Machine Learning? In ***My*** Election? It's More Likely Than You Think: Voting Rules via Neural Networks

Daniel Firebanks-Quevedo
Advisor: Sam Taggart

Abstract

Impossibility theorems in social choice have represented a barrier in the creation of universal, non-dictatorial and non-manipulable voting rules, highlighting a key trade-off between between social welfare and strategy-proofness. However, a social planner may be concerned with only a particular preference distribution and wonder whether it is possible to better optimize this trade-off. To address this problem, we propose an end-to-end, machine learning-based framework [1] that creates voting rules according to a social planner's constraints, for any type of preference distribution. After experimenting with rank-based social choice rules, we find that automatically-designed rules are less susceptible to manipulation than most existing rules, while still attaining high social welfare.

1 Introduction

In the field of social choice, we are concerned with aggregating individual preferences in order to make collective decisions. Whether we are looking into choosing policies within a company or candidates for an election, we want to create a procedure that, given elicited preferences from a population, either chooses a candidate or derives a collective ordering from better to worse. As social planners, our main goal is to think about the characteristics that make a procedure "fair" or "reasonable" and find rules that satisfy them. For instance, we may want our rule to select an alternative that is preferred by the majority of the population if there is one, or to choose a candidate that beats all other candidates in head-to-head comparisons. Some economists thought that reasonable voting rules should make as many people happy as possible while not allowing unpopular candidates to change the winner. Satterthwaite in particular, valued a voting rule that would incentivize people to reveal their true preferences.

Unfortunately, in scenarios with 3 or more alternatives, satisfying some of these criteria simultaneously is not possible. Gibbard-Satterthwaite's theorem reminds us that there will always be scenarios where under any rule, an agent can benefit from lying and effectively manipulate the outcome of the election ([14], [24]). Similarly, if we want a rule where a voter cannot be better off without making everyone else worse-off (*Pareto efficiency*), as well as not allowing a winner to change in the presence of an irrelevant option (*Independence of Irrelevant Alternatives*), Arrow's impossibility theorem tells us that our social welfare

[1] Code for tool and experiments: https://github.com/thefirebanks/AutomatedVoting

function has to be dictatorial [2]. A general theme in impossibility theorems is the existence of certain forms of social welfare and non-manipulability that are incompatible. In practice, popular voting mechanisms such as majority rule and Borda count have also been found to be manipulable. We use these theorems as motivation to design new voting rules.

Manipulability in voting rules does come with assumptions in the setting. For instance, voters are assumed to have perfect information on everyone's preferences, and potential manipulators are assumed have the computational resources to determine whether misreporting their preferences can change an election outcome to their favor. We may question how realistic it is to hold these beliefs in a real-world scenario, but we will continue to make those assumptions for the purposes of a fair comparison with other social choice rules. So far, researchers have tried to circumvent impossibility results on non-manipulability by making further assumptions about preference structures or including randomness in the rule, yet still aiming for "universal" voting rules - that is - rules that would work in any voting scenario. We want to focus on the idea that, rather than aiming for universal voting rules that are bounded by impossibility theorems, we should create voting mechanisms that depend on the population's preferences and satisfy our requested conditions. Another way of framing it is, we want to *learn* how a specific population's preferences allow for welfare and manipulability constraints to be effectively traded.

1.1 Methodology

We can see the objective of a voting rule as some combination of desired properties for which it needs to be optimized. If we think of our problem as a *learning problem*, we are seeking a model that learns how to identify rankings or winners in a manner that satisfies a combination of desired properties (e.g., Pareto efficiency and non-manipulability) when given a population's set of preferences as input. Specifically, we want to approximate a function that learns a (non-linear) relationship between preferences and expected winners. Thus, we will use neural networks' capacity to be universal function approximators. Not only does this approach allow the designer to be flexible in the constraints that she desires to satisfy, but it optimizes this objective based on the data (a population's preferences) that is given as input.

1.2 Contributions

Our contributions are:

1. We propose an end-to-end, machine learning-based framework for the automated design of voting rules that only requires samples from a population's preferences and can be driven by a variety of desired properties, both welfare and manipulability-related.

2. We present AVNet, a neural network model that outperforms some existing voting methods in susceptibility to individual manipulation while maintaining similar levels of social welfare.

This paper is organized as follows: In Section 2, we discuss past results in social choice, machine learning in mechanism design, and the work on which our framework is based on. In Section 3, we introduce the necessary concepts and notation for the rest of the paper, as well as a formal description of our problem. In Section 4 we describe our proposed solution to the problem, in general terms. In Section 5 we dive into the design of the neural network we will be using to conduct our experiments in Section 6. Finally, Section 7 summarizes our insights and proposes ideas for future research.

2 Related Work

Social Choice. As mentioned earlier, the Gibbard-Satterthwaite theorem is one of the impossibility results that has steered a lot of the research in social choice. An important finding is that non-manipulability (*strategy-proofness*) implies both IIA and weak Pareto efficiency [1]. Previous work has also shown that general scoring rules such as Borda count or Plurality can be manipulated by a coalition of voters [21]. In light of these results, two main strategies have risen to achieve some notion of strategy-proofness.

