UNIVERSITAT POLITÈCNICA DE CATALUNYA

COMPUTATIONAL INTELLIGENCE

NEURAL NETWORKS

Authors:

BENJAMÍ PARELLADA
CLARA RIVADULLA

Fall Term 2022/2023

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Contents

1 Introduction **1**

1.1 Architecture 1

1.2 Loss Function 1

1.3 Optimizer 1

1.4 Metrics to monitor 2

1.5 Fixed Parameters 2

2 Results **3**

3 Conclusions **3**

1 Introduction

The aim of this report is to study the performance of different Multi-Layer Perceptron (MLP) configurations used to classify a multi-class image set pertaining of *CalTech 101 Silhouettes Data Set*. This dataset contains 8671 images of size 28x28 which are flattened to a singular vector. Moreover, each image vector can be any of the 101 classes, which we will have to one-hot encode for proper use in the feed-forward neural networks. These classes are not balanced, i.e., each class does not appear with the same frequency.

The goal of the project is to understand the basic parameters of an MLP configuration, and test them out in the dataset. There are four main parameters we can change in any neural network.

1.1 Architecture

The actual architecture of the network: amount of layers, and the amount of nodes for each layer. Moreover, at each layer an activation functions needs to be defined as well.

– **Size of the hidden layers**: we will test the sizes of a 1 layer MLP with sizes: 50, 100, 200, 500, 1000. And multi-layer with sizes: (50, 100), (50, 100, 50), (100, 200, 50).

– **Activation Function (hidden)**: for the hidden layers, we will always use either the *logsig* or the rectified linear unit function (ReLU), which in Matlab is *poslin*.

– **Activation Function (outer)**: for the outer layer, we either use the *logsig* function or the *softmax*. The amount of nodes for the outer layer depends on the problem, in this case 101.

1.2 Loss Function

This is how the network will be able to measure how good a job it is doing on its training data, and thus how it will be able to steer itself in the right direction. This should not be picked randomly, and should be done according to the type of problem and the transfer function from the outer layer. Hence, we will use:

– For the *logsig*, we should use the Mean Square Error.

– For the *softmax*, we should use the Cross-Entropy.

We would have liked to test out the *negative log likelihood loss*, which should be used with a *log softmax*, since these have proven to be more numerically stable. However, Matlab does not have any of these two functions for the feedforward training function we are using.

1.3 Optimizer

This is the mechanism through which the network will update itself based on the data it sees and its loss function. Since we are using a GPU not all optimizers can be used, e.g. *trainlm*, and it will make us prefer gradient based optimizers. Moreover, due to the GPU, we need to map the minimum and maximum of each row to [-1, 1], otherwise Matlab will reject the data. The actual optimizer we would like to use is either a *Stochastic Gradient Descent* one, or *Adam*, which are the two most typically used. Moreover, this should be done using a scheduler, which modifies the learning rate on plateau. However, the function we are using on this assignment does not have these functions, so we will have to satisfy ourselves with other optimizers. We will test the following three:

1. **RP** – *trainrp*, resilient backpropagation – which in theory is the fastest for pattern recognition problems out of the ones in the toolbox. And has a learning rate parameter, and four more parameters regarding its delta.

2. **GDX** – *traingdx*, gradient descent with momentum and adaptive learning rate backpropagation – which Matlab says goes pretty slow compared to the other methods. It has a learning rate, plus a ratio to increase it and decrease it. Additionally, it also has a momentum constant.

3. **SCG** – *trainscg*, scaled conjugate gradient backpropagation – which seems to work quite well for various problems according to Matlab. It does not have a learning rate parameter per se, though it does have a Marquardt adjustment (μ), and a parameter (σ) to change the weight in the second derivative approximation.

1.4 Metrics to monitor

Metrics to monitor during training, validation, and testing. We will evaluate the quality of the predictions of a network with the mean accuracy. To obtain the accuracy, we get the amount of total correct matches of predicted label against the true label and divide it by the total number of observations in the partition. For each of the configuration runs, we will run the experiment 3 times to get its mean accuracy. To get a predicted label, we do the argmax of the network's output.

Moreover, we need to define how to split the dataset randomly to be able to effectively get the test accuracy for an unbiased evaluation of the configuration. This is done doing a random split of the indices, where we configure the ratios of each of the different partitions. We calculate the accuracy of a partition using only data from the partition.

P1: Train Ratio = 80%, Validation Ratio = 10%, Test Ratio = 10%

P2: Train Ratio = 40%, Validation Ratio = 20%, Test Ratio = 40%

P3: Train Ratio = 10%, Validation Ratio = 10%, Test Ratio = 80%

No other partition will be tested, since the ratios are not usually a parameter to modify when parameter tuning in neural networks. It will be of interest in this report to see how, in theory, using fewer data for train will achieve worse results.

1.5 Fixed Parameters

Since running the Cartesian product of all the elements presented would be too time, and electrical, consuming, we decide to fix some parameters first and maintain them while running a smaller Cartesian product of experiments, which can be seen in Table 3.

