Utrecht University

A Non-disruptive Crossover Operator for Multi-layer Perceptron Networks using Parametric Similarity

Ronald Janssen

supervised by
Dr. Ir. D. Thierens

June 8, 2020

Contents

1 Abstract

Functionally equivalent multi-layer perceptron networks (MLPs) can be written in many different forms. This presents difficulty when trying to recombine these networks using the crossover operator. This thesis aims at finding a method to identify similar neurons in different MLPs by their parameters. This allows for defining crossover operators that do not depend on the forms in which different networks are written. Two new crossover operators are presented that use similarity between the parameters of different neurons to facilitate a non-disruptive crossover. These crossover operators have been found to be relatively non-disruptive compared to uniform and one-point crossover.

2 Introduction

Neural networks have been gaining in popularity over the past years for their applicability as universal function approximators (Hornik et al., 1989), combined with their ability to 'learn' from data. Neural networks come in many forms. In this paper we will focus on multilayer perceptron networks (MLPs), which are among the most common. The most common way to train neural networks is by gradient descent. This is a process of incrementally making small changes to reduce the error of the network. Gradient descent is a form of local search, meaning it searches solutions that lie close to the point where it starts its search. Gradient descent therefore only searches a small part of the total available solutions. As only a small part of all potential solutions are searched, there is a decent chance gradient descent won't find the best solution. Instead gradient descent finds local optima (definition 2.1). In order to search all solutions, and find the global optimum (definition 2.2), a global search operator is required. Evolutionary algorithms are global search operators.

Definition 2.1. *Local optimum*: The best of all nearby solutions.

Definition 2.2. *Global optimum*: The best solution that can be found.

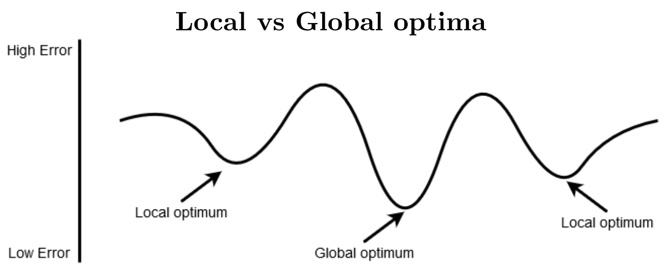

Figure 1: This figure shows a fitness landscape with two local optima (definition 2.1) and a global optimum (definition 2.2).

Evolutionary, or genetic, algorithms (EAs / GAs) have a long history as an optimization tool for neural networks (NNs), both for training their parameters (Montana and Davis, 1989) and designing their architecture (Miller et al., 1989). The main advantage of population based algorithms, such as EAs, is that they search the search space more globally, reducing the chance of ending up in local optima. Although multiple operators exist in the field of EAs, we will focus on one, crossover. The crossover operator is arguably the most innovative operator of in the field of genetic algorithms (Ortiz-Boyer et al., 2007). This operator allows for recombining traits in individual solutions (parents), resulting in new individuals (children) who inherit their strengths (and weaknesses). The goal of this operator is to mix the different traits of two parents, while still producing individuals that behave like their parents. Producing children that behave similarly to their parents is called being non-disruptive (definition 2.3). When applying a simple crossover operator to neural networks the property of non-disruptiveness does not hold. This is due to the competing conventions problem. Which brings us to our goal: Defining a crossover operator for neural networks that is untouched by the competing conventions problem.

Definition 2.3. *Non-disruptive*: A crossover operator is called non-disruptive if it produces children that are similar in behaviour to their parents.

We will go deeper into what the competing conventions problem is in the following section, but for now it is sufficient to know it has to do with the fact that the same network can be written in many different ways. One way to circumvent this issue could be to have our crossover operator use some mapping between differently written networks, such that the way the network is written becomes irrelevant. This will be achieved by finding similar neurons in different networks. This leads us to our research questions:

Main Question. Is it possible to facilitate a non-disruptive exchange of information between multi-layer perceptron networks by matching similar neurons?

Subquestion 1. Given 1 'base' neuron and 2 others, can we predict which of the 2 others behaves most similar to the base?

Subquestion 2. Does this prediction still hold when these neurons are situated in a trained network?

Subquestion 3. Is it possible to use these predictions to facilitate a non-disruptive exchange of information between individuals?

In the upcoming section we will dive deeper into the competing conventions problem. This starts with a more detailed explanation, followed by examples of earlier attempts at tackling this issue. In section 4 individual neurons will be investigated with the aim of answering the first subquestion. Section 5 expands on this by validating these results for neurons in a network (subquestion 2). The goal of section 6 is to answer the third subquestion by defining crossover operators using what we have learned so far. To satisfy our curiosity the performance of the new crossover operators will be measured and discussed in section 7. In the final section we will answer our main question, and briefly look forward at what can be done to improve our operators.

2.1 About the code

Programming has been done in Python (Van Rossum and Drake, 2009). For neural network generation and training the pytorch package (Paszke et al., 2017) was used. Pytorch and Numpy (Oliphant, 2006) have been used in calculations. Distance measures and clustering based thereon has been done using the scipy package (Virtanen et al., 2020). Visualisations were made using matplotlib (Hunter, 2007). Pandas (McKinney et al., 2010) has aided in dataprocessing.

3 Competing Conventions

In this section previous attempts at solving the competing conventions problem are discussed. But first an explanation will be given of what a multilayer perceptron is, along with how this results in competing conventions and why this is problematic when defining a non-disruptive (definition 2.3) crossover operator.

A multilayer perceptron (MLP) network consists of layers of nodes. Except for the first layer, where the nodes contain the input variables of the network, each node is a neuron. For n inputs, neurons perform the following computation:

$$tanh(bias + x_1 * w_1 + x_2 * w_2 + ... + x_n * w_n)$$

Where x_1 to x_n are the inputs to that neuron. The weights (w_1 to w_n) and $bias$ are the parameters of the neuron, these parameters are optimized trough training. The transfer function, $tanh()$, is the hyperbolic tangent. The hyperbolic tangent maps any number to a unique number between -1 and 1, where $tanh(0) = 0$.

Above is shown that neurons use their inputs to calculate a weighted sum. This means the order of the inputs to a neuron is irrelevant for its calculation, so long as the weights still correspond to the correct input. In turn, this means that the location of a neuron in a layer is similarly irrelevant for the functioning of an MLP. Figure 2 shows an example of two MLPs that compute the same function, but where the neurons in the second layer are ordered differently. The outputs of the networks, calculated by N_{out}, are $tanh(x_1 * w_1 + x_2 * w_2)$ and $tanh(x_2 * w_2 + x_1 * w_1)$, proving the networks' computations are equivalent. The idea that the competing conventions problem is caused by the fact that the order, or permutation, of neurons in a layer is irrelevant is where this problem gets its second name: the permutation problem.

Two Multilayer Perceptrons Performing the Same Calculation

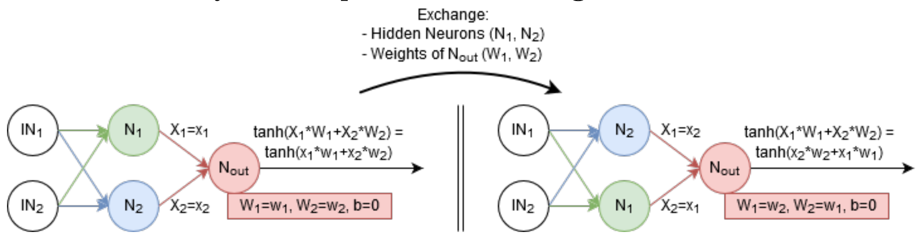
Exchange:
- Hidden Neurons (N_1, N_2)
- Weights of N_{out} (W_1, W_2)
IN_1
IN_2
N_1
N_2
$X_1=x_1$
$X_2=x_2$
N_{out}
tanh(X_1*W_1+X_2*W_2) = tanh(x_1*w_1+x_2*w_2)
$W_1=w_1$, $W_2=w_2$, b=0
IN_1
IN_2
N_2
N_1
$X_1=x_2$
$X_2=x_1$
N_{out}
tanh(X_1*W_1+X_2*W_2) = tanh(x_2*w_2+x_1*w_1)
$W_1=w_2$, $W_2=w_1$, b=0

Figure 2: Shown here are two multilayer perceptrons (MLPs). These MLPs differ solely in the position of the neurons in the second layer and the weights of the output neuron. Both those neurons and the weights are swapped with respect to the other MLP. As such both MLPs calculate the same function.

Generally, the crossover operator exchanges parts of the parent genome based on them being in the same location. As the location of a neuron within a layer is irrelevant, exchanging neurons based on their location can be very disruptive (definition 2.3). Figure 3 shows children that could be generated by applying a standard crossover operator to the networks in figure 2. The outputs of these children are $tanh(x_1 * w_1 + x_1 * w_2)$ and $tanh(x_2 * w_2 + x_2 * w_1)$. Neither of these children behaves similar to its parents, implying the requirement of non-disruptiveness has not been met.

Children of Parents with Differently Ordered Neurons

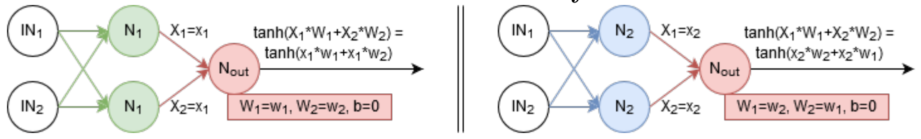
IN_1
IN_2
N_1
N_1
$X_1=x_1$
$X_2=x_1$
N_{out}
tanh(X_1*W_1+X_2*W_2) = tanh(x_1*w_1+x_1*w_2)
$W_1=w_1$, $W_2=w_2$, b=0
IN_1
IN_2
N_2
N_2
$X_1=x_2$
$X_2=x_2$
N_{out}
tanh(X_1*W_1+X_2*W_2) = tanh(x_2*w_2+x_2*w_1)
$W_1=w_2$, $W_2=w_1$, b=0

Figure 3: Shown here are two multilayer perceptrons (MLPs) which are children of the MLPs shown in figure 2) formed by crossover. Both children perform quite different calculations from their parents, implying the requirement of non-disruptiveness (definition 2.3) is not met.

In multilayer neural networks the problem of competing conventions enlarges. For deep layers not only the neurons of the layer itself but also the former layer can be randomly permuted, effectively permuting the incoming weights. Figure 4 displays 4 network sub-graphs that are functionally equivalent. In general the amount of permutations for subsequent layers i to j equals $\prod_{l=i-1}^{j} N_l!$, where N_l equals the number of neurons in layer l. This holds for any layer except the first, as the inputs to the first layer come from outside the network and are, as such, given in a set order. It is good to note that, as the input to every layer is a function of the layer before, neurons in later layers of equivalent networks are affected by the possible permutations in former layers. Because of this extra redundancy, layers beyond the second layer are not part of this research.

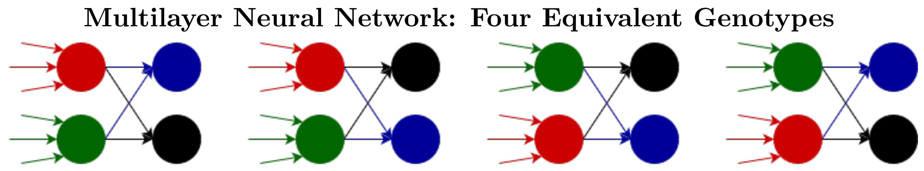

Figure 4: Shown above are 4 permutation of a functionally equivalent part of a multilayer neural network. Different neurons are shown as having different colors.

In the rest of this section, previous attempts at defining a crossover operator for neural networks are discussed. However, before we continue, we should have a look at two terms that will be used often throughout this section, genotype and phenotype. In a nutshell, the genotype refers to the encoding of an individual, whereas the phenotype refers to it's behavior. Genotype and phenotype are, as much of the terminology used in evolutionary computing, lend from biology. A common biological example of the difference between genotype and phenotype is eye color. There are multiple DNA structures (genotypes) that encode for brown eyes (phenotype). Figure 2 shows two genotypes that correspond to one phenotype.

Definition 3.1. *Genotype*: The encoding of a neural network.

Definition 3.2. *Phenotype*: The functional behaviour of a neural network.

Using these definitions it can be stated that the competing conventions, or permutation, problem arises from the fact that the ordering of neurons in a fully connected layer results in the same phenotype (definition 3.2), for many different genotypes (definition 3.1).

3.1 A Non-Redundant Genetic Encoding

The competing conventions problem, as described before, was solved by professor Thierens over two decades ago (Thierens, 1996). In his paper he defined equivalence classes along with a way to reduce every member of these classes to one representative of its class. In other words, he proposed a standardized genotype (definition 3.1) representation along with a way to rewrite networks of the same phenotype (definition 3.2) in the form of this standardised genotype. The proposed method works for multilayer perceptrons using the hyperbolic tangent as their transfer function. In this case, besides the redundancy in the order of neurons in a layer, any neuron can also be rewritten into a functionally equivalent neuron using sign flips.

