Hochschule Rhein-Waal

Evolutionary algorithms and pruning: comparison of three after-prune training methods

Applied Research Project

by
Martín Alejandro Gorosito
26567
Martin-alejandro.gorosito@hsrw.org

Supervisors:
Prof. Dr. Achim Kehrein
Prof. Dr. Matthias Krauledat

Winter Semester 2021

1 CONTENTS

2 ABSTRACT

Designing and developing a neural network can be challenging as there are many parameters and design decisions that have to be made. Pruning is one of these decisions, and one that is taking more relevance for its ability to reduce the network's overall size and cost and increase its speed. However, the lack of processes for pruning makes the whole venture a matter of trial and error. On this paper, we propose an evolutionary approach to find pruned networks out of feed forward fully connected ones that improve their overall performance. Furthermore, we test three different approaches regarding the training of the pruned network, i.e. no training at all, retrain the network after the pruning or instantiate a new network with its connection pruned, on six different datasets. Each individual is represented as a binary string, that represents the connections on the network and three different fitness functions are implemented, one for every approach mentioned before. The algorithms do not provide just one network, but also all those that have the same fitness as the best one, avoiding duplicates. This helped us compare the methods and evaluate the algorithms. Plus, each pruned network found is tested using the 5x2 cross-validation combined F test to determine whether the pruned network improves or deteriorates the performance on unseen data of the fully connected one. The results show that the "No Training" approach is completely unviable as no positive outcomes are found. However, the remaining approaches retrieve improved networks, having the best effects on larger networks. The overall results show a positive step towards a possible method for finding a pruned topology that performs better than the fully connected one.

3 INTRODUCTION

3.1 OVERVIEW

Neural networks are very useful and popular tools that provide a framework for classification problems. They are widely used and have proven to be effective in a wide array of problems. It is relatively easy to start using them, however it becomes very challenging and time consuming to master their designs, which can vary extensively with each different problems. Many papers, books and courses refer to designing a neural network as “black art” mainly because there is not a deterministic way to do this. It takes a lot of practice and experience to be able to pinpoint accurately whether your network topology has a design flaw and in such case, where is it.

The term topology describes how the neurons of the neural network are connected between each other. This is a key factor that affects the performance of the network as such. The classic topology is the feedforward network, in which the input neurons are connected to hidden layer ones and these in term are connected to the following hidden neurons or, eventually, to the output neurons (Figure *1*). The most common one among these is the fully connected three layer network, i.e. an input layer, a single hidden layer and an output layer. In this case the input layer neurons are also connected to the output neurons (Miikkulainen, 2011). As it is evidenced, we have here the first design choice to make. How many layers should my network have? Usually deeper networks (those that present more hidden layers) tend to be used for more complex problems and can be very precise. But these also can overfit to the data used to train the network and is rendered useless for new unseen data. Furthermore, the designer has to decide how many neurons each layer will have. Other popular topologies are the recurrent network, which is mainly used for sequential data and convolutional network, most notably used in image recognition. This adds a new level of difficulty, as these topologies can be combined to perform even more complex tasks. As mentioned before, there are not any design techniques or methods that can assure the programmer that the topology that they are using is optimal or if there is another one that has a better performance.

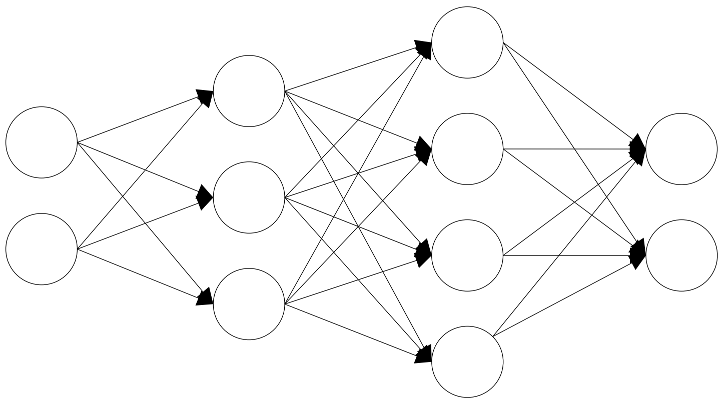

Figure 1. A simple example of a feed forward network.

Furthermore, there is a process called ‘Pruning’ which, as the name suggests prunes the network. This means that the networks connections, or “branches” are going to be cut off. This is done to achieve smaller, cheaper and faster networks but also to prevent overfitting to training data. Pruning can be nowadays useful tool as it can easily reduce the size of a network with a simple process (Reed, 1993)(Setiawan et al., 2019)(Blalock et al., 2020)(He et al., 2018). However, the same issues as before apply, there are not any popular methods to determine which connections to remove. This creates a new layer of complexity in designing a neural network.

This is where Evolutionary Algorithms (EA) come into play. An EA is a heuristic approach that is based on the evolutionary process to solve large scale problems that are not usually solvable in real time. These algorithms have been proven to be efficient in finding a suitable solution to a problem with a large search space. They do this by keeping a population of possible solutions which have a 'fitness' value, a measure of how good the solution is. Then new populations are created iteratively by combining the 'fitter' solutions and generating new candidates from them. This process is repeated until the algorithm converges or an acceptable solution is found (De Jong, 2006) (Michalewicz & Fogel, 2004)(Soni, 2018). Because of their search capabilities, evolutionary algorithms have been used in combination with neural networks in order to make the process of building one, simpler. EAs have been used to search for the neural network's weights, hyperparameters and topology. This combination has the potential to help the design and development process of new networks making it easier for newcomers and seasoned developers to speed up their projects.

3.2 AIMS AND OBJECTIVES

The aim of this project is to develop an effective method for finding pruned network topologies that improve the performance of the network by using evolutionary algorithms. The objectives of this project are:

1. To develop an evolutionary algorithm that is able to find pruned networks from fully connected feedforward networks.
2. To find a quantitative measure of the performance of the evolutionary algorithm in finding a pruned network that does not dampen the performance of the fully connected network provided that these networks exist.
3. To determine which of these methods is the most reliable in improving the performance of a network: to retrain the pruned network, to train the pruned network from scratch or to prune the network without any further training.

4 LITERATURE REVIEW

4.1 NEURAL NETWORKS

Neural networks are an attempt to reverse engineer the human brain and its functions, the objective being, to recreate the brain's learning capabilities. The human brain can easily perform tasks, such as speech recognition, that are very hard for a computer to do. If we recreate the brain as a computer program, then perhaps, computers will be able to perform these tasks easily too. The problem is that we have yet to fully understand the human brain, in particular, how it learns. Alpaydin (Alpaydin, 2014) describes three levels of analysis: the computational theory, representation and algorithm and the hardware implementation. The first level, corresponds to the objective of the task at hand. The second level describes the inputs, the outputs and how the algorithm transforms said inputs into the outputs. The final level is the actual system realized physically. The main idea from these levels is that given a computational theory, there may be different algorithms and representations to it and in subsequently, there may be different hardware implementations for each of these algorithms and representations. An example would be how birds and airplanes reach the same objective of flight, using different implementations. Planes do not possess feathers nor flap their wings, but rather use turbines. The problem is the field of aerodynamics is well developed and understood but the ways in which the brain learns are not. Therefore, what we attempt to do is to recreate the brain as faithful as possible using our current understanding, which leads to neural networks.

4.1.1 Neuron

The neuron is a specialized cell that is able to combine information coming from several other neurons and generate an output. A typical neuron is composed by a soma or body, dendrites, an axon and an axon terminal (Figure 2).The dendrites are protrusions from the soma while the axon is an elongated dendrite. A neuron receives information from other neurons through its dendrites, processes it and outputs a response through the axon. The axon terminal then connects to one or several neurons, as it can branch out (Newman, 2010).

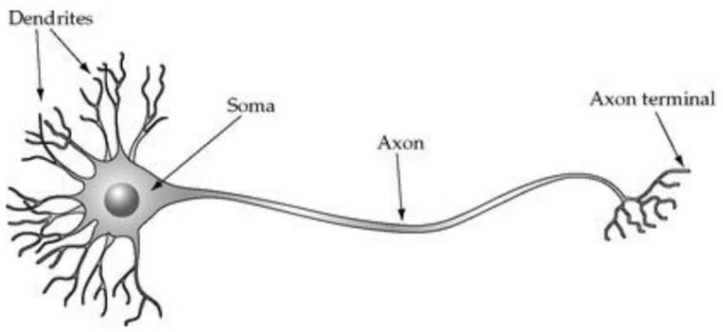

Figure 2. Structure of a neuron (Newman, 2010).

The connections between neurons are called synapses and they are of electrochemical nature. The information travelling through them is basically voltage generated by an exchange of sodium and potassium ions through the cell membrane. This voltage is called 'action potential'. A neuron will receive the action potentials from other neurons, add them and fire its own action potential. However, this is not always the case. Sometimes, the sum of the dendrites' inputs does not trigger the action potential, as it does not reach the excitatory threshold required by the neuron. Furthermore, the inputs can also be inhibiting, meaning that they will make the neuron less likely to fire. Finally, not all excitatory or inhibitory neurons contribute the same amount of voltage to the dendrites of the neuron, meaning that one input can be twice as helpful than other regarding threshold. A neuron can combine excitatory and inhibitory inputs allowing it for complex information processing functions.

The brain is composed of approximately 10^{11} neurons that form around 10^4 synapses between each other. The different sectors of the brain are specialized for different tasks, for instance, on the occipital lobe of the brain one can find the centre for vision (Newman, 2010).

4.1.2 The perceptron

The perceptron is the basic unit of a neural network in the same way the neuron is the basic cell of the brain. As the neuron, it has inputs and outputs that connect the perceptron to other perceptrons. To simulate the excitatory and inhibitory nature of inputs, each input $x_j \in \mathbb{R}, j = 1, \dots, d$ is associated with a connection or synaptic weight $w_j \in \mathbb{R}, j = 1, \dots, d$ plus a bias unit b (Alpaydin, 2014). The sum of all inputs can be computed as:

$$z = \bar{w}^T \bar{x} + b = \sum_{j=1}^{d} w_j x_j + b$$

The perceptron is also associated with an activation function which acts as a threshold setter for the neuron itself. There are different activation functions and it is a design choice which function to use on each perceptron. A typically used activation function is the sigmoid function, defined as:

$$y = \sigma(z) = \frac{1}{1 + \exp(z)}$$

Therefore, on each perceptron there are two calculations involved, one of z and another of y, as shown in Figure 3.

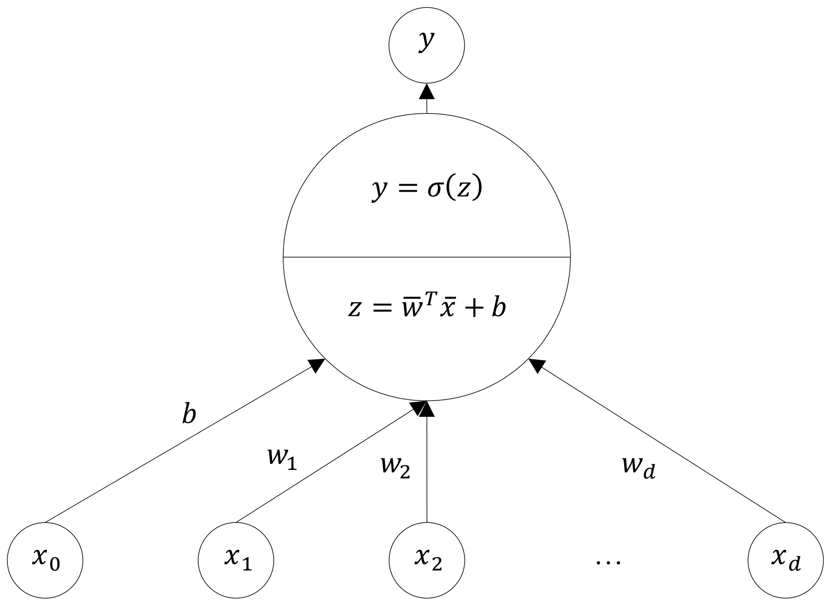

Figure 3. A single perceptron with its inputs, weights and calculations involved in it.

