Individual Project

Report

Bartosz Cyprian Kowalewski

Project repository: <https://github.com/Kovali0/Neural-network-with-genetic-algorithms>

**Genetic algorithms in the optimization of neural networks weights**

1. **Introduction**

Genetic algorithms are intended for optimization and finding global minimum and maximum. They are inspired by the natural evolution process. In neural networks whole learning based on evolution of weights and biases, which are like generations in genetics. How about combine both of these specific and “natural” algorithms.

Genetic algorithm can be used for train a neural network. This report show an idea how it can be done in a specific way. Network weights change during training, especially in backpropagation, but before this, network start with random weights and biases. Sometimes they are better, sometimes worse. In fact initialized weights in random way, always have some accuracy, even if it is only 20% or 10%. What is more, they are different each time, when new network is created. That’s what is perfect for genetic algorithm.

This project implements the idea of optimize initial weights for neural network, by genetic algorithm, what gives gradient descent a better starting position and allows for fewer training epochs with higher model accuracy. Consider the following neural network where layers are treated as chromosomes and weights as genes fields in a computational object.

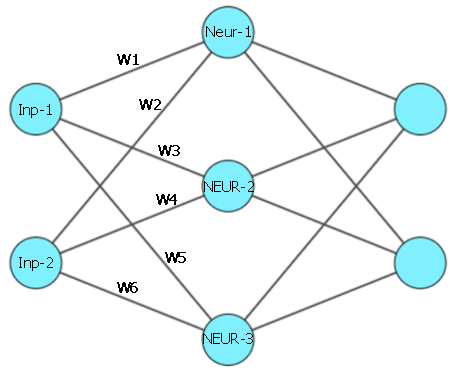
For example, on the first diagram there is neural network with marked weights and layers elements.

W[number] – is weight with id

NEUR-[number] – specific neuron in layer

Inp-[number] – input data from last layer with activation status

NN – neural network on diagram



Neural network diagram 1

For this NN, the elements necessary for a genetic can be constructed as follows:

* One NN is a one individual in generation
* Each individual have three chromosomes and each chromosome is one layer. In this case the chromosome assigned to hidden layer is built from three neurons (NEUR-1, NEUR-2, NEUR-3).
* Genes are weights.

W1 🡪 One Gene

[ [W1, W2] [W3, W4] [W5, W6] ] 🡪 Chromosome

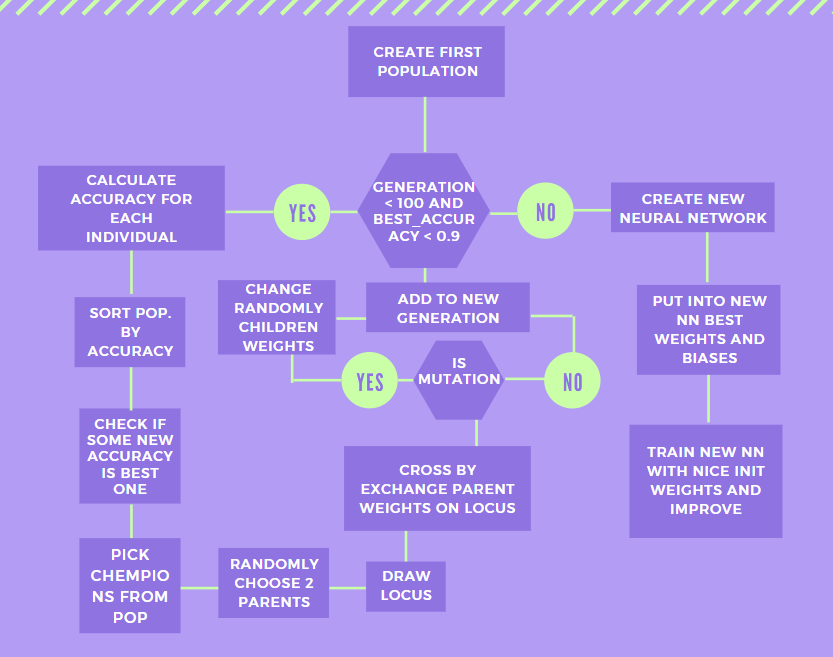
3 x Chromosome 🡪 Individual

When the NN is coded into this form, which is readable by the genetic algorithm, crossover between NNs and mutations during this becomes possible. Thanks that whole NN can be used as main subject in genetic, only what is need to is generate an initial population from many of these NNs and put it into customized genetic. The algorithm will select the best weights which can then be used for initialization for a new NN, which on start will get high accuracy. That should make this NN faster and easier to learn and increase efficiency.

