Genetic Programming and Evolvable Machines (2024) 25:7
https://doi.org/10.1007/s10710-024-09481-7

Neural network crossover in genetic algorithms using genetic programming

Kyle Pretorius[1] · Nelishia Pillay[1]

Received: 23 April 2023 / Revised: 20 December 2023 / Accepted: 19 January 2024

Abstract
The use of genetic algorithms (GAs) to evolve neural network (NN) weights has risen in popularity in recent years, particularly when used together with gradient descent as a mutation operator. However, crossover operators are often omitted from such GAs as they are seen as being highly destructive and detrimental to the performance of the GA. Designing crossover operators that can effectively be applied to NNs has been an active area of research with success limited to specific problem domains. The focus of this study is to use genetic programming (GP) to automatically evolve crossover operators that can be applied to NN weights and used in GAs. A novel GP is proposed and used to evolve both reusable and disposable crossover operators to compare their efficiency. Experiments are conducted to compare the performance of GAs using no crossover operator or a commonly used human designed crossover operator to GAs using GP evolved crossover operators. Results from experiments conducted show that using GP to evolve disposable crossover operators leads to highly effectively crossover operators that significantly improve the results obtained from the GA.

Keywords Genetic programming · Genetic algorithms · Neural networks · Evolutionary algorithms · Crossover operator

Area Editor: Sebastian Risi.

This work was funded as part of the Multichoice Research Chair in Machine Learning at the University of Pretoria, South Africa. This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number 138,150). Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the NRF.

✉ Kyle Pretorius
u16234805@tuks.co.za

Nelishia Pillay
nelishia.pillay@up.ac.za

[1] Department of Computer Science, University of Pretoria, Pretoria, South Africa

Published online: 21 February 2024

1 Introduction

The crossover operator has always been a controversial topic surrounding neuroevolution [1], with the general consensus being that crossover is not a beneficial or necessary operator to include in a genetic algorithm (GA) that optimizes neural networks (NNs) by evolving their weights. The main reasoning behind this belief is that the crossover operator has been shown to be highly destructive in the past and therefore is expected to produce offspring with worse fitness than that of its parents [2]. One of the main explanations for the destructiveness of crossover when applied to NN weights is known as the permutation problem [3–5]. The permutation problem refers to the fact that there exists a one to many mapping between the NN phenotype and genotypes, meaning there exists many different chromosomes that represent functionally equivalent NNs [5]. This is because the neurons within a layer of a NN can be arbitrarily permutated or rearranged without having any effect on the output of the NN. This means that it is not possible to determine which two weights in two different NNs are functionally equivalent when performing crossover.

Most works in neuroevolution choose to completely omit the crossover operator and use a GA that only consists of mutation and selection operators [6]. While these GAs perform well and on the surface appear to operate normally without a crossover operator, the omission of the crossover operator may have a more detrimental effect on the behaviour and performance of a GA than what is initially apparent. This is because crossover is a core operator of a GA and plays an important role in allowing the GA to take large steps in the search space while also promoting convergence of a population to promising regions of the search space [7].

However, a recent study has suggested that the permutation problem should in theory not be such a major problem in GAs [3]. This is due to the fact that once the population converges it is highly unlikely that multiple representations of the same phenotype will exist since the many genotypes that map to the same phenotype should be spread out across the search space [8]. This argument has been termed the convergence argument by Froese et al. [8]. If the permutation problem is not of concern, it may be the case that existing crossover operators that are commonly used for NNs have not been designed well enough or that the design of crossover operators for NNs is a highly complex task. Designing effective crossover operators has been seen as one of the major remaining challenges surrounding neuroevolution for NN weight optimization [9].

For this reason, we explore the use of genetic programming (GP) [10] to automatically design crossover operators for NNs. GP is an evolutionary algorithm similar to GAs but are used to evolve programs or trees that can be executed or evaluated to produce a desired output. If we treat crossover operators as programs that can be applied to two weights, GP can be applied to search for crossover operators. Utilizing GP enables the design of highly complex crossover operators that might be nearly impossible for humans to design or that may seem counter intuitive. Furthermore, it also allows for crossover operators to be tailored to different problem domains by rerunning the GP on each problem domain.

As seen later in this paper, GP can also be used to design disposable crossover operators instead of reusable operators. Disposable crossover operators are optimized to perform crossover on two specific weight vectors and may perform poorly when applied to others. This seems undesirable at first since operators are produced for each weight pair which is more expensive, but this strategy may produce highly effective crossover operators that consistently outperform reusable operators. In this study, we compare the use of GP to design both reusable and disposable crossover operators.

The main contributions of this study are as follows:

- To propose a novel GP to evolve crossover operators that can be used in GAs that optimize NN weights;
- To use the proposed GP to evolve both reusable and disposable crossover operators and compare their performance;
- To show that GP evolved crossover operators are more effective than commonly used crossover operators when applied to NN weights;
- To show that including GP evolved crossover operators in GAs that optimize NN weights is beneficial and improves the results obtained from the GA.

Section 2 explores work related to this study, after which a high level overview of the GA used for NN weight optimization is provided in Sect. 3. The proposed GP used to evolve crossover operators is described in detail in Sect. 4, followed by a description of how GP is used to evolve both reusable and disposable crossover operators in Sect. 5. The experimental setup of this study is outlined in Sect. 6 after which the results are presented in Sect. 7. Finally, the paper is concluded in Sect. 8.

2 Related work

The use of GAs to evolve NN architectures has proven to be highly effective. The work of Real et al. [11] used a relatively simple GA without crossover to evolve a high quality NN architecture named AmoebaNet-A that was competitive with the best image classifiers [11]. However, this approach only evolved the NN architectures and the NN model was trained using gradient descent. Real et al. also employed evolutionary algorithms to simultaneously evolve NN architectures and weights in order produce a fully trained model [12]. The technique was able to produce models that performed competitively with human designed models on standard benchmarks. An interesting point to note is that both of these approaches completely omit the crossover operator, which is commonly seen in GAs that evolve NN weights [6, 12–14].

The crossover operator is seen as one of the remaining challenges surrounding the evolution of NN weights, especially when used on more modern NN architectures that are much larger than the architectures that were used when neuroevolution [1] was first proposed and studied. In this section, we look at other related work that has focused on designing new and safe crossover operators for NN weights.

One of the most well known neuroevolutionary technique named NeuroEvolution of Augmenting Topologies (NEAT) [15] contains a crossover operator that was specifically designed to overcome the permutation problem. This is accomplished by using historical markers that track genes throughout the lifetime of the GA, these markers can then be used to ensure that crossover is only performed between the compatible genes of the two parents leading to a less destructive crossover operator. As the name suggests, NEAT is not a fixed topology method, which means that the architecture or topology of the NN is evolved together with its weights. For this reason the crossover used was specifically designed for this use case. NEAT was originally designed and tested on small NNs solving relatively simple problems and has shown to struggle when applied to higher dimensionality problems [16]. NEAT was later extended to HyperNEAT [17] which uses indirect encoding to reduce the dimensionality of the problem of evolving NN architectures. This allowed HyperNEAT to scale to evolve larger NN architectures, however the architectures evolved by HyperNEAT are still relatively small in comparison to larger modern architectures with millions of weights [18–20].

A safe crossover operator designed by Uriot et al. [9] attempts to functionally align the neurons of the two parent NNs to mitigate the negative effects of the permutation problem. This crossover operator performs two steps, the first of which aligns the neurons within each layer of the parent NNs by measuring how well they correlate. After the neurons in each layer have been rearranged so that the highest correlating neurons align, an arithmetic crossover is performed by interpolating between the weights of the two parents. In their study, performing crossover by interpolating between the weights with and without neuron alignment was compared, where it was found that neuron alignment reduces the destructiveness of crossover. However, crossover was found to still on average produce offspring with worse fitness than their parents.

Imitation learning has also been used to create crossover operators that do not perform crossover in the parameter space but rather on the phenotype of individuals. Such a crossover operator was designed for genetic policy optimization (GPO) [21]. GPO is a GA that evolves NNs for reinforcement learning (RL) problems using policy optimization as its mutation operators and a modified imitation learning as its crossover operator. When using imitation learning as a crossover operator the offspring is essentially trained to behave similar to both parents, in the context of RL this is referred to as a state space crossover since the offspring will have state visitation distributions similar to both parents.

As seen above, most work done surrounding the crossover operator in neuroevolution is aimed at mitigating the effects of the permutation problem by either attempting to detect compatible neurons [15], align neurons [9] or avoid parameter space crossover entirely [21]. The convergence argument termed in [3] argues that the permutation problem should not be an issue once populations have started converging. The basis of this argument is that there is no room in a converged population for multiple genetic representations of the same phenotype to exist simultaneously.