Rewriting a neuron using sign flips is possible because the hyperbolic tangent has a rotational symmetry around the origin with a degree of 2. I.e. when a plot of this function is rotated 180 degrees around the origin, the original function is retrieved. This can be written as $tanh(-x) = -tanh(x)$, from which can be derived that $tanh(x) = -tanh(-x)$. The latter is the interesting part, it means that if the sign of the input (x) to the hyperbolic tangent $(tanh(x))$ is flipped $(- \leftrightarrow +)$ along with the sign of the output of the function, the result does not change. For a neuron we saw that the input to the hyperbolic tangent equals: $bias + x_1 * w_1 + x_2 * w_2 + ... + x_n * w_n$. This means we can flip the sign of the input (x) of the hyperbolic tangent $(tanh(x))$ by flipping the signs of the bias and all of the weights. The sign of the output can be flipped by flipping the signs of of the weights of all outgoing connections from a neuron. Thierens choose to solve this redundancy by enforcing positive biases. When the bias of a neuron is negative, this neuron is rewritten by 1) flipping the signs of the bias and all weights of this neuron, then 2) flipping the signs of the weights of all outgoing connections from this neuron.

At the start of this section it was discussed that the order of neurons in a layer does not affect the behavior of a network. As neurons in a layer containing N neurons can be ordered in $N!$ different ways we can conclude there are N different genotypes (definition 3.1) corresponding to a single phenotype (definition 3.2). Above was discussed that, when using the hyperbolic transfer function, it is possible to write each neuron in two different ways. As such it is possible to write a layer containing N neurons in 2^N different ways. Combining these two, it can be concluded that a layer consisting of N neurons with the hyperbolic tangent as their transfer function can be written in $2^N N!$ functionally equivalent ways (Thierens, 1996). In other words, there are $2^N N!$ genotypes corresponding to a single phenotype for a layer of size N. To achieve his non-redundant encoding, Thierens chose to rewrite different genotypical representations of the same phenotype in two steps 1) by flipping the signs such that all biases in the networks became larger than zero, and 2) by sorting the networks' layers by increasing bias. The eight different genotypes for a layer containing two neurons are shown in figure 5. Step 1 would reduce any genotype to one the two on the left, which one would depend on step 2. The resulting genotype was named the non-redundant encoding.

Genotypes for a two-neuron layer corresponding to any phenotype

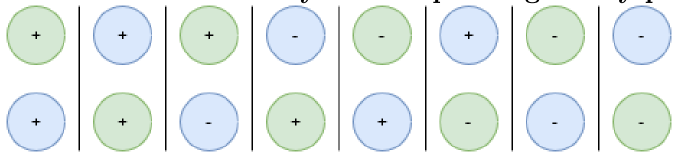

Figure 5: Shown are eight different representations (genotypes, definition 3.1) that can correspond to a single network behavior (phenotype, definition 3.2.)

Definition 3.3. *Non-redundant encoding* (Thierens, 1996): A neural network encoding that has been reduced to the member of its equivalence class where all biases are positive, and neurons in a layer are sorted by increasing bias.

The crossover operator would operate on complete neurons, which were defined as their incoming weights and their bias. The result was that when crossover was applied to the networks written in non-redundant form, the crossover correlation coefficient (CCC) (definition 3.4) was significantly higher than when written in any (random) form. This implies that the children of two non-redundant networks indeed behave more like their parents than their redundant counterparts.

Definition 3.4. *Crossover correlation coefficient (CCC)* (Manderick et al., 1991): A measure of how correlated the fitness landscape (a simple example of such a landscape is given in figure 1) appears to the crossover operator. Calculated as:

$$\frac{cov(F_p, F_c)}{\sigma(F_p)(\sigma(F_c)}$$

Where: F_p and F_c are arrays containing the average fitnesses (sum of squared errors) of parents and their respective children. In this thesis the CCC is used as a measure of non-disruptiveness.

Mapping genes along the genome by sorting on a single feature does in effect solve the competing conventions problem. However, when the goal is to perform crossover effectively, such a method gives rise to a new problem. When applying the standard crossover, neurons are exchanged based on their location in the genotype. As such, similar neurons should ideally occupy similar locations in genotypes of different individuals. An ideal solution would not just yield a single representation for each network phenotype, but also similar representations for similar phenotypes. This is however not necessarily the case when sorting a network by bias. Figure 6 shows an example of this. The first network (left) shows two neurons, in orange and blue, that are sorted by bias. Then a minor change occurs to the orange neuron, increasing it's bias, becoming the red neuron in the second network (middle). Sorting the second neuron by bias, as is done to achieve the non-redundant encoding, results in the third network (right). Even though the red neuron is only a small variation on the orange, it is now located in an entirely different place on the genome. On top of that the blue neuron, which has not changed at all, is also displaced.

Figure 7 shows a practical example of how this may impact children's performance on the XOR problem. In figure 7 the difference between both parents is also only a swap between biases of two neurons, after which the neurons are sorted.

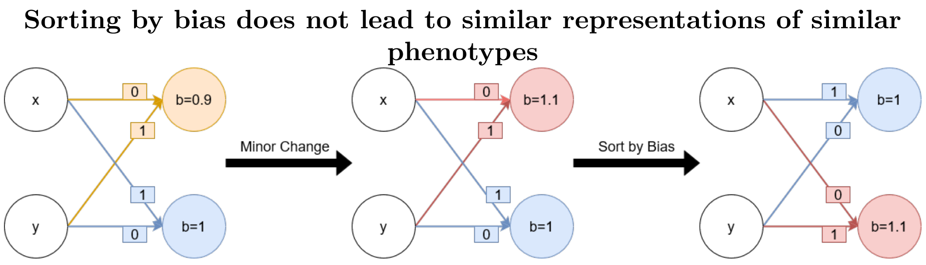
Sorting by bias does not lead to similar representations of similar phenotypes
x
y
0
1
1
0
b=0.9
b=1
Minor Change
b=1.1
Sort by Bias

Figure 6: This figure shows three genotypes, corresponding with two phenotypes. The first nodes (x and y) are input nodes. The colored nodes represent neurons, whose weights are shown along the connection to their inputs. The genotypes on the left and right are sorted by bias, however the middle representation matches the left better than the right.

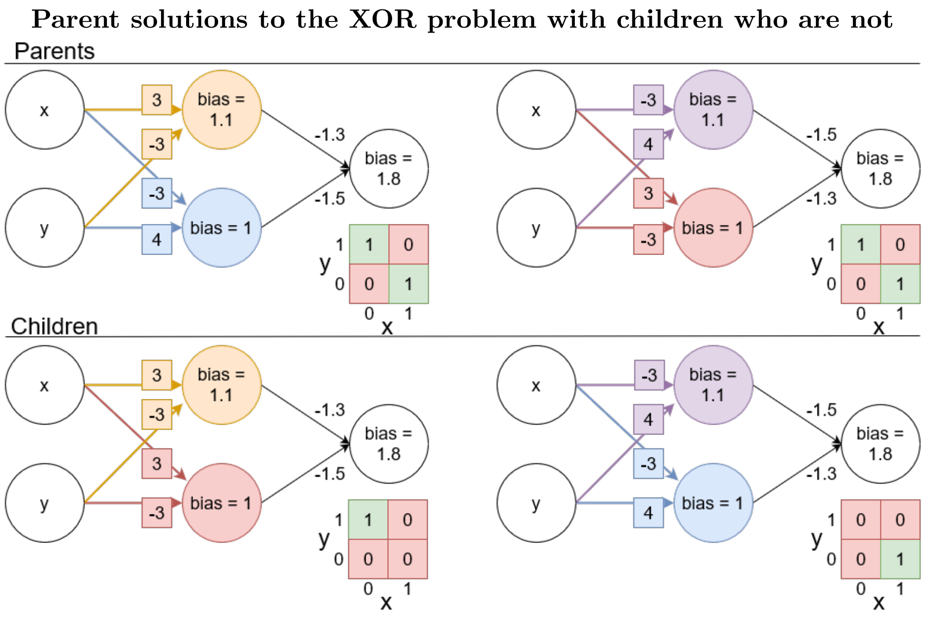
Parent solutions to the XOR problem with children who are not
Parents
Children
bias = 1.1
bias = 1
bias = 1.8
-1.3
-1.5
3
-3
4
x
y
0
1

Figure 7: Two parent networks in non-redundant form (definition 3.3), both solutions to the XOR problem, along with two children who are not solutions. Chidren were generated by crossing the second neuron in the hidden layer (with the lowest bias). As this regards the XOR problem, the inputs can be either 0 or 1. Arrows represent connections between neurons, the numbers show their respective weights. The transfer function is the hyperbolic tangent.

3.2 Redefining the Crossover Operator

In 2006 another approach was made to combat the competing conventions problem, by redefining the crossover operator such that it would not be effected (García-Pedrajas et al., 2006). As in the formerly discussed paper (Thierens, 1996), no neurons were split during crossover. Instead the goal of the algorithm was to find the optimal combination of the neurons in the layers of five parents. The essence of the algorithm is as follows. When trying to optimize a certain layer in a neural network, five networks are trained independently. If we make the layer of interest a size of 10, that results in 50 candidate neurons for the optimized layer. An individual is then defined as a binary array of length 50, where 1 means that the corresponding neuron will be a part of the individual and 0 the contrary. These individuals are then optimized by a genetic algorithm.

The crossover that is used is the half uniform crossover (HUX), which means that exactly half on non-matching bits are exchanged between parents. After crossover the best N individuals are selected from the parents and children to form the new generation. The nodes in this new generation are then, with some probability, subject to parametric mutation, either random or backpropagation for a fixed number of cycles. An exponential regularization term, using the amount of included neurons as input, was added to the fitness function, mainly to prevent the rise of very large networks.

This algorithm has a few advantages over regular crossover. As the order of neurons is irrelevant to this algorithm, the permutation problem is of no consequence. Also, the size of a network layer does not need to be pre-specified, as the option to combine multiple networks allows for large layer sizes, whereas the regularization term keeps the network from growing unnecessarily large.
The amount of combinations that can be made from multiple layers however grows exponentially with the total amount of neurons. Combinatorial optimization could be done more efficiently if it would be possible to group similar neurons, such that functionally overlapping neurons can be avoided in solution. This would reduce the search space dramatically. That clustering of nodes before crossover could be beneficial has been recognized by the authors in the discussion. Later they tried such an approach using 'virtual parents' (Ortiz-Boyer et al., 2007). An alternative approach could be to try to measure similarity more directly, clustering neurons based on that measure. Finding such a measure is one of the goals of this thesis (subquestions 1 and 2).

3.3 Growing Neural Networks

Ways to exploit neuron similarities without measuring them do exist, such as the innovation number in the NEAT algorithm. NeuroEvolution through Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) is one of the most impactful algorithms in the field that combines neural networks with evolutionary algorithms. It has the ability to optimize both structure and weights

by incrementally growing and combining small networks.
The evolutionary methods employed by NEAT can be described in four parts: encoding, crossover, mutation, and selection using speciation. NEAT uses a direct encoding, i.e. the phenotype (definition 3.2) of an individual can be read directly from the genotype (definition 3.1). Neurons (or nodes) are numbered, such that a connection between neurons (or edge) can be described using two numbers (e.g. 1-3 would describe a connection going from neuron 1 to 3). Connections are then described by the two neurons it connects, a weight and a binary number, the last allowing a connection to be turned off so that it is not present in the phenotype. Most importantly, evolved edges will the tracked using a innovation number, this innovation number plays an important part during crossover.

The crossover operator is relatively straight forward to describe. Important to note is that it is only applied to the connection genes, as the required neurons for a functional individual can be derived from its connections alone. The first step is to align the genes of the two parents that share an innovation number. The genes that are not aligned are taken from the best performing parent, the genes that are aligned are crossed uniformly. Having the structure of the offspring be determined by one of its parents ensures that it is functional (assuming its parent is).

NEAT makes use of five mutations. Simplest are the weight mutations. One being the weight shift, where a small number is added or subtracted from a weight present in the network. The other being weight randomization, where a weight is replaced by a random number. There are three mutations that change connections in the phenotype. Firstly, the enable/disable edge mutation flips the boolean that describes whether an edge will be present in the phenotype. Secondly, the edge addition mutation generates a new edge between two neurons, with a random weight. If an edge is new, where new edges are kept track of on a per generation basis, it receives a new (incremental) innovation number. The third is the node addition, this effectively splits an edge down the middle and places a node in between. More precisely, it disables an edge, generates a node, then generates two edges that reconnect the original nodes through the newly generated node.

Selection through speciation grants newly generated individuals a chance to optimize before being removed from the population. Before selection, the algorithm groups individuals based on similarities in the genome. These groups, or species, are then made to compete against each other, instead of the whole population. This means that after selection the best individuals of each species are kept, contrarily to the best individuals on the whole. This protects species that contain structural innovations that still need some generations to optimize.