A perceptron can only output one value, so if several outputs are required, several perceptrons need to be implemented (Figure 4). In this case, outputs can be written as an output vector $\bar{y}$, which can be calculated as:

$$y_i = \sigma(z_i) = \sigma\left(\sum_{j=1}^{k} w_{ij}x_j + b\right) \Rightarrow \bar{y} = \sigma(\bar{\bar{w}}\bar{x} + \bar{b})$$

Where $\bar{\bar{w}}$ is the matrix of weights associated to every perceptron.

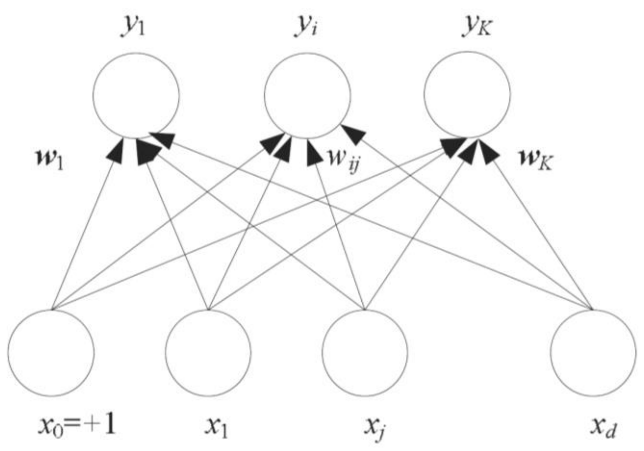

Figure 4. Implementation of several perceptrons (Alpaydin, 2014).

Normally, when solving classification problems, the SoftMax activation function is used instead of the sigmoid function. The SoftMax function is defined as:

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_k)}$$

This function will output the probability that y_i belongs to class C_i.

4.1.3 Multiple Layer Networks

A single layer of multiple perceptrons is limited to solving only linear problems, however this does not occur when there are intermediate or hidden layers between the input and the output layers. In this case, the outputs of the first layer of neurons serve as input for the following layer of neurons, and so on. Each neuron of each layer performs the weighted sum and activation function as before. Furthermore, each layer can have its own activation function, i.e. layer one can use the sigmoid function, whereas layer two can use the tanh function, which ranges from -1 to +1. However, all the neurons belonging to a layer must use the same activation function.

MLP or Multilayer Perceptrons are a linear combination of nonlinear functions. To ensure this, the outputs of the hidden units should be non-linear. Otherwise, we would obtain linear combinations of linear combination. To achieve non-linear outputs, we use non-linear activation functions, such as the previously mentioned (Alpaydin, 2014).

It is at this point where neural networks start to get complex as they can have as many layers as the programmer wants to and each layer can have different numbers of units. Plus, in this paper we have only mentioned feed-forward networks, which are just one of the types of layers one can find in the literature. It is common knowledge that simpler networks tend to generalize better than overcomplicated ones, however, it is not always clear what is considered simple enough. The structure of the network is completely problem-dependent and it is up to the researcher to determine which architecture suits his/her problem the best.

4.1.4 Forward Propagation and Back Propagation

After defining the network, it is time for it to learn. To do so, two distinct processes are performed, forward propagation and back propagation.

As mentioned before, the data is inputted to the network, then each layer computes a weighted sum and passes it through an activation function. This new data is fed into the next layer which repeats the process until the data is transformed into a prediction. This process is called forward propagation.

If we want our network to learn, then we need to use the error the prediction made by the network has. To make the prediction better, we need to update the network's weights accordingly. We use stochastic gradient descent, which uses the following formula:

$$Update = LearningFactor * (DesiredOutput - ActualOutput) * Input$$

The learning factor is a number that is tuned to make training more efficiently. The other factors of the formula can be rewritten as the gradient of the error function with respect to the weights, so we can rewrite the whole formula as:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \alpha \frac{\partial E}{\partial w_{ij}}$$

When updating a multiple layer perceptron, the process is the same, but to update the weights, we need to use the chain rule:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

The process of updating the weights is called backpropagation, as the error propagates from the output backwards towards the input (Alpaydin, 2014).

For the network to learn, this process of forward and backpropagation is repeated several times. Each time it is repeated is called an epoch, therefore, when saying that the network was trained for a number of epochs, we mean that this process was repeated said number.

4.1.5 Pruning

We can define a network's architecture as the way its parameters are configured to transformed its inputs into outputs. This includes number of layers, units on each layer, activation functions, etc. The architecture turns into a model when the weights are set. Pruning entails removing some weights from the network by setting them to zero (Blalock et al., 2020). This can be done to an architecture or to a defined model. What we are doing essentially when pruning a network is cutting some connections between neurons, making the network simpler (Figure 5).

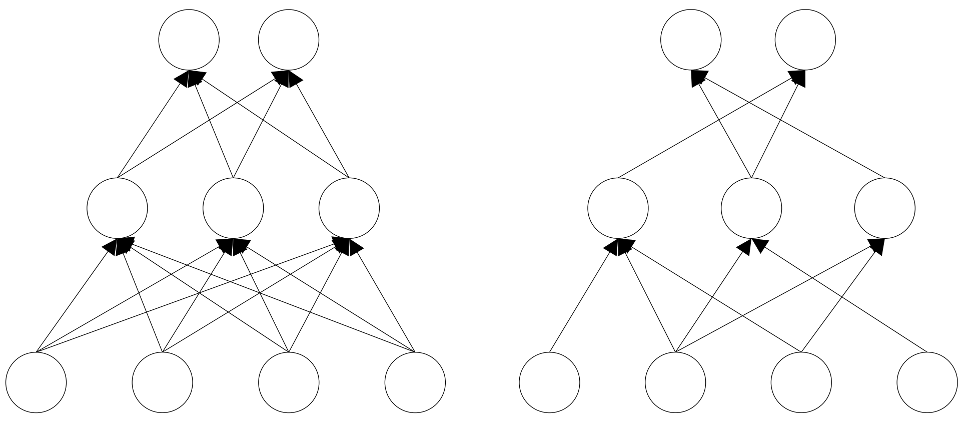

Figure 5. Visual representation of pruning. To the left, a fully connected feed forward network. To the right, the same network after being pruned.

Simpler networks are usually preferred over more complex ones as they avoid overfitting. Overfitting is a common problem in large complex neural networks usually associated to the network having more degrees of freedom than the number of training examples. We can see our network overfitting when the training error decreases but the generalization error increases. This means our network is taking outliers in the training data and adopting them as part of the general rule, which is something we do not want. Overfitting can also occur when training the network more than needed, although this is easily solved by stopping training sooner. Having a simpler and smaller network also has the benefit of being faster and cheaper to build and to train. This is not a small feat. Real world applications nowadays require their networks to be as resource efficient as possible, wasting the least amount of computing time and memory as possible (Reed, 1993)(Anwar et al., 2017)(He et al., 2018).

Regarding pruning methods, the most common way to prune a network is to set the desired weights to zero, which can be referred to as a "brute-force" method. There are however, some considerations to take into account as to how the pruning itself is performed. First, we have structure matters to take into account. This means whether we prune individual parameters (unstructured pruning) or groups of them (structured pruning), the main difference being that structured pruning can help optimize the algorithm. Second, is scoring. Some pruning methods use a scoring model to give each parameter a comparative value. Then a percentage of the lowered-scored parameters are pruned. The alternative to scoring is to select the weights randomly. Thirdly, we have what is referred to as scheduling considerations. Some algorithms prune all their weights simultaneously while others prune them step by step evaluating the process at each step. The number of weights pruned at each step can vary. The last consideration we need to take into account involves training the network after pruning, or rather re-train it. Some algorithms choose to train their networks for a few epochs after being pruned, which is called fine-tuning. Others prefer to leave the network as is. A final group trains the network "from scratch" i.e. resets the network to its original state before any training has been done and trains it with the new architecture. All of these aspects make up the different pruning algorithms, however, there is not yet a clear and definitive method for efficient and effective pruning (Reed, 1993)(Blalock et al., 2020).

4.2 EVOLUTIONARY ALGORITHMS

An evolutionary algorithm is used as a broad term to describe a metaheuristic approach that takes evolutionary systems as inspiration to solve large scale problems. There are three variations to an evolutionary algorithm: evolutionary programming, evolutionary strategies and genetic algorithms. In this paper, the three are included when discussing evolutionary algorithms (De Jong, 2006)(Michalewicz & Fogel, 2004)(Soni, 2018).

As mentioned before, evolutionary algorithms base their methods on evolutionary systems. This concept was presented on Darwin's book "On the Origin of Species", where the main principles of an evolutionary process were described. These include 'phenotypic variation', which explains that within a population, individuals have different shapes, behaviours and physiologies, even though they belong to the same species; 'differential fitness', which states that there are characteristics better adapted to survive in different environments than others; and 'fitness is heritable', which implies that fitness can be passed on to future generations. These concepts combined with the facts that one of more populations of individuals are competing for limited resources and that these populations are dynamic due to the births and deaths of the individuals in them, embody a Darwinian evolutionary system (Darwin, 1859)(Lewontin, 1970)(De Jong, 2006). These elements can be simulated in an evolutionary algorithm. In in, a population is created, its evaluated or given a fitness value, they reproduce creating a new population and the offspring may replace the existing population. This cycle is called a generation and is repeated, new potential solutions are found until the algorithm converges to an optimal solution.

4.2.1 The Individual and its Representation

Any individual in an evolutionary algorithm is a possible solution to the problem tackled. Sometimes called 'candidate solution', the individual and its representation will define the search space of all possible solutions, therefore, it is one of the most important design choices early on.

There are two main approaches regarding an individual's representation, the phenotypic and the genotypic approach. On the first one, the individual's characteristics are explicit, that is, they do not need to be transformed or decoded to be understood or used. This takes inspiration on the phenotype of any organism, which are the observable characteristics, such as eye colour. On the genotypic approach, the individual's characteristics are implicit, which indicates they need some form of decoding to be understood or used. In nature, these are called genes, which encode the individual's characteristics in a set of proteins called DNA. An example of these representations would be if the algorithm was trying to determine the minimum value of a function that takes as input natural numbers. A phenotypical representation of a candidate solution could be the number 6. A genotypical representation of the same candidate can be 100, which is 6 in binary. The genotypical representation needs to be decoded into 6 to be used in the function, as it only takes natural numbers as input.

Another decision to make is which geometry will the solutions have. The simples way to do this is representing the individual as a fixed-length linear object. The candidate solutions are then a vector of parameters, which are usually referred to as 'genes' and the search space is an N-dimensional one. Another approach is to have the candidate solutions be nonlinear objects, such as matrices or graph structures. In this case, the challenge is to design reproductive operators that work effectively in creating interesting solutions. This approach makes more sense when you can find subassemblies in the candidates solutions. For example, in a connectivity matrix between two layers of a neural network, a row indicates the weights that lead out of a node while a column indicate the weights that connect into a node. Individuals can also have variable-length, although this is usually reserved for lists of items, e.g. rules. The items are the "genes" in the reproductive operators. The difficulty in this case appears when the order of the list matters, otherwise, the reproductive operators for them are easier to design. The final and most complex representation is to make the solution candidates as nonlinear, variable-length objects. This representation is usually reserved for graph representations in which you need to add or delete nodes. As previously, the difficulty comes when designing effective reproductive operators.

4.2.2 Fitness

The fitness of an individual in an evolutionary algorithm can be regarded as the quality of the candidate as a solution to the particular problem. Each solution needs to be associated with a fitness value in order to determine which individual is better and to do so, one needs to design a fitness function. This function will take the individual's traits as inputs and return a fitness value. Usually, a higher fitness equates a better solution. However, this is not always as straightforward. When designing the function, we need to take into account the range of values we can have as an output. For instance, if a function can output zero or negative fitness values, how will assigning a probability work for these individuals? Questions like these should arise in the designing process. Furthermore, constraint problems usually have a Yes/No approach, as the individuals need to satisfy certain conditions. But assigning a Yes/No or a Boolean fitness function may lead to the whole population having a No fitness, and making the selection process irrelevant. In these cases, a constructive feedback approach can be used, i.e. 'This individual has 45% of the constraints fulfilled'. The calculation of a solution candidate's fitness is, in most cases, independent to the rest of the population. There are population-dependent fitness approaches such as using relative fitness which is calculated as:

$$\text{relative fitness} = \frac{\text{individual's fitness}}{\text{sum of population's fitness}}$$

This approach is also used as probability for the individual of being chosen during the selection process. Another approach is to use fitness proportional reproduction, suggested by Whitley (Darrel Whitley, 2001). This can be calculated as:

$$\text{fitness proportional reproduction} = \frac{\text{individual's fitness}}{\text{average population's fitness}}$$

4.2.3 Selection

The selection process is performed twice during a single iteration of an evolutionary algorithm. The first one, is to select a determined number of parents for reproduction and the second one is to choose from the pool of the previous generation and offspring, which individuals will move on to the next iteration, i.e. survive.

