**Texas Tech University**

**Department of Computer Science**

**Course:** Introduction to Artificial Intelligence **Group:** 1

**Student:** Felipe Chavarria Fernandez **Email:** fchavarriafernandez@gmail.com

**Date:** Friday, July 12, 2019

Table of Contents

[Week 7 - Homework 7 1](#_Toc13785985)

[Practice 1 1](#_Toc13785986)

[Practice 2 2](#_Toc13785987)

[50/50 2](#_Toc13785988)

[60/40 2](#_Toc13785989)

[40/60 3](#_Toc13785990)

[70/30 3](#_Toc13785991)

[30/70 3](#_Toc13785992)

[Results 4](#_Toc13785993)

[Practice 3 4](#_Toc13785994)

[dropout\_rate = 0.1 4](#_Toc13785995)

[dropout\_rate = 0.3 5](#_Toc13785996)

[dropout\_rate = 0.5 5](#_Toc13785997)

[dropout\_rate = 0.7 5](#_Toc13785998)

[dropout\_rate = 0.9 5](#_Toc13785999)

[Random Forest 6](#_Toc13786000)

[Results 6](#_Toc13786001)

# Week 7 - Homework 7

## Practice 1

Change the training constants as follows:

# Training constants

n\_nodes\_l1 = 200

batch\_size = 32

n\_epochs = 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Optimizer** | **Number of Hidden Nodes** | **Max Test Accuracy** | **Max Train Accuracy** | **Min Test Cost** | **Min Train Cost** |
| Adagrad | 200 | 0.9830 | 1 | 0.06139 | 0.00198 |
| AdaDelta | 200 | 0.9835 | 1 | 0.06992 | 0.00027 |
| Adam | 200 | 0.9831 | 0.9991 | 0.08106 | 0.00439 |
| RMSprop | 200 | 0.9820 | 0.9999 | 0.08623 | 0.00107 |

From the results above, it looks like AdaDelta gave the best results overall, so no, RMSprop will not be ideal for this problem.

## Practice 2

# Training constants

total\_nodes = 300

batch\_size = 32

n\_epochs = 100

regularization\_scale = 0

### 50/50

total\_nodes = 300

n\_nodes\_l1 = round(total\_nodes \* 0.5)

n\_nodes\_l2 = total\_nodes - n\_nodes\_l1

Max test accuracy: 0.9836 at epoch: 26

Max train accuracy: 1.0000 at epoch: 20

Min test cost: 0.07610 at epoch: 3

Min train cost: 0.00027 at epoch: 99

### 60/40

total\_nodes = 300

n\_nodes\_l1 = round(total\_nodes \* 0.6)

n\_nodes\_l2 = total\_nodes - n\_nodes\_l1

Max test accuracy: 0.9835 at epoch: 45

Max train accuracy: 1.0000 at epoch: 20

Min test cost: 0.07668 at epoch: 2

Min train cost: 0.00027 at epoch: 99

### 40/60

total\_nodes = 300

n\_nodes\_l1 = round(total\_nodes \* 0.4)

n\_nodes\_l2 = total\_nodes - n\_nodes\_l1

Max test accuracy: 0.9825 at epoch: 23

Max train accuracy: 1.0000 at epoch: 21

Min test cost: 0.07896 at epoch: 4

Min train cost: 0.00027 at epoch: 99

### 70/30

total\_nodes = 300

n\_nodes\_l1 = round(total\_nodes \* 0.7)

n\_nodes\_l2 = total\_nodes - n\_nodes\_l1

Max test accuracy: 0.9847 at epoch: 52

Max train accuracy: 1.0000 at epoch: 15

Min test cost: 0.08121 at epoch: 7

Min train cost: 0.00027 at epoch: 99

### 30/70

total\_nodes = 300

n\_nodes\_l1 = round(total\_nodes \* 0.3)

n\_nodes\_l2 = total\_nodes - n\_nodes\_l1

Max test accuracy: 0.9826 at epoch: 98

Max train accuracy: 1.0000 at epoch: 23

Min test cost: 0.08298 at epoch: 3

Min train cost: 0.00054 at epoch: 99

### Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **split** | **Max Test Accuracy** | **Max Train Accuracy** | **Min Test Cost** | **Min Train Cost** |
| **50/50** | 0.9836 | 1 | 0.0761 | 0.00027 |
| **60/40** | 0.9835 | 1 | 0.07668 | 0.00027 |
| **40/60** | 0.9825 | 1 | 0.07896 | 0.00027 |
| **70/30** | 0.9847 | 1 | 0.08121 | 0.00027 |
| **30/70** | 0.9826 | 1 | 0.08298 | 0.00054 |

