Optimization of Neural Networks using Deep Genetic Network Algorithm

Siddhartha Dhar Choudhury, Kunal Mehrotra, Shashank Pandey, Christhu raj, Rajeev Sukumaran

Abstract*: *The optimization of performance of a neural network is a time taking and tedious process, this iterative and continuous process has no definite solution that works well for every possible use case. To tackle this problem we propose an architecture of neural networks called "Deep Genetic Network", which can help in automatic selection of hyper parameter values based on fitness measures during training of the network. The algorithm is a confluence of deep neural networks and genetic algorithm. The problem of optimizing a neural network can be classified into - Architecture and Hyperparameter optimization. A variety of algorithms have been proposed to solve this issue. Our approach uses concepts of mutation and mating (from genetic algorithms) for helping the neural net in finding the optimal set of hyperparameter values during training without requiring any manually setting the values in an iterative trial and error approach. The architecture that we propose here works well in optimization of hyperparameter values in convolutional, recurrent and affine layers. The usage of genetic algorithms for resolving this issue has worked well given adequate training time and computational resources.

Keywords: Hyperparameter optimization, Neural Networks, Neural Network Optimization, Genetic Algorithms

I. INTRODUCTION

Optimizing a neural network is an important step in the field of deep learning, it plays a major role in achieving high accuracy in the trained model. The task mentioned here can be divided into two categories - optimizing the neural net architecture or the hyperparameters used.

The optimization of a neural network's architecture involves two different tasks - finding the optimal node count in a particular layer and optimal count of layers in the network. This problem does not have any fixed formulae or static value which can work in every problem, and for finding the best value we have to approach this using a continuous trial-and-error method in which different values are tried out until the best values are found. There are various algorithms that are currently used for solving this issue, but this is not addressed by the algorithm that we are proposing in this particular paper.

The second type of problem in optimizing neural networks is that of finding the best values for different hyperparameters such as - rate of learning, optimizer used, rate of dropout, size of each batch and so on. The solution to this problem is again based on a continuous trial-and-error method, as there is no static value which can work for all use cases. The architecture that we introduce in this paper helps in solving this problem efficiently as it leverages the power of genetic algorithms to automate the process of selecting hyperparameter values. The usage of neural net helps in solving varied problems and can even work better than a human in some cases, this is due to its ability to handle large amounts of data in an efficient manner and in a short span of time. Their ability to form highly precise relation from given data is what makes them better as humans cannot fail to identify these. This led us to think, why should we perform the iterative method of finding the best set of hyperparameters when we can assign this task to the neural network itself. By further developing on this base idea, we propose Deep Genetic Network (DGN) for automatic selection of hyperparameters when a neural network is given a pool of different hyperparameter values thus removing the need for applying the continuous trial-and-error approach.

DGN makes use of a technique called genetic algorithm in combination with neural nets for getting the best set of hyperparameter values during training. The training process begins with 'n' neural networks ($n > 1$), and proceeds to find the fittest networks from pairs of networks after training for a set number of epochs. The time required for training multiple networks in a parallel fashion requires much more time than for a single network, but this method is still faster than performing the continuous trial and error method for selection of hyperparameter.

The proposed algorithm makes use of a technique called mating (or combination of genes of parents) for finding the fittest possible hyperparameters after every generation of training, during this process the more fit (one whose dominance is more) is allowed to pass greater number of parameters to the next generation of children neural networks as compared to the recessive (less fit or parent whose dominance is less) parent. The idea of mating is taken from genetic algorithms, this aids the passage of more fit parameters to the next generation and hinders the passage of less fit parameters to subsequent children networks. Thus allowing only the best set of parameters to be passed to later generations this resembles Darwin's "Theory of Natural Selection" or the principle of "Survival of the fittest", where only the most fit genes are allowed to survive in any environment.

From the different experimentations we prove that the proposed algorithm or network architecture aids in automatic selection and optimization of hyperparameter values thus

Revised Manuscript Received on October 30, 2019.
***** Correspondence Author
Siddhartha dhar Choudry, Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India. siddhathadhar_soumen@srmuniv.edu.in
Kunal Mehrotra, Shashank Pandey, Christhu Raj, Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India. mrchristhuraj@gmail.com
Rajeev Sukumaran, Teaching Learning Centre, Indian Institute of Technology Madras, Chennai, India. rajeev.s@wmail.iitm,ac.in

allowing building of neural networks much robustly rather than applying the iterative trial and error approach of selecting hyperparameter values.