1. **Project struct**

Project structure is very simple. There are two files, one with code for neural network and main file with custom code for genetic algorithm. Neural network class is composed of objects of the class Layer, which stored in numpy arrays weights and biases. Also in Layer is defined activation function. For the experiment architecture of NNs is defined manually in NN class constructor, because only one was used during the research. This architecture is show on NN diagrams 1 and 2.

In the main file located is custom genetic algorithm working on NNs and presenting weights and accuracy. Whole code is documented and described with python helper and comments in genetic part.



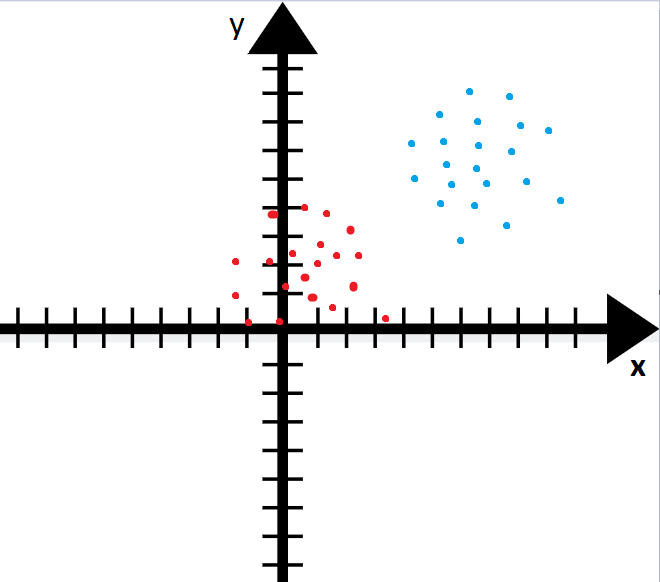
Genetic Algorithm diagram 1

The picture above shows a diagram of the genetic algorithm's operation on the NNs. In short this genetic is based on basic assumptions of natural evolution algorithms. The main different is that, it doesn’t work on binary coded chromosomes but on NN weights as genes. Also mutations change whole weights value.

1. **Use case and dataset**

This genetic implementation can be use on special built NN class for this problem or with Keras NN models after some changes. The NN object is designed for simple problem, classifying x and y coordinate points into two classes.

For test purpose dataset which was used is very simple. Dataset include points with coordinate (x, y) and label to which class this point belongs. Points were generated around the main point with one label. And in that way all point with same class was near by each other. The sample of points is show on the Dataset draw 1.



Dataset draw 1

The NN get the values X and Y from point and deduces which class is point with given coordinates. On the picture above they are two simple generated groups of points. Groups number can be bigger, for example it can be 4 groups, 2 for first label, and 2 for second label. Moreover, if these groups are positioned alternately on the plane then this is hard dataset for learning, for which genetic results were much worse.

1. **Experiments**

The experiments mainly concern the modification of the genetic algorithm to find it in the shortest possible time, as much as possible best NN weights for higher NN accuracy, with as little computer resource consumption as possible. Experiments are divided into 3 main parts: part of research on the best type of crossover, finding best mutation method, compare selections.

**Research on the best type of crossover**

The first part is the largest and has caused the most confusion during the various trials. At the beginning it should be noted that, using the evolutionary naming convention, in fact only one type of crossover was used throughout the entire experiment, single point crossover. However, it is not a basic genetic, but an algorithm designed specifically for this problem. So in this case, there are many more possibilities of crossing individuals for one generation, especially due to the fact that each individual has 3 chromosomes. In research were used 3 types of crossover. Each type was sensitive to data complicated level.

