Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Taking an Electoral Photograph with Neural Networks

Semester Project

Pascal Sommer

pasommer@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Peter Belcák, Andrei Constantinescu
Prof. Dr. Roger Wattenhofer

August 25, 2022

Abstract

We design and evaluate different methods for finding an embedding of voters and candidates in low-dimensional spaces, where euclidean distances in the embedding respect ordered ballots filled in by the voters.

A triplet loss approach is shown not to yield perfect results, even under the addition of various modifications. We further develop ideas to leverage neural network approaches in this problem, and while they currently provide lower performance than the alternative, they might form the basis for future methods.

For all experiments we provide numerical results using a metric designed to be invariant to the problem size.

Contents

Chapter 1

Introduction

1.1 Voting Systems

Elections in democracies can only fulfil their purpose if voters feel like their opinion was taken into account. Combining preferences of a large variety of people, especially with wildly differing opinions, can be tricky. While a simple tally might provide sufficient insight if voters are only presented with two options, things get more difficult when many candidates are present.

The most common voting systems currently used in democracies around the world to elect a single winner let voters choose their preferred option among a list of candidates. Afterwards, either the candidate with the most votes directly wins the election, or a second round with only the two most popular candidates is held.

In such systems, a voter who is happy with either of a few candidates but feels strongly against one of the candidates is not necessarily incentivised to vote for their favourite. If they instead vote for the candidate with the most existing support among their "tolerated" set of candidates, they minimise the chance of their least favourite candidate winning. A consequence of this is that the election authorities do not learn the true opinion of that voter but only the result of a strategic decision.

Various alternative voting systems have been proposed to at least partially mitigate the incentives for tactical voting:

- In approval voting, voters state for each of the candidates whether they approve of them or not.
- In ranked voting, voters rank the candidates in order from most to least approval. Winners are then usually chosen through the instant-runoff process, where the least popular candidates are iteratively removed until one of the remaining candidates has majority support.

1.2 Electoral Photographs

While all of the mentioned alternatives can still incentivise some kind of tactical voting as shown by the Gibbard–Satterthwaite theorem [1, 2], note that they can sometimes yield more information per voter. Beyond enabling the choice of an election winner, this information can provide insight into the opinions of the population as a whole.

This quote by Laslier [3] explains the utility of such information.

> Even though an election's goal is to designate a winner, an election not only reveals the name of the elected candidate, it also provides a kind of official photograph of voters' preferences as expressed by the votes. Large national elections are not only mechanism for choice, they are also very specific democratic moments at which a nation learns about itself by facing a picture of itself, a picture somehow taken by the voting rule. For the democratic political system to function well, it is important that this picture not be distorted.

Technically, a voting system usually consists of two parts: a ballot format in which voters express their preferences, and a set of rules to determine the winner or set of winners given the ballots as input. Since we are only interested in obtaining such "electoral photographs", we do not care about the outcome of the election and thus only look at the provided ballots without any of the subsequent process.

It should also be noted that while the most prominent kinds of elections are political elections, similar concepts appear in other applications like distributed protocols, recommendation systems, or resource allocation. These fields could also potentially benefit from interpretable insight into the process.

One could imagine various visualisation approaches that provide such insight. Our definition of an electoral photograph is a low-dimensional arrangement of voters and candidates. Euclidean distance in this space should correlate to similarity of political views, where nearby people likely share similar views. An electoral photograph should allow the viewer to quickly determine at a glance whether a voter is more likely to vote for candidate c or candidate c' by comparing distances and seeing which of the candidates is closer to the voter.

The simplest form of this is in one dimension, for example a left-right political spectrum. Adding more dimensions can show more information, such as a political compass that contains an economic left-right and a social authoritarian-libertarian axis.

An example of a higher-dimensional embedding is the smartspider, where 8 axes are displayed in a spiderweb-like fashion. The additional axes can provide more detailed information, but the downside is that we cannot easily visualize an 8-

dimensional space where viewers could immediately compare proximities of different points. It is thus beneficial to keep the dimensionality of the electoral photograph low.

While all the examples mentioned here assign an interpretation to every axis, such interpretations are not required for the basic similarity insight provided by an electoral photograph. For the purposes of this project we will only pay attention to euclidean distances and assume the axes to be interchangeable.

1.3 Problem Statement

In this project, we attempt to find a method to generate electoral photographs given only ordinal ballots from the voters and the desired number of dimensions of the photograph. Our inputs and desired outputs can be described as follows:

Our population consists of two disjoint sets, the voters V and candidates C. Every voter $v \in V$ independently provides an ordinal ballot, i.e. a total order $\succ_v$ on C such that $c \succ_v c'$ means that voter v prefers candidate c over candidate c'. In our code we express this as a list of candidates $\texttt{ranking}_v$ where $\texttt{ranking}_v^{(i)} \succ_v \texttt{ranking}_v^{(j)} \Leftrightarrow i < j$. Both voters and candidates are identified through their zero-based indices in the code.

