MSc Artificial Intelligence
Master Thesis

Principled Learning of Voting Rules

by
Valentina Lilova
15111172

September 30, 2025

36 Credits
6th January – 30th September, 2025

Supervisor:
Prof. Dr. U. Endriss

Examiner:
Dr. R. de Haan

Second reader:
Prof. Dr. U. Endriss

Universiteit van Amsterdam

Contents

Abstract

This thesis investigates whether neural networks can learn voting rules that are both accurate and principled, extending the Axiomatic Deep Voting (ADV) framework. Social choice theory defines axioms such as anonymity, neutrality, the Pareto principle, Condorcet consistency, and independence, while Arrow's impossibility theorem shows that no single rule can satisfy them all. The ADV framework addresses this limitation by encoding axioms as semantic losses, guiding models toward decisions that better reflect normative principles.

Three experiments are presented: (1) supervised learning of classical voting rules (Borda, Plurality, and Copeland), (2) axiom-guided training with a single principle, and (3) training with multiple axioms simultaneously. The results assess how effectively neural networks approximate classical rules and how axiom-based objectives improve compliance with fairness criteria. The multi-axiom setting illustrates the difficulty of reconciling conflicting principles and highlights the inherent trade-offs involved.

In summary, semantic losses can encourage neural networks to satisfy axioms, but they also bring trade-offs familiar from impossibility theorems in social choice. Training on a single axiom enforces the corresponding property but reduces similarity to classical rules. Optimizing for multiple axioms can produce conflicts, instability, or trivial solutions. Outcomes also depend on initialization: randomly initialized models typically converge faster, whereas Borda-pretrained models adapt more slowly but remain closer to classical behavior.

Acknowledgements

I would like to thank my supervisor, Ulle Endriss, for his guidance, patience, and support. He gave me the freedom to explore my own ideas while also giving me the necessary encouragement to focus on completing specific tasks. He was always there to give advice, pointed me to relevant papers and perspectives I might have missed, and ensured that I stayed on track with my submission. Before taking his electives, I knew little about computational social choice, but they drew me into the field and motivated me to write my thesis in this direction. I also thank Levin Hornischer and Zoi Terzopoulou for giving me access to their codebase before its release and for the help they offered. I am grateful to my colleagues—now friends—for the time we shared together; the study groups we formed kept us productive and helped me stay organized. Finally, I thank my family for their constant support throughout this process.

Chapter 1

Introduction

Social choice theory studies how to combine individual preferences, opinions, or votes into a single group decision. Many voting aggregation methods, also called voting rules, have been proposed, each satisfying different axiomatic properties like anonymity, neutrality, Condorcet consistency, independence, Pareto principle, and others. However, no voting rule can satisfy all desirable properties at once. A well-known result illustrating this limitation is Arrow's Theorem (Arrow, 1951), which proves that certain fairness conditions cannot be met simultaneously. As a result, each voting rule involves trade-offs between competing principles (Brandt et al., 2016b; Grandi, 2017). Neural networks, on the other hand, have proven to be effective in learning patterns from data and making complex decisions. This raises the question: can they learn voting rules that are accurate and aligned with theoretical principles?

The Axiomatic Deep Voting (ADV) framework, proposed by Hornischer and Terzopoulou (2025), combines ideas from both the fields of Computational Social Choice and Artificial Intelligence. The main idea is to train neural networks not just to replicate the output of voting rules but also to guide them toward learning rules that adhere to axiomatic principles. By defining these axioms as training objectives, the ADV framework ensures that decisions are made for the "right reasons"—that is, in a way that reflects the normative values behind the voting rules (Bender and Koller, 2020; Guerzhoy, 2024).

This thesis [1] builds on the ADV framework and extends it in several ways. It compares neural network architectures and explores possibilities to satisfy axioms by design, without explicit training. Different implementations of axioms and training methods are considered. Besides supervised rule imitation (training models to predict the winners of a known voting rule) and constraint-based learning (training models to minimize predefined objectives), both of which were implemented in the ADV framework, this thesis considers a hybrid training approach that combines the two: a model which is trained to match the output of a single voting rule is then used as a base for further training for axiom compliance.

The goal of this thesis is aligned with the goal of the ADV framework—to design models that are accurate, principled, and interpretable. The work is guided by the following research questions:

- How do architecture choices help or hurt the satisfaction of axioms?
- How to encourage axiom satisfaction during training?
- Under what conditions can learned rules outperform classical rules in terms of axiom satisfaction?
- How can we compare and explain the learned rules?

[1] The source code is available here: https://github.com/valentina98/principled-voting

In particular, it explores how to:

- train models that satisfy axioms like neutrality and anonymity by design,
- analyze trade-offs between axioms and investigate the results when trying to satisfy several at the same time,
- pretrain models on a classical rule like Borda, and use it to improve on axiomatic criteria,
- compare the learned voting rules to classical ones.

To support these objectives, the thesis first reproduces the original work of Hornischer and Terzopoulou. It then makes adaptations to the training process, introduces a new architecture (Set Transformer), and conducts a series of experiments to evaluate how well the models learn, how closely they satisfy axioms, and how similar they behave to classical voting rules.

Chapter 2

Related Work

This chapter gives an overview of the main ideas and methods related to voting rule design and learning. It starts with an explanation of some well-known voting rules and axioms from social choice theory. Then, it discusses machine learning approaches to preference aggregation, including the ADV framework. Finally, it presents some ideas related to neural network design and the methods discussed in later chapters.

2.1 Classical Voting Rules

In social choice theory, a *voting rule* is a method for selecting one or more winning alternatives from a group of candidates based on the preferences of multiple voters. Classical voting rules are well studied. Zwicker (2016) describes three commonly used ones:

- **Plurality:** Each voter gives one point to their top-ranked alternative. The alternative with the most points wins. In case that alternatives get equal points, we say there is a tie.
- **Borda:** Alternatives receive points based on their position in each voter's ranking. An alternative ranked first gets $m-1$ points, second gets $m-2$, and so on (where m is the number of alternatives). The alternatives with the highest total score win.
- **Copeland:** Alternatives compete in pairwise elections. Each alternative gets a point for every head-to-head win against another alternative and half a point for each tie. The alternatives with the highest total score win.

Despite being widely used in practice, each of them has its limitations. Depending on the context, a rule may violate established fairness criteria, for example, neutrality or monotonicity (Brandt et al., 2016a), or become vulnerable to manipulation and paradoxical outcomes when voters behave strategically (Grandi, 2017).

2.2 Axiomatic Properties

Dating back to Arrow's impossibility theorem (Arrow, 1951) and May's characterization of the majority rule (May, 1952), researchers have isolated minimal logical properties, called *axioms*, each either satisfied or unsatisfied and serving as a building block to crafting solutions with specific fairness requirements as described by Zwicker (2016).

We use axioms as a measure of how "fair" a rule is. However, while classical voting rules are designed to satisfy a set of axioms, they often violate others. For example, the Borda

rule satisfies unanimity and monotonicity but fails independence of irrelevant alternatives, as guaranteed by Arrow's impossibility theorem. Verifying by hand if a rule satisfies a set of axioms is logically demanding, time-consuming, and prone to errors. Geist and Peters (2017) point out that the idea of automating reasoning using SAT solvers and encoding axioms in propositional logic, as minimal logical blocks, is widely accepted by researchers. One direction of this research is identifying which sets of axioms can be satisfied together and which cannot. Previously, Geist and Endriss (2011) developed a SAT-based framework to search for impossibility theorems. Later, Nardi et al. (2022) developed a graph-based algorithm that finds the minimal set of axioms explaining a decision and automatically generates human-readable explanations for it.

While the binary measures of satisfaction that SAT solvers return are useful for finding strict logical constraints, they don't take into account knowledge about the expected distribution of preferences and don't provide information about the likelihood of occurring violations. Plassmann and Tideman (2014) investigated the frequency with which voting paradoxes (a violation of a desired axiom for a given profile) arise in different voting rules. They showed that for some rules, the violations occur more often than for others.

It is an appealing research goal to find voting rules that, in realistic settings and under a given preference distribution, satisfy a set of axiomatic properties more often.

2.3 Social Choice and Neural Networks

Recent research at the intersection of social choice theory and machine learning investigated how neural networks can be trained to approximate voting rules. Unlike classical voting rules designed with certain properties, neural networks learn from data. They have the potential to find approximate optima of a given objective, such as discovering an optimal voting rule for a given use case. However, key concerns are interpretability and failure to follow principled vote aggregation. This section reviews studies that have integrated neural networks into social choice research.

2.3.1 Learnability and Data Efficiency

Learnability theory studies the conditions under which a system can effectively learn patterns from data. A widely used framework for this is Probably Approximately Correct (PAC) learning, introduced by Valiant (1984, 2013). PAC learning defines a method as learnable if it can, with high probability, produce outcomes close to correct after training on a reasonably-sized dataset.

Recent work by Ge et al. (2024) shows there is a gap between learning to predict pairwise comparison outcomes and learning the true linear utility function. They assume that preferences come from an unknown linear utility function and show that when systems passively observe pairwise comparisons, they can predict future choices but cannot recover the true utility function. However, when the system actively selects which comparisons to query, it can also learn the correct utility structure.

This is relevant for voting rule learning because it shows that accurate predictions do not imply interpretability or alignment with the underlying principles. Their work uses simple linear models and does not involve neural networks, while the ADV framework addresses a related concern differently. Instead of actively selecting informative comparisons, ADV defines continuous learning objectives that provide useful training signals even when data is passively observed.

2.3.2 Predicting Outcomes Versus Learning Principles

Neural networks can accurately imitate classical voting rules. As shown by Kujawska et al. (2020), multilayer perceptrons (MLPs), a simple neural network architecture, can achieve near-perfect 99% accuracy in predicting Borda winners and a bit lower for more complex rules such as Kemeny. Burka et al. (2016, 2022) observed that MLPs, even when trained on different voting rules, tend to generalize toward Borda outcomes, which suggests an inherent bias toward simpler scoring rules. This is not necessarily a bad finding regarding MLPs. We can relate it to Occam's Razor, the idea that among competing hypotheses, the one with the fewest assumptions should be preferred (Guerzhoy, 2024). Guerzhoy argues that learning systems with simple inductive biases can still form meaningful abstractions from data. He illustrates it using the example of an octopus that learns concepts from books alone, without interacting with the world.

However, while a certain generalization is desired, an interpretable and guidable neural network has to adhere to fairness principles. When such a network is trained on target outputs, it is not guaranteed that it will "understand" the task. Bender and Koller (2020), who specialize in the natural language processing domain, demonstrated that models that perform well on benchmarks often rely on superficial patterns rather than true reasoning. This means that the accuracy of a neural network's prediction is not enough as an evaluation metric. A similar concern appears in voting theory. Hornischer and Terzopoulou (2025) show that neural networks trained to imitate classical rules can predict outcomes accurately, while still breaking axioms like neutrality and unanimity.

2.3.3 Learning The Principles Behind Voting Rules

The issue of misalignment discussed above can be addressed by training neural networks directly on axiom compliance. Armstrong and Larson (2019) were among the first to take this approach, training neural networks with the goal of satisfying the Condorcet criterion. Their results show that neural networks produce outcomes more aligned with voting principles when they are trained directly on axiomatic constraints rather than replicating existing voting rules. Using data from real elections, they demonstrate that neural networks can effectively learn rules that reflect desirable properties like Condorcet consistency.

A key difference between their method and the ADV framework is in how voting principles are treated. Armstrong and Larson enforce the Condorcet criterion as a hard constraint. Their network is trained to always choose a majority winner if it exists. This type of training provides a guarantee that the correct behavior will be learned, but it doesn't work if we don't know what the correct behavior should be. In contrast, the ADV framework is more flexible as it is designed to distinguish the severity of misalignment and to balance multiple principles at once, even if they are conflicting with one another.

Soft Condorcet Optimization (SCO) by Lanctot et al. (2025) takes a similar approach to ADV, but focuses on a single principle. It encourages the neural network to choose the Kemeny-Young winner as a "soft" incentive rather than using a "hard" penalty for violations of the Condorcet axiom during training. The main contribution of SCO is that it is fast and handles large elections where traditional methods would be too slow or infeasible to compute. The paper also analyzes when the approach works well and when it might fail, explaining how often the learned rule actually picks the correct winner. By comparison, the ADV framework is designed to support a broader set of voting principles and to balance trade-offs rather than focusing on the scalability of one principle.

2.3.4 Investigating Manipulability

Holliday et al. (2025) studied how easily different voting rules can be manipulated when voters

have limited information. They trained neural networks to discover strategic voting behaviors that influence election outcomes. The experiments showed that some rules, such as Borda, are particularly vulnerable and can be consistently manipulated even with minimal information. In contrast, Instant Runoff Voting proved to be much harder for the networks to exploit.

This type of analysis complements the ADV framework by highlighting that theoretical properties do not always predict practical behavior. While the Gibbard–Satterthwaite theorem (Gibbard, 1973; Satterthwaite, 1975) shows that many voting rules can be manipulated, it only proves that manipulation is possible in some cases. It does not quantify how often it occurs, nor does it account for realistic constraints like limited information or bounded computation. Both the manipulability study and the ADV framework rely on simulations to evaluate axiom satisfaction in practice, rather than providing theoretical guarantees.

2.3.5 Adapting to Changing Normative Criteria

In practice, the normative goals of a voting system may shift over time. A rule that was initially designed to approximate Borda scoring might later be expected to satisfy principles such as Condorcet consistency or monotonicity. This can happen, for example, when social norms change and different fairness criteria become important. In such cases, a model should adapt to new axioms without forgetting how to satisfy earlier ones.

Blair et al. (2024) address this problem in the context of continual learning with their Liquid Ensemble Selection method. Instead of relying on a single neural network, their system uses a group of models that learn together. Each model in the group can either update itself based on new data or choose to rely on another model that is performing better at the moment. This setup is inspired by liquid democracy, where agents can delegate decisions to others. By allowing models to pass responsibility dynamically, the system can adapt to new tasks without forgetting what was learned before. Their method performs well on benchmarks where the data changes over time, showing that this kind of delegation helps the system remain flexible and stable.

While the ADV framework balances normative voting principles during training, the Liquid Ensemble Selection provides a mechanism to dynamically shift the training focus over time. Both approaches use social choice theory not only to define what a good outcome is, but to steer learning toward better outcomes, even when the ideal solution is unknown.

2.4 Neural Architectures

Neural network architectures vary in how well they capture the structure of data. Some axioms, such as anonymity, can be represented as invariances in the input or output. The training process can benefit when the architecture naturally supports these invariances. This section reviews the architectures used in this thesis and discusses their strengths and limitations in the context of voting.

- **Multi-Layer Perceptrons (MLPs):** Introduced by Rumelhart et al. (1986), MLPs are simple networks that take a fixed-length vector as input and produce a fixed-length vector as output. They consist of fully connected layers of neurons, where each layer applies a linear transformation followed by a non-linear activation. MLPs can approximate complex functions, but they are sensitive to the order of the input.