The first one involved taking advantage of a population's preference structure, such as when every voter has their most preferred alternative in the middle of a single-dimensional spectrum (*single-peaked*) [18]. Unfortunately, single-peaked preferences are very unlikely to appear in the worst-case scenarios used to make mathematical models of voting behavior [17]. Another route has involved utilizing probabilistic social choice, specifically maximal lotteries [4], where the outcome is the probability distribution over the alternatives that is preferred the most. While there exists a tradeoff between efficiency (no agent can be made better-off without making another agent being worse off) and strategy-proofness [3], Brandl *et al.* proved that maximal lotteries can satisfy certain notions of strategy-proofness that non-probabilistic social choice rules cannot. Yet the main objection to probabilistic social choice is the fact that randomness would not be well seen in a real-world voting scenario, as it would not always reflect a population's preferences.

Machine Learning in Social Choice. Procaccia *et al.* [22] were the first to describe a voting rule as a black-box that should satisfy constraints imposed by a designer, through learning-by-example. In particular, they assumed the designer knows the "right" winner for every preference profile that she feeds to the model, and that the black-box rule belongs to a specific family of voting rules such as score-based or tree-voting. With this information, they used pairs of (preferences, winner) to model rules from the aforementioned families. Their results highlight the existence of polynomial algorithms that can approximate any score-based voting rule and the limitations of approximating tree-voting rules. Our work, although heavily inspired by [22] differs from it in that we specifically use neural networks to approximate voting mechanisms that could belong to any family, as well as including the specific objective of optimizing the trade-off between welfare and strategy-proofness. Building up from this work, Xia [27] proposed narrowing down the role of machine learning in the design of social choice rules to either elicit truthful preferences from a population or reach a consensus under some axioms defined by the designer. We extract the idea of using such axioms in our objective function when training our neural network.

Automated Mechanism Design (AMD). In 2002, Conitzer and Sandholm [8] introduced the idea of automated mechanism design, an approach that framed the mapping of preferences to outcomes as an optimization problem (similar to what we described in our methodology in 1.1). Given a class of mechanisms (or rules) to search over, the designer would find the mechanism that optimized an objective function using linear programming [9] and breadth-first search algorithms [7]. The general advantage of AMD was that, since the mechanism does not have to be general, it can circumvent impossibility results for particular settings.

Later on, convex optimization techniques from machine learning were introduced in by Dütting *et al.* [13], where a model learns payment rules that minimize an objective for combinatorial auctions. This idea was expanded to tackle problems of ranging complexity from the single-facility location problem, one-to-one matchings between hospitals and doctors, and many-to-one matchings between students and schools [20]. In particular, taking advantage of the nature of single-peaked preferences, they used Support Vector Machines to search over all possible class of weighted-generalized median rules. The objective function to minimize in this case is the difference between the generated mechanism and a strategy-proof social choice mechanism, or a stable two-sided matching.

Deep Learning in AMD. The most recent papers on AMD have seen the usage of general function-approximator rules from deep learning. In particular, Dütting *et al.* [12] leveraged neural networks to design low-regret mechanisms for the multi-item auction design problem. This work inspired Shen *et al.* [25] to use deep neural networks for revenue maximizing multidimensional auctions, revealing novel mechanisms for settings that had been unsolved for years. Another line of research is *reinforcement mechanism design* where agents learn behavior models from existing data and use deep reinforcement learning to produce a mechanism according to the environment [6]. Shen *et al.* [26] have managed to use these models to accomplish comparable results with the second price auction alternatives used in companies such as Baidu for sponsored-search auctions.

Following the first wave of neural network usage in mechanism design problems, Golowich et al. [15], proposed an extension to the multi-facility location design problem, in which we have to decide where to physically allocate a number of facilities given people's preferences. Concretely, they made the assumption that the preferences were single-peaked, and then extended their model to minimize regret. An advantage from using neural networks in these papers was the freedom to use any type of objective function (i.e constraints) as long as they can be included in the learning process of the network (i.e the loss function is differentiable). Our setting (voting) is one that resembles the multi-facility location problem and we draw ideas from such work to design the framework in this paper, yet we relax the assumption that preferences have to be single-peaked.

3 Preliminaries

We will now proceed to define our problem formally.

3.1 Environment

In a voting scenario, we have the following:

- A finite set N of n voters.
- A finite set A of m alternatives or candidates.
- For each voter $i \in N$, there is a ballot (or preference ranking) $b_i = \{c_1, c_2, ..., c_m\}$ that reflects the voter's preferences as a (descending) ranked ordering of the alternatives. We interpret $x \succ_i y$ as candidate x is preferred against y by voter i.

- A preference profile $P = \{b_1, \ldots, b_n\}$ is the set of all ballots for each voter $i \in N$.

We assume that the preferences of voters come from a particular known distribution D, and that we can draw a certain number of preference profiles from D. D can be either an existing synthetic preference/population distribution, or real-world data that contains a history of past preference profiles from people. The intuition behind this is that several examples of preferences coming from the same distribution should give the model an idea of how preference structure affects the outcome of the election.

There are multiple ways of representing preference profiles that can be used to design voting rules. Two representations will come in handy later.

