Evolutionary Deep Neural Network

Afsaneh Shams
School of Computing
University of Georgia
Athens, GA
Afsaneh.Shams@uga.edu

Drew Becker
Institute for Artificial Intelligence
University of Georgia
Athens, GA
Andrew.Becker@uga.edu

Kyle Becker
Institute for Artificial Intelligence
University of Georgia
Athens, GA
Kyle.Becker@uga.edu

***Abstract*—As we know these days neural network method is frequently used for different machine-learning problems. Two important factors in designing a neural network architecture are the number of hidden layers, the number of units per each hidden layer, and the way nodes in the former layer are connected to the current layer. So, if the goal is to increase classification accuracy. In this paper, the evolutionary algorithm method is implemented for determining the number of hidden layers, and units. Also, adding dropout, adding layer, removing layer, changing layer size, and activation function for each dense layer that is added are done using the mutation concept. This model is tested on two benchmark datasets, Fashion_MNIST, and MNIST datasets. It seems that the works well on both datasets. During the experiment, each model is compiled 30 times to validate the result statistically and compared to the results achieved using other methods.**

***Index Terms*—Neural Network, Fully-Connected, Dense, Evolutionary Algorithm, MNIST**

I. INTRODUCTION

An artificial neural network is a computational model that approximates a mapping between inputs and outputs. It is inspired by the structure of the human brain, in that it is similarly composed of a network of interconnected neurons that propagate information upon receiving sets of stimuli from neighboring neurons. Figure 1, shows a fully-connected, feed-forward Neural Network .

Training a neural network involves a process that employs the back-propagation and gradient descent algorithms in tandem. As we will be seeing, both of these algorithms make extensive use of calculus. In this tutorial, you will discover how aspects of calculus are applied in neural networks. After completing this tutorial, you will know:

- An artificial neural network is organized into layers of neurons and connections, where the latter are attributed a weight value each.
- Each neuron implements a nonlinear function that maps a set of inputs to an output activation.
- In training a neural network, calculus is used extensively by the back-propagation and gradient descent algorithms [1].

To touch on the evolutionary algorithm concept we can say that, evolutionary Computation is a sub-field of Computational Intelligence, a branch of Machine Learning and Artificial Intelligence. solving optimization problems, designing robots, creating decision trees, tuning data mining algorithms, training neural networks, and tuning hyper-parameters, can be counted as evolutionary computation applications.

Special thanks to Dr. Khaled Rasheed and Dr. Frederick Maier for their support and their cooperation to use AI machines.

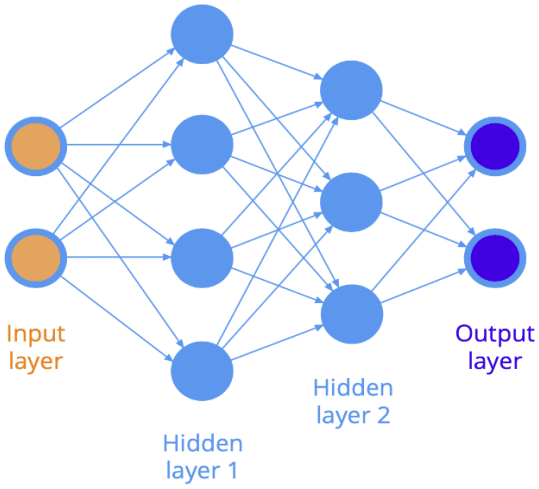

Fig. 1. A fully-connected, feed-forward Neural Network [1].

As Spears believes [2], evolutionary computation uses computational models of evolutionary processes to design and implement computer-based problem-solving systems. There are a variety of evolutionary computation models which are referred to as evolutionary algorithms. Genetic algorithms can be counted as one of the evolutionary methodologies as well as evolutionary programming and evolution strategies.

From Shapiro's [3] point view, to find a solution in ML we have to either create the solution using data like what happens in Decision tree induction by ID3 and nearest-neighbor classification or we have some preexisting solutions then by searching among them we may find the best one like applying the gradient-descent algorithm on a neural network to find the best weights for each layer. GA is a stochastic and uncontrolled search method on the basis of blind search rather than any information and assumption. However, the Evolutionary algorithm is approved to be very effective in solving problems based on experiments and conducted research. At this point let's count some Evolutionary algorithm advantages over traditional algorithms. Traditional algorithms use only

one set of solutions while Evolutionary algorithm uses several sets of solutions as their search space. Search space is a subset of all possible solutions. Also, traditional algorithms need more information to perform and can only generate one result in the end, whereas the Evolutionary algorithm fundamental is the fitness function and can create multiple optimal solutions from different generations. On the other hand, an Evolutionary algorithm is not efficient to solve simple problems and there is no guarantee for the solution of a problem computational challenges may happen due to repetitive calculation of fitness function.

