Laboratory exercise 2: NEURAL NETWORKS
Computational Intelligence

Joan Llop Palao - Jordi Armengol Estapé

28-11-19

1 Description of the runs with the different configurations

We have run our network 18 times, each time with a different configuration of the following hyperparameters, the ones we want to study:

- The output activation (or transfer) function and the error (or performance) function.
- The number of units in the hidden layer.
- Train, validation and test ratios.

For all of the 18 possible configurations, we have run the network three times, with different seeds (1, 2, and 3, in order to make our results reproducible). We have taken the mean accuracy as an estimator of the expected accuracy with the given configuration (and given a random seed). The results shown in table 2 are the mean of the three runs.

The hyperparameters that were not compared were fixed for all the runs, in order to have a fair comparison. In section 2 we detail how he have selected these other hyperparameters.

The train set is used for training the networks. The validation set is essentially used for monitoring the training procedure and the early stopping (a regularization technique that we will detail later on) criterion. The accuracy is taken from the test set, which has not been seen during training. Notice that in each run the train-valid-test split is different because, by design of the Matlab package we are using, it is done within the train function (it cannot be decided from outside of the function, by us). We have some methodological concerns about this matter, because for comparing different runs the sets should be equal (except in the cases in which the ratios are different, but those should be different experiments), but it is how the package works.

2 Selection of the rest of parameters

Before experimenting with the aforementioned configurations, we did a previous exploration for selecting the rest of the parameters. Performing an exhaustive search over the full search space of all the possible hyperparameter combinations was not feasible. Instead, we performed a grid search for some of them, and set the other ones to arbitrary values according to some heuristics.

The used training function, also known as *optimizer*, was gradient descent with momentum and adaptive learning rate backpropagation. The reason why we discarded the Levenberg-Marquardt optimizer is the amount of memory needed to store the Jacobian matrix, which in our case would have had a size of about 14GB. We chose the variant with adaptive learning rate over the other alternative since an adaptive learning rate generally leads to better results, by letting the optimizer have large learning rates when gradients are big and the optimization direction is more consistent (typically, at the beginning of the training procedure), and decreasing the learning rate when the

loss is already small, gradients are tiny and the search must be fine-grained (usually, at the end of the training procedure).

We set the number of epochs, the early stop patience and the minimum gradient to the default values. By default, the used Matlab package performs early stopping, which consists in stopping the training once the validation loss starts to increase. It is both a regularizer, in the sense that it prevents the optimizer to further optimize the training loss by overfitting, and a way of decreasing the execution time, in the sense that it decreases the number of required epochs. In this context, and this specific problem, the default number of maximum epochs, 1000, is an arbitrarily big enough value that will probably never be reached due to early stopping. The patience is set to 6 checks, the default value.

In the case of the learning rate and momentum parameter, we *did* search for a good enough value. Specifically, we performed a grid search (even though a random search can sometimes be more efficient) over the logarithmic scale (ie. different orders of magnitude). We experimented with the following values:

$$l_r = \{0.1, 0.01, 0.001\}$$
$$m_c = \{0.9, 0.09, 0.009\}$$

In the case of the learning rate, for selecting the values to test, we took the default value, 0.01, and added and removed one order of magnitude, respectively. With respect the momentum parameter, the maximum allowed value is 1, while the default was already set to 0.9, so we tested 0.09 and 0.009.

Since it was not computationally feasible to explore these values with all the combinations, we fixed the following hyperparameters:

- The same hyperparameters that we have already said that would be fixed for all the experiments were constant as well, with the same values we detailed.
- Sets ratios: Since there are different set ratios in the configurations we want to experiment with, we decided to set the three sets to have the same size, in order not to bias the search towards the optimal values for big train sets and small test sets (or the reverse): `trainRatio` = 0.34, `valRatio` = 0.33, `testRatio` = 0.33.
- Number of hidden units: For computational reasons, it was set to 50, the smallest of the values that we will compare. While it is true that this could bias the search towards the optimal values for less hidden units, we believe that it could be the case that the results with 50 units were somehow indicative of the results with more units.

Nevertheless, in order not to bias the search towards a specific error function or output activation function, we performed the execution with the two relevant combinations (the ones we want to compare, the output activation function and the error function) and took the mean accuracy.

In this case, we did not run more than once each configuration, for computational constraints.

The values with the best accuracy were: $l_r = 0.1$ (so, one order of magnitude above the default value) and $m_c = 0.9$ (the default value), with an accuracy of 0.15.

In table 1 we summarize the hyperparameters that we used for all the experiments.

3 Results

Apart from the aforementioned fixed hyperparameters, notice that there is one additional setting that must be constant, this time because of the requirements of the assignment: the activation function of the hidden layer will be set to `logsig` for all the runs.