- A *voting situation* is a representation of a preference profile P which contains the number of people that have a particular set of preferences b [23]. If we define P to be a profile of $n = 9$ and $m = 3$, where 4 people have the ranking $[a \succ b \succ c]$, 3 people have $[b \succ c \succ a]$ and 2 people have $[a \succ c \succ b]$ then we can define Z as the tuple of $Z = \{(4, [a \succ b \succ c]), (3, [b \succ c \succ a]), (2, [a \succ c \succ b])\}$:

$$Z = \begin{array}{c} \\ Rank_1 \\ Rank_2 \\ Rank_3 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 3 & 2 \end{array} \\ \begin{bmatrix} a & b & a \\ b & c & c \\ c & a & b \end{bmatrix} \end{array}$$

- A *voting rank matrix* R is an alternative representation of a *voting situation*, where element $R_{i,j}$ is the number of people that assigned rank i to candidate j. An important note is that this compression process leads to a loss of information that was initially given to us in the voting situation. For instance, we know that 6 people rank a as their first choice, but we don't know how they ranked b against c. If we transform our voting situation above to a voting rank matrix, we would get:

$$R = \begin{array}{c} \\ Rank_1 \\ Rank_2 \\ Rank_3 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} 6 & 3 & 0 \\ 0 & 4 & 5 \\ 3 & 2 & 4 \end{bmatrix} \end{array}$$

3.2 Social Choice Functions

Let $\mathcal{P}$ represent the set of all possible profiles for set A given n voters and $S(X)$ denote the set of all nonempty subsets of a set X. Then:

Definition 3.1. A ***voting rule*** is a social choice function $SCF : \mathcal{P} \rightarrow S(A)$ that maps sets of preference rankings to a particular winner(s).

There are many types of voting rules and they can be classified in different families depending on their procedure to choose a winner. For instance, in *Borda Count*, each candidate gets assigned a score depending on their ranking in a ballot, where a candidate in position k gets $m - k$ points (i.e $m - 1$ for the candidate preferred the most, and 0 for the candidate preferred the least). We add these points for every candidate and for every ballot, which end up yielding the final score for each candidate across the preference profile - the winner being the candidate with the highest final score. This is an example of a *scoring* rule. On the other hand, an example of a *tree-voting* rule is Copeland, where pairwise comparisons are made between each candidate to choose the winner. We define $\mathrm{Net}_P(x \succ y)$ as the difference between the number of people that prefer candidate x over y and the number of people that prefer y over x in preference profile P. *Copeland's method* essentially chooses the candidate with the highest Net score. These 2 families are some of the most explored classes of voting methods in social choice, and Procaccia's [22] results were tested on each of these families separately. In contrast, our model is not reduced to learning rules from solely these families, but it rather explores the space of all possible voting methods.

3.3 Constraints

In addition, the social planner will define a particular set of constraints C, which will both guide the learning process of our model and the evaluation metrics of the voting rule we produce. Intuitively, they should reflect the principles that a population holds with respect to making collective decision. For this to work, we have defined two possible types of constraints:

3.3.1 Welfare constraints

In a *welfare constraint*, the selection forces the winner to be a candidate that is good for a large portion of society. These candidates can generally be determined from the original preference profile, without using a voting rule. Generally this happens through counting votes and making comparisons between candidates, and the winner does not change under any rule.

Some examples of welfare constraints are:

- A *Condorcet winner* for a profile P is an alternative x that defeats every other alternative in the strict pairwise majority sense. A candidate x wins against y in the *strict pairwise majority* sense if $\mathrm{Net}_P(x \succ y) > 0$. Note that a Condorcet winner may not always exist, especially in cases with more than 3 candidates. [5]. Then, **Condorcet complicity** enforces a rule to choose the Condorcet winner if there is one.
- A *majority winner* is a candidate m that is preferred by more than half the population. Then a **majority criteria** would enforce the rule to choose the majority winner if there is one.
- Under *plurality rule*, the candidate with the most votes wins. Then a **plurality criteria** can enforce a rule to choose the plurality rule winner if there is one.

3.3.2 Counterfactual constraints

These measure the strength and consistency of a voting rule in alternative scenarios. A social choice rule SCF is susceptible to ***individual manipulation (IM)*** if given a preference profile P and an alternative profile P' with only voter i's ballot being altered, $SCF(P') \succ_i SCF(P)$. In other words, there is an alternative ordering that voter i could report in order to change the election outcome in their favor. SCF is *single voter strategy-proof* if it is not susceptible to individual manipulation.

3.4 Design Objectives

General Goal: Given a particular distribution D and a set of constraints C, we want to find a voting rule that satisfies the constraints in C with high probability over preference profile $P \sim D$.

Specific Goal: In this paper, let $C =$ *Condorcet complicity, majority criteria, single-voter strategy-proofness.* Given a particular distribution D, we want to find a voting rule for a preference profile $P \sim D$ that is better than existing voting rules in efficiently trading off susceptibility to individual manipulation, Condorcet consistency and the majority criteria.

4 Automated Mechanism Design Framework

4.1 Social Choice Rule Design as a Learning Problem

We will first give some intuition to think of social choice in machine learning terminology, then describe our formal approach to the problem. In machine learning, a traditional supervised learning task is to categorize an unlabeled item after learning a mapping from a data set of pre-labeled items. Concretely, a model learns a function $h : X \to Y$ that maps inputs $x \in X$ to labels $y \in Y$. This is called a *classification* task because the labels are *categorical*. When there are more than 2 possible categories, we end up with a *multiclass classification problem.*

We can frame the problem of finding a social choice rule $f : \mathcal{P} \to S(A)$ that maps preference profiles to a winner as a multiclass classification problem, where we have a set of inputs $X = \mathcal{P}$ and a set of possible labels $Y = S(A)$. The intuition behind the labels in our problem is that for every preference profile (our input) there exists a winner (a label) that represents the best candidate according to the constraints we have previously defined. *But where is the learning, specifically?* Well, let us assume that we can represent a voting rule as some parameterized black-box model. Then we want to learn which parameters allow our model to choose the right winner.