The desired output is a k-dimensional electoral photograph, i.e., an embedding $f : V \cup C \to \mathbb{R}^k$. The rankings by euclidean distances in this space should closely match the preference rankings expressed by the voters.

A natural metric for how closely a ranking is approximated is given by the inversion count between the two permutations. For every pair of candidates in a voter's ballot, we add one to the inversion count if the candidate ranked higher is further away from the voter in the electoral photograph than the candidate ranked lower. The inversion count therefore gives us the number of errors in the embedding as seen from one voter.

Our normalised inversion cost can be expressed as a sum of individual inversion counts, divided by a normalizing factor to make the metric independent of problem size.

$$\texttt{inv}(v) = \left|\left\{(c, c') \mid c \succ_v c' \wedge \|f(v) - f(c)\|_2 > \left\|f(v) - f(c')\right\|_2\right\}\right| \tag{1.1}$$

$$\texttt{norminv} = \frac{\sum_{v \in V} \texttt{inv}(v)}{|V| \binom{|C|}{2}} \tag{1.2}$$

A `norminv` score of zero means that we have found a perfect solution such that the behaviour of all voters can be explained by condensing their political stance to a k-vector and comparing that vector to the candidates' vectors.

A `norminv` score of higher than zero however means that we have not found an embedding that completely explains the ballots. The score is bounded by one, at which point every ballot would be exactly reversed.

When performing an analysis in a real-world situation, one might be able to gather additional data from which the electoral photograph could be derived. For example, demographic information might be available to help us classify voters into groups. Candidates might also have stated their positions on various issues and thus already placed themselves somewhere on a political spectrum. For this project though we limit ourselves to just the information from the ballots.

Our input data is generated by placing voters and candidates randomly in a space of the desired dimensionality and generating voter ballots by comparing the euclidean distances from a voter to the candidates. This means that our data always has a 0-cost solution which our algorithm should in theory be able to find. Note, however, that after this exact solution is generated, only the derived rankings and the number of dimensions are passed to the algorithm. All the other information that was generated in this process is discarded again.

Chapter 2

Related Work

Finding embeddings to satisfy voter preferences has been previously investigated in various contexts. Theoretical investigations concerning the existence of exact solutions have shown that the dimensionality has a large impact.

As mentioned in the introduction, to preserve interpretability it is desirable to not have too many dimensions in an electoral photograph. It turns out that beyond a certain threshold, the result also becomes uninteresting in a mathematical sense. Bogomolnaia and Laslier have shown [4] that if $k \geq \min(|V|, |C| - 1)$, there always exists an exact solution for any given set of ballots.

Knoblauch [5] has shown that in the one-dimensional case it is possible to both determine the existence of an exact solution and to find one if one exists in polynomial time. For two or more dimensions however, determining existence is known to be NP-hard [6].

Various methods to try to solve the problem have been presented. Busing et al.[7] generate pseudo-distances and try to minimise the badness-of-fit between these pseudo-distances and euclidean distances. A survey of Peters [6] mentions that earlier methods can suffer from degeneracy problems where a voter is at an equal distance from multiple candidates. Solutions to this degeneracy problem are discussed in Busing's PhD thesis [8].

A method for the case where the available data does not contain ranked ballots but instead approval ballots has been presented by Laslier [3]. There, a distance between candidates is derived and then, using PCA, a low-dimensional arrangement of the candidates is found.

Chapter 3

Method

The baseline method that all of our later variations are based on, and that we use as a comparison reference, is a gradient descent using a triplet loss cost function.

3.1 Baseline

Triplet loss [9] is commonly used in machine learning for applications where some data point is mostly defined through its relation to other data points. An example of this is facial recognition in a system that has to deal with faces that were not seen in the training data. The network can be trained by inputting three images at a time, where two of the images show the same person and the third image shows a different person.

A triplet always consists of an anchor, a positive input, and a negative input. The training data consists of many triplets such that the anchor is more closely related to the positive input than to the negative input, where the meaning of "related" depends on the problem being solved.

In our case we form triplets by choosing a voter v and two candidates c and c' such that $c \succ_v c'$. The triplet then contains v as an anchor, c as a positive input, and c' as negative input.

3.1.1 Implementation

We use Keras, a deep-learning framework in Python, to build up a siamese network structure and the triplet loss calculation.

