GENEVO: Genetic Evolutionary Optimisation

Cleveland James Crossling
Academy of Computer Science and Software Engineering
University of Johannesburg
Johannesburg, South Africa
cleveland1.crossling@gmail.com

***Abstract*—This paper introduces GENEVO (Genetic Evolutionary Optimisation), a neuroevolutionary framework that leverages genetic algorithms as an alternative to traditional backpropagation for training neural networks. Focusing on classification tasks using a variant of the MNIST dataset, GENEVO represents each individual in the population as a fully connected neural network with a fixed topology. The genetic algorithm optimizes network weights through crossover and mutation operations, with the fitness evaluated using binary categorical cross-entropy. Extensive experimentation examines the impact of various hyperparameters, including population size, mutation rate, tournament size, and batch size on model performance and generalisation. Results demonstrate that appropriately tuned neuroevolutionary models can achieve competitive validation accuracies of up to 89.16%, while maintaining computational efficiency with smaller batch sizes. Additionally, GENEVO is applied to variational autoencoders (VAEs) to assess generative capabilities, though reconstruction quality remains limited. The findings indicate that neuroevolution is a viable and competitive alternative to conventional neural network training methods.**

***Index Terms*—Neuroevolution, Genetic Algorithms, Neural Networks, Evolutionary Variational Autoencoder, Evolutionary Algorithms, Fitness Evaluation**

I. Introduction

Evolutionary algorithms (EA) resulted from modelling observed biological processes such as natural selection, genetics, and mutation [1]. Algorithmic expressions of these processes were the primary approach to machine learning tasks before more modern approaches, such as deep neural networks (DNN), became popular. Most notably, genetic algorithms were commonly used, as they could be structured to most optimisation problems [2], [3]. As a result of the rise in popularity of DNNs, applications of evolutionary algorithms to neuron based problems have emerged. This field is known as neuroevolution [4].

Often, neuroevolution has the same structure as neural networks, which are constructed from matrices of numbers [5]. Within a typical DNN, weight updates would be performed through a backpropagation algorithm [6] based on the gradient of a loss function [7]. The loss function computes the gradient based on model inference predictions which are compared to the expected output. Neuroevolution utilises a different approach for weight updates, which are, instead, computed by an evolutionary algorithm [8]. This evolutionary algorithm can vary depending on the goal of the model; some examples of this include: Differential evolution [9], which approximates a population of weights that are graded based on the model's output quality, and genetic reproduction [10], which models a population of networks that reproduce to hone weights.

This paper introduces GENEVO (Genetic Evolutionary Optimisation) a specific neuroevolution approach, which incorporates a genetic algorithmic approach to weight updates. Each individual in the population is modelled to a fixed size topology and its weights are treated as its genetic sequence. The population is evaluated on individual batches of training data in order to compute an individual's fitness score. During reproduction, weights are exchanged in crossover and more accurate models can be computed.

II. Literature Review

A. Deep Neuroevolution

The paper [11] explores neuroevolution as a gradient computing model. It introduces a novel deep genetic algorithm (GA) model that performs competitively compared to other deep neural network based approaches. Using the deep GA model, they are able to successfully train approximately 4 million parameters in a reasonable training time. Their model takes a similar approach to the proposed model GENEVO, however, each individual in their model is a parameter of the neural network, while GENEVO sets each individual as its own independent set of matrices. They also do not do tournament selection, and instead, utilise a truncation selection operation for reproduction. A population storage mechanism is also used so that individuals are not saved as objects, but instead, are converted to vectors which can be collected as they are needed.

B. Large-Scale Evolution of Image Classifiers

Other papers also explore using neuroevolution to perform tasks that would typically be done by neural networks. This paper by Real et al. [12] explores using neuroevolution for computer vision, specifically image classification. Their approach also uses full architectures as individuals, however, selection is done with a kill tournament. Reproduction is also done via mutation, where parts of the parent are included in the offspring parameters. Offspring are trained on what is already known by the parent and then are evaluated using validation data before being placed back into the population. Using this methodology, they were able to attain 94.6% (95.6% for ensemble) on the CIFAR-10 dataset and 77.0% on the CIFAR-100 dataset. These results indicate that neuroevolution is a viable competitor to traditional neural networks.

III. Implementation

GENEVO performs neuroevolution through a genetic evolution algorithm. Each individual in the population is a fully connected neural network based on a predefined structure that is given to the model as a parameter during population initialisation. An individual consists of several layers of weights and biases, as well as a reproduction function.

Reproduction may only occur between two layers which have the same shape weights and bias matrices. Genetic crossover is done based on a randomized mask; this is done to ensure genetic material is transferred evenly, and to prevent alleles from becoming prematurely common within a population. Mutation is applied to weights and biases using the following algorithm:

$$weights_t \leftarrow weights_{t-1} + \mathcal{N}(0, 0.1^2)$$

$$bias_t \leftarrow bias_{t-1} + \mathcal{N}(0, 0.1^2)$$

Where $\mathcal{N}(0, 0.1^2)$ represents a Gaussian random variable with the mean 0 and the standard deviation 0.1.

Individuals within the model population are initialised by combining several of these layers together. It is specifically ensured that each layer output is equal to the following layer input. This is done for simple computation during the forward pass of an individual. Each individual completes its forward pass by passing an input of predefined size (input size) through the weight and bias matrices of the model with a dot product operation. This is consistent with forward propagation in a traditional multi-layer, fully connected perceptron (fully connected neural network). When the final layer is reached, the individual provides an output of a predefined size (output size).

