Deep Learning - Assignment 1

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

In this assignment three neural network models will be tested in the CIFAR10 dataset, which contains images of size 32x32 pixels in RGB format. It shows how the performance significantly improves in images when using CNN networks instead of basic Perceptrons.

Task 1

Firstly, I run experiments changing parameters individually to find what specific modifications improve the accuracy of the model. Then, I made different combinations of the best results to find the best model.
From the individual experiments, I found that the following settings improved the model:

- Decreasing the learning rate to a number around 0.001
- Incrementing the number of iterations
- Increasing the number of hidden neurons to 200
- Having a small weight regularization parameter

While these settings made the model worse:

- Increasing the learning rate
- Doing more than 5000 iterations, as it starts overfitting the model
- Increasing the number of hidden layers
- Having a big (>0.001) weight regularization parameter

For the weight scale parameter, no clear conclusion was found, as it affected the Perceptron in misleading ways, although it generally improved when changed from it's default value (1e-4).

Submitted to 30th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

Table 1: Results on Multi-Layer Perceptron in NumPy

	Parameters					
Experiment with...	Steps	Hidden units	Learning rate	Weight Scale	Weight Regl.	Model Accuracy (Test data)
Default params.	1500	100	2e-3	1e-4	0	0.4099
Iterations	3000	100	2e-3	1e-4	0	0.4604
	4000	100	2e-3	1e-4	0	0.479
	5000	100	2e-3	1e-4	0	0.4597
	6000	100	2e-3	1e-4	0	0.4243
Hidden units	1500	100,200,100	2e-3	1e-4	0	0.1
	1500	200,100	2e-3	1e-4	0	0.1
	1500	**200**	2e-3	1e-4	0	0.4638
	1500	300	2e-3	1e-4	0	0.4315
	1500	400	2e-3	1e-4	0	0.4243
Learning rate	1500	100	15e-4	1e-4	0	0.4763
	1500	100	12e-4	1e-4	0	0.4793
	1500	100	11e-4	1e-4	0	0.4823
	1500	100	**1e-3**	1e-4	0	0.4896
	1500	100	7e-4	1e-4	0	0.4889
	1500	100	5e-4	1e-4	0	0.4835
Weight Scale	1500	100	2e-3	5e-4	0	0.4341
	1500	100	2e-3	1e-3	0	0.4376
	1500	100	2e-3	2e-3	0	0.4189
	1500	100	2e-3	**1e-5**	0	0.4455
	1500	100	2e-3	1e-6	0	0.4447
Weight Regl.	1500	100	2e-3	1e-4	0.1	0.2456
	1500	100	2e-3	1e-4	0.01	0.3977
	1500	100	2e-3	1e-4	1e-3	0.3945
	1500	100	2e-3	1e-4	**1e-4**	0.4199
	1500	100	2e-3	1e-4	1e-5	0.4149
All parameters	3000	200	1e-3	1e-4	0	0.5099
	4000	200	1e-3	1e-4	0	0.5149
	5000	200	1e-3	1e-4	0	0.5152
	6000	200	1e-3	1e-4	0	0.5129
	3000	200	1e-3	1e-5	1e-4	0.5034
	4000	200	1e-3	1e-5	1e-4	0.5082
	5000	**200**	**1e-3**	**1e-5**	**1e-4**	**0.5216**
	6000	200	1e-3	1e-5	1e-4	0.5168

Task 2

For the experimentation of this task, the same strategy is done. Only in this case, the regularization parameters (Weight Regl and Dropout rate) will be used with more iterations than in the default case. By doing this, it is expected to see how well this regularization methods do to prevent overfitting. When trying different hidden layers setups (the other parameters in default), the model did not improve and always stayed at a 0.1 accuracy. But, after changing the initialization to xavier, the model showed a really good improvement. Specially when creating a symmetric set of layers. Also, the more hidden units per layer the more it seemed to improve, as it can be seen in Table 2. Besides this, the accuracy obtained in xavier and uniform initializations was very small, and were therefore not used in future experiments.

Table 2: Hidden Layer performance depending on weights initialization (all other parameters set to default. Default parameters in Table 3)

Parameters			
Experiment...	Weight Initialization	Hidden units	Model Accuracy (Test data)
Default params.	normal	100	0.4581
Weight Initialization	xavier	100	0.3015
	uniform	100	0.2150
normal with Hidden layers structures	normal	100,100	0.1000
	normal	50,100,50	0.1000
	normal	100,200,100	0.1000
	normal	200,400,200	0.1000
	normal	100,200,400,200,100:	0.1000
xavier with Hidden layers structures	xavier	100,100	0.2526
	xavier	50,100,50	0.1973
	xavier	100,200,100	0.3065
	xavier	200,400,200	0.3529
	xavier	400,800,400	0.3935
	xavier	500,1000,500	0.3999
	xavier	100,200,400,200,100:	0.3736

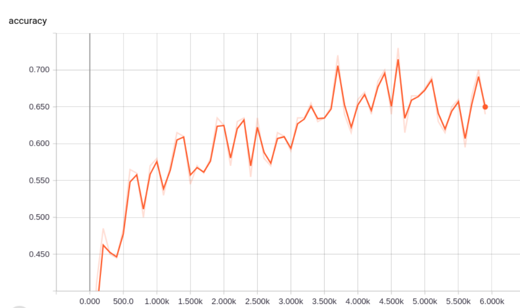

Figure 1: Accuracy on the train data. (Task 2)

Task 3

In this task, due to computational restrictions, first some test where done on less iterations than the whole model (1500) to see how well the models improved depending on the parameters settings. When running the model on default settings with Gradient Descent Optimizer, I got a test accuracy of 0.3458. Then, after changing to ADAM optimizer it went up to 0.4836. I run the whole model and got a final test accuracy of 0.7052. After that, I applied Batch normalization on the dense layers, as many websites encouraged and got a slight result when running 1500 iterations (0.4931). When running 15000 iterations the accuracy went up to 0.7250. I felt running the whole model with batch normalization on the pooling layers as well could have improved more the model, but considering that my machine took too long to run the whole model, and SURFsara has been busy during the last hours, I could not consider doing that experiment.

