Abhishek Tomar

[tomar.abhishek76@gmail.com](mailto:tomar.abhishek76@gmail.com) [drop me an email in case of any issue/query]

# Program to implement Generic Neural Network architecture

## Objective:

To build a library to implement a generic neural network architecture i.e NN architecture is built at run time according to the “number of layers” and “number of neurons at each layer” passed as argument to the program. We will learn about the following after this exercise:

* Implementation of Generic Neural Network (without using any NN python library APIs)
* One-hot encoding concept & generation
* Back propagation algorithm implementation to train the network using SGD
* Confusion matrix (implementation & interpretation)
* “sigmoid” and “relu” as the activation functions
* A [small exercise](#_(e)_“relu”_activation) for the reader to optimize the cost while using “relu” function

[Note: refer [link1](https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/) and [link2](https://en.wikipedia.org/wiki/Confusion_matrix) to learn more about confusion matrix]

## Methodology Used:

The network is trained using the Stochastic Gradient Descent (SGD) algorithm with batch size as the configurable input to the program.

## Data set Used (Training/Testing):

Generic Neural Network is trained and tested over the [Poker Hand](https://archive.ics.uci.edu/ml/datasets/Poker+Hand) dataset available on the UCI repository. The training set contains 25010 examples whereas the test set contains 1000000 examples each. The dataset consists of 10 categorical attributes. The last entry in each row denotes the class label. More details about the data set are available in detail over the link given above.

## Problem Statement:

Solve the following problems “a-e” detailed below by implementing the Neural Network architecture as per the details below:

(a) The Poker Hand dataset described above in “Data set Used” section has 10 categorical attributes. Transform and save the given train and test sets using one hot encoding to convert categorical features to binary. We will use these new train and test sets for the subsequent parts.

(b) Write a program to implement a generic neural network architecture. Implement the backpropagation algorithm to train the network. We will train the network using Stochastic Gradient Descent (SGD) where the batch size is an input to the program. Program shall accept the following parameters:

* the size of the batch for SGD
* the number of inputs
* a list of numbers where the size of the list denotes the number of hidden layers in the network and each number in the list denotes the number of units (perceptrons) in the corresponding hidden layer. Eg. a list [100 50] speciﬁes two hidden layers; ﬁrst one with 100 units and second one with 50 units.
* the number of outputs i.e the number of classes Assume a fully connected architecture i.e., each unit in a hidden layer is connected to every unit in the next layer. You should implement the algorithm from ﬁrst principles and not use any existing MATLAB/python modules. Use the sigmoid function as the activation unit.

(c) In this part, we use the above implementation to experiment with a neural network having a single hidden layer. Vary the number of hidden layer units from the set {5, 10, 15, 20, 25}. Set the learning rate to 0.1. Choose a suitable stopping criterion and report it. Observe and plot the accuracy on the training and the test sets, time taken to train the network. Plot the metric on the Y axis against the number of hidden layer units on the X axis. Additionally, report the confusion matrix for the test set, for each of the above parameter values. What do you observe? How do the above metrics and the confusion matrix change with the number of hidden layer units?

(d) In this part, we will experiment with a network having two hidden layers, each having the same number of neurons. Set the learning rate to 0.1 and vary the number of hidden layer units, as described in part (c). Report the metrics and the confusion matrix on the test set, as described in the previous part. How do the metrics and the confusion matrix change with the number of hidden layer units? What eﬀect does increasing the number of hidden layers, keeping the number of hidden layer units same, have on the metrics and the confusion matrix?

(e) In this part, we will use ReLU as the activation instead of the sigmoid function, only in the hidden layer(s). ReLU is deﬁned using the function: g(z) = max(0, z). Change your code to work with the ReLU activation unit. Make sure to correctly implement gradient descent by making use of subgradient at z = 0. Here is a [resource](https://en.wikipedia.org/wiki/Subderivative) to know more about sub-gradients. Repeat part (e) using ReLU as the activation function in the hidden layers and report the metrics and the confusion matrix and described previously. What eﬀect does using ReLU have on each of the metrics as well as the confusion matrix?

## Implementation:

File “nn\_main.py” contains the control code to test the solution for parts “a-e” as specified in the “Problem Statement”

### (a) One-hot encoding

This part requires the training and test set to transform using one-hot encoding. Implementation is present in “data\_handler.py” file.