The network architecture is shown in fig. 3.1. The three inputs are integer indices of the voter and the candidates that make up the current triplet. These integers get converted into k-dimensional vectors using embedding layers[1], our desired embedding $f : V \cup C \to \mathbb{R}^k$ is thus represented by these layers.

[1] See the relevant Keras API documentation [10] for more information

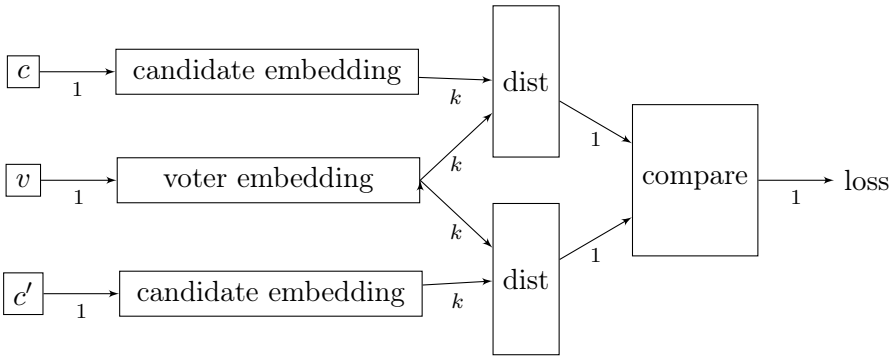

Figure 3.1: Baseline network architecture

Since any candidate might sometimes appear as a positive input and sometimes as a negative input, the parameters for the two candidate embedding layers need to be shared. This sharing is what turns this into a siamese network.

The loss is computed by comparing the euclidean distances between the voter and candidate positions. In our implementation we add a margin of 0.5 to avoid the degeneracy problem discussed in previous works where multiple candidates are collapsed to a single point.

$$\text{loss} = \max(0, \|f(v) - f(c)\|_2 - \left\|f(v) - f(c')\right\|_2 + \text{margin}) \tag{3.1}$$

3.2 Variations

3.2.1 Batching Strategies

As is the case with many machine-learning frameworks, Keras also provides built-in batching support. During training, all the inputs in a batch of the training data is fed through the network, after which the accumulated gradients are used to adjust the parameters of the network.

A popular batching strategy is to partition the training data randomly into sets of equal sizes (up to rounding issues). This is also the default strategy employed in Keras, the user only has to provide the desired batch size.

For our application, such random batching could lead to issues due to the way batching interacts with embedding layers. As a simple example, consider a network that only consists of an embedding layer with vocabulary size n and output dimension 1. This network should be able to learn a function mapping from $\{0, \ldots, n-1\}$ to $\mathbb{R}$, so we try to learn an identity function, where our training data contains every possible input exactly once. But if we now set the batch size to anything larger than 1, the network will end up mapping all inputs to the

average of the inputs (assuming quadratic loss). This is because the gradients for all inputs in the batch get accumulated and applied all at once.

To observe the influence of this effect, we compare the performance of using random batches to that of a batching strategy devised to minimise the effect. If we make sure that all triplets in one batch belong to the same voter, then no voter will be affected by gradients that were caused by a different voter.

3.2.2 Skipping Voters

Based on the observation that the ratio of voters to candidates seems to have an influence (c.f. section 3.3), it might be the case that the training process behaves differently if we do not include every voter in every epoch.

The training process is split up into 15 iterations, where at the beginning of each iteration, a random subset of voters is selected that will not participate in this iteration. The fraction of skipped voters remains constant across all iterations.

We imagine two plausible effects that could occur with such a modification:

- Letting the training process focus on only a subset of voters at a time might be beneficial, because it could allow the network to find a solution for a smaller subset first, and then using that as a basis to extend the solution to the full data set.
- Looking at the values in table 3.1, decreasing the voters to candidates ratio seems to yield worse results. It is not clear whether this effect will also be relevant in this case when we skip some of the voters each iteration, but still have a larger number of voters overall.

3.2.3 Re-initialising Voters

Since the previously listed methods usually do not achieve a perfect solution, even though we know that a zero-cost solution must exist, we infer that the process usually gets stuck in some local minimum. One idea to get out of these minima is to take the voters that currently incur a high cost and to re-initialise them randomly in a new location. This should give them a second chance to find a better location.

Similarly to the voter skipping process, we implement this process in an iterative fashion. In each of the 10 training iterations, the voters are sorted by their `inv` scores (c.f. eq. (1.1)) and the worst performers are re-initialised. This is implemented by overwriting the learned embeddings for these voters with new randomly generated coordinates. This adjusted set of embeddings is then used as the starting point for the next iteration.

We run this experiment with different re-initialisation fractions between 0 and 0.5. A high fraction might end up destroying too much of the already learned

information, therefore we expect there to be a sweet spot where this method is most useful.

