MLP Coursework 4: Final Report

G52: s1775157, s1784139, s1778848

Abstract

In our research we focus on hyperparameter optimization for standard fully-connected neural network architectures using the evolutionary algorithms of Evolution Strategies (ES) and the Genetic Algorithm (GA). We extend our previous research from using the GA for classification on the MNIST and EMNIST datasets to using both ES and GA for classification on the much more expansive OMNIGLOT dataset. Our experiments with different configurations of the ES and the GA algorithms indicate that both ES and GA can find networks with comparable accuracy on OMNIGLOT. However, when using the GA it is difficult to find and set the initial ranges for the hyperparameters that produce the best results. At the same time, we note that the gradient-following process of ES risks entrapment in a local optima. When we combined both the GA and ES algorithms, we found our best network architecture. As the purpose of hyperparameter optimization is to find the best neural network architecture in the first place, we report an accuracy of 51.42% on the test set of the OMNIGLOT dataset with this architecture. We conclude that both GA and ES are suitable for hyperparameter optimization of fully-connected neural networks, and future research can work towards extending these techniques to hyperparameter optimization of convolutional layers on OMNIGLOT.

1. Introduction

When designing a fully-connected neural network, there are many different hyperparameters to set in order to achieve best performance on a task. In the first semester, we tried to set these hyperparameters through time-consuming manual search processes. However, manual search can be inefficient and also suboptimal. Consequently, this provides the motivation for our research in finding alternate methods of hyperparameter optimization for neural network architectures. Each neural network has four different hyperparameters: number of hidden layers, number of hidden units, the activation function between layers, and the learning rule. We investigate two main approaches to optimizing these hyperparameters in our experiments: the Genetic Algorithm (GA) and Evolution Strategies (ES). Both GA and ES are evolutionary algorithms, which are a class of heuristic algorithms that use techniques inspired by nature for solving optimization problems. To test these different approaches, we measure their performance in designing fully-connected neural networks for the MNIST, EMNIST, and OMNIGLOT datasets for character recognition.

We had originally planned to use the MNIST, EMNIST, and CIFAR-10 datasets to examine how different settings of the GA affected the accuracy of the neural networks. In our previous work with the GA, we achieved good baseline results on MNIST and EMNIST of similar accuracy (approximately 98% and 85% for MNIST and EMNIST respectively) to the random search methods we performed in our coursework of the previous term. However, we noticed earlier that it was hard to do analysis on the GA with the MNIST and EMNIST datasets because they seemed to have more limited sensitivity to both GA parameters and hyperparameter optimization under GA. Regardless of the GA settings, we obtained comparable accuracy. Thus, we decided we needed a harder dataset. However, if we used the CIFAR-10 dataset we would have had to move into convolutional networks for these image classification tasks. Considering how many networks we need to train for different GA settings, we decided that the computational intensity of using convolutional networks was too much for this research. Consequently, we decided to use the OMNIGLOT dataset to continue our work in deep hidden networks. Thus, our new objective became to use the OMNIGLOT dataset to study how different GA settings, such as encoding method and population size, affected the accuracy of neural networks. The idea being to find GA settings that produced accurate neural networks, while minimizing the number of networks that needed training.

Additionally, we now have expanded our work to a new objective of examining how ES does in comparison on the same classification tasks with OMNIGLOT. As ES is an alternate method for hyperparameter optimization, we can use it on the OMNIGLOT dataset to try to achieve accurate neural networks. Since both the ES and GA are different optimization techniques for the network architecture, we decided to perform a comparison of different ES techniques against different settings of the GA to provide insights into the advantages and disadvantages of each method.

2. Methodology

2.1. Genetic Algorithm

We described in extensive detail the GA in our previous work, so we provide a brief overview of the process here. Basically, the GA is a heuristic process that takes a popula-

tion of solutions and evolves them in an attempt to maximize their performance on a task (Whitley, 1994). In our case we seek to optimize the hyperparameters, or the network architecture. The GA starts with a population of different neural network architectures and these solutions are then evaluated based on their fitness or performance on the classification task given. The best solutions then have a higher chance of producing offspring in the next generation that combine the hyperparameters of their parent solutions. The idea is that the best solution will come from the repeated combination of previous solutions after a fixed number of generations. The main techniques relevant to the GA are: Selection, Crossover, and Mutation and are explained extensively in Coursework 3 and in (Carr, 2014). As a brief orientation: Selection is the process of how the GA selects parents for the offspring in the next generation; Crossover governs how the parents combine their architectures to produce an offspring; Mutation prescribes the random chance that an offspring changes its hyperparameters after it is produced before training.