- **Convolutional Neural Networks (CNNs):** They were introduced by LeCun et al. (1998) to process grid-like data, such as images. They use filters that scan across the input and are effective at detecting local patterns. However, they assume a fixed grid

structure and are not invariant to permutations in the input layout (rows and columns in the grid).

- **Word Embedding Classifier (WEC):** This architecture is used in natural language processing (Bojanowski et al., 2017), where input elements are vectors associated with "word" tokens (each token in our case represents a ranking of a voter, which may be full or partial). The vectors are averaged and passed through a classifier to determine the outcome. Because this aggregation does not depend on the order of the tokens, WEC models are invariant to the order of the input.

- **Set Transformers:** Proposed by Lee et al. (2019), these models adapt the Transformer architecture to handle sets. They use attention mechanisms that are permutation-invariant, so the order of the elements does not affect the output. This permutation invariance makes them well-suited for learning from set-structured data

Architectures like WEC and Set Transformers are promising for learning voting rules because they are invariant to the order of voters. This directly supports the anonymity axiom and reduces the need for extra training tricks like data augmentation. The ADV framework has shown that models trained with fairness constraints benefit from architectures that respect such invariances. If the architecture also treats alternatives symmetrically, this would satisfy neutrality as well. Using architectures that match the structure of voting data can make the learned rules easier to interpret and more consistent with fairness.

2.5 Connections to AI Alignment and Interpretability

While this is not the main focus of this thesis, it also connects to ongoing discussions in AI about interpretability and alignment. Recent work by Conitzer et al. (2024) supports this connection. They argue that social choice theory is well-positioned to solve problems with diverse or conflicting preferences. AI models can be trained to rely on clear axiomatic principles, which would make their behavior easier to understand and more aligned with human values. Their work highlights that ideas from voting theory are not only useful in elections but also in building fair and transparent AI systems. This is a valuable perspective, pointing out that the AI field might also benefit from voting theory concepts.

Chapter 3

The Model

This chapter discusses the technical definitions that are used throughout this thesis, as well as the structure of the axiomatic deep voting framework. It introduces notation, defines classical voting rules and desirable axioms, describes how voting rules are learned using neural networks, and explains how the final voting rule is extracted and evaluated.

3.1 General Definitions

We denote the set of *alternatives* as $A = \{a, b, ...\}$ where the number of alternatives $|A| = m$. The set of all n *voters* who submit their preferences is denoted as $N = \{1, 2, ..., n\}$. A *ballot* is a ranking (a single preference order) submitted by voter i where $R_i \in \mathcal{L}(A)$ and $\mathcal{L}(A)$ represents all possible preference orders over the set A. The preference orders are complete, meaning that every pair of alternatives is ranked, and transitive, meaning that the rankings are consistent. $\mathcal{L}$ hints that the preference orders are linear. A *profile* is the collection of all voters' ballots. Formally, if each voter $i \in N$ submits a ranking $R_i \in \mathcal{L}(A)$, then the profile is

$$\boldsymbol{\mathcal{R}} = (R_1, \ldots, R_n) \in \mathcal{L}(A)^n.$$

A *voting rule* is a function that aggregates the voters' preferences. It is denoted as F and maps each possible declared profile $\boldsymbol{\mathcal{R}}$ to a winning set so that:

$$F : \mathcal{L}(A)^n \to 2^A$$

In this framework, the voting rule F is not defined by hand. Instead, it is learned by a neural network. Note that the F does not exclude $\emptyset$ as an output. Although in principle we try to learn rules that elect winners, we don't guarantee it. The voting rules can also be irresolute, electing more than one winner.

3.2 Axioms

In social choice theory, a number of desirable properties—called *axioms*—describe how preferences should be aggregated. These axioms help evaluate how fair or consistent a voting rule is.

In the ADV framework, two forms of each axiom are defined:

- A *discrete* form of the axioms is used for evaluation. For each axiom, a function is written, returning 1 if satisfied, -1 if violated, and 0 if not applicable. 0 is needed because while some axioms (like anonymity and neutrality) apply to all profiles, others (like Condorcet) would only apply in specific cases.

- A *continuous* form of the axioms which can be used during training of the neural networks. For each axiom, a function is defined that returns a real-valued measure of how far the model's prediction is from satisfying the axiom. These loss functions, referred to as *semantic losses*, are described in more detail in Section 4.4.

The discrete target axioms considered in this thesis are:

Condorcet. If a candidate x is preferred over every other candidate by a strict majority of voters, then x must be the only winner:

$$\forall y \in A \setminus \{x\}, \quad |N^{\boldsymbol{\mathcal{R}}}_{x \succ y}| > \frac{n}{2} \quad \Rightarrow \quad F(\boldsymbol{\mathcal{R}}) = \{x\}.$$

Independence. If x is a winner and y is a loser, and the pairwise comparison between them is the same in another profile $\boldsymbol{\mathcal{R}}'$, then y should not become a winner:

$$x \in F(\boldsymbol{\mathcal{R}}),\ y \notin F(\boldsymbol{\mathcal{R}}),\ N^{\boldsymbol{\mathcal{R}}}_{x \succ y} = N^{\boldsymbol{\mathcal{R}}'}_{x \succ y} \quad \Rightarrow \quad y \notin F(\boldsymbol{\mathcal{R}}').$$

The continuous axioms and their implementation will be discussed in Section 4.5.

3.3 Neural Networks

Neural networks are used to learn the voting rule F. They take an input profile $\boldsymbol{\mathcal{R}}$ and output an arbitrary score (logit) for each alternative. These scores are then passed through a sigmoid function, resulting in winner predictions and thresholded to obtain the winners.

The process works as follows:

1. A profile $\boldsymbol{\mathcal{R}} \in \mathcal{L}(A)^n$ is sampled.
2. It is transformed into a different representation as described in Section 4.2 and passed to one of the model architectures from Section 4.3.
3. The model outputs logits $z_1, \ldots, z_m$, one per alternative.
4. A sigmoid function $\hat{y}_i = \sigma(z_i)$ is applied to obtain the prediction $\hat{y}_i \in [0, 1]$.
5. The predicted winners are the alternatives corresponding to $\hat{y}_i > \tau$, where τ is threshold.

The threshold determining the winners can be tuned, but it is set to 0.5 by default. During training, as explained in Section 4.4, the model adapts its parameters to make better predictions, so we found that for the purposes of this thesis, setting a different threshold or changing it dynamically only makes learning harder.

3.4 Training Objectives

The neural network is trained in two ways: (1) *Supervised Learning*, where it learns to replicate the output of a classical voting rule, and (2) *Axiom-based Learning*, where it is optimized to satisfy fairness principles expressed as axioms. The objective of training is to minimize the value of a loss function given the training data.

Supervised Learning. The network learns to mimic a classical voting rule using labeled data. The loss that is used is Binary Cross-Entropy, which is commonly used for binary classification problems:

$$\mathcal{L}_{\text{BCE}} = -\sum_{i=1}^{m} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]. \tag{3.1}$$

Here, m denotes the number of alternatives, $y_i \in \{0, 1\}$ is the ground-truth label for alternative i, and $\hat{y}_i \in (0, 1)$ is the predicted probability for alternative i, typically obtained from a sigmoid activation. The term $\log(\hat{y}_i)$ is applied when the label is 1, whereas $\log(1 - \hat{y}_i)$ is used when the label is 0.

Axiom-based Learning. The network is trained to minimize violations of predefined fairness principles, referred to as *axioms*. Each axiom j has an associated violation loss $\mathcal{L}_{\text{axiom}_j}$, which quantifies the extent to which the model's predictions fail to satisfy that axiom. The total axiom loss is computed as a weighted sum:

$$\mathcal{L}_{\text{axioms}} = \sum_{j} \lambda_j \mathcal{L}_{\text{axiom}_j}, \tag{3.2}$$

where $\mathcal{L}_{\text{axioms}}$ is the overall axiom-based loss, $\mathcal{L}_{\text{axiom}_j}$ is the violation loss for axiom j, and $\lambda_j \geq 0$ is the weight controlling the importance of axiom j relative to the others.

A common choice for $\mathcal{L}_{\text{axiom}_j}$ is the *Kullback–Leibler divergence* (KLD), which measures how different two probability distributions are. In this case, the model's predicted distribution $\hat{p}$ is compared to a target distribution q that satisfies the axiom:

$$D_{\text{KL}}(q \,\|\, \hat{p}) = \sum_{i=1}^{m} q_i \log \frac{q_i}{\hat{p}_i}. \tag{3.3}$$

Here, q_i is the target probability for alternative i and $\hat{p}_i$ is the model's predicted probability. The divergence is 0 when the two match exactly and positive otherwise, which makes it useful for pushing the model toward axiom-consistent predictions.

Chapter 4

Methods

This chapter describes how the data is generated, how profiles are represented as input to the models, what architectures are used, how the models are trained, and how performance is evaluated. The methods build on the original axiomatic deep voting framework (Hornischer and Terzopoulou, 2025), and are extended with new components or modified as needed for the thesis.

4.1 Dataset Generation

Synthetic datasets are generated to train and evaluate the model. The *PrefLib* Python library (Mattei and Walsh, 2013) is used to draw voters' rankings from a distribution and, when needed, to find the labels (winners) according to a given classical voting rule.

Dataset Splits. Three datasets are needed to train a model. A *training* dataset is used to update the model's parameters, a small *development* dataset is used to monitor the learning progress, and an *evaluation* dataset is used for a more accurate final assessment of the model's performance.

Determining Profile Size. We sample profiles with a varying number of voters and alternatives. A maximum number of voters n and the maximum number of alternatives m are set when the dataset is initialized. When we determine the number of voters and alternatives for a given profile (n^* and m^*), we have to prioritize bigger numbers, as there is a larger space of possible profiles $R \in \mathcal{L}(A)^{n^*}$, where $|A| = m^*$ when n^* and m^* are larger. For each generated profile, the probability of a number of voters to be n^* and the number of alternatives to be m^* is chosen as follows:

$$P(n^*) = \frac{\sum_{j=1}^{m} (j!)^{n^*}}{\sum_{i=1}^{n} \sum_{j=1}^{m} (j!)^i} \tag{4.1}$$

$$P(m^*) = \frac{\sum_{i=1}^{n} (m^*!)^i}{\sum_{i=1}^{n} \sum_{j=1}^{m} (j!)^i} \tag{4.2}$$

The denominator of both equations represents the number of all possible profiles where $n^* \in \{1, ..., n\}$ and $m^* \in \{1, ..., m\}$. The numerator of Equation 4.1 represents the number of profiles for a specific n^* and $m^* \in \{1, ..., m\}$, and the numerator of Equation 4.2 represents the number of profiles for $n^* \in \{1, ..., n\}$ and a specific m^*.

Generating Profiles. After determining the profile size, rankings are sampled from the *Impartial Culture (IC)* distribution. Each voter's ranking is drawn uniformly at random from the set of all linear orders.

An *Exhaustive Enumeration* of all possible profiles is used to evaluate models when the maximum numbers of voters n and alternatives m are small enough. However, as they increase, the number of possible profiles grows exponentially. The total number of profiles that we would have to check during exhaustive enumeration is the denominator of Equations 4.1 and 4.2:

$$total = \sum_{i=1}^{n} \sum_{j=1}^{m} (j!)^i \tag{4.3}$$

This means that, for example, for $n = 5$ and $m = 5$, there are 25,100,620,741 ways to initialize a profile. To save time, an approximate performance result is obtained by a random sampling of profiles as initially described, with a cap of 300,000 samples. This is an arbitrary number chosen so that the Python kernel does not crash while executing the experiments.

4.2 Input Representations

Each profile $\boldsymbol{\mathcal{R}}$ from the dataset is converted into a format that can be used by the neural network. Following the implementation of the axiomatic deep voting framework, the following input representations are considered:

Ordinal Encoding. In this representation, each ranking is stored as a list of numbers that show the position of each alternative. For example, $[2, 1, 3]$ means that the second alternative is ranked first. It is used as a simple and compact way to store profiles in a dataset.

One-hot Encoding. This representation is used as an input to most of the models. Each voter's ranking is represented by a matrix with shape (m, m). It contains m vectors filled with 0 and a 1 indicating the ranking position of an alternative. The profile in this case has a shape (n, m, m), where n is the maximum number of voters and m is the maximum number of alternatives.

While ordinal encoding can also be passed as an input, it may introduce unintended ordinal relationships between alternatives, which could mislead models that assume numerical rankings imply meaningful distances (e.g., interpreting a rank of 2 as "twice as preferred" as a rank of 1). One-hot encoding avoids this issue by treating each ranking position as an independent category, making it more suitable for preference aggregation tasks where the relative order matters, but the numerical differences between ranks do not.

Word Embedding. This representation is used for the WEC model. Each ranking is treated like a "sentence" made of alternative orders, and these are turned into word vectors using the *Word2Vec* method from the Genesim Python library (Rehurek and Sojka, 2011). The final input is the average of all voters' vectors, which ensures the input does not depend on the voters' order.

4.3 Model Architectures

Different model architectures can be used to learn a voting rule. Each architecture differs in how it processes the input, the fairness properties it supports by design (such as anonymity).

- **MLP:** The Multilayer Perceptron is a simple type of neural network (Lewicki and Marino, 2004). The input preferences are first flattened into a long one-hot-encoded vector and then passed through several layers of the neural network, which are made up of neurons. The neurons apply small computations to produce an output. MLPs are suitable for simple predictions. However, they do not take advantage of the structure of the input and do not support anonymity by design.

- **CNN:** The Convolutional Neural Network treats the input as an image-like grid and uses filters to detect patterns (LeCun et al., 1998). For voting data, this allows the model to capture local structures, such as how often one alternative is ranked ahead of another across different voters or positions. For instance, a CNN can identify consistent pairwise dominance (e.g., candidate A beating B) across multiple rankings and can outperform an MLP when such spatial patterns are informative.

- **WEC:** The Word Embedding Classifier (Bojanowski et al., 2017) reads each voter's ranking as a single token (for example, "2-0-1") and treats the entire profile as a sequence of such tokens. This sequence is then converted into embeddings (learned numerical representations based on their distribution across training data). The model averages the embeddings across voters to classify which alternatives are the winners. Because of the averaging operation, the order of voters does not affect the output, making the WEC model anonymous by design. For example, a profile with 5 voters each ranking 5 alternatives would be represented as a sequence of 5 ranking tokens. The model learns to associate certain ranking patterns with higher scores for particular alternatives.

- **Set Transformer:** The Set Transformer model (Lee et al., 2019) adapts the Transformer architecture to sets. It applies attention to the elements of $\boldsymbol{\mathcal{R}}$ in a permutation-invariant way, so the order of voters does not matter. This makes it anonymous by design, like the WEC model.

