F21BC: BIOLOGICALLY INSPIRED COMPUTATION

Coursework 1 – Black-box optimization

Adrien Chevrot & Cedric Tremolosa
ac87@hw.ac.uk - cpt4@hw.ac.uk

Introduction to the algorithm and general description

To solve the problem given in this coursework we decided to implement a Multi-Layer Perceptron with supervised learning using Java and the IDE used is Eclipse Neon, release 4.6.1. This MLP consists in an input layer, one or several hidden layers and an output layer. The number of neurons in each layer is determined by the number of inputs. During this coursework we used different configurations and each will be detailed later, including the number of layers and neurons used.

Our algorithm begins with the creation of the neural network. We decide the number of layers we want and the number of neurons. So far, the network has a static architecture, it can't modify itself. The freedom given by this type of architecture allows us to run various experiments and keep control of the size of our network.

Once the network has been created, we initialize all the weights with random values between -0.5 and 0.5.

Then we start training the network with the data provided. For each set of inputs, we evaluate the output using the neural network in feed-forward way. Then, comes the backpropagation where we compare the real output to the expected output and using the mean-square error calculation in order to adjust the weights of the neurons. These steps are performed for all the training data provided until the error is fewer than a certain constant we fixed.

Once the network is trained enough and respects this criteria of global error, we can use it with data to approximate the function we want.

The code of the project is available on Github at this address: https://github.com/CedT83/BIC_CS

Experiments

All along the experiments we will use some constant characteristics for the neural network such as the momentum which will remain at 0.9, all the activation functions are sigmoid functions, the maximum number of iterations allowed is 70 million and the desired error must not be greater than 0.0001.

The first experiment was to test the network using logical function to make sure the algorithm works perfectly. Thus, we used a 2-2-1 perceptron with a learning rate of 0.7 and a momentum of 0.9, also with a sigmoid as activation function for all the neurons. The aim was to recreate the XOR function and our neural network is able to do so with a training of approximately 10 000 iterations.

Then we decided to model the sphere function to increase the difficulty. We tried with several configurations in order to determine some optimal parameters that can be easily found by hand. We tried with multiple data normalizations, learning rates, number of layers and neurons per layer.

The first configuration was: a 2-2-1 architecture, 0.7 for the learning rate and normalized data within the interval [-1, 1]. The algorithm is still the same than the one detailed in the introduction. The number of neurons in the hidden layer is 2 because after some researches we found a certain rule of thumb saying that the number of neurons must be between the number of inputs and outputs, or may be 75% or the number of inputs in the previous layer.

The second configuration was quite the same but with a different learning rate. I order to find betters parameters we only modified one to compare its effects. This time the parameters were: a 2-2-1 architecture, 0.3 for the learning rate and normalized data within the interval [-1, 1].

For the third one, we simple added a new hidden layer to the first configuration. So the characteristics were a 2-2-2-1 architecture, 0.7 for the learning rate and normalized data within the interval [-1, 1]. We took the learning rate of the previous experiments because the results were better with this value. All the result will be explained in the dedicated part, but this had to be explained at this stage.

The penultimate experiment was to use a binary normalization, so for this we created a neural network with the following characteristics: a 64-64-32 architecture, 0.7 for the learning rate and normalized data using their binary values. So we used only integer values for simplicity purpose. The number of neuron has increased but if we consider that we have 32 inputs now for a previous value, the ratio remains the same.

Experiments

Concerning the results, by using the multilayer perceptron with three layers, one for the inputs, one hidden and one for the outputs, we managed to make the MLP behave in order to get the good results for a XOR. Basically, we instantiate the weights of the synapses between the neurons to a random value between -0,5 and 0,5.