From the table above we can see that the best test accuracy was a split of 70/30 while the best min test cost was for the 50/50 split and from that point on the results got worse over.

Another thing to notice in this example is that if the second hidden layer has more nodes than the first hidden layer, the results provided are worse that its counterpart, example 70/30 provided better results than 30/70.

## Practice 3

# Training constants

n\_nodes = 1000

n\_layers = 3

batch\_size = 64

n\_epochs = 2000

### dropout\_rate = 0.1

Batch size: 64 Num batches: 375 Num epochs: 2000

Num layers: 3 total num nodes: 1000

Dropout rate: 0.1

Max test AUC of ROC: 0.7229 at epoch: 25

Max train AUC of ROC: 0.9855 at epoch: 1957

Min test cost: 0.47470 at epoch: 25

Min train cost: 0.12772 at epoch: 1946

Training time: 2345.180669784546

### dropout\_rate = 0.3

Batch size: 64 Num batches: 375 Num epochs: 2000

Num layers: 3 total num nodes: 1000

Dropout rate: 0.3

Max test AUC of ROC: 0.7278 at epoch: 90

Max train AUC of ROC: 0.8700 at epoch: 1867

Min test cost: 0.46984 at epoch: 73

Min train cost: 0.34697 at epoch: 1867

Training time: 2387.3647649288177

### dropout\_rate = 0.5

Batch size: 64 Num batches: 375 Num epochs: 2000

Num layers: 3 total num nodes: 1000

Dropout rate: 0.5

Max test AUC of ROC: 0.7393 at epoch: 517

Max train AUC of ROC: 0.7459 at epoch: 1586

Min test cost: 0.46362 at epoch: 609

Min train cost: 0.46401 at epoch: 1586

Training time: 2390.5741119384766

### dropout\_rate = 0.7

Batch size: 64 Num batches: 375 Num epochs: 2000

Num layers: 3 total num nodes: 1000

Dropout rate: 0.7

Max test AUC of ROC: 0.7379 at epoch: 1806

Max train AUC of ROC: 0.6974 at epoch: 1170

Min test cost: 0.46812 at epoch: 1625

Min train cost: 0.49153 at epoch: 1169

Training time: 2397.321302175522

### dropout\_rate = 0.9

Batch size: 64 Num batches: 375 Num epochs: 2000

Num layers: 3 total num nodes: 1000

Dropout rate: 0.9

Max test AUC of ROC: 0.6690 at epoch: 26

Max train AUC of ROC: 0.5594 at epoch: 28

Min test cost: 0.51189 at epoch: 23

Min train cost: 0.54278 at epoch: 1428

Training time: 2406.491350889206

### Random Forest

Test AUC score: 0.741

Training AUC Score: 0.872

### Results

|  |  |  |
| --- | --- | --- |
| **Dropout rate** | **Test AUC score** | **Train AUC score** |
| dropout\_rate = 0.1 | 0.7229 | 0.9855 |
| dropout\_rate = 0.3 | 0.7278 | 0.87 |
| dropout\_rate = 0.5 | 0.7393 | 0.7459 |
| dropout\_rate = 0.7 | 0.7379 | 0.6974 |
| dropout\_rate = 0.9 | 0.669 | 0.5594 |
| Random Forest | 0.741 | 0.872 |

Given these results we can see that the best configuration was with a dropout rate of 0.5, but if we then compare it with the random forest example it is clear that random forest is the clear winner since it not only provides a better result but it is significantly faster than using deep neural networks.