II. ARCHITECTURE

The neural networks were built using TensorFlow and implemented in Python. The structure of the nets was very simple to allow for easy manipulation of the architecture through the genetic algorithm. Every neural network produced included an input layer (Flatten layer, size 28x28 to account for the size of the input data) and an output layer (dense layer, size 10). Since this project is building deep neural networks, resulting networks may also include hidden layers between the input and output layers. These layers are all dense layers, although they have varying sizes with the smallest allowed size as 1 node and the largest size as 128 nodes. Each layer also includes an activation function. We allowed four types of activation functions for the networks, relu, tanh, softmax, and sigmoid. Although other layers, such as dropout, resize, etc are allowed through the TensorFlow library, we restricted our hidden layers to include only Dense layers for ease of manipulation.

III. Optimization Procedure

For this paper, we used a genetic algorithm that is very similar to evolution programming to build an optimized neural network to solve image classification problems. In this section, we discuss the architecture of the neural networks that were built by the algorithm. In this section, we will discuss the genetic algorithm used for optimization.

A. Representation

Each individual is represented in the algorithm as a list containing one fully completed neural network. Representing individuals as lists containing neural networks allowed for easier manipulation of the individuals using the DEAP evolutionary computation package for Python. Regardless, since individuals were directly represented as neural networks, and not encodings for architecture, we applied new mutation methods that acted on the networks themselves (see further sections. Each individual was trained on the dataset and compiled upon creation in preparation for the evaluation step. The population size for these runs was a population of 1, and every population produced four children, achieved by applying the mutation operators to the individual in the population.

Seeding Initial Population. The initial population for this algorithm is seeded with an empty dense neural network, that is, a neural network with only an input layer and an output layer. This allowed us to essentially "start from scratch" when building the neural networks.

B. Fitness Function

Each model was trained on training data generated from the Fashion-MNIST dataset, using 2 epochs during the run of the genetic algorithm. After being trained on the training data, each network was evaluated using the testing data, and the fitness score was equal to the test accuracy of the network. The fitness function is to maximize the accuracy of the network.

At the end of each run, the best individual was trained on the training data for 10 epochs, and then evaluated on the testing data. The model's accuracy on this evaluation is the final evaluation of the best individual, and this is the value reported in the results.

C. Mutation

Rather than relying on crossover between individuals to create genetic variation in our populations, we instead rely on a selection of mutations that are designed to change the architecture of the neural net. Each of the 4 mutations are applied to the parent to create 4 offspring, which then compete to be the parent of the next generation. The mutations we selected are:

- **Add a hidden layer.** This mutation adds a hidden layer just before the output layer of a random number of nodes sampled from the range [0,128]. The activation function of this layer is randomly selected between relu, tanh, softmax, and sigmoid. In order to handle bloating (i.e., the algorithm will continue to select larger and larger neural networks over time), we cap the size of the neural networks to 3 hidden layers. Keeping the networks relatively small helped us to avoid set limitations on computational complexity and also helped us to evaluate the usefulness of each mutation without the algorithm trending toward larger and larger networks. If the network already has 3 hidden layers, this mutation instead randomly chooses between changing the activation function or the size of the selected layer.
- **Remove a hidden layer.** This mutation removes a random hidden layer from the model. The main purpose of this mutation is to allow the algorithm to explore space of potentially smaller networks with different size relationships between subsequent layers, helping to avoid the assumption that larger networks will perform better than smaller ones, regardless of internal architecture. It also may help to remove layers that are maladapted in terms of both size and activation function in one mutation, rather than relying on a separate mutations to change these two attributes. If the network is already empty (i.e. has no hidden layers), this mutation instead adds a hidden layer to avoid mutations that do not change the individual.
- **Change size of a hidden layer.** This mutation changes the number of nodes present in a random hidden layer in the network to a number randomly sampled in the range [0,128]. Mutating the size of the hidden layers is likely

the main cause for genetic variation in networks that will most likely be the same size after multiple rounds of selection. Changing the size of a random hidden layer rather than always on the last hidden layer allows the evolutionary algorithm to design the neural architecture in a non sequential way and allows it to explore a greater solution space of potential architectures. It makes no assumptions about the progression of layer size across the network (i.e., each layer being larger/smaller than the last layer), and thus may help to design neural architectures without the limitations of human assumptions behind architecture design.