Growing a network gives NEAT the ability to track innovative changes. The innovation number tracks something similar to homologous genes in evolutionary biology, i.e. genes with the same ancestor. The other type of similarity that genes can have is analogy. Even though analogous genes share no ancestor they are still alike, this happens when two individuals evolve the same structure independently of one another. When training different individuals separately, there can be no homologous genes. Analogous genes do however arise frequently, as different solutions often have commonalities. The similarity measure that this thesis aims to find should make it possible to find analogous genes, which could be treated similarly to homologous genes by the crossover operator.

3.4 Ignoring crossover

Although it is not the focus of this research, it is important to note that some evolutionary algorithms ignore the crossover operator all together, thereby avoiding the competing conventions problem as a whole. An example of this is Deepmind's population based algorithm (Jaderberg et al., 2017). The weights in the population's networks are optimized using gradient descent and are not directly impacted by genetic operators. Instead, the operators are applied in two ways, which are fairly standard aside from some modifications to allow parallelization. Firstly 'exploit', meaning selection is performed. Selection was done using two methods. One was by binary tournament, meaning two individuals are compared and the best individual replaces the worst. The other is truncation selection, where the bottom x% of solutions are replaced by solutions randomly chosen from the top x%. Secondly 'explore', an operator which only affected the hyperparameters such as the learning rate, thus not the weights directly. This operator could affect hyperparameters either by perturbing or resampling them. The success of this algorithm proves that adding just basic genetic operators can result in improvements over current state of the art. However, the crossover operator can be a strong addition to an algorithm (Jansen and Wegener, 2002), and attempts to make its inclusion possible, have sparked quite some research.

3.5 Summary

It has been shown that there are solutions to the permutation problem (Thierens, 1996), in the sense that any network can be written in a standardized form. However, the difficulties of facilitating a non-disruptive exchange of information between neural networks (per our main research question) do not stop there. Ideally, not only could we match the same genes in differently permuted networks, but also similar genes in otherwise different networks.

This idea is not new. The NEAT algorithm keeps track of homologous genes (genes with a shared ancestry) via its innovation number. Crossover is then applied to homologous genes only. Due to how the algorithm grows its networks, it is believable that most similar genes will be homologous. As such, in the context of the NEAT algorithm, tracking homologous genes is a good way of finding similar genes. However, when trying to combine separately trained networks this is not possible. Similar genes in this case are not homologous but analogous. Meaning they are similar, but lack a shared origin. As we cannot keep track of our neurons throughout the process, a computationally feasible way in which to predict whether two neurons are similar is desired. To be feasible our resulting measure should not require functional network analysis, meaning it must be able to determine similarity using only the available weights and biases. Such a measure may also make it possible to improve a sorting method as done by D. Thierens (Thierens, 1996), or to improve the crossover operator such as one relying on combinatorial optimization (García-Pedrajas et al., 2006) by grouping similar neurons.

This research aims to define a crossover operator that is minimally disruptive. The approach is to find some measure that allows us to easily predict whether two neurons behave similarly. These predictions may then be used to facilitate a non-disruptive crossover (subquestion 3). Potential similarity measures will be evaluated by testing if neurons that are similar according to the measures also yield similar outputs. This is done for both individual neurons (subquestion 1) and networks containing these neurons (subquestion 2).

4 Distance Measure - Single Neuron Network

In this subsection we will be looking at individual neurons, which alternatively can be viewed as networks that consist of a single neuron. Recapping our first research question, the aim is to predict if the behavior of some neuron is similar to some other neuron (definition 4.1), based on their parameters (definition 4.2). The possibility of this will be investigated by comparing a list ordered by neurons' behavior with a list ordered based on their parameters.

Definition 4.1. *Behavioral similarity*: Viewing a neuron as applying a function to its inputs. Two neurons are considered to behave similarly if they yield similar outputs for the same input.

Definition 4.2. *Parametric similarity*: Neurons can be described by their parameters (weights and biases). Two neurons are considered parametrically similar if the values of their parameters are similar.

We are looking for parametric based orderings that are much alike to the behavior based ordering, as this implies that we can accurately predict the similarity in 2 neurons' behaviour based on their parameters. This results in a three-step process:

1. Order neurons based on behavioral similarity (definition 4.1)
2. Order neurons based on parametric similarity (definition 4.2)
3. Compare the ordering based on parametric similarity with the ordering on behavioral similarity

All three steps mentioned above can be done in a variety of ways. As this phase of the study is exploratory in nature, the choice was made to cast a wide net. This effectively means that multiple methods are used for all three steps. These will be discussed not far from now, but first we should have a look at the neurons themselves and the way the data is gathered.

4.1 Methods

The neuron-similarity-measures will be tested on two types of neuron. One where there are two inputs (figure 8A), and one where there are ten (figure 8B). In both settings the inputs are connected to this one neuron. Different neurons will receive the same set of inputs, so that we can compare their difference in outputs.

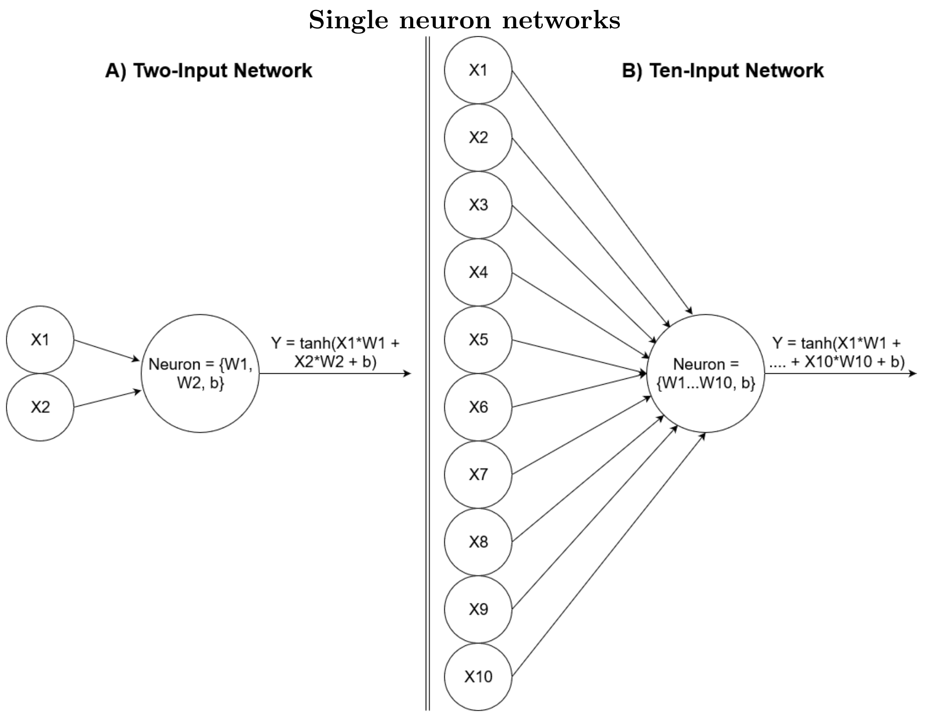

Figure 8: Two (A) or ten (B) controlled input neurons, one output neuron which can be replaced.

In the two-input setting an exhaustive search in the input space is performed, by varying both inputs from -1 to 1 in steps of 0.1. The differences in output are calculated from the 441 results obtained this way. How will be discussed momentarily. We can describe our neurons using their three parameter values weight1, weight2 and bias, or w1, w2, b. Weight1 is held constant at 5, weight2 ranges from 1 up to and including 10 in steps of 1, as symmetry in inputs around 0 allows us to only research positive weights. Biases range from -10 up to and including 10 in steps of 5. In the ten-input setting the neuron-similarity-measures are tested using 1000 random inputs. A hyperbolic tangent is used as the transfer function.

So far we have discussed sorting (neurons) based on similarity. However we have ignored an important question: Similarity to what? If we wish to sort 100 neurons based on similarity we need some baseline for these neurons to be similar to. As there is no clear 'baseline neuron', the decision was made to repeat the sorting and comparison for all neurons. In the list of all (100) neurons, first neuron is chosen as the comparison neuron. The similarities of other neurons are then calculated using that as the base, after which neurons are ordered accordingly. We will then iterate over our list of neurons, until each neuron has had the role of comparison neuron. We end by averaging the results obtained to score the similarity measures.

4.1.1 Similarity measures

We have discussed that lists of neurons ordered by behavioral similarity (definition 4.1) will be compared with lists ordered by parametric similarity (definition 4.2). How exactly these lists are made depends on how similarity is measured. In this subsection we will define the similarity measures that will be researched along with discussing why these may be relevant.

Behavioral similarity measures
Behavioral similarity is defined by the relation between the input and output of a neuron (definition 4.1). We approximate behavioral similarity by applying distance measures to lists of generated outputs. Three distance measures were chosen to compare the outputs of different neurons: cityblock, sqeuclidian, and classification.

Definition 4.3. *Cityblock distance*: The cityblock distance d between vectors $\mathbf{p}$ and $\mathbf{q}$ of length n is given as follows:

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n} |\mathbf{p}_i - \mathbf{q}_i|$$

The cityblock distance, also known as the Manhattan distance, is probably the most straightforward way of measuring a distance. This distance is computed by taking the sum of all absolute differences.

Definition 4.4. *Sqeuclidian distance*: The sqeuclidian distance d between vectors $\mathbf{p}$ and $\mathbf{q}$ of length n is given as follows:

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n} (\mathbf{p}_i - \mathbf{q}_i)^2$$

The sqeuclidian distance equals the sum of all squared differences between two lists (which equals the euclidian distance squared). Compared to the absolute difference, the sqeuclidian distance (4.4) weighs larger differences more heavily. This type of weighing is common when training neural networks, where gradient descent generally optimizes over the squared error. For a single input given to two neurons, yielding the outputs out_1 and out_2, this looks like: $(out_1 - out_2)^2$

Definition 4.5. *Classification distance*: The number of inputs that two networks classify differently.

Later, the promising methods will be tested on bi-classification problems, meaning a classifier is trained to classify inputs as belonging to one of two possible categories. It may therefore be useful to see whether we can predict the impact of different neurons on the classification directly.

Parametric similarity measures
Parametric similarity is a measure of the similarity between the weights and bias of neurons (definition 4.2). Six distance measures were chosen to compare the parameters of different neurons. The first four of these apply a single distance measure to all parameters, summing the total to calculate the distance between two neurons. The latter two combine two distance measures, the reason for this is discussed as we reach them.

The cityblock (definition 4.3) and the sqeuclidian distance (4.4) that we discussed for measuring behavioral similarity are reused for measuring parametric similarity. The arguments of simplicity and wide usage apply here as well.

Definition 4.6. *Canberra distance*: The Canberra distance d between vectors $\mathbf{p}$ and $\mathbf{q}$ of length n is given as follows:

$$d(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} \frac{|\mathbf{p}_i - \mathbf{q}_i|}{|\mathbf{p}_i| + |\mathbf{q}_i|}$$

For each value pair in the two vectors, the absolute distance between two points divided by the sum of their absolute values $\frac{|\mathbf{p}_i - \mathbf{q}_i|}{|\mathbf{p}_i| + |\mathbf{q}_i|}$ is calculated. Summing these values yields the Canberra distance (definition 4.6). The Canberra distance can be seen as a distance relative to the size of the points the distance is calculated between. Relative differences may be a useful way to look at variation in parameters, as the weights of a neuron (which all parameters but 1 are) are used in a multiplicative way (as described in example 4.7).

Example 4.7. *Neuron parameters are used multiplicatively*: Let us look at the calculation performed by a neuron with one input (i), no transfer function, and a bias of 0. This means the calculation it performs is simply a multiplication of the input (i) with its weight (w): $i * w$. Let's compare 2 sets of 2 such neurons,

defined by their weights: $w^1 = 0.1, w^2 = 0.2$ and $w^3 = 1.0, w^4 = 1.1$. Using either absolute or squared difference as a metric, the difference between w^1 and w^2 is equal to the difference between w^3 and w^4. However, it may be better to view a change from w^1 to w^2 as doubling the impact of its corresponding input (in this case the only input), whereas this cannot be said for a change from w^3 to w^4.
For two neurons (n^1 and n^2) with J parameters ($p_1...p_J$), this difference can be formally described as: $\sum_j (|p_j^1 - p_j^2|/|p_j^1| + |p_j^2|)$

Definition 4.8. The cosine distance d between vectors $\mathbf{p}$ and $\mathbf{q}$ of length n is given as follows:

$$d(\mathbf{p}, \mathbf{q}) = 1 - \frac{\mathbf{p} \cdot \mathbf{q}}{||\mathbf{p}|| + ||\mathbf{q}||}$$

Where $\cdot$ is the dot product and $||\mathbf{v}||$ is the magnitude of vector $\mathbf{v}$

Neurons can be viewed as drawing hyperplanes (or just planes since we're researching neurons with two inputs) through the input space. As such their difference may be best described by the angular difference between these planes. This is represented by the cosine distance (definition 4.8).

