Coursework 3

Baseline Experiments on CIFAR-10 and CIFAR-100

Niki Stavropoulou s1641718 | Machine Learning Practical | Ferbruary 2017

1 CONTENTS

2 RESEARCH QUESTIONS

2.1 CIFAR-10 AND CIFAR-100 DATASETS

CIFAR-10 and CIFAR-100 are two image classification datasets quite more complex than the MNIST dataset we were using so far. CIFAR-10 consists of 6000 images for the 10 different classes, with a total of 60000 overall size, while CIFAR-100 has the same size and divides the 10 classes of images to 100 finer ones. In addition, there are now three color channels (RGB) per image. Following the same approach as for the MNIST data set on the previous coursework, before implementing complicated methods, in this coursework I will first try to define a baseline architecture that is appropriate.

2.2 BASELINE ARCHITECTURE

To define a baseline system, I am experimenting using feed-forward networks and testing different parameters for the initialization as well as the architecture of the model itself. The first question that arises is for which width and depth the hidden layers produce better results. To answer, I am testing both CIFAR-10 and CIFAR-100 with one to four hidden layers. Then I keep the number with the better performance and compare the performance of hidden layer with 50 to 500 units, noting the effect on time and performance. I use the most effective combination of number and size for the rest of the experiments. The next question is finding a learning schedule, with the appropriate learning rate. For this, I am used the standard stochastic gradient descent, momentum training, Adagrad and Adam TensorFlow optimizers. After, I tested the nonlinearities. I compared the three activation functions we experimented with on the second coursework, the sigmoid, tanh and relu functions. Finally, I am experimented with L2 regularization and also dropout layers and noting how the model performs.

3 METHODS AND TENSORFLOW IMPLEMENTATION

To start the experiments, I altered the code given on lab 9. Using TensorFlow, we create a graph to compute predicted outputs. After creating instances of CIFAR-10 and CIFAR-100 data providers, I initialize the layers of the model. For this I use the `fully_connected_layer` function, were, given a layer as input, the weights, randomly initialized, and the biases, initialized to zero, are used to output another layer, according to a specific nonlinearity. To do this, the `tf.Variable` constructor and the operation `tf.matmul` are used, as well as the different nonlinearities provided by TensorFlow. The module `softmax_cross_entropy_with_logits` combines the output with a cross entropy error function during the training. To calculate the accuracy, `tf.argmax` gives the maximum class probability. Both error and accuracy are calculated per data point, so they need to be averaged with `tf.reduce_mean`. After defining these, `tf.train` is called to start the training session, for a chosen number of epochs (iterations) and using an optimizer to define the

learning schedule. In this part I will explain which method calls I used; more details on the experiments and the actual results will be discussed in the following part.

3.1 HIDDEN LAYERS

Specifically, I experimented for defining two characteristics of the hidden layers: depth and width. First, I created and trained four different models, using one to four hidden layers respectively. For this part, each hidden layer had a default number of units 200 and standard stochastic gradient descent with a rate of 0.5 was used. The `fully_connected_layer method` was called to create the output layer of the first that is passed as input for the next and so-on. For all models, the non-linearity used on the method is changed to the sigmoid function (`nn.sigmoid`). After plotting the results, I chose the number of hidden layers for the rest of the experiments. Then, I changed the number of units in the hidden layer (the `num_hidden` value) to define the width of the model. After training the model, the combination that produced better results was used as a baseline for the next experiments.

3.2 LEARNING RATE SCHEDULES

In this part, I experimented using different optimizers to compute the gradients and their updates for the `train_step`. The learning schedules used are:

- the `GradientDescentOptimizer` for rates of 0.5 and 0.1
- the `MomentumOptimizer` for a rate of 0.01. and momentum of the default 0.9, since during coursework 1 this combination produced better results
- the `AdagradOptimizer` for rates 0.1 and 0.01, again chosen based on the previous coursework
- the `AdamOptimizer` for rates 0.01, 0.001 and 0.0001

After the experiments, I chose the optimizer that I used for the rest of the experiments, according to the results.

3.3 ACTIVATION FUNCTIONS

Then, I changed the non-linear transformation function that is used before returning the output layer in the `fully_connected_layer` method. The layers on the experiments so far were computed using `tf.nn.sigmoid`. I also tested `tf.nn.tanh` and `tf.nn.relu`, whose performance was evaluated during the previous coursework, and chose the nonlinearity to continue with in the next part.

3.4 REGULARIZATION AND DROPOUT

3.4.1 L2 Regularization

For the next part, I tested by adding a regularizing factor in the error computation. To do that, I altered the `fully_connected_layer` method, so that it returns a regularizer, using TensorFlow' s `tf.nn.L2_loss` module, computed on the respective weights of each layer. The

regularizers were added to the error computation. Here I should note that for the validation set I kept the previous error computation, a not regularized one.

3.4.2 Dropout

Finally, I altered the `fully_connected_layer` method again, so that is will now add a dropout layer, given a specific probability. To do that `tf.nn.dropout(targets, prob)` is used, that will add an input layer to the targets given a probability. The new output layer after the dropout is the one the method `fully_connected_layer` now returns. My final output layer is not a dropout one.

3.4.3 Combination

As a final experiment, I combined the previous two methods, adding a dropout layer with the better probability and L2 penalty from the previous parts.

4 EXPERIMENTS AND RESULTS

4.1 HIDDEN LAYERS

As mentioned above, the first experiments were conducted to determine the width and depth for the layers of the following experiments.