We explain here the method of evaluation for a network as it fundamentally affects how GA – and ES – work. After training the network, and obtaining its best validation accuracy, we apply a fitness function to transform the accuracy to its fitness value. We need a fitness function because validation accuracies can become very similar (frequently within a percent) in experiments, so the Roulette-Wheel Selection process has little distinction between the network's fitness level. Essentially this means that its harder for the Selection process to bias parent selection towards the relatively more fit network architectures (see previous work or (Whitley, 1994) for a more thorough explanation). Our fitness function thus emphasizes the fitness differences between networks that perform relatively worse and those that perform better. Additionally, our fitness function also we need to give bigger penalty to the networks that perform the worst. Finally, for implementation reasons, we used the fitness function to revert from maximization to minimization problem. Let x be a network's validation accuracy, then the fitness function f is defined as:

$$f(x) = 2^{(1-x) \times 10} \tag{1}$$

In addition to our previous work, we also experiment with the use of Binary Encoding of the chromosome, or the network architecture's representation. With Integer Encoding we observe that the offspring is always bounded inside the hyperparameter space of its parents that we can call a tesseract, as we optimize 4 hyperparameters. Through Integer Encoding the offspring of two parents will inherit part of the first and part of the second parent, and in case the crossover point is on a hyperparameter, the average value of the parents is inherited. Binary Encoding combines all hyperparameters into a binary chromosome, where each section of the chromosome represents a hyperparameter (Whitley, 1994). The length of the binary chromosome depends on the range of values of each hyperparameter. When it comes to Binary Crossover, the binary chromosome is a one dimensional array of binary digits where we randomly select an index as our crossover point and we take all digits up to that index from the first parent, and the remaining from the second. This allows us to escape the tesseract that limits us with Integer Encoding and we can produce a completely new network architecture in the Crossover process. Integer Encoding Crossover is more limited as there are fewer possibilities for combinations of parent architectures.

To illustrate the difference between Binary and Integer Encoding, we use a simple example of a single hyperparameter representing the number of hidden layers as illustrated in Figure 1. There are five possible crossover points for Binary Encoding. With Integer Encoding, there are only three different outcomes. Either the offspring will fully copy one of the parents, or will take the average of them, resulting in the following possible values for that hyperparameter: [6,8,9], were the value 8 is the rounded value of the average. On the other hand, Binary Encoding with 5 different crossover points is able to produce eight different values from those two parents: [1,5,6,7,8,9,10,14], depending as well on which chromosome is the first parent. In the example, we see that the actual crossover point for Binary Encoding is after the second cell, which means that the offspring will take each half from its parents.

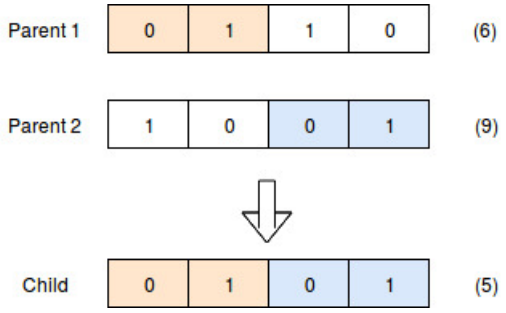

Figure 1. Binary Crossover of a single hyperparameter (Hidden Layers) chromosome. Figure is our own.

Similarly, Binary Mutation, which is the process of randomly selecting a binary digit of the chromosome, and switching its value, has the potential to either make a small or a large alteration to the value, depending on the digit's position. In our example, assuming a mutation occurs on the child's first digit, the value changes from 5 to 13, but if the mutation occurred on the least significant bit, the value would have change from 5 to 4.

2.2. Evolution Strategies

An alternative method to approach hyperparameter optimization is through the use of ES. The basic idea behind ES is to start with an initial solution and then use a multivariate Gaussian distribution to form a population of solutions through sampling the hyperparameter space around the initial solution. The number n of solutions in the population, and the covariance matrix Σ of the multivariate normal distribution are all parameters of the ES. We detail their exact values in our experiments. Each neural network in the new population is trained and then achieves a fitness score com-

miserate to its performance in validation accuracy on the classification task, with the fitness score calculated similarly to the GA process above. Then based on the fitness scores of the other sample solutions, the initial solution updates its hyperparameters to achieve a better fitness score. The exact update process depends on the type of ES, and the update process repeats for a fixed number of generations (another parameter of the ES) or until convergence where the solution does not improve. In our research we explore two different types of ES. For clarity, we now refer to the updated initial solution in any generation as the *candidate* solution.