- **Change activation function of a hidden layer.** This mutation changes the activation function of a hidden layer to one randomly chosen between relu, tanh, softmax, and sigmoid. Selection of neural network activation functions are largely based on empirical data, with theory behind what is suitable for particular architectures lagging behind the empirical research [6]. Because of this, we wanted our algorithm to explore the effects of varying activation functions within the hidden layers to produce different architectures. The mutation makes no assumptions about what functions or combinations of functions will be suitable for the classification problem at hand, and so the algorithm is free to explore a wide solution space of many architectures of various combinations of activation functions.

D. Selection Scheme

The overall selection scheme we decided on was a $(\mu + \lambda)$ tournament selection. In practice, in order to reduce computational complexity, we ran our algorithm with a parent population size of 1, resulting in $(1 + 4)$ selection with a tournament size of 4. By making the tournament size slightly smaller than the number of individuals that will be competing, we hope increase selection pressure and have the algorithm overall favor the individual with the best fitness. By using an elitist plus selection strategy, we ensure that the best individual will be retained even if it is the parent of the current generation.

E. Termination Criteria

Due to the computational complexity of NAS algorithms and the time it takes to train neural networks, we decided to keep our termination criteria short. We ran each instance of our evolutionary algorithm for 50 generations, then after 50 generations produced the individual with the best fitness score. Future work that allows for more time to run the algorithms might increase this number to allow for a more thorough exploration of the solution space.

IV. DATASET

A. Fashion_MNIST Dataset

The Fashion_MNIST dataset is a large freely available database of fashion images that is commonly used for training and testing various machine learning systems. Fashion_MNIST was intended to serve as a replacement for the original MNIST database for benchmarking machine learning algorithms, as it shares the same image size, data format and the structure of training and testing splits.

Fashion_MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

- Homepage: https://github.com/zalandoresearch/fashion-mnist
- Source code: tdfs.image_classification.FashionMNIST.
- Versions: 3.0.1 (default): No release notes.
- Download size: 29.45 MiB.
- Dataset size: 36.42 MiB.

TABLE I
FEATURE STRUCTURE FOR FASHION_MNIST DATASET [4].

Feature	Shape	Dtype	Number of Classes
image	(28, 28, 1)	uint8	-
label	-	int64	10

In the following figure we can see the labels example:

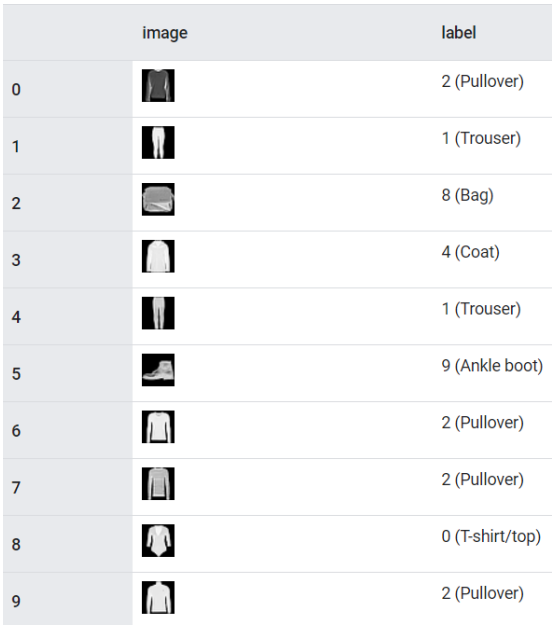

Fig. 2. Fashion_MNIST label example. [4].

B. MNIST Dataset

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high

school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

- Homepage: https://github.com/zalandoresearch/fashion-mnist
- Source code: tdfs.image_classification.MNIST.
- Versions: 3.0.1 (default): No release notes.
- Download size: 11.06 MiB.
- Dataset size: 21.00 MiB.

TABLE II
FEATURE STRUCTURE FOR MNIST DATASET [5].

