CI Practice 2: Neural Networks

Rueben Alvarez, Nafis Banirazi, Jan Carbonell

I. Abstract

In this session, we have studied the performance of different neural networks used for image classification with the CalTech 101 Silouette DataSet. The Data conists of 101 silouette classes and 8671 instances of 28x28 pictures.

II. Neural Network Configuration

For our neural network we have set up a Multi layer Perceptron with 781 input variables and 101 outputs (defined by the desired loading and printing files). In essence, the 781 variables correspond to each of the pixels of the image (28x28 pixels). Assuming it is defined on one single channel (white and black only), each 0 and 1 of the matrix corresponds to an activation of the pixel.

By creating MLPerceptron with several hidden layers and training it with the desired outputs, it then creates a system that is able to extrapolate a specific output given new data. For example, given a dataset of human drawn digits, we could train it with different sets of digits so that the final outputs are 0...9. When we sent that MLP a new digit that it had not previously seen it would be able to guess the number it more closely resembles to. The accuracy is given by the amount of times it gets it right over the total attempts.

In order to accelerate the gradients in the right directions we use the SGD algorithm with momentum. The momentum is nothing more than a 'moving' average that gives us a frozen view of where the data lies at a given point, in order to reduce the noise and a smooth continuous function. We then use this to update the weight of the network.

The network is capable of adaptive learning. In other words, it can generalize from past experiences (train data) and extrapolate. Such process is done by adjusting the specific weights of the hidden layers of the network. The gradient is required in order to do that and since it is computationally expensive to calculative in a forward manner, we use the packpropagation of errors, since the output error can be distributed back though the layers and get the gradient via the chain rule.

A. Hidden and output Layers

As for the functions of the hidden and output layers they are H: `Logsig` O: `Logsig` in case 1 and H: `Logsig` and O:`Softmax` in the case 2. In the case of logsig, it goes from -1 to 1 in a non-linear way, which allows the network to learn non-linear relationships between input and output vectors. Softmax goes from 0 to 1 and in a way, it synthesizes numbers into a normalized scale (or probabilities).

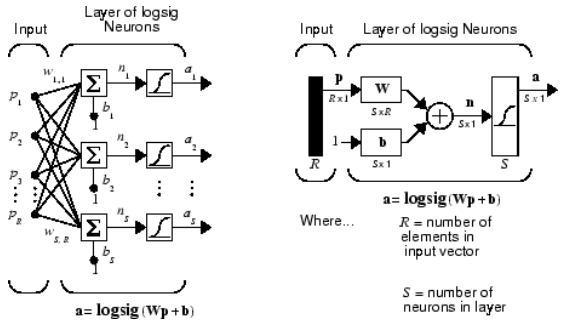


Fig. 1: Case 1 Hidden and Output Layers [4]

B. Performance function

The two options for the performance function were the mean squared error, which measures the average of the squares of the errors and the cross-entropy errors, which calculates an output with the input x taking into consideration the w vector of weights -calculated using gradient descent-[1]. While there are certain applications in which SSE performs better than Cross-entropy *-Example: Using ReLU as the output of the hidden layer, since it won't always produce values between (0,1)-* normally the Cross-entropy is seen as a better performing error function since it is better fitted to maximize the probability of a correct output for a given input. [2]

C. Percentage of Training

As for the percentage training/testing/validation split we have tested three different approaches (80/10/10, 40/40/20, 10/10/80). One extreme minimizes the variance in the training data and the other one in the testing data. The key part of this is that some model need substantial amounts of data to train on so normally the rule of the thumb seems to be to have a greater chunk of training that testing. However there are novel approaches such as one-shot-deep learning that mirror the way humans often learn (once seen one banana, we know what a banana is) that would resemble the latter one.

III. Selection of Hyper-Parameters

In order to find the best performing results for the given configurations, we had to manually test them before benchmarking the results against one another. Since the overall objective of the report is also to evaluate the value of such configurations for the hidden layers, output layers, percentage of training and performance function, it was clear that once we settled on a set of numbers we would have to stick with them until the end of the simulation even if we though they could be improved mid run.