Looking back at the reasoning for using the cosine distance, it seems incomplete. This is because angle alone does not fully specify the difference between two lines. Two lines with the same angle are only parallel, not necessarily equal. We can however fully describe these two lines if we also include their distance from 0, given that all input parameters are 0. Luckily for us it is quite straightforward to handle angle and this distance from 0 separately, as they are already separated in our list of parameters. The angle is described by the weights, and the bias gives the distance from 0 given that all inputs are 0. The final two parametric similarity measures, cos-abs distance (definition 4.9 and cos-sq distance (definition 4.10) use this.

Definition 4.9. *Cos-abs distance*: A weighted sum of the cosine distance (definition 4.8), calculated over the weight vector and the absolute difference in bias. The weighting is determined by the full range of neurons that the operator receives, such that the absolute difference is scaled between 0 and 1.

Definition 4.10. *Cos-sq distance*: A weighted sum of the cosine distance (definition 4.8), calculated over the weight vector and the squared difference in bias. The weighting is determined by the full range of neurons that the operator receives, such that the absolute difference is scaled between 0 and 1.

Two ways are used to compare the lists ordered by parametric similarity with those on behavioral similarity. When discussing the choices made we should keep our goal in mind: To find neurons that are similar to one another. The first comparison, simset intersection (definition 4.12), is designed to test for the minimally necessary quality, namely that both orderings agree on the most similar neurons. The second comparison will use Kendall's tau (definition 4.14) to compare the relative position of each pair of neurons in both ordered lists. A parametric similarity measure will be considered successful at finding the most similar neurons if 75% of the predicted most similar neurons are also most similar according to the behavioral similarity measures. A parametric similarity measure will be considered successful at pairwise ordering of neurons if its pairwise orderings of agree with those made using behavioral similarity 75% of the time. A 95% confidence interval (definition 4.11) will be used to determine whether results are considered to lie above or below these bounds.

Definition 4.11. *95% confidence interval*: The range in which the true mean lies with 95% certainty. The general expression for the 95% confidence limits (Daly, 1998) is:

$$statistic \pm 1.96 * SE(statistic)$$

For the usage in this paper this translates to $mean \pm 1.96 * SE$, where SE refers to the standard error.

4.2 Results & Discussion

4.2.1 Simset intersection

The simset intersection (definition 4.12) is a measure of how well two sorting methods agree on the most similar neurons. It yields a number between 0 and 1, where 0 means no agreement and 1 means full agreement. A parametric similarity measure will be considered successful at finding the most similar neurons if 75% of the predicted most similar neurons are also most similar according to the behavioral similarity measures.

Definition 4.12. *Simset intersection*: A measure of correspondence between two orderings. It takes lists of neurons ordered by similarity, list B and list P, and generates the sets containing the 20% most similar neurons, B_{sim} and P_{sim}. The simset intersection is then calculated as the intersection of the two sets divided by their size.

Example 4.13. *Simset intersection*: Say we have ordered 25 neurons both by behavior (list B) and parameters (list P). We now create two sets by taking the 5 neurons that are most similar to our comparison neuron from both lists, $B_{sim} = \{a, b, c, d, e\}, P_{sim} = \{b, f, a, d, g\}$. The intersection of these lists is $\{a, b, d\}$, which has a length of three. Here, the simset intersection equals 3/5

The tables below show how sorting by different parametric similarity measures (rows) compare with sorting by certain behavioral similarity measures (columns). The tables below show the results of the neuron distance measure tests for individual neurons formatted as: mean ± standard error

Simset intersection, two-input neurons

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.753 ± 0.020	0.747 ± 0.020	0.745 ± 0.020
Sqeuclidian	0.724 ± 0.018	0.725 ± 0.018	0.718 ± 0.019
Canberra	0.725 ± 0.022	0.727 ± 0.022	0.713 ± 0.023
Cosine	0.776 ± 0.016	**0.778 ± 0.014**	0.765 ± 0.017
Hybrid: Cos-abs	**0.833 ± 0.023**	**0.820 ± 0.024**	**0.822 ± 0.024**
Hybrid: Cos-sq	**0.820 ± 0.016**	**0.811 ± 0.016**	**0.802 ± 0.018**

Table 1: This table shows the simset intersection (definition 4.12) between the ordered lists generated by the behavioral and parametric distance measures for the neurons with 2 inputs. Values are shown as mean ± standard error. Performance significantly above 75% is shown bold.

The parametric similarity methods based on the hybrid distance measures perform significantly (95% confidence interval, definition 4.11) above 75% accuracy in predicting behavioral similarity (however measured). The parametric similarity method based on the cosine distance also performs significantly above 75% accuracy, although only when the behavioral similarity is measured using the squeclidian (definition 4.4) distance. All other methods' results are inconclusive, 75% lies within the 95% confidence interval for each of those results.

Simset intersection, ten-input neurons

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.436 ± 0.013	0.432 ± 0.010	0.430 ± 0.013
Sqeuclidian	0.443 ± 0.011	0.447 ± 0.010	0.436 ± 0.011
Canberra	0.381 ± 0.013	0.380 ± 0.013	0.372 ± 0.013
Cosine	0.441 ± 0.013	0.440 ± 0.012	0.434 ± 0.012
Hybrid: Cos-abs	0.450 ± 0.012	0.451 ± 0.011	0.443 ± 0.012
Hybrid: Cos-sq	0.449 ± 0.012	0.449 ± 0.011	0.442 ± 0.012

Table 2: This table shows the simset intersection (definition 4.12) between the ordered lists generated by the behavioral and parametric distance measures for the neurons with 10 inputs. Values are shown as mean ± standard error.

All parametric similarity measures perform significantly (95% confidence interval, definition 4.11) below the 75% accuracy that was aimed for. This means no method is considered successful at predicting the behavioral similarity of neurons with 10 inputs.

4.2.2 Kendall's tau

Kendall's tau (definition 4.14) gives a measure for how well the total orderings compare to one another. It compares each pair that can be found in the orderings. Pairs can be concordant, meaning they agree, or discordant, meaning they disagree. Kendall's tau is, for as far as we're concerned here, calculated as the number of concordant pairs (C) minus the number of discordant pairs (D), divided by the total number of pairs (N): $(C-D)/N$. Kendall's tau lies between -1 and 1, meaning complete disagreement and complete agreement respectively. A 75% accuracy in pairwise ordering corresponds to Kendall's tau equalling 0.5.

Definition 4.14. *Kendall's tau* (Kendall, 1945): A measure of correspondence between two rankings. For the ordered lists that occur in this paper, Kendall's tau (tau) reduces to:

$$tau = \frac{C-D}{C+D}$$

Were C is the number of concordant (agreeing) pairs, and D the number of discordant (disagreeing pairs)

Example 4.15. *Kendall's tau*: Let's compare two orderings: $[1 > 2 > 3]$ and $[1 > 3 > 2]$, where $>$ means 'more alike to our comparison neuron'. As we have three neurons, we have three pairs: $\{1, 2\}$, $\{1, 3\}$ and $\{2, 3\}$. Our concordant (agreeing) pairs are: $\{1, 2\}$ and $\{1, 3\}$. We have one discordant (disagreeing) pair: $\{2, 3\}$. Thus Kendall's tau equals $(2-1)/3 = 1/3$.

The tables below show how sorting by different parametric similarity measures (rows) compare with sorting by certain behavioral similarity measures (columns). The tables below show the results of the neuron distance measure tests for individual neurons formatted as: mean ± standard error.

Kendall's tau, two-input neurons

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.213 ± 0.038	0.217 ± 0.034	0.233 ± 0.041
Sqeuclidian	0.245 ± 0.041	0.215 ± 0.036	0.212 ± 0.039
Canberra	0.220 ± 0.035	0.208 ± 0.033	0.219 ± 0.033
Cosine	0.198 ± 0.030	0.208 ± 0.032	0.185 ± 0.025
Hybrid: Cos-abs	0.221 ± 0.034	0.266 ± 0.041	0.183 ± 0.027
Hybrid: Cos-sq	0.214 ± 0.033	0.242 ± 0.036	0.176 ± 0.024

Table 3: This table shows the Kendall's tau (definition 4.14) for the ordered lists generated by the behavioral and parametric distance measures for the neurons with 2 inputs. Values are shown as mean ± standard error.

Kendall's tau, ten-input neurons

Output Distance / *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.218 ± 0.029	0.216 ± 0.028	0.215 ± 0.029
Sqeuclidian	0.233 ± 0.028	0.230 ± 0.028	0.230 ± 0.029
Canberra	0.227 ± 0.030	0.232 ± 0.030	0.234 ± 0.031
Cosine	0.236 ± 0.030	0.227 ± 0.031	0.213 ± 0.028
Hybrid: Cos-abs	0.233 ± 0.031	0.234 ± 0.031	0.232 ± 0.031
Hybrid: Cos-sq	0.251 ± 0.031	0.224 ± 0.029	0.255 ± 0.032

Table 4: This table shows the Kendall's tau (definition 4.14) for the ordered lists generated by the behavioral and parametric distance measures for the neurons with 10 inputs. Values are shown as mean ± standard error.

Kendall's tau is measured to be significantly (95% confidence interval, definition 4.11) below 0.5 in all instances. This implies that none of the parametric similarity measures are a successful basis for pairwise ordering of neurons. The highest measure of Kendall's tau, in all experiments, lies around 0.25. A Kendall's tau of 0.25 corresponds to 5/8, or 62.5%, of all pairs being concordant (pairs both sorts agree on).

4.3 Conclusion

In this section we set out to answer if we could predict which of two neurons would behave more similar to a third neuron by looking at their parameters alone (subquestion 1). A consistently low value for Kendall's tau implies we cannot do this effectively. A less stringent requirement for our crossover operator, predicting if two neurons are similar, was however met with reasonable success for 2 input neurons. This may be enough for our crossover operator to function. Best performance was achieved using the hybrid similarity measures, of the non-hybrid measures the cosine distance performed best. In the next section this experiment shall be repeated using neurons as part of network instead of single neuron networks.

5 Distance Measure - Neuron in Network

In the former section we looked at predicting the behavior of individual neurons based on their parameters. The aim now is to find out if these predictions hold when a neuron is situated in an organised (trained) network. In this section the same behavioral and parametric similarity measures as in the last (section 4.1.1) are applied to predict the impact of changing individual neurons in trained networks.

5.1 Methods

Networks consisting of an input layer with 2 nodes, a second (hidden) layer with 15 neurons, and an output layer containing single neuron (simple net, figure 9 on the right) will be trained on both the two spirals (definition 5.2) and the checkerboard (definition 5.3) problem (shown in figure 10). A neuron, chosen randomly from the hidden layer, will then be replaced by others to determine the accuracy of the similarity-predictions made by our measures for trained networks.

Definition 5.1. *Simple net*: A small multilayer perceptron network consisting of the following three layers:

1. An input layer containing 2 input nodes
2. A hidden layer containing 15 neurons
3. An output layer containing 1 neuron

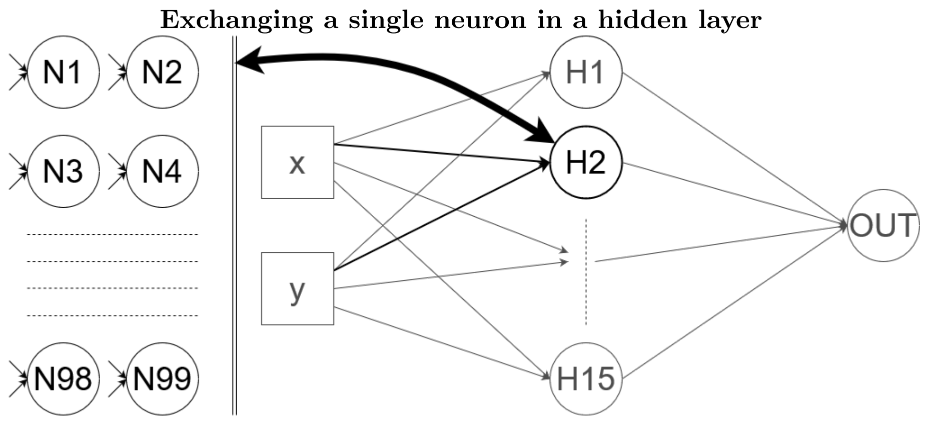

Figure 9: Neurons with randomly generated parameters (left), and a network with one hidden layer containing 15 neurons (right). In the experiment one of the hidden neurons (here: H2) is chosen at random to be the comparison neuron, meaning it is exchanged with the random neurons from the left.

Definition 5.2. *Two-spirals dataset*: A dataset for a classification problem where two differently classified spirals are entwined. Shown in figure 10 on the left.

Definition 5.3. *Checkerboard dataset*: A detaset of a classifiaction problem where two differetly classified tiles are alternated both horizontally and vertically. Shown in figure 10 on the right.