Feature	Shape	Dtype	Number of Classes
image	(28, 28, 1)	uint8	-
label	-	int64	10

In the following figure we can see the labels example:

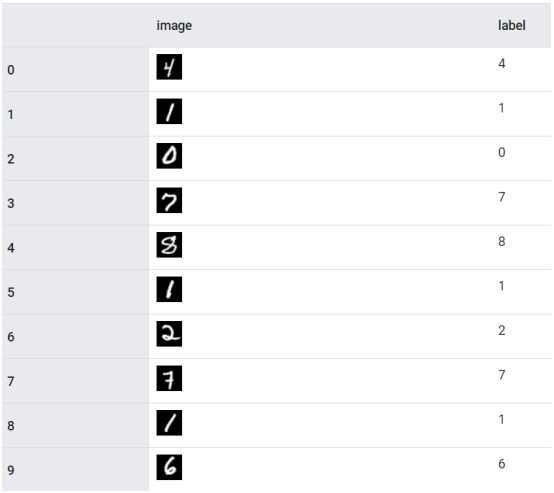

Fig. 3. MNIST label example. [5].

V. EXPERIMENTS

A. Method

The evolutionary algorithm was run 30 times for 50 generations for each data set. Each individual neural network was trained for a small number of epochs (10) before being assigned a fitness score; therefore, the fitness scores of each individual are used as a heuristic for greater classification accuracy at a higher number of training epochs once the architecture is generated. The accuracy and architecture for the most fit individual at the end of each run were saved and recorded. These individuals were compared based on their accuracy scores to a set of N randomly-generated neural networks to assess the effectiveness of the evolutionary algorithm over the chance performance of random DNNs.

To do these experiments two machines of the Artificial Intelligence department, UGA, are used. The first one is aigpc7 with IP address 172.22.147.6. This is benefited from Intel i9 10900X CPU, Quadro RTX 5000 GPU, and 96GB DDR4 RAM. The second machine is aigpc9 with IP address 172.22.147.100. Some features of this machine are Intel i9 10900K CPU, GeForce GTX 1660 Sup GPU, and 16GB DDR4 RAM. For this research, we used the CPU of these machines rather than their GPU. So for future work, the model can be run on these systems using their GPU and then compare the total time taken to get each result. We can say in almost all cases of these 60 experiments it took between one hour to one hour and twenty minutes for the system to compile the code. This time was almost the same for both experiments on Fashion_MNIST and MNIST datasets.

B. Fashion_MNIST result

In this part, we see the results for Fashion_MNIST dataset. We started the experiment first with a small number of generations. So we set the number of generations to 10 at first. In the following, the result for 10 generations is shown. For this case, the accuracy is 0.87559998035.

Layer (type)	Output Shape	Param #
flatten_24 (Flatten)	(None, 784)	0
dense_100 (Dense)	(None, 50)	39250
dense_101 (Dense)	(None, 106)	5406
dense_123 (Dense)	(None, 59)	6313
dense_124 (Dense)	(None, 10)	600

Total params: 51,569
Trainable params: 51,569
Non-trainable params: 0

Fig. 4. Result of the model using 10 generations on Fashion_MNIST dataset.

We tried to run the model on this dataset for 100 generations. We reached to an accuracy equal to 0.885399997234. Here you can see its result:

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_16 (Dense)	(None, 99)	77715
dense_24 (Dense)	(None, 38)	3800
dense_34 (Dense)	(None, 48)	1872
dense_35 (Dense)	(None, 10)	490

Total params: 83,877
Trainable params: 83,877
Non-trainable params: 0

Fig. 5. Result of the model using 100 generations on Fashion_MNIST dataset.

We decided to do the experiment using a higher generation value so we select to set the number of generations to 50 and perform the experiment 30 times to give static validity to this research. The outcome for those 30 runs using 50 generations on the Fashion_MNIST dataset is shown below.

Experiment No.	Total Parameters	Accuracy
1	55,973	0.888000011
2	75,789	0.877600014
3	98,426	0.894400001
4	103,027	0.879999995
5	67,414	0.886699975
6	63,927	0.886300027
7	113,631	0.88349998
8	109,007	0.877799988
9	91,013	0.882700026
10	82,592	0.884199977
11	87,111	0.884800017
12	84,455	0.881699979
13	108,667	0.880400002
14	107,930	0.879899979
15	108,656	0.886500001
16	61,307	0.881200016
17	83,459	0.879899979
18	115,307	0.880500019
19	76,910	0.884800017
20	82,481	0.877600014
21	104,165	0.878000021
22	48,514	0.880100012
23	104,336	0.885900021
24	65,995	0.881500006
25	90,311	0.883300006
26	98,426	0.879000008
27	98,423	0.885699987
28	91,207	0.879100025
29	98,432	0.887799978
30	120,499	0.885299981

Fig. 6. Results of 30 experiments done on Fashion_MNIST dataset.