Use following command to generate one-hot encoding –

“Python nn\_main.py a config.txt poker-hand-training-true.data poker-hand-testing.data 1”

### (b) Generic Neural Networks

Use following command to run the program using generic neural network APIs –

“Python nn\_main.py b config.txt poker-hand-training-true.data poker-hand-testing.data 1”

Neural Network algorithm has been implemented with following configurable parameters –

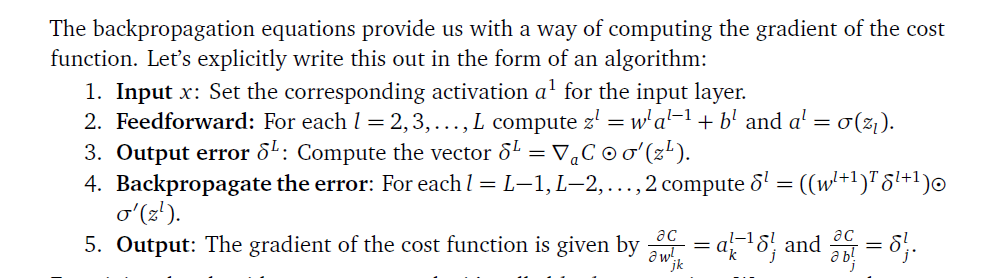
The format of the configuration file “config.txt” is as follows.

* The first line will consist of a single integer INPUT, the size of the input layer.
* The second line will consist of a single integer OUTPUT, the size of the output layer.
* The third line will consist of a single integer BATCH, the size of the batch size during training.
* The next line will consist of a single integer N, the number of hidden layers in the network.
* The next line will consist of N space-separated integers, h1, h2 ... hN where hi denotes the number of neurons in the ith layer.
* The next line will consist of a single string NONLINEARITY, which will either be "relu" or "sigmoid"

Key functions implemented:

1. **generic\_NN\_model**() is the main interface function implemented that takes the configuration file parameters & contructs the neural network graph.
2. **sgd\_algo**() encapsulates the stochastic gradient descent functionality over the batch of samples
3. **batch\_func**() encapsulates the forward & backward propagation over the batch of samples selected by sgd\_algo()

**Backpropagation Algorithm**



### (c) Single hidden layer

Modify the following in config.txt-

* line 5 to “5”

Give following command –

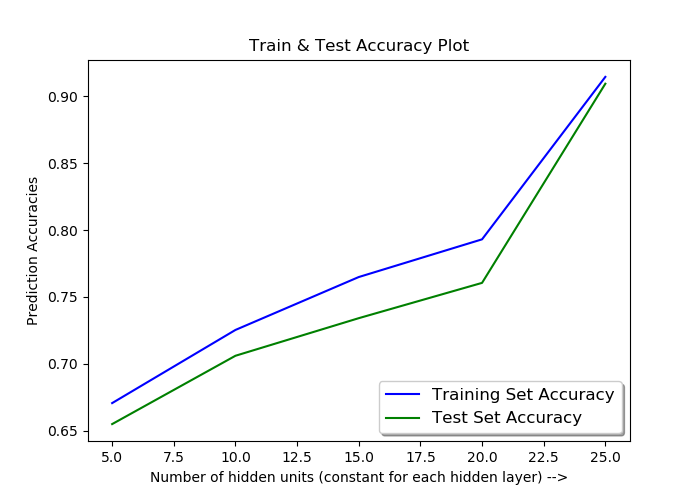
“Python nn\_main.py b config.txt poker-hand-training-true.data poker-hand-testing.data 5”

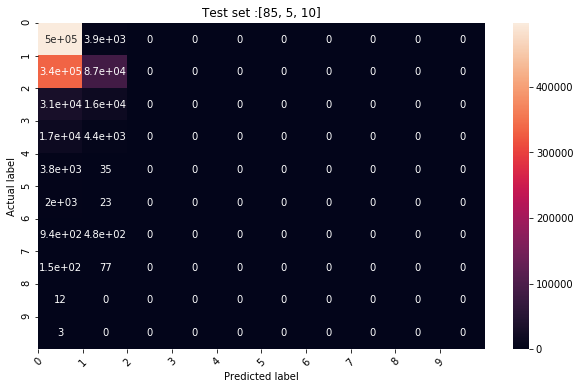
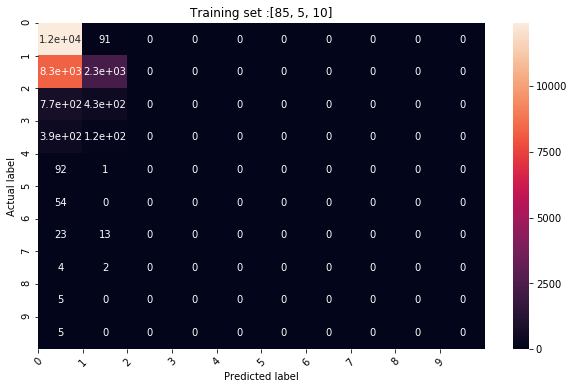
|  |  |  |  |
| --- | --- | --- | --- |
| Hidden Units (Count) | Training | Test | Cost Error |
| 5 | 0.58 | 0.58 | 0.60 |
| 10 | 0.70 | 0.68 | 0.52 |
| 15 | 0.72 | 0.69 | 0.46 |
| 20 | 0.83 | 0.80 | 0.44 |
| 25 | 0.90 | 0.89 | 0.33 |