Something to highlight is that we do not want to set only one constraint like strategy-proofness for our model, as the rule could learn to always pick the same candidate without taking in account the preferences of people. Conversely, a rule that only values social welfare runs into the same impossibility bariers as existing voting mechanisms.

4.2 Proposed Approach

1. ***Input:*** Either draw a set Ω of preference profiles from distribution D, or read in a set of past preference profiles for a real-world population (e.g past ranked ballots from the last 10 years of elections in the UK).

2. ***Learning:*** The goal of the neural network is to find a parameterized social choice rule that minimizes a loss function designed by the social planner, composed of the constraints that they defined. The network will take preferences as input and output a score vector for each candidate, for us to choose the one with the highest score. We will optimize the parameters of the model using stochastic gradient descent, a standard method that . Since the constraint satisfaction rate function is not differentiable by

itself because it is a step function, in the next section we will explain how to use the cross-entropy function to mitigate this issue.

3. ***Evaluation:*** For a large number of candidates and voters, it would be intractable to compute whether an SCF is *single voter strategy-proof.* We instead measure strategy-proofness by sampling alternative profiles and calculating the consistency of a particular SFC.

    - Formally, the *individual manipulation rate* of a social choice function SCF is the ratio of alternative profiles for which SCF was not manipulable to the number of alternative profiles tested. Let w be the winner of a particular rule SCF for a preference profile P. Now, define a set $\mathcal{P}'$ of alternative preference profiles $\{P_0', P_1', ..., P_k'\}$ that contain a ballot b' where some candidate c_1 gets randomly swapped with candidate c_2 from rank r_1 to rank r_2. Then, we sample alternative preference profiles from $\mathcal{P}$ and check whether the winner remains to be w or not. If it is some other $w' \neq w \in A$, then we can say that the rule is manipulable for that particular preference profile. The individual manipulation rate ends up being the number of alternative profiles for which a rule is susceptible to individual manipulation divided by the number of alternative profiles tested. It is important to note that we are not aiming for a rule to be entirely single voter strategy-proof, but rather that we will look for improving upon the *IM rate* of existing voting rules.

    - We extend the idea of *IM rate* to any constraint C_i. In particular, we define the ***constraint satisfaction rate*** as the number of times a social choice function satisfied a particular constraint divided by the number of profiles it was evaluated in. For instance, if a voting rule picked the Condorcet winner for 2 out of 3 scenarios, the Condorcet satisfaction rate would be 0.66. We will evaluate voting rules in two ways: the loss function of the network, and the constraint satisfaction rate.

5 AVNet

We use neural networks to take advantage of their flexibility in approximating non-linear functions, with the idea that we can find a rule from any parameterized family of voting methods, and tweak such parameters to optimize over our loss function that represents how well the resulting voting rule satisfies our constraints. As universal approximators, neural networks *learn* relationships between their input and the output, and we can use this property to specialize voting rules according to particular distributions, which could perform better than other generalized rules.

5.1 Network Design

Recall that $\mathcal{P}$ is the set of all preference profiles that we will give as input to the network after applying transformation $T(P)$ to all $P \in \mathcal{P}$. We need to format our input as a feature vector for the network to read. For example, we can construct a voting rank matrix R from a particular preference profile P, and flatten it to get a single column vector. In this case, the data transformation is simple - flatten a matrix - and can be encoded in a function $T(P) = \text{Flatten}(P)$.

We need a hypothesis function $f : \mathcal{P} \to S(A)$ that will serve as the function template for the voting rule. In neural networks, the hypothesis function is composed from the *activation* functions that we are using in the layers. These are the transformations that will be applied at every layer of the network as the input moves forward. As opposed to the network architectures in the auction [12] and multi-facility location problems [15], the activation function and structure of the layers is not inherently related to our social choice problem. Instead, we try starting from the simplest architecture possible with the most used and recommended settings and build it up from there until we achieve good results.

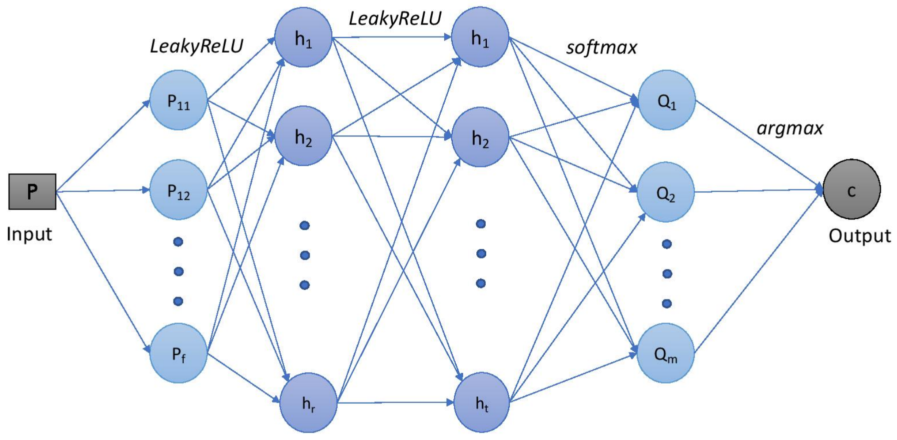

Figure 1: Proposed Architecture #5

We will use 2 hidden layers that could have either the ReLU or the LeakyReLU activation functions, followed by a final layer with a softmax activation. The softmax layer allows us to output "scores" to every candidate from $0 - 1$, which we purely do for differentiability purposes. To avoid overfitting, we added a dropout layer after each hidden layer.

$$ReLU : g(w, x, b) = max(0, wx + b) \qquad \text{(Most recommended activation function)}$$
$$Leaky\ ReLU : h(z, \alpha) = max(\alpha z, z) \qquad \text{(To avoid the dying ReLU problem)}$$
$$Softmax : s(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \qquad \text{(For calculating the candidate scores)}$$

We have attempted 14 different simple network architectures for this problem, and we will show the ones that were the most successful. These are the forms of the 4 most successful (#1, #5, #9 and #10,) architectures (shown in the Experimental Results section).

