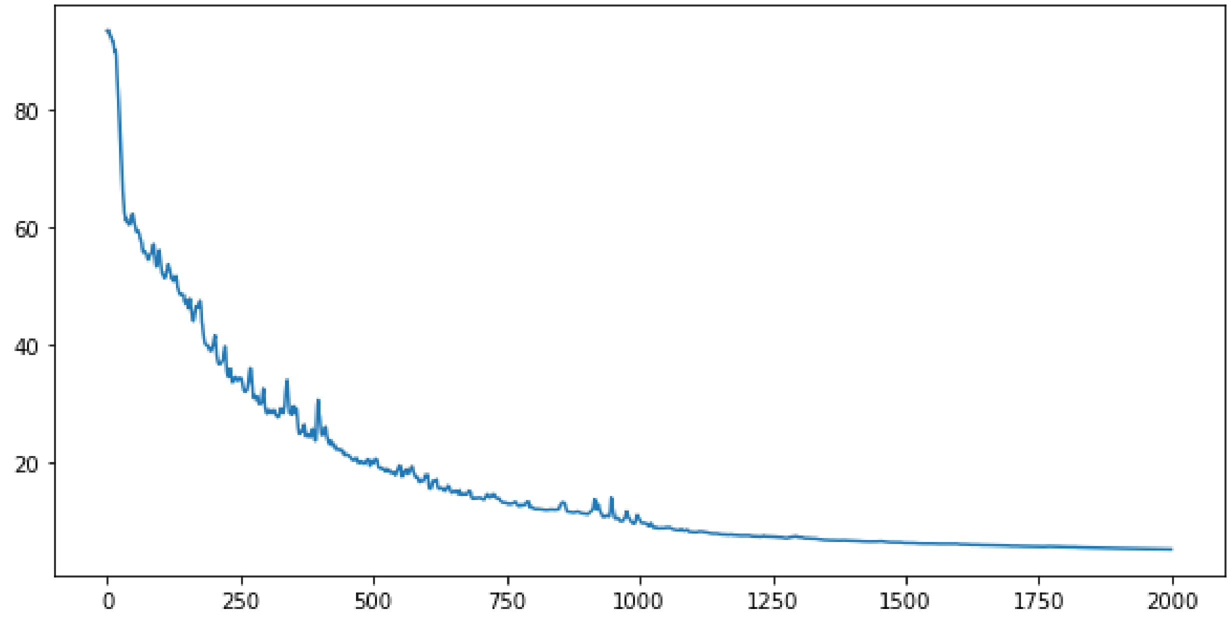
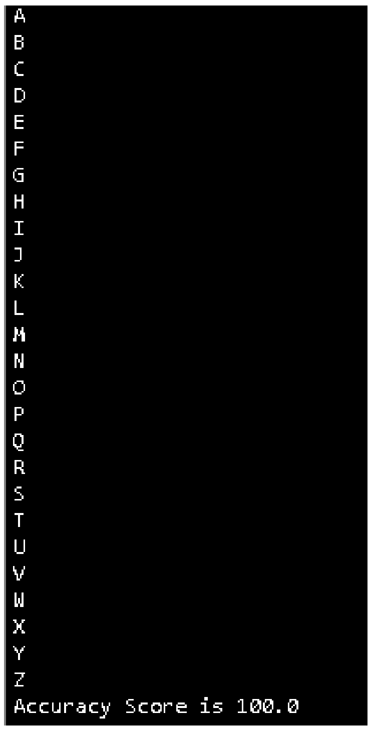
Solutions and Screenshots

1 Solutions 1 and 2 are in file **Sol1-2.py**

Weights are kept constant and loaded from the csv file included wh.csv for weights of hidden nodes and wi.csv for weights of output nodes



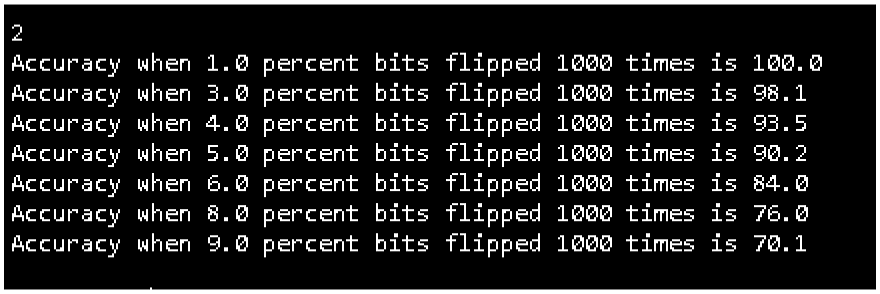


```
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Accuracy Score is 100.0
```

2 Sol1-2.py

Take a character in this case ‘A’ and randomly add noise to 1 percent of 156 bits (features). Repeat this 1000 times on the same percentage of noise.

Similarly add noise to 3, 4, 5, 6, 8, 9 percent of features.