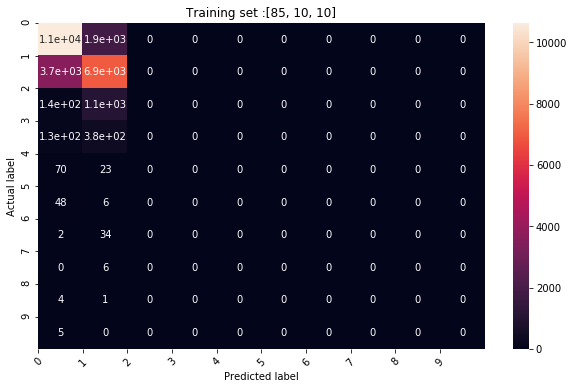
**Observation:** As the number of hidden units increases, the accuracy increases & the cost error decreases consistently.

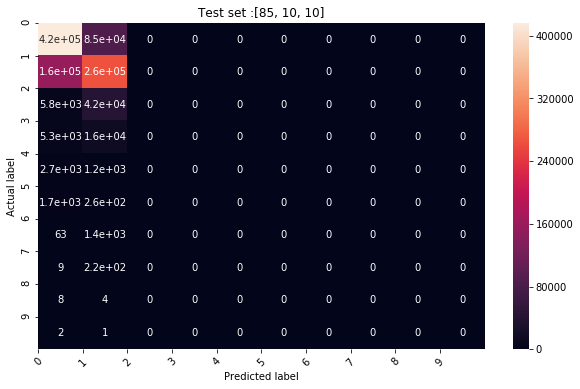
**Confusion Matrix**: However, upon plotting the confusion matrix, it becomes clear that the model is facing trouble in classifying most of the classes with low presence frequency in data. Only class 0 & 1 are classified correctly to some extent. With the increase in hidden units, the “True Positives” of both classes improves consistently.

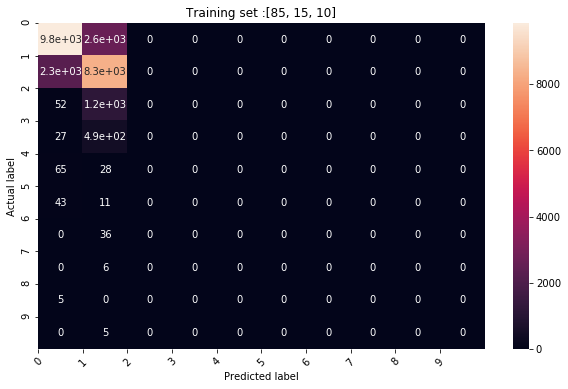
**Training Time** (one epoch): between 1-2 seconds