- Architecture form #1: Input, ReLU, ReLU, softmax
- Architecture form #2: Input, LeakyReLU, LeakyReLU, softmax

The number of neurons in the input layer corresponds to the number of features (the size of the flattened voting rank matrix), $r = m \cdot$ (# of features) for the first hidden layer, $t = m \cdot n$ for the second hidden layer, and m for the final layer. Note that while we are using the number of voters to define the number of neurons in one of the hidden layers, this is not necessary (nor recommendable) for cases where the ratio of voters to candidates is very large, as it can cause bottlenecks in the learning process. Instead, we can use r number of neurons for both hidden layers, shown in architecture #5 shown in figure 1.

5.2 Loss Function

The loss function for our neural network will be composed of two components, corresponding to the two types of constraints that we have defined above. We will use cross-entropy, a common loss function used in supervised learning to mitigate the non-differentiability of the constraint-satisfaction rate. Given a target distribution P and an approximation of such target distribution Q, cross-entropy measures the average number of bits that it takes to represent an event from Q instead of P [19]. Translated to our problem, we want to measure how likely our voting rule is to output the right winner.

Welfare loss
Let c^* be the candidate that would be chosen according to a constraint C_i, and let c_i be the candidate chosen by our network. Then let $p^*(c_i)$ be the probability that candidate $c_i = c^*$ and $p(c_i)$ be the probability that candidate c_i will be chosen as the winner by our network. Then the welfare loss L_o would be:

$$L_w(P) = \lambda_w \left[-\sum_{c_i \in A} p^*(c_i) \cdot \log(p(c_i)) \right]$$

Where λ_w is a multiplier that denotes how important this constraint is to our rule.

Counterfactual loss
Let $\mathcal{P} = \{P\} \cup \mathcal{P}'$ be the set of all preference profiles, where P is the original preference profile we give as input to the network, and $\mathcal{P}' = \mathcal{P} \setminus \{P\}$ be the set of alternative preference profiles with length delimited by the user. If $p'(c_i)$ is the probability that our network will choose candidate c_i as the winner given alternative profile P', then the *Counterfactual loss* L_s would be:

$$L_s(\mathcal{P}) = \lambda_s \frac{1}{|\mathcal{P}'|} \sum_{P' \in \mathcal{P}'} \left[-\sum_{c_i \in A} p(c_i) \cdot \log(p'(c_i)) \right]$$

Total loss

$$\mathcal{L}(\mathcal{P}) = \sum_{C_i \in C} L_{C_i}(P) = L_w(P) + L_s(P)$$

$$\mathcal{L}(\mathcal{P}) = \lambda_w \left[-\sum_{c_i \in A} p^*(c_i) \cdot \log(p(c_i)) \right] + \lambda_s \frac{1}{|\mathcal{P}'|} \sum_{P' \in \mathcal{P}'} \left[-\sum_{c_i \in A} p(c_i) \cdot \log(p'(c_i)) \right]$$

If we were to define any other constraint, we can add it up to $\mathcal{L}$ as long as it is differentiable. In addition, we can adjust the parameters λ_i according to the level of importance that we want to assign to our constraints relative to each other. For example, if Condorcet complicity is twice as important as non-manipulability, we would reflect it in $\lambda_{Condorcet}$ and λ_{IM}.

6 Experimental Results

6.1 Design

The goal of this experiment is to compare the performance of AVNet with other voting rules in different distributions. We made use of the SVVAMP [10] and WHALRUS [11] Python packages, which offer ways of generating preferences according to a wide array of distributions, and implementations of existing voting rules that serve as baselines to compare. For our set of experiments, we took all the available general rank-based rules as baselines. Before running our experiments, we first generated preference profiles from all the available distributions in the SVVAMP package and used them to calculate the baselines' IM rate. Then, we chose the 3 distributions with the highest IM rate and trained the AVNet on each of the distributions, comparing its performance to the baselines on individual distributions on average.

It is worth stressing that even though we tested the effectiveness of this method in different synthetic distributions, the same procedure could be applied to real-world distributions of preferences (i.e a history of past preferences of voters). Since we decided to test for instances of individual manipulation, we realized that there should not be a large difference between the number of candidates and the number of voters, as the larger the voter-to-candidate ratio, the harder it is for individual voters to make an impact in the election outcome by changing their ballots.