Hyperparameter	Value
Optimizer	`traingdx`
l_r	0.1
m_c	0.9
epochs	1000
`max_fail`	6
`min_grad`	1e-5

Table 1: Hyperparameters fixed for all the experiments. Some of them were decided by grid search and some of them were arbitrarily set following some heuristic.

In table 2, we summarize the obtained results. The best results are consistently achieved when we set softmax as the output activation function and cross-entropy as the error function, independently of the rest of the hyperparameters. In order to find a reasonable explanation to this results, let's take a look to the logsig and the softmax equations:

$$\text{Logsig: } f(x_i) = \frac{1}{1+e^{-x_i}}$$

$$\text{Softmax: } f(x_i) = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

outputFnc	performFnc	nHiddenUnits	trainRatio	valRatio	testRatio	accuracy
logsig	mse	50	0.8	0.1	0.1	0.28
logsig	mse	50	0.4	0.2	0.4	0.28
logsig	mse	50	0.1	0.1	0.8	0.22
logsig	mse	200	0.8	0.1	0.1	0.391
logsig	mse	200	0.4	0.2	0.4	0.35
logsig	mse	200	0.1	0.1	0.8	0.35
logsig	mse	500	0.8	0.1	0.1	0.35
logsig	mse	500	0.4	0.2	0.4	0.33
logsig	mse	500	0.1	0.1	0.8	0.31
softmax	crossentropy	50	0.8	0.1	0.1	0.53
softmax	crossentropy	50	0.4	0.2	0.4	0.49
softmax	crossentropy	50	0.1	0.1	0.8	0.41
softmax	crossentropy	200	0.8	0.1	0.1	**0.55**
softmax	crossentropy	200	0.4	0.2	0.4	0.51
softmax	crossentropy	200	0.1	0.1	0.8	0.42
softmax	crossentropy	500	0.8	0.1	0.1	0.54
softmax	crossentropy	500	0.4	0.2	0.4	0.49
softmax	crossentropy	500	0.1	0.1	0.8	0.41

Table 2: Results summary for the studied hyperparameters: output activation function, error function, number of hidden units and train, validation and test ratios, respectively. The accuracy is the mean accuracy over three runs with different seeds.

Where k is the number of outputs and x_i is the ith output with $i \in 1..k$. We can observe that the Logsig function is a specific case of the softmax when the number of classes is 2 ($k = 2$) and

assuming that when the class of an instance is A cannot be B:

$$\begin{aligned}\text{Softmax } = f(x_i) &= \frac{e^{x_i}}{\sum_k e^{x_k}} \\ f(x_i) &= \frac{e^{x_i}}{e^{x_i} + e^0} \\ f(x_i) &= \frac{e^{x_i}}{e^{x_i}(1 + \frac{1}{e^{x_i}})} = \frac{1}{1 + e^{-x_i}} = \text{ Logsig}\end{aligned}$$

Since we had only two output classes, we calculated the output value for one class without considering the other class (we set the output value for the other class to zero). Therefore, we can see the logsig function as the softmax function assuming independence between classes.

In this experiment, we had to classify an instance among 101 classes. Therefore, there was a dependence between the outputs. The softmax function gives us exactly this property: when the output for one class increases, the output for the other classes decreases. Thus, the softmax function has much more sense to use than the logsig function.

With regard to the other studied hyperparameters, the best results are obtained the bigger the train set, since the model has more data to learn (with the risk of obtaining non-representative validation and test sets). The results are slightly better with 200 hidden units than with 50, but they do not improve when increasing this number to 500, probably because the model is way more complex and more prone to overfitting for this particular problem. Provided a certain performance goal is met, the simpler the model, the better.

The best configuration is obtained with softmax, cross-entropy, 200 hidden units and 08-0.1-0.1 as the train-valid-test ratios, although some similar configurations get results that are very close to them (and the difference is not necessarily significant).

4 Conclusions

To sum up, we have studied the effect of the output activation function, the error function, the number of hidden units and the train-validation-test split ratios in a Multi-Layer Perceptron and found the best configuration. In order to have a fair comparison, the non-studied hyperparameters have been fixed once we had selected them by the use of heuristics and a grid search, and all the given configurations have been run thrice in order to diminish the effect of the stochasticity of neural networks training.

We have found the combination of the output activation function and the error function to be the most relevant parameters. The ones typically used in classification settings clearly outperform the other alternative, as we have detailed. Apart from using softmax and cross-entropy, the best configuration was the one with the most data in the train set, which makes the learning easier, and having many hidden units was not such a good idea, since the model does not take advantage of that complexity and is more prone to overfitting. We consider the best results (¿ 50% accuracy) to be good enough, since the dataset had as many as 101 classes. Even though it was somehow imbalanced, the results are way better than the ones that a constant classifier would have obtained.

5 Implementation

The experiments are implemented in Matlab in the attached file `silhouettes_net.m`. We are committed to reproducibility, by specifying the random seeds, and legibility, by refactoring the code into functions whenever possible.