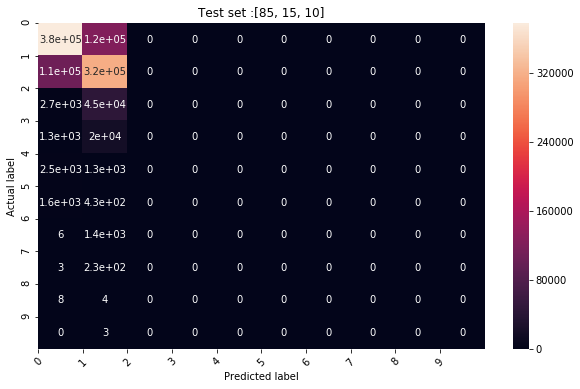




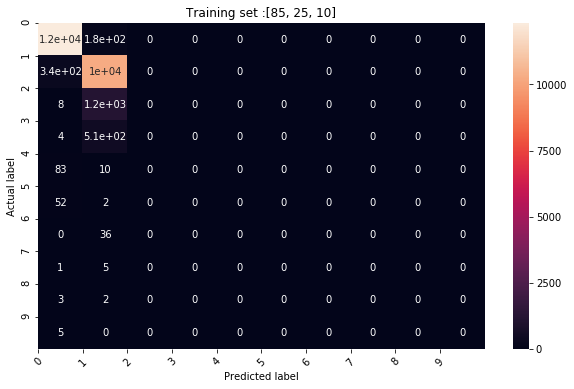


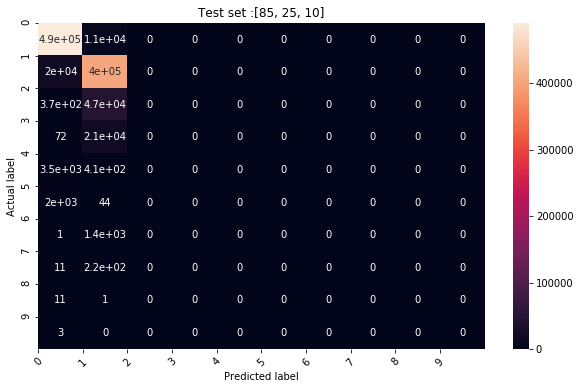






## 





### (d) 2-hidden layers

Modify the following in config.txt-

* line 4 to “2”
* line 5 to “5”

Give following command –

“Python nn\_main.py b config.txt poker-hand-training-true.data poker-hand-testing.data 5”

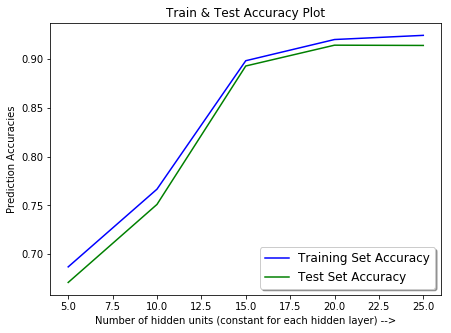
**Epoch = 100**

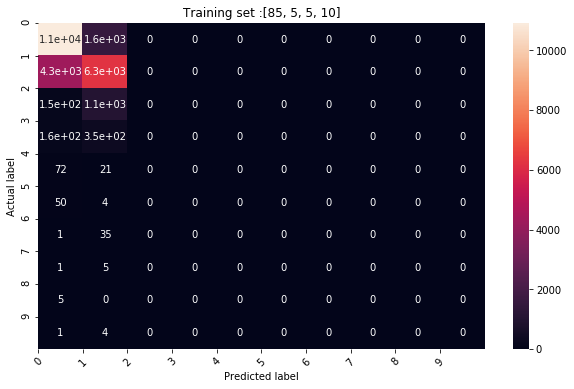
|  |  |  |  |
| --- | --- | --- | --- |
| Hidden Units (Count) | Training | Test | Cost Error |
| 5 | 0.68 | 0.67 | 0.59 |
| 10 | 0.76 | 0.75 | 0.49 |
| 15 | 0.898 | 0.892 | 0.34 |
| 20 | 0.92 | 0.914 | 0.310 |
| 25 | 0.924 | 0.914 | Stopping criteria hit at epoch 90 |

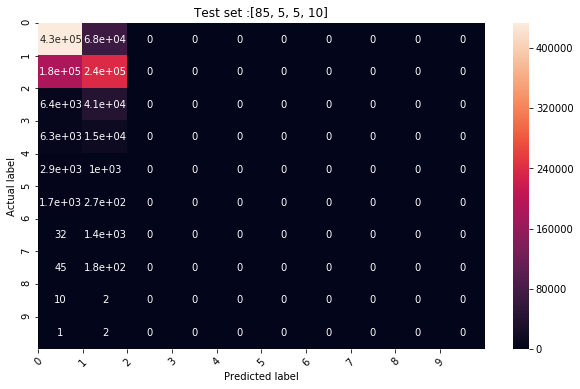
**Accuracy:** As the number of hidden units increases, the accuracy increases & the cost error decreases consistently & hits stopping criteria with 25 hidden units at epoch-90.