The parameters we will fix are: maximum number of epochs, the optimizer, the optimizer parameters, minimum gradient where it will stop, and the maximum validation errors where it will stop. All these parameters are mostly talking about early stopping parameters of the training. Hence, it makes sense to test these values with the largest architectures we will use, since they will have the most parameters to estimate. Moreover, we will train it with the partition that is 80% training data, since it will require more computation time to estimate the weights.

Therefore, we will train and test the MLP with sizes $\{1000, 500, (100, 200, 50)\}$, return their mean accuracy, and mean number of epochs to train, which we rounded up to be discrete, and the minimum final gradient observed.

Size	**Accuracy (%)**			**Epochs**			**End Gradient**		
	RP	**GDX**	**SCG**	**RP**	**GDX**	**SCG**	**RP**	**GDX**	**SCG**
500	10.61	23.56	35.17	13	272	136	-	0.001	0.001
1000	8.36	20.31	30.91	9	453	253	-	0.001	0.0001
(100, 200, 50)	2.83	8.73	15.34	25	763	485	-	0.0227	0.0002

Table 1: Summary of mean test accuracy, epochs, and gradient each optimizer reaches with its default parameters, and increasing maximum validation fails to 15. Using the different size architectures with *logsig* as the internal transfer function, and *softmax* with *cross-entropy* as outer.

In Table 1, we can see how larger architectures do not necessarily return better accuracy, hence we can omit {1000, (100, 200, 50)} from the search. Additionally, even though RP finishes in the fewest epochs, it always returns worse accuracies than the other two. Moreover, GDX seems to be pretty slow overall, which could be improved using higher learning rate, however, it still returns worse accuracy than the SCG. Overall, we prefer the SCG, since it returns the highest accuracy.

With the same parameters as before, we modify the optimizer parameter by evaluating different intermediate values linearly in the log-domain, starting from the default given from Matlab. The results can be seen in Table 2. We can observe that the best value for Marquardt adjustment is 0.0005, however it does not really seem to effect the final performance that much. The weight in the second derivative (σ) has not been shown to be as relevant in our dataset, so we leave it at 5e-5.

	$\mu = 0.0005$	$\mu = 0.005$	$\mu = 0.05$	$\mu = 0.5$	$\mu = 0.8$
Accuracy	37.90	36.71	31.68	37.54	34.98

Table 2: Summary of the mean test accuracy it reaches with different μ. Using size 500, with *logsig* as the internal transfer function, and *softmax* with *cross-entropy* as outer, and (80/10/10) partition.

Once we determine the SCG optimizer with parameters $\mu = 0.0005$ and $\sigma =$ 5e-5, we run a few more runs with a very large number of the validation fails, such that it is not the stopping reason. With this we can observe two things, SGC terminates usually under 1000 epochs, and its breaking condition is having minimum gradients of under 1e-10. Therefore, we can raise the minimum gradient for breaking in order to train for more epochs. We raise it to be 1e-20, while maintaining the epochs to be maximum of 1000. We also set the maximum validation fails to be 1000. This will allow all the tests to have more or less the same number of epochs trained in order to be able to compare better between them. This will probably induce overfitting, though.

2 Results

Observing Table 3, it might seem that the MLP did not perform very well in classifying the silhouettes, since the best test accuracies are a bit below 50%. While this might seem like a failure, take into consideration that this is multi-class labeling, hence the random probability of guessing the class is $1/101 \approx 1\%$, and we can see that most of our classifiers return better accuracy than this. Additionally, the classes are not balanced, so it makes it even harder to train and evaluate the classifiers.

There are a few key observations we can make observing the results. *1)* there is a lot of overfitting, this is mostly due that we made the early stopping parameters very lenient. Reducing the validation fails, or increasing the minimum gradient, would result in lower overfitting since the training would stop earlier. However, we decided not to do this in order to showcase the clear overfitting if we were to train for more than necessary. We can see this very clearly when we obtain accuracies of almost 90% on the train data, and then we dip under 40% on the validation. Moreover, reducing the training partition increases the amount of the overfitting, due to training with fewer data. This can clearly be seen in the (10/10/80) partitions, where even though we get high score in the train, it falls quite low on the test set. *2)* using *logsig + MSE* seems to work much worse compared to using the *softmax + CE* combination, except on the multi-layer architecture, where it works better. *3)* increasing the size of the architecture does not really help in the classification in this dataset, and it increases the overfitting of the data compared to smaller architectures. *4)* while we had high hopes for the *ReLU*, it did not seem to work nearly as well as the *logsig*, this is probably due to the network being too small to be worrying about the vanishing gradients. *5)* even though a partition with (40/20/40) returned the best test accuracy, we rather use (80/10/10) since it will generalize better, and not overfit as much.

