

Artificial Neural Networks Applications

4th year computer and control engineering Graduation project

Name | Course Title | Date

# DEVELOPMENT OF OUR NETWORK ARCHITECTURE:

We reached our final performance after many trails and changes in our neural network architecture and by architecture we mean four things:

* The parameters we choose to construct the multi-layer perceptron network which sometimes called the hyper-parameters like (learning rate – number of epochs – batch size – number of layers – number of neurons in each layer).
* The activation function used in each layer.
* The cost function that we choose to minimize.
* The optimization algorithm.

We will describe every trail we did in terms of the architecture we used and the results we obtained, but before that a description of the mechanism of the training is given, this mechanism is used for all the trails.

## mechanism of training:

Our approach in training is based on the vectorized implementation of the computational graph of the standard (fully connected) multi-layer perceptron neural networks, by this we mean that we are doing any computation for all the neurons of a certain layer in the same time, not only this but we used batch training which means that this computation is done in parallel for a certain number of examples that is determined by the batch size, of course this vectorization idea made the implementation quite easier and faster.

It goes like this, first we prepare our input data and labels which is the truth table of the XOR logic function using “InputOutput(X, Y)” function, this will return two matrices X and Y holding the training data in X and the labels in Y, then these X and Y is fed into our neural network class, now it is time for the journey of training.

It starts with initializing the matrices W[L] and b[L] which hold the weights and biases respectively for the layer L, this is done for every layer and this initialization is random, then we construct our first dictionary “parameters” which holds the weights and biases for all the layers. When using “Adam” or “LM” algorithms we initialize some matrices with zeros needed in the computation and put them into the dictionary “grades” which holds the gradients of any parameter used in the network, these matrices are then used in backpropagation.

After initialization we start our feedforward journey using the “feedforward()” function by feeding the input data to the first layer and then using the weights and biases that was initialized recently to compute the activations of the next layer and so on until we reach the final layer which outputs the prediction according to the current weights and biases of the network.

The generated output is then fed into the cost function with the desired labels and the cost function does its work and calculates how our network behaves.

In the beginning the cost is utter trash, which means our network is completely lost in the nothingness, we need to tell it how to behave and here comes the backpropagation, we back-propagate the error through the network using the “BackProp()” function and compute a gradient for every parameter in the network, this gradient tells us how every parameter in the network should change in order to give us a better performance in predicting the labels, after computing the gradients we update the network parameters according to optimization algorithm we use, and this completes our first iteration in training!

The previous process is iterative by nature, it is repeated for number of times until we reach our desired performance and then calculate the accuracy, the process of training the network with the whole training data for one time is called an epoch.

Next we are going to list our trails

## trail 1:

Architecture:

* Learning rate = .8
* Number of epochs = 15000 to 20000
* Batch size = 4096
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 119 -1)
* Activations used in each layer respectively : (non – relu – sigmoid)
* The cost function : Cross Entropy
* The optimization algorithm : Gradient Descent

Results:

* Accuracy = 99.7% ± 0.2%
* Number of false predictions = 5 ± 2
* Time = 2500s to 4000s
* Convergence rate = 19/20

Comments:

The previous architecture is a little bit naïve, the gradient descent optimization alone is very slow the and this made it hard to reach the 100% accuracy, we used 0.5 threshold in the output layer, by this we mean that if the output form the sigmoid activation in the last neuron > .5 we consider it 1, otherwise it is considered zero.

## trail 2:

Architecture:

* Learning rate = 0.09 to 0.01
* Number of epochs = 600 to 1000
* Batch size = 1024
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 119 -1)
* Activations used in each layer respectively : (non – relu – sigmoid)
* The cost function : Cross Entropy
* The optimization algorithm : Adam

Results:

* Accuracy = 100%
* Number of false predictions = 0
* Time = 600s to 1000s
* Convergence rate = 19/20

Comments:

With the introduction of Adam optimization algorithm we reached 100% accuracy, the convergence to minimum became faster, the number of epochs used is much smaller than previously which decreases the execution time significantly, an important thing to mention here is the choice of the learning rate was critical in this trail, after we implemented Adam we used 0.8 learning rate and it was not working at all, when we changed it to .09 it worked well and did the job. The classification in this trail was based on the concept of dead-band; that is if the output of the last layer is >= .8 then it is considered 1 and if it is < 0.2 it is considered zero.