3.2.4 Permutation Inverse Initialisation

While the re-initialising method focuses on getting out of local minima, we could also try to find methods that generate a better starting point for the gradient descent in the first place, and thus improve our chances of finding a zero-cost solution. Such an initialisation method would ideally look at global properties of the data because intuitively, local problems can be fixed by the gradient descent process later on, while global problems are more difficult for gradient descent to deal with.

One idea for an initialisation method is to generate initial positions for each voter based on that voter's ballot. Consider the list $\texttt{ranking}_v$ for a voter v. Trying to interpret this as a vector in some $|C|$-dimensional space would not yield very meaningful axes, because the entries in the vector are categorical values. This means that the distance between two such vectors would mostly depend on the indexing order of the candidates more than anything else.

However, if we interpret this list as a permutation $\pi_v \in S_{|C|}$, which is possible because every candidate index appear exactly once in a ranking, this opens up new possibilities. The list of integers $\texttt{invranking}_v$ that corresponds to the inverse permutation π_v^{-1} can be interpreted as a vector with much more meaningful coordinates.

The indices of this vector $\texttt{invranking}_v$ now correspond to candidate indices, and an entry in the vector represents the rank of that candidate. This means that the distance between two such vectors now has a more natural interpretation, it will be low if candidates have similar ranks for both voters.

Given these vectors, we could immediately use them as voter positions and start the training process to find corresponding candidate positions, but that would give us an electoral photograph in $|C|$ dimensions. This is usually not what we want, such a high-dimensional visualisation would be hard to interpret. Instead, we map the obtained vectors down to k dimensions using PCA, similar to the process described by Laslier et al. [3].

These k-dimensional vectors that we get from the PCA can now be used as the input, together with randomly initialised candidates, for our triplet loss training process. To ensure that the process actually uses the information provided in this initialisation, we first freeze the voter positions and only train the candidate embedding for a while. Afterwards, we unfreeze everything and continue the training as described in the normal baseline algorithm.

3.2.5 Simplex Initialisation

Another initialisation strategy is to generate the candidate positions first and then let triplet loss find suitable voter locations around the generated candidate positions.

The idea here is to first spread out the candidates evenly in a high-dimensional space. By arranging the candidates in a standard simplex in $\mathbb{R}^{|C|-1}$, we have a lot of freedom to move the voters around to satisfy their individual preferences. In this high-dimensional space, we now freeze the candidate positions and just train voter positions.

Afterwards, similarly to the process for the permutation inverse initialisation, we use PCA to map both the voters and candidates to $\mathbb{R}^k$. In this low-dimensional space, we again run the training process, now with everything unfrozen.

3.2.6 Neural Network Initialisation

Both of the previously mentioned initialisation methods use a rather simple initialisation method in a high-dimensional space and then remove excess dimensions with PCA. In this next method we try to find an approach to generate initial candidate positions that can make more use of problem-specific information. Specifically, we train a neural network to generate the candidate positions that will then be used as starting points in the baseline algorithm.

This network will take some candidate descriptors and output a vector in $\mathbb{R}^k$, where every candidate is fed through the network individually[2]. Such an initialisation should not just work for one data set, otherwise we would have to retrain the network for every data set which would defeat the purpose. Instead, we have to find a structure that generalises to new inputs.

We now have to find properties of candidates that could generalise across data sets. We define three features to be used as inputs to the network:

- A measure of centrality. Candidates who rarely appear at the bottom of ballots are most likely located in a central position. As a metric for centrality, we look at all the ranking positions in which a candidate appears and take the mean square value of them. The lower this mean square value is, the more central a candidate is likely located. Using the $\texttt{invranking}_v$ notation from section 3.2.4, the centrality measure for the candidate with index i is given as:

$$\text{centrality}(i) = \frac{1}{|V|} \sum_{v \in V} \left(\texttt{invranking}_v^{(i)} \right)^2$$

[2] In practice we generate all outputs at once by using the batching feature, but still the candidates do not influence each other.