6.2 Setup

We generated 100 preference profiles, 80 for training and 20 for testing. These came from the Cubic, Spheroid and Ladder distributions, as they had the highest IM rate with 0.3, 0.34 and 0.35 respectively. We experimented with preference profiles with 3 candidates and 20 voters, 5 candidates and 40 voters, and 5 candidates and 80 voters. For the learning process, we use the Adam [16] optimizer, the most recommended optimization algorithm because it performs better than stochastic gradient descent, and reaches convergence faster. We set the learning rate to 0.01, and ran the training of the network for 200 epochs. For the loss function, we included 3 constraints: Condorcet complicity, majority criteria, and resistance to individual manipulation, all valued the same ($\lambda_{C_i} = 1$). We will observe both the IM rate and loss function form of the IM rate (the *IM score*) in the baselines and AVNet, while looking at the Condorcet and majority satisfaction rates to analyze the nature of the trade-off.

6.3 Results

6.3.1 Summary

During training, the network was able to balance welfare and manipulation constraints, many times achieving high scores in the Condorcet and Majority satisfaction rates, as well as achieving high IM rates. At the same time, when picking a Condorcet or majority candidate was not possible, the network focused on maximizing the IM rate. The existence of a majority/Condorcet winner in the training set heavily affected the performance of the network in the testing set. In the cubic distribution setting with 3 candidates and 20 voters, more than 50% of the profiles contained a Condorcet winner and 20% contained a majority winner. This was not the case for the setting with 5 candidates and 40-80 voters, where there was hardly a majority winner (if at all) and less than half of the profiles contained a Condorcet winner.

The rules produced by AVNet outperformed most of the baselines in almost all the distributions, finishing either first, second or third in terms of welfare and manipulation scores. However, it must be noted that the difference in performance was not large. For instance, in the setting with 3 candidates the outputs from our AVNet rules were able to be consistent in 1-4 more alternative profiles than the outputs from the baseline rules. In the settings with 5 candidates, that number ranged between 10-30 alternative profiles. As expected, AVNet achieved its highest performance during training, above any of the baselines, indicating that it was either able to learn how to navigate the trade-off between welfare and non-manipulability, or memorize the settings and their expected output. It could be concluded that had the network been exposed to more data, the gap in performance between training and testing would have been smaller. Finally, two general patterns that we will see are that different voting rules from the baseline list perform well in different distributions, and that even though AVNet had the lowest IM score in comparison to the baselines, it did not always outperform all of them in the IM rate.

While the architectures that performed best in each setting are different, we denote *AVNet** to represent a rule created by our neural network independent of architecture. In addition, all the tables presented below belong to the results of the methods in the testing set.

6.3.2 3 candidates

As we can see in tables (a) and (b), the Borda rule had a very high score in comparison to other voting rules in both the cubic and ladder distributions, but in (a) the AVNet rule ended up achieving better IM rates. In the spheroid setting from table (c), we can see that even though the AVNet rule has the lowest IM loss, the Bucklin rule ends up having a higher IM rate. For all 3 distributions, AVNet was able to attain high welfare scores, often differing from the other baselines in 1-2 preference profiles. In particular, when AVNet comes in second in (b) and (c) in terms of IM rate, it achieves better welfare scores than the next best rule in (b) and Bucklin in (c). Overall, these settings show that, despite the small amount of data, enough samples of Condorcet and majority winners in the input taught the network to recognize such winners and balance them with attaining non-manipulability.

Voting Rule	**Condorcet rate**	**Majority rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	1.0	1.0	0.5	0.91	1.41
RuleMaximin	1.0	1.0	0.45	0.872	2.082
RuleCopeland	**1.0**	1.0	0.45	0.91	1.41
RuleCondorcet	1.0	1.0	0.5	0.808	3.089
RulePlurality	0.875	1.0	0.45	0.91	3.022
RuleSchulze	1.0	1.0	0.45	0.885	1.881
RuleBucklinInstant	1.0	1.0	0.45	0.885	1.881
RuleVeto	1.0	1.0	0.45	0.885	1.813
*AVNet**	0.9375	**1.0**	**0.556**	**0.936**	**0.592**

(a) Cubic, 3 candidates, 20 voters, architecture #5

Voting Rule	**Condorcet rate**	**Majority rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	**1.0**	**1.0**	**0.65**	**0.848**	2.418
RuleMaximin	1.0	1.0	0.65	0.709	4.701
RuleCopeland	1.0	1.0	0.55	0.759	3.895
RuleCondorcet	1.0	1.0	0.55	0.696	4.903
RulePlurality	0.929	0.75	0.6	0.808	4.768
RuleSchulze	1.0	1.0	0.65	0.759	3.828
RuleBucklinInstant	0.714	0.75	0.5	0.812	7.052
RuleVeto	0.786	0.75	0.55	0.75	7.253
*AVNet**	*0.928*	*1.0*	*0.625*	*0.823*	**0.535**

(b) Ladder, 3 candidates, 20 voters, architecture #9

Voting Rule	**Condorcet rate**	**Majority rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	1.0	0.75	0.5	0.848	4.097
RuleMaximin	1.0	0.75	0.5	0.873	3.627
RuleCopeland	1.0	0.875	0.55	0.823	3.694
RuleCondorcet	1.0	0.75	0.45	0.835	4.298
RulePlurality	0.786	0.625	**0.6**	0.812	7.858
RuleSchulze	1.0	0.625	0.5	0.861	4.701
RuleBucklinInstant	0.929	0.625	0.4	**0.937**	4.231
RuleVeto	0.929	0.625	0.4	0.911	4.634
*AVNet**	**1.0**	**0.875**	*0.417*	*0.911*	**0.270**

(c) Spheroid, 3 candidates, 20 voters, architecture #1

6.3.3 5 candidates

Notice that in these settings, AVNet achieves the highest IM performance in all scenarios, with a difference in IM rate larger than the one found in the 3-candidate settings, yet there were no majority candidates in the preference structure. We could interpret this as the network becoming good at being less manipulable than the other rules, yet not having a good sense of welfare because of the lack of examples. In most scenarios, the network picked the plurality winner about 20% of the times, and the Condorcet winner 40% of the times (on average). These results yield questions on how to improve the performance on the welfare constraints, and a possible solution would be to have a larger training set, or increase the welfare lambda parameter in the loss function.