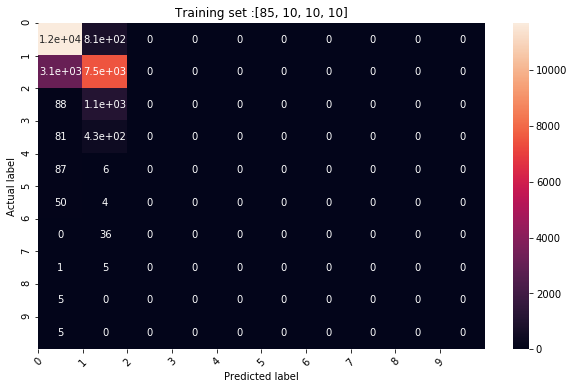
If we compare with single-hidden layer model, the maximum improvement of ~20% is observed with 15 hidden units. If we increase the hidden units beyond 15, we do get more improvement but the percentage of improvement is not that fast thereafter.

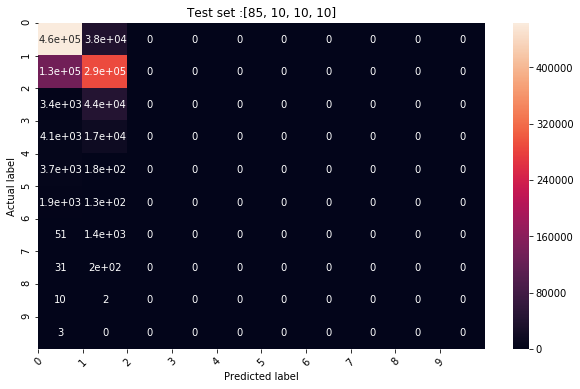
**Confusion Matrix**: With the increase in hidden units, the “True Positives” of classes improves consistently. “True Positives” of class-3 also appears with atleast 20 hidden units and the TP counts increases further with higher no. of units thereafter.