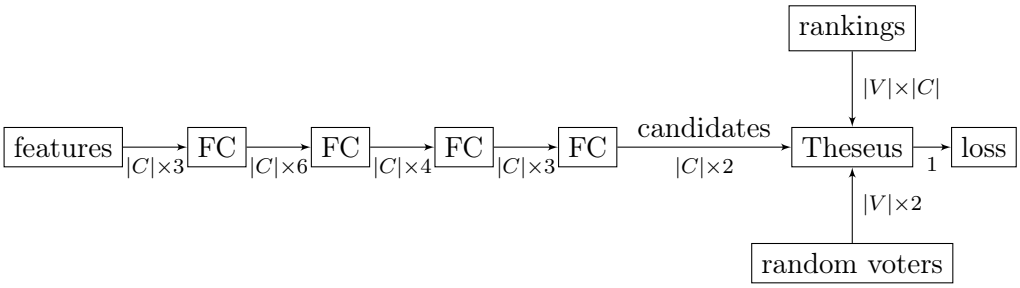

Figure 3.2: Architecture of the candidate initialisation network

- The leading fraction. This counts the fraction of voters that prefer this candidate over all others. Again, for a candidate with index i we have:

$$\text{leading}(i) = \frac{1}{|V|}\left|\left\{v \in V \mid \texttt{invranking}_v^{(i)} = 0\right\}\right|$$

- The position as seen by a mainstream voter. We first select a voter v_m that, as their favourite, chose the candidate c_m that was most often listed in the first place across all ballots.

$$v_m \in \left\{v \in V \mid \texttt{ranking}_v^{(0)} \in \arg\max_{c \in C}(\text{leading}(c))\right\}$$
$$\text{mainstream}(i) = \texttt{invranking}_{v_m}^{(i)}$$

Getting the gradients of the loss to adjust the initial candidate positions requires some modifications to our architecture, since we're not just trying to reduce the loss for this current problem but to instead adjust the initialisation network to minimise the loss for all future data sets. Essentially the goal is to differentiate the output of a nonlinear optimisation with respect to some of its parameters, for which we use the Theseus library [11] which was created specifically for this purpose.

The architecture is shown in fig. 3.2. FC denotes a fully connected layer, where the first three FC layers have a sigmoid activation function.

During training, a new data set is generated for every pass, and the non-linear optimisation in Theseus is configured to use the same loss as we did before, but to only adjust the voter positions, not the candidate positions. For the backpropagation, the gradient of the loss with respect to the generated candidate positions is computed. With this information, the earlier layers are then corrected.

3.3 Results

We start out by running the baseline algorithm (cf. section 3.1) to get a general overview of what is possible without much tweaking. Inputs are generated for different numbers of candidates, numbers of voters $|V|$, and numbers of dimensions k.

The average scores of 8 runs per configuration are listed in table 3.1.

	7 candidates			15 candidates			22 candidates		
$\|V\|$ \ k	2	3	4	2	3	4	2	3	4
40	0.054	0.023	0.032	0.083	0.108	0.124	0.224	0.211	0.193
100	0.023	0.013	0.016	0.033	0.085	0.074	0.051	0.102	0.113
400	0.022	0.013	0.006	0.033	0.043	0.049	0.038	0.070	0.068
1000	0.014	0.010	0.007	0.029	0.039	0.050	0.075	0.057	0.060

Table 3.1: Averaged scores of the baseline algorithm for different problem sizes.

As one would expect, the problem gets harder with an increasing number of candidates. Beyond that, there are some interesting insights to be gained from this table.

We notice that increasing the number of voters per candidate seems to improve the performance. This dependence on the ratio could indicate that the relative mass of voters and candidates plays a role in the gradient descent. Secondly, limiting the electoral photograph to two dimensions appears to be a bit more prone to outliers, as seen by the occasional spikes in the standard deviations listed in table 3.2. Intuitively this makes sense, since more dimensions give the points more freedom to move around each other instead of having to pass through potentially undesirable regions to get to a desirable place.

	7 candidates			15 candidates			22 candidates		
$\|V\|$ \ k	2	3	4	2	3	4	2	3	4
40	0.083	0.015	0.034	0.052	0.043	0.041	0.091	0.053	0.045
100	0.007	0.007	0.017	0.005	0.052	0.016	0.011	0.048	0.028
400	0.007	0.003	0.002	0.012	0.009	0.006	0.004	0.036	0.006
1000	0.004	0.003	0.002	0.005	0.005	0.007	0.115	0.029	0.009

Table 3.2: Standard deviations of scores of the baseline algorithm for different problem sizes.

3.3.1 Variations

For all the variations listed in section 3.2, we run experiments to investigate whether they provide any benefit over the baseline method.

Some of the experiments will focus on only a subset of the configurations listed above. For example, most experiments will use just two dimensions for the electoral photograph. This choice was made because higher dimensions tend to get easier anyway, and also, as mentioned in the introduction, limiting the photograph to two dimensions has the benefit of producing a more interpretable image. Two dimensions could be a sweet spot that might be most useful for real world applications.

Batching Strategies

Table 3.3 shows the influence of the batching strategies introduced in section 3.2.1. Every listed score is an average of 24 measurements with different numbers of candidates (7-22) and dimensions (2-4).