In our experimentations we propose four types of schemes of mating - extreme-ends, adjacent-ranked, cousin to cousin, and sibling to sibling. These schemes are only applicable when the number of parent networks in the inception generation are more than or equal to two. Choosing a particular scheme depends on various factors as will be evident from later sections.

II. RELATED WORK

There are a variety of different works done for solving the problem of optimizing hyperparameter values, some of the most notable algorithms are presented below:

A. Random Search

The random search [2] algorithm derives the idea from Grid Search, the difference in this algorithm is that this does not search for the fittest value from the population of hyperparameters, rather the values are sampled from a random distribution. Thus the process aids in reduction of time that is needed for finding best set of hyperparameters, this makes this algorithm much more efficient than grid search.

This algorithm begins execution by defining a random distribution of hyperparameters, next the number of training steps are set for which the algorithm will sample values from the defined distribution. Since in most use cases, the values of hyperparameters are not of equal importance and sampling their values from a randomly initialized distribution population can help in solving the issue of choosing the best hyperparameter values, this performs much better than grid search form most problems.

B. Meta Learning

The meta learning [3] algorithms are given a huge variety of tasks during training and their generalization capabilities are tested on unseen tasks. For instance, a particular task may be trained to classify an image into five different classes, can learn optimal navigation within a entirely new maze just after a single iteration in the entire maze. The difference between this and general machine learning algorithms is that the latter are given only one task for training and are tested on examples other than the training data points from the dataset.

The meta training stage involves training on tasks from the meta training set. In this there are two optimizers - meta learning who aids the learners in training and learner who is actually responsible for learning new task. The methodologies for meta learning fall into these categories: metric learning, learning optimizers, and recurrent modes.

C. Grid Search

In the grid search [1] algorithm hyperparameter values are tuned by evaluation and building of every possible hyperparameter set present in a pool called grid. In this approach a pool of hyperparameter values are specified which are optimized for the use case that they are applied to. A searching function then combines different hyperparameter values that are specified in the population of available values in the grid.

The algorithm keeps a copy of the most fit set of hyperparameter as a set is processed from the grid, if one provides with better result than the previous sets then this is set as the best possible option out of the scanned possibilities. This continues to update the best values for hyperparameter as the scanning continues.

This algorithm thus continuously chooses the best set of hyperparameter values and the network is retrained repeatedly until the best set of values are found, this makes the approach extremely time intensive and tough to train without powerful hardware resources.

III. GENETIC ALGORITHMS

Genetic algorithm [4] belongs to the class of evolutionary algorithms which helps in solving problems related to optimization. This draws inspiration from Darwin's "Theory of Natural Selection" and borrows the concept of evolution [5, 8] which is applied in the context of programming.

The algorithm begins processing with an inception or first generation species population (which are deep neural networks in our case), these participants vary from each other. These individuals from the population are then allowed to mate and generate children who consist of the mixture of genes from both the parents involved, and in some cases can be better than the previous generation in terms of fitness. There are two important features in this algorithm: Mutation and Mating.

A. Mating

The process of producing children (one or more than one child) by parents is referred to as mating. The children produced exhibit characteristics that resemble both parents. Genes from the parents are passed onto the child networks, these genes help in expressing characteristic traits of the parents that the children inherit. These parents are of two types - recessive and dominant, the dominant parent has more chances of expressing genes in the child than the recessive parent.

In the algorithm we propose here, biases and weights are the representation of genes that a neural network possesses, these are passed to the children networks which are formed by combination of parent's genes. These child neural networks have the best set of parameters (with a greater density of dominant genes), and are an improvement over the parent generation of networks. The dominant parent passes most genes to child networks, as these are more fit and can survive in subsequent generations.

B. Mutation

The variations in child neural networks that were previously not present in parent generations is referred to as mutation. Mating helps in production of children, and this process passes forward the most fit set of parameters (or genes) from both parents involved in the process, but mutation is responsible for making a greater positive change towards efficiency in the children. These changes might be minute or drastic, and leads to a wider variety among a generation of children.