There are two basic categories for selection mechanisms, deterministic and stochastic. In a deterministic method, each individual is assigned a number of times to be selected whereas in a stochastic method, each individual has a probability to be selected. De Jong recommends that if the population is small (less than 20 individuals) a deterministic method is used, as sample sizes selected can be too small to represent the whole population, if the selection probability is uniformly used. Using a deterministic method can avoid this problem by selecting each individual once (De Jong, 2006). Furthermore, a stochastic method can lead to loss of diversity or 'genetic drift' due to sampling error. When the population is large enough, a stochastic method is preferred, as this can avoid the algorithm to converge to a suboptimal solution. In this case, a stochastic method is adding noise to the population and increasing the overall robustness of the algorithm.

Another aspect of selection is how the selection pressure is distributed. On the previous paragraph, a uniform selection was presented. However, one can introduce fitness into consideration when selecting individuals. This can be used to shift the selection pressure onto fitter individuals. An extreme method to put this into practice is elitism, where only the top percentage of individuals is viable to be selected. In this case, unfit individuals have no chance of being selected. The opposite of this would be to apply a uniform distribution probability to the population, where every candidate has an equal probability of being chosen. Less extremist methods include tournament selection and fitness-proportional selection, sometimes called 'Roulette Wheel Selection'. In tournament selection, a number of participants are chosen randomly from the population, then the fittest individual from these is the tournament winner. The tournament needs to be repeated as many times as individuals one needs to select. The more participants are included in a tournament, the more this method resembles an elitist method. Fitness-proportional selection, on the other hand, assigns a probability p_i

$$p_i = \frac{\text{individual's fitness}}{\text{sum of population's fitness}}$$

to each individual. This way, all candidates have a chance of being chosen, but fitter individuals are more likely to be selected. This method can be elitist if the population is heterogeneous.

When the selection mechanisms are pressure based, the algorithm will converge to different optimal points on the fitness landscape. The amount of selection pressure will determine the rate at which the algorithm will converge. However, too much pressure is not always welcome, as it has the tendency to lead to suboptimal candidates. This is due to the fact that relying heavily on selection pressure is similar to exploitation activities.

4.2.4 Reproduction

In an evolutionary algorithm, such as in nature, individuals can reproduce, creating new and different candidate solutions that can replace them. The idea of reproduction is to generate new traits, via genotype or phenotype, that allow for the population to evolve into a more fit one. As in nature, evolutionary algorithms present two types of reproduction, sexual and asexual. Contrary to nature, these are not exclusive and can be combined in an evolutionary algorithm.

Asexual reproduction is characterized by being performed by a single individual. The most common form of asexual reproduction is mutation. The process of mutation involves the cloning of a parent and variate one or more characteristics in the offspring. One can control the amount of variation by determining the number of characteristics to be changed during the process. The most frequent method of mutation is the Gaussian mutation in which the operator mutates a number x of genes by a given step size s (De Jong, 2006). Each gene has a probability x/L of being mutated, which on average will provide x mutations. If we refer to a fitness landscape, the distance between the parent and the offspring is given by a combination of x and s. The larger these two values are, the more different the offspring will be form the parent. A particular case of the Gaussian mutation is the 'bit flip' mutation, through which the individual, represented by a binary array, is mutated by 'flipping' a number x of its bits, i.e. a 1 is switched to a 0 and vice-versa. In this case, the distance is just determined by x, which corresponds to the Hamming distance. It is generally recommended that in a 'bit flip' mutation, the number of mutations is kept low, as a small Hamming distance does not always equals a small Euclidean distance and can mean a large step in the fitness landscape.

Sexual reproduction is characterized by being performed by two or more individuals. The classic mechanism is recombination in which two parents concede parts of their genes to a single offspring. 'Crossover' is a typical operator for fixed-length linear genome representations. In 'K-point-crossover' the operator selects a number k of positions in the genome and slices both parents through them. The offspring is then reassembled using segments from each parent. Note that in this case, it is possible to create two separate offspring, with the remaining genome segments (Figure 6, A). This operator is fast and can be tuned with the amount of crossover points. However, it can generate a location bias, as traits closer to each other will have a higher chance to be from the same parent. An alternative is 'uniform crossover', in which each characteristic is individually picked randomly from each parent (Figure 6, B). This eliminates the possibility of location bias but is a longer process and can take computational time, especially for longer genomes.

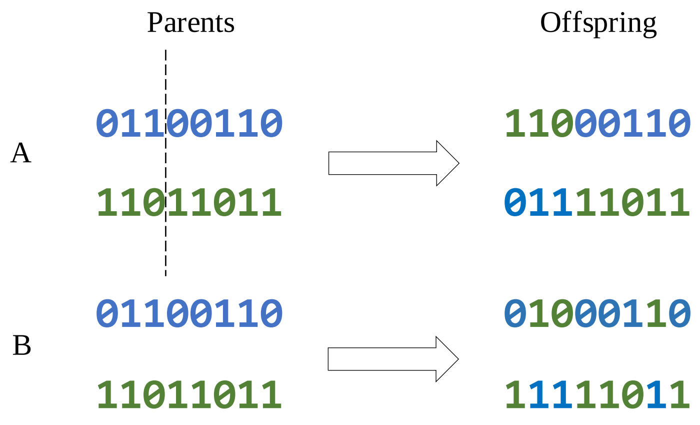

Figure 6. A, One-point crossover. B, Uniform crossover. To the left, the parents. To the right, the offspring. The different colours in the offspring indicate the parent from which that section comes.

It is very common to see recombination and mutation used together in a single algorithm, as recombination usually favours exploration while mutation usually favours exploitation. This gives the heuristic a balance of both activities, making it more robust. The choice of reproductive mechanisms is closely related to the choice for representation. How an individual is characterized will make the programmer find a way into making the operators combine its traits in interesting ways. However, some problems have highly epistatic genes, which means that they are very dependent to one another and changing a few may not affect fitness at all. An operator achieving high fitness correlation is desired, although this is not always achievable.

4.3 Combining Neural Networks and Evolutionary Algorithms

Even though neural networks have been around for years, there are no standard ways to design one, given a specific problem. There has been some insight regarding which structures may be better suited for specific tasks, such as Convolutional Neural Networks are mostly used in image recognition problems, but this is as far as we can find regarding the implementation of a net from scratch. It is common to see words like 'black art' when browsing the literature on the design of networks, although all agree that the best practice is get experience on them and test different topologies and methods to define the best network to use. To ameliorate this issue, researches have used evolutionary algorithms to help them design and even improve their neural network design. Evolutionary algorithms have been used as optimizers and as search algorithms, and can therefore be used to train the weights in a neural network and to find an optimal topology.

4.3.1 Training a Neural Network using Evolutionary Algorithms

Finding a network's weights is an optimizing problem, as we want to find the set of weights that minimize the error on the task we are trying to achieve. The search space is usually highly dimensional and may contain several minima. The widely used method for this is the backpropagation algorithm, described on section 4.1.4. However, this method is a local gradient search method, therefore it can get stuck in local minima. Evolutionary algorithms are experts in avoiding this by using parallel search and increasing exploration qualities. The simplest way to train a network using evolutionary algorithms is to search for the weights for a fixed-structured network using the algorithm, inputting them into the net and testing accuracy on the different sets. These accuracy values usually serve as fitness for the individuals, which are a concatenation of the different weights. This method is widely used when the activation function of the neurons is nondifferentiable therefore traditional backpropagation (gradient-based) is not possible.

An alternate version of this method is to perform an initial search of weights using evolutionary algorithms, finding promising regions of the search space and using backpropagation to fine tune the weights. This method tends to increase the computation time per individual, but it can reduce the overall training time (Branke, 1995) (Cantú-Paz & Kamath, 2005).

Training a network using evolutionary algorithms is a viable method but it is not scalable to larger domains as the individuals get progressively longer and the algorithm gets slower. Furthermore, it has not been found that training a network with these methods improves on backpropagation algorithms.

4.3.2 Finding the Topology of a Neural Network using Evolutionary Algorithms

The neural networks performance is highly relying on its structure. A simpler topology may not be able to learn the task at hand, but an overly complicated one can overfit and may not generalize well. Furthermore, the structure will have an effect on the computation requirements on training and final implementation. Even though the structure is highly important, there is no way to determine an optimal structure nor to prove that the one being used is optimal. There have been ways to determine if one network generalizes better than another one, but there is not a method that allows you to define a better structure. Fortunately, there are several methods to find and evaluate network structures using evolutionary algorithms.

The methods can be divided into those that use a direct encoding to specify every connection of the network and those that use an indirect specification (Branke, 1995). The first one usually involves a binary connectivity matrix, in which a “1” indicates a present connection and a “0” the absence of one. The matrix can also be used as a concatenated array to facilitate its manipulation. The matrix is used to build and train a neural network which then can be tested. This method is simple enough, however, the individuals’ lengths are $O(n^2)$, therefore it is not scalable to larger problems. Direct encodings are mostly used to prune networks. As mentioned previously on section 4.1.5, this means eliminating existing weights from a fully connected neural network in an attempt to improve its generalization abilities. Whitley et al. proposed using genetic algorithms to find new network topologies in combination with a reward system. The networks were trained after pruned based on the number of connections pruned. This way, those networks that had less connections and were more susceptible to noise, were given more epochs to retrain. The idea was to find the smallest network with the best learning capability (D. Whitley et al., 1990). Hancock also tested whether pruned networks were capable of improving the performance of a fully connected network. He tested pruning a pre-trained network using genetic algorithms, re-training them later and compared the process to instantiating the network with new random weights. His results showed that at first, that his initial approach was more beneficial, but proved that the algorithm was also able to find networks that improved their performance by using linear interpolation and gradually changing the starting weights. This method was however, extremely costly regarding computing time (Hancock, 1992). Cantú-Paz tested a simple genetic algorithm against three estimation distribution algorithms, showing that the first one had better results in finding networks. Plus, he stated that the most benefited networks were those that performed poorly initially and that retraining after pruning provided little to no benefit (Cantú-Paz, 2003).

Indirect encodings, on the other hand, use higher levels of encoding resulting in many different methods. A simple method is to define a networks topology e.g. fully connected, and using evolutionary algorithms to find different parameters, such as number of layers, number of neurons on each layer, learning rate, etc. The main drawback to this method is having to commit to one topology (Belew et al., 1990)(Marshall & Harrison, 1991). More complex methods come from obtaining a connectivity matrix using graph rewriting methods. These use grammar-based rules to obtain 2x2 matrices that subsequently are rewritten. The drawbacks in this case stem from the complexity of the method and the fact that the size of the final matrix will always be a power of 2 (Kitano, 1990).

5 MATERIALS AND METHODS

In this section we will describe the experiments performed during this project, which took inspiration on a paper by Cantú-Paz and Kamath (Cantú-Paz & Kamath, 2005). In summary, we took six different datasets and implemented a different neural network for each of this. Then we took binary strings that represented the weight matrices, and used it as individuals for our evolutionary algorithms. The ones and zero represented whether or not the weight or connection was going to be present or pruned respectively. The network was pruned and the algorithm was run. We designed three different algorithms with the same characteristics but one, the fitness function, which was valued accordingly to the method being tested "No Training", "Train after Prune" and "Train from Scratch". We used Python to code all sections of this project.