The activation of each layer, which provides non-linearity to the output of layers, is done with a sigmoid operation. The function sigmoid is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This converts the output to a value within the range 0 - 1.

Traditionally, gradients would need to be computed for backpropagation, however, this function is replaced by reproduction, and therefore, updating weights with gradients is not necessary.

A. Training

Before beginning the generation loop, the model parameters must be defined and its population must be initialised. Each individual's structure is constructed from a predefined structure. This is created from combining three parameters provided to the constructor: the size of a single training data example (input size), each output size of the consecutive layers, and finally, the output size. Since the model is being evaluated on a classification task, the output size of the model is the same length as the quantity of classes being classified. The population is created based on the structure and predefined population size.

At the beginning of each generation, for each individual, a random training data example is selected from the training data and is repeated until the batch is full (this is determined by batch size). Each example in the batch is passed to the individual, which then produces a probability map. Fitness is determined using binary categorical cross entropy.

$$F_i = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Where:

- N is the number of samples.
- y_i is the true label for sample i.
- $\hat{y}_i$ is the predicted probability for sample i.
- F_i is the fitness of the model for that sample i.

A fitness value is computed for each example in the batch. The collected fitness values are then added together and assigned as the overall fitness for the individual. Typically, binary categorical cross entropy is used as a loss function. For this reason, lower fitness indicates a better individual.

After evaluation, a set number of individuals are selected from the population. They are sorted into ascending order according to their fitness and the top two individuals are extracted for reproduction.

The offspring is then added to a separate population group. This is repeated until the offspring pool contains the same amount of individuals as the population size. This methodology of reproduction is known as tournament selection. Finally, both lists are concatenated and shuffled together; half the individuals in this new list are collected and will make up the population for the next generation.

The final step is to evaluate the models on unseen test data. Each individual in the population is given the entire test set for evaluation. Typically, this step slows the training of the model, however, it is unnecessary for every generation. Within the scope of this project, a smaller validation set is used and the model performs an evaluation of individuals for every generation. This is done for comprehensive and continuous metric collection.

As an optional step, mutation annealing is added to assist the model in avoiding early convergence. The annealing mechanism monitors validation accuracy change and adjusts the value according to a scaling factor. The scaling factor is predefined and is added depending on the sign of the validation accuracy change. To prevent unnecessary scaling, a mutation threshold is defined that determines how much change must happen before scaling can occur.

The hyperparameters for the classification model are as follows:

- Input and output size - Defined by the size of the training data and labels.
- Structure - Describes the size of the hidden layers.
- Batch size - The size of the training batch used to evaluate fitness of individuals.

- Tournament size - The number of individuals evaluated for selection and reproduction.
- Population size - Indicates the amount of individuals that will be in the population pool.
- Mutation rate - Percentage describing how often mutations take place in reproduction.
- Mutation scaling - Values that indicate how much to change mutation if validation accuracy increases or decreases.
- Mutation threshold - A gating value that prevents changing the mutation rate unless the validation accuracy change is greater or less than this value.
- Epochs - The number of generations before stopping.

The GENEVO classifier is trained on two versions of the MNIST dataset [13]. The first version is an 8 x 8 pooled version of the original MNIST dataset, and the second is the original dataset imported from the Tensorflow library.

B. Variational Autoencoder

As a way to explore the generative capabilities of GENEVO, a variation specialised to handle individuals with the variational autoencoder (VAE) structure has been developed.

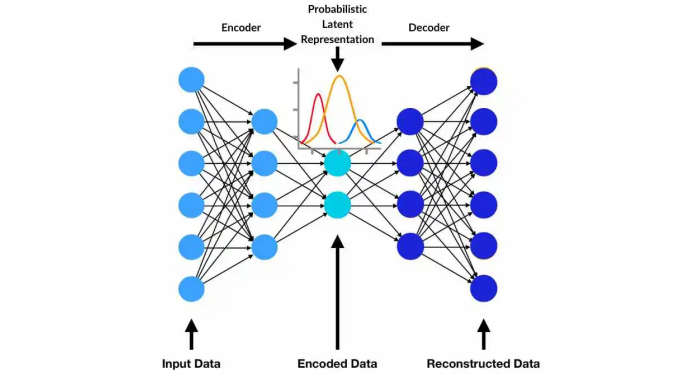

Fig. 1. The generic structure of an autoencoder [14].

As can be seen in Figure 1, the input size is equal to the output size of the model for this use case. GENEVO mimics a traditional VAE in the following way: Input is passed to an individual, similarly to the methodology in the classifier structure. The propagated value has its dimensionality reduced till the model reaches reparameterisation. Reparameterisation attempts to change the distribution of the reduced values and represent it in a way that is easier for sampling, in this case a normal distribution.

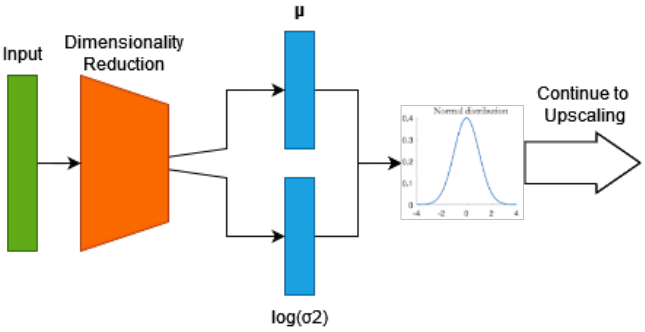

Fig. 2. Reparameterisation after dimensionality reduction to change latent space distribution.