Capacity of a neural network refers to the complexity of functions it can represent. Different architectures structure their computations differently, resulting in different types of representational power. For instance, an MLP has limited capacity due to its uniform way of processing input. CNNs, WEC, and Set Transformer each have distinct architectures that provide unique representational strengths. Choosing the appropriate architecture depends on the specific requirements of the voting task, the structure of the input data, and the importance of fairness properties or output format.

4.4 Training

This section describes how the models are trained. Depending on the objective, training can be based on supervision, axiom-based losses, or a combination of both. The following training strategies are considered:

- **Supervised training:** The model is trained to match the outcomes of a known voting rule using binary cross-entropy loss (a loss commonly used for multi-label classification problems). Each profile input $\boldsymbol{\mathcal{R}}$ corresponds to a winner set label $\mathcal{S}$ derived from a specific classical voting rule (Borda, Plurality, or Copeland). This setup provides a clear and interpretable training objective. If a model has the capacity and is trained in an optimal way on enough data, it will, in theory, be guaranteed to converge to the correct target F.

- **Axiom-based training:** The model is trained in an unsupervised way, without being given the correct solution $\mathcal{S}$. Instead, a differentiable (continuous) signal is used as a loss. Each axiom is defined as a function that takes a batch of profiles and returns a real-valued number, measuring how far the model's predictions are from satisfying the axiom. Each axiom can be implemented in various ways, and different distance functions can be used for the measurement. The continuously defined axioms are also called *semantic losses*, loss functions designed to enforce logical constraints and guide the model during training (Xu et al., 2018).

  A common good practice is to perform both the transformations and the loss calculation at the matrix level. This is more efficient and can lead to better gradient flow, since every variable defined in the continuous axiom is tracked by PyTorch and contributes to the *gradients*. The way in which the dependent variables are defined also affects the gradients. Because model updates are driven by gradients, continuous axioms are sensitive to the details of their implementation.

  For some axioms, a target profile matrix is constructed and compared against the model's predictions. For others, the input is shuffled or otherwise modified, and the original logits of the model are compared to the new logits. For example, the independence axiom is implemented by computing the model's prediction, permuting irrelevant alternatives, and measuring how much the prediction changes for the relevant alternatives.

  Simply using the discrete form of an axiom (e.g., "is the independence satisfied: *yes*, *no*, or *not applicable*" as described in Section 3.2) would not provide as good training signal, since it returns no information on how to improve when the answer is "*no*".

  Depending on the chosen axiom(s), the optimization target might be satisfied by multiple F (for example, training a model to be neutral). This means that for axiom-based training, the training will converge eventually, but there is no single "correct" target that we expect it to converge to. If the training is initialized from a random state and no other constraints are enforced, we can see that an F that returns the $\emptyset$ would not be penalized by a model trained on the Anonymity axiom.

- **Multi-objective and constraint-based training:** When optimizing for multiple axioms, the corresponding semantic losses are multiplied by a chosen importance weight and then averaged. This is a highly nonconvex optimization problem where multiple F could satisfy the constraints for a certain input $\boldsymbol{\mathcal{R}}$, and at the same time, they might be unsatisfiable for another input $\boldsymbol{\mathcal{R}}'$.

Optimization. All models are trained using the Adam optimizer (Kingma and Ba, 2017) with standard hyperparameters such as batch size, learning rate, and number of training steps. These are reported per experiment.

4.5 Implementation of Semantic Losses

This section describes how the semantic losses were implemented in practice. For each axiom, the construction of the target and the way the model's predictions or logits are compared against it are outlined.

Condorcet. The PrefLib library is used to find all Condorcet winners for the batch of input profiles. A Condorcet mask is defined with dimensions (batch size $\times$ number voters $\times$ number alternatives) with values $\in \{0, 1\}$. The KLD distances between the model's predictions and the

Condorcet mask are found, and only the ones for which a Condorcet winner exists are kept. This is done to avoid penalizing the model for selecting winners that are not the Condorcet winner, but to penalize it for not selecting it. The obtained distances are averaged.

Independence. The input batch of profiles is given to the model to obtain the arbitrary output logits. 2 alternatives are picked as relevant ones. The input batch is shuffled across the alternatives dimension, keeping the relevant alternatives in the initial order for all voters in the batch. This is done only once per profile. The KLD distances are calculated between the original output logits and the output logits after the permutation is performed on the batch. The obtained distances are averaged.

Inadmissible. A valid and invalid alternative masks are defined, given the input batch. The distances between the model predictions and the valid alternatives mask are found. Only the distances between the trailing 0-s of the mask and the predictions corresponding to the invalid alternatives are kept and averaged.

No Winner. The maximum model prediction is found from all of the alternatives (a number $\in [0, 1]$). If this prediction is lower than the winner prediction threshold, the penalty is equal to the difference between the threshold and the highest prediction.

4.6 Evaluation Metrics

During training, the model is regularly evaluated on a small development set of 100 profiles. These profiles are sampled from the same distribution as the training data. After training, the model is tested on a larger evaluation set that includes all unique permutations of the voters and alternatives, if possible. If the number of permutations is too big, as described in Section 4.1, the evaluation dataset is randomly sampled and capped at 300,000.

Depending on the experiment, different subsets of the following metrics are reported:

- **Exact accuracy:** Measures how often the model output exactly matches the winners from a classical voting rule.
- **Hamming accuracy:** Gives partial credit when the model's winners overlap with the correct ones. It counts how many individual alternatives are predicted correctly.
- **Admissibility:** Checks how often the model returns at least one winner from the valid alternatives. Invalid alternatives are not considered.
- **Discrete axiom satisfaction:** For each axiom, reports how often the model satisfies it.
- **Rule similarity:** Compares the model's predictions to those of classical rules, describing how closely the model behaves to them. This is similar to the Exact Accuracy metric, but it is calculated for a list of rules.

These metrics show whether the model makes correct predictions and whether it satisfies axioms from voting theory. All results are reported in %.

It is worth noting that the loss values in the plots are shown on a logarithmic scale. This makes it easier to see both large and small changes, since losses can vary across several orders of magnitude. Unlike accuracy metrics, losses such as KLD defined in Section 3.4 are not limited the interval $[0, 1]$. In particular, divergences such as KLD can take values greater than 1 when the predicted distribution differs strongly from the target distribution. Using the log scale

ensures that both the early, high-loss phase and the later, low-loss phase of training are visible in a single figure.

Chapter 5

Experiments

This chapter describes three experiments. Each experiment builds on the previous one and explores a different way of training models to aggregate preferences. The first experiment uses supervised training to mimic classical voting rules. The second continues training a learned rule approximation with a single fairness principle. The third trains on multiple axioms and analyzes trade-offs and similarities.

5.1 Exp1: Supervised Learning of Classical Rules

The first experiment establishes a baseline by training neural networks to replicate classical voting rules through supervised training. This setup provides a controlled environment in which each model can directly learn and be compared to well-defined outcomes, before introducing fairness objectives in later experiments.

5.1.1 Objective

Supervised training requires labels of the correct winning sets. As described in Section 4.4, these labels are determined by a known voting rule: Borda, Plurality, or Copeland. A benefit of having this as a first stage in the training process is that it helps us to establish a baseline, assess the capacity of different models, and check how well they can learn the mapping from profiles to winner sets.

The original paper (Hornischer and Terzopoulou, 2025) also used supervised learning in its first experiment. It trained neural networks on data from multiple classical voting rules and discussed their misalignment and poor "understanding" of the goal. While the learning target in the original paper was labels from different voting rules, here, the labels from one classical voting rule are used at a time. Despite the argument that a neural network that merely mimics data is not aligned, one that is trained to achieve a prediction accuracy of a classical voting rule close to 100% is a near-perfect approximation of that rule. This allows us to treat the learned rule as interpretable since the outcomes of the classical rule are known.

The learned rules are not that useful on their own in small settings (small number of voters and candidates) because we can easily compute the outcome without a neural network, but they are a stable and interpretable foundation for subsequent axiomatic learning. As mentioned in Section 4.4, semantic losses are often complex to define, difficult to optimize, and sensitive to the balance between competing constraints. When used alone, they can lead the model to converge to unintended or poorly performing voting rules. In contrast, classical voting rules are well-understood and already satisfy some axioms. Therefore, rather than starting from a random state, a model that is pretrained to match exactly the output of (overfit to) a classical voting rule is a better starting point for further axiom-based optimization.

5.1.2 Experimental Setup

Settings with a maximum number of 3 or 5 alternatives and 3 or 5 voters are used to train each model. The binary cross-entropy loss explained in Section 4.4 is used. The batch size is 32, meaning that each update step is based on the training loss from 32 profiles. The random seed is set to 42 for reproducibility. The models are trained on a CPU. The evaluation is based on the exact match accuracy, Hamming accuracy, and rule similarity metrics defined in Section 4.6. The WEC is chosen as the main model for comparison, as it is the best-performing model in the ADV Framework.

Configurations. The experiment is conducted in 4 configurations that vary the maximum number of voters and alternatives. The number of steps for which each model is trained is manually passed as an input parameter to the experiment. In the setting 3×3 (a maximum of 3 alternatives and a maximum of 3 voters) and in the setting 3×5 (a maximum of 3 alternatives and a maximum of 5 voters), the model is trained for 1,000 steps. In the 5×3 and 5×5 settings, the training is done on 10,000 steps to account for the increased complexity.

Data splits. The *training set* is sampled from the IC distribution, as described in Section 4.1. Its size depends on the batch size and the number of training steps. 32,000 profiles and their corresponding winning sets are generated for the setups 3×3 and 3×5, and 320,000 for the setups 5×3 and 5×5. The *dev set* consists of 100 profiles sampled from the IC distribution. They do not change throughout training and are used to track loss and the accuracy metrics. On every 100 gradient steps, a partial evaluation is run on them. In the *test set* are included all possible input profiles for the given number of voters and alternatives (when possible), allowing full evaluation of the performance of the model. This is a convenient way to verify the results, but it isn't a scalable approach. As described in Section 4.1, when an exhaustive enumeration is computationally hard (for the setups with 5 alternatives), a random generation with a cap of 300,000 is used to approximate the results. This results in 275 evaluation profiles for the 3×3 setup, 9,397 for the 3×5 setup, and 300,000 random evaluation profiles sampled from the full spaces of 1,757,219 and 25,100,620,741 profiles for the 5×3 and 5×5 setups.

Model Parameters. The *Multi-Layer Perceptron (MLP)* has three hidden layers of size 256, each followed by layer normalization and ReLU activation, and an output layer of size equal to the number of alternatives. Its total/trainable parameters for the 5×5 setup are 166,661. The *Convolutional Neural Network (CNN)* uses two convolutional layers with kernel sizes $[3, 1]$ and $[1, 3]$ (32 output channels each), followed by three fully connected layers of size 256 with layer normalization and ReLU after each, and a final output layer of size equal to the number of alternatives. Normalization is applied to the fully connected layers. It has 212,005 total/trainable parameters for the 5×5 setup. The *Word Embedding Classifier (WEC)* uses a corpus of 10,000 profiles to learn 100-dimensional embeddings with the skip-gram algorithm. The context window equals the maximum number of alternatives in the experiment, and the averaged embedding is passed through three hidden layers of size 256 with layer normalization and ReLU before the output layer of size equal to the number of alternatives. The total/trainable parameters for the 5×5 setup are 12,200. The *Set Transformer* uses 256-dimensional embeddings, one Induced Set Attention Block with 8 inducing points and 4 attention heads, Pooling Multihead Attention with a single seed, and a final output layer of size equal to the number of alternatives, with layer normalization applied after the embedding, inside the attention block, and before the output head. The total/trainable parameters for the 5×5 setup are 1,198,597.

5.1.3 Results

The MLP, CNN, WEC, and Set Transformer models all learn to mimic the classical voting rules in the easiest 3×3 setting with 100% exact match accuracy. Most models don't fully learn the classical voting rules in the 3×5 setting, however, the accuracy values are close to 100%, as shown in Table 5.1. Only the Set transformer is an exception, being able to learn Borda and Plurality with 100% exact match accuracy and Copeland with 99.66% exact match accuracy. The MLP, CNN, WEC, Set Transformer models take between half a minute and 3 minutes to train, the MLP being the quickest, and the Set Transformer being the slowest. The learning curves across the 4 models, 3 rules, and 2 setups are similar, so only the WEC learning curves are shown in Figure 5.4.

For the 3×5 configuration, performance remains high for Borda across all models, but none of them fully reach 100%. Plurality is learned with 100% exact match accuracy by all models except the WEC, which has 99.99%. When learning Copeland, the MLP and CNN show a clear decline in accuracy (87.43% and 86.19%), while the WEC maintains high performance at 98.55%, and the Set Transformer reaches 97.69%. In terms of runtime, the MLP and CNN complete training in about half a minute, while the WEC and Set Transformer require around 2–3 minutes, the Set Transformer being slower.

For the 5×3 configuration, all models learn Borda well (as seen in Table 5.2 and Figure 5.5), with the lowest exact match accuracy being 98.38% for the CNN model. Plurality is learned perfectly by all 4 models, reaching 100% exact match accuracy except the CNN model, which has 99.99%. On Copeland, the MLP and CNN show a substantial drop in accuracy (87.43% and 86.19%), whereas the WEC remains robust at 98.55%, and the Set Transformer follows closely with 97.69%. Training times vary from about 3 minutes for the MLP and CNN to the WEC model, taking 6-7 minutes, and the Set Transformer, 11-13 minutes.

For the 5×5 configuration, the performance on Borda remains high. The WEC and Set Transformer are the most accurate (99.03% and 99.68%), while the CNN is the least accurate at 92.42%. Plurality continues to be learned almost perfectly, with all models reaching at least 99.97%. The MLP and CNN show a sharp decline in accuracy of the learned Copeland (73.12% and 71.55%), while the WEC maintains a strong performance at 95.56%, followed by the Set Transformer at 92.77%. In terms of runtime, the MLP remains the fastest at around 2.5 minutes, followed by the CNN at about 4 minutes, the WEC at 10-11 minutes, and the Set Transformer at 13-14 minutes. The learning curves for this configuration are shown in Figures 5.1, 5.2, and 5.3

5.1.4 Discussion

This experiment shows that the models can learn to mimic classical voting rules, however, with the increased complexity, the learning takes more time and requires a bigger capacity of the models. As described in Section 5.1.2, the 3×3 and 3×5 setups are trained on more profiles than the number of exhaustively enumerated evaluation profiles, meaning that it is likely that all unique profiles are seen during training. On the other hand, during the training of the 5×3 and 5×5 setups, most of the possible profiles have not been encountered, so to have a high accuracy, the models have to learn to generalize to unseen profiles. A model that learns to approximate a voting rule shouldn't necessarily have to see all possible profiles, and a model that encountered all possible profiles during training, in theory, may not have the capacity to make the correct prediction for all of them. The performance on the 5×5 setup will be used as a benchmark for the ability of the models to generalize. From best to worst performance (as reported in Section 5.1.3), they are ordered as follows: set transformer, WEC, MLP, CNN. The set transformer is expected to perform best and to be followed by the WEC, as they are anonymous and have the highest number of learnable parameters as described in Section 5.1.2.