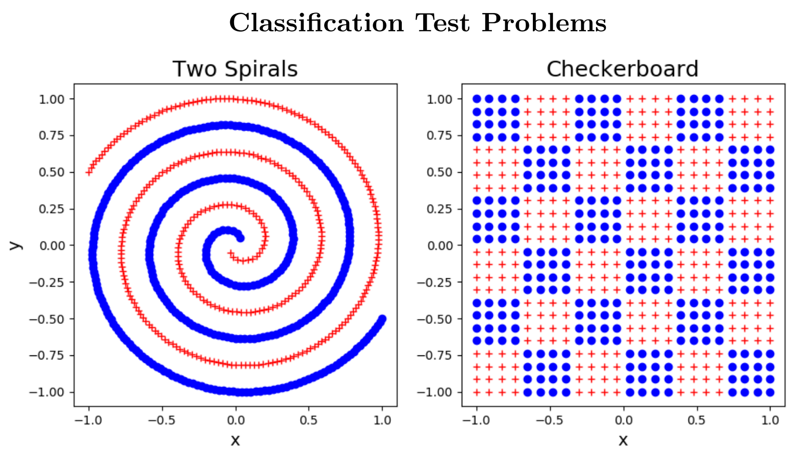

Figure 10: Two classification problems on which to test applications of similarity measures

5.2 Results & Discussion

This section is divided in two parts corresponding to the two methods used for the comparison of the ordered lists. First the results for the simset intersection (definition 4.12) are discussed, followed by those for Kendall's tau (definition 4.14). Predicting the effects of different neurons within a network is expected to be more difficult than predicting that of individual neurons. As such the performance that is considered adequate is lowered to 70% accuracy (from 75% in the former experiment) for both predicting the most similar neurons and comparing neuron pairs. In order to decide if these conditions are met the 95% confidence interval (definition 4.11) is used.

5.2.1 Simset intersection

The simset intersection measures how well the 20% most similar neurons coincide between our parametric and behavioral similarity (definitions 4.2 and 4.1) measures. The tables below show the results of the neuron distance measure tests for trained networks where a single neuron is replaced formatted as: mean $\pm$ standard error.

Simset intersection - two-spiral dataset

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.622 ± 0.015	0.648 ± 0.016	0.618 ± 0.015
Sqeuclidian	0.621 ± 0.015	0.646 ± 0.016	0.615 ± 0.015
Canberra	0.559 ± 0.011	0.569 ± 0.011	0.541 ± 0.011
Cosine	0.665 ± 0.014	0.687 ± 0.014	0.643 ± 0.014
Hybrid: Cos-abs	0.714 ± 0.013	**0.743 ± 0.013**	0.699 ± 0.013
Hybrid: Cos-sq	0.695 ± 0.014	0.719 ± 0.014	0.676 ± 0.014

Table 5: This table shows the simset intersection (definition 4.12) between the ordered lists generated by the behavioral and parametric distance measures. Networks were trained with and outputs were measured using inputs from the two-spirals data set (definition 5.2). Values are shown as mean ± standard error. Performance significantly above 70% is shown bold.

Simset intersection - checkerboard dataset

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.644 ± 0.011	0.649 ± 0.011	0.625 ± 0.010
Sqeuclidian	0.649 ± 0.011	0.654 ± 0.011	0.632 ± 0.010
Canberra	0.588 ± 0.010	0.588 ± 0.010	0.575 ± 0.010
Cosine	0.675 ± 0.011	0.675 ± 0.011	0.659 ± 0.011
Hybrid: Cos-abs	**0.723 ± 0.011**	**0.727 ± 0.011**	0.703 ± 0.011
Hybrid: Cos-sq	0.703 ± 0.011	0.706 ± 0.011	0.685 ± 0.010

Table 6: This table shows the simset intersection (definition 4.12) between the ordered lists generated by the behavioral and parametric distance measures. Networks were trained with and outputs were measured using inputs from the checkerboard data set (definition 5.3). Values are shown as mean ± standard error. Performance significantly above 70% is shown bold.

The cos-abs hybrid distance measure (definition 4.9) is the only measure with which results significantly (95% confidence interval, definition 4.11) above 70% accuracy are reached. The other hybrid, cos-sq (definition 4.10), results in inconclusive results. The non-hybrid measures perform significantly below the 70% accuracy at which they would be deemed successful. Except for one instance where the cosine method of measuring parametric similarity could predict the impact of changing neurons in a network that has been trained on the two-spiral problem, when this is measured by the squeclidian method. Generally, the cosine, squeclidian and cityblock distance yield very comparable results. The Canberra distance method performs significantly worse than the others.

5.2.2 Kendall's tau

Here the ordered lists resulting from our parametric (definition 4.2) and behavioral (definition 4.1) similarity measures will be compared using Kendall's tau (definition 4.14). A 70% accuracy in in pairwise ordering corresponds to Kendall's tau equalling 0.4.

Kendall's tau, two-spiral dataset

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.223 ± 0.029	0.206 ± 0.027	0.231 ± 0.029
Sqeuclidian	0.201 ± 0.025	0.208 ± 0.027	0.208 ± 0.028
Canberra	0.183 ± 0.027	0.199 ± 0.027	0.200 ± 0.027
Cosine	0.195 ± 0.025	0.206 ± 0.027	0.222 ± 0.029
Hybrid: Cos-abs	0.198 ± 0.026	0.206 ± 0.027	0.195 ± 0.025
Hybrid: Cos-sq	0.211 ± 0.027	0.208 ± 0.026	0.191 ± 0.026

Table 7: This table shows Kendall's tau (definition 4.14) for the ordered lists generated by the behavioral and parametric distance measures. Networks were trained with and outputs were measured using inputs from the two-spirals data set (definition 5.2). Values are shown as mean ± standard error.

Kendall's tau, checkerboard dataset

Output Distance *Neuron distance*	*Cityblock*	*Sqeuclidian*	*Categorization*
Cityblock	0.217 ± 0.028	0.195 ± 0.026	0.203 ± 0.026
Sqeuclidian	0.208 ± 0.028	0.211 ± 0.027	0.187 ± 0.025
Canberra	0.173 ± 0.023	0.172 ± 0.022	0.186 ± 0.026
Cosine	0.195 ± 0.024	0.191 ± 0.024	0.185 ± 0.024
Hybrid: Cos-abs	0.185 ± 0.024	0.177 ± 0.022	0.187 ± 0.025
Hybrid: Cos-sq	0.206 ± 0.027	0.184 ± 0.024	0.199 ± 0.025

Table 8: This table shows Kendall's tau (definition 4.14) for the ordered lists generated by the behavioral and parametric distance measures. Networks were trained with and outputs were measured using inputs from the checkerboard data set (definition 5.3). Values are shown as mean ± standard error.

Kendall's tau is measured to be significantly (95% confidence interval, definition 4.11) below 0.4 in all instances. This implies that non of the parametric similarity measures are a successful basis for pairwise ordering of neurons that are situated in a network. The methods that resulted in a Kendall's tau of 0.25 before, now yield approximately 0.2. This means that the amount of correct pairwise comparisons has dropped to around 60% (from 62.5%).

5.3 Conclusion

We set out to verify the results from the last section on neurons in trained networks (subquestion 2). Although generally performance was a bit worse that in single neuron networks, the results for this experiment tell much the same picture as the last. Kendall's tau was still consistently low, implying that the measures are not very effective at differentiating behavioral similarity of any two neurons to a third. The results for the simset intersection however imply that a simple prediction of similarity is still possible for neurons in a network.

The hybrid measures are the best performing measures in predicting the 20% most similar neurons (simset intersection, definition 4.12). The cosine, cityblock and sqeuclidian form the middle of the pack of parametric similarity measurements, where the cosine distance pulls ahead of the other two. It seems predicting neuron similarity by relative differences in parameters is inefficient. These findings hold irrespective of the way behavioral similarity is measured.

6 Designing Crossover Operators

This section is aimed at researching if the predictive distance measures can be used as a basis for a non-disruptive crossover operator (the third subquestion). Standard crossover operators will be compared with newly designed operators that use distance measures to estimate behavioral similarity. Non-disruptiveness will be tested by measuring the crossover correlation coefficient (definition 3.4) of these different crossover operators on a set of random parents. The new crossover operators will be defined at the start of this section, followed by a discussion of the experimental results. Then the behavior of the more complex operator (analogue cross, definition 6.6) will be investigated further. At the end of the discussion a seeming replication error will be investigated, gaining some insights in the complexities of crossovers on neural networks. Our conclusions will be reiterated at the end of this section.

6.1 Methods

A number of crossover operators will be defined. Operators using the similarity measures will be tested for non-redundancy and standard operators will be used for comparison. These operators are tested on both random networks as generated, and these same networks in their non-redundant encoding (definition 3.3). In this experiment the simple net (definition 5.1) is used, meaning the networks have an input layer containing 2 nodes, a hidden layer containing 15 nodes and a single output node. The crossover correlation coefficient (definition 3.4) is used as the metric to determine the quality of the crossover operators. The two-spiral and checkerboard classification problems, shown in figure 10 are used for this testing.

After this we graphically compare analogue crossover with uniform crossover. For this we use the relation between the cluster distance for the analogue operator and the amount of locations in which crossover will occur.

Example 6.1. *Analogue vs uniform crossover experiment*: A single data point is collected as follows: a cluster distance of 0.2 may cause crossovers in, on average, 9 locations. We then compare the crossover correlation coefficient of analogue crossover with a cluster distance of 0.2 with uniform crossover on 9 locations.

First we define the standard crossovers that will be used for comparison:

Definition 6.2. *Halfway crossover*: This operator cuts the two parent layers down the middle. Children are then generated by exchanging the second half of both parents' layers.

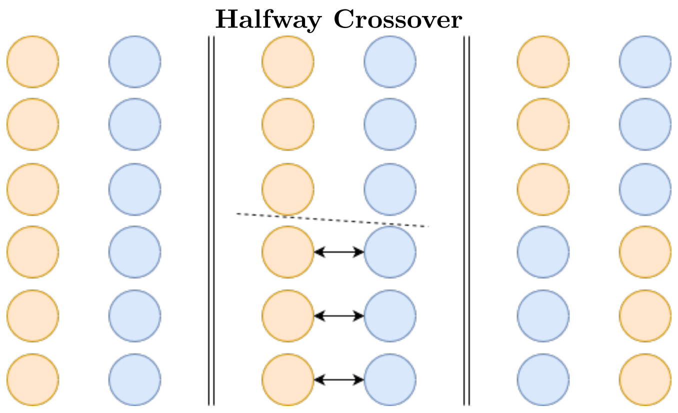

Figure 11: Two parent genomes (left) are both 'cut in half' (center). The children are then created by combining the different halves of the parents.

Definition 6.3. *Uniform crossover*: This operator generates a list of neuron locations of half the length of the number of neurons. Children are generated by exchanging the neurons in these locations.

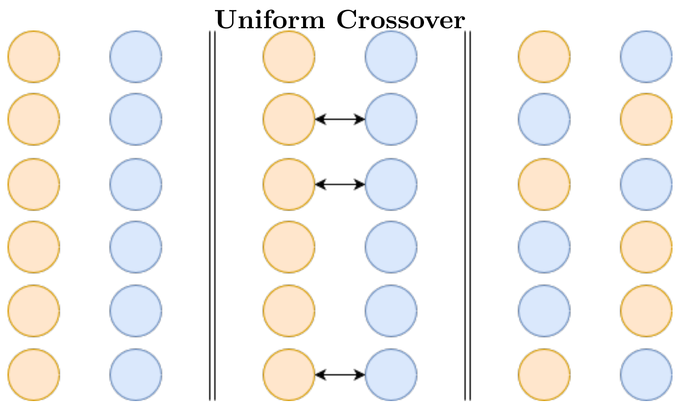

Figure 12: Half of the neurons from the parents (left) are randomly selected (middle) to be exchanged with one another to form the child genomes (right).

Definition 6.4. *Partial uniform crossover*: This operator generates a list of neuron locations of half the length of a given number. Children are generated by exchanging the neurons in these locations. If this number is set to the total number of neurons, this operator behaves the same as uniform crossover (definition 6.3).