In context of biology this is what helped in formation of complex organisms from the single cells that were present in the beginning of time on Earth. The process of mutation has a key role in the evolutionary process, these variations make the child more strong and gives resistance to external dangers that the previous generation was susceptible to.
In the case of this algorithm, mutation takes place in the training iterations for which a network is allowed to train before mating process, during this process the parent networks update their parameters continuously using backward propagation (gradient descent) until they are ready to give rise to child neural networks in which a mixture of genes from two parents is passed on. These mutations help the parent networks learn and pass their learnt features to the next generation, and this is what makes the current generation more efficient and a better choice than the previous generation networks. These networks are thus in a way better version of their parents.
Both of the above explained processes have a key role in genetic algorithms and in the proposed algorithm, these help the neural networks in choosing the best set of hyperparameter values on their own.

IV. UNDERSTANDING DEEP GENETIC NETWORK

Deep Genetic Network's process of learning begins by creating a sample collection, that consists of many sets of hyperparameter values (each associated to a neural network) - a pool of networks with unique hyperparameters. As the process of training advances, the various neural networks continuously update the values of their parameters (weights and biases) by back propagating error using gradient descent. After a fixed number of iterations of training (epochs), neural networks are selected for mating in pairs to generate child neural networks. These child neural networks carry the fittest set of parameters from the previous generation. These then undergo mutation for a certain number of training iterations then are ready to produce children neural networks of their own. These strategies (called mutation and mating) are taken from genetic algorithms [6, 9].

A. Mating

The idea of mating is to combine two neural network parents to generate neural network children. Parent networks share their gene to the child network, the child thus has features from both the parent networks. In this case, the biases and weights of the parent network can be considered to as genes which are inherited by the children in subsequent generations.
This process of mating is done in every 'n' iterations. The inception generation (very first) parent neural networks get 'n' iterations of training, further they are ranked according to their fitness (the efficiency with which they can minimize loss). Child neural networks that are generated after this process can be of two different types - 60:40 and 80:20. These ratios signify the percentage of genes inherited by the dominant (greater efficiency) and recessive (lesser efficiency) parent, for instance the 80 to 20 neural network child has 80 percent gene parameters inherited from the parent neural network which is more dominant and rest belong to the recessive neural network parent, this holds true for the 60:40 child neural network.
On the basis of the count of inception parents, the algorithm can be classified into two types:

1. Network of double inception parents: This type of network has only two parents in the very first or inception generation, and in the subsequent child network generations. After mating the pair of parents give rise to two neural network child, this occurs in subsequent "n" training epochs. In the final "n" training epochs, the mating process generates only one network, this produces output that the network is expected to produce.

2. Network of double-N inception parents: The inception parent generation has a total of "2 times N" neural networks, here $N \geq 1$. In the case when N = 1, this will resemble the network of double inception parents. As training proceeds and newer generation of child networks evolve, parent networks produce a pairs of child networks. The child neural networks evolve to become parent network and produce their own child networks. As the training reaches last generations, child neural networks start combining exponentially by factor of 2^N, this continues till there is only one network which is responsible for producing the required output.

We propose two different schemes of mating (techniques) which can be used in the Network of **double-N inception parents**:

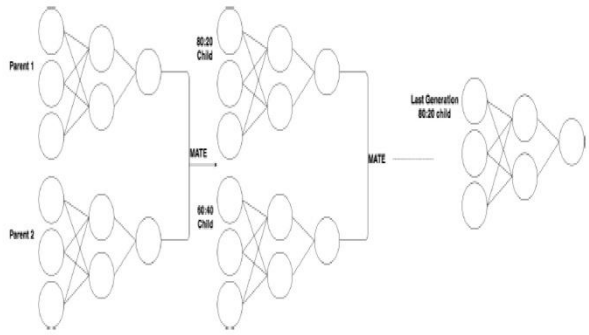

Figure 1. Two Parent Deep Genetic Network

1. Extreme-Ends Mating: This technique or scheme of mating requires the process of mating to occur between networks belonging to extreme ends when they are ranked according to their fitness (greater efficiency) in descending order. For instance, in the ordered list of networks first and last parent mate to produce a pair of children, second and last but one mate together, and so on. This mating scheme mixes together the genes (or parameters) of the fittest parent network with the least fit one. This makes sure, that even the most recessive generation does not get completely suppressed, as these can turn out to be dominant in later generations.