Only when the number of voters is small do we see a noticeable improvement when constructing batches such that every batch contains all the triplets for one voter. This dependence on the number of voters can be explained intuitively. For a small number of voters, a random batch will contain triplets belonging to a significant proportion of the voters. As the number of voters increases, the strategies become more similar because only a small fraction of the voters will be affected by any given random batch.

	Batching	
$\|V\|$	Random	Voter
40	0.1388	0.0959
100	0.0539	0.0604
400	0.0388	0.0379
1000	0.0430	0.0340

Table 3.3: Influence of the batching strategy

Skipping Voters

The results for various skip fractions are listed in table 3.4. All runs were done with 100 voters and 12 candidates. Considering the large variances across all results, it appears that no measurable improvement can be gained by implementing such a voter-skipping mechanism.

Fraction	0	0.1	0.2	0.5
Mean	0.0177	0.0312	0.0415	0.0260
Stdev	0.0046	0.0445	0.0507	0.0188

Table 3.4: Averages and standard deviations of scores across 10 runs each for different skip fractions.

Fraction	0.0	0.05	0.1	0.2	0.5
Mean	0.0161	0.0211	0.0344	0.0218	0.0399
Stdev	0.0026	0.0068	0.0499	0.0111	0.0392

Table 3.5: Averages and standard deviations of scores across 8 runs each for different re-initialise fractions.

Re-initialising Voters

We run experiments for 100 voters and 15 candidates in two dimensions. The results are shown in table 3.5.

No measurable benefit is observed with periodic re-initialisation of the worst performing voters as described in section 3.2.3. It does however seem that the variance increases with the re-initialisation fraction.

Initialisations

Initialisation	Random	Inverse	Simplex	Neural
Mean	0.0132	0.1429	0.0118	0.2187
Stdev	0.0037	0.0273	0.0024	0.0917

Table 3.6: Averages and standard deviations of scores across 8 runs each for different initialisation strategies.

Table 3.6 shows that while the results for the random initialisation as used by the baseline algorithm (cf. section 3.1) and the simplex initialisation described in section 3.2.5 are in a similar range, the initialisations from the permutation inverse (cf. section 3.2.4) and the neural network (cf. section 3.2.6) perform significantly worse.

Looking at the outputs of the neural network, we see a possible reason for the bad performance. A typical example of the outputs is shown in fig. 3.3.

The values appear to collapse almost completely into a one-dimensional subspace, with only slight variation in the second axis. This effect was observed repeatedly across many runs, and even persists when an additional error term is added that encourages the sample covariance matrix to approximate an identity matrix.

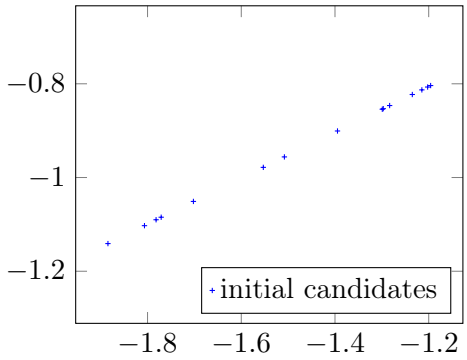

Figure 3.3: A typical output from a trained candidate initialisation model.

3.4 Visual Inspection

Since we start the process by generating zero-cost ground truth data, we can compare the learned embedding to this ground truth to get an insight into the quality of the obtained results.

We expect the found electoral photograph to approach the arrangement in the underlying data, up to some degrees of freedom that are not constrained in our algorithm. In this inspection we thus align the translation, uniform scale, rotation, and chirality of the result to approximate the ground truth in a least squares sense. Note that we limit this analysis to the 2-dimensional case.

Assuming the ground truth and output are perfectly related through rotation θ, translation t, and scale s, the mapping from ground truth position p to output position p' can be written as

$$\begin{pmatrix} s\cos(\theta)p_0 + s\sin(\theta)p_1 \\ -s\sin(\theta)p_0 + s\cos(\theta)p_1 \end{pmatrix} + t = p'$$

Now denoting $x = (s\cos(\theta) \quad s\sin(\theta) \quad t_0 \quad t_1)^T$, we solve for a least squares solution to get a vector x corresponding to our actual data. The following equation contains two rows for every voter and every candidate.

$$\begin{bmatrix} p_0 & p_1 & 1 & 0 \\ p_1 & -p_0 & 0 & 1 \\ & \vdots & & \end{bmatrix} x = \begin{pmatrix} p'_0 \\ p'_1 \\ \vdots \end{pmatrix}$$

This process does not deal with the chirality, but for that we can just solve the equation twice, once normally and once mirrored, and pick the better result.