4.1.1 Depth

Firstly, I added one to four hidden layers with 200 units, with a sigmoid transformation and gradient descent with 0.5 rate for the learning and trained on the CIFAR-10 dataset for 10 epochs. The results are shown in Table 1 and are plotted in Figure 1. Having two hidden layers achieved higher training and validation accuracy and lower error. I retrained the models for 50 epochs. Now, the model with the three hidden layers resulted in approximately the same accuracy and error values, but the training did become slower. The results of the training for 50 epochs are shown in Table 2 and Figure 2.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
1	1.5944	1.6317	0.4306	0.4237
2	1.4490	1.5577	0.4852	0.4447
3	1.5261	1.5740	0.4514	0.4359
4	1.6620	1.6846	0.3965	0.3908

Table 1: (CIFAR-10) Training with different hidden layer number, 10 epochs

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
1	1.8712	1.7337	0.4591	0.4518
2	1.0757	1.4799	0.6132	0.4934
3	1.1824	1.4739	0.5737	0.4993
4	1.2520	1.4950	0.5495	0.4857

Table 2: (CIFAR-10) Training with different hidden layer number, 50 epochs

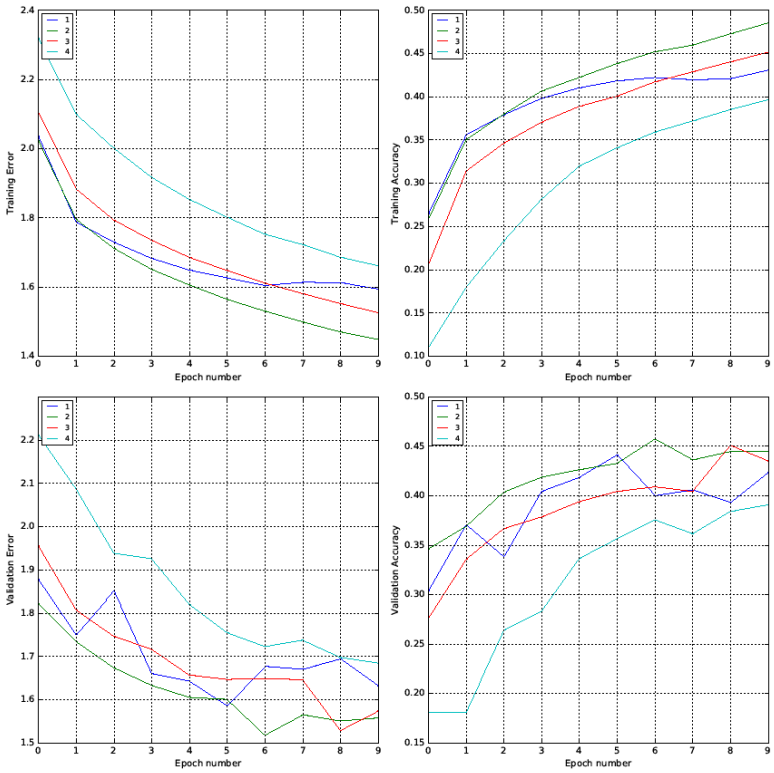

Figure 1: (CIFAR-10) Training with different hidden layer number, 10 epochs

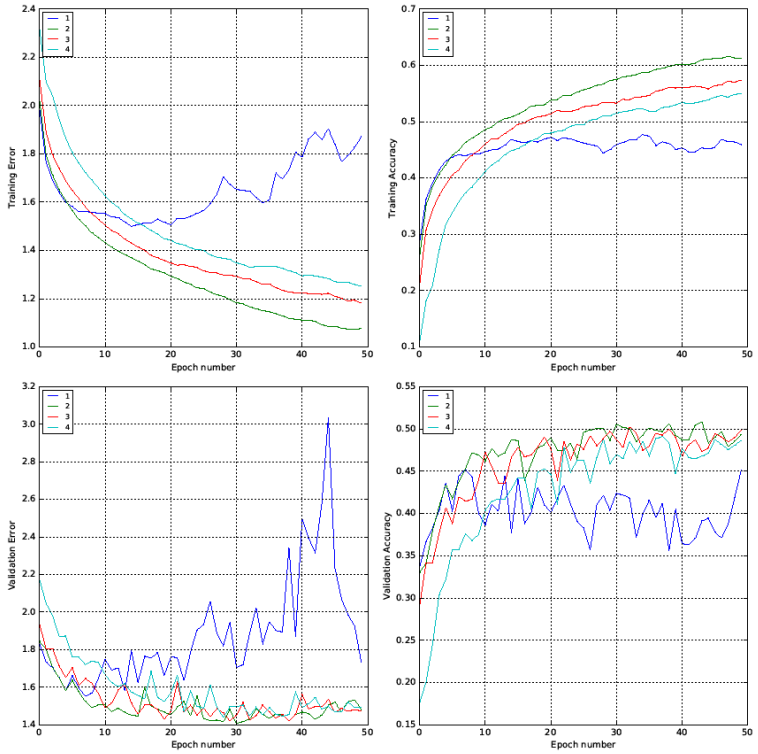

Figure 2: (CIFAR-10) Training with different hidden layer number, 50 epochs

I repeated the same process for the CIFAR-100 dataset, only training for 50 epochs. This time the model with two hidden layers reached better values of the rest. The results are accumulated and plotted in Table 3 and Figure 3.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
1	2.6349	3.6551	0.3362	0.1976
2	2.2983	3.3978	0.4024	0.2277
3	2.7453	3.4145	0.3027	0.2071
4	3.1201	3.4083	0.2291	0.1923