Then we set a threshold of 0.0001 which means that our retro-propagation loop will keep going while the network's average error is superior to this threshold. Once we have respected this error's threshold we stop the program and display the results which are as expected for a XOR operation, as

0 – 0 -> 0

0 – 1 -> 1

1 – 0 -> 1

1 – 1 -> 0

In this XOR learning, we reach this average error's threshold with a number of iterations usually between 10000 and 13000. This iterations' number depends on the weights that are initially set. Actually, if randomly, the weights are initialized to values which are not far from their final value using the retro-propagation, the neural network will have less work to do than if the weights were initially far from their final value. That is why we get always a different number of iterations.

Our multilayer perceptron can handle different logic operations such as XOR, AND, OR etc...

We tried to make the perceptron behave correctly using a sphere function in order to see if the neural network could handle a continuous function.

Unfortunately, the neural network does not manage to converge toward a value.

We tried to add another hidden layer to the network but it did not work, the network still cannot converge.

We also tried to modify the learning rate. By doing that, we affect the retro-propagation process but the network is not able to converge.

We tried to use two different types of standardizations:

The first one was to make the perceptron work by setting the inputs between -1 and 1 but unfortunately it did not change anything concerning the results since it did not converge toward a correct value.

The second one was the binary solution: instead of using numbers like 1 or 0 as inputs, we tried to convert these numbers into bytes and set inputs equal to these bytes. The bytes are then sent to the perceptron. This method gives us the same results than the previous ones, the perceptron does not behave as we expected.

Conclusion

As we said previously, our neural network gave us correct results concerning a logic operation but we meet difficulties when it comes to a continuous function. We know that we dealt with a part of the coursework only, but we only had the time to do this since we began the coursework one week and a half ago.

Indeed, we are following a MSc in Software Engineering, unlike others students in the course. We do not know the amount of work they had to deal with before working on this assessment but for us we had almost one coursework per week with the other courses (Industrial programming, Rigorous Methods and Info System Methodologies).

We are not hiding behind these facts in order to explain why we did not manage to handle the coursework but we wanted to give the situation in which we have been. We built this neural network but unfortunately with the time we attributed to the coursework it was not enough to complete it with others parts of the coursework. We clearly know that we should have dealt better with the previous weeks by treating the different courseworks at the same time but with the continuous deadlines and overlapping courseworks, we actually treated the courseworks in the order of their deadlines.

In addition to this we did not find any tutorial to use COCO with Java and we were unable to compile any test code from the COCO's website in Java. When looking for a solution on the website of INRIA we found a solution which consists in recompiling C libraries and include them to the Java release of COCO. But once these ones downloaded and included another error occurred during compilation to inform us that it was not compatible with our system. And so far, we could not find any solution for this problem.

Again, we do not ask for kindness but it is hard to deal with this kind of problem when this happens a couple of days before the deadline so now, we move, from now, forward with the second coursework.

References

En.wikipedia.org. (2016). *Mean squared error*. [online] Available at: https://en.wikipedia.org/wiki/Mean_squared_error [Accessed 5 Nov. 2016].

Fröhlich, J. (2016). *Backpropagation - Neural Networks with Java*. [online] Nnwj.de. Available at: http://www.nnwj.de/backpropagation.html [Accessed 7 Nov. 2016].

Gurney, K. (2003). *An introduction to neural networks*. 1st ed. Boca Raton, FL: CRC Press.

Krose, B. and van der Smagt, P. (1996). *An introduction to neural networks*. [online] http://lia.univ-avignon.fr/. Available at: http://lia.univ-avignon.fr/chercheurs/torres/livres/book-neuro-intro.pdf [Accessed 7 Nov. 2016].

Oracle, (2016). *Java Platform Standard Edition 7 Documentation*. [online] Docs.oracle.com. Available at: https://docs.oracle.com/javase/7/docs/ [Accessed 4 Nov. 2016].

StatSoft, (2016). *Neural Networks*. [online] Fmi.uni-sofia.bg. Available at: http://www.fmi.uni-sofia.bg/fmi/statist/education/textbook/eng/stneunet.html#artificial [Accessed 13 Nov. 2016].