5.1 INSPIRATION PAPER

The paper that served as inspiration was "An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems" (Cantú-Paz & Kamath, 2005). In this paper, the authors describe a series of experiments testing out different combinations of evolutionary algorithms and neural networks, determining which ones are more useful than others. They use EAs to train neural networks by using the algorithm to search for weights and to find initial weights and refine them using backpropagation. Cantú-Paz and Kamath also use binary-encoded individuals to select the networks' features. Finally, they test the efficiency of evolutionary algorithms on designing the network itself. On an initial experiment, they search for a connectivity matrix, on a second experiment, they train a fully connected network and use the EA to find a pruned connectivity matrix and on the last experiment, they use the algorithm to find parameters, such as number of hidden units and parameters of backpropagation, among others. The authors perform these experiments on a number of datasets, each with its own set of pre-set parameters, such as number of hidden units and epochs to be trained. All of the networks have a fully connected feed-forward architecture and use the tanh activation function for the hidden layers. The neural networks are trained using backpropagation with a 0.15 learning rate and a 0.9 momentum term. As a final note, the authors use five iterations of two-fold cross-validation combined with F tests to compare the accuracy of algorithms and confirm that the errors come from the same distribution (T. G. Dietterich, 1998)(Alpaydin, 1999)(Cantú-Paz & Kamath, 2005).

The experiments from the paper by Cantú-Paz and Kamath that are of interest to this paper are the ones concerning pruning a fully connected network. As the paper states, to prune a network, we only need to set the weights we want to prune to zero. Therefore a binary string was used to represent the individuals. This string was a concatenation of the rows of the connectivity matrix. The population size was given by $3\sqrt{l}$ where l is the length of the string, which in this case is

$$l = (n^\circ\ of\ hidden\ units + n^\circ\ of\ output\ units) * n^\circ\ of\ input\ units + \ n^\circ\ of\ hidden\ units \\ * n^\circ\ of\ ouptut\ units$$

As for reproduction, the paper uses multipoint crossover with a 1.0 probability and $l/10$ crossover points plus a mutation rate of $1/l$. The algorithm was run for 50 generations or five generations with no improvement of the best solution. As for fitness, they state that a copy of the initial fully connected network was pruned and used the accuracy of it on the training data. Furthermore, the pruned network was not retrained. It is not clear whether the training set used for calculating fitness was the inner subdivision or the initial division. Finally, the results they presented were the accuracies of the best individuals found, tested on unseen data and concluded that in general, pruning does not improve or worsen the networks performance (Cantú-Paz & Kamath, 2005).

5.2 IMPORTING THE DATASET AND SETTING IT UP FOR ITS USE

The datasets used were six, and are all detailed in Table 1. All received the same treatment to be used therefore, we will explain them as one. The datasets were obtained from the UCI repository (Dua & Graff, 2019), except for the "PIMA Diabetes" dataset which was obtained from Kaggle (Kaggle, 2016). To avoid confusion, it is important to clarify that the "Breast" dataset is the "Breast Cancer Wisconsin (Original)" dataset and the "Sonar" dataset is the "Connectionist Bench (Sonar, Mines vs. Rocks)" dataset, as the UCI repository has many sets using similar denominations. The missing values in the "Diabetes" dataset were denoted with zeroes, and were not removed but rather treated as meaningful values. Finally, the values obtained for the neural network sizes and epochs were obtained from a paper by Maclin and Opitz (Maclin & Opitz, 1999). Those datasets that had only one output unit were changed to two output units, as shown in Table 1 to keep the output function, a SoftMax function, consistent.

Table 1. Datasets used and their characteristics.

			Features			Neural Network			
Dataset	Examples	Classes	Cont.	Disc.	Miss	Input	Hidden	Output	Epochs
Iris	150	3	4	-	N	4	5	3	80
P-Diabetes	768	2	9	-	Y	8	5	2	30
Breast-W	699	2	9	-	N	9	5	2	20
Wine	178	3	13	-	N	13	5	3	15
Ionosphere	351	2	34	-	N	34	10	2	40
Sonar	208	2	60	-	N	60	10	2	60

First, the output classes were turned into one-hot encoded arrays using the Pandas library (McKinney et al., 2021) turning categorical variables into indicator ones. This allows for easy manipulation and efficiency regarding the algorithm, as it is less complex working with numerical values rather than labeled data.

Next, the dataset was normalized by rescaling it between [-1; 1], to avoid overflows with large values. The data was then converted into NumPy (Harris et al., 2020) arrays to keep all data types coherent.

The data division happens in two different sections in different ways. To avoid confusion, these will be explained in their respective sections (5.4.1 and 5.5.1)

5.3 NETWORK

The networks created for each dataset and used on the experiment worked the same way. They were all feed forward fully connected networks (until pruned) with the inputs connected to the outputs. The networks had only one hidden layer and their size varied according to the dataset being tested on (Table 1). All networks were trained using backpropagation with a learning rate $\alpha = 0.15$, a momentum $M = 0.9$ and a batch size of one. These values and those from Table 1, were obtained from a paper by Maclin and Opitz (Maclin & Opitz, 1999).

As for the initialization, there were 12 different parameters to initialize, three weight matrices W_1, W_2 and W_{io}, three bias vectors b_1, b_2 and b_{io}, three momentum weight matrices V_dW_1, V_dW_2 and V_dW_{io} and three momentum bias vectors V_db_1, V_db_2 and V_db_{io}. The weight matrices were all initialized randomly using the `randn` command from the NumPy library (Harris et al., 2020),which generates random samples from a normal distribution. These were multiplied by a factor $\sqrt{1/t}$ where t is the number of rows the matrix being initialized has. This was done in order to have smaller numbers on the weight matrix. The bias vectors were initialized as ones, while the momentum parameters, both matrices and vectors, were initialized to zeros (Table 2).

Table 2. Summary of parameters and their initialization.

Parameter	Initialization
Weight matrices (W_1, W_2 and W_{io})	Random initialization, multiplied by $\sqrt{1/t}$
Bias vectors (b_1, b_2 and b_{io})	Ones
Momentum matrices (V_dW_1, V_dW_2 and V_dW_{io})	Zeros
Momentum vectors (V_db_1, V_db_2 and V_db_{io})	Zeros

Before the forward propagation started, the weight matrices (W) are multiplied by their respective masks (P) so they were pruned. Forward propagation used the following formulas:

$$A_0 = x\_train$$

$$Z_1 = (W_1 \odot P_1) * A_0 + b_1$$

$$A_1 = \tanh(Z_1)$$

$$Z_2 = (W_2 \odot P_2) * A_1 + b_2$$

$$Z_{io} = (W_{io} \odot P_{io}) * A_0 + b_{io}$$

$$A_2 = softmax(Z_2 + Z_{io})$$

Next, the backpropagation started using the following formulas:

$$dZ_2 = (A_2 - y_{train}) * \left(\frac{e^{Z_2}}{\sum e^{Z_2}}\right) * \left(1 - \frac{e^{Z_2}}{\sum e^{Z_2}}\right)$$

$$dW_2 = \frac{1}{m}(dZ_2 * A_1^T)$$

$$db_2 = \frac{1}{m}\left(\sum dZ_1\right)$$

$$dZ_{io} = (A_2 - y_{train}) * \left(\frac{e^{Z_{io}}}{\sum e^{Z_{io}}}\right) * \left(1 - \frac{e^{Z_{io}}}{\sum e^{Z_{io}}}\right)$$

$$dW_{io} = \frac{1}{m}(dZ_{io} * A_0^T)$$

$$db_{io} = \frac{1}{m}\left(\sum dZ_{io}\right)$$

$$dZ_1 = W_2^T * dZ_2 * (1 - (\tanh Z_1)^2)$$

$$dW_1 = \frac{1}{m}(dZ_1 * A_1^T)$$

$$db_1 = \frac{1}{m}\left(\sum dZ_1\right)$$

With m being the number of examples in the training batch.

After backpropagation ended, the accuracy on the validation data was computed. To do so, the output obtained from the SoftMax function was converted to categorical by taking the maximum argument of the array. Then, we checked which of these were equal to the labels from the validation data. If the prediction was equal to the label, a 1 was added to an accuracy array. Otherwise, a 0 is added. The overall accuracy was the mean of this array. In this way, if we had

5 predictions correct out of 10 examples, then the accuracy is 0.5. This is exemplified in the following example:

$$1)\ [0.89 \quad 0.1 \quad 0.01] \Rightarrow [1 \quad 0 \quad 0]$$

$$2)\ \begin{matrix}[1 & 0 & 0] \\ [0 & 1 & 0] \\ & \dots & \end{matrix} = \begin{matrix}[0 & 0 & 1] \\ [0 & 1 & 0] \\ & \dots & \end{matrix} \Rightarrow \begin{bmatrix}0 \\ 1 \\ \dots\end{bmatrix}$$

$$3)\ accuracy = mean\left(\begin{bmatrix}0 \\ 1 \\ \dots\end{bmatrix}\right)$$

Figure 7. Accuracy computing example. In 1) the output from the forward propagation is converted into categorical by taking the maximum argument of the array. In 2) we check which predictions are equal to the labels and add a 1 if is equal and 0 otherwise. In 3) the mean of the array is computed.

Finally, the parameters needed to be updated using the parameters that backpropagation provided. These were updated as follows:

$$V_dW_2 = M * V_dW_2 + (1 - M) * dW_2$$

$$V_db_2 = M * V_db_2 + (1 - M) * db_2$$

$$W_2 = W_2 - \alpha V_dW_2$$

$$b_2 = b_2 - \alpha V_db_2$$

$$V_dW_1 = M * V_dW_1 + (1 - M) * dW_1$$

$$V_db_1 = M * V_db_1 + (1 - M) * db_1$$

$$W_1 = W_1 - \alpha V_dW_1$$

$$b_1 = b_1 - \alpha V_db_1$$

$$V_dW_{io} = M * V_dW_{io} + (1 - M) * dW_{io}$$

$$V_db_{io} = M * V_db_{io} + (1 - M) * db_{io}$$

$$W_{io} = W_{io} - \alpha V_dW_{io}$$

$$b_{io} = b_{io} - \alpha V_db_{io}$$

With α being the learning rate and M the momentum.

5.4 EVOLUTIONARY ALGORITHM

For this project, we used three similar evolutionary algorithms. The only aspect in which they differ was the fitness function, therefore they will be explained as one.

Each algorithm will be referred to as "No Training" (NT), "Train after Prune" (TAP) and "Train from Scratch" (TFS) versions, hinting at the 'after-prune' process the networks receive regarding training.

5.4.1 Data handling

For the algorithm, the whole data set was divided in two, a training set and a test set. The test set was not used through the evolutionary algorithm. The training set was then subdivided into a new sub-training set and a validation set (Figure 8).

The sub-training set is used to train the fully connected network, at the beginning of the algorithm, and those networks found that require training, i.e. in the “Train after Prune” and “Train from Scratch” versions of the algorithm. The validation set is used to test the network’s accuracy.

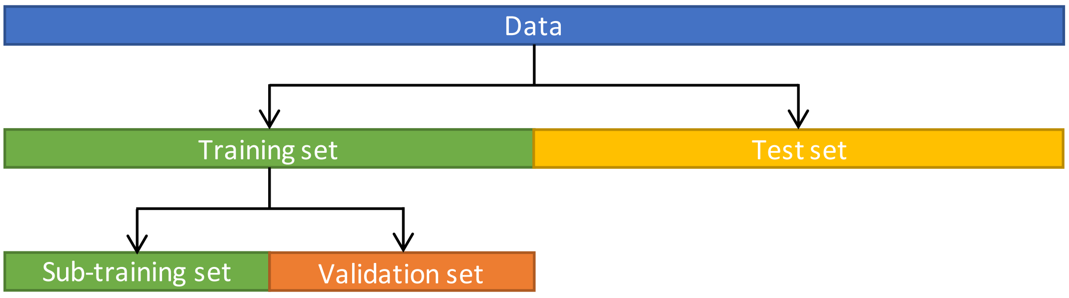

Figure 8. Data handling diagram for the evolutionary algorithm

5.4.2 Individual representation

Every individual was represented as a binary string in the same way the paper by Cantu-Paz and Kamath did. The string was a concatenation of three matrices, each of which was the same size as the weight matrices used in forward propagation. The idea was that the string could then be reshaped into three matrices and applied to the weight matrices as a mask, setting those weights to zero, or pruning them (Figure 9).

$$[1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0] \Rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.56 & 0.26 & 0.3 \\ 0.42 & 0.12 & 0.1 \\ 0.67 & 0.97 & 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.56 & 0 & 0.3 \\ 0 & 0.12 & 0.1 \\ 0.67 & 0 & 0 \end{bmatrix}$$

Figure 9. Example on how the algorithm applies the weight mask. (Above) The binary string is reshaped into a matrix. This matrix is the weight mask. (Below) The weight matrix is multiplied element-by-element, generating a pruned weight matrix.