Table 3: (CIFAR-100) Training with different hidden layer number, 50 epochs

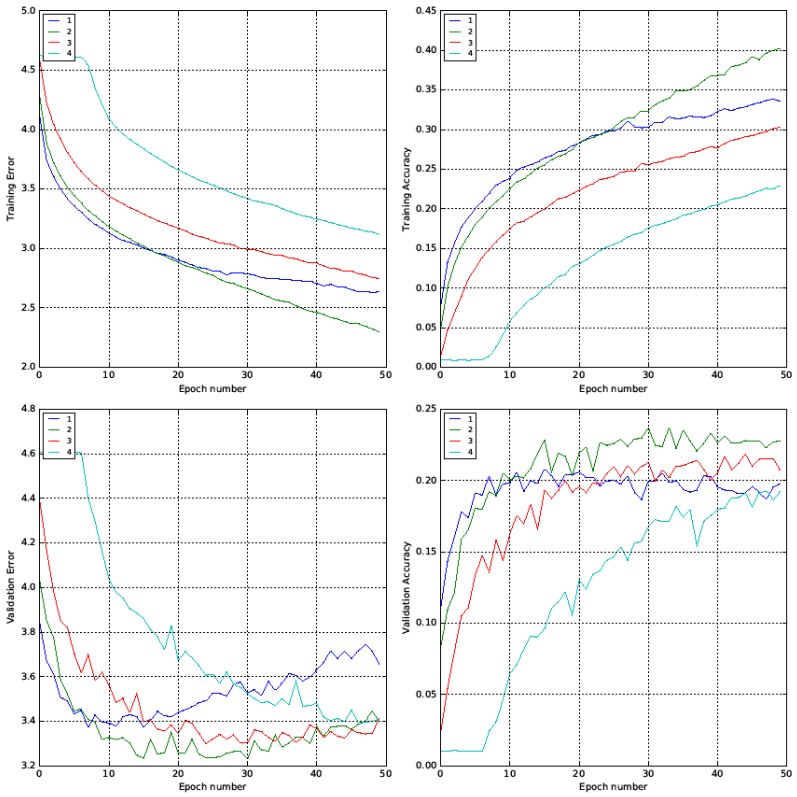

Figure 3: (CIFAR-100) Training with different hidden layer number, 50 epochs

From the above, I chose to continue training with a model with two hidden layers on both datasets.

4.1.2 Width

After having decided on the number of layers, I experimented with the number of hidden units in one. On the CIFAR-10 set, I initialized the models with 50, 100, 200, 300, 400 and 500 units. As expected, the more units the layers had the better the final values it reached were. I trained both for 10 and 50 epochs again, using gradient descent of rate 0.5 and the sigmoid transformation. The results are shown in Table 4, 5 and Figure 4. The values

looked similar, with 300 units per layer for 10 epochs having better results, but for 50 epochs the plot shows that the 500 unit model reached higher results faster.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
50	1.5793	1.6068	0.4321	0.4251
100	1.4943	1.5235	0.4618	0.4531
200	1.4490	1.5577	0.4852	0.4440
300	1.4448	1.4528	0.4567	0.4920
400	1.4620	1.4756	0.4752	0.4757
500	1.4488	1.4818	0.4823	0.4786

Table 4: (CIFAR-10) Training with different hidden layers' units, 10 epochs

50 epochs	error (train)	error (valid)	accuracy (train)	accuracy (valid)
50	1.2986	1.5216	0.5332	0.4700
100	1.2159	1.4822	0.5635	0.4794
200	1.0757	1.4799	0.6132	0.4934
300	0.9934	1.5950	0.6463	0.4874
400	0.8795	1.6081	0.6835	0.5054
500	0.8447	1.6437	0.5976	0.4759