However, although the CNN has more learnable parameters than the MLP, it has a worse performance.

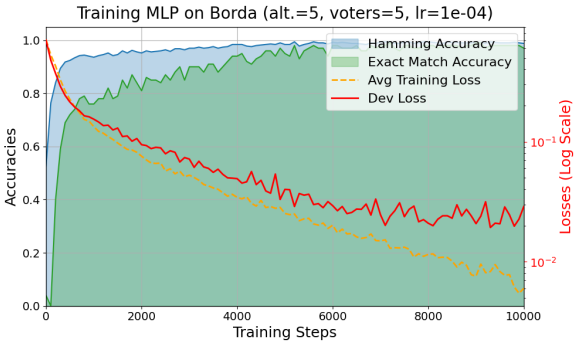
Another finding is that Plurality is simpler for the models to learn than Borda, while Copeland is harder than Borda. This makes sense as for Plurality, only the first-ranked alternative is meaningful. For the WEC model, this would mean that all ranking tokens with the same first alternatives should be treated the same (have the same representations). For Borda, the alternatives become less important as they appear later in the ranking. The WEC model manages to update its parameters so that when the word tokens are averaged, the alternative(s) that are most similar to the initial positions of the rankings are selected. In order to mimic Copeland, a model has to point to the alternative(s) winning the most pairwise comparisons. The WEC model performs best on this classical voting rule, although it also has a lower accuracy than for the other 2 voting rules. Given that this model works with word tokens representing rankings (the smallest unit that it has access to is the ranking of a voter), it is harder for it to select the correct winners after averaging the tokens.

	3×3				3×5			
Rule & Metric	**MLP**	**CNN**	**WEC**	**ST**	**MLP**	**CNN**	**WEC**	**ST**
Borda								
Test Loss	0.0002	0.0001	0.0001	0.0001	0.0008	0.0005	0.0002	0.0000
Exact M. Acc.	100.0%	100.0%	100.0%	100.0%	99.96%	97.99%	99.65%	100.0%
Hamming Acc.	100.0%	100.0%	100.0%	100.0%	99.99%	99.57%	99.81%	100.0%
Time (min)	0.24	0.33	1.38	1.50	0.31	0.31	0.40	2.43
Plurality								
Test Loss	0.0001	0.0001	0.0001	0.0000	0.0004	0.002	0.0001	0.0000
Exact M. Acc.	100.0%	100.0%	100.0%	100.0%	99.99%	99.96%	99.97%	100.0%
Hamming Acc.	100.0%	100.0%	100.0%	100.0%	100.0%	99.99%	99.99%	100.0%
Time (min)	0.26	0.33	1.33	2.55	0.31	0.31	1.74	2.43
Copeland								
Test Loss	0.0002	0.0001	0.0001	0.0000	0.00028	0.0030	0.0005	0.0002
Exact M. Acc.	100.0%	100.0%	100.0%	100.0%	93.86%	93.00%	99.32%	99.66%
Hamming Acc.	100.0%	100.0%	100.0%	100.0%	97.25%	97.11%	99.69%	99.89%
Time (min)	0.26	0.30	1.68	1.68	0.28	0.23	1.41	2.75

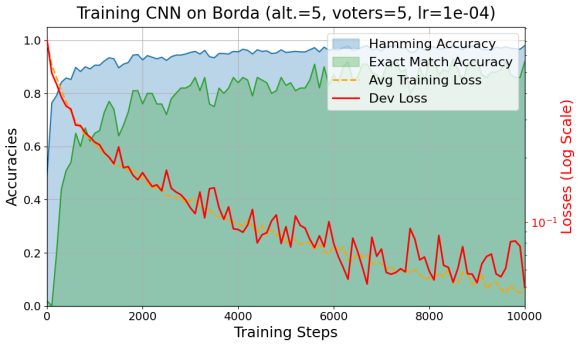
Table 5.1: **Supervised Learning Results on Easy Setups:** We present the evaluation results of the MLP, CNN, WEC, and Set Transformer models following the first experiment on the easy setups. On the left side of the table are shown the results for 3 alternatives and 3 voters, and on the right side, for 3 alternatives and 5 voters.

	5×3				5×5			
Rule & Metric	**MLP**	**CNN**	**WEC**	**ST**	**MLP**	**CNN**	**WEC**	**ST**
Borda								
Test Loss	0.0001	0.0005	0.0002	0.0000	0.0003	0.0013	0.0002	0.0001
Exact M. Acc.	99.88%	97.99 %	99.14%	99.94%	99.06%	92.42 %	99.03%	99.68%
Hamming Acc.	99.97%	99.57 %	99.82%	99.99%	99.79%	98.38 %	99.80%	99.93%
Time (min)	3.03	3.96	7.18	13.2	2.44	4.44	6.90	13.55
Plurality								
Test Loss	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Exact M. Acc.	100.0%	100.0%	99.99%	100.0%	99.99%	99.98%	99.97%	100.0%
Hamming Acc.	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	99.99%	100.0%
Time (min)	3.11	2.77	6.48	11.95	2.39	3.35	10.84	13.90
Copeland								
Test Loss	0.0022	0.0024	0.0004	0.0005	0.0051	0.0054	0.0010	0.0014
Exact M. Acc.	87.43%	86.19%	98.55%	97.69%	73.12%	71.55%	95.56%	92.77%
Hamming Acc.	96.98%	96.61%	99.66%	99.48%	92.94%	92.47%	99.00%	98.27%
Time (min)	2.88	2.72	7.02	11.12	2.35	4.18	10.42	13.1

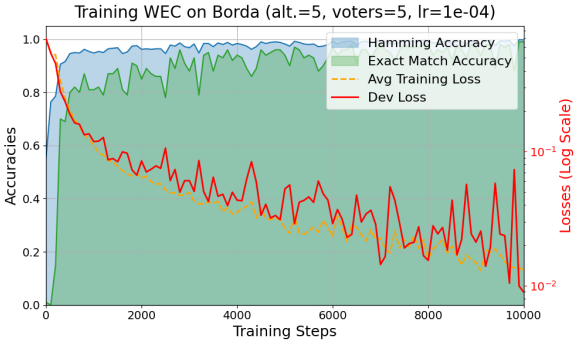
Table 5.2: **Supervised Learning Results on Harder Setups:** We present the evaluation results of the MLP, CNN, WEC, and Set Transformer models following the first experiment on the easy setups. On the left side of the table are shown the results for 5 alternatives and 3 voters, and on the right side, for 5 alternatives and 5 voters. Note that the evaluation on these setups wasn't exhaustive, and the obtained numbers are approximate.



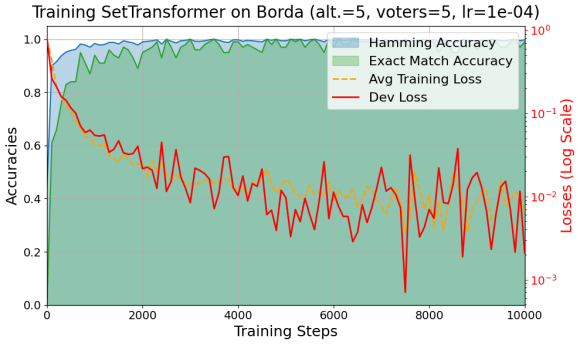
(a) Learning Curve MLP



(b) Learning Curve CNN

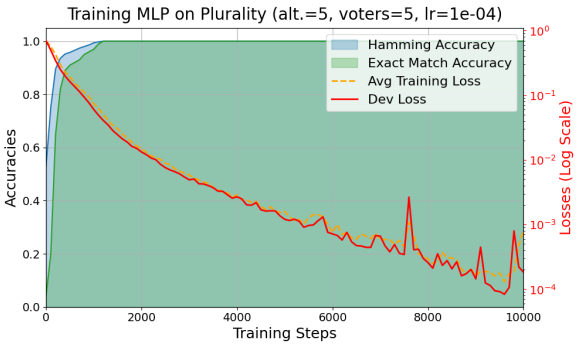


(c) Learning Curve WEC

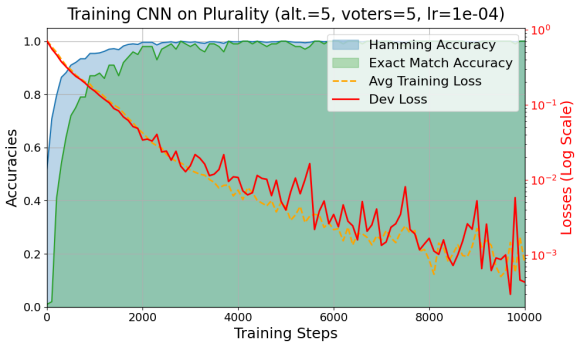


(d) Learning Curve ST

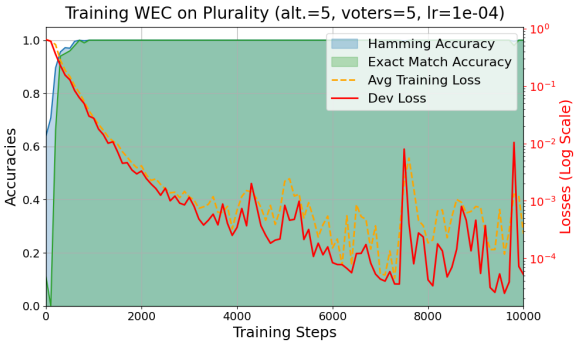
Figure 5.1: **Learning Curves of Borda:** The first experiment is conducted for each model on the Borda rule and the hardest 5×5 setup.



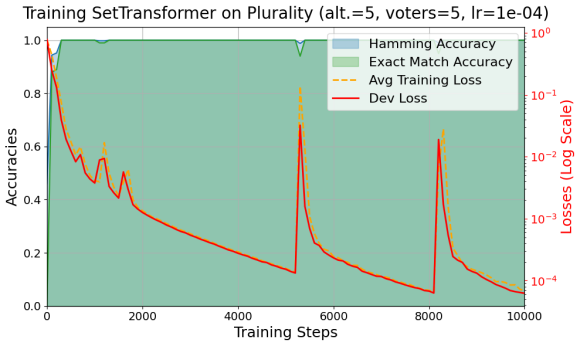
(a) Learning Curve MLP



(b) Learning Curve CNN

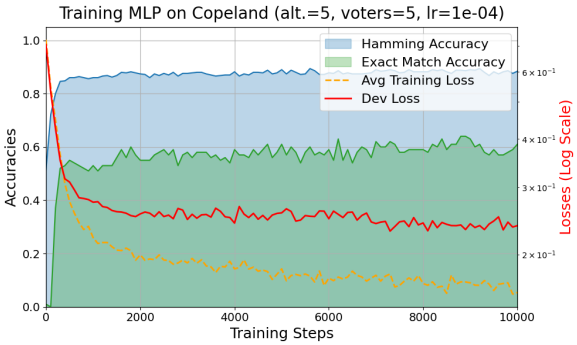


(c) Learning Curve WEC

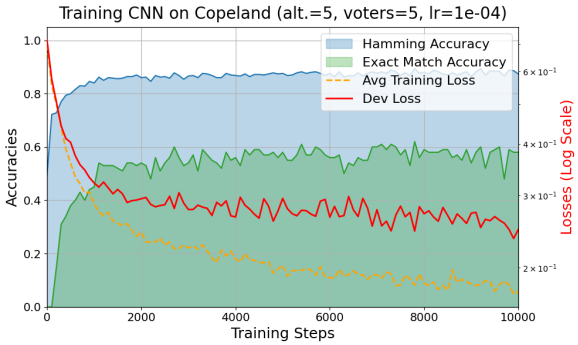


(d) Learning Curve ST

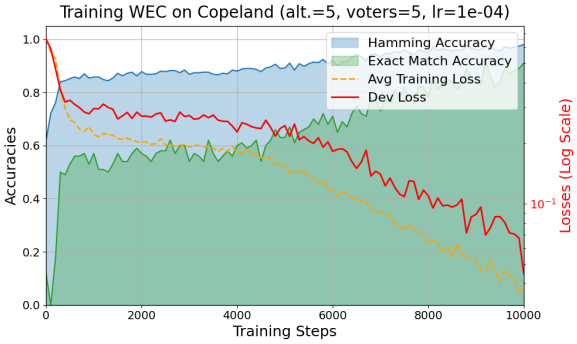
Figure 5.2: **Learning Curves of Plurality:** The first experiment is conducted for each model on the Plurality rule and the hardest 5×5 setup.



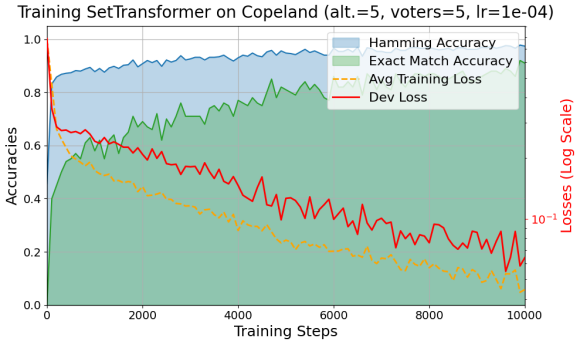
(a) Learning Curve MLP



(b) Learning Curve CNN

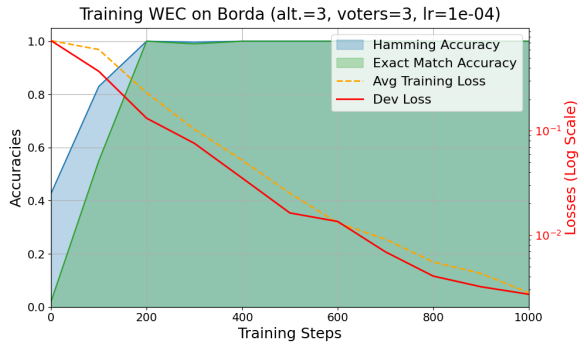


(c) Learning Curve WEC

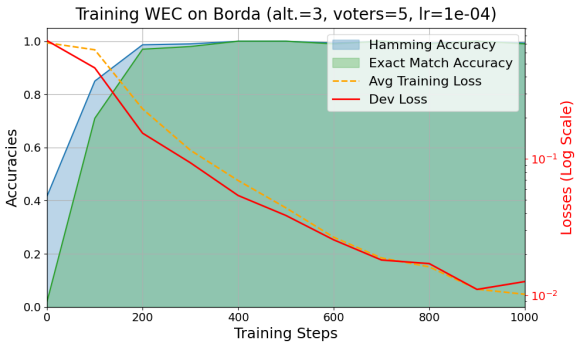


(d) Learning Curve ST

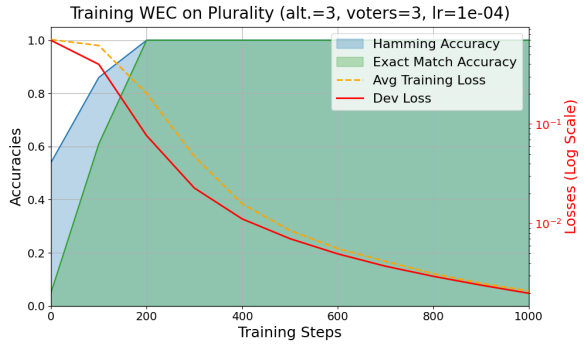
Figure 5.3: **Learning Curves of Copeland:** The first experiment is conducted for each model on the Copeland rule and the hardest 5×5 setup.