To compare the results for the case we did 50 generations, with the output for 50 and 100 generations we can say there is a difference among them in the case that we used 100 generations the result is the best and in the situation that we used 10 generations, we have the lowest accuracy in this comparison. Of course, it should be taken into consideration that it is just the result for one run for both 10 and 100 generations, and hence it may not be statistically guaranteed.

As figure 6 shows, among 30 experiments done using Fashion_MNIST dataset, the third experiment has the best/highest accuracy (equal to 0.894400000572204). Also, note that in this

No. of generations	Total Parameters	Accuracy
10	51,569	0.87559998
50	75,789	0.877600014
100	83,877	0.885399997

Fig. 7. Comparison of 10, 50 and 100 number of generations..

figure, figure 7 and figure 8 the value for accuracy shows only 9 digits after decimal point in order to prevent more complex and small value. Hence if you compare rows 2 and 20 you may find same accuracy but different number of parameters. The reason is that actually these two values for the accuracy are not the same and if we would write all digits.

C. MNIST result

In this part the model is used to classify the MNIST dataset. Again here we did 30 experiments to give validity to the output of this research. For this dataset, we did not do the experiment for cases where we have 10 and 100 number of generations, as we did it for the Fashion_MNIST dataset it is expected that we reach the same result here again. Hence we allocated all of our resources to perform these thirty experiments and then compare it with the accuracy obtained by other models designed by others. The results for this session are summarized in figure 8.

By comparing these 30 experiments with each other we figure out that all in all experiments we reach good values for accuracy so that the lowest value for the accuracy is related to experiment number 28 where the accuracy is 0.970399975776672. This is when the experiment number 20 has the best accuracy (equal to 0.980499982833862). So this experiment confirms that this model works almost well for classifying MNIST dataset.

D. Comparison

In this part, the goal is to compare performance of our model for Fashion_MNISt and MNIST datasets. In addition, by referring to other works done to classify these two dataset we will see if our model is compatible.

To compare the output for these two datasets, it is visible that for Fashion_MNIST dataset the value for the accuracy is in range of 0.877600014209747 to 0.894400000572204. And the accuracy range for MNIST datset is 0.970399975776672 to 0.980499982833862. So, we can say the model perform almost well in classifying both datasets. However, it outperforms when the MNIST dataset is used.

In ”an analysis of Convolutional Neural Network for Fashion images classification (Fashion-MNIST)” [7], Different deep neural network models are implemented and compared using Fashin _ MNIST dataset. So as they have mentioned in tbhis paper, After 40000 iterations, AlexNet, GoogleNet, VGG, ResNet20, ResNet32, ResNet44, DenseNet, SqueezeNet and SqueezeNet with batch normalization gives an accuracy of 88.50%, 88.75%, 90.28%, 90.62%, 90.80%, 91.40%, 90.37%,

Experiment No.	Total Parameters	Accuracy
1	73,085	0.975600004
2	90,377	0.976199985
3	70,150	0.977299988
4	115,386	0.978999972
5	103,468	0.973200023
6	53,329	0.97359997
7	119,057	0.978999972
8	102,899	0.973999977
9	91,178	0.979099989
10	89,044	0.978799999
11	101,844	0.979600012
12	100,727	0.976800025
13	73,430	0.976499975
14	116,423	0.978600025
15	87,839	0.979499996
16	89,537	0.977900028
17	68,069	0.975000024
18	79,352	0.976199985
19	78,007	0.976100028
20	120,829	0.980499983
21	108,320	0.978299975
22	109,394	0.980199993
23	85,205	0.971000016
24	51,207	0.974799991
25	115,581	0.978900015
26	112,645	0.978100002
27	108,230	0.978100002
28	117,905	0.970399976
29	85,345	0.975000024
30	115,715	0.976899981

Fig. 8. Results of 30 experiments done on MNIST dataset.

88.93% and 92.56% respectively. In comparison to this study our model performs almost the same as AlexNet, GoogleNet, SqueezeNet and SqueezeNet.

To compare our model's performance with others, we can refer to "comparisons of deep learning algorithms for MNIST in real-time environment" [8] where the accuracy of Regression model, CNN, ResNet and CapsNet are compared with each other based on their accuracy in classifying MNIST dataset samples. Based on the outputs of this paper the accuracy values for Regression model, CNN, ResNet and CapsNet are equal to 92.1%, 98.1%, 97.3% and 99.4% respectively. Comparing our model with these deep learning models we can come to the conclusion that our model outperforms Regression model and its performance is compatible with CNN, ResNet models.

