UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Computational **I**ntelligence

Laboratory Exercise 2: Neural Networks

Course 2020-2021
November 20st, 2020

Students:

Victor Badenas - *victor.badenas.crespo@estudiantat.upc.edu*
Roger Marrugat - *roger.marrugat@estudiantat.upc.edu*

Contents

Contents 2

Different configurations performed 3

Parameters 4

Tables 5

Conclusions 6

Different configurations performed

As a first test, the experiment was run with some of the configurations mentioned in the task. However, they were run with adaptive stochastic gradient descent (traingdx) with the default parameters. The best model, after running all the required explorations (50, 200 and 500 hidden units, Logsig/MSE and Softmax/Cross-entropy, and the split percentages) and all of their combinations, was found to be at a 80-10-10 split with 200 hidden units and a softmax/cross-entropy configuration.

The results obtained for the model are 23.43% in train accuracy, 25.38% in validation accuracy and 22.38% in test accuracy after 2000 epochs. The fact that the network has the same accuracy on the train test and validation leads us to believe that it still has some room for improvement, as it still has not arrived at the inflection point where it starts to overfit. Looking at the accuracies extracted from all the combinations it can be inferred that configuration 1 (Logsig/MSE) is not good for this kind of task and the splits other than 80-10-10 are not suitable for the training of a network. Because of that, the next experiments will be only performed on the 80-10-10 split configuration and the softmax/cross-entropy functions.

		Layers Configurations	
Split:	Test Split	LogSig/MSE	Softmax/C-E
Hidden Units	50	4.84%	11.77%
	200	6.46%	22.38%
	500	7.04%	0.58%

Table 1: Accuracies for Train / Validation / Test splits: 80-10-10

The results from the brute force all-vs-all lead to main conclusions:

- The only partition that has some respectable results is the 80-10-10 partition. This makes sense as most of the data should be in the training partition to best train the model. Some data has to be assigned to test and validation in order to have a better understanding on how the model is able to generalize, but the most data is needed for the network to train.

- Softmax/Cross-entropy functions are way better for classification with respect to Logsig/MSE. It makes sense for Logsig and MSE to perform way worse as they are functions usually used for regression and not classification tasks. The Softmax/Cross-Entropy functions are more suitable for that kind of task. Softmax is usually used to compress the range of an *n* dimensional vector of probabilities.

- 50 and 500 hidden units present comparable but worse results compared to the 200 configuration. This can mean that the 200 hidden units is a sweet spot in terms of neuron connections between the two fully connected layers.

Parameters

After several tries with the aforementioned configurations, we came to the conclusion that a change in the network's hyperparameters was needed, specially focusing on the net training function (traingdm).

Using the best model configurations from the last section, the trainFunction was first modified. Using the default parameters for the backpropagation algorithms, a run was performed with each of the following functions: trainrp, trainscg, traincgb, traincgf, traincgp, trainoss, traingdx, traingdm, traingd. All these backpropagation algorithms were tested for 50, 200 and 500 hidden units as can be seen in Table 2.

	Units	trainrp	trainscg	traincgb	traincgf	traincgp	trainoss	traingdx	traingdm	traingd
Train	50	72.08%	76.47%	77.28%	79.80%	71.41%	55.15%	78.33%	8.71%	0.74%
	200	87.75%	94.31%	93.31%	89.84%	83.16%	68.24%	84.79%	0.91%	0.78%
	500	27.69%	76.52%	89.84%	94.16%	91.93%	85.83%	84.66%	0.76%	3.06%
Test	50	56.29%	56.52%	52.48%	**59.63%**	54.33%	45.21%	56.17%	7.96%	0.69%
	200	51.67%	59.05%	**60.44%**	59.52%	59.29%	51.33%	55.36%	1.38%	0.23%
	500	25.84%	55.02%	57.67%	**58.59%**	57.79%	58.36%	55.36%	0.23%	3.23%
Validation	50	51.44%	56.63%	59.40%	59.17%	57.32%	47.06%	56.17%	9.34%	0.58%
	200	56.75%	57.44%	56.29%	61.02%	58.48%	51.56%	57.79%	0.92%	0.46%
	500	26.30%	56.06%	56.06%	62.51%	57.09%	55.25%	56.52%	0.23%	3.23%

Table 2: Accuracies obtained for each training function with default parameters.

The best results were obtained by the 'traincgb' training function for 200 hidden units with a 60.44% of accuracy. However, the best results overall were achieved by 'traincgf' as for 50 hidden units, it is able to get a 59.63% of accuracy which is a very good result for that configuration, so maybe a tuning of the parameters in 'traincgf' would yield better results than the best of this table.