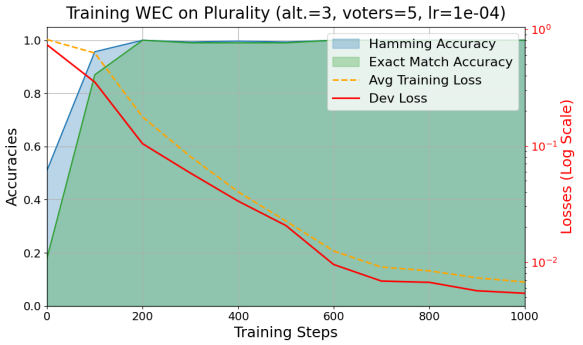
(a) Borda Learning Curve 3×3



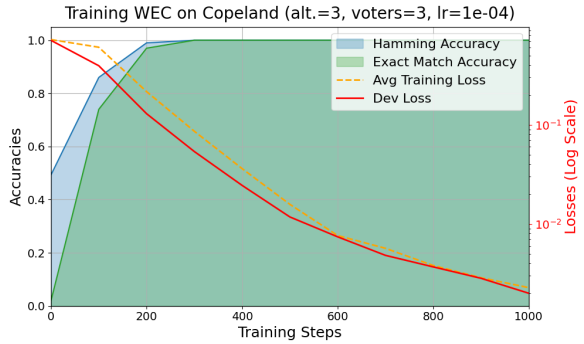
(b) Borda Learning Curve 3×5



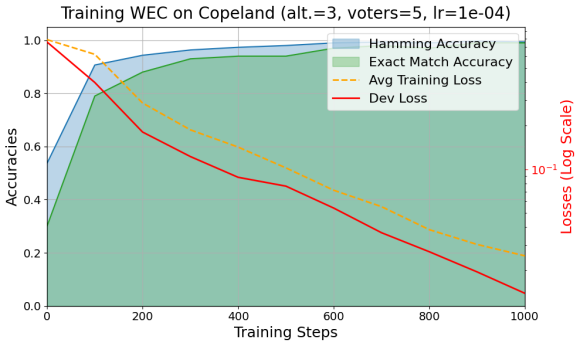
(c) Plurality Learning Curve 3×3



(d) Plurality Learning Curve 3×5

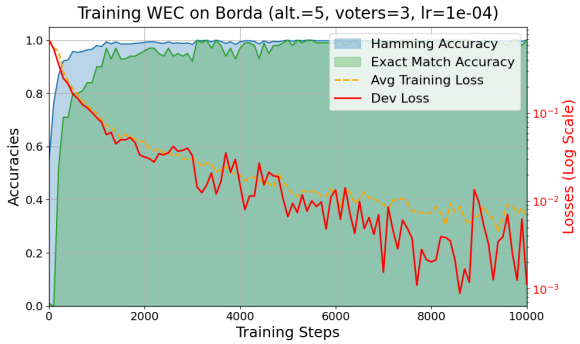


(e) Copeland Learning Curve 3×3

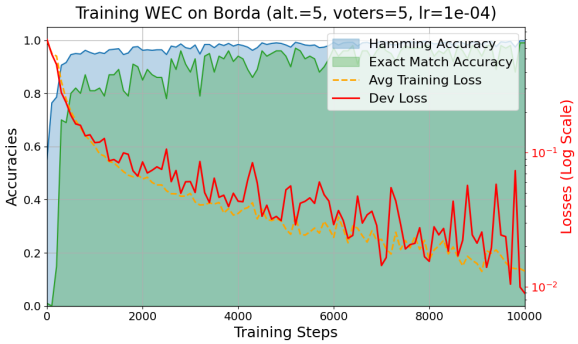


(f) Copeland Learning Curve 3×5

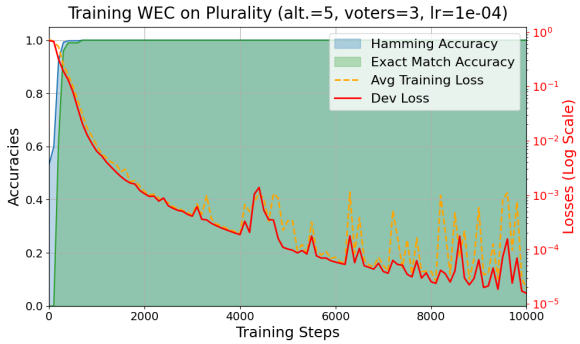
Figure 5.4: **Learning Curves for 3×3 and 3×5:** The WEC model is trained for 1,000 steps to approximate three classical voting rules for the 3×3 and 3×5 configurations. We see that the learning curves are similar.



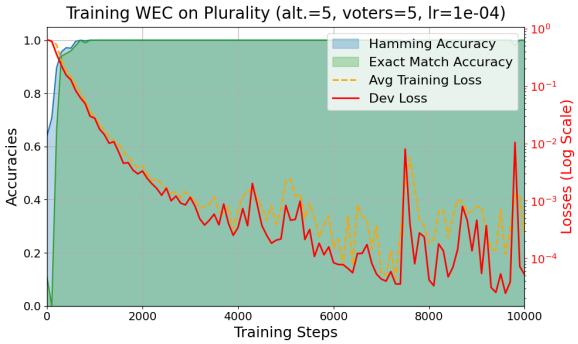
(a) Borda Learning Curve 5×3



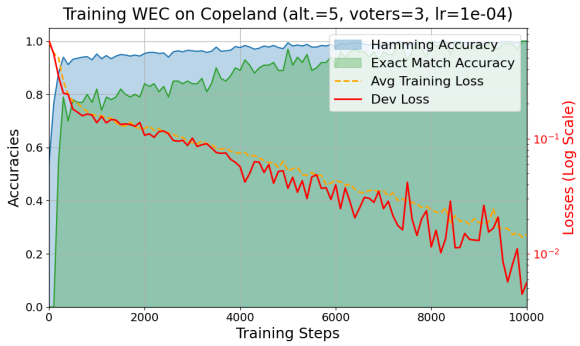
(b) Borda Learning Curve 5×5



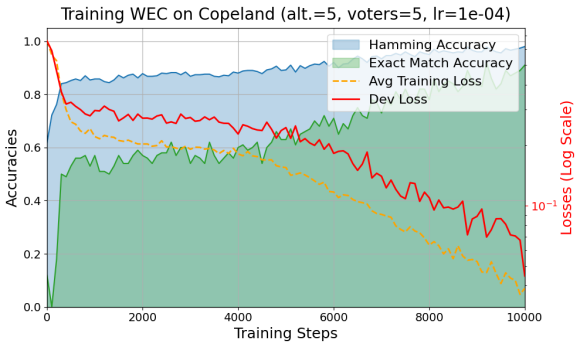
(c) Plurality Learning Curve 5×3



(d) Plurality Learning Curve 5×5



(e) Copeland Learning Curve 5×3



(f) Copeland Learning Curve 5×5

Figure 5.5: **Learning Curves for 5×3 and 5×5:** The WEC model is trained for 10,000 steps to approximate three classical voting rules for the 5×3 and 5×5 configurations. The easiest voting rule for the model to learn is Plurality, then is Borda, and the hardest one is Copeland. The difference is more noticeable for the higher maximum number of voters.

5.2 Exp2: Axiom-Optimized Training (Single Axiom)

This experiment investigates whether the WEC model trained to mimic a classical voting rule can be guided to satisfy a fairness principle, using an axiomatic objective. A model pretrained on the Borda classical voting rule (see Section 5.1) is further trained in an unsupervised way.

5.2.1 Objective

The goal is to determine whether a model trained on a classical voting rule can adapt to satisfy a single voting axiom (like Condorcet or Independence), while retaining the structure it learned during the supervised training phase. The expectation is that initializing the model with a classical voting rule will improve its ability to learn a principled solution during axiom-guided training.

5.2.2 Experimental Setup

For the optimization, continuous forms of the Condorcet and Independence axioms are used, as explained in Section 4.4. An importance weight of 1 is used for both axioms. The WEC model is initialized either with a random state or with a Borda-pretrained state from the previous experiment. It is optimized so that it minimizes the KLD loss defined in Section 3.4. The KLD is calculated on a prediction level for the Condorcet axiom and on the logits level (before the arbitrary output logits are used to make a prediction) for the Independence axiom.

The same dataset splits, configurations, and model parameters are used as in Section 5.1. The training is conducted on 1,000 steps, except for individual cases in which a longer training of 5,000 steps is performed to investigate how the learning develops. A batch size of 32 and a learning rate of 1e-4 are used. The random seed is set to 42. During evaluation, admissibility, axiom satisfaction, and rule similarity to classical voting rules are measured, as described in Section 4.6.

5.2.3 Results

From the results in Tables 5.3 and 5.4 and the training logs, we track that, while learning each of the axioms, the similarity to Borda decreases over time, starting from 100%.

Learning Condorcet. As seen in the learning curves in Figure 5.6, the WEC model improves its satisfaction of the Condorcet axiom during training. It succeeds at learning Condorcet fully for the 3×3 setup, both from a randomly initialized state and from a Borda checkpoint (see Table 5.3). The model learns an above 99% Condorcet-compliant rule for 3 alternatives and 5 voters when initialized randomly, but when initialized with a Borda checkpoint, it reaches only 88.75%. With a longer training of 5,000 steps (the last 2 columns in Table 5.3), both model initializations lead to above 99% Condorcet-compliant rules. The learning curves in Figure 5.7 show a noisier loss function for the Borda-initialized model. In Table 5.3 we see that the models that were pretrained on Borda are more similar to the Borda rule than the models initialized with a random state. The learned rules that have a high compliance with the Condorcet axiom are more similar to Black's rule (Black, 1958) than to the Borda rule. This is interesting because the Black's rule is a compromise between the Condorcet method and the Borda count - the Condorcet winner is selected if it exists, and the candidate with the highest Borda score is selected if it doesn't. At the **B3x5** column, the model is still more similar to Borda (92.28%) than Black's (84.64%). After longer training (the **B3x5-L** column), the model starts behaving closer to Black's (92.83%) than to Borda (83.69%). The model doesn't learn Black's at 100% in any of the setups, initializations, or training lengths that were performed.

Learning Independence. Figure 5.8 shows that for a random initialization of the WEC model, at the start of the training, Independence is satisfied at 100% (for the dev set), although the loss is not exactly 0. It is further minimized over time and reaches 0. When it does, we see that the loss line disappears from the plot as $log(0)$ is undefined. We notice spikes in the dev loss line as the model parameters update, and a nonzero dev loss is recorded. On the other hand, the Borda pretrained WEC model does not comply with independence at the beginning of the training, but it quickly learns it. The rule similarity evaluation in Table 5.4 shows that a model initialized from Borda learns voting rules that align more closely with all classical voting rules than a model trained from a random initialization. Although the model is not trained on admissibility, all of the model's predicted winners are admissible except for the 3×5 configuration initialized from scratch in Figure 5.8, where admissibility drops towards the end of the training.

5.2.4 Discussion

This experiment shows that a semantic loss can guide a model toward satisfying individual axioms, both when initialized from a random state and from a state that mimics a classical voting rule. When initialized with Borda, the model has to learn to respect an axiom, given the prior learned structure. This causes noisier updates, however, some of the initial "good" outcomes are retained.

From learning Condorcet and the similarity measures between the learned rules and classical voting rules like Borda and Black's, we conclude that further guidance or constraints should be used to keep the model aligned with the initial state (forcing the model to keep its predictions for cases when an axiom is not applicable) while updating it to satisfy Condorcet. In the ideal case, if we started training from a Borda checkpoint and trained the model on the Condorcet axiom, we should have learned a voting rule that is 100% similar to Black's.

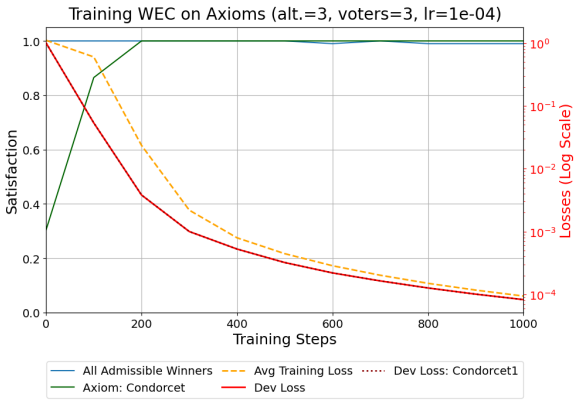
From learning independence, we see that learning voting rules that have high similarity to classical voting rules is desired when an axiom is easily learned. There are many easy but "bad" ways in which the independence axiom can be satisfied. The model learns to respect one of them (usually following the simplest way to do this given the definition of the semantic loss). While initializing the model with Borda is not enough on its own to keep the good fairness properties throughout training, the results from the model pretrained on Borda are preferred, as the learned rules are significantly more similar to all classical voting rules.