In the simple ES, the candidate solution simply clones, or adopts, all of the hyperparameters of the best sample solution in a greedy update process. The next generation then forms around the new candidate solution according to the multivariate normal distribution. The algorithm evaluates these networks and the process repeats.

At the end of 2017, researchers demonstrated how a version of *Natural Evolution Strategy*, NES, can be effective and efficient in solving parameter optimization for complex problems such as humanoid walking (Salimans et al., 2017). In their research, NES is essentially doing gradient descent on the weight parameters for neural networks. The NES is similar to the simple ES, but instead of just copying the best solution as the candidate solution, NES uses the fitness or performance of the other solutions to estimate the gradient of the solution space and better guide the candidate solution. Each different solution i of the n total solutions is formed by altering the candidate solution with Gaussian noise ϵ_i from a Normal distribution with variance σ. Let each solution i have an associated fitness F_i. Then, letting θ_t be the weights at time t and α be the learning rate, we have that the updated weights are given by (Salimans et al., 2017):

$$\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i. \tag{2}$$

While the research in (Salimans et al., 2017) focused on using NES for weight optimization, we can also adapt NES for hyperparameter optimization. The advantage of NES is that it does not require computing exact gradients, which means that it can work on non-continuous and non-differentiable spaces. So although the hyperparameter space we use for neural networks is neither continuous or differentiable, we can still use NES to optimize the architectures. For hyperparameter optimization, the NES now starts with a candidate neural network architecture and the hyperparameters are modified with Gaussian noise to generate a new population of solutions. The Gaussian noise now comes from a multivariate normal distribution with a mean vector $\vec{\mu}$ and a covariance matrix Σ. For each of our experiments we detail these choices. The solutions are then trained and evaluated to obtain a fitness ranking as before. The algorithm then updates the candidate architecture in the direction of the gradient in proportion to the learning rate α. The value of α is an additional parameter of the NES not seen in simple ES and so has to be also set separately. The process of updating the hyperparameters with NES can repeat until convergence, but due to computational restraints we repeat for a fixed number of generations.

2.3. Datasets

In our research we used the MNIST, EMNIST, and OMNIGLOT datasets. All three datasets are for character recognition tasks and each represents a higher level of difficulty. Previously we focused on the MNIST (LeCun et al., 1998) and EMNIST (Cohen et al., 2017) datasets which has 10 and 37 characters respectively. Our previous work provides further details on these two datasets. As mentioned in Section 1, we discovered that the MNIST and EMNIST datasets were not very sensitive to changes in the GA parameters and hyperparameter optimization under GA. We wanted to keep within the character recognition task as we had experience working on this task, but we wanted a harder task that could be done with fully connected networks. Consequently, we decided to move to the harder OMNIGLOT dataset (Lake et al., 2015).

In OMNIGLOT, there are fifty different character alphabets for a total of 1,623 different classes. Additionally, each character class consists of twenty renditions of the character from different people, so there are 32,460 images total. We adapted the dataset from the associated GitHub source to work in our current TensorFlow setup. We divided the dataset up into training, validation, and test sets of sizes 22,722 (70%), 4,869 (15%), 4,869 (15%) respectively.

3. Experiments

We first layout the commonalities in the design setup across all of our experiments for both GA and ES. The GA and ES trained fully connected standard neural networks on the training set and then evaluated their performance on the validation set to obtain fitness scores as explained in Section 2. To initialize the weights for all of the neural networks, we used the Glorot initialization process that sets all the weights to very small random values (Glorot & Bengio, 2010). Though the number of hidden layers was variable, each neural network had a final softmax layer for classification and the networks trained on the cross entropy error. Additionally, we used a common mini-batch size of 100 for all of our experiments with the OMNIGLOT dataset. To help standardize the experiments, we used the same random seed each time.

As for the hyperparameter solution space, we did have some common ranges for both the learning rules and the activation functions. For learning rules, both ES and GA had the choice of SGD with learning rate $\eta = 0.15$, a momentum learning rule with momentum of 0.9 and learning rate of .001, and either RMSProp or Adam with the initial setup suggested by their creators in (Tieleman & Hinton, 2012) and (Kingma & Ba, 2014) respectively. For further explanations of the choices of learning rules, as well as their explanations, please see our previous work. As for activation functions between each hidden layer, the GA and

ES could choose ReLU, Leaky-ReLU, ELU, or SELU. We suggest (Clevert et al., 2015) for a comparison of these activation functions as well as their mathematical definitions. Finally, we specify the ranges for the other hyperparameters of hidden layers and hidden units in each round of experiments as we changed these based on the results obtained.