Table 5: (CIFAR-10) Training with different hidden layers' units, 50 epochs

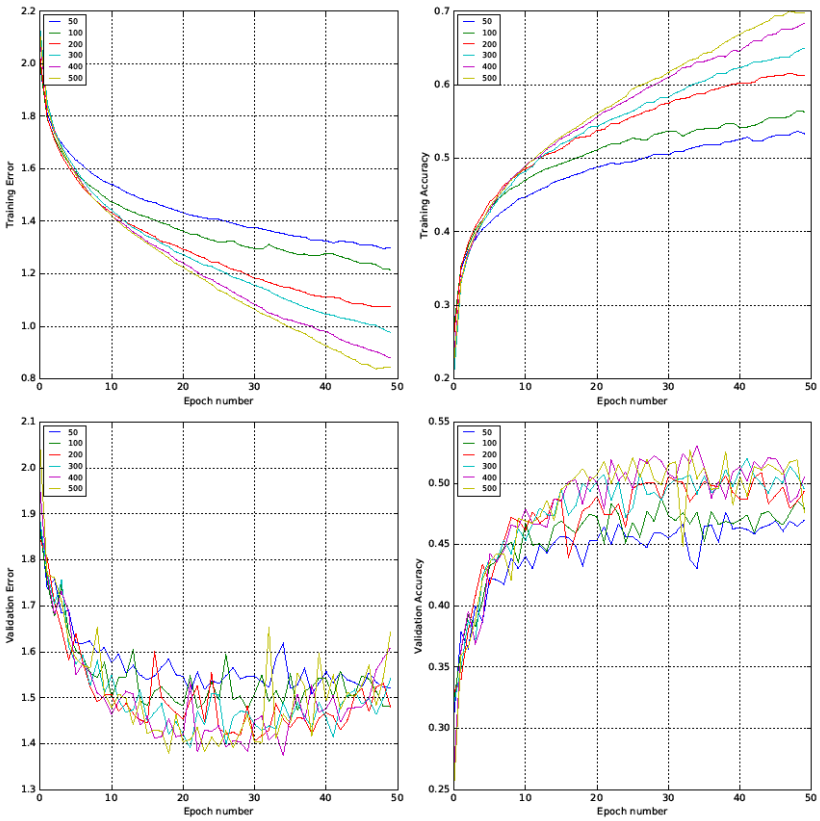

Figure 4: (CIFAR-10) Training with different hidden layers' units, 50 epochs

I repeated again the process on the CIFAR-100 dataset only for 50 epochs. This time, it is obvious in Table 6 and Figure 5 that the 500 unit layers gave better accuracy and error values on both training and validation processes.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
50	3.1421	3.4884	0.2331	0.1794
100	2.8413	3.3958	0.2908	0.2042
200	2.2983	3.3978	0.4024	0.2277
300	0.8256	3.5840	0.5108	0.2341
400	1.5022	3.6922	0.5966	0.2384
500	1.2919	3.7109	0.6541	0.2417

Table 6: (CIFAR-100) Training with different hidden layers' units, 50 epochs

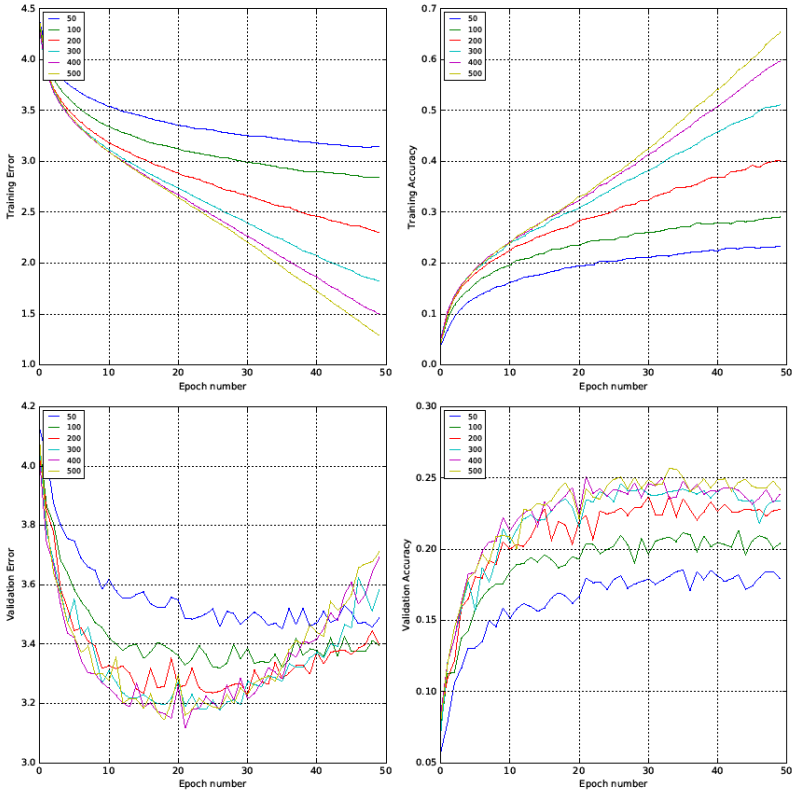

Figure 5: (CIFAR-100) Training with different hidden layers' units, 50 epochs

4.1.3 Conclusion

The number of hidden layers had a big part in the final values the models reached. For more layers, the model became unstable, as is seen from the learning curves, and the training was significantly slower. But, having only one layer, or even no hidden layer was out of the question when considering the results; the final values were low, and the validation plot curves show that the model starts to overfit, since it cannot adapt as well as the ones with more layers. Furthermore, as was expected, a bigger number of units resulted in better validation values for the bigger dataset, since having 100 classes to

classify is more complex than 10. But, again the validation error plot shows that after some epochs the error value for the bigger number of units is increasing. This was somewhat expected, since on the training plots it is seen that the models converged faster. I continued the experiments with models with two hidden layers of 500 units.

4.2 LEARNING RATE SCHEDULES

On the next experiments I tried to establish a learning schedule baseline. For the CIFAR-10 dataset I trained a model with two hidden layers of 500 units with standard gradient descent of rates 0.5 and 0.1, given the results of previous coursework. I also tried momentum learning with rate of 0.01 and momentum 0.9. After, I used Adagrad for rates 0.1 and 0.01, and finally the Adam optimizer for rates 0.01, 0.001 and 0.0001. The results are shown in Table 7 and Figure 6.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
Gradient descent 0.5	0.8447	1.6437	0.6976	0.4759
Gradient descent 0.1	1.0858	1.3917	0.6722	0.5214
Momentum 0.01	0.9794	1.4140	0.6490	0.5303
Adagrad 0.1	0.7805	1.4583	0.7264	0.5285
Adagrad 0.01	1.4004	1.4851	0.5092	0.4801
Adam 0.01	2.2691	2.1847	0.1470	0.1551
Adam 0.001	1.1259	1.5110	0.5926	0.4831
Adam 0.0001	1.0149	1.3551	0.6420	0.5196