To do reparametrisation this way within the context of neuroevolution, two new layers are introduced: log variance and mu. These layers define how the low dimension data will be represented on a normal distribution. This operation is done in the following way:

$$\sigma = e^{(0.5 \times \log(\sigma^2))}$$

$$\epsilon = \mathcal{N}(0, 1)$$

$$z = \mu + \epsilon \cdot \sigma$$

Where:

- $\log(\sigma^2)$ - Log variance which is defined by the log variance layer.
- μ - Mu which is defined by the mu layer.
- ϵ - A random variable sampled from a standard normal distribution.
- z - Normal distribution representation of the values.

After the values have been redistributed, they are passed to an upscaling set of layers. The values are increased in dimensionality until they are the same size as the input. Individuals are then evaluated with a different fitness function that is specialised to scale the difference between the input and output of an individual. Ideally, the input of the individual is equal to the output of the individual during training. This is computed using mean squared error:

$$F_r = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Where:

- N is the number of samples.
- y_i is the sample i.
- $\hat{y}_i$ is the output after inference on sample i.
- F_i is the fitness of the model for sample i.

In order to ensure that the low dimensional data remains in a normal distribution, a second factor is considered when computing fitness. Kullback-Leibler (KL) divergence [15] observes the values in the mu and log variance layers and attempts to force the parameters onto a normal distribution. This is done by punishing predictions that are outside of the normal distribution. Kullback-Leibler loss is defined as follows:

$$F_{kl} = -\frac{1}{2}\sum\left(1 + \log\sigma^2 - \mu^2 - \exp(\log\sigma^2)\right)$$

Where:

- F_{kl} is the KL fitness of the individual based on mu and log variance.

The fitness values F_r and F_{kl} are combined and serve as the overall fitness for each individual F_i. As before, lower fitness values indicate a fitter individual. Unlike the GENEVO classification model, accuracy is not a useful metric when determining metrics based on validation data. To account for this, fitness is used instead. Selection and reproduction are computed the same way as in the classification model and can be adjusted as necessary.

To sample from an individual, the layers intended for dimensionality reduction are removed and a set of random values in the range 0 - 1 are provided in place of the low dimension data. This acts as the ϵ value for calculating the normal distribution representation of the value (z). Those values are then passed to the upscaling layers and a sample that can be observed to have come from the training data distribution is output.

The GENEVO variational autoencoder(VAE) is trained on the Tensorflow import of the MNIST dataset. Labels are unnecessary and are discarded. The data is split into training and validation distributions.

C. Libraries and Hardware

The GENEVO classifier and VAE models are programmed in Python 3.12.2 [16], using only the Numpy library for linear algebra and simple mathematical operations [17].

The machine used to train the GENEVO framework has 16GB of RAM and a Ryzen 2700X, no operations are performed on GPU.

IV. Methodology

In order to gather thorough results for GENEVO, an experimentation methodology must be devised. Since there are two pipelines being explored in this paper, classification with standard neuroevolution and generative modelling with a VAE structure, experiments must be separately conducted.

A. Classification

Before experimentation can occur, the dataset being used needs to be evaluated. The MNIST dataset typically contains 28 x 28 size images, which take significantly longer to process than the pooled version of MNIST, which is 8 x 8. To justify using the lower dimension dataset for experimentation, an equal partition of training and validation data is created from each dataset. Three hyperparameters are adjusted: population, tournament, and batch size. All other hyperparameters remain constant.

The validation fitness of the full MNIST and small MNIST are displayed on a graph for each set of hyperparameters being compared.

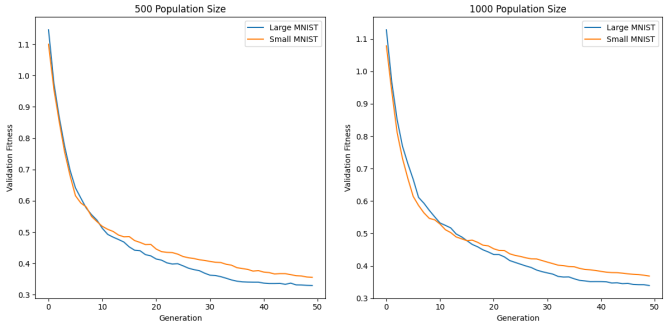

Fig. 3. Validation fitness for 500 population size (left) compared to 1000 population size (right) on a 8 x 8 (Orange) and 28 x 28 (Blue) MNIST dataset.

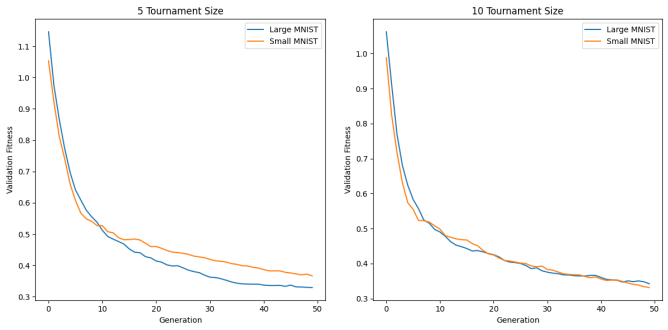

Fig. 4. Validation fitness for 5 tournament size (left) compared to 10 tournament size (right) on a 8 x 8 (Orange) and 28 x 28 (Blue) MNIST dataset.