To standardize the comparisons of our results across our different neural network setups, we always report the highest validation accuracy a network obtained. Different network architectures require varying numbers of epochs to train, and so we used only the best validation accuracy results from each network for comparison. In effect, this meant that we were doing a version of early stopping where the networks are compared at their best results.

3.1. First Round

Our first round of experimentation focused on achieving diversity in the population of solutions for the GA. The idea being that a more diverse population more thoroughly explores the hyperparameter space. Thus, the GA has more probability of finding betters solutions. In our previous work we had tried modified elitism on a population of size 10, where we kept the bottom five and top five network architectures as possible parents for the next generation. The idea behind modified elitism being that we could keep training a population of size 10 only, but still try to achieve diversity in the population by having more potential parent networks. However, we still were getting less diversity of population across generations as we see in Table 2. Thus, extending from our previous coursework, we decided upon a different tactic of using a larger population of size 100 with regular elitism. The idea being that a greater population from the start means that the crossover process of the GA has more possible solutions to combine or mutate in order to find a better solution.

In the first experiment we used the GA to train neural networks on the EMNIST dataset with a solution population of size 100 (instead of 10 as in our previous work) over 10 generations. The GA had control over the number of hidden layers (1-8), number of hidden units (32-128), activation functions, and the learning rule just as in the final experiments in Coursework 3. For consistency, the number of hidden layers possible was one to eight and hidden units was 32 to 128. Additionally, we used Roulette-Wheel as the crossover process because it had marginally better performance than Tournament selection in our previous experiments on EMNIST. The mutation rate was 0.4 and the mutation possibilities for each hyperparameter were -3, -2, -1, 1, 2, 3. As a reminder from previous work, if a mutation occurs then each hyperparameter has a uniformly random chance of selecting one of the mutation possibilities and adding that to its current integer value since we are using integer encodings (including for activation functions and learning rules). We used regular elitism, where the best solution from the previous generation is copied into the next generation. The idea is that the best solution from the previous generation can act as a guide in the direction of a relatively well-performing part of the hyperparameter space, but the GA still can use other solutions to explore other hyperparameter choices. As we now had many more potential parent networks across generations, we no longer needed to use modified-elitism.

In Table 1, we show the final validation accuracy achieved by GA with a population of 100 and regular elitism in comparison with the best baseline accuracy we achieved with modified elitism on a population of size 10. We see that the increase in population had little effect on the final validation accuracy achieved with both having comparable accuracy within 0.2%, a likely result of different initialization. Consequently, for the EMNIST dataset using a bigger population size of 100 did not actually help achieve better results.

Population Size	10	100
Classification Accuracy	85.361%	85.114%

Table 1. Table comparing best validation accuracy of GA obtained on the EMNIST dataset with population size of 10 with modified elitism and an increased population size of 100 with elitism.

We examined the percentage of unique solutions across each generation in Table 2 to see whether the increase in population size actually helped with genetic diversity. We compare against our baseline experiment from our previous work with population size 10, with the additional 10 networks from the previous generation retained under modified elitism (20 total). Our results show that the population of size 100 had a higher percentage of unique solutions. The higher population size means there are more potential parents in the previous generation for offspring in the subsequent generation. As a result, the offspring have a higher chance of choosing different parents in the crossover process and thereby obtaining different hyperparameters. While the Roulette-Wheel Selection biases the parent selection to the more successful networks, the higher population means that there are more networks with comparable accuracy and so more parent choices for offspring.

Generation	10-Population Uniqueness	100 Population Uniqueness
1	100%	100%
2	90%	100%
3	95%	100%
4	95%	100%
5	80%	100%
6	75%	100%
7	95%	100%
8	90%	100%
9	80%	100%
10	85%	100%

Table 2. The percentage of unique neural networks per generation under the GA with population sizes of 10 with modified elitism and 100 with elitism.

While the GA did not obtain higher accuracy with a larger

population, it did maintain a more diverse population. Especially when there are only twenty networks total (ten from the previous generation under modified elitism) that are potential parents, a loss of diversity means that there is less thorough exploration of the hyperparameter space. While having a more diverse population is not the goal of our experiments, it does indicate that the algorithm did a more thorough search through the hyperparameter solution space. On more complicated datasets that are more sensitive to hyperparameter changes, a more extensive search of the hyperparameter space helps avoid entrapment in local optima. Consequently, moving onto the harder OMNIGLOT dataset with the GA, we decided to keep the diversity of the population with a population size of 100.

