in this research, does not generalize as well as other optimizers like SGD (K. & C., 2020), I will use the smaller batch size if results for two or more batch sizes are similar.

After the batch size is decided, I will optimize the parameters in (2) and (4). Since the first utility function is set as the identity function, the parameters that will be optimized are $\beta_2, \alpha_2, \lambda_2, \delta_2, \gamma_2$ and η. The initial plan was do that by performing a non linear regression on these parameters, after fixing the ωs as the ωs found after training the model. However, because it kept going to local optima which did not get valid functions, I decided to use a grid search. Because this is a brute force method I had to impose some restrictions to keep te run time down. These restrictions are based on the utility function used in Tversky & Kahneman (1992). The restrictions were: $\beta_2 = 1$, $\alpha_2 = \gamma_2$ and $\delta_2 = 1$. Hence, the grid search will be done over 3 parameters: α_2, λ_2 and η. I will let η range from 0.1 to 1, α_2 from 0.1 to 2 and λ_2 from 0.1 to 3. All parameters are incremented with steps of 0.1. Hence, I will perform a grid search on all $10 * 20 * 30 = 6000$ possible combinations, and use the combination of parameters that provides the lowest value of

the loss function.

Next, I will use these parameters to train the neural network again and find new ωs. The performance of this model will be better, because the parameters are now optimized. This is the model I will compare with the MoT model.

5 Results

In the first subsection of this section I will present the results of setting the (hyper)parameters. In the second subsection I will compare the eventual model performance to the MoT model presented in Peterson et al. (2021).

5.1 Setting (hyper)parameters

The first step I take is to determine which batch size to use. After training the neural network for batch sizes 32, 64, 128, 256 and 512, I got the following results:

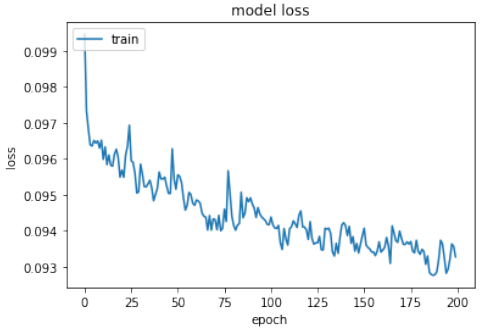

Figure 3: Batch size = 32, minimal loss = 0.0928

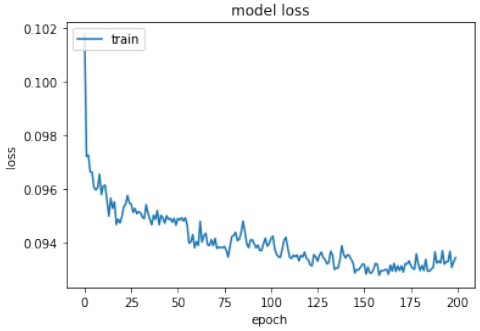

Figure 4: Batch size = 64, minimal loss = 0.0928

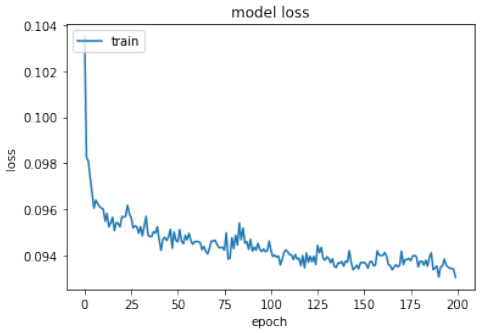

Figure 5: Batch size = 128,
minimal loss = 0.0928

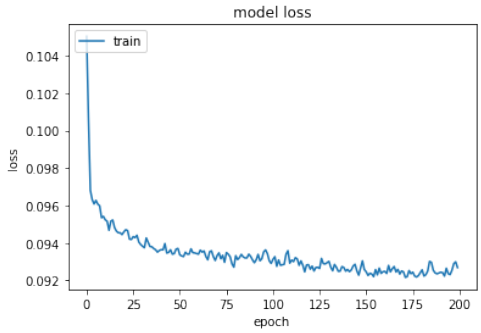

Figure 6: Batch size = 256,
minimal loss = 0.0923

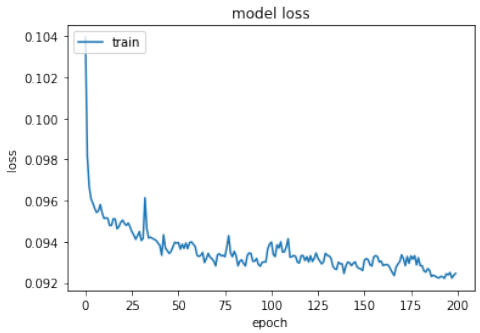

Figure 7: Batch size = 512,
minimal loss = 0.0927

If we take a look at the figures above, we can see that the batch size of 256 has the lowest value for the loss function. It is important to note that there is a random factor to these outcomes. However, I trained the model a couple of times for all batch sizes, and the minimum value of the loss function never deviated more than 0.0002 from the values in the figures. For that reason I chose to use a batch size of 256 from now on.

The next step is to optimize over the parameters $\beta_2, \alpha_2, \lambda_2, \delta_2, \gamma_2$ and η. As explained, that was done imposing restrictions on $\beta_2, \alpha_2, \gamma_2$ and δ_2, which made it so that I optimize over three parameters: α_2, λ_2 and η. That resulted in the following values:

	η	α_2	λ_2
values	0.1	0.1	0.1

Table 1: Outcomes of the grid search

This gave a loss value of 0.0550. But, because these parameters are at the lower bound of the range I searched in, I will perform another optimization for the parameters, but now with η ranging from $0.01 - 0.11$, and increments of 0.01. I will let the other two parameters stay in the same range and same increments as before.

The new outcomes were:

	η	α_2	λ_2
values	0.01	0.6	3

Table 2: Outcomes of the grid search with new range for η

When I plot the utility function with these parameters, it becomes clear that it does have a similar shape as the utility function in Peterson et al. (2021):

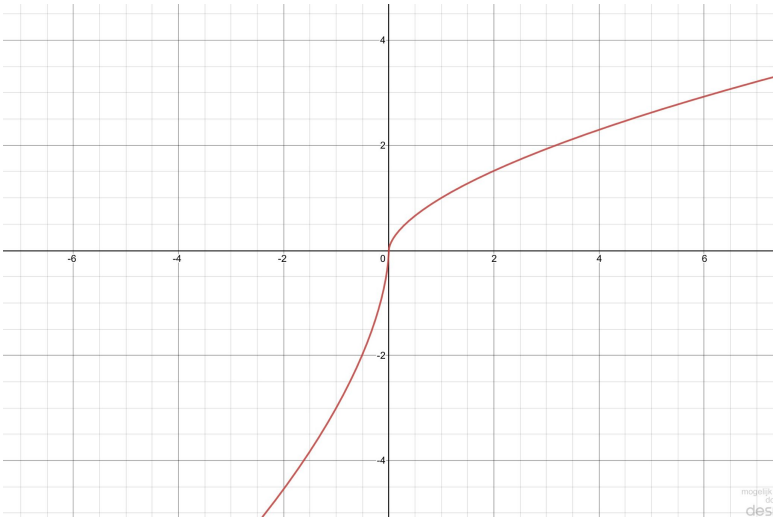

Figure 8: $U_2(x_{G_j,i})$ with optimized parameters from Table 2

One can see that the utility function is such that people value great wins less than big losses. Hence, based on this utility function, it seems that people have more risk-averse behavior. This is in line with the findings using the MoT model.

The MSE for the model that was trained with the initialized values, while using these parameters was 0.0497, which is way better than the 0.0923 found previously.

Because of the big decrease in loss I found using these parameters, I will train the neural network again while making use of the parameters in Table 2, to find new ωs and get an even better performing model. I also implement the restrictions $\beta_2 = 1$, $\alpha_2 = \gamma_2$

and $\delta_2 = 1$. This results in a new optimized model, which has a slightly lower value for the loss:

	init. param. (trained)	opt. param. (untrained)	opt. param. (trained)
min. loss	0.0923	0.0497	0.0495

Table 3: Comparison model performance with initialized/optimized parameters

In Table 3 one can see that the MSE lowers about 46% when I use the optimized parameters to train a new model, when we compare it to the old model. In addition, there is a very slight performance increase after training the model with the optimized parameters, compared to the optimized parameters in the old trained model, based on the initialized parameters.