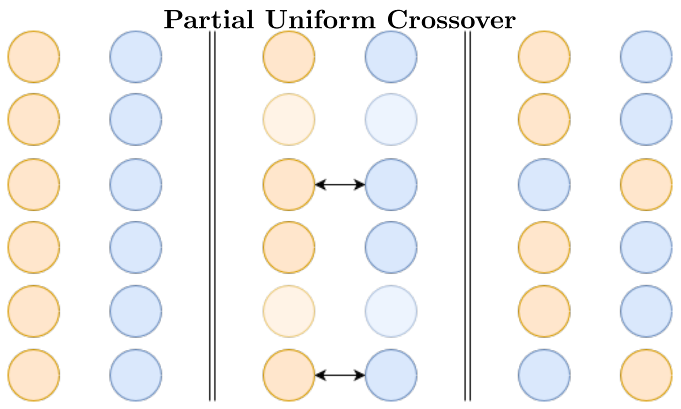

Figure 13: Shown here is partial uniform crossover on 4 neurons. Of the parent genomes (right), 4 neurons are randomly selected to be eligible for crossover (non-selected neurons shown opaque). Of the selected neurons, half are randomly selected to be exchanged. Resulting in the new children (right)

Following are the definitions of the newly designed crossover operators. These will use the parametric distance measures in two ways. The mindist crossover (definition 6.5) generates a mapping of minimal distance between two parents, which is used as a basis for uniform crossover. The analogue crossover (definition

6.6) lends its name from analogues, meaning similar in activity, genes in biology. This operator uses the distance measures to create sets of similar neurons, which are then randomly permuted.

Definition 6.5. *Mindist crossover*: This operator uses a distance measure to find the pairs between the neurons from both parents that are most similar while using each neuron once. Crossover is applied to these pairs uniformly. This is a three step process:

1. Generate the distance matrix for all neurons in parent1 to all neurons in parent2
2. Create a mapping between the neurons from the two parents such that the total distance is minimized, using the Hungarian algorithm (Kuhn, 1955).
3. Apply uniform crossover (definition 6.3) on the mapped pairs.

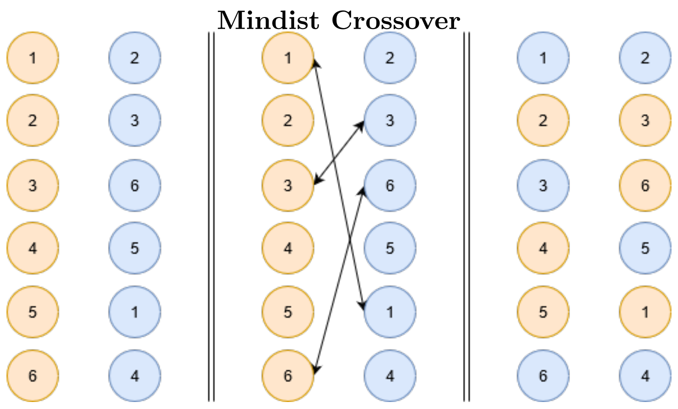

Figure 14: A minimal distance mapping is generated between the two parent genomes (left). Half the neurons are randomly selected for crossover with their corresponding neuron from the other parent (middle). Exchanging these neurons yields the new child genomes (right).

Definition 6.6. *Analogue crossover*: This operator uses a distance measure to generate sets of neurons such that no two neurons in a set can be more than a certain distance from one another. It then randomly permutes neurons within these sets. This is done in a three step process:

1. Generate the distance matrix for the complete set of neurons (thus from both parents)
2. Group neurons into sets using the distance matrix and some maximum distance, using farthest point linkage. Farthest point was chosen because no neurons within a set should be very different from one another.

3. For each set that contains at least one neuron in both parents, randomly permuted the neurons in that set.

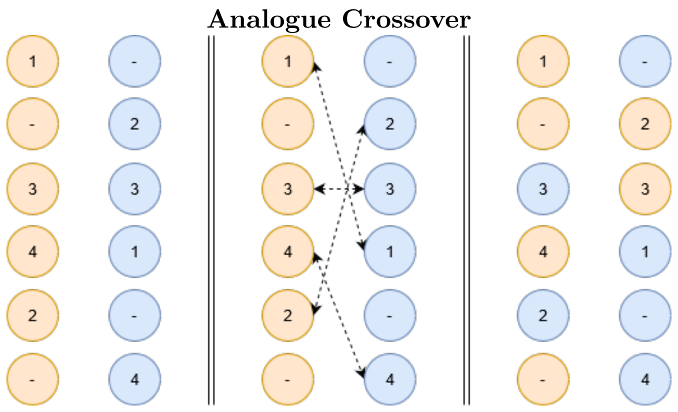

Figure 15: Sets of similar neurons are generated from the set of all neurons (left), not all neurons necessarily belong to a set. Neurons within each set are randomly permuted, this may result in the same permutation as they already were (center). Resulting in new child genomes (right).

In this experiment, the following distance measures (defined in section 4.1.1) are used by our crossover operators:

- Cityblock distance (definition 4.3)
- Cosine distance (definition 4.8)
- Cos-abs distance (definition 4.9)
- Cos-sq distance (definition 4.10)

6.2 Results & Discussion

The tables below show the crossover correlation coefficients (definition 3.4) for various crossover operators formatted as: mean ± standard error

Crossover Correlation Coefficients - redundant networks, two-spiral dataset

Neuron distance / *Crossover Method*	*Cityblock*	*Cosine*	*Hybrid: Cos-Abs*	*Hybrid: Cos-Sq*
Mindist	0.620 ± 0.015	0.686 ± 0.012	0.611 ± 0.016	0.615 ± 0.015
Analogue (12 clust)	0.676 ± 0.014	0.745 ± 0.011	0.693 ± 0.012	0.647 ± 0.014
Halfway	0.296 ± 0.019			
Uniform	0.310 ± 0.019			
Uniform (12 clust)	0.443 ± 0.018			

Table 9: Shown are the crossover correlation coefficients (definition 3.4) for the novel crossover methods per similarity measure and for the standard crossover methods. Measurements were made by crossing over random parents, and measuring performance on the two-spiral data set (definition 5.2). Values are shown as mean ± standard error.

Crossover Correlation Coefficients - non-redundant networks, two-spiral dataset

Neuron distance / *Crossover Method*	*Cityblock*	*Cosine*	*Hybrid: Cos-Abs*	*Hybrid: Cos-Sq*
Mindist	0.696 ± 0.013	0.750 ± 0.011	0.670 ± 0.012	0.683 ± 0.013
Analogue (12 clust)	0.745 ± 0.012	0.819 ± 0.010	0.743 ± 0.010	0.757 ± 0.010
Halfway	0.269 ± 0.019			
Uniform	0.426 ± 0.020			
Uniform (12 clust)	0.541 ± 0.019			

Table 10: Shown are the crossover correlation coefficients (definition 3.4) for the novel crossover methods per similarity measure and for the standard crossover methods. Measurements were made by crossing over non-redundant (definition 3.3) parents, and measuring performance on the two-spiral data set (definition 5.2). Values are shown as mean ± standard error.

Crossover Correlation Coefficients - redundant networks, checkerboard dataset

Neuron distance / *Crossover Method*	*Cityblock*	*Cosine*	*Hybrid: Cos-Abs*	*Hybrid: Cos-Sq*
Mindist	0.305 ± 0.021	0.368 ± 0.021	0.319 ± 0.023	0.291 ± 0.021
Analogue (12 clust)	0.431 ± 0.020	0.474 ± 0.019	0.431 ± 0.019	0.364 ± 0.019
Halfway	0.103 ± 0.023			
Uniform	0.092 ± 0.021			
Uniform (12 clust)	0.203 ± 0.018			

Table 11: Shown are the crossover correlation coefficients (definition 3.4) for the novel crossover methods per similarity measure and for the standard crossover methods. Measurements were made by crossing over random parents, and measuring performance on the checkerboard data set (definition 5.3). Values are shown as mean ± standard error.

Crossover Correlation Coefficients - non-redundant networks, checkerboard dataset

Neuron distance / *Crossover Method*	*Cityblock*	*Cosine*	*Hybrid: Cos-Abs*	*Hybrid: Cos-Sq*
Mindist	0.463 ± 0.018	0.456 ± 0.019	0.481 ± 0.017	0.442 ± 0.019
Analogue (12 clust)	0.517 ± 0.016	0.577 ± 0.017	0.492 ± 0.019	0.511 ± 0.019
Halfway	0.336 ± 0.020			
Uniform	0.376 ± 0.021			
Uniform (12 clust)	0.426 ± 0.020			

Table 12: Shown are the crossover correlation coefficients (definition 3.4) for the novel crossover methods per similarity measure and for the standard crossover methods. Measurements were made by crossing over non-redundant (definition 3.3) parents, and measuring performance on the checkerboard data set (definition 5.3). Values are shown as mean ± standard error.

The halfway crossover on redundant networks is equivalent to the uniform crossover, as the neurons are permuted randomly. It therefore is no surprise that the crossover correlation coefficients of these two seem to come from the same distribution. For non-redundant networks this difference differs per problem and will be discussed later.
Minimal distance crossover outperforms non-distance-dependent crossovers in terms of crossover correlation coefficient. This implies it pairs similar neurons more accurately than random (as in redundant encodings) and Thierens' non-redundant encoding (Thierens, 1996).

Parametric similarity measures
The cosine distance almost always results in the best performance for the distance-based crossover operators. The cityblock and cos-abs hybrid distances do not perform significantly differently from one another, their performance generally lying within each others standard errors. The Cosine-Squared Hybrid often performs worst of the metrics, sometimes performing equally to the absolute distance and its hybrid counterpart. This performance is a deviation from the former experiment, where the hybrid methods yielded the highest performance.

(non-)redundancy
Writing networks in their non-redundant form (definition 3.3) improves the crossover correlation coefficient across the board. This could be caused by it being easier to match similar neurons when they are written in a standardized form. Another explanation is that there are more similar neurons in non-redundant networks. For example, say we have neurons, f and f', such that f(inputs) = -f'(inputs). As for now, crossover on these two neurons will be quite disruptive. We can however change f' to f without changing the network's functionality, if we change all outgoing weights of f' to be their negative counterpart as well. After changing f' to f we can obviously apply crossover on both versions of f without this being disruptive to the network's functionality. This is exactly

what Thierens' non-redundant encoding does (Thierens, 1996). Figure 16 shows that there are more crossover locations (implying more similar neurons) for any cluster distance in non-redundant networks compared to their redundant counterparts.

dataset-dependent results
The crossover correlation coefficients in the test on the checkerboard dataset are lower than those on the two-spirals dataset. This implies that the fitness landscape for multilayer perceptrons on the checkerboard dataset appears less correlated to the crossover operator than the fitness landscape corresponding to the two-spiral dataset. The crossover operator defines a way to move through the fitness landscape. If this movement results in relatively well functioning children for relatively well-functioning parents, then this results in a high crossover correlation coefficient. Whether this happens may not solely depend on the movement dictated by the operator, but it may also depend on the shape of the fitness landscape itself. A possible explanation for the observed difference in CCC between the two datasets is that the movement through the fitness landscape, determined by the crossover operator, corresponds better to the fitness landscape of the two-spiral dataset than that of the checkerboard dataset.

It is also interesting that on the two-spiral dataset the halfway crossover is outperformed by the uniform crossover (on non-redundant networks of course, for redundant networks these operators are equivalent). A performance worse than random implies that some set of neurons which is relevant to the networks' performance is often taken apart. This implies that, at least with respect to some data sets, neurons with biases that are relatively far apart from one another interact in some way and should not be automatically separated by our operator.

6.2.1 Graphical Comparison

In this section we see five plots aiding in comparing the crossover correlation coefficient of analogue crossover with that of uniform crossover. The first plot shows the relation between the cluster distance for the analogue crossover and the number of crossover locations. The other four plots show the crossover correlation coefficients of the analogue crossover operator and the uniform crossover operator with respect to the number of crossover locations. Cosine distance was used as the distance measure for the crossover operator. The crossover correlation coefficient achieved by the minimal distance crossover is also shown.

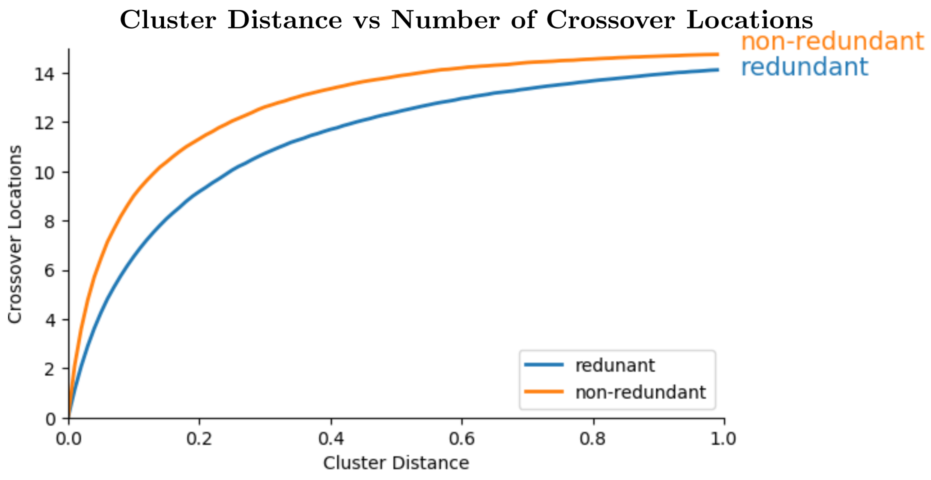

Figure 16: The data plotted above shows the cluster distance vs the average number of crossovers performed during analogue crossover using the cosine distance metric. Data was gathered from networks with random weights in [-10, 10) and their non-redundant (Thierens, 1996) counterparts.