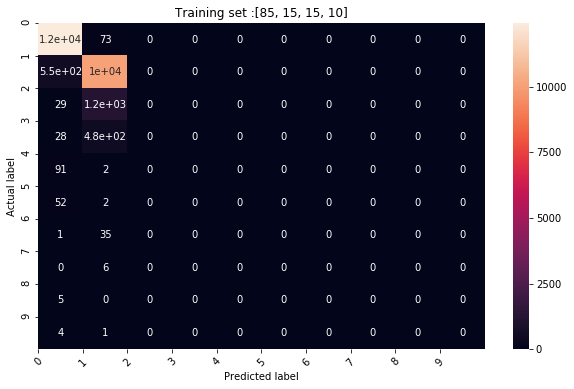


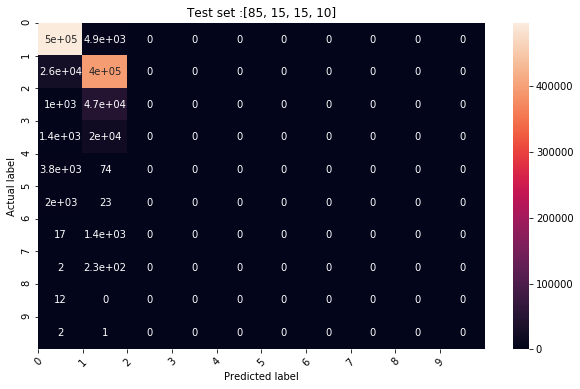


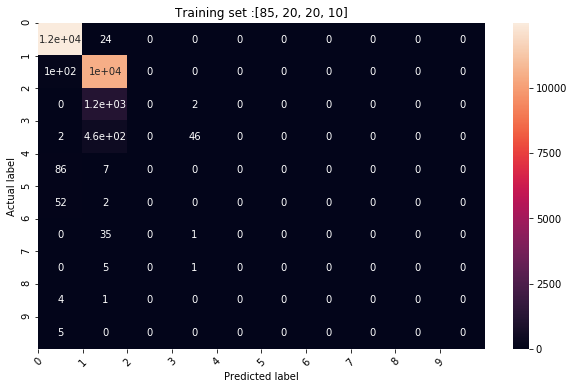


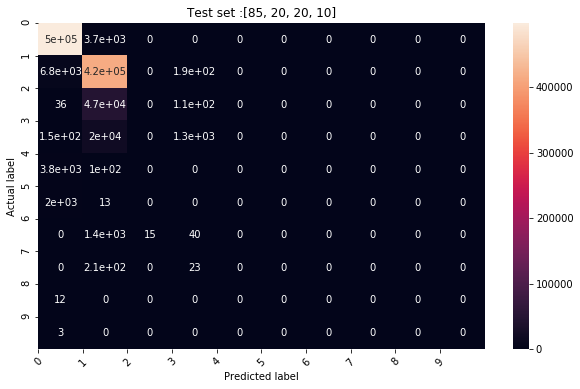


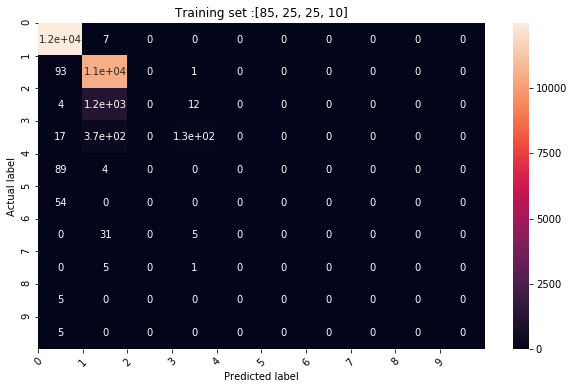


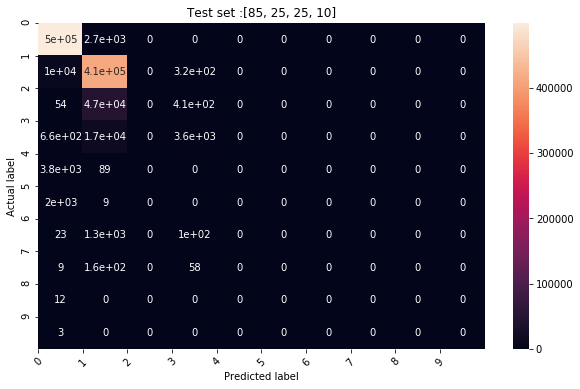












**Additional try** –

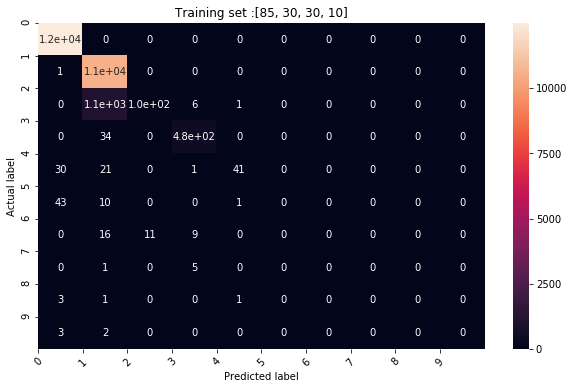
Epoch – 1000

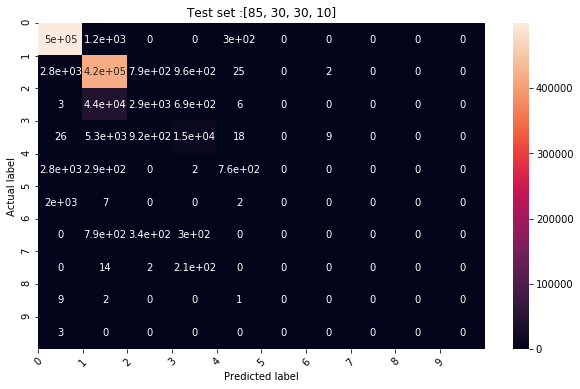
Training Cost:0.18184726109556176

Stopping criteria not met.

Training Set Accuracy: 0.9482606957217113

Test Set Accuracy: 0.93616





### (e) “relu” activation function –

Modify the config.txt –

* change “sigmoid” to “relu”

Command - ‘python nn\_main.py b config.txt poker-hand-training-true.data poker-hand-testing.data 5’

**Epoch = 100**

**Stopping criteria:** loss < 2e-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Units | Training Accuracy | Test Accuracy | Cost | Adjusted Learning rate |
| 5 | 0.56 | 0.55 | 0.89 | Stopped as learn rate hit the stopping criteria  3.2000000000000005e-05 |
| 10 | 0.55 | 0.54 | 0.90 | 3.2000000000000005e-05 |
| 15 | 0.60 | 0.60 | 0.81 | 6.400000000000001e-06 |
| 20 | 0.72 | 0.71 | 0.60 | 3.2000000000000005e-05 |
| 25 | 0.65 | 0.64 | 0.72 | 6.400000000000001e-06 |

[Note: There is lot of scope to optimize the cost (0.72) further. **Exercise for the reader – Try playing around with the config parameters and increase the epoch to optimize the cost further**]