The convergence argument was supported by a number of experiments conducted on standard benchmarks where it was shown that crossover is not as destructive as initially believed [3]. This study was the main inspiration for our work since it suggests that crossover between NN weights is not inherently destructive but likely just not designed well enough in previous work.

For this reason, this study aims to focus on designing crossover operators for NNs by leveraging GP. Using GP allows us to move away from human designed crossover and explore the use of more complex crossover operators that may not have been tested before.

3 GA for NN weight optimization

To provide a framework in which crossover operators for GAs can be evaluated, a simple GA that evolves a population of NN weights is used. During each generation this GA will apply the basic evolutionary operators such as selection, mutation and crossover. This allows us to compare the effectiveness of crossover operators by running the GA with different crossover operators and tracking the performance of the GA as a whole. Algorithm 1 gives a high level overview of the GA used in this study to evaluate crossover performance. This algorithm with fixed operator application rates was chosen to enable fine grained control over the crossover and mutation rates which was found to more consistently produce good results and more easily facilitates the use of dynamic crossover rates as described later. The use of application rates requires that the sum of all rates should be less than or equal to one. In the case where the sum of rates is less than one, the new generation is filled up to the required size using the selection operator. Furthermore, elitism is also used since NN weights are very brittle and elitism guarantees that the best weights remain in the population until they are successfully combined with other weights in the population. It should also be noted that the (μ, λ) variants of GAs and GP are used in this study. In other words, the offspring generated or selected for each new generation, replace the previous generation. The core components of the GA are described in more detail next.

Algorithm 1 GA overview

```
Require: mutation rate + crossover rate + elitism rate $\leq 1$
$M \leftarrow$ population size                                      ▷ $M \in \mathbb{N}$
population $\leftarrow \pi_0, ..., \pi_m$          ▷ Initialize NN population with random weights
$\mu \leftarrow$ mutation rate $* M$              ▷ Domain for all rates: r $\in \mathbb{R} \mid 0 \leq$ r $\leq 1$
$\gamma \leftarrow$ crossover rate $* M * 2$      ▷ Two crossover parents are required per offspring
$\iota \leftarrow$ elitism rate $* M$
$\alpha \leftarrow M - \mu - \gamma - \iota$
for each generation do
    EVALUATE(population)
    mutation parents $\leftarrow$ SELECT(population, $\mu$)
    mutation offspring $\leftarrow$ MUTATE(mutation parents)
    crossover parents $\leftarrow$ SELECT(population, $\gamma$)
    crossover offspring $\leftarrow$ CROSSOVER(crossover parents)
    elite $\leftarrow$ ELITE(population, $\iota$)
    population $\leftarrow$ mutate offspring + crossover offspring + elite +
    SELECT(population, $\alpha$)
end for
```

3.1 Individual representation

Each individual in the population represents a NN or more specifically the weights of a NN. The weight matrix of the NN is flattened to a 1D vector which is the chromosome or genotype of the individual. Hence, each individual does not need to be a functioning NN in itself but rather a 1D vector of weight values which can be used to set the weights of a NN before inference or training.

3.2 Fitness function

The fitness function is used to evaluate an individual's efficacy at solving the problem at hand. Any loss function that is typically used in the training of NNs could be used as a fitness function depending on the problem domain. Since this paper will focus on classification problems, categorical cross entropy loss is used as the fitness function. This loss measures how close the NNs predicted class probabilities are to the actual class labels for a set of input examples, where a lower value indicates a fitter individual. Hence, in the context of the GA, the fitness is being minimized and the selection operator should prefer individuals with a lower fitness.

3.3 Selection

To apply selection pressure during the evolutionary process a selection operator is used when selecting parent individuals that will be used to produce offspring for the next generation through crossover or mutation. Tournament selection is used in this study as it is a simple selection operator that is commonly used in GAs. Tournament selection randomly selects a number of individuals from the population which forms the tournament, the individual with the best fitness is then returned as the selected individual. The number of individuals randomly selected is determined by a parameter called the tournament size, a larger tournament size imposes a greater selection pressure and promotes convergence of the population due to a reduction in diversity in the population.

3.4 Mutation

In older literature surrounding neuroevolution for NN weight optimization the most common mutation operators would make random changes to the NN weights such as adding gaussian noise. However, in recent work the use of gradient descent as a mutation operator has become more popular and has proven to be more effective [21–23]. This essentially means that when an individual is mutated, it is trained for a number of epochs using gradient descent.

3.5 Crossover

The goal of the crossover operator is to combine the genotypes or chromosomes of two parent individuals in such a way that the produced offspring NN is phenotypically similar to both parents. As discussed earlier, this operator is commonly left out in neuroevolution when evolving NN weights since it is seen as destructive and produces offspring with much worse fitness than that of its parents. This operator is the main focus of this study as we aim to show that crossover between two NNs is not as destructive as previously reported. We propose using GP to produce crossover operators for NN weights and compare the results to using a simple commonly used crossover operator.

4 GP for crossover evolution

In this study, we propose using GP to design a crossover operator for weight optimization in NNs. The designed crossover operator should accept the weights of two NNs as input and produce the weights of the offspring as output. Using GP to evolve crossover operators comes with many advantages over using simple human designed crossover operators. Firstly, since GP is able to perform a multipoint search in the program space it is capable of evolving crossover operators that humans might not think of, either because they are highly complex or because the might be counter-intuitive. For this to be possible, the function and terminal set of the GP should be designed in such a way that it is as flexible as possible while not overcomplicating the search space. Another major advantage is that GP can design problem domain specific crossover operators. Since it stands to reason that the shape of the NN loss landscape likely plays a role in what crossover may be effective, GP can automatically discover an appropriate crossover operator for each problem it is applied to.

Furthermore, GP can be used to design crossover operators using two distinct strategies by either designing disposable or reusable crossover operators. This allows different crossover operators to be evolved for different use cases. Using GP to evolve disposable and reusable crossover operators is described in detail in the next section.

At a high level the GP that is used to evolve crossover operators follows the same algorithm as the GA shown in Algorithm 1. In the case of the GP the population as initialized on the first step of the algorithm are crossover operators represented by trees instead of NN weights.

4.1 Individual representation

Individuals within the population of the GP are crossover operators represented as trees. These trees essentially accept the weight vectors of the two parents as input, perform arithmetic on these vector and produce a single offspring weight vector as output. The terminal set of the GP is listed below:

- W1—weight vector of parent 1;
- W2—weight vector of parent 2;
- F1—fitness of parent 1;
- F2—fitness of parent 2;
- Constant values: 1, 2 and 0.5.

The fitness of the two parents are included to allow the GP to evolve smart trees that can select vectors based on logical conditions using these fitness values. The function set of the GP is as follows:

- +, -, *—Standard arithmetic operations;
- Sum(V1, V2)—sums two vectors element wise;
- Mean(V1, V2)—returns the mean of two vectors element wise;
- 1Point(V1, V2)—performs 1 point crossover between vectors V1 and V2 (selects a random point when the node is created which remains fixed for its lifetime);
- Select(C, V1, V2)—return V1 if C is true, else it returns V2;
- >, <—Logical operators to produce condition C for Select.

To guarantee that the tree produces valid output i.e a vector, strong typing is used. For example, the sub-tree that produces the condition C for the Select operator is generated in such a way that it produces a boolean output. In contrast, the sub-tree that produces V1 and V2 for the Select operator will always produce vector outputs. Furthermore, it should be noted that in order to be used in a tree that produces a single offspring weight vector, the 1Point crossover used in the function set is a specialized version of the classic one point crossover. This 1Point crossover only produces one offspring by only returning the first of the two offspring produced during one point crossover. In other words, the elements in the weight vector before the crossover point are used from parent 1, and the elements after the crossover point are used from parent 2.

An example of a tree representing a crossover operator is shown in Fig. 1. This simple crossover operator returns the weight vector of parent 1 if the fitness of parent 1 is more than double that of parent two, otherwise it returns the element wise mean of the two parent vectors.

4.2 Fitness function

The fitness of a tree/individual is determined by how well the tree is able to perform crossover and produce offspring weights given two parent NN weights. Since the GP is evolving crossover operators for the GA, the GPs fitness function depends on the GAs fitness function. More specifically, the GPs fitness function compares the GA fitness of the offspring produced by the crossover operator to the GA fitness of the parents given as input to the crossover operator. If the offspring produced by a the crossover has a better GA fitness relative to its parents, the crossover operator that produced the offspring should have a good GP fitness.