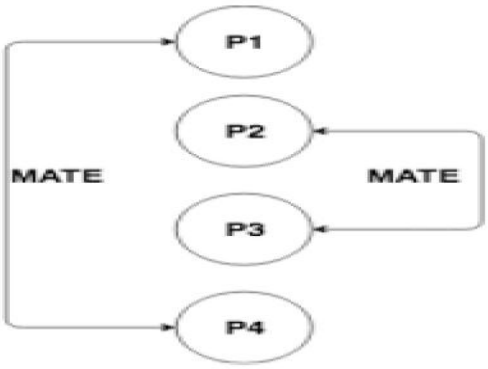

Figure 2. Extreme Mating Scheme

2. Adjacent-Ranked Mating: This scheme of mating allows parents who are adjacent to each other when they are ranked in decreasing order of fitness to mate and produce child networks. This involves parents who are not from extreme ends of fitness ordering. For instance, if we have ten networks in a particular generation of training and they are ordered after "n" iterations according to their fitness, then the first and second in the list of fitness will mate together and so on.

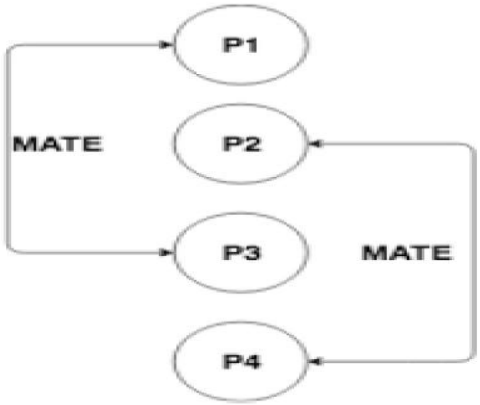

Figure 3. Adjacent Mating Scheme

On the basis of the immediately previous generation of parent networks there are two different categories:- cousin to cousin scheme and sibling to sibling scheme.

1. Cousin to Cousin Mating: This scheme of mating of double-N inception parents involves combination of two parents who did not have the same immediate parent (i.e. they belong to different parent neural networks). Using this scheme helps in choosing more fit set of parameters, as the genes are derived from a diverse parent pool. This thus has more chances of producing stable and efficient child networks, these children have better chances of passing their genes to subsequent generations as parents, due to the varied exploration of parameters in ancestor generations.

2. Sibling to Sibling Mating: This scheme of mating involves combination of parameters in children who are generated from same pair of neural network parents, these children have the best set of parameters from both the parents. In one generation even though a particular set of genes might seem to be recessive, it can later revive as a dominant gene, but this scheme does not allow such parents to pass on their parameters for long.

B. Mutation

The changes in a child neural network that makes it more fit than the parent network is mutation. In this process minor changes in the genes of the child network are done, these tweaks in the parameters help the child network to produce better results than the previous generation. The changes made to the child helps the network to reduce the errors in the parent generation.

Deviating from the technical aspects, the concept of mutation can be best understood from a biological example - the case of peppered moths during industrial revolution. Our focus is on a particular species of moths called 'peppered moths'. Before the beginning of industrial revolution and the spread of factories in cities these moths were black and white in color. This color provided them camouflage in the similar colored barks of trees, thus decreasing their chances of getting in the field of vision of their predators. Due to mutation (changes in genes) a few individuals of this species were completely black in color and were more susceptible to be seen by the predator, thus making this gene recessive. When industrial revolution began, smoke from the factories covered the trees nearby with a layer of ash and now the population with recessive genes were less susceptible than the black and white gene moths, and this led to increase in dominance of the former gene population over the black and white population.

In the case of this algorithm, parent networks combine (or mate) to give rise to child networks, these children get trained for "n" iterations till they themselves are capable of producing children of their own. During these "n" training steps, the child networks continue learning and tweaking their parameters to minimize error, these more trained parameters (or genes) are much better than the previous parent generation, in a way representing mutation in these networks.