3.2. Second Round

In the next round of experimentation, the major change was using the OMNIGLOT dataset for classification accuracy. In our previous work, we had already seen that the GA was quickly obtaining the best classification accuracy within a few generations and thus did not seem to be very sensitive to hyperparameter change. Now that we had settled on a higher population from our first experiment to help do a more thorough search of the hyperparameter space, we moved to the harder OMNIGLOT dataset. As there are many more characters of many different languages, the classification task became significantly harder. Consequently, we believed that the hyperparameter selection for the GA is more critical in determining classification accuracy.

This round of experimentation was about finding suitable hyperparameter ranges for the GA on the OMNIGLOT dataset. In the first experiment of this round, we used GA on the OMNIGLOT dataset. We also changed our mutation rate from 0.4, down to 0.1, but allowed the mutation greater variability for hidden units allowing values of -30,-20,-10, 10, 20, 30. The GA could choose 1 to 8 hidden layers, 32 to 128 hidden units, the standard learning rules and activation functions. The GA had a population size of 100 over 10 generations. We observed that we needed to increase the number of hidden units and hidden layers available as the best networks had architectures at the end of the range of the hyperparameter space. As we added more hidden units and layers, we also increased the number of epochs (up to 200) that the neural networks trained because more hidden units led to higher training epochs to get the best validation accuracy. The results with the highest accuracy achieved for each hyperparameter range are in Table 3.

Hyperparameter ranges	Validation accuracy
32-128 HU, 1-8 HL	46.58%
64-256 HU, 4-12 HL	49.04%
128-384 HU, 4-12 HL	50.45%

Table 3. GA performance on the OMNIGLOT dataset with different hyperparameter ranges, where HU stands for hidden units, and HL for hidden layers.

We stopped at 384 hidden units because the best architecture used fewer than the limit (later in Round 5 of experiments we realized we had stopped prematurely). From this round of experimentation, we also realized that the current implementation of GA does have a drawback in the difficulty to find an appropriate range for hyperparameter settings. We note that we did limit GA to not be able to mutate outside of the initial range. However, we did this because we wanted GA to do a thorough search of a given hyperparameter space. A random mutation outside of the initial hyperparameter space may lead to a better solution, but it's also very likely to lead to just a worse solution because of its randomness. Thus, we wanted the GA to focus on one area of the hyperparameter space and not just experiment with random mutation outside of it. In addition, our following rounds of experiments would be with ES, which follows an estimated gradient of the hyperparameter space to find the best solutions. Thus, ES actually has a more refined method for searching outside its initial hyperparameter bounders than just random mutation.

3.3. Third Round

We now moved into experimentation with ES. First, we wanted to see if either evolutionary algorithm had better performance on finding network architectures with higher classification accuracy. Further, we wanted to investigate and better understand the differences between the search techniques for GA and ES. This way we could analyze their performance and identify the relative advantages and disadvantages of each search technique on hyperparameter optimization. Consequently, our first experiment in this round consisted of using Simple ES on the OMNIGLOT dataset. The initial candidate solution had a random chance of appearing in the hyperparameter ranges of: 1-8 for Hidden Layers, 32-128 for Number of Hidden Units, and any combination of four Activation Functions and four Learning Rules that were described in coursework 3. Besides defining the initialization space, for Simple ES there is also the hyper-hyperparameter of σ. As ES requires a normal distribution around the individual for the generation's sampling, finding the correct σ value for each hyperparameter is crucial. In addition, using the same σ value for all four hyperparameters is not appropriate, as for example a noise value of 2 is not actually proportionate when it comes to hidden units, compared to hidden layers. For that reason, we defined four σ values, [2,2,2,20], where the last one is for hidden units. We show the results in Table 4 of the initial experiment with ES in comparison to the GA performance of the previous round of experiments.

EA Algorithm	Validation accuracy
ES (best)	50.60%
GA (best)	50.45%

Table 4. Best network accuracy from ES and GA for 100 generations and population size of 10.

We note that ES and GA both found networks of near comparable accuracy. However, we realized that the ES might be not performing as well as possible because its evolution process relies on gradually shifting the candidate in the direction of the gradient. The total number of networks trained in all 4 cases, still was 1000, which meant that both GA and ES were training the same number of networks so we could continue to compare results. The experiments with different generation and population setups and the results are presented in Figure 5.

.