Table 7: (CIFAR-10) Training with different optimizers, 50 epochs

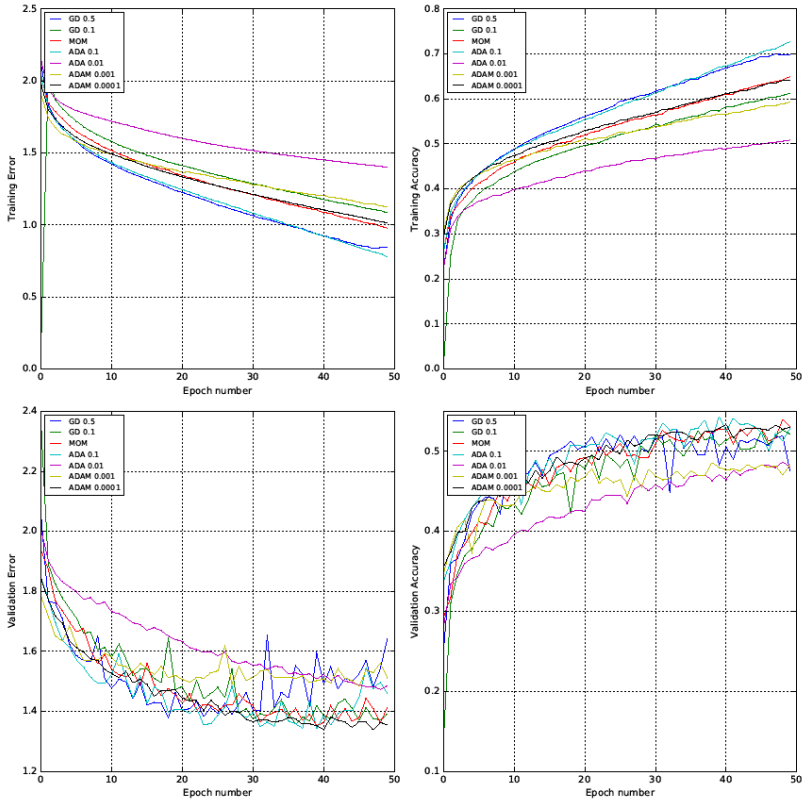

Figure 6: (CIFAR-10) Training with different optimizers, 50 epochs

From the learning curves and the final values, Adagrad with 0.01 rate and Adam with 0.0001 rate performed better on the training set. But, on the validation set, the learning curves for momentum training, Adagrad and Adam were better.

Repeating the process for the CIFAR-100 set, the results of the training over 50 epochs are shown in the Table 8 and Figure 7. Again, Adagrad and Adam gave better curves and values on the training set, but momentum surpasses Ada's performance on the validation.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
Gradient descent 0.5	1.2919	3.7109	0.6541	0.2417
Gradient descent 0.1	2.6531	3.0846	0.3348	0.2506
Momentum 0.01	2.6386	3.1047	0.3370	0.2492
Adagrad 0.1	1.8087	3.2233	0.5328	0.2546
Adagrad 0.01	3.4130	3.4787	0.2014	0.1889
Adam 0.01	4.6622	4.6685	0.0096	0.0099
Adam 0.001	1.8979	3.4971	0.5019	0.2265
Adam 0.0001	2.7677	3.1504	0.3178	0.2438

Table 8: (CIFAR-100) Training with different optimizers, 50 epochs

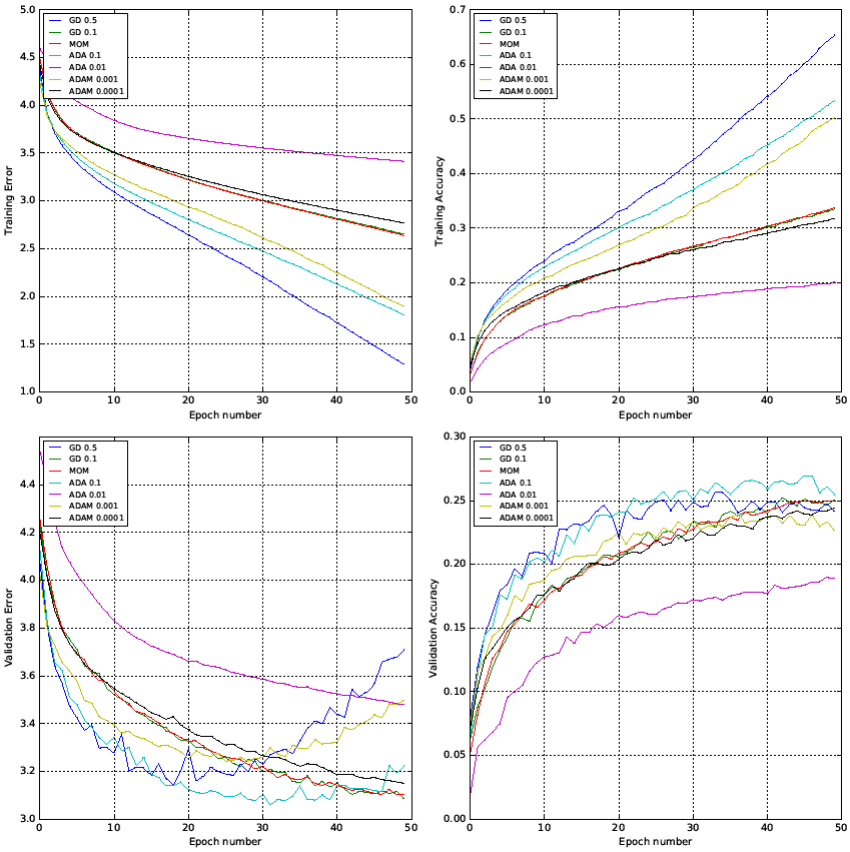

Figure 7: (CIFAR-100) Training with different optimizers, 50 epochs

4.2.1 Conclusion

Overall, using momentum, Ada with 0.01 rate and Adam with the default rate gave good results across both datasets. The model using Adam optimizer was slightly slower during the training than using the others. When training on the CIFAR-100 set with Adagrad for rate 0.01, the values were similar to those of Adam with a rate of 0.01. But, their learning curves hint at overfitting, as was expected on the bigger dataset. Momentum training also produced good final values, close to those of using the Adam optimizer for the default (0.0001) rate. Given the learning curves plotted above and taking into consideration the fact that Adam and Ada showed a faster convergence, for the next experiments I kept the default Adam optimizer. In retrospect, momentum or Adagrad optimizers might perform better as well. But, the problem with Adagrad remains; after many repetitions the learning rate will eventually start diminishing, until the model will be not learning efficiently.