3 Conclusions

We have seen that setting the architecture of a neural network is not a trivial task, and changing the different components will result in different performances. We have compared quite a few architectures and parameters on the Silhouettes image dataset, and found an adequate performance on some of these. Our best results reach almost 50% mean accuracy on the test set, which is acceptable since we are dealing with a multi-class labeling problem, and this is much better than a random guess, or guessing always the most frequent class. Moreover, we found that for the multi-label problem, using the *softmax* with the *cross-entropy* results in the best scores. This was to be expected, since it is a generalized version of the logistic function for multi-class classification. Finally, the preferred

Size	Outer Layer	Hidden Layer	(80/10/10) Tes.	Val.	Tra.	(40/20/40) Tes.	Val.	Tra.	(10/10/80) Tes.	Val.	Tra.
50	logsig + MSE	logsig	0.77	1.19	0.86	3.53	3.61	3.31	2.30	5.38	5.50
		poslin	2.77	2.54	3.14	3.35	3.36	3.24	1.54	2.88	3.31
	softmax + CE	logsig	37.45	37.64	53.22	43.08	43.00	70.28	37.08	42.91	88.43
		poslin	10.15	11.69	11.47	13.49	12.53	14.15	4.20	5.54	7.23
100	logsig + MSE	logsig	3.61	3.34	3.41	3.35	3.33	3.36	7.40	8.54	8.00
		poslin	10.03	8.73	9.18	6.31	6.75	6.20	5.67	5.19	5.04
	softmax + CE	logsig	**43.75**	45.41	62.05	**47.64**	47.25	74.09	**39.37**	44.83	85.54
		poslin	9.77	10.88	11.04	7.05	7.17	7.51	5.89	9.15	10.00
200	logsig + MSE	logsig	2.19	1.85	2.22	3.30	3.63	3.42	4.59	4.19	5.38
		poslin	9.50	9.23	9.04	9.19	9.52	8.78	5.80	5.04	6.11
	softmax + CE	logsig	42.10	40.48	56.25	35.33	34.93	56.33	32.31	41.21	77.74
		poslin	2.15	2.42	2.12	0.52	0.46	0.49	1.92	2.08	2.61
500	logsig + MSE	logsig	4.04	4.77	4.49	2.36	2.11	2.06	6.92	8.07	7.07
		poslin	9.57	9.38	9.07	8.91	9.65	9.16	9.13	10.92	11.46
	softmax + CE	logsig	28.68	30.37	38.01	30.42	32.51	42.53	26.95	31.91	57.21
		poslin	0.88	0.73	0.77	2.37	2.86	2.60	1.01	1.15	1.50
(50, 100)	logsig + MSE	logsig	29.64	28.37	31.70	25.34	25.36	27.88	20.39	21.34	27.30
		poslin	9.07	9.53	9.35	9.45	9.07	9.45	9.26	9.92	11.42
	softmax + CE	logsig	0.81	0.85	0.53	3.79	3.77	3.34	0.53	0.50	0.73
		poslin	0.31	0.65	0.54	0.85	0.73	0.94	0.67	0.62	0.81
(50, 100, 50)	logsig + MSE	logsig	23.61	22.80	24.09	20.43	20.57	21.78	19.11	19.11	21.03
		poslin	27.49	25.61	28.29	25.04	26.18	27.15	13.64	14.30	14.96
	softmax + CE	logsig	7.34	7.88	7.86	9.54	9.05	8.94	5.64	8.34	8.57
		poslin	4.81	4.38	4.23	5.15	4.77	4.81	3.01	7.57	6.96

Table 3: Summary of all the configuration runs done. *Tes.* represents the mean test accuracy, while *Val.* represents the mean accuracy obtained in the validation set, and *Tra.* the mean accuracy obtained in the training set. All in %. Best configuration of each partition in bold.

architecture seems to be one layer with not that many nodes in the layer, around 100-200 nodes seems best, and more leaning on the latter. Nevertheless, more exploration inside the range of 50-250 nodes could be done to find the best network. Finally, most of our configurations are overfitting, due to the lenient early stopping parameters we set. We have tried running more experiments reducing the overfitting, however it is still quite prevalent as we increase the size of the architecture and reduce the training set. More data would be required to reduce it or other techniques such as regularization, dropout, or data augmentation of the images (geometric transformations of the current image set).

We have some overall comments of the observed. *1)* using the GPU to train made it much faster, however, some of the accuracy values diminished significantly compared to training on the CPU. Where some runs reached up to 60% mean test accuracy, however, this discrepancy is probably due to the optimizer. *2)* much of the overfitting could have been avoided by reducing some of the parameters as explained in section 2, however, we feel the purpose of the practical is to explore how to use neural networks more than to obtain the best possible accuracy. If the former was our objective, a more detailed search of the parameters should have been done, expanding the grid search from Table 3, to include the optimizer parameter search. Nevertheless, there are more advanced neural networks techniques, such as convolutional neural networks, that work much better on image data and these should be explored before trying to fine tune the network so much. *3)* to our surprise (10/10/80) partition results in pretty high test accuracy. This is probably due to the "memorization capacity" where it basically learned a dictionary-like mapping of the data. Which is not a positive quality for generalizing, however, since silhouettes are pretty similar between classes, worked on this dataset. *4)* the 48x3 experiments from Table 3 takes around 3.5 hours on the GPU to compute.