ES Setup	Best Validation accuracy
100 Gen 10 Pop	51.97%
10 Gen 100 Pop	50.60%
50 Gen 20 Pop	51.20%
20 Gen 50 Pop	50.31%

Table 5. Best network accuracy from ES and GA, with the same initialization space, for 100 generations and population size of 10.

We see that training the simple ES for 100 generations actually achieved the best performance. This is likely due to how ES requires many generations of adaption for the candidate solution to follow the locally estimated gradient to an optima. Finally, when examining the solution from ES that performed the best we saw that it actually had more hidden units than we had allowed the GA. Our GA implementation does not explore outside of the hyperparameter ranges given (see previous round for the explanation of our reasons), so it did not and could not find the better solution that ES found. The candidate solution in ES has no bounds on its movement in hyperparameter space.

3.4. Fourth Round

In the next round of experiments, we wanted to experiment with more configurations of the GA, including using what we learned from the ES. This time we increased the range of the GA's hidden units to be from 384-640 hidden units (trained for 200 epochs) so that it could explore the same area of hyperparameter space where ES had found its best solution. Following the same reasons, we scale the range of Hidden Layers to [4-12]. The results are in Table 6.

Algorithm	Best Validation accuracy
GA (Integer Encoding)	52.35%
Simple ES	51.97%

Table 6. Best network accuracy from ES for 100 generations and population size of 10 and GA for 10 generations and population size of 100. 384-640 Hidden Units, 4-12 Hidden Layers initialization space for the GA.

In comparison with the simple ES, GA now actually perform better. This outcome is probably related to how the GA has the ability to throughly search its defined area, in contrast with the simple ES that is actually moving around the solution space, chasing the better candidate. At this point we applied Binary Encoding on the GA for the reasons explained more extensively in Section 2, mainly to allow for more choice in both the Crossover and Mutation processes. The results are in Table 7.

Algorithm	Best Validation accuracy
GA (Binary Encoding)	52.67%
GA (Integer Encoding)	52.35%
Simple ES	51.97%

Table 7. Best network accuracy with Integer and Binary Encoding for 10 generations and population size of 100. 384-640 Hidden Units, 4-12 Hidden Layers

The different technique offered by the Binary Encoding for crossover and mutation seemed to allow GA and ES to achieve slightly better results. We offered explanations for this in Section 2. However, further experimentation is needed to confirm such findings in the first place.

3.5. Fifth Round

Under Simple ES, the candidate function follows a greedy strategy of simply copying the best performing network. On the other hand, NES follows a more nuanced maximization strategy where the candidate solution estimates a local gradient in each generation and then follows it for the subsequent generation to achieve better performance (Salimans et al., 2017). In this section, we wanted to experiment with NES to see if its local gradient estimation could help with the finding better hyperparameter solutions. We could then compare NES with simple ES as well.

As explained above in Section 2, the learning rate α in Equation 2 determines how much the candidate solution adapts in the direction of the local gradient. Consequently, we needed to find a suitable α value for NES in order to progress in the direction of an estimated local gradient but not too quickly to skip the direction of the real gradient. In order to pick the best suitable learning rate constant α, we ran an NES experiment with 5 generations of 20 population each combining with the selected α constant values of 0.01, 0.05, 0.1, 0.5, 1, 1.5, 2, 2.5, 3 and 3.5. We found that an α value of three had the best progression in validation accuracy by the end of fifth generation (46.50%), so we used it for our actual experimentation with NES.

We now were ready to use an $\alpha = 3$ for trying NES and comparing it to Simple ES. So we ran an experiment of population 20 and 50 generations with NES. All of the other hyperparameters were the same as our runs with Simple ES. Table 8 shows the comparison of accuracy value attained for NES against the best simple ES configuration from the previous experimental results.

Upon examination, we see that NES actually performed worse than simple ES. While further experimentation was unfortunately not possible due to computational and time

Algorithm	Best Validation accuracy
NES	48.31%
Simple ES	51.97%

Table 8. Best accuracy from NES for $\alpha = 3$ against best simple ES configuration.

constraints, we hypothesize that the result was likely due to needing a more careful selection of the learning rate parameter for NES α. A too high α value may have resulted in the candidate solution continually skipping over the optimal region for the hyperparameters. Further tuning the α parameter must then be a task for future work.

3.6. Sixth Round

From the analysis of experimental results of GA and ES individually, we decided to move forward in our research by implementing the combination of GA with ES in this section for the sixth round of experiment. The idea being that a combination of GA and ES will result in the best of both techniques. The main objective then is to obtain an even higher validation accuracy.