In order to obtain one of such parameters such as λ we also used this localized approach. We froze the rest of the

parameters, picked the best performing one and then tweak it until we got a decent performance. We are aware that, since those parameters are often related, fitting one to its best value and then changing the following one might not reach a global optimum but given the time restrictions on the project, we felt that a local maximum was already enough fine-tuning. As for the epochs, we tested them in the most demanding conditions (500 layers, softmax, 80/10/10 split) and based on the number obtained (around 200) we set an additional threshold so it would never overcome it (250 epochs).

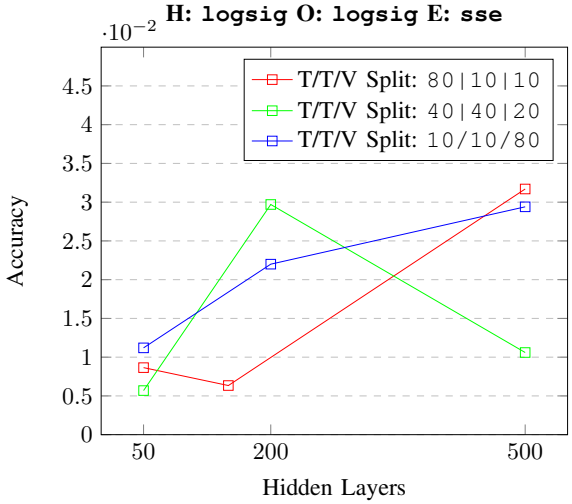
Given the following methodology, ur optimal values have been the following:

$$(\lambda, \mu, epochs)(0.05, 0.9, 250)$$

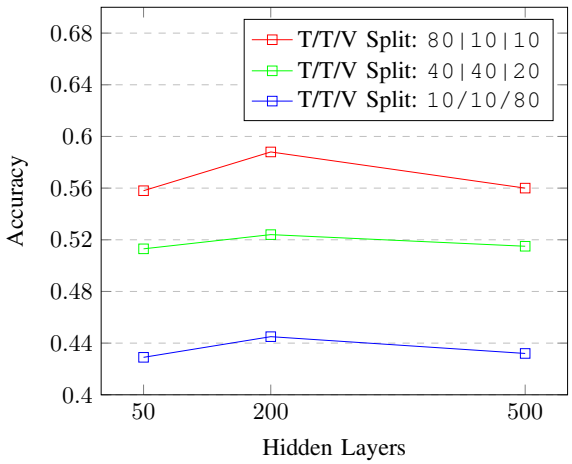
IV. Results

After 5 runs, we created a graph that plots the average results. In this aspect of the report, our focus was on visualizing the variations in performance with the change of the given hyperparameters.

H: `logsig` O: `logsig` E: `sse`



H: `logsig` O: `softmax` E: `cross-entropy`



V. Conclusions

As shown in the first figure in the results above, containing data of the hidden layer, output layer and performance function, **H: `logsig` O: `logsig` E: `sse`** respectively. There is a clear distinction in the increase of the green line (train/test/val percentage: 40/40/20), compared to the red (80/10/10) and blue (10/10/80), around 200 hidden layers. Given the significant decrease to 500 hidden layers, it hints to an over-fit after training with 200 hidden layers, with 200 being a more suitable amount to train the MLP neural network for that configuration. However, the other lines, red and blue, show a significant increase after 200 hidden layers, meaning that the optimal parameters are more near the 500 hidden layers or have not yet been reached. The best score finally is the 80—10—10 split, as expected, since for a large training set and a small test set would create a better trained network than a smaller train set. Nevertheless it must be noted that the accuracy is extremely low and thus it may be concluded that the configuration with logsig as an output layer is not a good model to fit this particular MLP neural network and with all likelihood, additional MLP NN's [3].

Lastly, the second figure in the results above shows the configuration **H: `logsig` O: `softmax` E: `cross-entropy`**, for the hidden layer, output layer and performance function respectively. The graph clearly portrays the expected relative behavior of the division of train, test and validation. With a higher accuracy for a 80/10/10 distribution, a lower for 40/40/20 and 10/10/80 being the lowest accuracy. The constant decrease of each line after 200 hidden layers, is a sign of over-fitting. The optimal amount of hidden layers is between 200 and 500.

In summary, the configuration with softmax as an output layer is significantly better than logsig, and would be recommended for future work using MLP neural networks. For the best results obtained with the second configuration had an accuracy of almost 60 percent, which is considering the sheer amount of instances, about 80, an impressive result.

References

[1] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[2] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[3] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[4] Mathworks. Multilayer shallow neural network architecture.