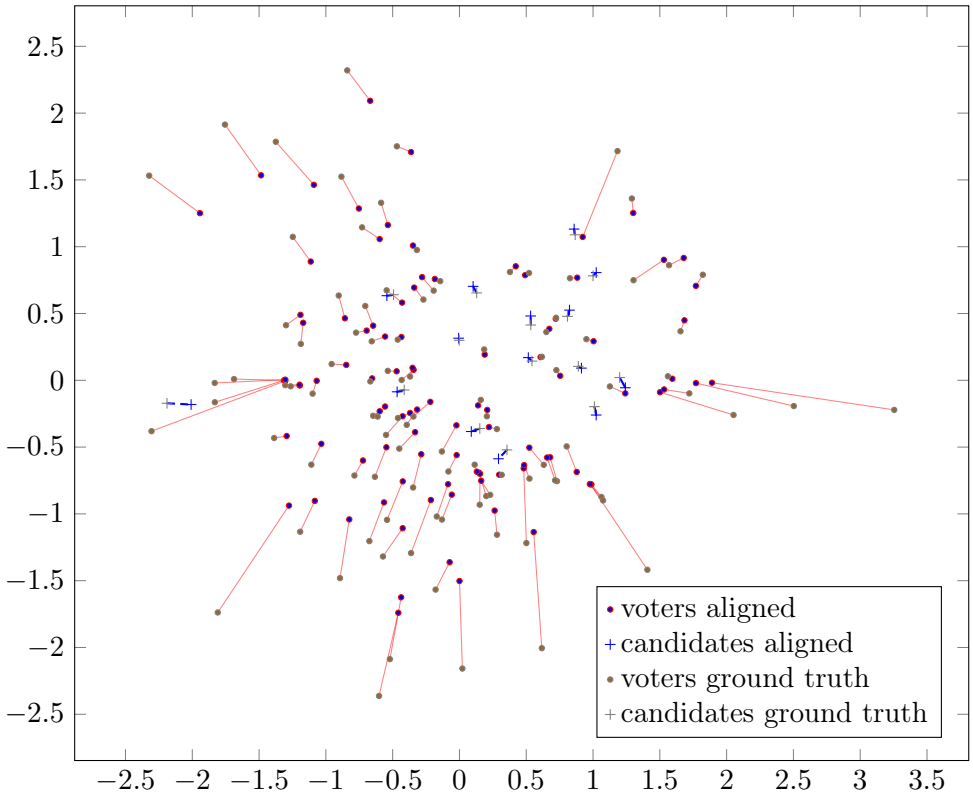

Figure 3.4: Visual comparison of results with a `norminv` score of 0.0112 to the ground truth data.

Using this alignment process we visualise the results of a run with a decent `norminv` score of 0.0112 in fig. 3.4 and one with a not-so-good score of 0.0409 in fig. 3.5. Both runs show 100 voters and 15 candidates and were performed with the baseline algorithm.

Corresponding voters from the ground truth and output have been connected with lines, same for the candidates. In both visualisations the main problems seem to occur for the outermost voters. Notably, most candidates are located close to where they should be according to the ground truth. This indicates that general political groupings can be observed even with a non-perfect solution.