The best configuration for GA and ES was chosen for combination based on the previous rounds of experiments. From the result of fourth round experiment Subsection 3.4, we found that Binary Encoding of GA perform slightly better on comparison with Integer Encoding. Additionally, from the results and conclusion of Subsection 3.5, simple ES performed better than NES for our experiments.

The working concept behind the combination of GA and ES experiment is that for every evolved GA's generation, we collect the best of that generation, and use it as the initial candidate on ES. ES then evolves for a further generation, and if a better solution is found, we replace that, as the new best, on the GA's following generation. Thus, simultaneously we benefit from GA's wide search on the solution space, and ES' dense exploration around GA's best. Moreover, this technique also helps the ES to avoid being trapped on local optima.

With the configured network settings for combined GA and ES architecture, we first ran an experiment with the same hyperparameter settings as configured in the Subsection 3.4. Additionally, to check if we could attain better accuracy with an even higher number of hidden units, we ran second experiment of same architecture with only change of increasing the hyperparameter space neuron values ranging from 500 to 755. The results are in Table 9.

From our results we see that the first experiment produced the network best validation accuracy value we achieved so far at 52.87% (network described in following Section). We believe that this is because GA offers the advantages of searching a wider hyperparameter space, while ES helps fine tune a search in a local area by following a gradient. Of note, increasing the number of hidden units did not allow for better accuracy. In the long version of our results we saw that networks with relatively higher values of hidden units had similar or lower accuracies than our best network with 527 hidden units per layer. This is important as it indicates that just adding more hidden units would not make our results any better, and that ES and GA had found an optima in network architectures.

Hidden units (GA + ES)	Best Validation accuracy
384 - 640	52.87%
500 - 755	52.47%

Table 9. Best validation accuracy achieved with respective ranges of hidden units for GA + ES architecture for 10 generations and population size of 100.

3.7. Seventh Round

In our final experiment, we trained the best network architecture from all of our experiments on the training set and then evaluated it on the test set. The idea being that the performance on the test set would give an idea of the network's ability to generalize onto unseen data. As the purpose of GA and ES in hyperparameter optimization is to find the best settings of hyperparameters, we needed to actually give a test of this performance. The best network came from the first experiment of the combination of ES and GA. The network consisted of 7 hidden layers, 527 hidden units per layer, Leaky-ReLU, and SGD. The network achieved a 51.42% accuracy on the test set, while it had achieved a 52.88% accuracy on the validation set. As this is a relatively small difference between validation and test accuracy, we propose this as a suitable baseline for the OMNIGLOT dataset for standard fully connected neural networks without additional modifications like Batch Normalization or Dropout.

4. Related Work

The field of hyperparameter optimization continues to expand as it directly impacts the performance of increasingly complicated neural networks and large hyperparameter spaces make efficiently searching the space increasingly important. Many researchers have already looked at how to use various evolutionary algorithms to discover better network architectures. As far back as 2000, researchers were looking at using the GA to train the weights in simple neural networks and then combine these results with backpropagation to achieve better performance (Castillo, 2000). Later work expanded this to hyperparameter optimization, with Neuroevolution of Augmented Topologies (NEAT) being a continually improved and currently used GA-based method to optimize neural network architectures while focusing on keeping the size of the networks small (Stanley & Miikkulainen, 2002). In ES, the paper of (Salimans et al., 2017) revitalized interest in using ES for weight optimiza-

tion. Recent research in 2017 also looked at using ES for hyperparameter optimization (Vidnerová & Neruda, 2017).

For the OMNIGLOT dataset, all of the available literature appears to only focus on using it for one-shot and five-shot learning approaches. The complicated network architectures involving Siamese neural networks and convolutional layers achieve accuracies of around 93% for classification (Vinyals et al., 2016). However, we wanted to investigate how well the GA and ES methods could do on the simpler fully-connected networks as we only had the computational resources for such a task. A survey of the research in this area indicates that no researchers have attempted using fully connected neural networks on this task, nor using GA and ES on the dataset itself. Consequently, our research offers new results in this area. We also provide our own comparison and analysis of the GA and ES algorithms on the OMNIGLOT dataset in the context of hyperparameter optimization.

We believe further testing of the ES algorithms could lead to new insights. First, there are different ES strategies that could result in even better performance. A promising direction is to use Covariance-Matrix Adaption (CMA) ES (Hansen, 2016). In CMA-ES, the covariance matrix of the multivariate distribution varies across generations based on how similar the fitness results from the previous generation were. In this manner, CMA-ES can better search smaller or larger areas of the hyperparameter space based on its confidence that the candidate is evolving in the right direction. The next main avenue of research we propose is to use ES or GA in the context of convolutional networks. Though we did not have the computational resources to explore the hyperparameter space of convolutional networks, such hyperparameter optimization is very important as convolutional networks have the best results on many neural network tasks. Our work and analysis on GA and ES may help provide guidance for such future work.