One works on layers and swap between two individuals one, randomly chosen layer. This option was the worst, most of tries return accuracy around 50% on easy dataset, what may indicate the selection of weights for the NN with a tendency to guessing on the basis of 50:50 chance. On the other hand, values for hard data was not very better than first random accuracy. More or less the improvement was 10%. This method has one more problem, sometimes algorithm went into dead loop and all new generations for last 20 generations have almost the same genes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cross whole layers | Cross neurons | Cross weights from input |
| Simple data accuracy | ~60% | ~80% | ~95% |
| Hard data accuracy | ~40% | ~55% | ~75% |
| Avrg generations | 100 | 100 | 5 / 100 |

Table 1 crossover comparison

Second type is cross one neuron in each layer, what gives really nice results for simple data, but in case with hard dataset, final accuracy was weak and after initial new neural network with selected weights and biases, training required almost the same amount of time as with traditional machine learning for NN.

The last method consists in crossing the genes of one randomly selected input between two parents, for each their chromosomes. That’s mean the swapped weights were connected to one input. Like on the neural network diagram 1, for the hidden layer chromosome with locus value 2, then it will cross of all weights from Inp-2: W2, W4, W6. And this one method, have the best results and what is more, it’s the only one method which on simple data could find accuracy over 90% in smaller number of generation, even in few (eg. 5-6).

**Finding best mutation method**

In this case measure of tests was positive influence on performance by introducing qualitative diversity in genes. By quality diversity should be understood as added sometimes to the genes pool new genes which improve NN scores and help avoiding dead loops. Mutation chance was low, 1%-2%.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Negative value | Input weights | Random neuron |
| Impact | Small | Meaning | Nothing |

Table 2 mutations comparison

Negative value just pick randomly weights and negate their values. This have really small impact, since in most cases the changes made are not significant, but they help to avoid dead loops.

Mutation based on randomly changing input weights, the same which are chosen during cross to swap with other parent, is the best mutation, very stable and sometimes it can provide weights which improve NN efficiency.

Last one mutation pick random neuron from the layer and totally change all its weights in random way. The results of this method were surprisingly low. These type of mutations did not bring anything positive, and sometimes even makes NN miscalculated the results.

**Ranking Selection vs Champions Selection**

Most of experiments time, genetic algorithm used Ranking Selection. Over new population, after calculated fitness (NN accuracy with prediction method) and sorted by fitness, top 5 (in latter tests 10) units were chosen and they were parents for next generation. That method give stable results, thanks what by improving other parts it was possible to get higher final number of good NN predictions.

Champions selection had been different from Ranking selection in one step. After calculated fitness and sorted generation individuals, top 5 was picked and then they automatically move on to the next generation, rest next generation was creating by crossing this 5 champions, each one 3 times. This selection have one problem, when number of generation is high and the mutation did not occur for a long time, then the algorithm fell into the death loop and by the end of the 100th generation all individuals were almost identical to the genotype.

1. **Conclusions**

During the experiments, it was possible to find a relatively stable structure of the genetic algorithm used to optimize the starting weights and biases for neural networks. Best structure will be selected parents for new generation by ranking method and crossover parents on each chromosome (layer) swapping random input weight. In case of mutation it will randomly choose new weights values for the new input weights after cross. Such a geneticist makes it possible to find starting weights for a neural network with an initial accuracy of about 70%. This shortened the learning time of the neural network.

Still the research and experiments based on one problem, with one type of dataset, which means that the results cannot be treated generally. Genetic algorithms can be well and usefully composed with neural networks, but in this case it is not certain that despite the promising results that there will not be a case in which the work of the genetic algorithm will extend the whole program working time.

1. **Sources**

<https://ichi.pro/pl/10-funkcji-aktywacji-o-ktorych-powinien-wiedziec-kazdy-analityk-danych-216332217495600>

<https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2018-03-1/NET-2018-03-1_08.pdf>

<https://medium.com/swlh/genetic-artificial-neural-networks-d6b85578ba99>

<https://medium.com/@a.mirzaei69/implement-a-neural-network-from-scratch-with-python-numpy-backpropagation-e82b70caa9bb>

<http://tjaworski.kis.p.lodz.pl/zajecia/psi/genetyczne/wyklad_pu.pdf>