For these reasons, the presented neural network architecture helps in finding the fittest set of parameters that are available in the initialized pool of hyperparameters. This automates the process of finding the best set of hyperparameters, which till now is done manually in most cases using a iterative trial and error methodology. This thus requires less supervision than its manual counterpart and can save the time of the person responsible for designing the neural network for solving a particular problem.

V. EXPERIMENTATION

The below mentioned experiments were performed using deep genetic networks to verify their validity in neural network training process:

A. Convolutional Neural Network - Image Classifier

In deep learning, image classification tasks are done using an architecture called convolutional neural network [7, 12]. This experiment relies used this architecture to classify images into dog or not dog. The model we used consisted of five layers (hidden and output), this comprised of - three layers of two dimensional convolution operation and two affine layers (the final layer being the sigmoid layer which produces the output of the network).

The hyperparameters used to train this network were - learning rate (α) and the rate of dropout that is applied to the affine layers. Using iterative trial and error method the best set of values found for these two hyperparameters were - 0.0070 for Learning rate α and 0.7 for rate of dropout after the first affine layer. After training the model the resulting accuracy on the test set of images was 0.85 (or 85\%), after 2600 epochs.
The values obtained were a result of re-training continuously with different sets of hyperparameters, using these values the network achieved minimum loss and high accuracy, but required a lot of efforts to train. In this case the proposed algorithm comes into play.

A1. First Model: Double Inception Parents DGN

In this model two parent networks were used in the inception (or first) generation of training, the hyperparameter set used for these networks are mentioned below:-

1. **First NN Parent:**
    1. Learning Rate (α) = 0.007
    2. Dropout Rate = 0.75
2. **Second NN Parent:**
    1. Learning Rate (α) = 0.0055
    2. Dropout Rate = 0.8

A2. Second Model: Double-N Inception Parents DGN with Extreme-Ranked Scheme of Mating

In this model two parent networks were used in the inception (or first) generation of training, the hyperparameter set used for these networks are mentioned below:-

1. **First NN Parent:**
    1. Learning Rate (α) = 0.007
    2. Dropout Rate = 0.7
2. **Second NN Parent:**
    1. Learning Rate (α) = 0.0065
    2. Dropout Rate = 0.8
3. **Third NN Parent:**
    1. Learning Rate (α) = 0.0055
    2. Dropout Rate = 0.6
4. **Fourth NN Parent:**
    1. Learning Rate (α) = 0.002
    2. Dropout Rate = 0.2

In the model, neural network parents got 50 training steps before the mating process began, during this process the collection of parent networks were arranged in descending order of fitness, this was followed by combining of genes (or mating) to produce children. The scheme of mating used in this case was extreme-ends i.e. parent network 1 combined genes with 4 and network 2 with that of 3. These children were reduced in the final hundred epochs as four children were reduced to two (in the first fifty epochs) and from two to one (in the final fifty epochs). The resulting model gave a test image set accuracy of 0.87 (or 87\%).

A3. Third Model: Double-N Inception Parents DGN with Adjacent-Ranked Scheme of Mating

The last neural network model trained for image classification problem had four neural network parents. The following are the details of hyperparameter pool used:-

1. **First NN Parent:**
    1. Learning Rate (α) = 0.007
    2. Dropout Rate = 0.7
2. **Second NN Parent:**
    1. Learning Rate (α) = 0.0065
    2. Dropout Rate = 0.8
3. **Third NN Parent:**
    1. Learning Rate (α) = 0.0055
    2. Dropout Rate = 0.6
4. **Fourth NN Parent:**
    1. Learning Rate (α) = 0.002
    2. Dropout Rate = 0.2

The trained neural network with Adjacent-Ranked scheme of mating achieved a test image set accuracy of 0.88 (or 88\%).

B. Artificial Neural Network Based Regression Model

When working with numeric or text data (without any temporal coherence) using the basic multilayer perceptron [11, 13] model provides great results. Regression is a task in which continuous data is predicted. In this model the network had three affine layers [10] (single output layer and two layers of hidden units). This network was trained using the adam optimizer.
The three parameters that were involved in the training of the above mentioned neural network were - rate of learning (α, rate of dropout used in the affine layers, and momentum (required in adam optimizer). Using iterative trial-and-error process the ideal set of hyperparameters resulted in a final testing accuracy of 0.94 (or 94%). This model required 4500 training steps and had these set of hyperparameters - 0.06 (α), 0.6 (rate of dropout), and 0.999 (for momentum).
Double-N inception parent DGN performed great in the given problem definition. This increased the accuracy of testing set to 0.97 (or 97\%).