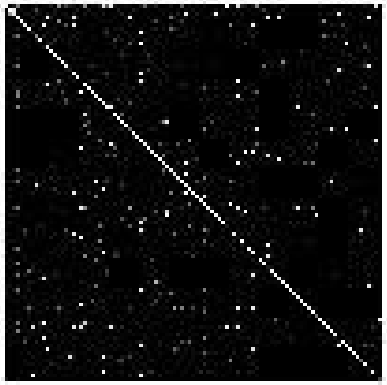

Figure 1: Confusion Matrix plot of the test split accuracies for 'traincgb' training function with 200 units

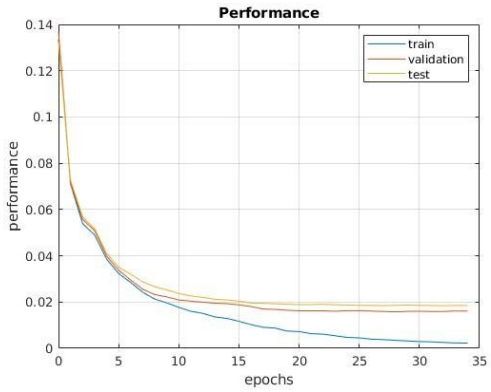

Figure 2: Train, Validation and Test loss plots for 'traincgb' training function with 200 units

In figures 1 and 2 we present graphically the results for the best model achieved by our parameter search. In figure 1 we present the confusion matrix in which it can be seen that the diagonal contains most of the values, implying that a vast amount of the examples used in the test split were correctly classified and there was no particular bias or misclassification between classes. In figure 2, we present the cross-entropy performance for the best model. It can be observed that the training performance monotonously decreases and the validation and test metrics do not improve once reached a specific value of epochs. This fact usually symbolizes that the model has been correctly trained even if the generalization is not as good as we would like.

As both 'traincgb' and 'traincgf' are Conjugate Gradient Algorithms instead of Gradient Descent ones, we don't need to specify concrete values of learning rate α or momentum γ. Also, due to its faster convergence rate we did not need to determine any specific number of epochs, so we decided to stick to the default 2000 epochs value which is never reached.

Conclusions

In this report we show the results of the multiple parameter scans and tweaks that have been done to an MLP shallow network for the classification of silhouette images dataset.
In the process of adjusting the parameters and hyperparameters (size of the network, learning rate, and so on) we could check how each one of them contributes positively or negatively to the performance of the ML system.

For instance, a small learning rate will make the training (and the learning of the target function itself) very slow and the network will be prone to get stuck in an under fitted model, as the weights of the layers are not twinked enough to approximate the target function correctly. However, when setting the learning rate to an extremely high value, we observed a tendendency of the loss function to diverge as it surpasses the minima. The optimal learning model will be the one that balances these two aspects.

Concerning the hidden units or the neurons of the hidden layers, a small number may result in a fast training of the network but might imply that our models lack degrees of freedom to approximate the target function. In contrast, a huge number of neurons not only increases the training time and the network size but also may implicate that the networks learns too well the training examples, producing an overfitting over this split.

Furthermore, it's really important to take into account not only the dataset and it's preprocessing but also the kind of task that we are dealing with at the time of choosing the appropriate activation, transfer or cost function of the network. As stated, in a classification task like the one presented in this exercise, a Softmax activation function for the output layer of the network much suits much better as it creates a distribution probability of the output classes, providing the membership values of the input to each one of the classes. Selecting Cross-Entropy or Negative Log Likelihood as the cost functions is the best option for this kind of task as they allow to compute the error of classification for all the classes with ease.

In another type of NN architecture, for example a Convolutional Neural Network (CNN), another kind of activation function like ReLU (Rectified Linear Unit) might suit better as the outputs of the convolution layers are closely related to the correlation function from which we are not interested in the negative values.

We conclude the report by stating that from the best results that could be obtained using the MLP setup, we can notice that the accuracy is not good enough for this kind of task, which is something expected as CNNs are a more suitable network architecture for this kind of tasks. That is due to the capability of the CNNs to limit the activation of a neuron to a small context within the image. Contrary to that special feature of CNNs, the MLP does not have this kind of spatial context restriction so all neurons in the output/hidden layers can be activated by all neurons in the hidden/input layers.

Additionally, a simple histogram was computed for the output labels of the ground truth, where we noticed that, as it can be seen in figure 3, the number of examples of each class is clearly unbalanced in the dataset. Maybe a balance in the distribution of the classes would also help the network to generalize better.

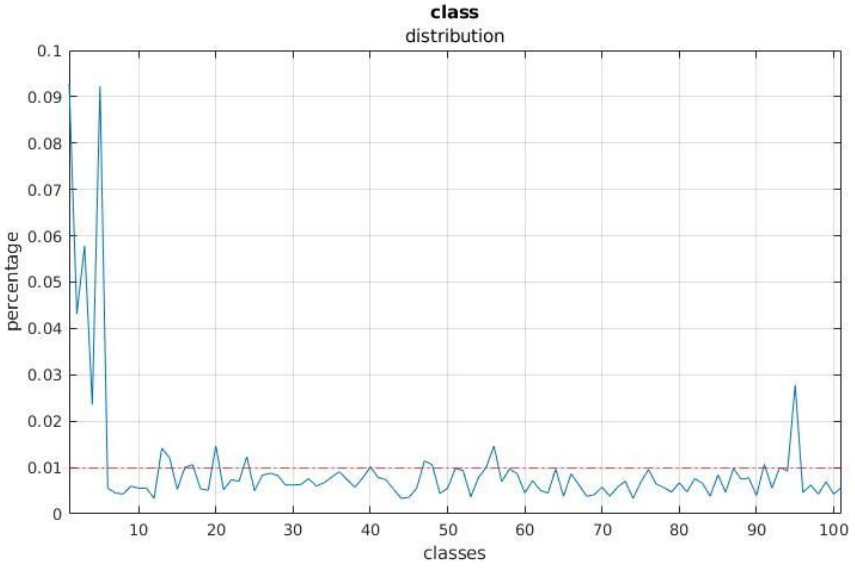

Figure 3: class distribution in the ground truth of the dataset