Table 3: Results on Multi-Layer Perceptron in TensorFlow

	Parameters							
Experiment...	Steps	Hidden units	Learn. rate	Optimizer	Activation	W. Regl.	Dropout	Model Accuracy (Test data)
Default params.	1500	100	2e-3	sgd	relu	l2	0.	0.4581
Iterations	**4000**	100	2e-3	sgd	relu	l2	0.	0.4796
	5000	100	2e-3	sgd	relu	l2	0.	0.4424
Hidden units	1500	200	2e-3	sgd	relu	l2	0.	0.4424
	1500	**300**	2e-3	sgd	relu	l2	0.	0.4547
	1500	500	2e-3	sgd	relu	l2	0.	0.4500
	1500	100,200,100	2e-3	sgd	relu	l2	0.	0.1000
Learning rate	1500	100	**1e-3**	sgd	relu	l2	0.	0.4952
	1500	100	5e-4	sgd	relu	l2	0.	0.4855
	1500	100	25e-5	sgd	relu	l2	0.	0.4455
	1500	100	1e-4	sgd	relu	l2	0.	0.3601
Optimizer	1500	100	2e-3	adadelta	relu	l2	0.	0.3698
	1500	100	2e-3	**adagrad**	relu	l2	0.	0.5106
	1500	100	2e-3	adam	relu	l2	0.	0.4011
	1500	100	2e-3	rmsprop	relu	l2	0.	0.2663
Activation	1500	100	2e-3	sgd	elu	l2	0.	0.4457
	1500	100	2e-3	sgd	tanh	l2	0.	0.3911
	1500	100	2e-3	sgd	sigmoid	l2	0.	0.3498
Weight Regl.	1500	100	2e-3	sgd	relu	none	0.	0.4487
	1500	100	2e-3	sgd	relu	l1	0.	0.4521
	6000	100	2e-3	sgd	relu	none	0.	0.4904
	6000	100	2e-3	sgd	relu	l1	0.	0.4879
	6000	100	2e-3	sgd	relu	l2	0.	0.4847
Dropout	1500	100	2e-3	sgd	relu	l2	0.1	0.4374
	1500	100	2e-3	sgd	relu	l2	0.2	0.4328
	1500	100	2e-3	sgd	relu	l2	0.3	0.4131
	1500	100	2e-3	sgd	relu	l2	0.5	0.3987
	5000	100	2e-3	sgd	relu	l2	0.2	0.4757
	6000	100	2e-3	sgd	relu	l2	0.2	0.4825
	7000	100	2e-3	sgd	relu	l2	0.2	0.4811
All parameters	5000	100	5e-4	sgd	relu	l2	0.	0.5032
	6000	300	5e-4	sgd	relu	l2	0.	0.5281
	6000	300	5e-4	adagrad	relu	l2	0.	0.5427
	7000	300	5e-4	adagrad	relu	l2	0.	0.5417
	6000	**300**	**1e-3**	**adagrad**	**relu**	**l2**	**0.**	**0.5458**
	7000	300	1e-3	adagrad	relu	l2	0.	0.5453
	6000	300	1e-3	adagrad	elu	l2	0.	0.5419

Conclusions

In this assignment it could be seen how hardly can be to create a neural network manually, while Tensorflow significantly speeds up the coding part for any type of Neural Network. It also shows how the computation for complicated models can be increased as the complexity grows, although improving as well the results of the models.
Regarding hyper parameters selection, this assignment helped to show how they affect the result on our neural networks, and thinking on how to plan and test them.

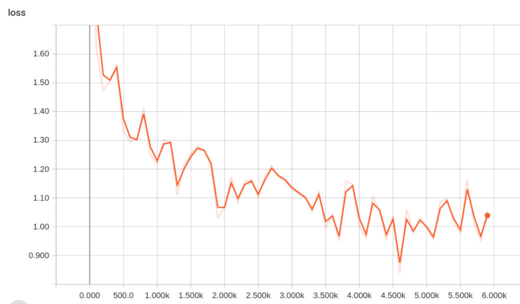

Figure 2: Loss on the train data. (Task 2)

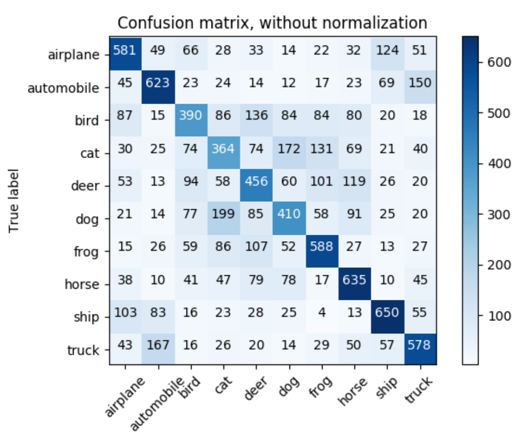

Figure 3: Confusion Matrix. (Task 2)