VU Robotics WS19/20 Genetic algorithms

Maximilian Gallinat, Michael Hauser

December 2020

1 Introduction

As we started with the project we got a FNN provided from Hector Manuel Perez Villeda. The network trys to fit a model to a complex trajectory and consists of layers of functions. Each Layer has a specific number of neurons and blocks of formerly specified functions. The functions which are already implmented are: idendity, cosinus and sinus. Another parameter is the complementary neuron. It is up to us to decide which complementary neuron is executed while running. At the end of the network a division layer is implemented. In addition to this we have different demonstrations provided. Furthermore there is the initial weights described with its initialisation value. The starting train step is defined at 50000 training steps. More values to change, without going to deep into the network itself is the number of product neurons, the number of hidden layers and the number of function blocks each hidden layer has. Our Task is to modify the network with genetic Algorithms. In the beginning we thought a approach to change the network itself could be considered. As we realized, that the network is already finished and works really well we needed to take a different way. With working really well it's said that the loss value is below 0.001. One thing which we understood pretty fast was that the network took about 10-15 minutes to train one time. For our concerns, training genetic algorithms which means try to create a few different networks and compare each one with another we needed to have a faster calculation. So the idea which we got was to try to make the network as fast as possible with around 100 steps. So on one side, we can try to tune the network before the training itself and also we can work with 100 steps for computation time. Another task which we got from Matteo Sevariano was to check for conflicting bundles, different combinations of weight with the same loss value.

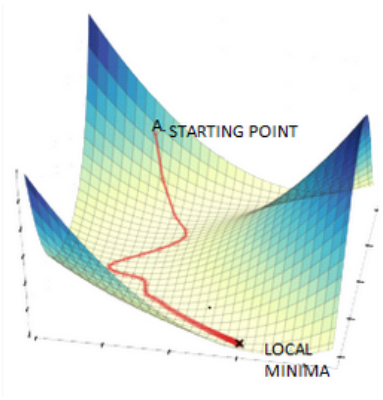

Figure 1: Local minima
https://www.hackerearth.com/blog/developers/3-types-gradient-descent-algorithms-small-large-data-sets/ (09.12.20 10:58)

The idea of tuning is to change the starting point closer to the local minima. So the neural network can reach its calculation for the local minima faster.

2 Genetic algorithms

The idea of a genetic algorithm is closely linked to the biology therm of evolutionary. It's the survival of the fittest. It is meant to create a population of candidate solutions. These candidates can be chosen relatively randomly, depending on the number of candidates which is possible to choose from. Afterwards, each solution candidate is valued with a number (the value of fitness) which in the case of a functional neural network is the loss function. Succeedingly a new generation of candidates is created from the first generation. There is various ways of creating new generations. The most common ones are crossover, selection and mutation. Cross over describes the creation of the next candidates by crossing candidates from the former generation. As an practical example, if we take two binary codes 111000_1 and 101010_2 and want to crossover these two we may do: 101000_1 and 111010_2. Mutation is meant by just randomly or purposely changing a part of the solution genetics. In the former example 111000_1 becomes 101000_1 we flipped the second bit. Selection means to choose a few good candidates which we want to have in the further generation. The idea is to keep good solutions(candidates) and to throw away bad solution to consecutively reach the best possible population and with the best population the best candidate.

3 Implementation

As already said the computation time of the network is the key point to our implementation. For our implementation we used a library provided by python.The Documentation can be found on
https://pypi.org/project/geneticalgorithm/.
First we thought about taking the whole number of weights into the genetic algorithm. But because genetic algorithms take a lot of computation time and each candidate is a neural network itself, we decided to tune the weights which are used to initialize the neural network. So we created the genetic algorithm with 13 parameters for each candidate. The 12 Initialized weights, and the complementary neuron. There is more possibilities to do, but the computation time is going through the roof pretty far. More implementations would consider the other parameters: No_Product_Neurons, Blocks_FN, Num_HLayers.With each candidate having its own parameters we decided to take less, otherwise the algorithm with our computational possibilities would take too long just trying different options instead of following the objective function. The time to evaluate a good network which is on a good way to the goal would be a massive population. The parameters to change at the neural network are mainly the number of iterations and the population size. We found out that, in our case, a

big population 100 is a good indicator and a iteration size of 10 to 20. When we put in these two parameters the algorithm usually stops between 13 to 17 iterations because no big improvement is achieved. The steps of each candidate, each neural network was set to 5. We got a Loss value below 1 which is already quite good. Sadly the number of iterations and the population is not good enough to tune the network properly. This would be achieved by