An overview of the GP fitness function is provided in Algorithm 2. The first notable part of this algorithm is that it uses a dataset referred to as the crossover train dataset that consists of pairs of NN weights that the crossover operator will be evaluated on. How this dataset is created is dependent on whether or not reusable or

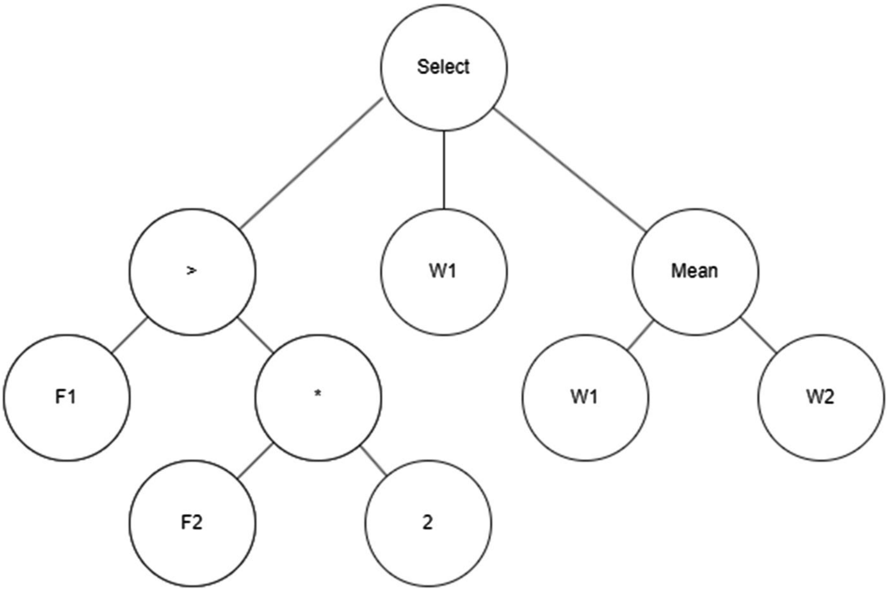

Fig. 1 Example crossover operator represented as a tree

disposable crossover operators are being evolved, more information on this is provided in Sect. 5.

Given this dataset, the fitness of a crossover operator in the GP population is calculated by performing crossover on each pair of NN weights in the dataset and using Eq. (1) to get a score for each produced offspring. Equation (1) which is at the core of the GP's fitness function effectively calculates the ratio of the parent weights' fitness to that of the produced offspring weight's fitness.

$$\frac{(p_1 + p_2)/2}{o} \tag{1}$$

where p_1 is the GA fitness of the first parent, p_2 is the GA fitness of the second parent and o is the GA fitness of the offspring produced by the crossover operator. Hence, a score larger than 1 indicates that the crossover operator was able to produce an offspring with a better GA fitness than that of the parent weights. The scores for all offspring produced when performing crossover on the NN weight pairs in the dataset are then averaged and returned as the GP fitness of the crossover operator. In contrast to the GA, since a larger fitness score is better the fitness in the GP is being maximized.

Algorithm 2 GP fitness function

```
crossover test dataset ← $\mu_0, ..., \mu_m$                ▷ Training set of NN weights
fitness sum *gets* 0
**for** each datapoint in crossover train dataset **do**
    parent1 weights ← datapoint[0]
    parent2 weights ← datapoint[1]
    offspring weights ← CROSSOVER(parent1 weights, parent2 weights)    ▷ Crossover represented by individual
    parent1 GA fitness ← GA_FITNESS(parent1 weights)
    parent2 GA fitness ← GA_FITNESS(parent2 weights)
    offspring GA fitness ← GA_FITNESS(offspring weights)
    fitness sum +=GP_FITNESS(parent1 GA fitness, parent2 GA fitness, offspring GA fitness)    ▷ Equation 1
**end for**
return (fitness sum)/size(crossover test dataset)
```

4.3 Tree generation

Trees are generated using the grow method which randomly selects from either the function set or terminal set with a fixed probability each time a node is generated. Hence, if a node from the function set is generated the process is repeated and the sub-tree continues growing until either all children are selected from the terminal set or the maximum depth is reached where the process is forced to only select from the terminal set. This results in trees within the population being diverse in shape and depth.

4.4 Mutation

The mutation operator randomly selects a node called the mutation point within the tree being mutated. The selected node and its sub-tree is then replaced with a new randomly generated sub-tree using the grow method as described above. The sub-tree being generated is required to output the same type as the sub-tree it is replacing, ensuring that the tree remains valid. This mutation operator is commonly used for GP [10] and offers a great degree of variability in the severity of mutation based on which mutation point is randomly selected. If the mutation point is close to the root of the tree the mutation will be drastic and change a large portion of the tree, alternatively if the mutation point is a leaf node or at a great depth, the mutation will be less severe. The maximum depth of the tree being generated using the grow method was set to the maximum depth allowed for trees minus the depth of the mutation point. This was done to keep the size of trees in the population under control.

4.5 Crossover

A simple one point crossover operator [10] is used where one node from each tree is selected as the crossover points. The two nodes at the crossover points including their sub-trees are swapped between the two parents to produce two new offspring trees. To ensure the sub-trees remain valid the two selected nodes are required to output the same type. Since the crossover operator used in the GP produce two offspring, half the number of crossover parents are selected in the GP in comparison to the GA (Algorithm 1 line 5).

It is worth noting that one point crossover can easily produce very large trees. In the extreme case the root node of one parent can be selected as the crossover point and a leaf node of the second parent as the other crossover point. This will produce one large offspring and one offspring consisting of a single node. The depth of trees in the population is controlled by modifying tournament selection to select the shallowest tree when there were multiple potential tournament winners with the same fitness. Hence, there is a preference in the selection operator towards smaller trees.

5 Evolving disposable versus reusable crossover operators

In this study, two different approaches were taken in order to produce disposable and reusable crossover operators. Firstly, it should be pointed out that a crossover operator is not either reusable or disposable but that there is rather a spectrum of reusability for crossover operators. For example, a crossover operator that can be applied to different NN architectures is more reusable than a crossover operator that can only be applied to a single NN architecture. However, when referring to disposable crossover operators in this paper, we refer to the extreme end of this spectrum where a crossover operator is designed to only be applied to a single pair of parent NN weights and is not expected to perform well on any other weights. The value of evolving such disposable crossover operators is that they can be highly customized and may give us insight into the expected upper boundary for how well crossover operators can be expected to perform when applied to NNs. The drawback of this approach is that it is expensive as the GP has to run for each pair of NNs weights, steps are taken to reduce the GP runtime in this case which are described later in this section.

Alternatively, when referring to reusable crossover operators we refer to operators that can be applied to different pairs of parent weights in a GA but that are evolved for a specific NN architecture and problem domain. Hence, a reusable crossover operator can be used throughout a GA each time crossover is performed, given that the NN architecture and problem domain remains consistent. We choose to focus on this level of reusability for the reusable version of our

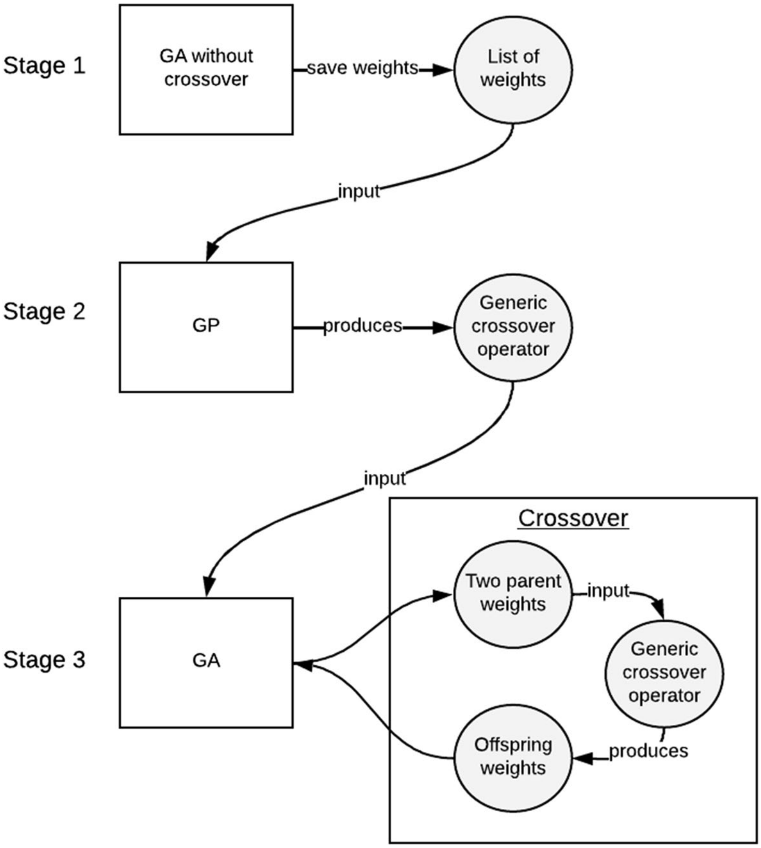

Fig. 2 Control flow of evolving reusable crossover operators

crossover operators since evolving highly reusable operators may not be feasible and this level of reusability provides a good starting point.