B1. First Model: Double-N Inception Parents DGN

The neural network model had two inception parent neural networks, these were trained for 100 epochs before mating to produce child neural networks (60 to 40 and 80 to 20 children). The initial neural network parents had the following configuration of hyperparameter pool:-

1. **First NN Parent:**
    1. Learning Rate (α) = 0.006
    2. Dropout Rate = 0.6
    3. Momentum = 0.999
2. **Second NN Parent:**
    1. Learning Rate (α) = 0.08
    2. Dropout Rate = 0.7
    3. Momentum = 0.99
    4.

These experiments show how Deep Genetic Networks provide the user with an automatic hyperparameter selection option which saves a lot of time that was previously used in iterative trial-and-error process for finding the best set of hyperparameters.

VI. CONCLUSION

From the experiments it can be established that Deep Genetic Networks help in training and optimization of the hyperparameters automatically without requiring manual supervision and iterative trial and error process of training. This process is done in a parallel fashion by using concepts like mating and mutation. The network provides the most fit set of genes (or hyperparameters) at the end of training from the available pool of hyperparameter. The proposed algorithm thus provides an efficient path for solving the hyperparameter optimization problem.

It is also seen in the various experiments conducted that the adjacent-ranked scheme of mating is a better option than extreme-ends scheme. A second observation is related to the choice of mating scheme based on immediate parent, among these the cousin to cousin scheme provides much more efficient and faster converging model than sibling to sibling scheme.

ACKNOWLEDGMENT

Firstly, want to acknowledge Andrew Ng, Geoffrey Hinton, Sebastian Thrun for their online training on deep and machine learning in Coursera and Udacity which ignited our interest in the field of deep learning and artificial intelligence. We would like to acknowledge Shashank Sharma for stimulating our creativity and proof reading this paper to provide his valuable inputs. Rushil Gupta provided much-needed support with LATEX typesetting.

REFERENCES

1. Taijia Xiao, Dong Ren, Shuanghui Lei, Junqiao Zhang, Xiaobo Liu, Based on grid-search and PSO parameter optimization for Support Vector Machine, Proceeding of the 11th World Congress on Intelligent Control and Automation.
2. James Bergstra, Yoshua Bengio, Random Search for Hyper-Parameter Optimization, Journal of Machine Learning Research 13 (2012) 281-305.
3. Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. Neural Networks, 16(1):5–9, 2003
4. K.F. Man, K.S. Tang, S. Kwong, Genetic algorithms: concepts and applications, IEEE Transactions on Industrial Electronics.
5. Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, Jeff Clune, Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, ICLR, 2019.
6. Adarsh Sehgal, Hung Manh La, Sushil J Louis, Hai Nguyen, Deep Reinforcement Learning Using Genetic Algorithm for Parameter Optimization, CoRR, 2019.
7. Yanan Sun, Bing Xue, Mengjie Zhang, Garg G Yen, Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classificiation, CoRR, 2018.
8. Eli Omid David, Iddo Greental, Genetic Algorithms for Evolving Deep Neural Networks, GECCO Comp'14, 2014.
9. Delowar Hossain, Genci Capi, Genetic Algorithm Based Deep Learning Parameter Tuning for Robot Object Recognition and Grasping, 2017.
10. David J Montana, Lawrence Davis, Training Feedforward Neural Networks using Genetic Algorithms, IJCAI'89, 1989.
11. Xiaodong Cui, Wei Zhang, Zoltán Tüske, Michael Picheny, Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks, NIPS, 2018.
12. Vishal Prem, Mark Sheridan Nonghuloo, Nagaraja Rao A, Optimization of Siamese Neural Networks Using Genetic Algorithm, International Journal of Engineering and Advanced Technology (IJEAT), 2019.
13. Gregory Morse, Kenneth O Stanley, Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks, GECCO, 2016.