These are thus the final results for the trained model, and in the next subsection I will compare the results of the test set to the results from Peterson et al. (2021) MoT.

5.2 Comparison simplified model with MoT

In Table 4 below one can find the values of the average MSE for out of sample predictions.

	MoT	simplified model
loss	0.0113	0.0502

Table 4: Comparison MoT and simplified model out of sample predictions

Clearly the Mixture of Theories model outperforms my simplified version. It performs around four and a half times better. This could be due to a couple of reasons. Firstly, the Prospect Theory approach outperforms the Expected Utility approach. This is shown in Peterson et al. (2021) Table s1. This clearly indicates that the Prospect Theory is a very useful approach for this problem, and in the simplified approach I used, it is completely let go of. Secondly, I did not make a distinction between dominated gambles and normal gambles. Dominated gambles are gambles that contain strictly higher payoffs than payoffs from the other gamble. For those gambles Peterson et al. (2021) finds another parameter, namely P_{dom}. This is a fixed probability for all gambles that are set up this way. However,

I treat all dominated gambles the same as all other, which could lead to worse predictions for normal gambles.

A similarity is the shape of the utility functions I found:

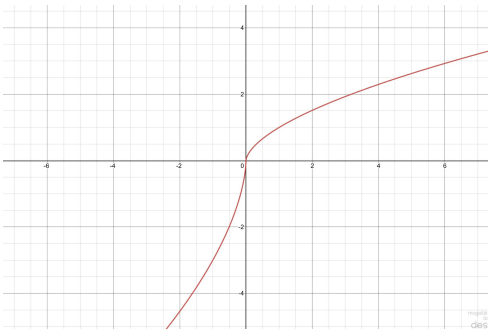

Figure 9: $U_2(x_{G_j,i})$ from simplified model

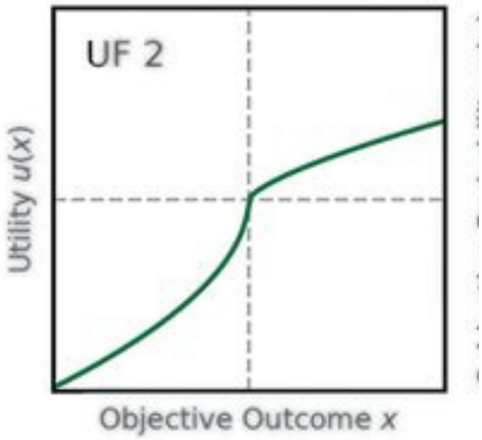

Figure 10: $U_2(x_{G_j,i})$ from Peterson et al. (2021)

In the Figure 9&10 above, it is clearly visible that the utility function I found has a similar shape as the one found by Peterson et al. (2021). That is, people value losses greater than wins. Hence, people tend to make risk-averse decisions. It is surprising that these are as similar as they are. That is, because we took out the whole part of the Prospect Theory. It is reasonable to believe that that would affect the way this function is shaped, because we now base the model on this utility function and the identity utility function instead of the four functions used in MoT. However, that is clearly not the case which is a very useful insight.

6 Conclusion

For this paper, the goal was to answer ther research question:

"Is it possible to make the MoT model more parsimonious?"

I do this by introducing a simplified setup of the MoT model, and then optimize the parameters in this model. After the parameters are optimized, I compare the MSE of

both models to draw conclusions regarding performance, and I compare the properties of the utility functions of both models.

The setup I used is as follows: using the choices13k dataset I declare a neural network, which takes the payoffs of every outcome in all gambles, so the matrices X_A and X_B, as input, and outputs weights ω_j and $1 - \omega_j$ for all j. These omegas are then used in a value function (V^s), which makes use of the output weights for a convex combination of two utilitity function. Based on these and the probabilities of all payoffs (P_A and P_B) the value of both gambles for every problem is calculated. I then use these values to determine $P(A)$, which is the probability of choosing gamble A over B for all gambles. Lastly, I use these estimated probabilities to calculate the MSE. The model is optimized by minimizing the MSE with the ω_js, parameter in $P(A)$ (η) and the parameters in one of the utility functions ($\beta_2, \alpha_2, \lambda_2, \delta_2, \gamma_2$) as parameters. I had to impose some restrictions on $\beta_2, \alpha_2, \gamma_2$ and δ_2 to make the grid search executable.