Figure 2 shows the approach taken to evolve reusable crossover operators which consists of 3 distinct stages. The first stage involves creating a diverse crossover train dataset to be used in the fitness function of the GP when evolving reusable crossover operators. This dataset is generated by running the GA with a mock crossover operator that instead of performing crossover, saves the pair of weights and returns the first weight matrix as the result of crossover. This means that the size of the dataset is the product of the GA population size, crossover rate and number of generations. For example, if the GA is run for 80 generations with a population size of 10 and crossover rate of 0.4, the generated dataset will contain 320 pairs of weights. The GP is then run in stage 2 using the dataset produced in stage 1 as the input dataset in its fitness function. The offspring with the best fitness in the last generation of the GP is then used as input for Stage 3. The third stage then runs the same GA as in Stage 1, but this time with crossover using the operator evolved in Stage 2.

The approach taken to evolve disposable crossover operators is depicted in Fig. 3. Using this approach the GP is effectively used as a crossover operator in the GA. This means that every time crossover is performed on a pair of parent weights, the pair of weights are given as input to the GP and the GP evolves a crossover operator specifically for that pair of weights. The fitness of an individual operator in the GP is measured by applying it to the pair of parent weights using Eq. (1). Once the GP has terminated, the best crossover operator in the last generation is applied to the input pair of weights to produce the offspring weights returned to the GA.

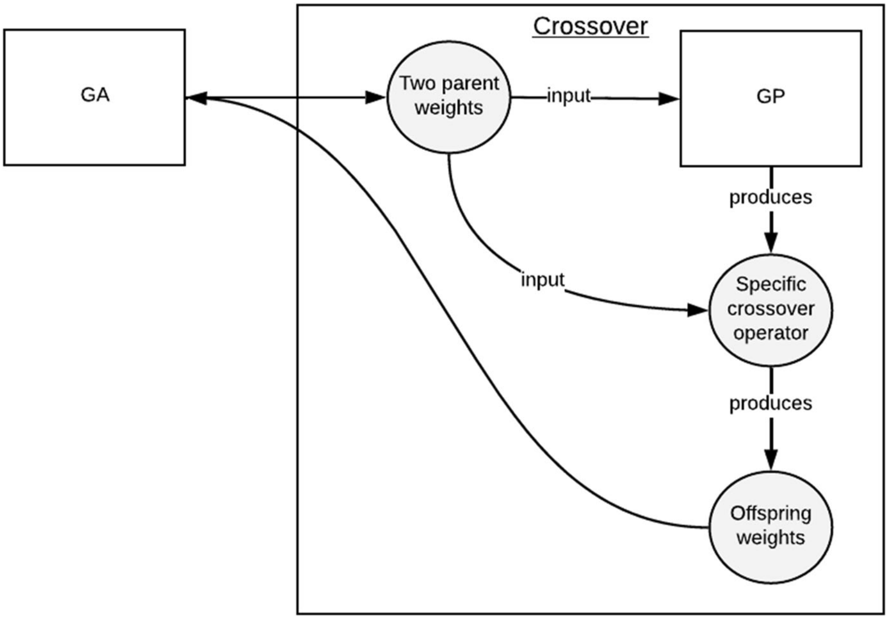

Fig. 3 Control flow of evolving disposable crossover operators

To reduce the runtime of the GP when evolving disposable crossover operators, the runtime of the fitness function had to be reduced. The most expensive operation in the GP is calculating the loss of the NN produced by crossover operators in the population during fitness evaluation. To reduce the runtime of this calculation we only estimate the loss of the offspring NN (i.e *loss*(*offspring*) in Eq. 1) by calculating the loss on a sample of the training set used in the GA instead of the full training set. The sample size is 10% of the training set and is randomly selected. Furthermore, the GP naturally converges much faster when used for evolving disposable crossover operators since the search space is considerable less complex, allowing fewer generations to be used to reduce the runtime of the GP.

6 Experimental set-up

This section outlines the experiments conducted in this study and describes the experimental setup used.

6.1 Baseline crossover

As a baseline crossover operator to compare the evolved crossover operators to a simple crossover operator that is commonly used in literature [1, 9, 24, 25] was used. This crossover operator simply takes the element-wise average of the two parent weight vectors to produce a single offspring weight vector. This operation is exactly the same as the Mean operator in the function set of our GP. We refer to this crossover operator as the mean crossover.

6.2 Statistical tests

In order to compare the performance of two GAs using different crossover operators, the GAs are run 30 times with the fitness (cross-entropy loss) of the best performing individual recorded at the end of each run to form a set of results μ. To test if the results obtained using one crossover operator is statistically significantly better than the results obtained when using another crossover operator, a left one-tailed Mann–Whitney U test is performed. The tests will be conducted using a confidence level of 95%, hence only tests resulting in a p-value less than 0.05 will indicate statistical significance.

The follow strategy is used when comparing the results obtained from two GAs. Given two sets of results μ_1 and μ_2 produced by GA 1 and GA 2 respectively, where $\mu_1 < \mu_2$. The null hypothesis will be that $\mu_1 >= \mu_2$ and the alternative that $\mu_1 < \mu_2$. If a p-value of 0.05 or less is obtained the null hypothesis is rejected in favour of the alternative hypothesis, indicating that GA 1 produced statistically significantly better results than GA 2. This strategy is used to conduct all statistical tests for all three experiments since each experiment is designed such that it compares only two GAs.

6.3 Datasets

To evaluate the performance of the proposed GP, the GP will be applied to evolve crossover operators for five different classification problems. Each chosen dataset represents a classification problem that can be solved using NNs. Hence, the GA will be applied to each dataset in order to evolve NN weights that can classify the examples in the dataset. To test the ability of the GP to evolve crossover operators for both standard fully connected feed forward NNs and convolutional neural networks (CNNs), three datasets containing standard numerical inputs and two datasets consisting of images were chosen. The datasets were chosen such that a combination of benchmark, real world problems and competition datasets are included. The datasets used in the experiments conducted in this study are outlined below.

1. CIFAR-10: A popular benchmark image classification dataset consisting of 60,000 images across 10 classes [26];
2. Thumbnail: A binary image classification dataset consisting of good and bad examples of video thumbnails, this dataset is a small real world dataset consisting of 2400 images from [27];
3. Higgs: A classification dataset consisting of 250,000 examples with 30 numerical features each and a single label indicating if a Higgs boson was detected as part of the event represented by the example [28];
4. Dry Bean: A classification dataset containing 13611 dry bean examples described by 17 real valued features across 7 classes [29];
5. Wine: A classification dataset containing only 178 examples described by 13 real valued features across 3 classes from the UCI Machine Learning repository [30].

Each dataset was randomly shuffled and split into training and validation datasets consisting of 80% and 20% of the full dataset respectively. The training dataset is used during mutation and evaluation in the fitness function, while the validation dataset is used to evaluate the best NNs produced by the GA. Hence, the results used in the statistical tests are as measured on the validation set. The input data for each dataset was preprocessed using mean normalization to scale and zero center the data. This is accomplished by subtracting the mean and dividing by the standard deviation for each feature.

6.4 NN architectures

CNNs were used for the image classification datasets (Cifar 10 and Thumbnail). Specifically, MobileNetV2 [19] was used since it is a small CNN architecture with few parameters which would prevent overfitting on the relatively small image

Table 1 Adam parameters

Parameter	Value
Learning rate	0.001
β_1	0.9
β_2	0.999
ϵ	$1e^{-7}$

Table 2 RMSprop parameters

Parameter	Value
Learning rate	0.001
ρ	0.9
Momentum	0
ϵ	$1e^{-7}$

classification datasets used in this study. Since the images in the datasets used are relatively small (32 × 32) in comparison to the resolution that MobileNetV2 was designed for (224 × 224), the stride sizes of the first 3 convolutional layers of the CNN were decreased from 2 to 1. A simple fully connected feed forward network consisting of 3 hidden layers with 128, 64 and 32 nodes was used for the other two datasets (Dry Bean and Wine). The fully connected feed forward neural networks used ReLU [31] and Sigmoid activation functions for the hidden and output layers respectively.

6.5 NN optimizer

For the GA mutation operator a gradient descent based optimizer is used. The exact optimizer used can easily be changed and can be seen as a parameter of the GA. By running some initial experiments it was found that using the Adam optimizer [32] resulted in the best results for the image classification datasets. The parameters used for Adam are shown in Table 1, where β_1, β_2 are the exponential decay of the first and second moment estimates and ϵ is the epsilon hat parameter used for numerical stability as defined in [32].