The length of the string is given by the formula:

$$L = (n^\circ\ of\ hidden\ units + n^\circ\ of\ output\ units) * n^\circ\ of\ input\ units + n^\circ\ of\ hidden\ units * n^\circ\ of\ ouptut\ units$$

The string was divided into three strings $L1, L2$ and $L3$ when computing the fitness to reshape each string into the weight mask. The size of these strings is given by:

$$L1 = n^\circ\ of\ input\ units * n^\circ\ of\ hidden\ units$$

$$L2 = n^\circ\ of\ hidden\ units * n^\circ\ of\ output\ units$$

$$L3 = n^\circ\ of\ input\ units * n^\circ\ of\ output\ units$$

The L_1 string corresponded to the weight mask for weight matrix between the input and the hidden layer, the L_2, to the matrix between the hidden and the output layer and the L_3, to the matrix between the input and the output layer.

5.4.3 Benchmark individual

On each evolutionary algorithm, we created a benchmark individual, which was a fully connected version of the network. The binary string for this individual was a single string of ones of length L. This is the network that was pruned through the search and was the one to be tested against. This means, this network was the one to improve with a pruned network.

At the start of the algorithm, the network was trained and tested on the validation data to obtain its fitness.

5.4.4 Fitness function

This is where the EAs differ, as there was a unique fitness function for each. They all tested on validation data, but they will prune the network and train it in the way the EA is supposed to.

For the "No Training" method, the fitness function took the individual's genome and divided it into the three strings $L1, L2$ and $L3$. Then it passed these as weight masks to the benchmark individual, pruning it. The fitness value was the accuracy of the pruned network on the validation set while having the pruned weight masks.

For the "Train from Scratch" method, the fitness function took the individual's genome and divided it into the three strings $L1, L2$ and $L3$. Then it instantiated a new network with these strings as weight masks and trained it using the sub-training set. The fitness value was the accuracy of the new network on the validation set after being trained.

For the "Train after Prune" method, the fitness function took the individual's genome and divided it into the three strings $L1, L2$ and $L3$. Then it instantiated a new network, copied the benchmark individual's weight matrices and applied these strings as weight masks, pruning it thus creating an already trained and pruned copy of the benchmark. Then the new network was retrained according to Table 3. The fitness value was the accuracy of the retrained network on the validation set.

Table 3. Datasets with their respective number of epochs to train and to retrain after being pruned.

Dataset	Number of epochs to train	Number of epochs to retrain
Iris	80	10
P-Diabetes	30	15
Breast-W	20	10
Wine	15	10
Ionosphere	40	20
Sonar	60	30

The values on the table were obtained by instantiating ten fully connected networks for each dataset using different seeds, training them, pruning them and retraining them. The number of epochs used for retraining were the same as the ones for the initial training. This was done to look for an "elbow" or the point where the network's training efficiency is saturated. This point indicates the optimal number of epochs for retraining (Figure 10). It is important to note that not all of the curves were as clear as the one on the example, and the final number was not obtained based on a calculation but rather on the curves, as the erratic behavior of some datasets did not allow for a saturation measurement.

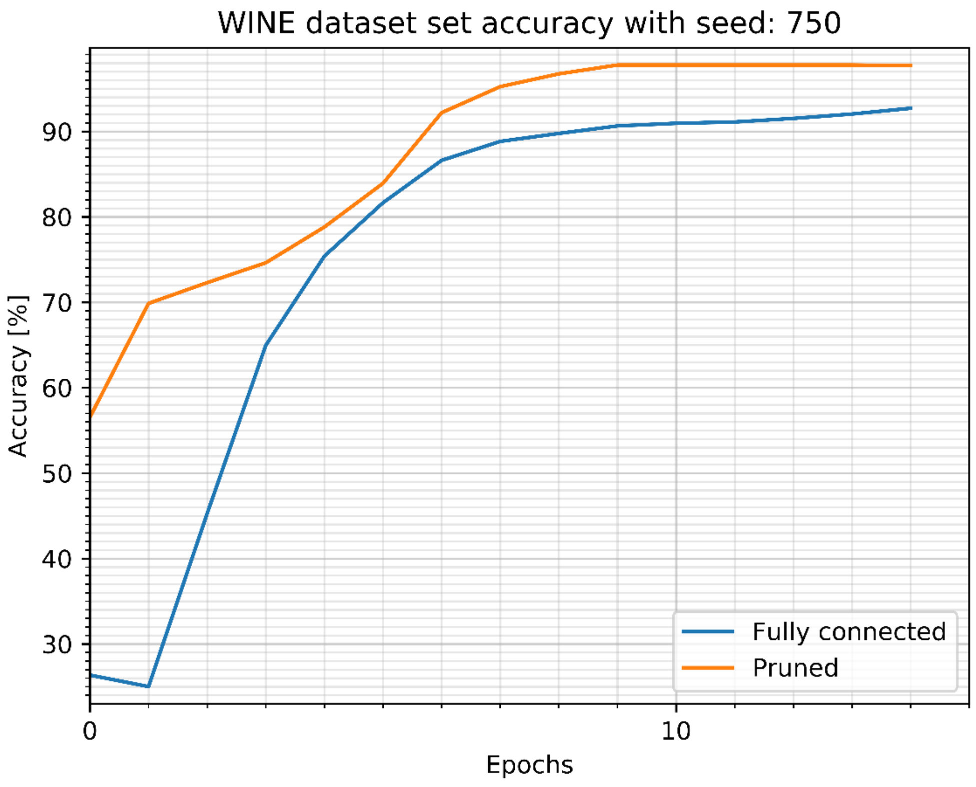

Figure 10. Example of the results for the short experiment in search for the number of epochs to retrain. The blue line shows the accuracy on validation data for the fully connected network, while the orange shows the accuracy on the same validation data after the network was trained and pruned. In the image, we can see that the accuracy 'elbow' of the pruned network lies between epochs six and nine.

5.4.5 Reproduction/Survivor

To choose parents, we used a tournament where we picked randomly three individuals from the current pool and from those, the two fittest were selected as parents. These created only one offspring. The tournament was repeated ten times to generate a new population.

The reproduction methods were uniform crossover with a 0.5 probability of each gene coming from each parent, followed by a bitflip mutation. The mutation rate was given by $1/L$.

As for survivors, we chose to keep all the offspring and discard the previous generation, to keep a high diversity given the large search space.

5.4.6 Best individuals found

Because we had small networks, with small datasets in combination with large search spaces, it was likely that some fitness values were repeated. As we only kept one "best individual" on each generation, it could occur that other candidates with the same fitness were discarded. Therefore, for each seed, we kept a log of all of the individuals found, deleted those with a duplicate genome and extracted all that had the same fitness as the final best individual found. These were all then kept as best individuals found and tested later on the combined 5x2 cv F test (Section 5.5.1).

5.4.7 Generations and population size

The evolutionary algorithms were all run for 50 generations without any other stopping criterion. For each dataset, the population size is 20 and was constant through the generations.

5.5 Testing the individual found

It would not be completely correct to evaluate the individuals found on their performance on validation data, but rather on unseen data. Therefore, to evaluate the individuals found and compare them with the benchmark individual we used the method proposed by Alpaydin called "Combined 5x2 cv F test" (Alpaydin, 1999). Please note that this step was included to test the effectiveness of the algorithms and would not be included in a real-world application.

5.5.1 Combined 5x2 cv F test

The combined 5x2 cv F test is a method used to compare two classifiers. To do this, we performed five iterations of twofold cross-validation. This means, dividing the whole data set in a training and a test set, which should be of the same size. Then, we trained both classifiers on the training set and got their accuracy on the test set. This accounts for the first fold. Next we re-instantiated the classifiers, switched the training set for the test set and repeated the training and testing process, which would be the second fold (Figure 11). This process needs to be performed five times.

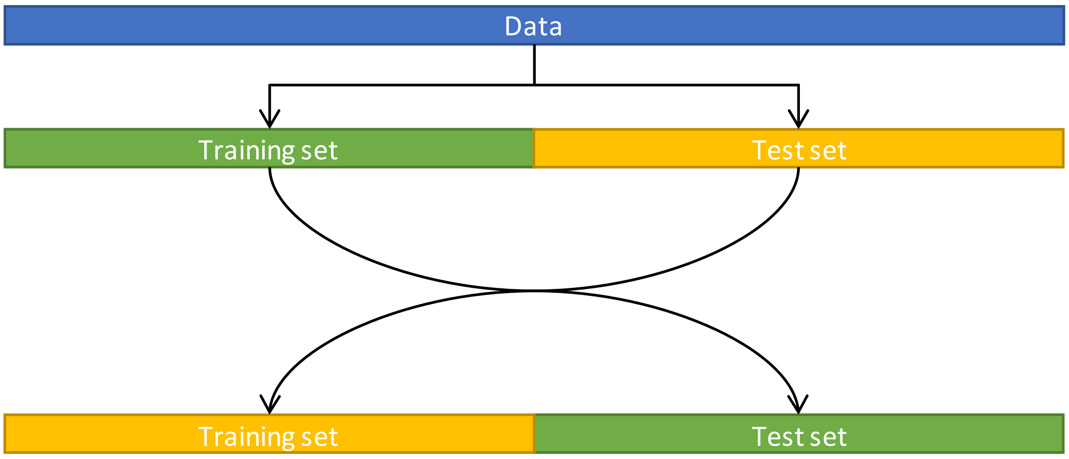

Figure 11. An iteration of a twofold cross-validation. The whole dataset is divided into equal sized training set and test set. After the networks are trained and tested, the sets exchange their functions. Now the training set is the new test set and the test set is the new training set. Networks are trained and tested again. This is repeated five times. On each iteration the data is shuffled.

Then we computed the difference between these accuracies in value $p_i^{(1)}$ for the first fold and $p_i^{(j)}$. As this was done five times, we obtained ten $p_i^{(j)}$, with $i = 1, \dots 5$ and $j = 1,2$. Finally, we computed the f value with the following formula:

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{2 \sum_{i=1}^{5} s_i^2}$$

With:

$$s_i^2 = \left(p_i^{(1)} - \bar{p}_i\right)^2 + \left(p_i^{(2)} - \bar{p}_i\right)^2 \text{ and } \bar{p}_i = \frac{p_i^{(1)} + p_i^{(2)}}{2}$$

The value of f determines whether the classifiers have the same error rates. If f is below 4.74 then we assumed they have the same error rate and vice-versa (Alpaydin, 1999).

The only modification done to this method was that we did not train both classifiers, but rather the benchmark was trained and the network given by the algorithm received the same training method that was being tested, i.e. in the "Train after Prune" method, the best individual found was only trained after being pruned from the benchmark individual, in the "No Training" method, the best individual found was not trained after pruned, and in the "Train from Scratch" method, the best individual was trained in the same way as the benchmark individual.

5.6 EVALUATION

5.6.1 Evaluation of the performance of the EA

To evaluate the Evolutionary Algorithm, we tried to determine its effectiveness in providing a network that is better when and if it exists. For this, we took which percentage out of the individuals tested on the 5x2 cv combined F test, had an improved accuracy and an f value below 4.74. This percentage would determine the chance that the algorithm provides an improved network.

5.6.2 Evaluation of each method

To compare the methods and try to determine which is best, we used the results from the combined 5x2 cross-validation F test. This test provided three numbers, the benchmark individual's performance on unseen data, the best individual's performance on unseen data and an f value which determines whether these two networks generate the same error rate.

Any two networks which provided an F test of 4.74 or higher was discarded. The two remaining values were compared to see whether the best individual found improves or deteriorates the performance of the network. Using these values we obtained an expected output for each method and dataset and compared them.