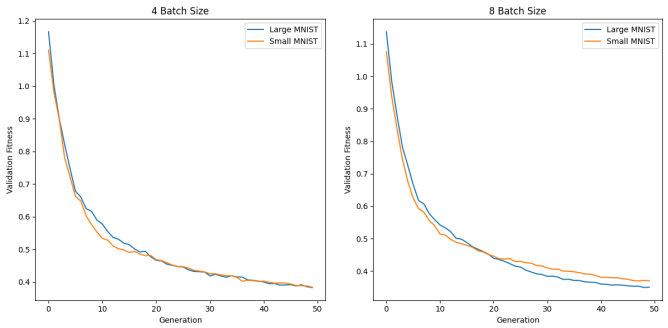

Fig. 5. Validation fitness for 4 batch size (left) compared to 8 batch size (right) on a 8 x 8 (Orange) and 28 x 28 (Blue) MNIST dataset.

As can be seen in the Figures 3, 4, and 5, similar trends indicate that the small MNIST is a comprehensive representation of the large MNIST dataset, and therefore, can be used in its place.

It has been established that the 8 x 8 MNIST dataset is sufficient to calculate metrics. Since GENEVO is being evaluated on a task specified to neural networks with a neuroevolution approach, the specific hyperparameters must be adjusted in order to observe its performance. The simplest metric to observe in order to determine the quality of the model is the validation accuracy and the accuracy of the best model

in a population. Accuracy can be compared to fitness in order to show the relationship between them.

The hyperparameters and paradigms that are adjusted for evaluation are as follows:

- Batch size - In order to evaluate the effectiveness of small batches in comparison to heavy large batches.
- Using a fixed batch instead of changing the batch for each individual evaluation.
- Structure - Comparing deep individuals to shallow individuals.
- Tournament size - Evaluating if a larger tournament size results in earlier convergence than a smaller tournament.
- Population size - To determine if the large variance in weights are due to larger population results in better generalisation.
- Mutation rate - To compare the effects of starting mutation at a higher or lower value and how it compares to the traditional neural network use of learning rates.

B. Variational Autoencoder

Variational autoencoders are a form of generative modelling, and therefore, using lower dimensional data may not fully represent the ideal task of the variational autoencoder variant of GENEVO. Since output is being generated for human observation, higher dimensional data would be more applicable, which is why the 28 x 28 MNIST dataset is used here.

The goal is to successfully reduce the dimensionality of input, reparameterise it to a normal distribution, and then upscale it again so that it looks like the input. The model must be able to create samples from random noise passed through the up scaling portion of the model.

The metric for fitness may be uninformative for this model because if the dimension reduction and upscaling parts of the model are removed, the model just learns a representation of the data that does not reduce its dimensionality.

V. Results

This section discusses the results from experimentation on the implementations of GENEVO. Each experiment is divided into the hyperparameter being tested, or the task being performed.

A. Classification

Each experiment has the following control, which shows a typical convergence before changing parameters:

TABLE I
Model Parameters

Parameter	Value
Input	64
Structure	[32]
Output	10
Batch Size	8
Tournament Size	10
Population Size	200
Mutation Rate	0.02
Mutation Scaling	[0.001, 0.001]
Mutation Threshold	0
Epochs	200

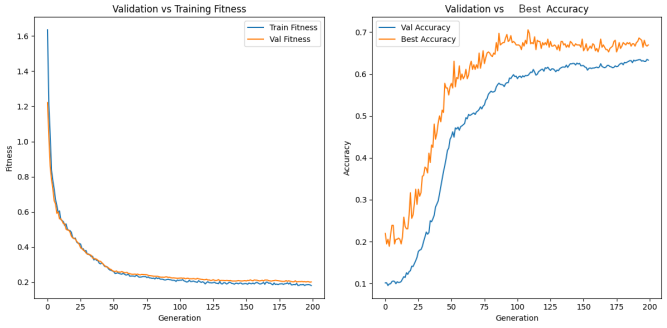

Fig. 6. Fitness for training and validation data (left). Validation and best individual accuracy (right) with the control parameters.

The control model attained an accuracy of 70.55% on validation data and 67.99% on training data. The fitness of the best model based on validation accuracy was 0.20852. The data was divided on a 1:5 ratio of validation to training data for all the models. The above results will serve as a baseline of comparison for all experimentation results.

B. Batch Size

It is hypothesised that, even though smaller batches are used, the population will still converge on an optimal result within less computation time. For this reason, average generation computation time is also included in batch size evaluation results.

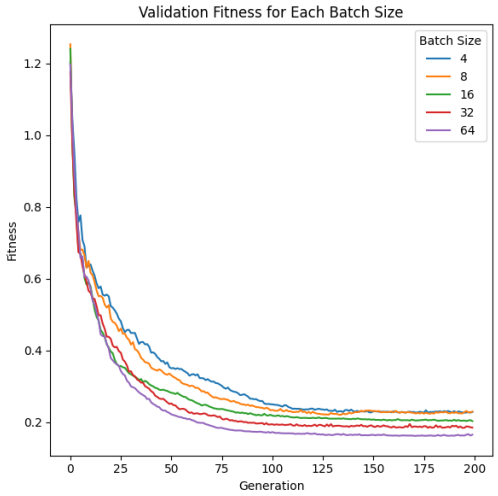

Fig. 7. Validation fitness for each batch size.