For the remaining two datasets (Dry Bean and Wine), RMSprop [33] produced the best results. The parameters used for RMSProp are shown in Table 2, where ρ and ϵ are the discounting factor for old gradients [33] and epsilon hat parameter used for numerical stability as defined in [32] respectively.

6.6 Parameters

The parameters used for the GA and GP in the experiments were determined empirically by starting with commonly used values in literature [21, 23, 34] that were reasonable given the runtime of the algorithms. In order to tune the GP parameters, the GA was first run once on the CIFAR-10 dataset to generate a dataset of NN weights that could be used to tune the GP parameters on. Given this dataset, the GP was then run once using the parameters from literature, with the result being recorded.

The fist GP parameter that was tuned was the population size, which started at a value of 10. The population size was gradually increased until no significant improvements in the performance of the GP was observed.

The next parameters tuned were the operators rates. Originally elitism was not included in the GP which means that the only rates that needed to be adjusted were the mutation and crossover rates. Since these two rates sum to 1, setting one rate also determines the other. Given the starting mutation rate, the direction of the search was determined by incrementing and decrementing the rate by 0.1 and comparing the GP performance using the two rates. The rate at which the GP performed best was then selected and either incrementally increased or decreased by 0.1, depending on if the larger or smaller rate was selected in the previous step. This process was continued until the GP performed worse than in the previous iteration. The rate at which it performed best during this search was then selected.

The next parameter tuned was the tournament size which was tuned similarly to the operator rates, using increments or decrements of 1 instead of 0.1. After this step it was found that a very small tournament size of 2 performed best, likely due to the relatively small population size. Given the small tournament size, it was decided to include elitism at a low rate of 0.1. This was done in order to retain the best individuals in the population since it is likely that the tournament selection may never consider them given the sample size of 2. This decision was validated by confirming that the GP performed better using elitism at a rate of 0.1 in comparison to using no elitism.

Lastly, the number of generations was determined by running the GP for 300 generations multiple times and observing at which point no more improvements were made. This was done both for the GP evolving reusable and disposable crossover operators separately.

The same parameter tuning process was then followed for the GA parameters, again using the CIFAR-10 dataset and keeping the GP parameters fixed at the values determined in the previous step. Since the fewest number of fitness evaluations per generation is performed by the GA using the baseline (mean) crossover operator, the number of generations used for the GA was determined by running the GA using the baseline crossover operator until no more improvements were made by the GA. This is done in an effort to make a fair comparison between GAs even if a different number of fitness evaluations are used since the GA using the fewest number of fitness evaluations would not produce better results if run for additional generations.

It should be noted that using automatic parameter tuning, or individually tuning the parameters for each dataset, may have further improved the results obtained, however, manual parameter tuning on a single dataset was decided to be sufficient. This is because the main goal of this study is not to obtain state of the art results, but rather to evaluate the performance of GP evolved crossover operators and compare how well GAs using various crossover operators perform relative to each other.

Table 3 GA parameters

Parameter	Value
Population size	10
Mutation rate	0.5
Crossover rate	0.4
Elitism rate	0.1
Tournament size	2

Table 4 Number of generations in GA per dataset

Dataset	Number of generations
Thumbnail	40
CIFAR-10	45
Higgs	50
Bean	75
Wine	40

Table 5 GP parameters

Parameter	Value
Population size	15
Mutation rate	0.3
Crossover rate	0.6
Elitism rate	0.1
Tournament size	2
Number of generations (Reusable)	150
Number of generations (Disposable)	10

Hence, suitable parameters should be sufficient for the experiments in the study, and near optimal parameters are not needed.

The GA parameters are shown in Table 3 with the number of generations used for each dataset shown in Table 4. The GP parameters are shown in Table 5. Some notable parameters that may seem unconventional are the small tournament sizes and small population sizes. It was found that small population sizes were sufficient and further increasing the population sizes did not lead to improved results, but greatly increased runtime of the algorithms. As a result of the small population size, the tournament size also had to be reduced to prevent the selection pressure from being too high, which was found to lead to early convergence.

The mutation rate in the GA may also seem unusually high, but due to the relatively stable nature of gradient descent based mutation in comparison to other more stochastic mutation operators commonly used in GAs, a higher mutation rate can be used and was observed to increase the rate of improvement in the GA.

6.7 Experiments

The experiments in this study aim to evaluate the performance of GP evolved crossover operators and to highlight the contributions of this study. The following three experiments are conducted:

6.7.1 Experiment 1: evolving reusable versus disposable crossover operators

The first experiment in this study aims to compare the performance of reusable and disposable crossover operators evolved by the GP. This is done by comparing the performance of the GA when using disposable evolved crossover to the GA when using reusable evolved crossover. This will give us insight into whether evolving disposable crossover operators is worth the increase in computational cost and results in a significant performance gain relative to evolving reusable crossover operators. The best performing approach will be used for the next experiments where GP crossover is compared to other approaches.

6.7.2 Experiment 2: GA without crossover versus GA with GP evolved crossover

Experiment 2 will compare the performance of the GA when using GP crossover with the performance of the GA when using no crossover. We perform this experiment since the crossover operator is often omitted in GAs that evolve NN weights because it is seen as destructive or redundant. Hence, the goal of this experiment is to determine if including GP evolved crossover in the GA is beneficial and significantly improves the results obtained.

Table 6 Initial experiment 1 results—Comparing GA using reusable and disposable evolved crossover operators

Dataset	Crossover	Mean	Best	SD
CIFAR-10	Reusable	**0.466**	**0.460**	0.006
	Disposable	0.485	0.477	0.006
Bean	Reusable	**0.169**	**0.157**	0.011
	Disposable	0.186	0.179	0.011

Best results are indicated in bold

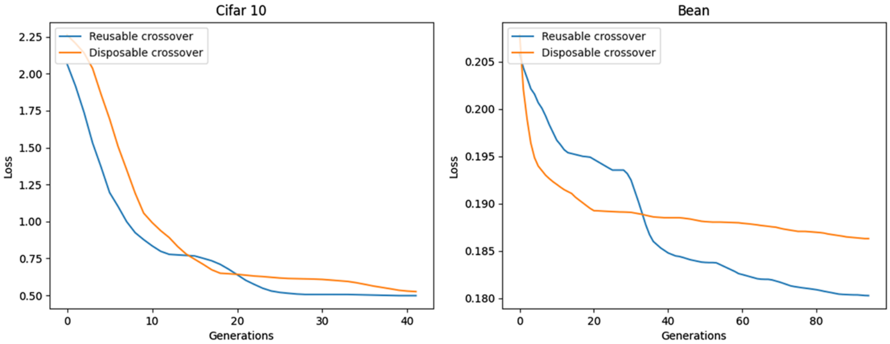

Fig. 4 Initial experiment 1 results—Comparing GA using reusable and disposable evolved crossover operators

6.7.3 Experiment 3: GA with mean crossover versus GA with GP evolved crossover

The final experiment in this study aims to compare the performance of GP evolved crossover operators to a commonly used NN weight crossover operator in literature (referred to as the mean crossover operator in this study). The goal of this experiment is to assess if the approach proposed in this study can contribute to the field of evolving NN weights by enabling the use of more effective crossover operators in comparison to what is typically used.

7 Results

This section presents and discusses the results obtained from the experiments conducted as part of this study. Each experiment relates to a specific contribution of this paper and the results are presented per experiment below.

7.1 Experiments 1

Initial results obtained on the CIFAR-10 and Bean datasets for experiment 1 are shown in Table 6 and Fig. 4. The results obtained were unexpected as it showed that reusable crossover operators outperformed disposable crossover operators. This was surprising since the disposable crossover operators are optimized specifically for a single pair of NN weights and would be expected to at worst match the performance

Table 7 Variable crossover rate

Percentage of generations completed	Crossover rate
< 33%	0.1
33–66%	0.2
> 66%	0.4

Table 8 Experiment 1 results—comparing GA using reusable and disposable evolved crossover operators with a dynamic crossover rate

Dataset	Crossover	Mean	Best	SD	*P*-value
CIFAR-10	Reusable	0.529	0.470	0.039	
	Disposable	**0.464**	**0.387**	0.049	0.0001*
Thumbnail	Reusable	0.228	0.206	0.011	
	Disposable	**0.221**	**0.204**	0.007	0.024*
Bean	Reusable	0.185	**0.159**	0.011	
	Disposable	**0.181**	0.162	0.013	0.183
Wine	Reusable	0.016	$1.325e^{-8}$	0.037	
	Disposable	**0**	**0**	0	$4.003e^{-9}$*
Higgs	Reusable	0.371	**0.358**	0.004	
	Disposable	**0.369**	0.361	0.003	0.049*

Best results are indicated in bold

of reusable crossover operators that are evolved for a set of weights. One possible reason for this is that evolving disposable crossover operators is highly exploitative, causing the GA to converge on a local optima early on. The gradient based mutation operator that is used is also not well suited for moving the population out of such optima.