6 RESULTS AND DISCUSSION

On this section, we will present the results and analyse them in order to see whether the objectives were fulfilled or not. For each dataset we birthed 10,200 individuals over ten seeds for each method. On Table 4 there is a small summary of each dataset and the networks' characteristics. There is also the percentage of search space covered on each run, i.e. the space the evolutionary algorithm will cover on a single 50-generation run. The total search space was calculated as 2^L with L being the length of the string that represents an individual. Each dataset has a different string size but the algorithm was limited to a population of 20 individuals.

Table 4. Networks characteristics. Their sizes are given in an array that represents their input size, their hidden size and their output size. The Retrain epochs are only for the "Train after Prune" method. The Space Covered references the percentage of the search space covered on each run for each seed.

	Networks Characteristics				EA's Characteristics	
	Size	Examples	Epochs	Retrain Epochs	String size	Space Covered
Iris	[04 05 3]	150	80	20	47	7.25×10^{-10} %
P-Diabetes	[08 05 2]	768	30	15	66	1.38×10^{-15} %
Breast-W	[09 05 2]	699	20	10	73	1.08×10^{-17} %
Wine	[13 05 3]	178	15	10	119	1.53×10^{-31} %
Ionosphere	[34 10 2]	351	40	20	428	1.47×10^{-124} %
Sonar	[60 10 2]	208	60	30	740	1.76×10^{-218} %

To analyse the methods, we took a few measures from each dataset. The first measurement is the Number of Optimal Individuals Found (NIF), these are all the different individuals that have the same fitness than the best individual found by the evolutionary algorithm. Because the search spaces are very large in comparison with the number of training examples, it was very likely that many network topologies shared the same accuracy on the validation set, but a different one on the test set. Since the EA only tested on validation data, these networks could be missed, therefore we saved them and tested, as explained on Section 5.4.6. These will be referred to as individuals found from now on, not to be confused with the individuals generated during the whole evolutionary process. The first filter we needed to make was to remove those networks that have an f value higher than 4.74. These networks generated different error rates and should not be compared. The value that shows the amount of networks that had a feasible f value is called Percentage of Accepted Networks (PAN). Out of the PAN, we took the percentage that performed better than the original fully connected network, that is, that had a better accuracy on test data. This measurement was called Percentage of Better Individuals (PBI). This value could be interpreted as "the probability that the best candidate solution found by the EA has a performance better than the fully connected network". Next, to see which method provided better improvements or worse deteriorations, we took the Average Improvement (AI) and Average Deterioration (AD) for each of the individuals found. Networks that did not show improvement were included in the deteriorated group. Finally, from these values we calculated the Expected Output (EO). This value determines whether or not the general outcome of the evolutionary algorithm is positive or negative. The formula for the Expected Output is:

$$EO = PAN(PBI * AI + (1 - PBI) * AD)$$

These values were obtained for each seed on each method for each dataset. In total, there are 18 tables with this information. An example of this table is Table 5, obtained from the Sonar dataset using the Train after Prune method. As it can be observed, for seed 10 there was only one individual found, while on seed 928 we obtained 112 individuals with the same 'best' fitness. While on seed 10, the individual was actually better than the fully connected one, on seed 928 only 94.64% performed better on unseen data than the fully connected one.

Table 5. Data obtained from running the Train after Prune on ten different seeds on the Sonar datasets. NIF: Number of Optimal Individuals Found. PAN: Percentage of accepted networks. PBI: Percentage of Better Individuals. AI: Average Improvement. AD: Average Deterioration. EO: Expected Output.

Seed	NIF	PAN	PBI	AI	AD	EO
10	1	100.00%	100.0%	1.92	0.00	1.92
175	4	100.00%	75.0%	3.24	-0.77	2.24
247	1	100.00%	100.0%	3.37	0.00	3.37
300	1	100.00%	0.0%	0.00	-0.19	-0.19
465	5	100.00%	40.0%	0.14	-0.45	-0.21
574	10	90.00%	88.9%	2.54	-0.67	1.96
663	2	100.00%	100.0%	3.65	0.00	3.65
750	55	96.36%	90.6%	3.61	-0.99	3.10
891	9	100.00%	77.8%	1.06	-0.77	0.65
928	112	94.64%	100.0%	3.57	0.00	3.38
Average	20.00	98.10%	77.2%	2.31	-0.38	1.99
Std. Deviation	36.21	3.43%	32.9%	1.45	0.39	1.46

Out of these tables, we obtained an average and a standard deviation for all seeds as a way to summarize the data and be able to compare it with other datasets and other methods. These are shown in the following tables (Table 6, Table 7 and Table 8). For simplicity, the same acronyms will be used, with the difference that from now on these make reference to the ten seed average.

Table 6. Mean and standard deviation results from ten runs on each dataset for the No Training method. NIF: Number of Optimal Individuals Found. PAN: Percentage of accepted networks. PBI: Percentage of Better Individuals. AI: Average Improvement. AD: Average Deterioration.

	No Training					
	NIF	PAN [%]	PBI [%]	AI	AD	EO
Iris	$16.00_{20.44}$	$64.44_{34.66}$	$0.00_{0.00}$	$0.00_{0.00}$	$-27.31_{12.22}$	$-19.45_{12.24}$
Diabetes	$4.80_{6.75}$	$71.61_{38.53}$	$0.00_{0.00}$	$0.00_{0.00}$	$-11.88_{5.89}$	$-9.94_{6.97}$
Breast	$41.60_{102.21}$	$85.89_{14.14}$	$0.00_{0.00}$	$0.00_{0.00}$	$-13.52_{3.46}$	$-11.47_{3.33}$
Wine	$43.80_{87.30}$	$57.98_{35.14}$	$0.00_{0.00}$	$0.00_{0.00}$	$-29.57_{4.88}$	$-16.15_{9.57}$
Ionosphere	$6.70_{16.98}$	$48.39_{44.78}$	$0.00_{0.00}$	$0.00_{0.00}$	$-19.58_{16.97}$	$-15.68_{14.42}$
Sonar	$9.10_{20.74}$	$73.94_{41.57}$	$0.00_{0.00}$	$0.00_{0.00}$	$-13.37_{7.43}$	$-12.45_{7.53}$

On Table 6 we can see the Percentage of Accepted Networks (PAN), Number of Optimal Individuals Found (NIF), the Percentage of Better Individuals (PBI), the Average Improvement (AI) and Deterioration (AD) and the Expected Output (EO) for the No Training method. From this table we can see clearly that this method is completely inviable. No dataset on any seed produced a single improved network as evidenced by the PBI being 0.00 on every example tested. Therefore this method is clearly incompatible with our initial objectives. The algorithm could not find any network that improved upon the original fully connected one. All of the EO calculations are negative showing that this method will return a deteriorated network most certainly. We will not venture to say that avoiding training after pruning is definitely a bad idea in general as this is not the aim of this paper and we do not have sufficient data to give such a statement. However, we can say that using evolutionary algorithms in the way we did, we will not find any improvements in the networks by pruning them, and using them without further training.

The next two methods have found improved networks on all datasets. The results vary greatly for each dataset and a more in depth look is required. We can say that for all the datasets, the standard deviation is quite high in comparison to its mean, implying that the data is widely spread and a single run of the algorithm can produce a highly different result from the mean. This aspect definitely hinges the confidence of the algorithm's effectiveness and it is definitely something to bear in mind in the following discussions.

Table 7. Mean and standard deviation results from ten runs on each dataset for the Train after Prune method. NIF: Number of Optimal Individuals Found. PAN: Percentage of accepted networks. PBI: Percentage of Better Individuals. AI: Average Improvement. AD: Average Deterioration. EO: Expected Output.

	Train after Prune					
	NIF	PAN [%]	PBI [%]	AI	AD	EO
Iris	$172.50_{208.42}$	$97.78_{1.98}$	$23.69_{28.57}$	$0.44_{0.40}$	$-2.41_{1.82}$	$-1.91_{2.12}$
Diabetes	$4.40_{2.99}$	$100.00_{0.00}$	$38.19_{44.15}$	$0.31_{0.42}$	$-1.04_{0.77}$	$-0.60_{1.05}$
Breast	$225.10_{279.79}$	$98.97_{1.14}$	$70.33_{35.06}$	$0.38_{0.29}$	$-0.14_{0.10}$	$0.27_{0.37}$
Wine	$203.20_{192.81}$	$93.63_{10.51}$	$14.79_{23.79}$	$0.51_{0.60}$	$-2.47_{1.57}$	$-1.99_{1.59}$
Ionosphere	$43.00_{127.20}$	$97.70_{6.29}$	$61.63_{44.03}$	$1.47_{1.24}$	$-1.00_{1.08}$	$0.62_{1.87}$
Sonar	$20.00_{36.21}$	$98.10_{3.43}$	$77.22_{32.90}$	$2.31_{1.45}$	$-0.34_{0.34}$	$1.99_{1.46}$

For the Iris dataset, the "Train after Prune" (TAP) method (Table 7) found on average 172.5 individuals with the same best fitness. Out of these, 97.78% had an f value below 4.74. From this subsets, only 23.69% were improved pruned versions of the fully connected network. The average improvement was just 0.44 percentage points on unseen data, whereas the average deterioration was -2.41 percentage points. Out of these values, we reached an Expected Output of -1.91 percentage points, meaning that in general terms, the TAP method will deteriorate a fully connected network for the Iris dataset by this amount.

For the PIMA-Diabetes dataset, the TAP method retrieved just 4.40 individuals on average, but with a PAN of 100%, meaning all of them were acceptable under the F test criterion. This method provided a 38.19% mean of better individuals, from which the average improvement was only 0.31 percentage points while the deterioration was -1.04 percentage points. The Expected Output for this dataset was -0.60 percentage points, a small deterioration on average.

As for the Breast dataset, this method had a mean value of 225.10 individuals found with a 98.97% PAN. The subset of accepted networks had a 70.33% mean of better individuals, out of which the average improvement was 0.38 percentage points and -0.14 percentage points of average deterioration. On this dataset we found our first positive, although small, Expected Output, with a mean value of 0.27 percentage points.

Regarding the Wine dataset, the "Train after Prune" method found 203.20 individuals on average with the same best fitness, but only 93.63% out of these had an f value below 4.74. This PAN value was the lowest out of all from this method. Out of the accepted networks, only 14.79% improved the fully connected network with an average increase of 0.51 percentage points, while the rest had an average decrease of -2.47 percentage points. The EO once again was negative, being -1.99 percentage points. This was the lowest value found on the TAP method, meaning that, on average, the Wine dataset's networks will suffer the most from this method.

On the Ionosphere dataset the TAP method had a NIF average of 43.00 and a PAN of 97.70%. From the subset of accepted individuals, a 61.63% were better than the fully connected network, with an average improvement of 1.47 percentage points. Those that performed worse, had an average deterioration of -1.00 percentage points on unseen data. As with the Diabetes set, the Expected Output was positive yet low, with 0.62 percentage points.

The last dataset tested on the "Train after Prune" method was the Sonar dataset, where this approach found on average 20 individuals with the same better fitness. Out of these, 98.10% were accepted as they have an f value below 4.74. Out of this subset, 77.22% performed better than the fully connected one with an average improvement of 2.31 percentage points. The remaining deteriorated the network with a mean value of -0.34 percentage points. These values led to an Expected Output of 1.99 percentage points. This was the highest value found on the "Train after Prune" approach.

Table 8. Mean and standard deviation results from ten runs on each dataset for the Train From Scratch method. NIF: Number of Optimal Individuals Found. PAN: Percentage of accepted networks. PBI: Percentage of Better Individuals. AI: Average Improvement. AD: Average Deterioration. EO: Expected Output.