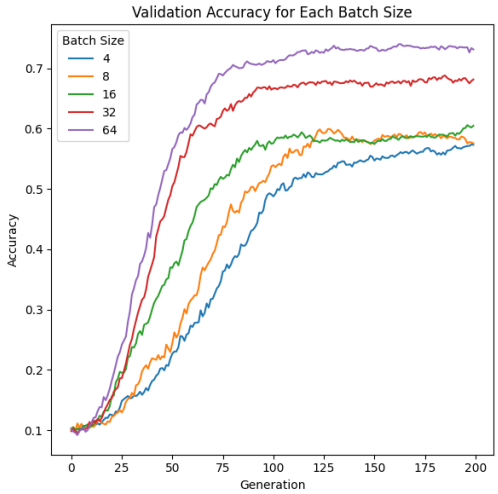

Fig. 8. Validation accuracy for each batch size.

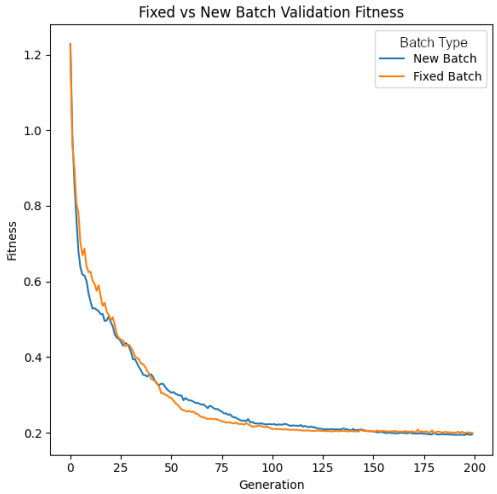

Fig. 9. The fitness of using a fixed size batch for all individuals in a generation compared to using a new generated batch for each individual in a generation.

TABLE II
FITNESS AND ACCURACY FOR DIFFERENT BATCH SIZES

Batch Size	Validation Data		Training Data	
	Fitness	Accuracy	Fitness	Accuracy
4	0.23262	0.63056	0.21951	0.63605
8	0.22092	0.68889	0.20275	0.65344
16	0.20994	0.67778	0.2016	0.64509
32	0.1853	0.72222	0.15853	0.77314
64	0.16619	0.78333	0.15105	0.81072

The Table II indicates that the best batch size within the tested batches achieved 78.33% accuracy on validation data and 81.07% accuracy on training data. The average step for each generation is observed.

TABLE III
GENERATION COMPUTATION TIME FOR EACH BATCH SIZE

Batch Size	Computation Time (s)
4	0.21
8	0.26
16	0.36
32	0.55
64	0.95

C. Fixed Batch vs New Batch

The effectiveness of generating a single batch compared to generating a new batch for each individual is compared here. The hypothesis is that generating new batches for each individual will result in better generalisation, and therefore, faster convergence and better accuracy.

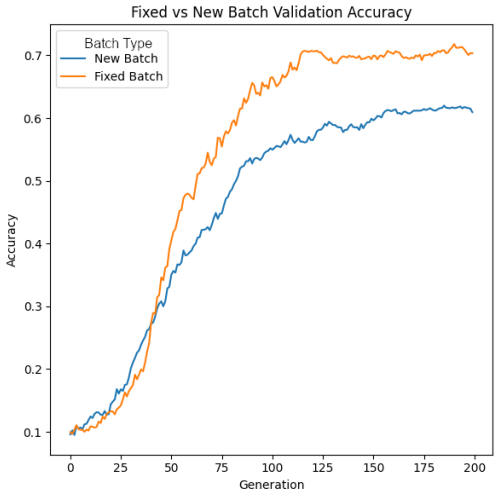

Fig. 10. The accuracy of using a fixed size batch compared to using a new generated batch for each individual in a generation.

TABLE IV
FITNESS AND ACCURACY RESULTS FOR NEW BATCH VS FIXED BATCH

	New Batch		Fixed Batch	
	Fitness	Accuracy	Fitness	Accuracy
Validation	0.20813	0.66111	0.19553	0.71677
Training	0.19848	0.74444	0.18734	0.71747

The best observed accuracy on validation data was the fixed batch, which attained 71.68%, while the best training accuracy was the new batch with 74.44%. It is noted that each generation of the fixed batch was computed slightly faster ($< 0.02s$) than the new batch approach.

D. Structure

Varying structures for each of the models will be evaluated and then compared to each other. Structure denotes the size

of the output of each of the hidden layers between the input and the output layer.

TABLE V
FITNESS AND ACCURACY RESULTS FOR DIFFERENT STRUCTURES

Structure	[16]	[32]	[64]	[32, 16]	[64, 32]
Validation					
Fitness	0.22651	0.20020	0.19086	0.24054	0.23495
Accuracy	0.63056	0.74167	0.73333	0.59167	0.63611
Training					
Fitness	0.21760	0.19973	0.16175	0.21983	0.23217
Accuracy	0.64092	0.69311	0.75017	0.65275	0.60752

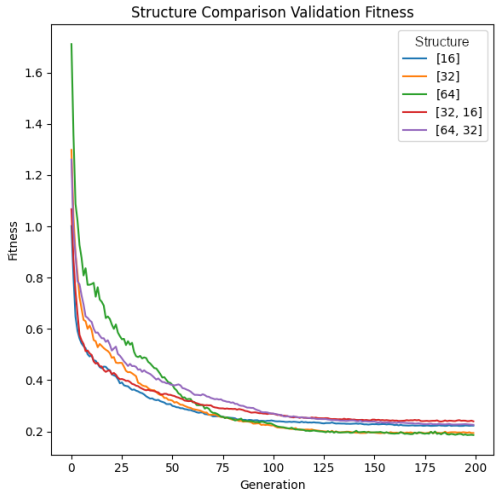

Fig. 11. The validation fitness of varying structures.