## trail 3:

Architecture:

* Learning rate = .08
* Number of epochs = 500
* Batch size = 512
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 34 – 1)
* Activations used in each layer respectively : (non – satlinear – satlinear)
* The cost function : Square Error
* The optimization algorithm : Adam

Results:

* Accuracy = 99.9%
* Number of false predictions = 4
* Time = 70s
* Convergence rate = 8/10

Comments:

This trail was a breakthrough in the execution time that is it was 70s, it was basically done due to 3 important improvements in our architecture:

* We introduced the square error cost function, with it we labeled our data to be -1,1 instead of 0,1 and.
* We tried using tanh activation function but it tends to be slower than saturated liner activation so we used satlinear instead of tanh, up to this moment the network was not doing so well and its performance wasn’t good enough, the inputs to the activations reached the saturation region quite early, which made the activations of some neurons output zero gradients and this means it stopped learning, this was a challenging problem of course and we partially came over it by changing our initialization to weights as illustrated in the next point.
* To make the training starting from a quite good location in the space of the training set and to make its values near to each other so that the activations are not saturated early, we randomly initialized the weights of the network and made its mean equal to zero and its standard deviation equal to 1, this prevented the neurons from being saturated early.

The previous improvements made us reach a quite good accuracy in small execution time, we could have stopped at this point but we weren’t satisfied with 99,9 % accuracy, at this point we decided to implement different optimization algorithm to make a better performance than that, here comes our 4th trail.

We used deadband classification with predictions > 0.8 classified to be 1 and predictions <-0.8 classified to be -1.

## Trail 4:

Architecture:

* Learning rate = .025
* Number of epochs = 500
* Batch size = 1
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 16 – 1)
* Activations used in each layer respectively : (non – satlinear – satlinear)
* The cost function : Square Error
* The optimization algorithm : LM

Results:

* Accuracy = undetermined
* Number of false predictions = undetermined
* Time = 15s / epoch
* Convergence rate = undetermined

Comments:

We implemented LM optimization algorithm which, we hoped that it would work but its results were very disappointing, as it includes 2nd order differentiation and some computation includes matrices inverse, the single epoch took 15s which is very large compared to our previous trails, we tried to tune it with different parameters but it did not work.

## Trail 5:

Architecture:

* Learning rate = .025
* Number of epochs = 500
* Batch size = 2048
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 32– 1)
* Activations used in each layer respectively : (non – satlinear – satlinear)
* The cost function : Square Error
* The optimization algorithm : Adam

Results:

* Accuracy = 100%
* Number of false predictions = 0
* Time = 60-90 s
* Convergence rate = 8/10

Comments:

In this trail we just added a small but very significant modification to the way we initialize the weights and biases, the weights for every layer is divided by square root of the number of neurons in the next layer, this forces the inputs to the activation of the next layer to be small enough to avoid saturation of neurons and made the weights initialization in a better location in the input data space, this way of initialization is called Xavier initialization.

The way we classified the predictions this time is the same as previous, we used deadband classification with predictions > 0.9 classified to be 1 and predictions <-0.9 classified to be -1.

## Trail 6:

Architecture:

* Learning rate = .095
* Number of epochs = 500
* Batch size = 2048
* Number of layers = 3
* Number of neurons in each layer respectively : (12 – 32– 1)
* Activations used in each layer respectively : (non – satlinear – adaptive\_satlinear)
* The cost function : Square Error
* The optimization algorithm : Adam

Results:

* Accuracy = 100%
* Number of false predictions = 0
* Time = 10-20 s
* Convergence rate = 8/10

Comments:

This is our final architecture for the parity problem, mainly it is the same as the previous trail but in it we did two modifications, one of them was changing the learning rate to be 0.095, the other was changing the activation of the last layer and use adaptive\_satlinear, so how is it works?

Adaptive\_satlinear is the same as the satlinear activation, the thing in it is that it has the ability to change its slope during the training time according to maximum error between output predictions and the desired labels, this made the input to this final neuron is not only the inputs from the previous layer but also the maximum error exists in the network “adaprive\_satliner(Z[L],maxErr)”.

The way we implemented it was straightforward, after the network finishes an epoch we compute the maximum error between the output predictions of the network and the desired labels, then this maximum error is forwarded into the adaptive\_satlinear neuron in the next epoch, if this maximum error is smaller than some value we choose before training, the slope of adaptive\_satlinear would change to a value that we alse choose before training, now you may ask why this operation of changing slope really works?

The answer to this comes from observing that the network decides on which examples must be classified as 1 and which examples must be classified as -1 in an early point of the training, say after 50 epochs, the rest of the time the network just does its best to make these examples reach absolute 1 and -1, when we reach this point we could simply make the slope of the satlinear larger -relatively speaking the satlinear approaches the step function- this will cause a lot of examples to reach their desired labels immediately which introduces a faster learning process.

The training stopes when the maximum error between outputs and labels reach zero, and this clearly means that all examples are perfectly learnt and we reached 100% accuracy with no deadband classification in small amount of time ranges from 10s to 20s.