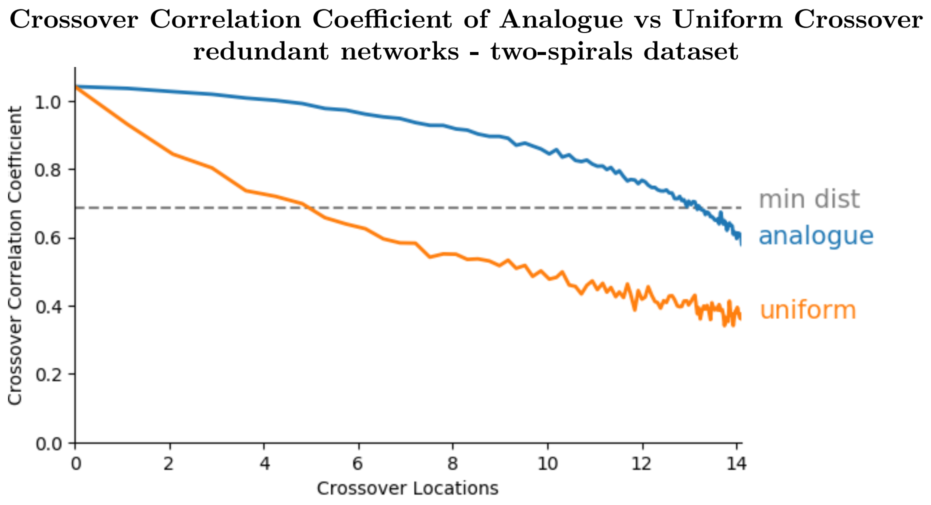

Figure 17: The data plotted above was generated by applying the respective crossovers to *redundant networks*. Performance was measured as the performance on the *two-spirals dataset*.

Crossover Correlation Coefficient of Analogue vs Uniform Crossover non-redundant networks - two-spirals dataset

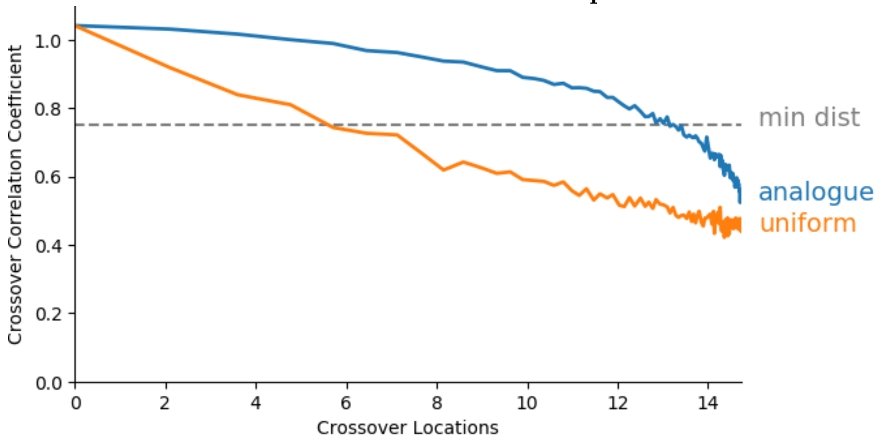

Figure 18: The data plotted above was generated by applying the respective crossovers to *non-redundant networks*. Performance was measured as the performance on the *two-spirals dataset*.

Crossover Correlation Coefficient of Analogue vs Uniform Crossover redundant networks - checkerboard dataset

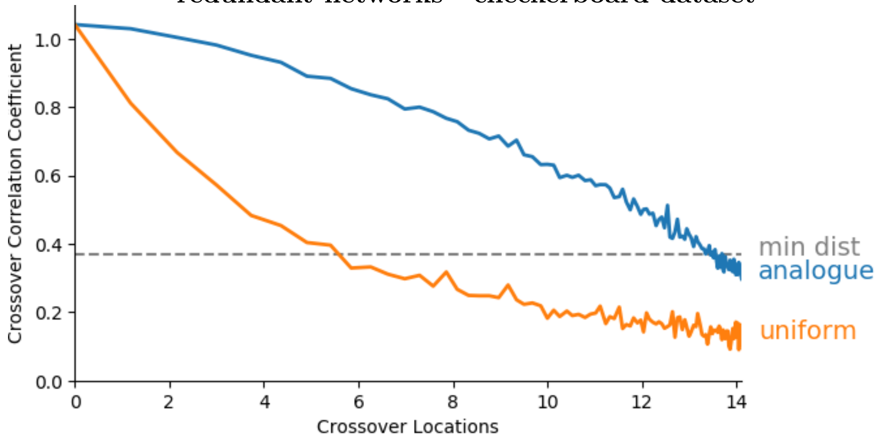

Figure 19: The data plotted above was generated by applying the respective crossovers to *redundant networks*. Performance was measured as the performance on the *checkerboard dataset*.

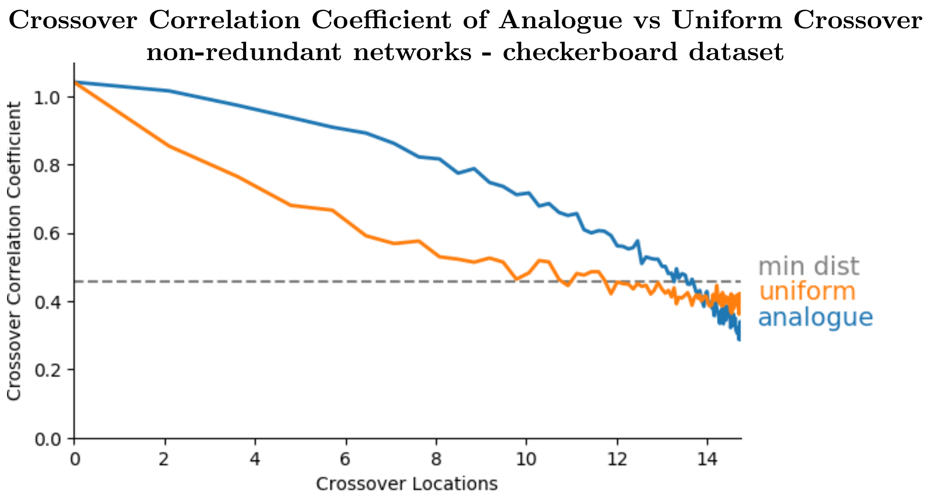

Figure 20: The data plotted above was generated by applying the respective crossovers to *non-redundant networks*. Performance was measured as the performance on the *checkerboard dataset*.

So long as the distances within sets of neurons are kept small, analogous crossover results in a larger crossover correlation coefficient than minimal distance crossover. The reason behind this is two-fold. Firstly, small distances result in a smaller number of crossover locations as shown in figure 16. This leads to fewer changes in the genome and thus a larger correlation between parents and children, as shown for both analogue and uniform crossover in figures 17, 18, 19 and 20. In these same figures we can also see that the crossover correlation coefficient for analogues crossover has a shallow curve for few crossovers, which gets steeper as more crossovers are performed, which brings us to the second reason for analogue crossover's success. Not only are fewer crossover locations chosen, the locations that are chosen are those where crossover is relatively non-disruptive.

As distances within sets of neurons are set too large, the analogue crossover operator becomes unstable. At some point its performance even drops below that of the minimal distance crossover, even when crossover is not performed on all neurons. Figure 21 shows the difference between the performed crossovers for small and large cluster distances respectively. What is important to note is that for large cluster distances, the sets that are randomly permuted may contain more than two neurons. Firstly, this makes it more likely neurons actually change place (a random permutation of two neurons has only a 50% chance that it's different from the original permutation). Secondly it allows for neurons to be swapped within one parent, which does cause disruption in the networks behaviour, but does not result in information exchange between networks. In the extreme, such as a cosine distance of 2 (the maximum this distance can be) one big cluster will be formed containing all neurons from both parents. In that case the operator will randomly permute all neurons in both parents, which is possibly more disruptive than even uniform crossover. It can therefore be said that the analogue crossover operator does not function properly, and should thus not be used, with large cluster distances.

Analogue Crossover Sets for Small and Large Cluster Distance

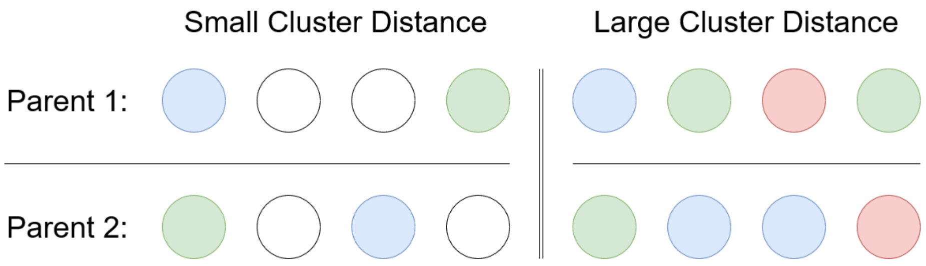

Figure 21: Shown are imagined sets as they could have been made by the analogue crossover operator. Circles represent neurons. Neurons of the same color belong to the same set, empty neurons do not belong to any set. It is noteworthy that for a large cluster distance, the sets can exceed a size of two neurons.

6.2.2 Network parameter scaling, reproduction error?

A notable result from the crossover correlation experiments is that they differ from the results in Thierens' experiment (Thierens, 1996). The code that was used in the original is no longer available, however it seems this difference can be explained by the size of the parameters.

Effect of parameter scaling on crossover correlation coefficient

	Thierens	*Parameters in [-10, 10)*	*Parameters in [-1, 1)*
Redundant	0.456	0.310 ± 0.019	0.356 ± 0.021
Non-redundant	0.892	0.426 ± 0.020	0.912 ± 0.007

Table 13: Crossover Correlation Coefficient for Uniform Crossover (mean ± standard error), Two Spiral Set

As we can see, the results from the original experiment with non-redundant encoding coincide with our replication if we choose the parameters to be in [-1, 1). The range of parameters of [-10, 10) has been chosen as it resembles the parameters of a network that was trained on the classification problems that were discussed. However, it is common practice to use smaller numbers for initializing networks. PyTorch's linear layers (the ones used in our experiments) are initialized from a uniform distribution with a maximum absolute value of $\sqrt{1/N_{inp}}$, where N_{inp} is the number of inputs (PyTorch Contributors, 2019). It is easy to see this number cannot exceed 1.

The effective difference in the behavior of a single neuron when it's parameters' magnitude is increased lies in the sharpness of the decision boundary, as shown in figure 22. It is noteworthy that the location of the decision boundary does not change, because the equation that corresponds to the neuron's behavior equals zero for the same inputs, shown in table 14.

For a network the decision boundary still sharpens with increased parameters, but the boundaries location changes as well, as shown in figure 23. This means we cannot just reduce the parameters in a network linearly as a transformation to improve the crossover correlation coefficient.

Decision boundaries of neurons are unaffected by parameter scaling

Neuron with parameters in [-1, 1)	Neuron with parameters in [-10, 10)
$tanh(x_1 * w_1 + x_2 * w_2 + b) = 0$	$tanh(x_1 * (10 * w_1) + x_2 * (10 * w_2) + (10 * b)) = 0$
-	$x_1 * (10 * w_1) + x_2 * (10 * w_2) + (10 * b) = 0$
$x_1 * w_1 + x_2 * w_2 + b = 0$	$10 * (x_1 * w_1 + x_2 * w_2 + b) = 0$

Table 14: Above is a short derivation showing that the decision boundaries for an individual don't change when parameters are scaled.

Parameter Scaling Single Neuron

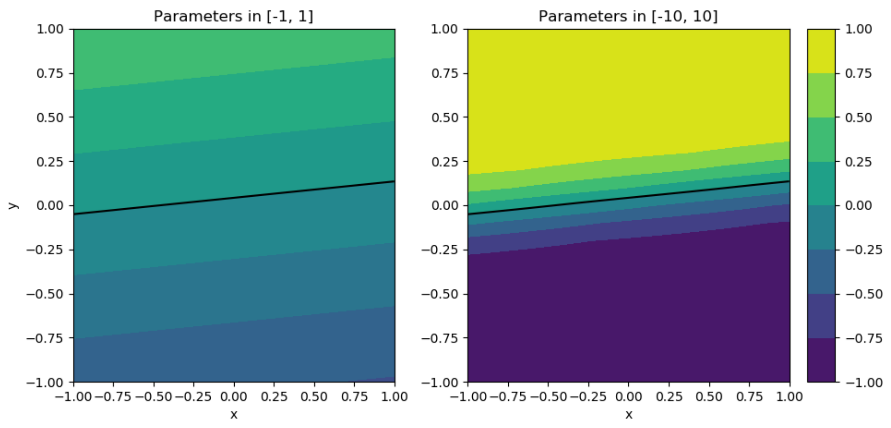

Figure 22: On the left we see the output of a neuron given two inputs, x and y. The neuron uses a hyperbolic tangent as the transfer function, such that it's behaviour is described by: $tanh(x * w_x + y * w_y + b)$. To the right we see the output for this same neuron with its weights (w_x, w_y) and bias (b) multiplied by 10.