4.3 Activation functions

The next experiments were conducted to determine which non-linear transformation is better for the layers. On the CIFAR-10 set, I trained the model with the two hidden layers of 500 units, using Adam optimizer, with the ReLu function, the standard sigmoid and the tanh one. The results are shown in Table 9 and Figure 8.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
Sigmoid	1.0149	1.3551	0.6420	0.5196
Tanh	0.5012	1.4524	0.8441	0.5448
Relu	0.4730	1.6637	0.8456	0.5364

Table 9: (CIFAR-10) Training with different nonlinear functions, 50 epochs

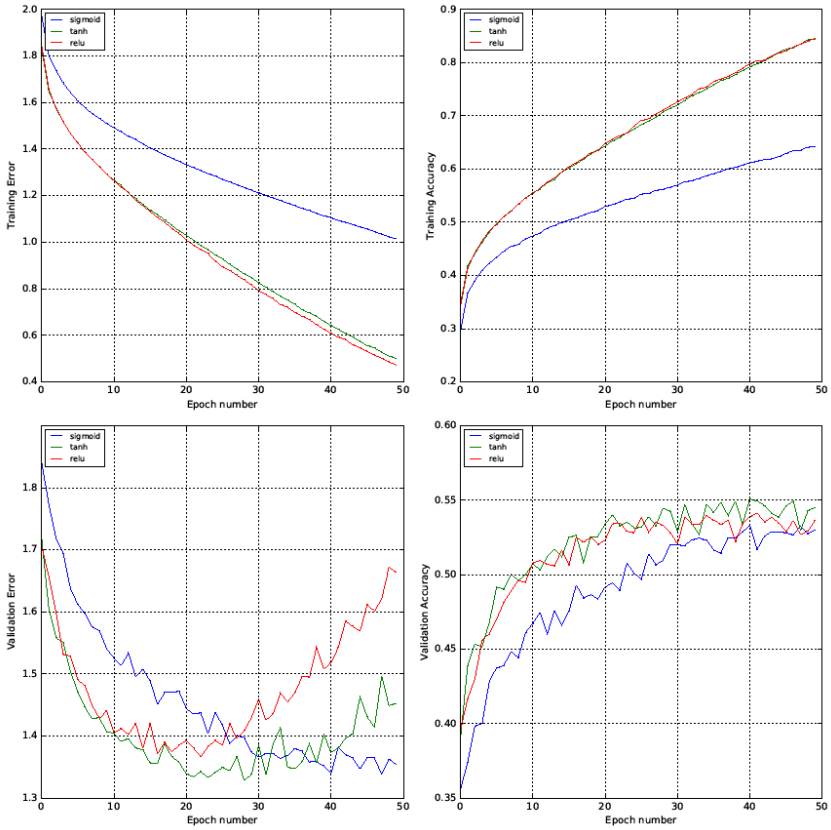

Figure 8: (CIFAR-10) Training with different nonlinear functions, 50 epochs

Tanh and ReLu functions gave better final values on the training set, surpassing the sigmoid' s previous ones. But, the plots of the validation set show that the ReLu model tends to overfit, even if it converged faster.

Repeating the training for the CIFAR-100 dataset for the same model, I accumulate the results in Table 10 and plot them on Figure 9. They show similar results to the model for the CIFAR-10 dataset, with ReLu and Tanh giving better final values, but the sigmoid has a more stable behavior.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
Sigmoid	2.7677	3.1504	0.3178	0.2438
Tanh	1.6026	3.3349	0.6097	0.2438
Relu	1.4250	3.6140	0.6421	0.2485