(1) More population and iteration.

(2)Tune all the weights with the genetic algorithm and not the initialization interval.

Next thing we going to try is to check for conflicting bundles. We already have a code, which is commented because it throws an error. Another thing which would be nice to speed things up a little bit would be an implementation with cuda. We had a proper look at it, but it takes a serios amount of time because each time we use a numpy array we either need to change it in the beginning as a cuda array or cast it. Another problem which is ahead of us is the functionality of tensorflow and cuda.

4 Example values and output

```
algorithm_param = {'max_num_iteration': 100,\
                   'population_size':10,\
                   'mutation_probability':0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3,\
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':3}
```

Figure 2: The parameter we used for the algorithm

```
{'variable': array([-0.79596206,  0.04603481, -0.90692849, -1.38617691, -1.99793338,
        1.68811629,  0.53249785,  0.72779847, -1.649684  , -0.82969249,
        0.25781283, -1.20216011,  0.41272264]), 'function': 0.697761993855238}
0.04603480653137351
```

Figure 3: The variable which the algorithm returned

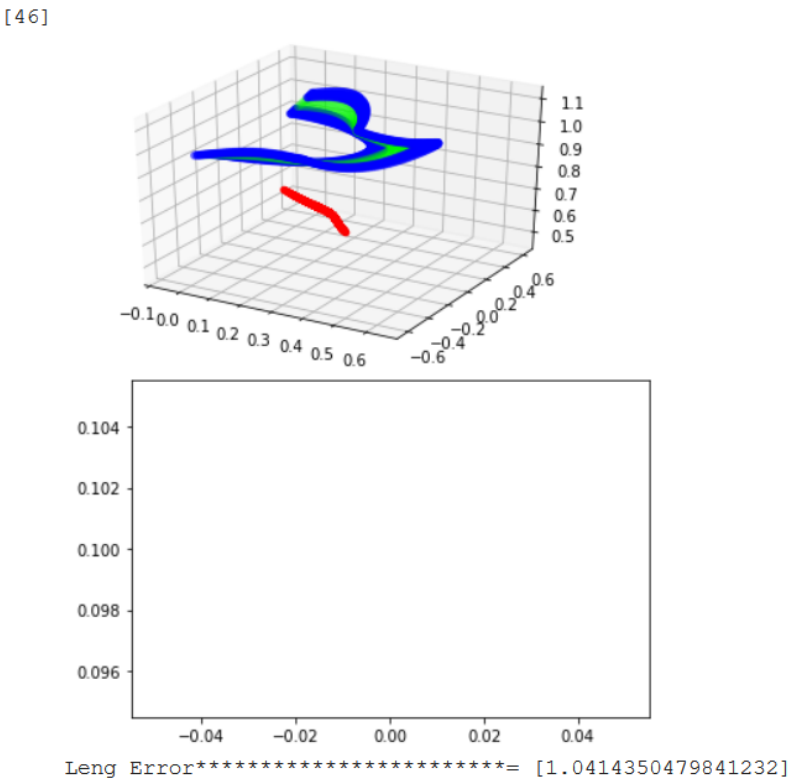

Figure 4: The neural network with 5000 steps variables are the weights initialization value and the function is the value we got returned by the loss function of the neural network with our algorithm and 5 steps.

As you can see on the last Figure, with 5000 steps and our algorithm, the red line which is the prediction line is far away from perfect. But as said before the computational aspect is key.