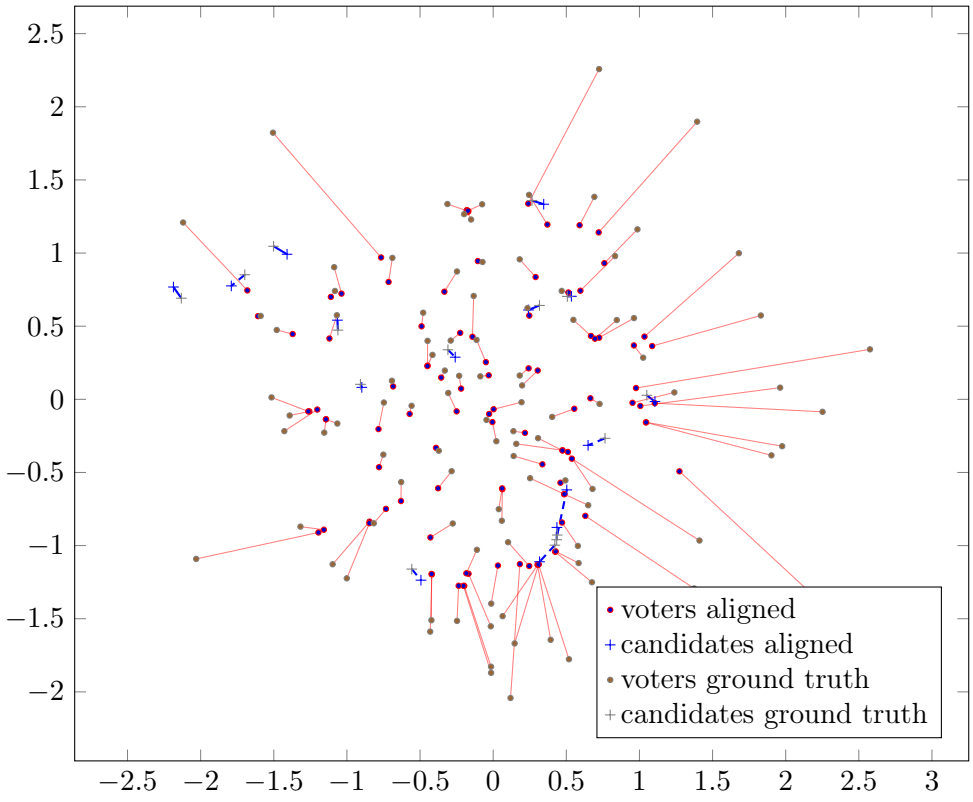

Figure 3.5: Visual comparison of results with a `norminv` score of 0.0409 to the ground truth data.

Chapter 4

Discussion

Unfortunately, none of the investigated methods was able to reliably find a perfect solution. Additionally, the variations over the baseline algorithm were surprisingly not able to provide any consistent improvements, with the only exception being adjustments to the batching strategy as described in section 3.2.1 and evaluated in section 3.3.1.

The simplex initialisation strategy described in section 3.2.5 showed slightly better results (cf. section 3.3.1), but the variances are too large to indicate any advantage beyond random chance. Looking at the visual inspections (cf. section 3.4) it doesn't seem like the candidate positions are the main issue holding back the algorithm, the only way the simplex initialisation could be beneficial is thus by finding good voter positions in $\mathbb{R}^{|C|-1}$ that remain useful after going down to k dimensions through PCA.

Ultimately, a basic triplet loss strategy seems sufficient to get a rough alignment. It remains unclear however how the remaining gap to a zero-cost solution can be closed.

4.1 Avenues for Future Work

To improve upon the results shown in this project, different ideas could be investigated as a follow-up:

- It might be possible to use the here presented baseline algorithm to get a rough solution, followed by a different method to refine the embedding.
- Alternatively, maybe a completely different approach where the embedding is built up incrementally is more successful. For example, reconstructing the 3d geometry of a static scene from photographs using a process called Structure-from-Motion works by adding cameras one by one to the scene, with occasional global adjustments to even out accumulated errors. This might inspire a process where an electoral photograph could be expanded by adding candidates one by one.

- Inspired by the approach by Laslier [3] in the setting of approval voting, a distance metric between candidates could be derived based on the ballot data, which is then used as a starting point for the electoral photograph.

Bibliography

[1] A. Gibbard, "Manipulation of Voting Schemes: A General Result," *Econometrica*, vol. 41, no. 4, pp. 587–601, 1973, publisher: [Wiley, Econometric Society]. [Online]. Available: https://www.jstor.org/stable/1914083

[2] M. A. Satterthwaite, "Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions," *Journal of Economic Theory*, vol. 10, no. 2, pp. 187–217, Apr. 1975. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022053175900502

[3] J.-F. Laslier, "Spatial Approval Voting," *Political Analysis*, vol. 14, no. 2, pp. 160–185, 2006, publisher: Cambridge University Press.

[4] A. Bogomolnaia and J.-F. Laslier, "Euclidean preferences," *Journal of Mathematical Economics*, vol. 43, no. 2, pp. 87–98, Feb. 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030440680600111X

[5] V. Knoblauch, "Recognizing one-dimensional Euclidean preference profiles," *Journal of Mathematical Economics*, vol. 46, no. 1, pp. 1–5, Jan. 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304406809000615

[6] D. Peters, "Recognising Multidimensional Euclidean Preferences," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10616

[7] F. M. T. A. Busing, P. J. K. Groenen, and W. J. Heiser, "Avoiding degeneracy in multidimensional unfolding by penalizing on the coefficient of variation," *Psychometrika*, vol. 70, no. 1, pp. 71–98, Mar. 2005. [Online]. Available: https://doi.org/10.1007/s11336-001-0908-1

[8] F. M. T. A. Busing, "Advances in multidimensional unfolding," Ph.D. dissertation, Leiden University, 2010.

[9] E. Hoffer and N. Ailon, "Deep metric learning using Triplet network," Dec. 2018, number: arXiv:1412.6622 arXiv:1412.6622 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1412.6622

[10] "Keras documentation: Embedding layer." [Online]. Available: https://keras.io/api/layers/core_layers/embedding/

[11] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam, "Theseus: A Library for Differentiable Nonlinear Optimization," *arXiv preprint arXiv:2207.09442*, 2022.