Voting Rule	**Condorcet rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	0.889	0.2	0.895	2.491
RuleMaximin	1.0	0.4	0.882	1.923
RuleCopeland	1.0	0.3	0.794	3.339
RuleCondorcet	1.0	0.15	0.496	8.194
RulePlurality	0.667	0.25	0.874	4.475
RuleSchulze	1.0	0.4	0.861	2.259
RuleBucklinInstant	**1.0**	0.35	0.941	0.946
RuleVeto	0.333	0.25	0.738	9.085
*AVNet**	*0.333*	**0.45**	**0.992**	**0.172**

(a) Cubic, 5 candidates, 40 voters, architecture #5

Voting Rule	**Condorcet rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	0.818	0.25	0.924	2.894
RuleMaximin	**1.0**	0.25	0.895	1.71
RuleCopeland	1.0	**0.3**	0.895	1.685
RuleCondorcet	1.0	0.25	0.634	5.892
RulePlurality	0.727	0.3	0.898	4.066
RuleSchulze	1.0	0.2	0.869	2.106
RuleBucklinInstant	0.818	0.25	0.882	3.535
RuleVeto	0.364	0.3	0.852	8.022
*AVNet**	*0.454*	*0.2*	**1.0**	**0.186**

(b) Ladder, 5 candidates, 80 voters, architecture #10

Voting Rule	**Condorcet rate**	**Plurality rate**	**Mean IM rate**	**Mean IM score**
RuleBorda	0.7	0.35	0.903	3.968
RuleMaximin	**1.0**	0.35	0.898	1.642
RuleCopeland	1.0	0.35	0.852	2.406
RuleCondorcet	1.0	0.2	0.578	6.801
RulePlurality	0.3	**0.35**	0.911	7.076
RuleSchulze	1.0	0.35	0.89	1.813
RuleBucklinInstant	0.7	0.3	0.886	4.231
RuleVeto	0.3	0.25	0.94	6.594
*AVNet**	*0.4*	*0.2*	**0.962**	**0.216**

(c) Spheroid, 5 candidates, 80 voters, architecture #1

7 Conclusion and Future Directions

In this paper we have presented a framework that produces voting rules given a preference distribution and a set of constraints. With enough data, our neural network can learn to effectively trade-off between welfare and non-manipulability constraints. The flexibility of our model incentivizes the exploration of unseen voting mechanisms and their exploitation in specific settings. The applications for this model range from multi-agent artificial intelligence systems to real elections, where we can now take advantage of the large space of possible rules and decide which one works better for our use case. In particular, our framework seems to perform well when we know the distribution of the preferences because we can sample as much data as necessary to improve the learning of the model. Nevertheless, AVNet could also be used in scenarios with little data as long as there is not a large voters-to-candidates ratio. Although a natural objection would be that neural networks are black-boxes and that they would not be the most democratic method to use, we can argue that the principles under which the black-box was trained were the ones that were supposed to be the most valued by a designer (and hopefully, society).

Even though we were not able to outperform all of the voting rules in our baseline list, we have seen that different rules perform better in different scenarios. Ultimately, this contributes to the idea that preference distributions should considered when deciding which rule to use. Our results highlight the importance of evaluating the practicality of impossibility theorems and not be limited by worst-case results. Once we relax the constraint of a social planner looking for a universal voting rule, we get to dive into the nuances of what they value and how to deal with variation in preferences from different distributions.

While this was a simple proof of concept, it shed light on many possible future avenues. On the social choice side, we could try to design and compare the network with other classes of voting rules, like the ones that include rounds (e.g Instant Runoff-Voting) or other types of ballots. At the same time, we can experiment with more constraints like Pareto efficiency or IIA, as well as generating more alternative preference profiles to measure different types of manipulation in voting, like coalition manipulation or bribe models. On the machine learning side, we can always add complexity to the neural network, try different values of lambda in the loss function, or simply generate more data to learn from. A key addition to the AMD Framework for this setting could be cross-validation and hyper-parameter tuning for model selection, which automates the process of testing possible architectures for AVNet, and more efficiently searches the space of possible hyper-parameters.

Acknowledgements

This work would have not come to fruition without the unlimited guidance, patience and support of my dear advisor Sam Taggart and professor Adam Eck, who constantly motivated me to explore new frontiers and taught me the beauty of theoretical and applied computer science.

I am extremely thankful for the relentless encouragement and inspiration from Charles Cui, Jane Hsieh, Sam Barr (title credits), Tumas Račkaitis, Marcelo Sandoval-Castañeda, Floor Fiers, and my mother. Finally, I want to express my appreciation to the entire Computer Science Department for its never-ending collaborative culture.

References

[1] Daron Acemoglu. *MIT 6.254, Game Theory with Engineering Applications Guest Lecture: Social Choice and Voting Theory.* May 2018.