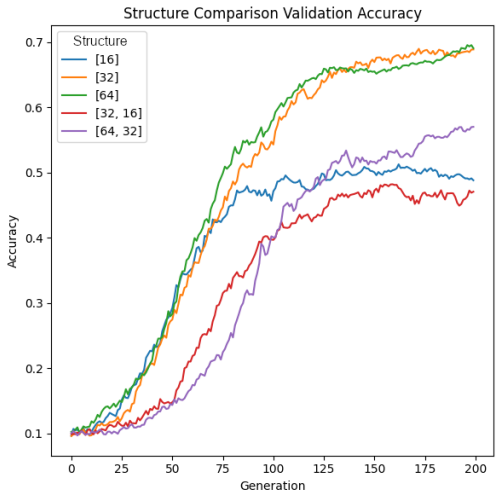

Fig. 12. The validation accuracy of varying structures.

The best validation accuracy was attained by the [32] single hidden layer model, which was 74.17%. The best training accuracy was the [64] single hidden layer, which attained 75.02%.

E. Tournament Size

During preliminary testing, it was hypothesised that the best tournament size was roughly 5% of the size of the population.

TABLE VI
FITNESS AND ACCURACY RESULTS FOR DIFFERENT TOURNAMENT SIZES

Size	5 (2.5%)	10 (5%)	20 (10%)	50 (25%)	100 (50%)
Validation					
Fitness	0.20913	0.17572	0.23400	0.41004	0.69727
Accuracy	0.72500	0.77500	0.60278	0.32778	0.22778
Training					
Fitness	0.19794	0.17145	0.23433	0.40811	0.73066
Accuracy	0.68267	0.75435	0.61308	0.27627	0.20320

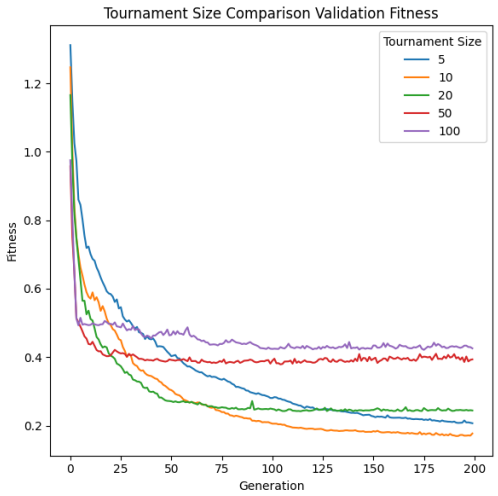

Fig. 13. The validation fitness for varying tournament sizes.

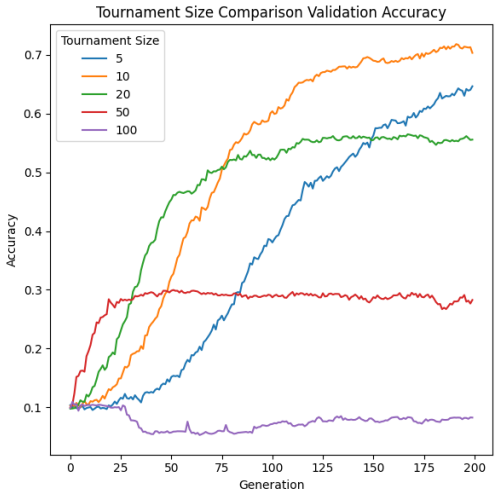

Fig. 14. The validation accuracy for varying tournament sizes.

The best validation and training accuracy was attained by the model with tournament size 10, with 77.50% and 75.44% respectively.

F. Population Size

GENEVO is given several different population sizes. It is hypothesised that the larger the training size is, the better the generalisation because there is more variance in the population.

TABLE VII
FITNESS AND ACCURACY RESULTS FOR DIFFERENT POPULATION SIZES

Size	50	100	200	500	1000
Validation					
Fitness	0.35585	0.28253	0.16884	0.15760	0.17511
Accuracy	0.29722	0.52500	0.78611	0.79444	0.78889
Training					
Fitness	0.36727	0.26596	0.16821	0.15575	0.14861
Accuracy	0.28114	0.49130	0.75922	0.78010	0.84760

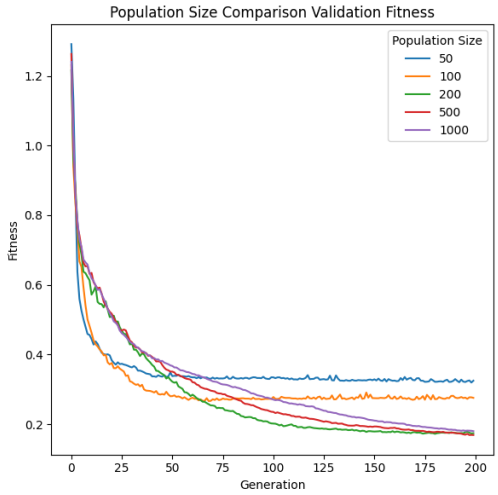

Fig. 15. The validation fitness for varying population sizes.

Fig. 16. The validation accuracy for varying population sizes.

The best accuracy on validation data was attained by the model using the population size of 500, which was 79.44%. The best training accuracy of 84.76% was attained by the model using the population size of 1000.

G. Mutation Rate

Different mutation rate values are compared to one another in order to assess the relationship between adjusting them and the convergence time. It is hypothesised that the mutation rate will act similarly to the behaviour of the learning rate in traditional neural networks. The mutation threshold and mutation scaling is set to the same values as in the control case.