Metric	R3x3	B3x3	R3x5	B3x5	R3x5-L	B3x5-L
Max # Voters	3	3	5	5	5	5
Max # Alternatives	3	3	3	3	3	3
All admissible winner	0.98	1.00	0.99	1.00	0.99	1.00
Training Time (min)	0.55	1.12	0.52	0.50	3.15	2.80
Evaluation Time (min)	0.01	0.01	2.36	0.35	0.40	0.37
Axiom: Condorcet	100.00	100.00	99.36	88.75	99.98	99.80
Rule Similarity						
Anti-Plurality	54.11	56.20	49.15	56.86	49.26	51.87
Baldwin	89.58	93.75	89.83	78.65	89.70	88.55
Banks	89.58	93.75	88.11	77.88	87.66	87.47
Blacks	93.75	97.92	90.61	84.64	90.98	92.83
Borda	87.02	91.19	81.48	92.28	81.84	83.69
Coombs	93.75	95.83	89.08	82.60	89.45	91.68
Copeland	93.75	97.92	90.61	82.22	90.98	91.29
Instant Runoff TB	84.72	86.11	90.68	76.94	92.02	88.49
Kemeny-Young	89.58	93.75	89.83	78.65	89.06	88.55
Llull	89.58	93.75	88.87	78.65	88.42	88.23
Plurality	78.69	78.69	67.14	66.68	65.93	65.85
Plurality With Runoff PUT	78.69	78.69	88.96	75.73	88.76	86.64
Stable Voting	93.75	97.92	91.57	82.73	91.61	91.87
Top Cycle	89.58	93.75	87.19	77.88	87.15	87.21
Uncovered Set	89.58	93.75	88.11	77.88	87.66	87.47
Weak Nanson	93.75	97.92	90.61	84.13	90.98	92.57

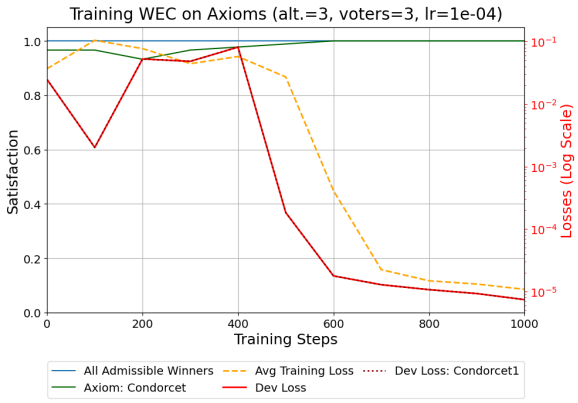
Table 5.3: **Learning Condorcet Results:** Final evaluation of the WEC model trained on the Condorcet axiom. Columns labeled with **R** indicate training initialized with a random state, while **B** indicates training initialized with a model pretrained on Borda. The second part of the column label denotes the maximum alternatives by the maximum voters used in the setup. The suffix **L** indicates a prolonged training of 5,000 steps instead of 1,000. The training and evaluation times are reported. The time for dataset generation is excluded because it is performed only the first time a given dataset configuration with a number of profiles is used.

Metric	R3x3	B3x3	R3x5	B3x5
Max # Voters	3	3	5	5
Max # Alternatives	3	3	3	3
All admissible winner	0.94	1.00	0.99	0.99
Training Time (min)	0.59	0.64	1.21	1.47
Evaluation Time (min)	0.03	0.03	4.65	4.87
Axiom: Independence	100.00	100.00	100.00	100.00
Rule Similarity				
Anti-Plurality	4.86	22.57	9.50	23.43
Baldwin	2.08	28.82	2.04	29.47
Banks	2.08	28.82	2.30	28.58
Blacks	0.69	30.21	0.60	30.92
Borda	2.78	28.12	3.42	28.08
Coombs	0.69	30.90	1.62	31.52
Copeland	0.69	30.21	0.85	30.03
Instant Runoff TB	0.00	24.65	0.00	36.41
Kemeny-Young	2.08	28.82	2.68	28.84
Llull	2.08	28.82	2.04	28.84
Plurality	2.78	24.65	11.31	21.69
Plurality With Runoff PUT	6.94	24.65	2.25	30.37
Stable Voting	0.69	30.21	1.23	30.28
Top Cycle	0.69	28.82	0.85	28.58
Uncovered Set	2.08	28.82	2.30	28.58
Weak Nanson	0.69	30.21	0.85	30.67

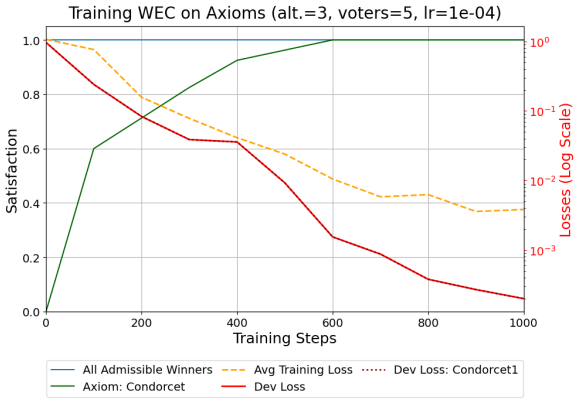
Table 5.4: **Learning Independence Results:** Final evaluation of the WEC model trained on the Independence axiom. Columns labeled with **R** indicate training initialized with a random state, while **B** indicates training initialized with a model pretrained on Borda. The second part of the column label denotes the maximum alternatives by the maximum voters used in the setup. The training and evaluation times are reported. The time for dataset generation is excluded because it is performed only the first time a given dataset configuration with a number of profiles is used.



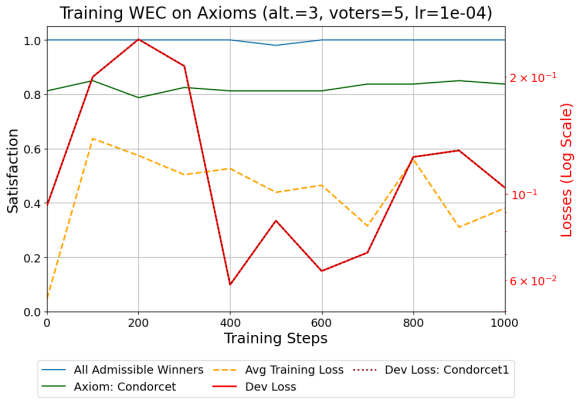
(a) 3×3 Condorcet Learning, Initialized with a Random State



(b) 3×3 Condorcet Axiom Satisfaction, Initialized with a Borda State

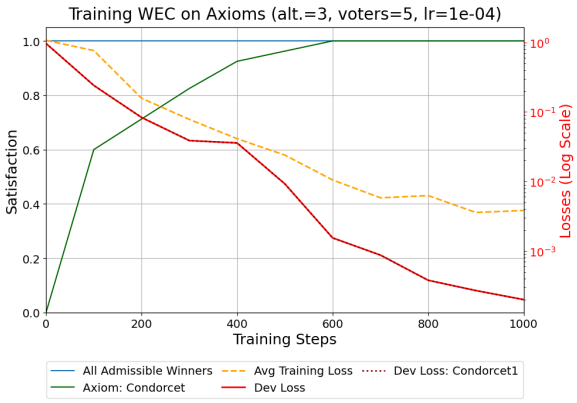


(c) 3×5 Condorcet Learning, Initialized with a Random State

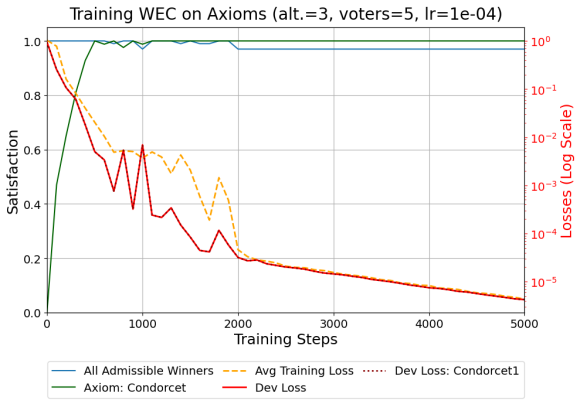


(d) 3×5 Condorcet Learning, Initialized with a Borda State

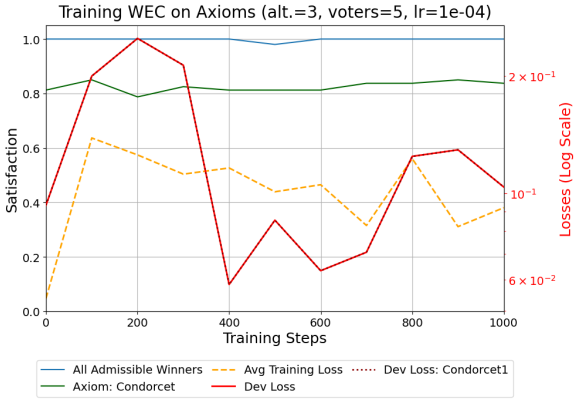
Figure 5.6: **Learning Curves Condorcet:** Learning Condorcet with randomly initialized models: a) and c), and models pretrained on Borda: b) and d).



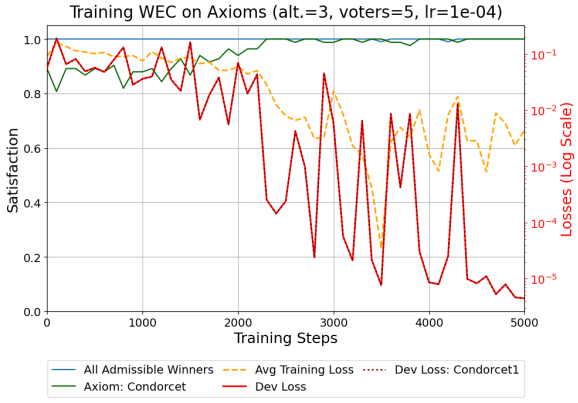
(a) 3×5 Condorcet Learning for 1,000 Steps, Initialized with a Random State



(b) 3×5 Condorcet Learning for 5,000 Steps, Initialized with a Random State

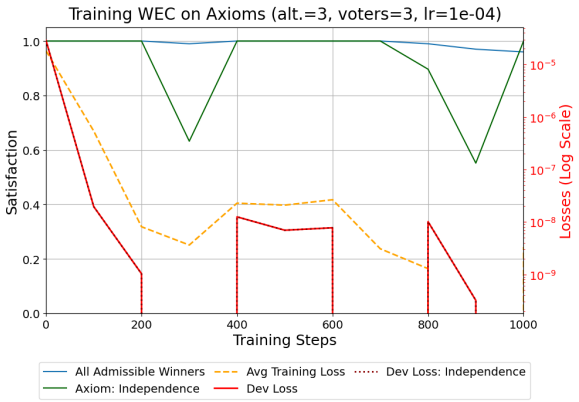


(c) 3×5 Condorcet Learning for 1,000 Steps, Initialized with a Borda State

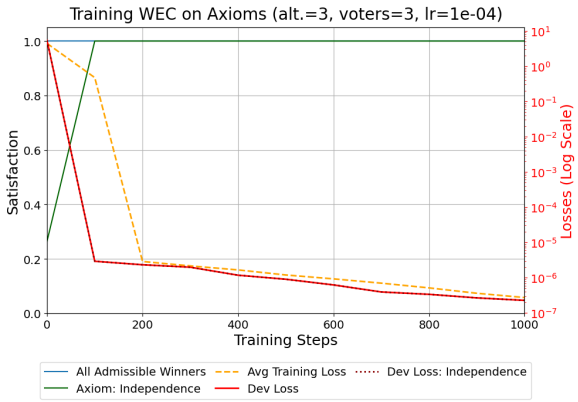


(d) 3×5 Condorcet Learning Trained for 5,000 Steps, Initialized with a Borda State

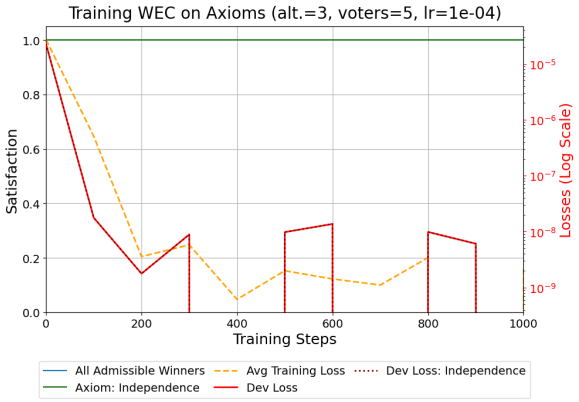
Figure 5.7: **Condorcet Learning Curves Prolonged:** Learning Condorcet for the 3×5 configuration from the random and the Borda initialization seen at Figure 5.6 b) and d), and the learning curves after a prolonged training for 5,000 steps.



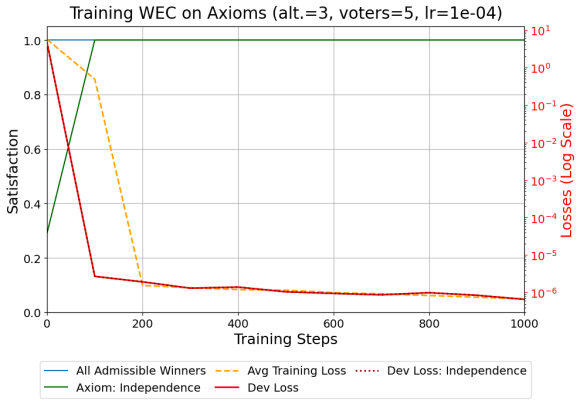
(a) 3×3 Independence Learning from a Random State



(b) 3×3 Independence Learning from a Borda State



(c) 3×5 Independence Learning from a Random State



(d) 3×5 Independence Learning from a Borda State

Figure 5.8: **Learning Curves Independence:** Learning Independence with randomly initialized models: a) and c), and models pretrained on Borda: b) and d).

5.3 Exp3: Axiom-Optimized Training (Multiple Axioms)

This experiment extends the approach described in Section 5.2 by optimizing for multiple axioms simultaneously. The goal is to evaluate whether a model can balance several fairness principles when guided by a combined semantic loss. Rather than prioritizing a single axiom, this setup encourages the model to satisfy more than one constraint at the same time, reflecting the reality that voting rules are judged by multiple normative criteria.

5.3.1 Objective

The experiment investigates whether combining axioms into a joint objective leads to a desired fairness behavior or to trade-offs between competing principles. The goal is to observe whether the model can learn to satisfy both axioms simultaneously, whether initializing the model with a Borda checkpoint helps the learning stability, and how it affects the learned rule's similarity to classical voting rules.

5.3.2 Experimental Setup

The experimental setup is the same as in Section 5.2, except multiple axioms are optimized at the same time. The WEC model is initialized randomly or from a checkpoint trained to replicate the Borda rule (presented in Section 5.1). The model is trained on continuous forms of *Condorcet*, *independence*, *inadmissibility*, and *no winner*, as defined in Section 3.2. Each axiom is assigned a corresponding importance weight. The violation rates of the axioms are multiplied by the weights and averaged into a single semantic loss.

The same dataset splits, configurations, and model parameters are used as in Section 5.1. The training is conducted on 1,000 steps with a batch size of 32, a learning rate of 1e-4, and a random seed set to 42. The same evaluation metrics are used as in Section 5.2. The individual losses for each axiom are plotted to visualize how they develop over time and interpret how the aggregated semantic loss is affected by them.

5.3.3 Results

Training the model on both axioms with equal importance weights of 1, as seen in Figure 5.11 a) and c), leads to poor learning curves where the model does not learn to adhere to Condorcet as well in a) and unlearns it together with admissibility in c). In b) and d), we notice that decreasing the importance weight of independence to 0.1 leads to better outcomes.

Training the model for the 3×3 configuration for both Condorcet and Independence objectives of the randomly initialized models leads to 100% learned Condorcet, and higher independence compliance (see Table 5.5). Interestingly, **B3x3-2**, which is a Borda-initialized model, is less similar to Borda than **R3x3-2**. We notice at the **R3x3-3** column that both axioms reach 100% compliance, while admissibility is compromised to 89%. From Figure 5.9 c) and e), as well as from the training logs, we notice that the model learns to select no winner in some cases, which helps it to satisfy the independence axiom.