VI. FUTURE WORK

For future work, we would like to see an experiment similar to the one we proposed, but with larger populations of neural networks to improve performance. This may become more feasible in the future, if researchers are able to build models for predicting the success of a neural network to use as a surrogate function to avoid long runtimes. Until that technology becomes available, however, the most reasonable way to advance this research would be to run trials over long periods of time with other GPUs. Another way to extend this research would be to experiment with different types of representations, such as representing the network encoded in a stream of digits. This method may provide a more robust evolutionary architecture search than starting from an empty net and building with mutations. It will also be important to gather data from other neural architecture searches and compare them against our evolutionary architecture search.

VII. CONCLUSION

This paper introduced an evolutionary algorithm for the purposes of deep neural net architecture design. Our aim was to see if an evolutionary algorithm could use mutation to create a deep neural network that would perform as well or better on classifying images from the benchmark MNIST and Fashion_MNIST than a human-designed neural networks.

As it is shown in the experiment part, we could reach the accuracy equal to 0.894400000572204 for Fashion_MNIST dataset and the maximum value for accuracy for our model when we used MNIST datset is equal to 0.980499982833862. We also showed the result for 30 experiments for each datset in order to guarantee our model performance statistically. Also we compared the model performance when we applied 10 generations, 50 generations and 100 generations to the model using Fashion_MNIST dataset. And it is shown that when the model is implemented based on 100 generations its performance is better. However due to some resources constraints we did all of our experiments with 50 generations. At last we show that the proposed model outperform some deep learning models both for Fashion _ MNIST and MNIST datasets and it is comparatible with some other models like AlexNet, GoogleNet, SqueezeNet and SqueezeNet models for Fashion _ MNIST dataset and like CNN, ResNet models for MNIST dataset.

VIII. STUDENTS CONTRIBUTION

A. *Kyle Becker*

Kyle designed the overall framework for the problem and evolutionary algorithm, including individual representation, mutations, and selection strategy. He helped decide on testing the resulting DNNs on the MNIST and Fashion_MNIST data sets. Throughout the development of the algorithm, he assisted

Drew in planning changes to individual representation and mutation implementation according to the problems that arose during testing. He also decided on the representations of the initial population, and helped determine the final algorithm procedure of mutating the single surviving individual according to all 4 mutations to create separate offspring before survivor evaluation.

B. Drew Becker

Drew worked on development and implementation of algorithm. He coded the algorithm in Python using the DEAP and Tensor Flow machine learning packages and helped coordinate communication between all group members. He also contributed general problem solving and bloat-reducing methods. Drew also helped run experiments on the AI machines and assisted in writing the report.

C. Afsaneh Shams

Afsaneh did coordination to get access to AI machines. Also, she did the troubleshooting and error handling for the code when we tried to run the code. I installed conda on the server number aigpc7, then CUDA version 11.2 and TensorFlow-GPU version 2.11 were installed (But unfortunately as I did not have the admin access control we preferred to use system CPU instead of GPU). Also, I edited the code for MNIST dataset. Afsaneh did 54 of these experiments and comparing them to each other and former works is what she did.

REFERENCES

[1] https://machinelearningmastery.com/calculus-in-action-neural-networks/

[2] SPEARS, W. M., JONG, K. A. D., BACK ¨ , T., FOGEL, D. B., AND DE GARIS, H. 1993. An overview of evolutionary computation. In Proceedings of the European Conference on Machine Learning (ECML93), P. B. Brazdil, Ed. Vol. 667. Springer Verlag, Vienna, Austria, 442–459.

[3] J.Shapiro: Genetic algorithms in machine learning. Lecture notes in computer science (Vol. 2049/2001, pp. 146–168). (2001)

[4] https://www.tensorflow.org/datasets/catalog/fashion _ mnist

[5] https://www.tensorflow.org/datasets/catalog/mnist

[6] Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems, 1*(4), 111-122.

[7] K. Meshkini, J. Platos, and H. Ghassemain, "An analysis of convolutional neural network for fashion images classification (Fashion-MNIST)," in Proc. Int. Conf. Intell. Inf. Technol. Ind. Cham, Switzerland: Springer 2019, pp. 85–95.

[8] Akmaljon Palvanov, and Young Im Cho, "Comparisons of Deep Learning Algorithms for MNIST in Real-Time Environment," International Journal of Fuzzy Logic and Intelligent Systems, vol. 18, No. 2, pp. 126-134, June 2018.