To test this hypothesis, experiment 1 was reran on the CIFAR-10 and Bean datasets using a variable crossover rate that incrementally increased from 0.1 to the set crossover rate over the course of the GA. The rate of change for the variable crossover rate that was found to work best was starting with a crossover rate of 0.1, increasing the rate to 0.2 when 33% of the total number of generations has completed and then finally increasing the rate to 0.4 (the previous fixed rate) once 66% of the total generations have completed. The change in crossover rates used is presented in Table 7. This should enable the GA to explore the solution space first, after which the increase in the crossover rate starts promoting convergence. Using a variable crossover rate in the GA when using disposable crossover operators greatly improved the results obtained in comparison to using a fixed crossover rate. Using a variable crossover rate in the GA when using disposable crossover operators did not seem to make a significant difference. In order to keep the number of times crossover was performed consistent between GAs using reusable and disposable crossover operators, the variable crossover rate was used for both GAs.

The results for experiment 1 after running the experiment for all datasets using the adjusted parameters as described above are shown in Table 8 and Fig. 5. These results show that GAs using disposable evolved crossover operators were consistently able to outperform the reusable evolved crossover operators with the GA using disposable crossover operators obtaining significantly better results in 4 out of 5 datasets. The p-values obtained from the one tailed Mann–Whitney U test comparing the fitness from the GA using disposable and reusable evolved crossover operators were 0.0001, 0.024, $4.003e^{-9}$ and 0.049 on the CIFAR-10, Thumbnail, Wine and Higgs datasets, respectively. The *p*-value on the Bean dataset was 0.183, which is outside the range to accept statistical significance with a 95% confidence. This

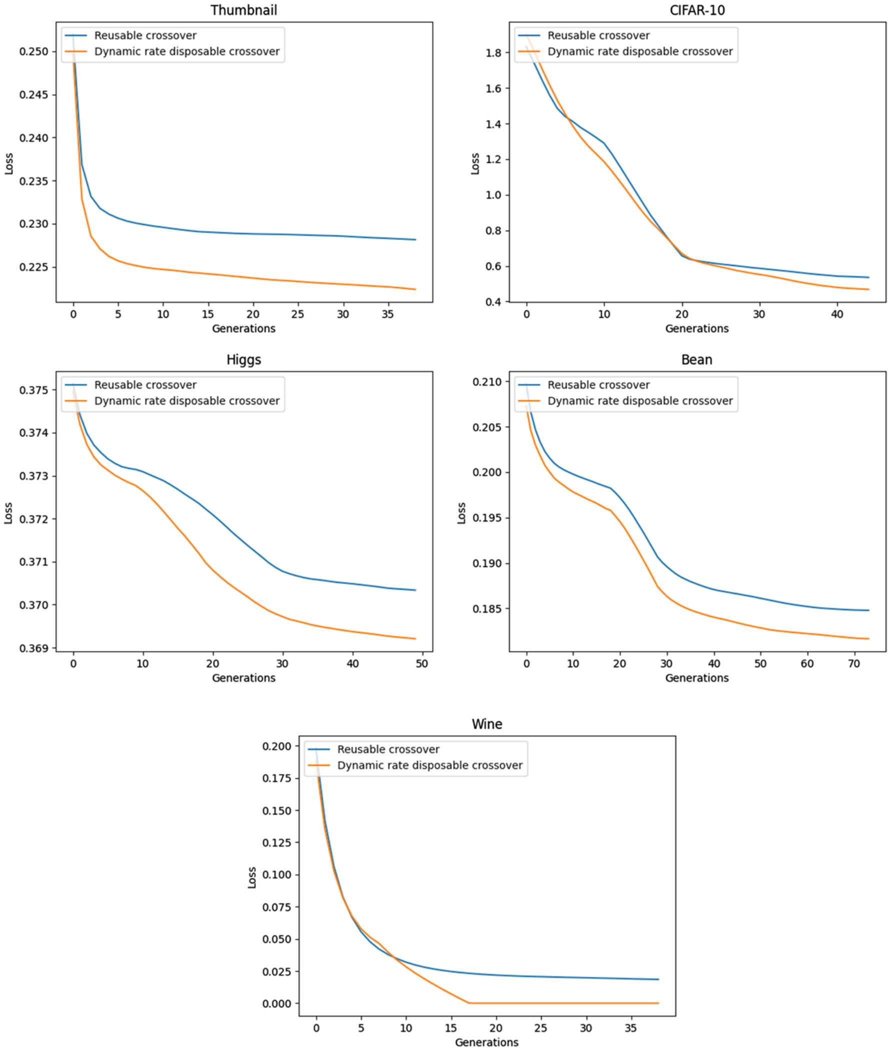

Fig. 5 Experiment 1 results—comparing GA using reusable and disposable evolved crossover operators with a dynamic crossover rate

suggests that the theory of disposable crossover operators causing early convergence may be correct, and that the early convergence can be avoided by limiting the rate of crossover in early generations and increasing it over time.

As part of the experiment, crossover operator created by the GP approach were studied. It was found that the disposable crossover operators that were evolved were highly diverse and complex with no clear patterns observed. An example of a relatively simple disposable crossover operator that was evolved for a pair of NN weights on the Wine dataset is shown in Fig. 6. Another interesting

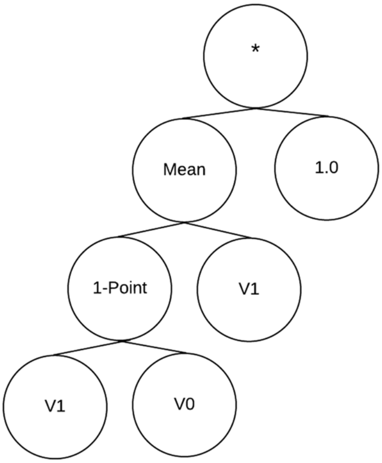

Fig. 6 Example disposable crossover operator evolved by the GP approach for the Wine dataset

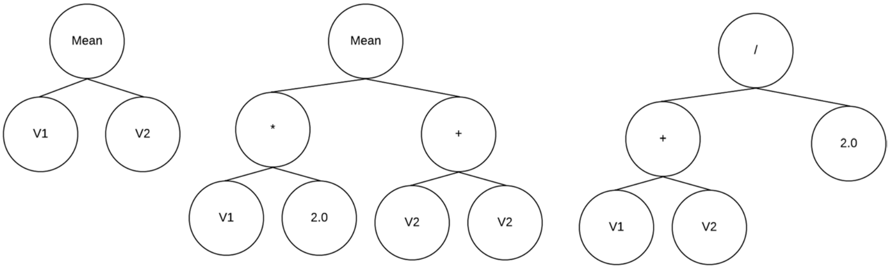

Fig. 7 Examples of reusable crossover operators evolved by the GP approach that are functionally equivalent to the mean crossover operator

Table 9 Experiment 2 results—comparing GA with no crossover and disposable evolved crossover operators

Dataset	Crossover	Mean	Best	SD	*P*-value
CIFAR-10	No	0.686	0.625	0.043	
	Disposable	**0.464**	**0.387**	0.049	$5.235e^{-7}$*
Thumbnail	No	0.225	0.216	0.006	
	Disposable	**0.221**	**0.204**	0.007	0.0414*
Bean	No	0.191	0.166	0.012	
	Disposable	**0.181**	**0.162**	0.013	0.0171*
Wine	No	0.039	$2.737e^{-7}$	0.050	
	Disposable	**0**	**0**	0	$4.0032e^{-9}$*
Higgs	No	0.372	0.364	0.003	
	Disposable	**0.369**	**0.361**	0.003	0.0060*