[2] Kenneth J. Arrow. “A Difficulty in the Concept of Social Welfare”. In: *Journal of Political Economy* 58.4 (1950), pp. 328–346. DOI: `10.1086/256963`. URL: `https://doi.org/10.1086/256963`.

[3] Haris Aziz, Felix Brandt, and Markus Brill. “On the Tradeoff between Economic Efficiency and Strategy Proofness in Randomized Social Choice”. In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems.* AAMAS ’13. St. Paul, MN, USA: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 455–462. ISBN: 9781450319935.

[4] Florian Brandl, Felix Brandt, and Hans Georg Seedig. *Consistent Probabilistic Social Choice.* 2015. arXiv: `1503.00694 [cs.GT]`.

[5] Felix Brandt et al. *Handbook of Computational Social Choice.* Cambridge University Press, 2016.

[6] Qingpeng Cai et al. “Reinforcement Mechanism Design for E-Commerce”. In: *Proceedings of the 2018 World Wide Web Conference.* Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018. ISBN: 9781450356398. DOI: `10.1145/3178876.3186039`. URL: `https://doi.org/10.1145/3178876.3186039`.

[7] Vincent Conitzer and Tuomas Sandholm. “An Algorithm for Automatically Designing Deterministic Mechanisms without Payments”. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems.* AAMAS ’04. USA: IEEE Computer Society, 2004, pp. 128–135. ISBN: 1581138644.

[8] Vincent Conitzer and Tuomas Sandholm. “Complexity of Mechanism Design”. In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence.* UAI’02. Alberta, Canada: Morgan Kaufmann Publishers Inc., 2002, pp. 103–110. ISBN: 1558608974.

[9] Vincent Conitzer and Tuomas Sandholm. “Self-Interested Automated Mechanism Design and Implications for Optimal Combinatorial Auctions”. In: *Proceedings of the 5th ACM Conference on Electronic Commerce.* EC ’04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 132–141. ISBN: 1581137710. DOI: `10.1145/988772.988793`. URL: `https://doi.org/10.1145/988772.988793`.

[10] François Durand. *SVVAMP.* In: `"https://github.com/francois-durand/svvamp"`. June 2015.

[11] François Durand. *WHALRUS: Which Alternatative Represents Us, A Package for Voting Rules.* In: `"https://github.com/francois-durand/whalrus"`. Mar. 2018.

[12] Paul Dütting et al. *Optimal Auctions through Deep Learning.* 2017. arXiv: `1706.03459 [cs.GT]`.

[13] Paul Dütting et al. “Payment Rules through Discriminant-Based Classifiers”. In: *ACM Trans. Econ. Comput.* 3.1 (Mar. 2015). ISSN: 2167-8375. DOI: `10.1145/2559049`. URL: `https://doi.org/10.1145/2559049`.

[14] Allan Gibbard. “Manipulation of Voting Schemes: A General Result”. In: *Econometrica* 41.4 (July 1973), pp. 587–601. URL: https://ideas.repec.org/a/ecm/emetrp/v41y1973i4p587-601.html.

[15] Noah Golowich, Harikrishna Narasimhan, and David C. Parkes. “Deep Learning for Multi-Facility Location Mechanism Design”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 261–267. DOI: 10.24963/ijcai.2018/36. URL: https://doi.org/10.24963/ijcai.2018/36.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[17] Marie-Louise Lackner and Martin Lackner. “On the likelihood of single-peaked preferences”. In: *Social Choice and Welfare* 48.4 (Mar. 2017), pp. 717–745. ISSN: 1432-217X. DOI: 10.1007/s00355-017-1033-0. URL: http://dx.doi.org/10.1007/s00355-017-1033-0.

[18] H. Moulin. “On strategy-proofness and single peakedness”. In: *Public Choice* 35.4 (Jan. 1980), pp. 437–455. DOI: 10.1007/BF00128122. URL: http://eprints.gla.ac.uk/92237/.

[19] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[20] Harikrishna Narasimhan, Shivani Agarwal, and David C. Parkes. “Automated Mechanism Design without Money via Machine Learning”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI’16. New York, New York, USA: AAAI Press, 2016, pp. 433–439. ISBN: 9781577357704.

[21] Geoffrey Pritchard and Mark C. Wilson. “Exact results on manipulability of positional voting rules”. In: *Social Choice and Welfare* 29.3 (2007), pp. 487–513.

[22] Ariel D. Procaccia et al. “The learnability of voting rules”. In: *Artificial Intelligence* 173.12 (2009), pp. 1133–1149. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2009.03.003. URL: http://www.sciencedirect.com/science/article/pii/S0004370209000460.

[23] Reyhaneh Reyhani. “Strategic Manipulation in Voting Systems”. PhD thesis. The University of Auckland, 2013.

[24] Mark Allen Satterthwaite. “Strategy-Proofness and Arrow’s Conditions: Existence and Correspondence Theorems for Voting Procedures and Social Welfare Functions”. In: *J. Econ. Theory* (1975), pp. 187–217.

[25] Weiran Shen, Pingzhong Tang, and Song Zuo. “Automated Mechanism Design via Neural Networks”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’19. Montreal QC, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 215–223. ISBN: 9781450363099.

[26] Weiran Shen et al. *Reinforcement Mechanism Design, with Applications to Dynamic Pricing in Sponsored Search Auctions*. 2017. arXiv: 1711.10279 [cs.GT].

[27] Lirong Xia. “Designing Social Choice Mechanisms Using Machine Learning”. In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 471–474. ISBN: 9781450319935.