	Train from Scratch					
	NIF	PAN [%]	PBI [%]	AI	AD	EO
Iris	$71.40_{136.62}$	$90.82_{18.11}$	$42.85_{39.41}$	$0.41_{0.30}$	$-1.08_{1.68}$	$-0.21_{0.69}$
Diabetes	$3.40_{2.67}$	$100.00_{0.00}$	$61.25_{44.27}$	$0.60_{0.61}$	$-0.28_{0.45}$	$0.33_{0.94}$
Breast	$60.20_{144.03}$	$99.65_{0.80}$	$32.03_{38.14}$	$0.12_{0.16}$	$-0.14_{0.08}$	$-0.03_{0.19}$
Wine	$17.20_{25.15}$	$96.61_{5.84}$	$7.35_{16.22}$	$0.10_{0.18}$	$-2.13_{1.13}$	$-1.95_{1.21}$
Ionosphere	$2.10_{1.45}$	$100.00_{0.00}$	$78.00_{41.58}$	$2.33_{1.73}$	$-0.31_{0.50}$	$2.05_{2.04}$
Sonar	$1.90_{0.99}$	$100.00_{0.00}$	$74.17_{33.44}$	$2.70_{1.92}$	$-0.59_{0.75}$	$1.73_{1.64}$

On the Iris dataset, the "Train from Scratch" (TFS) method (Table 8) found on average 71.40 individuals with only 90.82% of them being acceptable. This value had the lowest average, but also the highest standard deviation, so we could attribute the low mean to an unnormal run out of the ten seeds. From the accepted networks, only 42.85% were better than the fully connected network with an average improvement of 0.41 percentage points. The rest, deteriorated the performance by an average of -1.08 percentage points. The Expected Output was -0.21 percentage points.

On the PIMA-Diabetes dataset, the "Train from Scratch" approach retrieved on average just 3.40 individuals with 100% of them being acceptable. Out of these, 61.25% were better individuals with an average improvement of 0.60 percentage points. The average deterioration was -0.28 percentage points which led to an Expected Output of 0.33 percentage points.

Regarding the Breast datasets, this method had a NIF of 60.20 with a PAN of 99.65%. The Percentage of Better Individuals obtained is 32.03% with an average improvement of only 0.12 percentage points and an average deterioration of -0.14. The EO was -0.03, from which we can say that the network doesn't benefit nor suffer from pruning, as if it were indifferent to it.

On the Wine dataset, the "Train from Scratch" method found a mean value of 17.20 individuals out of which 96.61% are acceptable. The method had only 7.35% of improved individuals with an average improvement of just 0.10 percentage points and an average deterioration of -2.13 percentage points. The Expected Output was -1.95 percentage points, being this the network that most suffered from this approach.

The TFS method found just 2.10 individuals on average on the Ionosphere dataset with 100% of them having an f value below 4.74. The PBI equated to 78.00%, with an average improvement and deterioration of 2.33 and -0.31 percentage points respectively. The Expected Output was 2.05, which was the highest out of all the methods tested.

Finally, on the Sonar dataset, the "Train from Scratch" method retrieved an average of 1.90 individuals with 100% being acceptable i.e. having an f value lower than 4.74. The PBI was 74.17% with an average improvement of 2.70 percentage points and -0.59 percentage points of average deterioration. Lastly, the Expected Output for this dataset was 1.73 percentage points.

After going through the results, we realised that they are very dataset dependent. There was not one method that clearly stand out as they vary across every different set. Therefore, we compared the datasets number of examples and string size of their respective networks to be able to find any explanations to the results (Figure 12). The first one determines the amount of training examples used for the network and the second one the number of total connections.

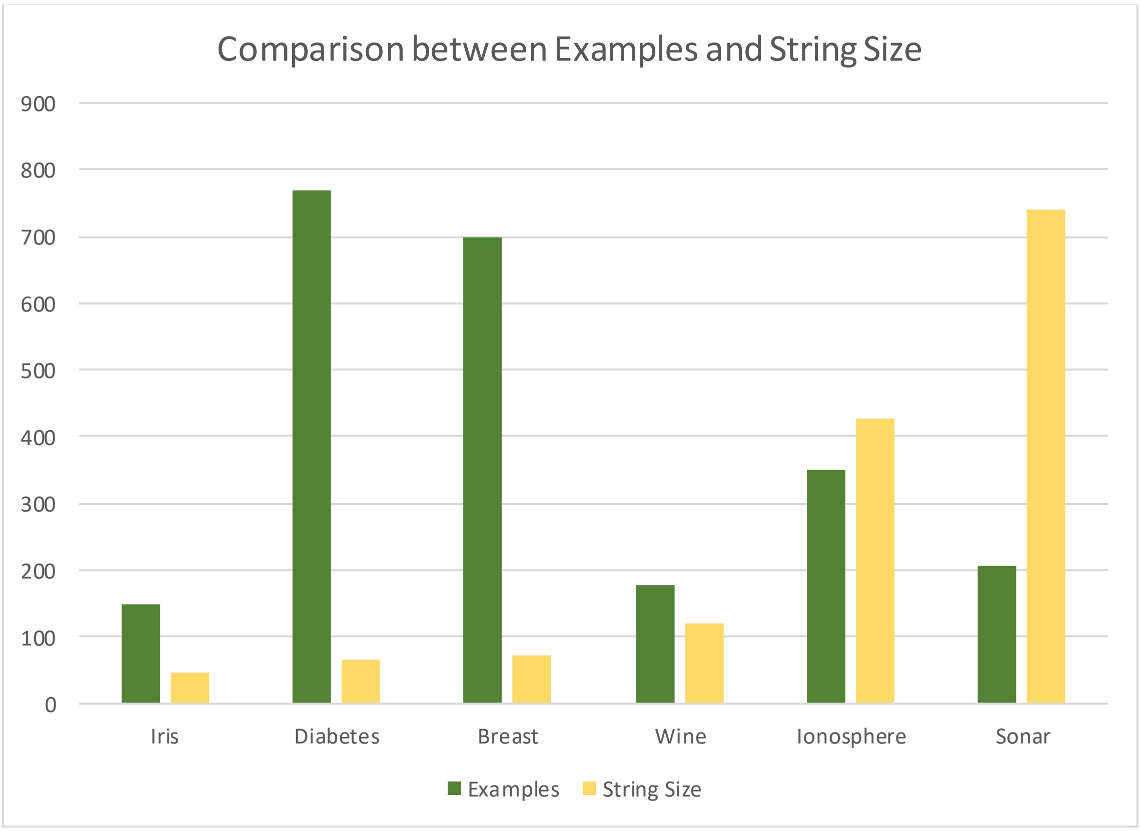

Figure 12. A comparison between datasets number of examples and string size.

At first glance, we note that the first four networks i.e. Iris, Diabetes, Breast and Wine are comparable between each other regarding size, while the Ionosphere and Sonar are the larger ones. On another note, the Diabetes and Breast networks have the largest number of examples out of all of them.

To compare the methods appropriately, a box and whiskers graph was created using each dataset's Expected Output values out of their ten runs (Figure 13). On this graph, we can notice the spread of the data as well as all the means next to each other. One thing to note is that the "Train after Prune" approach has, in general, the data more spread than the "Train from Scratch". This means that there is a higher variability among runs which leads to less confidence on the algorithm itself. However, we find that the spread on both methods is large and it is something to improve on later iterations.

Examining both the Iris and Wine datasets, we see that both methods are ineffective in finding an improved network, which leads us to believe that these networks simply do not benefit from pruning at all. These networks are deemed small on this project and both have few examples to train from.

If we look at the PIMA-Diabetes set, the "Train from Scratch" method did have a positive Expected Output, although small. The TAP method on the other hand had a negative EO, also small. We would not say that this dataset benefits nor suffers from either method, however, the results leads us to believe that it is not worthy to use these approaches on it, as the benefits that may come out are not good enough.

Contrary to other datasets, the Breast set was the one with less variability on its results. Same as the Diabetes set, the benefits that can come out from pruning this network are low, although in this case is the TAP approach that had better results. We have not found any reason as to why this difference appeared. One idea that floated around, was that on the Diabetes set, the epochs used to train were enough or too much, and the extra epochs on the TAP method overfitted the network to the training data. On the other hand, the Breast set may have needed more epochs on its initial training to reach saturation, which was achieved by retraining it later. However, this is conjecture as no real tests were performed to determine whether this is correct or not.

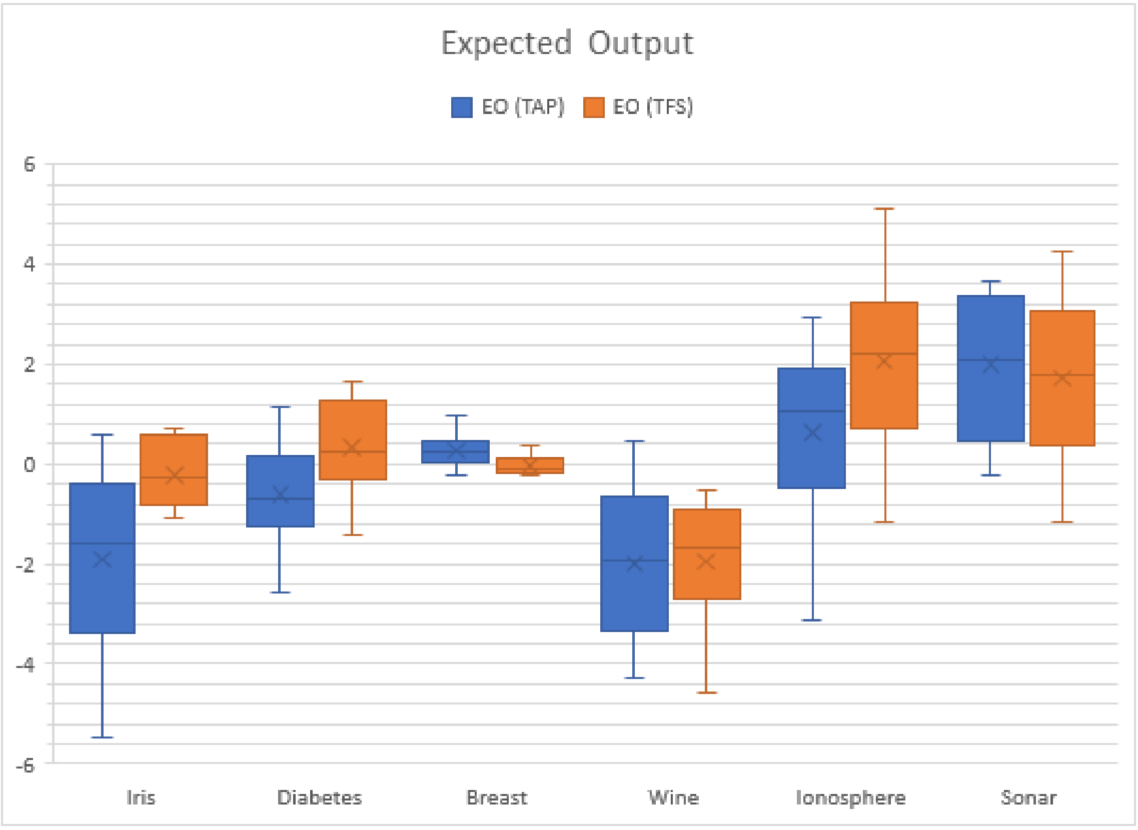

Figure 13. Expected Output for each dataset. On blue, the Expected Output for each dataset using the "Train after Prune" method. On orange, the Expected Output for each dataset using the "Train from Scratch" method.

The Ionosphere dataset was the first set to show great benefits from both approaches, with the TFS method having really good EO. Furthermore, almost the whole box and whisker plot for this method is on the positive section, meaning that few negative results were found. If we were to state which method is better, in this case, the TFS method clearly wins, as it has a higher mean output while the spread on both is similar.

Similarly, the Sonar dataset benefited from both methods although, in this case, the benefit was similar on both approaches. This leads us to believe that this network greatly benefits from pruning. If we were to decide on which method works best, we would be inclined to say that the TAP approach has a slight advantage as the data is not as spread as with the TFS approach.

These dataset's networks departed greatly in size than the previous, which only had five units on their hidden layer and up to 13 units on their input layer. These one doubled the hidden units to 10 and greatly increased the input layer's unit to 34 and 60. This not only increased the number of connections available to prune but also increased greatly the search space. This is why, for these datasets, we can say that the algorithm is useful in finding an improved pruned network with both methods.

Another noteworthy aspect is that the Diabetes and Breast sets have relatively smaller spread overall, when taking into account both methods. As said before, these methods have the highest number of examples, which also leads us to believe that this may have an effect on the variability of the results.