Best results are indicated in bold

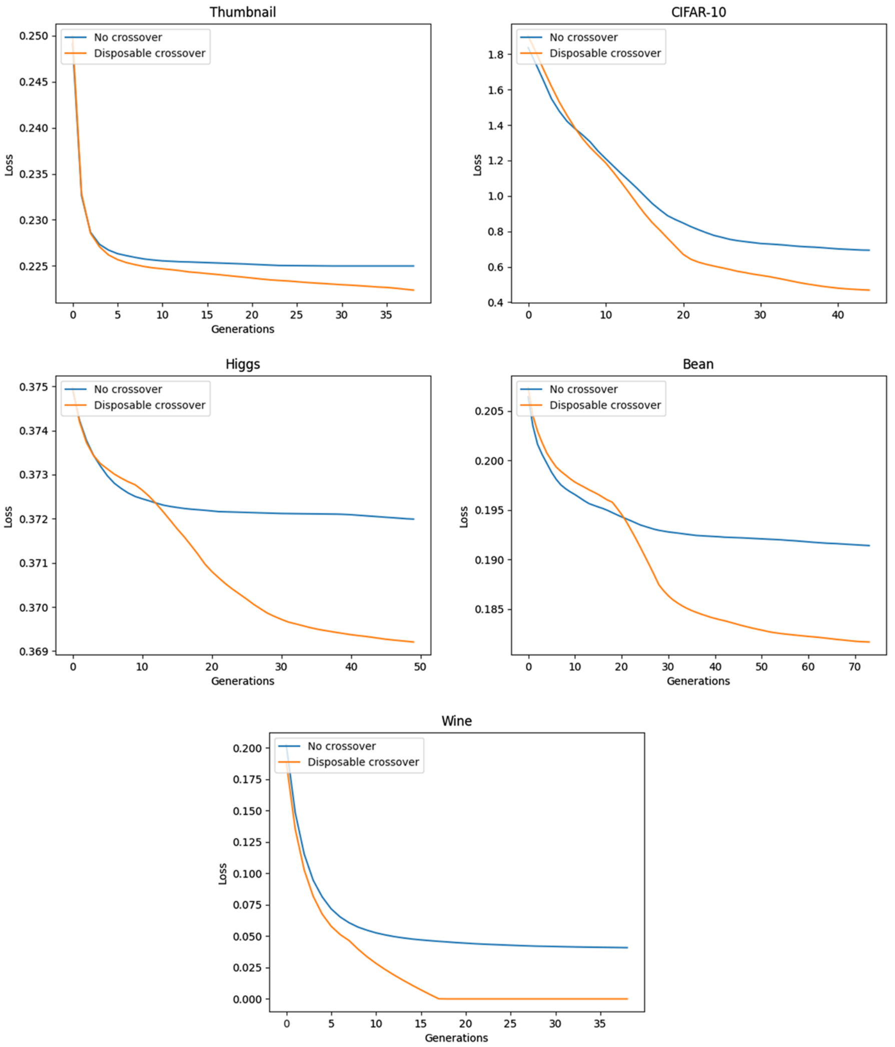

Fig. 8 Experiment 2 results—comparing GA with no crossover and disposable evolved crossover operators

observation made was that the reusable crossover operators evolved by GP approach were consistently functionally equivalent to the mean crossover operator. In other words, when evolving reusable crossover operators the GP always converged to a crossover operator that simply takes the element wise average of the two parent NN weight vectors (referred to as the mean crossover operator in this thesis). This found to be true for 100% of runs were the GP was used to evolve reusable crossover operators and was confirmed by manually inspecting the best operator evolved at the end of each run for each dataset. Examples of evolved reusable crossover operators are shown in Fig. 7.

Given that using disposable evolved crossover operators with a dynamic crossover rate outperformed reusable evolved crossover operators with both a fixed and dynamic rate, it was decided to use disposable crossover operators with a dynamic crossover rate for the experiments going forward.

7.2 Experiment 2

The objective of experiment 2 was to compare the performance of GAs using no crossover operator to GAs that are using GP evolved crossover operators. This is done in order to determine if including crossover operators evolved by the GP approach in the GA leads to improved results.

The results obtained in experiment 2 are shown in Table 9 and Fig. 8. These results show that the GA using GP evolved crossover operators consistently outperformed the GA using no crossover operator with statistical significance on all datasets. The p-values obtained from the one tailed Mann–Whitney U test comparing the fitness from the GA when using GP evolved crossover to the fitness from the GA using no crossover were $5.235e^{-7}$, 0.0414, 0.0171, $4.0032e^{-9}$ and 0.0060 on the CIFAR-10, Thumbnail, Bean, Wine and Higgs datasets respectively.

It can be observed that the GA using no crossover performs similarly, or in some cases even outperforms, the GA using evolved crossover in early generations. However, as the crossover rate increases the GA using evolved crossover operators is able to produce better results. The obtained results are encouraging as they suggest that crossover operators when applied to NN weights are not inherently destructive, but can be beneficial to the GA when designed and used correctly.

7.3 Experiment 3

The final contribution of this paper is to show that GP evolved crossover does indeed outperform the mean crossover operator that is commonly applied to NN weights in literature. While conducting experiment 1 it was noticed that the reusable crossover operators evolved by GP were consistently functionally equivalent to the mean crossover operator. In other words, when evolving reusable crossover operators the GP always converged to the mean crossover operator. This means that the results for experiment 3 would be the same as the results obtained in experiment 1 that compares reusable and disposable crossover operators. This shows that the GP evolved crossover operators only outperforms the standard mean crossover operator when the GP is used to evolve disposable crossover operators using a dynamic crossover rate.

The fact that the GP consistently converged on the mean crossover operator when used to evolve crossover operator may suggest that the mean crossover operator is a good reusable crossover operator. However, in order to improve efficiency of crossover operators for NN weights, disposable crossover operators may need to designed and used as done in this study.

Table 10 Additional experiment results—Comparing GA using the mean crossover operator to the GA using no crossover operator

Dataset	Crossover	Mean	Best	SD	*P*-value
CIFAR-10	No	0.686	0.625	0.043	
	Mean	**0.529**	**0.470**	0.039	$5.235e^{-7}$*
Thumbnail	No	**0.225**	0.216	0.006	0.2430
	Mean	0.228	**0.206**	0.011	
Bean	No	0.191	0.166	0.012	
	Mean	**0.185**	**0.159**	0.011	0.0398*
Wine	No	0.039	$2.737e^{-7}$	0.050	
	Mean	**0.016**	$\mathbf{1.325e}^{-8}$	0.037	0.0568
Higgs	No	0.372	0.364	0.003	
	Mean	**0.371**	**0.358**	0.004	0.0975

Best results are indicated in bold

To investigate the bad reputation that crossover operators have when applied to NN weights, the performance of a GA using no crossover operator was compared to a GA using the mean crossover operator. The results obtained are shown in Table 10 and Fig. 9. The results show that the GA using no crossover operator performed better in the Thumbnail dataset, while the GA using the mean crossover operator performed better in all other datasets. Furthermore, the difference in results obtained was only statistically significant for the CIFAR-10 and Bean datasets. Hence, for three of the five datasets tested there was no significant difference in results obtained when using the mean crossover operator in comparison to using no crossover operator.

In contrast, the previous experiments showed that the GA using disposable evolved crossover operators outperformed the GA using no crossover operator on all datasets. This suggests that in order for crossover operators to consistently be effective when applied to NN weights, they may need to be specifically customized for each pair of NN weights as done in this study when evolving disposable crossover operators.

8 Conclusion

This study aimed to investigate the use of crossover operators in GAs that optimize NN weights. Using crossover operators to combine two NN weight matrices is commonly seen as being destructive and not beneficial to include in a GA. The authors believe that the issue with NN crossover operators in the past is not that applying crossover to NN weights is infeasible but rather that the crossover operators are not designed well enough.

For this reason, GP was used in this study to search the program space of crossover operators in order to automatically design crossover operators that can be used in a GA to evolve NN weights. Furthermore, the GP was applied to evolve both disposable crossover operators that are evolved for a single pair of NN weights and

