Simple Neural Network

Marc Vernet Sancho

October 7, 2021

The goal of this project is to develop a fully functional neural network in C, that is trainable with backpropagation and can be used with any shape of input data. The architecture of the neural network consists in an input layer, a hidden neuron layer and an output layer. The number of neurons on each layer can be chosen freely by the user.

Following this introductory description, some test have been developed to prove the correct implementation of the network.

1 Cost function comparison

The default data example (1) has been used to experiment with the cost function input. Both matrices of weights can be initialized in two ways, with the already provided random initialization or with Xavier initialization [Xav].

$$X = [23.0, 40.0, 100.0] \quad y = [1] \tag{1}$$

The evolution of the cost function among several epochs can be seen of Fig.1. The cost has been plotted in green for three cases with different number of neurons in the hidden layer, all of them using a random weight initialization. In blue there's a 3 hidden neurons network initialized with Xavier initialization.

It can be seen that all of the plotted cases converge on a similar cost value, more or less on the same epoch. The number of neurons in the hidden layer afects the cost value on the first epochs and the speed of convergence. Xavier initialization doesn't bring any improvement, proving that is a technique working better on large networks.

2 Iris dataset

To further prove the correct working of the network, a real-life dataset in going to be used. The iris dataset [Iri] consists on 50 samples of 4 values each other that describe 3 flower species. So the input size will be 4 neurons and the output will be 3 neurons.

On a real machine learning task the dataset should be divided in test and train, but in this case considering the extremelly small size of both the dataset and the network, it was impossible to get the network to generalize and have meaningful test predictions. Anyway, this aproach is enough to test the correct working of the network.

2.1 Stochastic gradient and batch gradient

Having more than one item to train allows to test different methods for updating the neuron weights, in this case stochastic gradient and batch gradient. It's also an opportunity to test different learning rates.

In stochastic gradient descend, for each epoch each item is forwarded and backwarded, and weights are updated. In batch gradient, the derivatives are accumulated and averaged and only at the end of the epoch the parameters are updated. Batch gradient should provide a faster convergence with much less updates.

The result can be seen in fig.2. With batch gradient the learning rate didn't provide any meaningful changes. With stochastic gradient, a larger learning rate meant reaching convergence slightly faster,

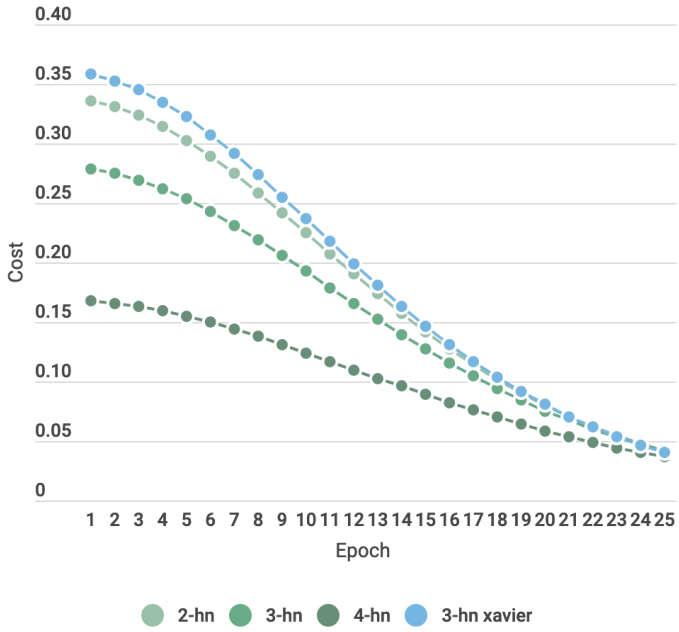

Figure 1: Comparison of cost value decline with different architectures

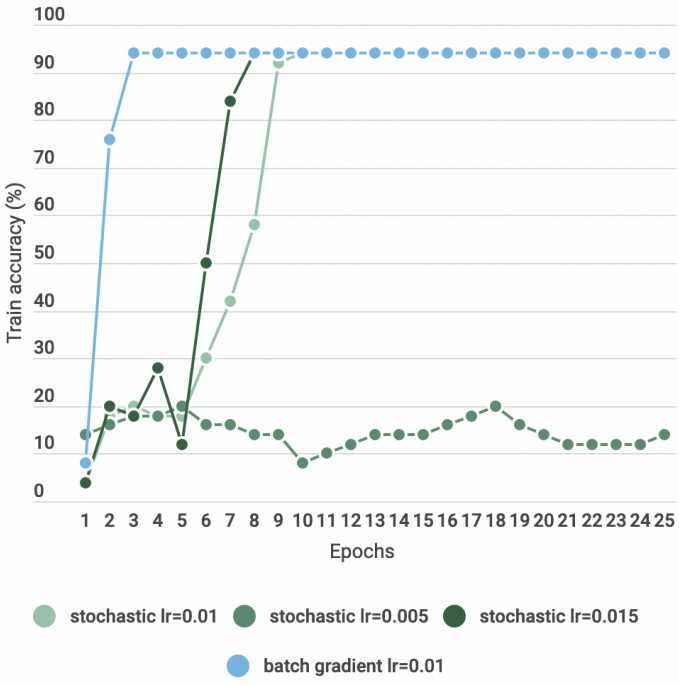

Figure 2: Comparison of accuracy between different gradient methods

and an smaller learning rate made the network not work at all. On the cases that worked correctly, the networks reach convergence at the same accuracy. As expected, batch gradient is faster in reaching convergence.

2.2 A note about code

All the main functions are defined on the *simple_neural_network.c* file, most of the functions follow the same definition that was given in the lab and there's a new implementation of column-wise normalization function. The results of the first section can be obtained with *test.c*, and the results for the second section with *stochastic.c* and *gradient.c*.

References

[Iri] Iris dataset. https://en.wikipedia.org/wiki/Iris_flower_data_set. Accessed: 2021-10-7.

[Xav] Xavier initialization. https://paperswithcode.com/method/xavier-initialization. Accessed: 2021-10-7.