I find that the simplified model performs significantly worse than the MoT, with a MSE of 0.0502 compared to the 0.0113 of MoT. In addition, I find that leaving out the probability functions does not cause the utility function to change in shape, which is quite surprising considering all the extra information that is now imbedded in the utility function.

The main conclusion, and thus the answer to our research question, is that the gain in simplicity comes at the cost of a drop in performance. In my interpretation of the word parsimonious, the performance of the simplified model should be similar to the performance of the MoT model. In that sense, I did not manage to make the MoT model more parsimonious. That is likely due to oversimplifying. However, this does not mean that MoT cannot be made more parsimonious. Unfortunately due to time limitations, I could not try more complex methods, that still simplify MoT. A good idea for further research could be to identify patterns in risky choice problems. Then one might be able to base decision rules for which utility or probability function to use on these patterns, which takes away paramaters for the weights of the convex combination to be estimated. If a similar performance as MoT is found with a decision rule, parameters ($\omega_j^{(\prime)}$) are taken out and the MoT model in thus made more parsimonious.

References

Bernoulli, D. (1954). Exposition of a new theory on the measurement of risk. *Econometrica*, *22*(1), 23–36.

Bourgin, D. D., Peterson, J. C., Reichman, D., Russell, S. J., & Griffiths, T. L. (2019, 09–15 Jun). Cognitive model priors for predicting human decisions. In *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 5133–5141).

Davis, G. C., & Jensen, K. L. (1994). Two-stage utility maximization and import demand systems revisited: Limitations and an alternative. *Journal of Agricultural and resource Economics*, 409–424.

Dawes, R. M. (1971). A case study of graduate admissions: Application of three principles of human decision making. *American psychologist*, *26*(2), 180.

Hodgson, G. M., et al. (2012). On the limits of rational choice theory. *Economic Thought*, *1*(1, 2012).

K., I., & C., M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, *6*, 312-315.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. Retrieved from `http://arxiv.org/abs/1609.04836`

Levison, W. H., & Tanner, R. (1972). A control theory model for human decision making. In *Seventh annual conference on manual control* (Vol. 281, p. 23).

Levy, J. S. (1992). An introduction to prospect theory. *Political psychology*, 171–186.

Noti, G., Levi, E., Kolumbus, Y., & Daniely, A. (2016). Behavior-based machine-learning: A hybrid approach for predicting human decision making. Retrieved from `https://arxiv.org/abs/1611.10228`

Peterson, J. C., Bourgin, D. D., Agrawal, M., Reichman, D., & Griffiths, T. L. (2021). Using large-scale experiments and machine learning to discover theories of human decision-making. *Science*, *372*(6547), 1209–1214.

Rosenfeld, A., & Kraus, S. (2018). Predicting human decision-making: From prediction to action. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *12*(1), 1–150.

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, *185*(4157), 1124–1131.

Tversky, A., & Kahneman, D. (1992). Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, *5*(4), 297–323.

Von Neumann, J., & Morgenstern, O. (1947). Theory of games and economic behavior, 2nd rev. Retrieved from `https://psycnet.apa.org/record/1947-03159-000`

Code Appendix

******Note that Python version 3.8 was used for this research******

In this section I will explain the code I used for the computations done in this paper. I will elaborate on all the steps that are taken. In the code, I put corresponding number of the steps below to see clearly which step is taken where.

1. First, I install the packages needed. These can be found in the requirements.txt file.
2. Secondly, I load the data and change them to NumPy arrays to make the computations later a little easier.
3. The third step is to split the data files in training and test sets.
4. The next step is to define all the functions. TensorFlow requires individual observations, while the parameter optimization uses the whole dataset. For that reason, I have to declare the functions twice, in a little different way.
5. The fifth step is to declare the neural network: set the amount of nodes and declare the activation functions. Also, declare the optimizer, learning rate and loss function of the neural network.
6. The next step is to train the neural network, and store the output.
7. After training, I evaluate the out of sample prediction performance.
8. For visualization purposes, I plot the loss curve of the training of the model.
9. The next step is to optimize the parameters of utility function 2 and the η using a grid search.

For plotting the utility function I found, I used Desmos, and filled in the following formula: $\{x \geq 0 : x^{0.6}, x < 0 : -3 * (-1 * x)^{0.6}\}$