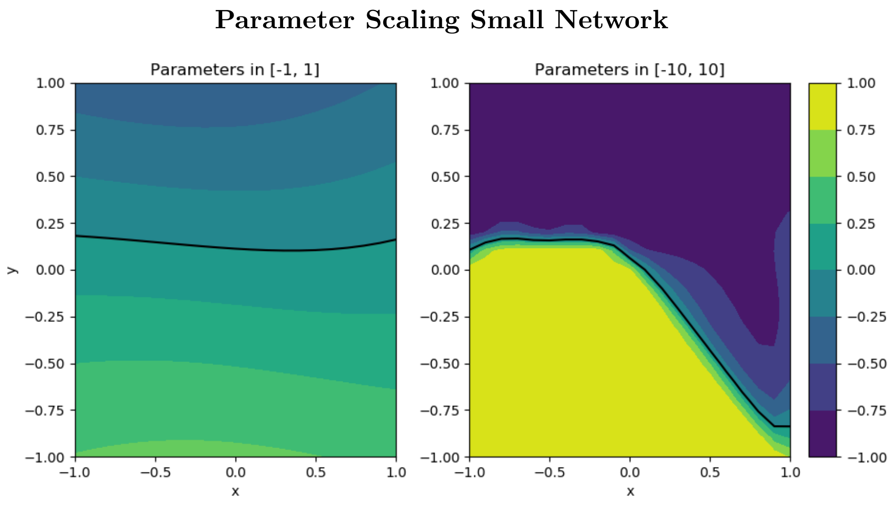

Figure 23: On the left we see the output of a simple network (definition 5.1) given two inputs, x and y. The network uses a hyperbolic tangents as it's transfer function. To the right we see the output for this same network with its weights and biases multiplied by 10.

6.3 Conclusion

Aiming to reduce the disruptiveness of crossover on neural networks, two crossover operators which use the parametric similarity measures have been defined. Both analogue (definition 6.6) and mindist (definition 6.5) crossover have been seen to improve the crossover correlation coefficient, compared to using standard operators. The cosine distance performed best as a similarity measure to base the operator on, outperforming the hybrid methods that seemed best before. This may be due to difficulties in balancing the two parts of the hybrid methods. However, the answer to whether these operators were truly non-disruptive (subquestion 3) is that it unfortunately seems to depend on the data set the operator is applied to. A possible explanation for this could lie in the shape of the fitness landscapes corresponding to the simple network (definition 5.1) and the datasets.

Next different cluster distances for the analogue crossover operator (definition 6.6) were compared with partial uniform crossover (definition 6.4) on an equal amount of neurons. The most important conclusion was that analogue crossover does not function correctly for large cluster distances.

7 Hybrid Genetic Algorithm

In this section we will have a look at the new crossover operators in action. Instead of answering a research question the aim is to gain some insight into whether the new crossovers function in their current form. In the introduction it was discussed that gradient descent is a very well-performing local optimizer, and that the aim for the crossover operator was to aid in global optimization. In this section the two will be combined into what is called a hybrid genetic algorithm.

Definition 7.1. *Hybrid genetic algorithm*: An algorithm that combines a genetic algorithm with another optimizer. In this case the genetic algorithm will solely be represented by our crossover operator, our second optimizer is gradient descent.

7.1 Methods

The training algorithm is simple and can be split in two parts. The first part is aimed at searching the global search space, which is done by alternating short intervals of training by gradient descent with crossover. In the second part the individuals in our population converge to their local optima using gradient descent. This experiment is repeated 10 times to allow for statistical comparisons.

Algorithm 1: Hybrid Genetic Algorithm

```
population = instantiate_population(population_size=40);
for _ in range(10) do
    population = gradient_descent(population, short_interval);
    parents = best_performing_half(population);
    children = crossover(parents);
    population = merge(parents, children);
population = gradient_descent(population, long_interval);
```

Crossover will be done using the following operators:

- No crossover: In this case crossover is skipped.
- Uniform crossover (definition 6.3)
- Mindist crossover (definition 6.5), using the cosine distance as the parametric similarity measure.
- Analog crossover (definition 6.6), using the cosine distance as the parametric similarity measure. This operator will be tested with three cluster distances: 0.15, 0.30 and 0.45.

7.2 Results & Discussion

	Checkerboard		**Two spirals**	
Method	*Mean*	*Sd*	*Mean*	*Sd*
No Cross	0.788	0.011	0.828	0.011
Uniform	0.814	0.022	0.833	0.017
Mindist	0.813	0.015	0.829	0.010
Analog 0.15	0.807	0.019	0.818	0.015
Analog 0.30	0.807	0.015	0.828	0.015
Analog 0.45	0.818	0.024	0.826	0.011

Table 15: The mean and standard deviation (Sd) of the best individuals in all ten experiments for each method.

Comparing the results using Welch's t-test (Welch, 1947) yields that no method performs significantly different from another when applied to the two-spirals dataset. When training on the checkerboard set applying no crossover performs significantly worse ($p < 0.05$) than any other, otherwise no method is significantly different from another.

7.3 Conclusion

The crossover operator may aid convergence of neural networks to a more optimal solution compared to using no crossover operator. There is however no hard evidence to suggest that the crossover operator developed for this thesis is superior to uniform crossover.

8 Conclusion

The goal of this research was to find a way to facilitate a non-disruptive exchange of information between multi-layer perceptron networks (main question, section 2). In order to answer the main question was divided into subquestions and experiments were performed to answer to these (sections 4, 5, and 6). It was concluded that the parametric similarity (definition 4.2) measures did not allow us to predict the ordering of neurons by behavioral similarity (definition 4.1). This resulted in a negative answer to the first two research questions. Nonetheless the parametric similarity measures could still form a strong enough basis for a non-disruptive (definition 2.3) crossover operator. Two such operators, analogue crossover (definition 6.6) and mindist crossover (definition 6.5), were tested in section 6. These operators turned out to be non-disruptive, at least for the two-spiral dataset.

In this section we will recap the conclusions that were drawn from the experiments, using this to answer the main question. At the end of this section ideas for further research on the subject are discussed.

8.1 Finding a distance measure

Section 4 was aimed at determining if, given some base neuron, other neurons' similarity in behaviour (subquestion 1) could be predicted by their parameters. This was approached by comparing lists of neurons ordered based on behavioral similarity with those based on parametric similarity. A consistently low value for Kendall's tau revealed that none of the six distance measures researched are a good basis for predicting the relative differences in neurons' behavior with high accuracy. However, measuring the overlap of similar neurons in both ordered lists (simset intersection, definition 4.12) yielded that the distance measures can distinguish between similar and non-similar neurons. As being able to make this distinction might be a good enough basis for a non-disruptive (definition 2.3) crossover operator, the research was continued.

In section 5 a similar prediction was researched, but this time neurons in trained networks were used instead of individual neurons. Trained networks more closely resemble the environment in which the crossover operator would operate. The results were quite similar to those in the first half, though generally not as good as in section 4. The lower values for Kendall's tau (definition 4.14) and simset intersection (definition 4.12) imply that the effect of a neuron on a network's behaviour is harder to predict. However the answer to the second subquestion is the same as to the first: The parametric similarity measures are not capable of ordering neurons by their similarity effectively, but they are capable of finding the most similar neurons reasonably well. Whether this second capability is enough for a non-disruptive crossover operator was researched next.

8.2 Designing Crossover Operators

In section 6 two crossover operators were defined, mindist and analogue crossover (definitions 6.5 and 6.6). These operators were made to use the distance measures that were researched in the former sections. The goal was to find out if these crossover operators would be non-disruptive (definition 2.3). The crossover correlation coefficients (definition 3.4) indicated that 1) the novel operators outperformed the standard crossover operators, and 2) the cosine distance (definition 4.8) was generally the best distance measure for the crossover operator. However, whether the new crossover operators could be called non-disruptive depended on the dataset. Both operators yielded little disruption with respect to the spirals set, but were quite disruptive with respect to the checkerboard set. This may be caused by the combination of the movement through the fitness landscape, as dictated by the crossover operator, and the shape of the fitness landscape.

The behavior of the analogue crossover was further discussed for different cluster distances. At small distances it would both allow for few crossovers, and the performed crossovers would yield small change. More importantly, for large cluster distances this operator breaks by randomizing neurons in large clusters (Figure 21 on page 37).

8.3 A non-disruptive exchange in information

With our subquestions answered, our main research question can be answered: Is it possible to facilitate a non-disruptive exchange of information between multi-layer perceptron networks by matching similar neurons?

The crux of the answer lies in a way to define a distance measure, indicating similarity of neurons. It was found that putting the weights and bias of neurons into an array, and computing the cosine distance (definition 4.8) between these arrays to be a valid method of computing neuron dissimilarity. This dissimilarity can then be used to 1) compute the minimal distance pairing of two parents (definition 6.5), or 2) generate sets of similar, or analogues, neurons (definition 6.6). We can then apply uniform crossover over these pairings or randomize neurons within sets respectively.

The disruptiveness for analogue and mindist crossover (definitions 6.6 and 6.5), is not the same across all datasets. At least for some (such as the checkerboard set, definition 5.3), applying these crossover operators are relatively disruptive. Measuring disruptiveness using the crossover correlation coefficient (definition 3.4), as done in section 6, is however computationally cheap (relative to training a neural network). As such the viability of these operators can be estimated without much trouble.

8.4 Hybrid Genetic Algorithm

In section 7 it was tested whether our crossover operators truly aided in optimizing neural network parameters. We found that using some crossover operator was an improvement to using no crossover operator when learning the checkerboard dataset. However we did not find a significant difference between any of the other tests. Therefore we conclude that there is no hard evidence that the newly designed crossover operators (mindist and analague) are superior to uniform crossover.

8.5 Further research

- **Scaling both parts of Hybrid Measures:** The hybrid distance measures seemed promising during the first experiments, but where less effective when applied in a crossover operator. This may have to do with balancing the size of the cosine distance with the (squared) bias distance. The cosine distance lies between -1 and 1, whereas the other is unbounded. Finding a way to scale these may result in increased performance for the crossover operator using hybrid distance methods.

- **Parameter Size and Disruptiveness:** We have seen that the size of the parameters in the neural network has an impact on the disruptiveness of the crossover operator (section 6.2.2). Further research is needed if we wish to fully understand why.

- **Fitness landscape of the two datasets:** In section 6 we observed a difference in CCC that depended on the dataset the networks were applied to. A difference in fitness landscape was given as a possible explanation for this. It could be interesting to try mapping the fitness landscapes corresponding to the simple networks (definition 5.1) on these datasets to see if this explanation is valid.

- **More Inputs:** We have seen that increasing the number of inputs for the neurons in our network decreases the predictive capacities of our neuron distance measures (section 4). We have not researched the effects this has on the crossover operator. This is paramount if we wish to apply this crossover to real problems, as they seldom have only two inputs.

- **Large Layers:** It is not uncommon for networks to be designed with layers much larger than what we've seen in this thesis. Research is needed to estimate the effect this has on the crossover operators.

- **layer-wise optimization:** Assuming other hurdles can be cleared, the most promising application of this operator could be greedy layer-wise training (Bengio et al., 2007). As the name suggests this method entails training a network in a layer by layer. Our crossover operator is only applicable if the inputs are the same for our population, which fits quite well with this method.

References

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

Daly, L. E. (1998). Confidence limits made easy: interval estimation using a substitution method. *American Journal of Epidemiology*, 147(8):783–790.

García-Pedrajas, N., Ortiz-Boyer, D., and Hervás-Martínez, C. (2006). An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization. *Neural Networks*, 19(4):514–528.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population based training of neural networks.

Jansen, T. and Wegener, I. (2002). The analysis of evolutionary algorithms–a proof that crossover really can help. *Algorithmica*, 34(1):47–66.

Kendall, M. G. (1945). The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Manderick, B., de Weger, M. K., and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. In *International Conference on Genetic Algorithms*.

McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *International Conference on Genetic Algorithms*, volume 89, pages 379–384.

Montana, D. J. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767.

Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Ortiz-Boyer, D., Hervás-Martínez, C., and García-Pedrajas, N. (2007). Improving crossover operator for real-coded genetic algorithms using virtual parents. *Journal of Heuristics*, 13(3):265–314.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

PyTorch Contributors (2019). torch.nn - pytorch master documentation. `https://pytorch.org/docs/stable/nn.html`. [Online; accessed 2020-26-03].

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Thierens, D. (1996). Non-redundant genetic coding of neural networks. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 571–575. IEEE.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Van der Plas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Welch, B. L. (1947). The generalization ofstudent's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35.