TABLE VIII
FITNESS AND ACCURACY RESULTS FOR DIFFERENT MUTATION RATES

Rate	0.005	0.02	0.05	0.1	0.25
Validation					
Fitness	0.35585	0.28253	0.16884	0.14598	0.17511
Accuracy	0.29722	0.52500	0.78611	0.80556	0.78889
Training					
Fitness	0.36727	0.26596	0.16821	0.1307	0.15371
Accuracy	0.28114	0.49130	0.75922	0.80237	0.77662

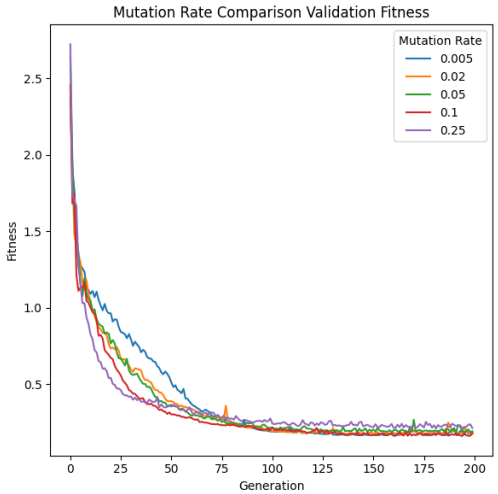

Fig. 17. The validation fitness for varying mutation rates.

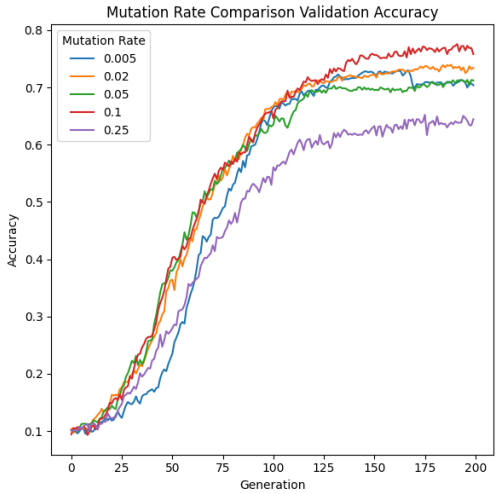

Fig. 18. The validation accuracy for varying mutation rates.

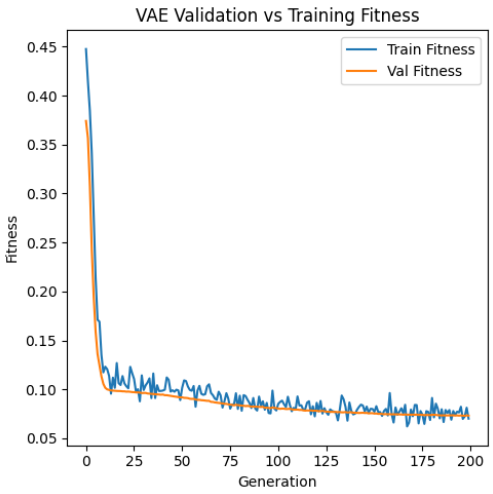

Fig. 19. Validation and training fitness over generations for the variational autoencoder.

The best accuracy on validation data for varying mutation rates was 80.56% with a mutation rate of 0.1. The best training accuracy was also observed with a mutation rate of 0.1, the attained result was 80.24%.

H. Variational Autoencoder

After observing the results from experimentation on the classifier model, parameters on the GENEVO VAE are adjusted accordingly. It is important to note that the experimentation for the VAE is computed using the 28 x 28 MNIST dataset in order to provide higher resolution generated samples. As was observed in Figure 5, 3, and 4, these parameters should follow the same trend as the smaller 8 x 8 MNIST dataset.

TABLE IX
PARAMETER VALUES FOR VAE CONFIGURATION

Parameter	Value
Encoder Structure	[32]
Reparameter Size	2
Decoder Structure	[32]
Batch Creation Methodology	Fixed Batch
Batch Size	16
Tournament Size	10
Population Size	500
Mutation Rate	0.1
Mutation Scaling	[0.001, 0.001]
Mutation Threshold	0
Epochs	200

The figures in Table IX are the parameters for the GENEVO VAE, which have been set based on trends in the classification model and computation time considerations.

The model began with a fitness of 0.16441, and after 200 epochs attained a best fitness of 0.07088.

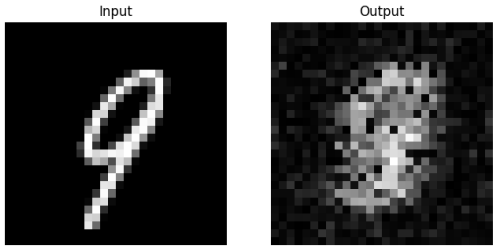

Fig. 20. Input (Left) compared to the reconstruction of the input (Right).

Figure 20 shows that reconstruction is poor but it does generalise around the center of the image. Similarly, sampling the low dimensional space of the VAE yields poor results that look similar to the reconstruction.

VI. DISCUSSION

Results from experimentation on the GENEVO framework have been collected and observations can now be made on the data.

A. Classification

Some results do match their hypothesised behaviour, however, there are several that do not. It is important to note that experimentation was done using a fixed fold of training and validation data, but the seed was permitted to be random. This means that the focus of experimentation is not on exact values, but rather on trends.

1) Batch Size: Increasing the batch size did improve the overall accuracy of GENEVO. It is suspected that this is because individuals in the population are initially exposed to more variation of the data, and therefore, performance across this broader spectrum of samples permits better generalisation. It should be noted that even though the batch size may be larger, the difference in computation time seems to only scale linearly.