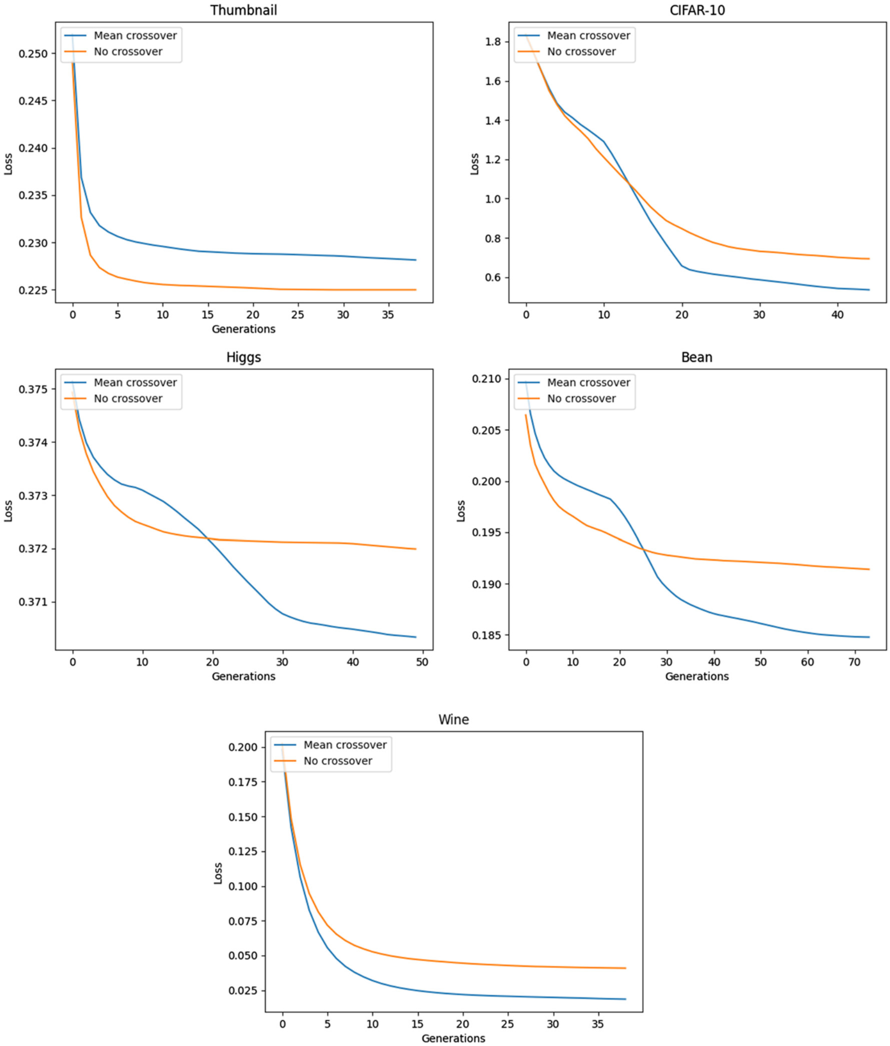

Fig. 9 Additional experiment results—comparing GA using the mean crossover operator to the GA using no crossover operator

reusable crossover operators that are evolved for a specific problem domain and NN architecture.

Experiments showed that using GP to evolve disposable crossover operators for a GA led to better results in comparison to using reusable evolved crossover operators. However, this was only true when using a dynamic crossover rate that slowly increases over time to prevent early convergence due to the exploitative nature of using evolved disposable crossover operators. Further experiments showed that the use of evolved disposable crossover operators in a GA significantly improved the results obtained by the GA in comparison to using no crossover operator.

It was also observed that when using GP to evolve reusable crossover operators, the GP consistently converged on a commonly used crossover operator in literature that simply averages the two parent NN weights, referred to as the mean crossover. Using the mean crossover operator in a GA only led to an improvement in results on some datasets while leading to worse results in other. This may indicate that in order for crossover to successfully be used in GAs that evolve NN weights, the crossover operators may need to be specifically designed for each pair of weights as done with the disposable crossover operators used in this study. In cases where the producing performant NNs is the priority, using GP to evolve disposable crossover operators should be worth the increase in computational cost as it leads to a significant improvement in the quality of NN weights produced by the GA.

Future work of this study includes investigating the use of other evolutionary techniques to design crossover operators as well as extending the function set of the GP used in this study to evolve more complex crossover operators.

Acknowledgements The authors would like to thank the Multichoice Group for funding this research as part of the Multichoice Machine Learning Chair.

Author Contributions KP and NP conceived of the presented idea. KP carried out the experiments and wrote the main manuscript text. NP provided KP with guidance as his research supervisor and helped to refine the presented idea. Both authors reviewed the manuscript.

Funding Open access funding provided by University of Pretoria. This work was funded as part of the Multichoice Research Chair in Machine Learning at the University of Pretoria, South Africa. This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number 138150). Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declarations

Conflict of interest Other than the above mentioned funding, the authors have no other competing interests to declare that are relevant to the content of this article.

References

1. X. Yao, Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)
2. P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. **5**(1), 54–65 (1994). https://doi.org/10.1109/72.265960
3. S. Haflidason, R. Neville, On the significance of the permutation problem in neuroevolution. In: Proceedings of the 11th annual conference on genetic and evolutionary computation. GECCO '09. (Association for Computing Machinery, New York, NY, 2009) pp. 787–794. https://doi.org/10.1145/1569901.1570010

4. P.J.B. Hancock, Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In: [Proceedings] COGANN-92: International workshop on combinations of genetic algorithms and neural networks, (1992) pp. 108–122
5. R. Zhou, C. Muise, T. Hu, Permutation-invariant representation of neural networks with neuron embeddings, in *Genetic programming*. ed. by E. Medvet, G. Pappa, B. Xue (Springer, Cham, 2022), pp.294–308
6. X. Yao, Y. Liu, Towards designing artificial neural networks by evolution. Appl. Math. Comput. **91**(1), 83–90 (1998). https://doi.org/10.1016/S0096-3003(97)10005-4
7. J.H. Holland, Genetic algorithms. Sci. Am. **267**(1), 66–73 (1992)
8. T. Froese, E. Spier, T. Froese, E. Spier, Convergence and crossover: the permutation problem revisited (2008)
9. T. Uriot, D. Izzo, Safe crossover of neural networks through neuron alignment (2020)
10. J.R. Koza, Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2), 87–112 (1994). https://doi.org/10.1007/BF00175355
11. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in Proceedings of the AAAI conference on artificial intelligence, vol. 33, pp 4780–4789 (2019). https://doi.org/10.1609/aaai.v33i01.33014780
12. E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, in Proceedings of the 34th international conference on machine learning, Vol. 70. (ICML'17, 2017), pp. 2902–2911
13. P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. **5**(1), 54–65 (1994)
14. F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning (2017). arXiv:1712.06567
15. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
16. T. McDonnell, S. Andoni, E. Bonab, S. Cheng, J.-H. Choi, J. Goode, K. Moore, G. Sellers, J. Schrum, Divide and conquer: neuroevolution for multiclass classification, in Proceedings of the genetic and evolutionary computation conference (2018)
17. K.O. Stanley, D.B. D'Ambrosio, J. Gauci, A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009). https://doi.org/10.1162/artl.2009.15.2.15202
18. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2015), arXiv:1409.1556
19. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in 2018 IEEE/CVF conference on computer vision and pattern recognition (CVPR), (IEEE Computer Society, Los Alamitos, CA, 2018), pp. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474 . https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00474
20. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in 2016 IEEE conference on computer vision and pattern recognition (CVPR) (2016), pp. 770–778 . https://doi.org/10.1109/CVPR.2016.90
21. T. Gangwani, J. Peng, Genetic policy optimization, in International conference on learning representations (2018). https://openreview.net/forum?id=ByOnmlWC-
22. X. Cui, W. Zhang, Z. Tüske, M. Picheny, Evolutionary stochastic gradient descent for optimization of deep neural networks, in Proceedings of the 32nd international conference on neural information processing systems. (NIPS'18, Curran Associates Inc., Red Hook, NY, 2018), pp. 6051–6061
23. M. Jaderberg, V. Dalibard, S. Osindero, W. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, K. Kavukcuoglu, Population based training of neural networks. arXiv:1711.09846 (2017)
24. N. Hansen, The cma evolution strategy: a tutorial. arXiv:1604.00772 (2016)
25. K. Lee, B.-U. Lee, U. Shin, I.S. Kweon, An efficient asynchronous method for integrating evolutionary and gradient-based policy search, in Proceedings of the 34th international conference on neural information processing systems. NIPS'20. (Curran Associates Inc., Red Hook, 2020)
26. A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario (2009)
27. K. Pretorius, N. Pillay, A comparative study of classifiers for thumbnail selection. In: 2020 international joint conference on neural networks (IJCNN), (2020), pp. 1–7 . https://doi.org/10.1109/IJCNN48605.2020.9206951

28. C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, D. Rousseau, The Higgs boson machine learning challenge. In: G. Cowan, C. Germain, I. Guyon, B. Kégl, D. Rousseau, (eds.) Proceedings of the NIPS 2014 workshop on high-energy physics and machine learning, in Proceedings of machine learning research, vol. 42. (PMLR, Montreal, Canada, 2015), pp. 19–55. https://proceedings.mlr.press/v42/cowa14.html
29. M. Koklu, I.A. Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques. Comput. Electron. Agric. **174**, 105507 (2020). https://doi.org/10.1016/j.compag.2020.105507
30. D. Dua, C. Graff, UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
31. V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on international conference on machine learning. ICML'10. (Omnipress, Madison, WI, 2010, pp. 807–814
32. D.P. Kingma, J. Ba, Adam: a method for stochastic optimization. In: Y. Bengio, Y. LeCun, (eds.) 3rd International conference on learning representations, (ICLR 2015, San Diego, CA, USA, May 7-9, 2015, conference track proceedings, 2015). arXiv:1412.6980
33. G. Hinton, N. Srivastava, K. Swersky, rmsprop: divide the gradient by a running average of its recent magnitude. Accessed: 10 April 2023. https://www.cs.toronto.edu/
34. S. Khadka, S. Majumdar, T. Nassar, Z. Dwiel, E. Tumer, S. Miret, Y. Liu, K. Tumer, Collaborative evolutionary reinforcement learning. In: International conference on machine learning. (PMLR, 2019), pp. 3341–3350

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.