Table 10: (CIFAR-100) Training with different nonlinear functions, 50 epochs

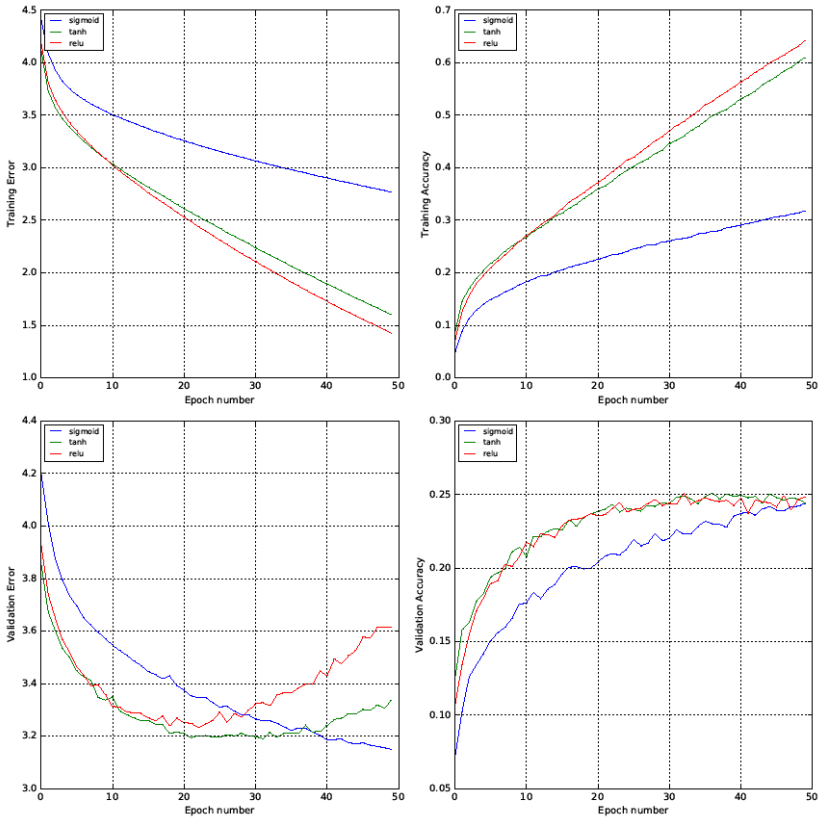

Figure 9: (CIFAR-100) Training with different nonlinear functions, 50 epochs

4.3.1 Conclusion

For both datasets, ReLu and tanh functions lead to a faster convergence and seemed to produce better results. The sigmoid is always the one with the lower final values, lower accuracy and higher error on the training set. But, on the validation set the sigmoid' s results were far better. When taking into account the plots ReLu and tanh' s performance tends to drop after reaching their max values. In contrast, sigmoid is slower but on the validation sets was far more stable. I will choose the ReLu function for the next experiment given its convergence speed, but a model using the sigmoid could be more reliable (robust) for training for more epochs.

4.4 Regularization and Dropout

The final questions to be answered were how much regularizing and dropout would affect the results. Using the previous models, and altering the fully_connected_layer method as was described in the previous section, with two hidden layers of 500 units and ReLu transformation, I experimented with adding L2 regularizers, for 0.01, 0.001 and 0.0001 over 50 epochs. Then, I added a dropout layer with probability 0.5 and 0.9. Finally, I combined the two, having a 0.001 L2 penalty and 0.9 probability dropout layer.

The results for the training of CIFAR-10 dataset are accumulated in Table 11 and plotted on Figure 10. For a low probability like 0.5, as was expected the model performed the worst. For 0.9, though, the results achieved were on par with those of adding 0.0001 L2 penalty. In fact, as can be seen, this model (0.0001 L2) achieved the best results of all in the training sets. On the validation set it was 0.001 L2 loss that gave slightly better values. The combination L2 and dropout model's values were average.

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
0.01 L2	1.8424	1.6116	0.4474	0.4310
0.001 L2	1.2650	1.3118	0.6620	0.5502
0.0001 L2	0.7289	1.5140	0.8111	0.5426
Dropout 0.5	1.9631	1.9930	0.2917	0.2770
Dropout 0.9	0.9293	1.5706	0.6705	0.5016
L2 - Dropout	1.4543	1.4229	0.5656	0.4992

Table 11: (CIFAR-10) Training with L2 regularization and dropout, 50 epochs

Repeating the same experiments but for the CIFAR-100 dataset, I ended with the values in Table 12, and the plot on Figure 11. The results were similar to the CIFAR-10 dataset's models. The worst values had the model with a dropout layer probability 0.5, while the best final values were those on 0.0001 and 0.001 L2 loss on both training and validation. The combination was average in respect these.