2) Fixed Batch vs New Batch: The original hypothesis that generating a new batch for each individual for evaluation was not only slower, but performed worse than a fixed batch. This is hypothesised to be because of the selection of samples; models that are evaluated on simpler samples yield better results even though they are not an improvement on the model. Creating a dataset of complex samples separate from the standard dataset may be a way to introduce necessary variance individuals in order to attain even better generalisation.

3) Structure: It is observed that larger single hidden layer structures tend to perform well in comparison to multi-hidden layer structures. Further exploration shows that this could be related to the inability of the model to create meaningful updates between layers. This is in contrast to traditional backpropagation, which propagates gradients through all individual hidden layers.

4) Tournament Size: Preliminary testing indicated that 5% of the population size was adequate to select the ideal tournament size. However, further experimentation showed that a tournament size of 10 performed well on a range of population sizes, including 200, 500, and 1000. These results indicate that either a larger population size does not necessarily correlate to better results, or, the selected tournament size of 10 allows for an ideal variation for convergence in genes.

5) Population Size: It was hypothesised that larger population sizes contain more variance, and therefore, will generalise to a higher accuracy. While this was true when comparing the smaller population sizes (50,100) to the larger ones (200,500,1000), the model seemed to generalise to the same accuracy. It is noted that the largest population size of 1000 performed better on training data, but seemed to perform worse on validation data. It is hypothesised that this is because of the variance in genetics introduced by the larger population. However, the evaluation data is a specific collection of necessary genes required for good performance.

6) Mutation Rate: The hypothesis suggested that the mutation rate would act similarly to the learning rate in traditional neural networks. Comparable results were observed when testing the mutation rate. Larger mutation rates over-adjusted the weights in the model and lead to worse accuracy and fitness values due to the lack of fine tuning. The smaller mutation rates had high variance in the results because lost, well performing genetics are never rediscovered. An interesting observation is that the ideal mutation rate is approximated to be 10%. This is suspected to be because of the sigmoid activation function used by the layers in the GENEVO framework, which limits the outputs to values between 0 and 1. As a result of this limitation on the output values, more mutations do not necessarily result in worse performance, as ineffective mutations correspond to only minuscule value changes in the layer outputs.

B. Fine Tuned Classification

After gathering the results of the experiments and selecting new hyperparameters, a new, fine tuned model can be devised. The best GENEVO model found for the 8 x 8 dataset had an average fitness of 0.0959 for validation and 0.05617 for training data. Its accuracy was 89.17% on validation data and 94.92% on training data. Based on these results, the model overall took roughly 55 minutes to train, with an average generation time of 5.6 seconds. However, the best results were observed after 25 minutes.

C. Variational Autoencoder

The GENEVO variational autoencoder did indicate basic generalisation to the location of digits in the MNIST dataset, however, it did not successfully reconstruct input. It is suspected that this is because of the lack of complexity in the model and relationships between layers in individuals.

D. Other Considerations

For further exploration in neuroevolution, the utilisation of sigmoid as the activation function across all layers in an individual can lead to vanishing gradients. To avoid this, evolution of activation functions can also be introduced, which may result in better generalisation [18].

VII. Conclusion

In conclusion, GENEVO (Genetic Evolutionary Optimisation) demonstrates the effectiveness of neuroevolution, particularly genetic algorithms, as an alternative to traditional backpropagation in training neural networks. The study highlights how varying parameters like population size, mutation rate, and tournament size impact model performance and convergence. A smaller batch size was shown to achieve comparable generalization while reducing computational cost. Furthermore, the exploration of variational autoencoders (VAEs) within the GENEVO framework revealed promising, though limited, potential for generative tasks. This work positions GENEVO as a competitive alternative to traditional neural network training methods.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.

[2] M. Mitchell, "An introduction to genetic algorithms," *Machine Learning*, vol. 4, no. 1, pp. 1–20, 1998.

[3] K. A. De Jong, "The use of genetic algorithms for combinatorial optimization," in *Proceedings of the Genetic Algorithms*, Morgan Kaufmann, 1997, pp. 1–12.

[4] K. O. Stanley, *Neuroevolution: From architectures to learning*. Springer, 2010.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[7] C. M. Bishop, "Neural networks for pattern recognition," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 314–319, 1995.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[9] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[10] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1998.

[11] Y. Such, M. Hord, and K. O. Stanley, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2017, pp. 49–57.

[12] E. Real, J. Clune, D. Ha, Q. Hu, A. Reynolds, and J. Y. White, "Large-scale evolution of image classifiers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, vol. 31, 2017, pp. 4780–4787.

[13] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[14] N. V. Otten. "Variational autoencoders (vaes) made simple & how to tutorial." Accessed: 2024-10-08. (2023), [Online]. Available: https://spotintelligence.com/2023/12/27/variational-autoencoders-vae/.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2014, Accessed: 2024-10-14. [Online]. Available: https://arxiv.org/abs/1312.6114.

[16] P. S. Foundation, *Python language reference, version 3.11*, Python Software Foundation, 2023. [Online]. Available: https://www.python.org/.

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[18] S. Scardapane, A. Baiocchi, A. Devoto, V. Marsocci, P. Minervini, and J. Pomponi, "Conditional computation in neural networks: Principles and research trends," *Papers with Code*, 2024, Accessed: 2024-10-14. [Online]. Available: https://paperswithcode.com/paper/conditional-computation-in-neural-networks.