Regarding the two approaches in general, we do not have enough data to say for certain that one is better than the other. They seem extremely data dependant, and we cannot give a definitive verdict. We can see however, that they seem most effective on larger networks, as they have more connections to prune. More connections mean that there is a higher chance that there exist unnecessary ones, therefore it makes sense that pruning is most effective on them. This is supported by the fact that none of the smaller networks showed significant improvement on unseen data.

To evaluate the evolutionary algorithm as a whole, we can take a look at the PBI data spread (Figure 14). Because we ruled out the four smaller datasets as being benefited from pruning at all, we will only look for the Ionosphere and Sonar networks. It does not make sense to test an algorithm's ability of finding improved networks on a dataset that generally will not benefit from it. On another note, the Percentage of Accepted Networks on both these datasets were quite high, being 100% on the TFS approach, therefore we will not take it into consideration as a measure for the algorithm's effectiveness.

The first thing to note is that in both datasets, both methods had runs with zero better individuals. This would mean the algorithm failed to provide an improved version. Moreover, the data is largely spread, especially on the Ionosphere set using the TAP approach. However, the data is skewed to the lower values on every box, meaning that generally, the PBI is high. Plus, on the Ionosphere-TFS chart and Sonar-TAP chart, the zero PBI is treated as an outlier. All this means, that in general, the PBI will be high on this sets.

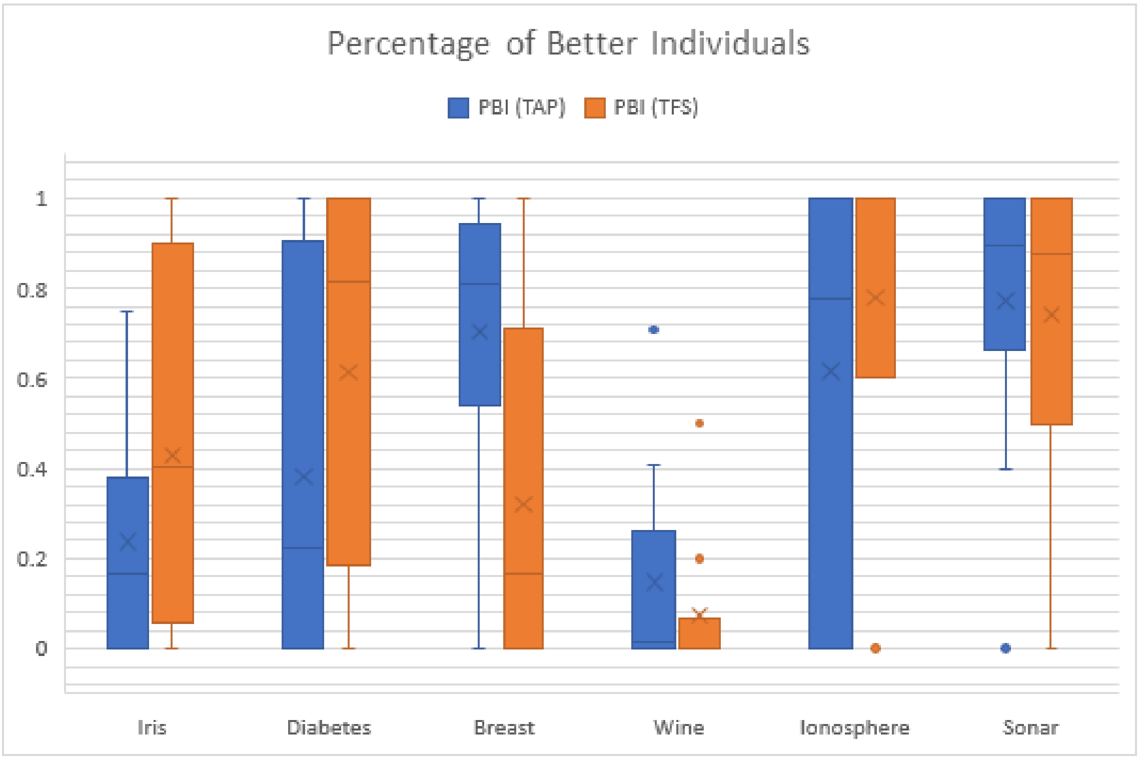

Figure 14. Box and whisker plot showing the Percentage of Better Individuals (PBI) for each dataset. On blue, the PBI for each dataset using the "Train after Prune" method. On orange, the PBI for each dataset using the "Train from Scratch" method.

If we think of the PBI as “the probability that a pruned network found by the algorithm has an F value lower than 4.74 and performs better than the fully connected network”, then we can say that this is a positive result, and that the algorithm is generally effective in finding an improved version. Nevertheless, we know that the networks are better because we tested them. On a real application, the testing would be the final step, and would not be included in the algorithm as a whole nor be a tool to choose a network. In this case, it was used to evaluate the general performance of the methods and algorithms. This is why, we would prefer a smaller spread, to have a certainty about the PBI. To be able to say, this algorithm has an X amount of probability to return an improved network.

7 CONCLUSION

The development of neural networks has proven to be complex and relying heavily on experience. This may deter newcomers to the area and hinders the work from seasoned developers. There is a lack of procedures that help notice or hint at which parameters are optimal for the neural network, other than trial and error. Plus, real world applications require cheaper, smaller and faster nets that are also effective. Pruning is a method that can help achieve this last requirement, even though not enough pruning methods are being used. There is also an ongoing discussion about what to do with the networks after pruning them, whether it is avoid extra training, training extra epochs, or instantiate a new network with the connections already pruned. On this paper, we developed three evolutionary algorithm with the objective of finding improved pruned networks out of a fully connected one and to evaluate its effectiveness. Each algorithm used one of the three methods of training after pruning, i.e. no training, train extra epochs and training from scratch, to compare and try to determine which of these is the most suitable.

Our results gave at least one definitive answer, the “No Training” approach is not suitable for this type of search, as not one improved network was found. As stated previously, this does not invalidate the idea of not training after pruning in general, but only as an approach in this algorithm. From the other two methods, we got different results for each dataset, but neither stood out clearly to state definitively which one is the best. However, our results also indicated that larger networks appear to be more suitable for pruning that smaller ones, as the Ionosphere and Sonar datasets showed the best improvements out of all the dataset. We would require more tests to dismiss this as a coincidence and state it as a general rule. On another note, our results were highly scattered with elevated standard deviations. This is not a minor aspect and should be addressed in future iterations of the algorithm. We questioned whether this was caused by the low number of examples for the datasets, as those with a higher number had smaller variations.

As for future work, there are several aspects on which to improve and others to test. First off, we need to reduce the variability of the results. This way we can provide a numerical value with a higher confidence that it is the normal value rather than just a mean. We may look into datasets with more training examples, or to modify the fitness value. This last piece can lead into a higher scrutiny on which is the “best individual” out of all those which have the same accuracy on validation set and even point to different networks that might be missed. There are several different aspect of the network we can look into such as number of connections pruned, or which weights are being pruned to help the fitness function improve. Second, we need to prove whether the largest datasets are better suited for pruning or not. If this was the case, we also need to test the scalability of the algorithm itself, as larger networks imply larger binary strings that represent the individual as well as larger search spaces. A final aspect to improve upon is on the algorithm’s run time. Both approaches that provided positive values had variable and sometimes very high run times, specially the “Train from Scratch” method. Run time was not a variable considered on this project but it is important nonetheless. Training over 1000 networks on each algorithm run can be time consuming. To do so, we would need to rework the fitness function to try to determine before hand if the pruning will improve or deteriorate the network without actually pruning it.

Even though the improvements are extensive, we find that the overall outcome of this project was positive, as we could rule out completely a method (i.e. “No Training”), and we developed a base algorithm that found positive results. Furthermore, we had a hint on which networks would be most suited for pruning which can take us a step forward on understanding the mechanisms lying behind the design of a network.

8 BIBLIOGRAPHY

Alpaydin, E. (1999). Combined 5x2 cv F Test for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, *11*(8), 1885–1892. https://doi.org/10.1162/089976699300016007

Alpaydin, E. (2014). *Introduction to Machine Learning* (T. Dietterich, C. Bishop, D. Heckerman, M. Jordan, & M. Kearns (eds.); 3rd ed.). The MIT Press.

Anwar, S., Hwang, K., & Sung, W. (2017). Structured Pruning of Deep Convolutional Neural Networks. *ACM Journal on Emerging Technologies in Computing Systems*, *13*(3), 1–18. https://doi.org/10.1145/3005348

Belew, R. K., Mcinerney, J., & Schraudolph, N. N. (1990). *Evolving Networks: Using the Genetic Algorithm with Connectionist Learning*.

Blalock, D., Ortiz, J. J. G., Frankle, J., & Guttag, J. (2020). What is the State of Neural Network Pruning? *Proceedings of Machine Learning and Systems 2020*.

Branke, J. (1995). Evolutionary Algorithms for Neural Network Design and Training. *Proceedings of the First Nordic Workshop on Genetic Algorithms and Its Applications*, 145–163.

Cantú-Paz, E. (2003). Pruning neural networks with distribution estimation algorithms. *Genetic Evolutionary Computation Conference*, 790–800.

Cantú-Paz, E., & Kamath, C. (2005). An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *35*(5), 915–927. https://doi.org/10.1109/TSMCB.2005.847740

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray.

De Jong, K. A. (2006). *Evolutionary Computation: A Unified Approach*. The MIT Press.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1923. https://doi.org/10.1162/089976698300017197

Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/ml

Hancock, P. J. B. (1992). Pruning Neural Nets by Genetic Algorithm. In I. Aleksander & J. Taylor (Eds.), *Artificial Neural Networks* (pp. 991–994). North Holland. https://doi.org/10.1016/B978-0-444-89488-5.50036-1

Harris, C. R., Millman, K. J., van der Walt, S. J., & Gommers, R. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

He, Y., Kang, G., Dong, X., Fu, Y., & Yang, Y. (2018). Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2234–2240. https://doi.org/10.24963/ijcai.2018/309

Kaggle. (2016). *Pima Indians Diabetes Database*. UCI Machine Learning. https://www.kaggle.com/uciml/pima-indians-diabetes-database

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, *4*(4), 461–476.

Lewontin, R. C. (1970). The Units of Selection. *Annual Review of Ecology and Systematics*, *1*, 1–18. https://doi.org/10.1146/annurev.es.01.110170.000245

Maclin, R., & Opitz, D. (1999). Popular Ensemble Methods: An Empirical Study. *Journal Of Artificial Intelligence Research*, *11*, 169–198. https://doi.org/10.1613/jair.614

Marshall, S. J., & Harrison, R. F. (1991). Optimization and training of feedforward neural networks by genetic algorithms. *1991 Second International Conference on Artificial Neural Networks*, 39–43. https://doi.org/0-85296-531-1

McKinney, W., Van den Bossche, J., Garcia, M., Zeitlin, M., Hawkins, S., Li, T., & Wörtwein, T. (2021). *pandas: powerful Python data analysis toolkit*. Pandas. https://pandas.pydata.org/docs/index.html

Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve it: Modern Heuristics* (2nd ed.). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-662-07807-5

Miikkulainen, R. (2011). Topology of a Neural Network. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning*. Springer. https://doi.org/10.1007/978-0-387-30164-8_837

Newman, M. E. J. (2010). *Networks an Introduction*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

Reed, R. (1993). Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, *4*(5), 740–747. https://doi.org/10.1109/72.248452

Setiawan, W. P. A., Gumelar, A. B., Rizqi, M., Putra, F. D. C., Romadhonny, R. A., & Nugroho, R. D. (2019). Development of First-Person Shooting Games Using Human Voice Command and its Potential Use for Serious Game Engines. *2019 International Seminar on Application for Technology of Information and Communication (ISemantic)*, 110–115. https://doi.org/10.1109/ISEMANTIC.2019.8884261

Soni, D. (2018). *Introduction to Evolutionary Algorithms*. Towards Data Science. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac

Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, *14*(3), 347–361. https://doi.org/10.1016/0167-8191(90)90086-O

Whitley, Darrel. (2001). An overview of evolutionary algorithms: Practical issues and common pitfalls. *Information and Software Technology*, *43*(14), 817–834. https://doi.org/10.1016/S0950-5849(01)00188-4