Training the model in the 3×5 configuration on both Condorcet and Independence objectives leads to a high satisfaction of the Condorcet axiom (but not 100%) and independence satisfaction between 26% and 33%. All models trained from a random state have a bit higher Condorcet satisfaction, while all models trained from a Borda checkpoint have a bit higher independence satisfaction. The Borda initialized models are more similar to Borda than the ones initialized from a random state.

5.3.4 Discussion

This experiment reveals difficulties in balancing learning objectives. An axiom like independence can easily change the model's parameters, unlearning the initial state or introducing a stronger signal that makes the model disrespect other axioms. Setting a lower importance weight in this case can help the model focus on the other objectives.

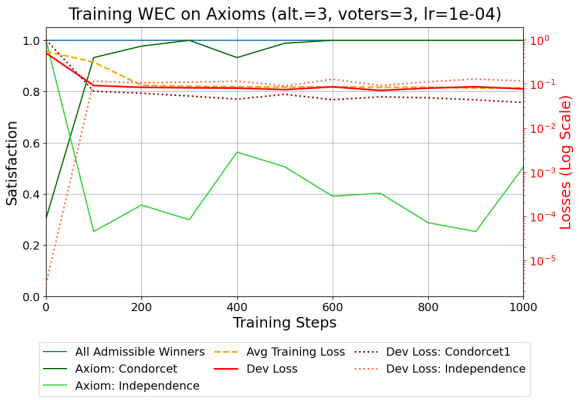
Random initialization could lead to faster convergence of the training, while pretraining a model could have more fluctuations of the learning curve and take more time. However, a Borda-pretrained model could be more likely to avoid shortcuts that the model can use to satisfy axioms (for example, selecting no winner). A direction for future research is finding other ways to interpret a learned voting rule like **R3x3-3** from Table 5.5.

Metric	**R3x3-2**	**B3x3-2**	**R3x3-3**	**B3x3-3**	**R3x3-4**	**B3x3-4**
Max # Voters	3	3	3	3	3	3
Max # Alternatives	3	3	3	3	3	3
All admissible winner	1.00	1.00	0.89	1.00	1.00	1.00
Training Time (min)	1.06	0.83	0.95	0.86	1.09	1.15
Evaluation Time (min)	0.05	0.03	0.03	0.03	0.04	0.05
Axiom: Condorcet	100.00	69.57	100.00	97.14	100.00	97.14
Axiom: Independence	44.07	32.75	100.00	30.68	43.11	30.68
Rule Similarity						
Anti-Plurality	51.33	46.71	47.86	48.56	49.95	48.56
Baldwin	90.28	64.77	86.81	87.72	88.89	89.80
Banks	90.28	64.77	86.81	87.72	88.89	89.80
Blacks	90.97	64.77	87.50	87.72	89.58	87.72
Borda	84.25	71.49	80.77	86.11	82.86	86.11
Coombs	90.97	62.68	87.50	85.64	89.58	85.64
Copeland	90.97	64.77	87.50	87.72	89.58	87.72
Instant Runoff TB	85.42	59.21	83.33	80.77	86.11	80.77
Kemeny-Young	90.28	64.77	86.81	87.72	88.89	89.80
Llull	90.28	64.77	86.81	87.72	88.89	89.80
Plurality	77.30	60.36	73.83	72.66	75.91	74.74
Plurality With Runoff PUT	77.30	62.68	73.83	72.66	75.91	74.74
Stable Voting	90.97	64.77	87.50	87.72	89.58	87.72
Top Cycle	88.89	68.93	86.81	91.89	87.50	89.80
Uncovered Set	90.28	64.77	86.81	87.72	88.89	89.80
Weak Nanson	90.97	64.77	87.50	87.72	89.58	87.72

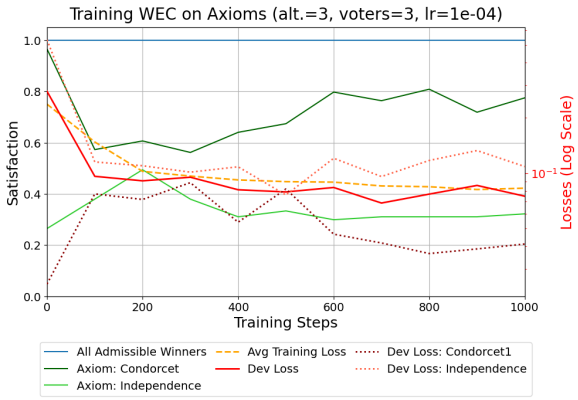
Table 5.5: **Learning Multiple Axioms 3×3 Results:** Final evaluation of the WEC model trained on the Condorcet and independence axioms. Columns labeled with **R** indicate training initialized with a random state, while **B** indicates training initialized with a model pretrained on Borda. Columns followed by the **-2** suffix are only trained on the mentioned 2 axioms. **-3** means that an objective penalizing inadmissible winners is added to the training. **-4** means that a penalty for selecting no winner is added to the training. The used weights are 1×Condorcet, 0.1×independence, 1×inadmissibility and 1×no winner. The training and evaluation times are reported. The time for dataset generation is excluded because it is performed only the first time a given dataset configuration with a number of profiles is used.

Metric	R3x5-2	B3x5-2	R3x5-3	B3x5-3	R3x5-4	B3x5-4
Max # Voters	5	5	5	5	5	5
Max # Alternatives	3	3	3	3	3	3
All admissible winner	1.00	1.00	1.00	1.00	1.00	1.00
Training Time (min)	2.93	2.21	1.85	1.60	1.78	1.69
Evaluation Time (min)	8.05	6.21	5.43	5.05	6.82	6.58
Axiom: Condorcet	97.81	83.16	99.37	84.39	98.75	86.56
Axiom: Independence	28.59	32.22	27.03	32.06	26.55	29.30
Rule Similarity						
Anti-Plurality	51.52	55.78	51.00	56.45	50.95	55.96
Baldwin	92.45	80.35	92.33	80.69	90.72	82.74
Banks	91.69	80.10	91.56	80.44	89.96	82.23
Blacks	89.88	81.33	89.60	82.37	89.08	84.72
Borda	82.79	89.82	80.64	90.86	80.78	93.21
Coombs	90.11	77.25	91.71	78.16	91.06	80.51
Copeland	90.33	80.82	90.37	81.29	89.53	83.64
Instant Runoff TB	83.78	70.53	85.23	71.51	84.70	73.26
Kemeny-Young	94.05	80.99	94.24	81.01	92.32	83.06
Llull	92.45	80.35	92.33	80.69	90.72	82.74
Plurality	66.37	60.96	66.71	62.07	66.12	63.29
Plurality With Runoff PUT	87.35	74.48	89.35	74.50	87.81	76.24
Stable Voting	91.16	80.69	91.52	81.10	90.36	83.44
Top Cycle	88.84	78.48	88.80	78.69	87.57	80.77
Uncovered Set	91.69	79.84	91.56	80.18	89.96	82.10
Weak Nanson	90.65	82.09	90.37	82.88	89.85	85.23

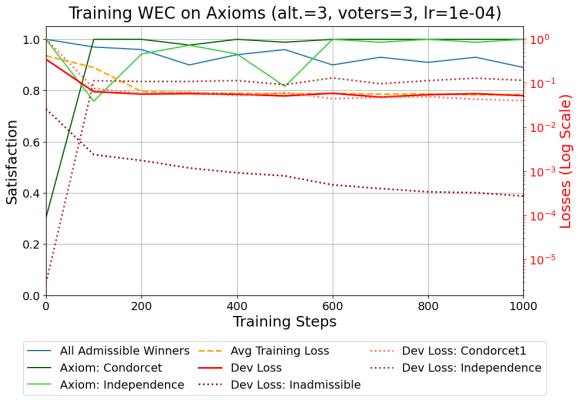
Table 5.6: **Learning Multiple Axioms 3×5 Results:** Final evaluation of the WEC model trained on the Condorcet and independence axioms. Columns labeled with **R** indicate training initialized with a random state, while **B** indicates training initialized with a model pretrained on Borda. Columns followed by the **-2** suffix are only trained on the mentioned 2 axioms. **-3** means that an objective penalizing inadmissible winners is added to the training. **-4** means that a penalty for selecting no winner is added to the training. The used weights are 1×Condorcet, 0.1×independence, 1×inadmissibility and 1×no winner. The training and evaluation times are reported. The time for dataset generation is excluded because it is performed only the first time a given dataset configuration with a number of profiles is used.



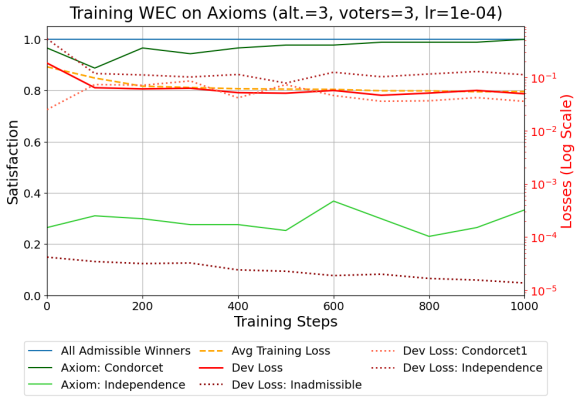
(a) 1×Condorcet and 0.1×Independence from a Random State



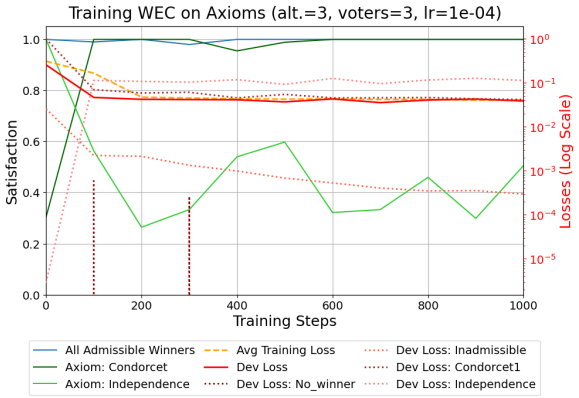
(b) 1×Condorcet and 0.1×Independence from a Borda State



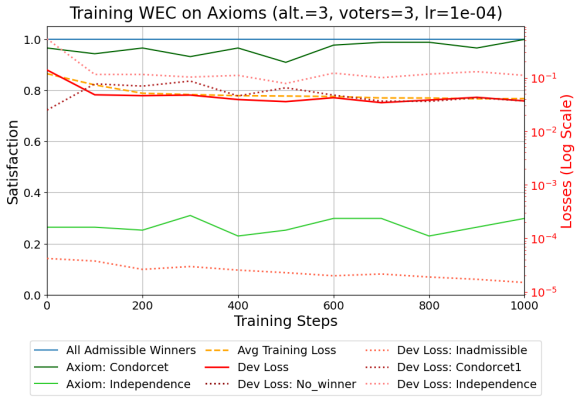
(c) 1×Condorcet, 1×Inadmissible and 0.1×Independence from a Random State



(d) 1×Condorcet, 1×Inadmissible and 0.1×Independence from a Borda State

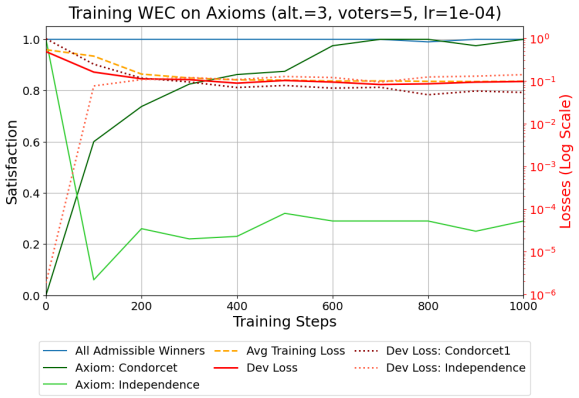


(e) 1×No Winner, 1×Condorcet, 1×Inadmissible and 0.1×Independence from a Random State

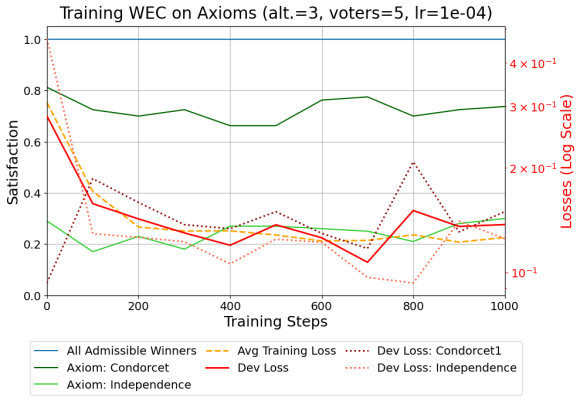


(f) 1×No Winner, 1×Condorcet, 1×Inadmissible and 0.1×Independence from a Borda State

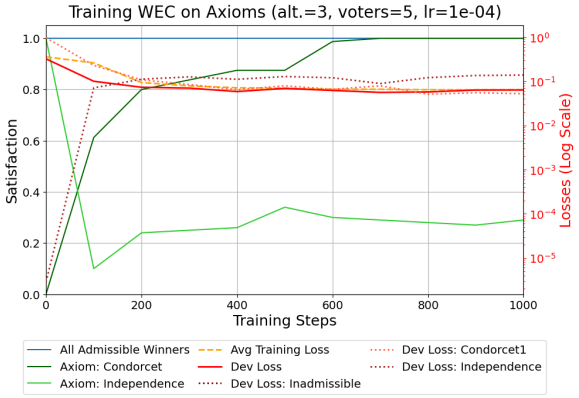
Figure 5.9: **Learning Multiple Axioms 3×3:** We observe that for 1,000 training steps, the training from a random state reaches 100% Condorcet satisfaction on the dev set. The Condorcet loss decreases, while the independence violations get bigger. In c), we observe that the model learns to respect independence better; however, this compromises admissibility. From Table 5.5, we see that the model starts to avoid selecting a winner. In e), this behavior is penalized by an additional no-winner loss (the 2 loss spikes at the 100th and 300th step). We observe that the admissibility is not compromised; however, the independence satisfaction stays low. The training initialized from a Borda-trained model maintains the axiom satisfaction rates of Condorcet and Borda in d) and e). In b), we observe some fluctuations; however, the tendency is for the training to return to the initial balance.