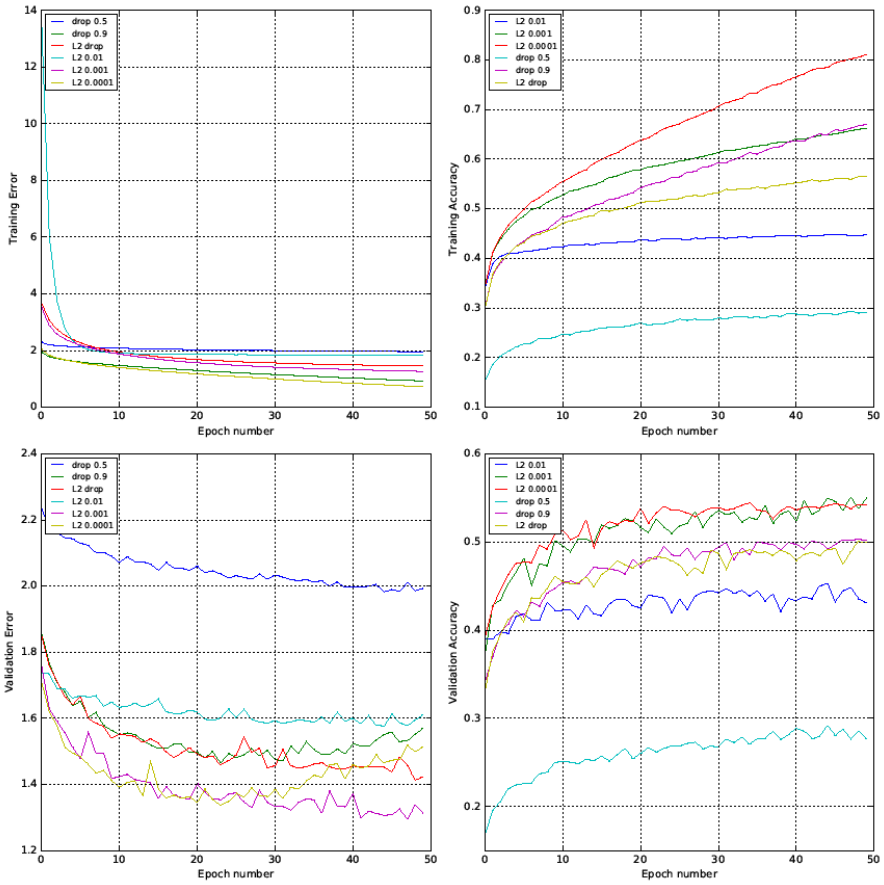

Figure 10: (CIFAR-10) Training with L2 regularization and dropout, 50 epochs

	error (train)	error (valid)	accuracy (train)	accuracy (valid)
0.01 L2	4.1099	3.7340	0.1393	0.1348
0.001 L2	3.1575	3.0645	0.3548	0.2567
0.0001 L2	1.9421	3.3706	0.5703	0.2515
Dropout 0.5	4.2640	4.2922	0.0741	0.0729
Dropout 0.9	2.3699	3.5665	0.4220	0.2240
L2 - Dropout	3.4423	3.321	0.2794	0.2258

Table 12: (CIFAR-100) Training with L2 regularization and dropout, 50 epochs

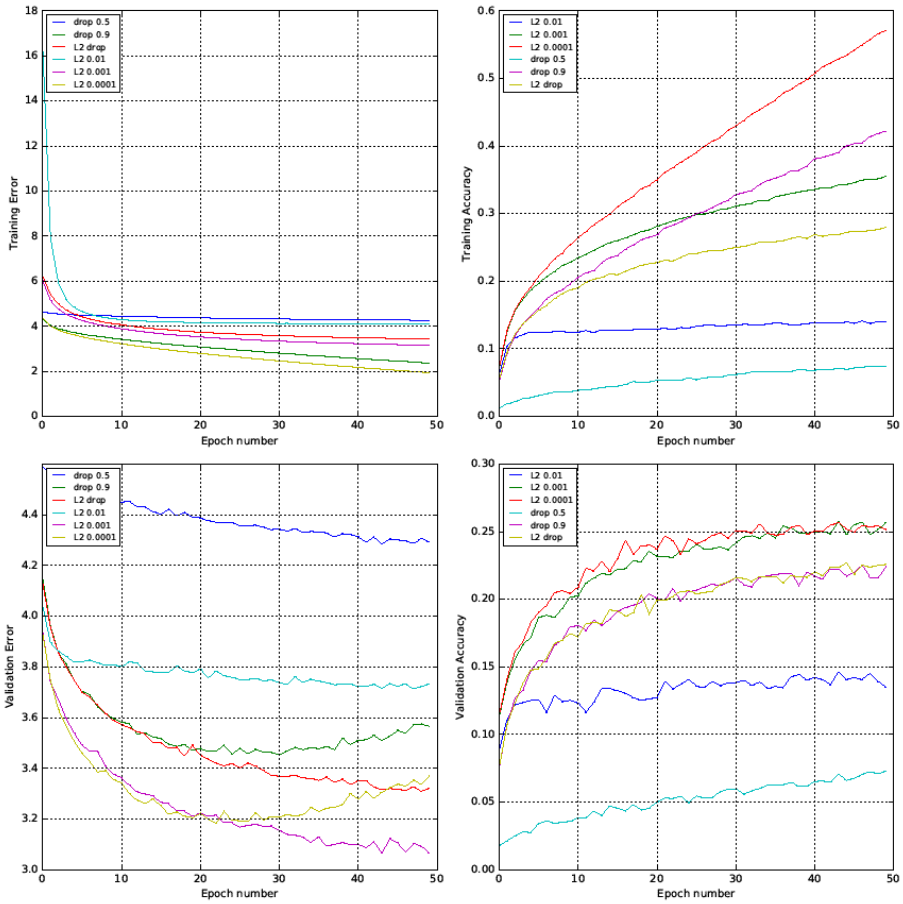

Figure 11: (CIFAR-100) Training with L2 regularization and dropout, 50 epochs

4.4.1 Conclusion

As was expected from, a low probability dropout layer performed the worst. Adding a loss of 0.0001 L2 to the error function, though, gave the best results on the training sets. It should be noted though, from the plots of both CIFAR-10 and CIFAR-100 validation sets, that the 0.0001 L2 model shows signs of overfitting (error plot). So, the 0.001 L2 model would be preferred. It should also be noted here that adding the loss function significantly slowed the training process. Training the L2 models for 50 epochs reached a 2-hour runtime per model, taking an average of 15 to 20 minutes per epoch to compute.

4.5 Conclusion

With these experiments I tried to establish a model that would give better final values during the training process, meaning higher accuracy and lower error values but taking into account the convergence speed. I compared different architectures and initializations, using the knowledge from the previous coursework. But, the two datasets are different from the MNIST dataset we were previously experimenting on and significantly more complicated. The final values achieved on the previous coursework compared to the best I could achieve here using familiar models prove this. Furthermore, I used the same models

for experimenting on both the CIFAR-10 and CIFAR-100 dataset; this was not a correct approach. The two datasets are different in themselves. CIFAR-100 is way more complicated and the results attest to that.

5 FURTHER WORK

For this coursework I experimented training with a feed forward network. On the next coursework I want to use more complicated networks, specifically convolutional networks. There are many things to explore, using different pooling functions, pooling layers, using different sizes and numbers of feature maps, starting from the baseline we used for MNIST and trying to define a new better one. But, the computational cost of CNNs is definitely not trivial and must be taken into account for timetable of the experiments.