5. Conclusions

Throughout all of our experiments our overall motivation was to explore hyperparameter optimization using the evolutionary algorithms of ES and GA. In the initial rounds of experimentation we focused on our first objective of examining what different settings of the GA could lead to the best classification accuracy on character recognition. In our first round of experimentation on EMNIST, we saw that a higher population size under GA helped maintain solution diversity in the hyperparameter space. This was important to ensure more thorough exploration of the hyperparameter space. However, we now had to change our focus to the harder OMNIGLOT dataset in this work as we had discovered that MNIST and EMNIST did not seem to be as sensitive to the parameters of the GA and hyperparameter optimization of the GA. Our initial experiments on OMNIGLOT showed us the difficulty in finding the appropriate hyperparameter ranges for GA under our implementation. Setting the bounds correctly turned out to be critical to getting GA to achieve its better accuracy. Additionally, later experimentation with GA (in the fifth round) showed us that binary encoding can also help achieve slightly better accuracy. The Binary Encoding provided more opportunities for the Crossover Process and binary mutation to experiment with different network architectures and might have therefore helped with finding better ones.

We then moved onto our second main objective which was examining how ES performs in comparison to GA on finding network architectures. We found that ES initially outperformed GA, and did even better when we decreased population size and allowed it more generations to experiment. We theorize that this is because ES needs more generations for the candidate solution to adapt in the direction of each locally estimated gradient. Additionally, we saw that we had not given GA the opportunity to have such a high number of hidden units as the ES eventually used, so adjusting the initial hidden unit range for GA (and using binary encoding) resulted in GA performing slightly better than ES. However, as both evolutionary algorithms came to comparable accuracy, we conclude that they both can be used effectively for hyperparameter optimization.

In a comparison of the two search techniques, we see that ES and GA both have their own advantages and drawbacks. In GA, it is necessary to find and set initial bounds on the range of the hyperparameters, which takes both time and experimentation. Allowing mutation outside the range may slightly mitigate the problem, but the randomness of mutation does not ensure a thorough exploration of hyperparameter space. In contrast, simple ES and NES both work under the principle of calculating and following the estimated gradient in the hyperparameter space. With simple ES, the gradient is assumed to be in the direction of the best performing network. In NES, the gradient is estimated by evaluating the performance of all sampled networks. Thus, both versions of ES can logically follow a gradient in hyperparameter space and do not require special initialization to find an optima. However, ES still risks entrapment in a local optima, as a bad initialization of the ES can cause a candidate solution to follow a gradient into a suboptimal architecture. Consequently, the GA over a wider area in hyperparameter space may avoid such a suboptimal result.

In light of this analysis, we tried combining both the ES and the GA to harness the strengths of each. In the penultimate round of experiments, the combination of ES and GA did actually find the best network architecture. We then used this architecture on the test set in the last experiment because although we focused on hyperparameter optimization, the underlying purpose of such hyperparameter optimization is to find the best neural network architecture in the first place. The network achieved relatively similar accuracy on test set compared to the validation set, so the hyperparameter optimization seems to have found a solution that can generalize to unseen data. Overall, in our experiments we observed that though ES and GA have different strengths and drawbacks, they are both viable hyperparameter optimization choices for neural networks.

References

Carr, Jenna. An Introduction to Genetic Algorithms. 2014. URL https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf.

Castillo, P. G-Prop II: Global optimization of multilayer perceptrons using GAs. *Neurocomputing*, 35(1-4):149–163, 2000. ISSN 09252312. doi: 10.1016/S0925-2312(00)00302-7. URL http://linkinghub.elsevier.com/retrieve/pii/S0925231200003027.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Hansen, Nikolaus. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lake, Brenden M, Salakhutdinov, Ruslan, and Tenenbaum, Joshua B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Stanley, Kenneth O. and Miikkulainen, Risto. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Vidnerová, Petra and Neruda, Roman. Evolution Strategies for Deep Neural Network Models Design. *ITAT 2017 Proceedings*, 1885:159–166, 2017.

Vinyals, Oriol, Blundell, Charles, Lillicrap, Tim, Wierstra, Daan, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.

Whitley, Darrell. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, Jun 1994. ISSN 1573-1375. doi: 10.1007/BF00175354. URL https://doi.org/10.1007/BF00175354.