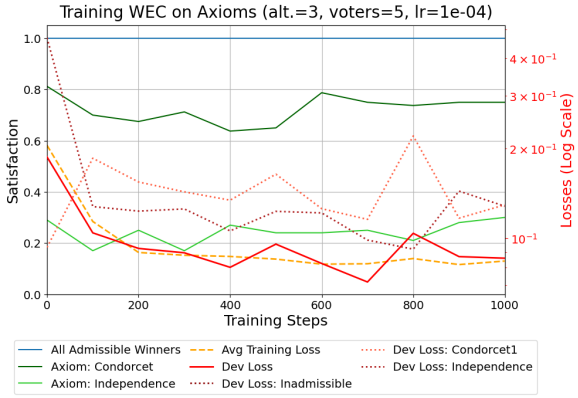
(a) 1×Condorcet and 0.1×Independence from a Random State



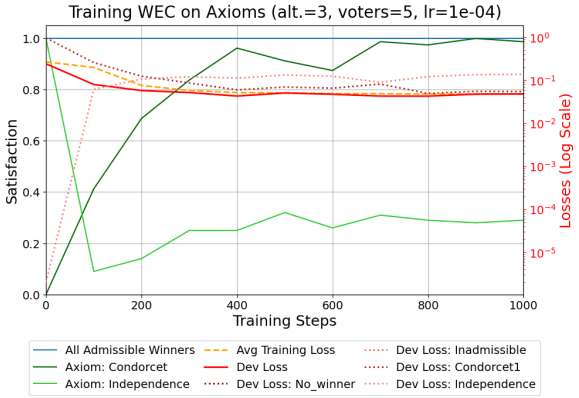
(b) 1×Condorcet and 0.1×Independence from a Borda State



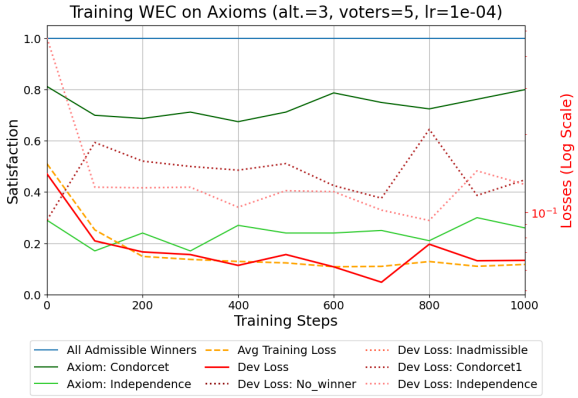
(c) 1×Inamissible, 1×Condorcet and 0.1×Independence from a Random State



(d) 1×Inamissible, 1×Condorcet and 0.1×Independence from a Borda State

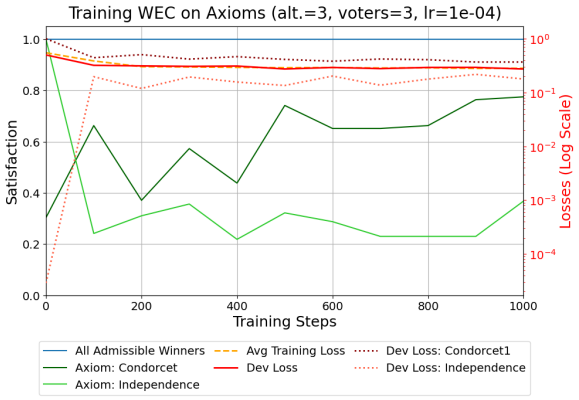


(e) 1×No Winner, 1×Inamissible, 1×Condorcet and 0.1×Independence from a Random State

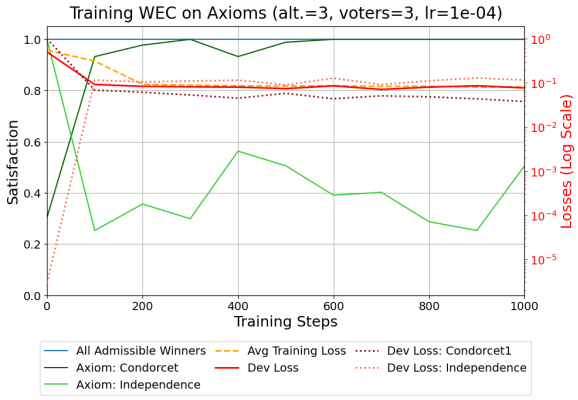


(f) 1×No Winner, 1×Inamissible, 1×Condorcet and 0.1×Independence from a Borda State

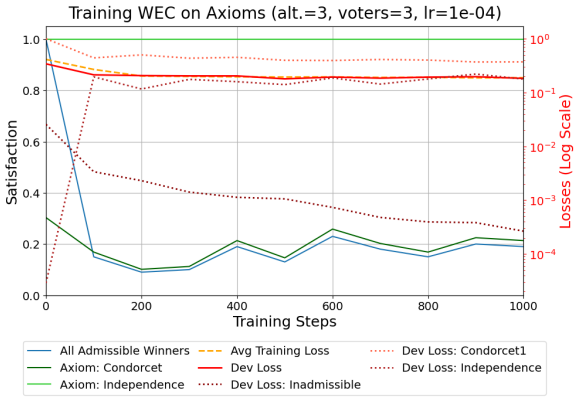
Figure 5.10: **Learning Multiple Axioms 3×5:** Here we show the learning curves of axiom-based training of multiple axioms starting from randomly initialized models: a), c), and e, and from Borda-pretrained models b), d), and f). We observe that, after 1,000 training steps, training from a random state achieves 100% Condorcet satisfaction on the dev set. The Condorcet loss decreases, while the independence violations get bigger. For the 1,000 steps that we recorded and the chosen learning rate, the training that is initialized with a Borda checkpoint did not converge.



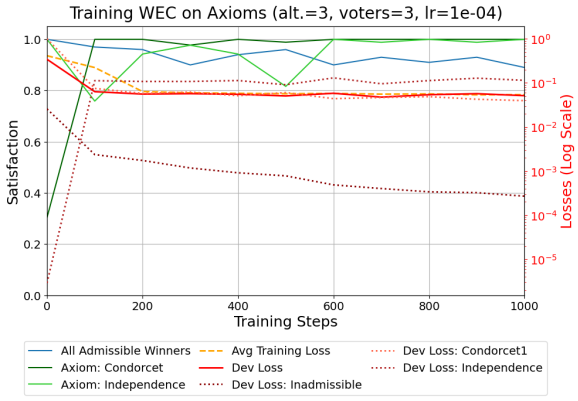
(a) 1×Condorcet and 1×Independence from a Random State



(b) 1×Condorcet and 0.1×Independence from a Random State



(c) 1×Condorcet, 1×Inadmissible and 1×Independence from a Random State



(d) 1×Condorcet, 1×Inadmissible and 0.1×Independence from a Random State

Figure 5.11: **Balancing Axioms:** Here, we show why we chose to assign a smaller weight to Independence. In a) and c), Independence is assigned a weight of 1 (equal to the weight of the other axioms), while in b) and d), it is assigned a smaller weight of 0.1. We notice worse performance of Condorcet in a) and c), and admissibility dropping in c).

Chapter 6

Conclusions

This thesis explored the use of neural models for learning voting rules. Three settings were investigated: supervised learning of classical rules (Experiment 1), axiom-optimized training with a single semantic loss (Experiment 2), and axiom-optimized training with multiple losses (Experiment 3). The findings highlight both the potential and the challenges of using neural models for axiomatic learning.

Experiment 1 showed that models can imitate classical rules such as Borda, Plurality, and Copeland when labels are provided. This confirmed that the WEC architecture is able to represent well-known voting rules and compared it with other architectures. The difficulty varied by rule: Plurality was the easiest for the models to learn, Borda was harder, and Copeland was the most difficult.

Training with a single axiom in Experiment 2 enforced the selected property. Condorcet optimization guided the model toward behavior that is more similar to (but not exactly) Black's rule, while Independence optimization produced outcomes less aligned with classical voting rules. Initialization mattered: initializing the model from a Borda checkpoint slowed down axiom learning but preserved some of the behavior of the original voting rule.

Optimizing Condorcet and Independence together in Experiment 3 introduced competing objectives and sometimes led to outcomes such as the loss of Condorcet consistency, violations of admissibility, or no-winner predictions. Adjusting the loss weights improved stability, but achieving both properties at once in a meaningful way was not possible.

In summary, semantic losses provide a way to guide neural models toward satisfying axioms, but they also face the trade-offs highlighted by impossibility theorems in social choice. Incorporating additional losses with carefully chosen importance weights can prevent trivial solutions, while initialization can help models remain closer to classical voting rules. Future work could examine how to combine semantic losses with supervised objectives to preserve the behavior of pretrained models during axiom optimization, and how to develop new methods for interpreting the learned rules once they diverge from classical ones.

Bibliography

Ben Armstrong and Kate Larson. Machine learning to strengthen democracy. In *NeurIPS Joint Workshop on AI for Social Good*, 2019. URL https://aiforsocialgood.github.io/neurips2019/accepted/track1/pdfs/69_aisg_neurips2019.pdf.

Kenneth Joseph Arrow. *Social Choice and Individual Values*. Number 12 in Cowles Commission for Research in Economics. John Wiley and Sons, New York and London, 1951.

Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198. Association for Computational Linguistics, July 2020. doi: https://doi.org/10.18653/v1/2020.acl-main.463. URL https://aclanthology.org/2020.acl-main.463.

Duncan Black. *The Theory of Committees and Elections*. Cambridge University Press, Cambridge, 1958.

Carter Blair, Ben Armstrong, and Kate Larson. Liquid ensemble selection for continual learning. *arXiv preprint arXiv:2405.07327*, 2024.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Felix Brandt, Markus Brill, and Paul Harrenstein. Tournament solutions. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Editors Procaccia, editors, *Handbook of Computational Social Choice*, chapter 3, pages 57–84. Cambridge University Press, 2016a. doi: https://doi.org/10.1017/CBO9781107446984.

Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. Introduction to computational social choice. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Editors Procaccia, editors, *Handbook of Computational Social Choice*, chapter 1, pages 1–20. Cambridge University Press, 2016b. doi: https://doi.org/10.1017/CBO9781107446984.

Dávid Burka, Clemens Puppe, László Szepesváry, and Attila Tasnádi. Neural networks would 'vote' according to Borda's rule. *KIT Working Paper Series in Economics*, 96, 2016. ISSN 2190-9806. doi: https://doi.org/10.5445/IR/1000062014.

Dávid Burka, Clemens Puppe, László Szepesváry, and Attila Tasnádi. Voting: A machine learning approach. *European Journal of Operational Research*, 299(3):1003–1017, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.10.005. URL https://www.sciencedirect.com/science/article/pii/S037722172100850X.

Vincent Conitzer, Rachel Freedman, Jobst Heitzig, Wesley H. Holliday, Bob M. Jacobs, Nathan Lambert, Milan Mossé, Eric Pacuit, Stuart Russell, Hailey Schoelkopf, Emanuel Tewolde, and William S. Zwicker. Social choice for AI alignment: Dealing with diverse human feedback. In *Proceedings of the Workshop on Social Choice and Learning Algorithms (SCaLA-24)*, Auckland, New Zealand, May 2024. URL https://sites.google.com/view/scala24.

Luise Ge, Brendan Juba, and Yevgeniy Vorobeychik. Learning linear utility functions from pairwise preference queries. In *Proceedings of the Workshop on Social Choice and Learning Algorithms (SCaLA-24)*, Auckland, New Zealand, May 2024. URL https://sites.google.com/view/scala24.

Christian Geist and Ulle Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research*, 40:143–174, 2011. URL https://eprints.illc.uva.nl/id/eprint/408/1/PP-2011-04.text.pdf.

Christian Geist and Dominik Peters. Computer-aided methods for social choice theory. In Ulle Endriss, editor, *Trends in Computational Social Choice*, chapter 13, pages 249–267. AI Access, 2017. URL http://www.illc.uva.nl/COST-IC1205/Book/.

Allan Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973.

Umberto Grandi. Social choice and social networks. In Ulle Endriss, editor, *Trends in Computational Social Choice*, chapter 9, pages 169–184. AI Access, 2017. URL http://www.illc.uva.nl/COST-IC1205/Book.

Michael Guerzhoy. Occam's razor and Bender and Koller's octopus. In Sana Al-azzawi, Laura Biester, György Kovács, Ana Marasović, Leena Mathur, Margot Mieskes, and Leonie Weissweiler, editors, *Proceedings of the Sixth Workshop on Teaching NLP*, pages 128–129, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.teachingnlp-1.18/.

Wesley H. Holliday, Alexander Kristoffersen, and Eric Pacuit. Learning to manipulate under limited information. In *Proceedings of the 39th Annual AAAI Conference on Artificial Intelligence (AAAI-25)*, 2025.

Levin Hornischer and Zoi Terzopoulou. Learning how to vote with principles: Axiomatic insights into the collective decisions of neural networks. *Journal of Artificial Intelligence Research*, 83, 2025. doi: 10.1613/jair.1.18890.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Hanna Kujawska, Marija Slavkovik, and Jan-Joachim Rückmann. Predicting the winners of Borda, Kemeny and Dodgson elections with supervised machine learning. *Multi-Agent Systems and Agreement Technologies*, pages 440–458, 2020. doi: https://doi.org/10.1007/978-3-030-66412-1_28.

Marc Lanctot, Kate Larson, Michael Kaisers, Quentin Berthet, Ian Gemp, Manfred Diaz, Roberto-Rafael Maura-Rivero, Yoram Bachrach, Anna Koop, and Doina Precup. Soft Condorcet optimization for ranking of general agents. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), A. El Fallah Seghrouchni, Y. Vorobeychik, S. Das, A. Nowe (eds.)*, pages 1253–1262, Detroit, Michigan, USA, May 2025. International Foundation for Autonomous Agents and Multiagent Systems. URL https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p1253.pdf.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, December 1998. doi: https://doi.org/10.1109/5.726791.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019. URL `https://arxiv.org/abs/1810.00825`.

Grzegorz Lewicki and Giuseppe Marino. Approximation by superpositions of a sigmoidal function. *Applied Mathematics Letters*, 17(10):1147–1152, December 2004. doi: https://doi.org/10.1016/j.aml.2003.11.006.

Nicholas Mattei and Toby Walsh. Preflib: A library for preferences. In *Algorithmic Decision Theory: Third International Conference, ADT 2013*, pages 259–270. Springer, November 2013.

Kenneth O. May. A set of independent necessary and sufficient conditions for simple majority decision. *Econometrica*, 20(4):680–684, 1952. doi: https://doi.org/10.2307/1907651.

Oliviero Nardi, Arthur Boixel, and Ulle Endriss. A graph-based algorithm for the automated justification of collective decisions. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 935–943. IFAAMAS, 2022. URL `https://ifaamas.org/Proceedings/aamas2022/pdfs/p935.pdf`.

Florenz Plassmann and T. Nicolaus Tideman. How frequently do different voting rules encounter voting paradoxes in three-candidate elections? *Social Choice and Welfare*, 42:31–72, 2014. doi: https://doi.org/10.1007/s00355-013-0720-8.

Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Mark A. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782. doi: https://doi.org/10.1145/1968.1972.

Leslie G. Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. Basic Books, New York, NY, 2013.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Andreas Krause Jennifer Dy, editor, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, July 2018. URL `https://proceedings.mlr.press/v80/xu18h.html`.

William S. Zwicker. Introduction to the theory of voting. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Editors Procaccia, editors, *Handbook of Computational Social Choice*, chapter 2, pages 23–56. Cambridge University Press, 2016. doi: https://doi.org/10.1017/CBO9781107446984.