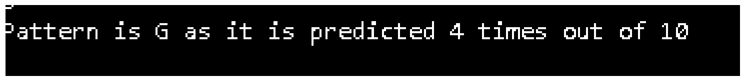


```
2
Accuracy when 1.0 percent bits flipped 1000 times is 100.0
Accuracy when 3.0 percent bits flipped 1000 times is 98.1
Accuracy when 4.0 percent bits flipped 1000 times is 93.5
Accuracy when 5.0 percent bits flipped 1000 times is 90.2
Accuracy when 6.0 percent bits flipped 1000 times is 84.0
Accuracy when 8.0 percent bits flipped 1000 times is 76.0
Accuracy when 9.0 percent bits flipped 1000 times is 70.1
```

It is observed that MLP can tolerate around 5 percent of the noise with above 90 percent of accuracy

3 Sol3.py

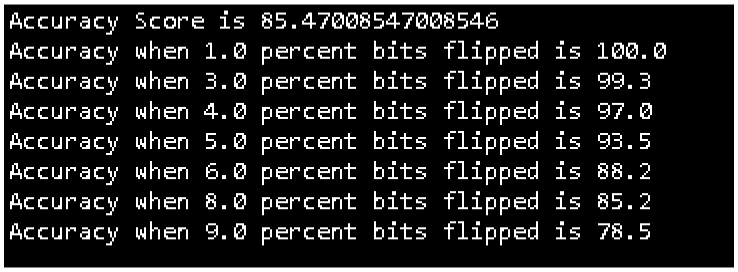
Pattern 2 comes different in each case. So, we run MLP 10 times and calculated which character is recognized max number of times and that is character G



```
Pattern is G as it is predicted 4 times out of 10
```

4.1 sol4_1.py

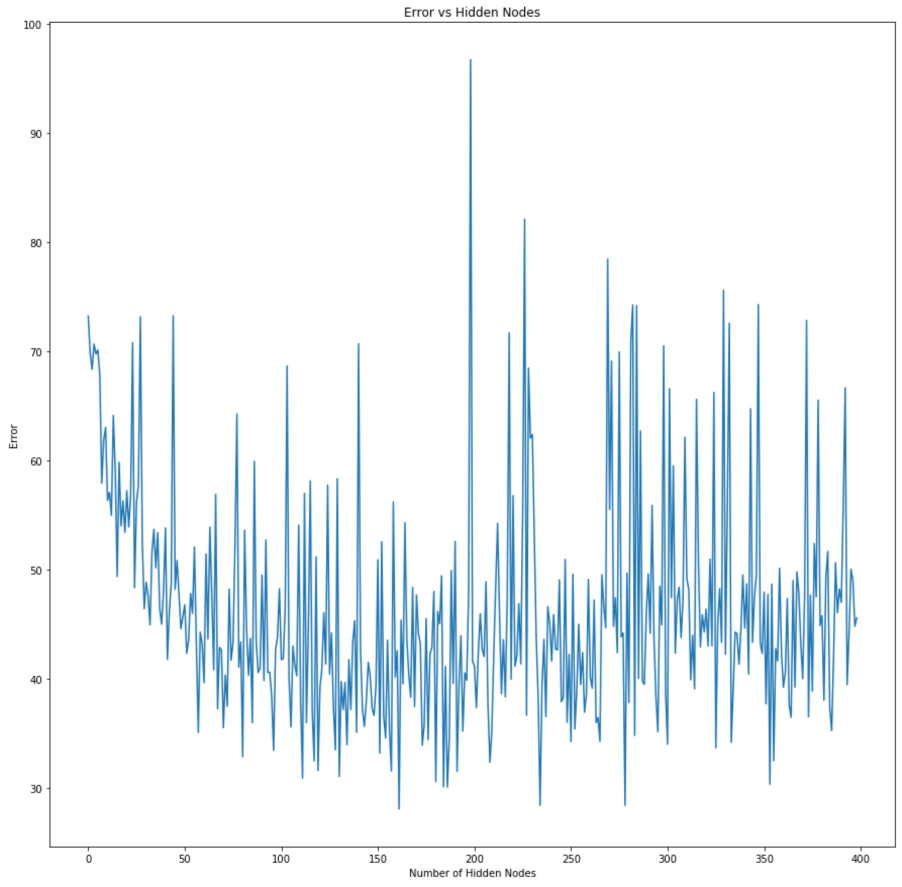
Like question 2 we flip the bits and added to our training set. Though the accuracy decreases for same number of trials (2000) our model becomes more efficient in recognizing noisy data. Even when 8 percent bits are flipped accuracy is close to 85 percent which is 10 percent more when model is not trained on noisy patterns.



```
Accuracy Score is 85.47008547008546
Accuracy when 1.0 percent bits flipped is 100.0
Accuracy when 3.0 percent bits flipped is 99.3
Accuracy when 4.0 percent bits flipped is 97.0
Accuracy when 5.0 percent bits flipped is 93.5
Accuracy when 6.0 percent bits flipped is 88.2
Accuracy when 8.0 percent bits flipped is 85.2
Accuracy when 9.0 percent bits flipped is 78.5
```

4.2 Sol4_2.py

Keeping the weight of hidden and output nodes as constant and starting hidden nodes as 1 to double the number of input features, i.e. 312 and even further like 400 we can see that error decreases till mean of input and output nodes that is around 82 and then remain almost similar around 156 (number of input nodes) and then keep on increasing.



We can take hidden nodes as 82 which has almost the least error and more efficient compared to other values.

References

[1] H. network?, "How to choose the number of hidden layers and nodes in a feedforward neural network?", Stats.stackexchange.com, 2017. [Online]. Available: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw. [Accessed: 26- Oct- 2017].