**Home Work 4:**

**Intelligent Systems-**

# 

# **Problem One**

In this problem we were required to simulate a flexible multi-layer feed-forward network with back-propagation including momentum step. Initial weights to the hidden layer are initiated using the weights from an “auto-encoder” network trained earlier.

## **System Description**

Since the objective of this problem is to test the features from the auto-encoder, the exact same

parameters (with **150 hidden neurons)** from the best-case simulation were used to enable

accurate comparison between the two scenarios. The learning rates ηh = ηo = **0.1** and the

momentum parameter (α) was selected as **0.15**. Input to hidden layer weights were initialized

from the best-case simulation earlier while the weights from hidden to output layer were

initialized randomly based on a **Gaussian** of mean (µ) =0 and SD (σ) = √(2/n) where n=number of

neurons in the layer same as the best-case simulation.

## **Results**

Using the above parameters, the final network was trained by Back-propagation using **stochastic gradient descent** method. Only the output layer weights were ‘learnt’ as part of this process. 100 random training points from the training set in each epoch and initially 1000 epochs were used. The error rates observed were higher when compared to the best-case simulation. Hence **3000 epochs** were used to observe the error rates over a longer simulation (300,000 points). The error rates were plotted for every 10 epochs as shown in Figure 1.

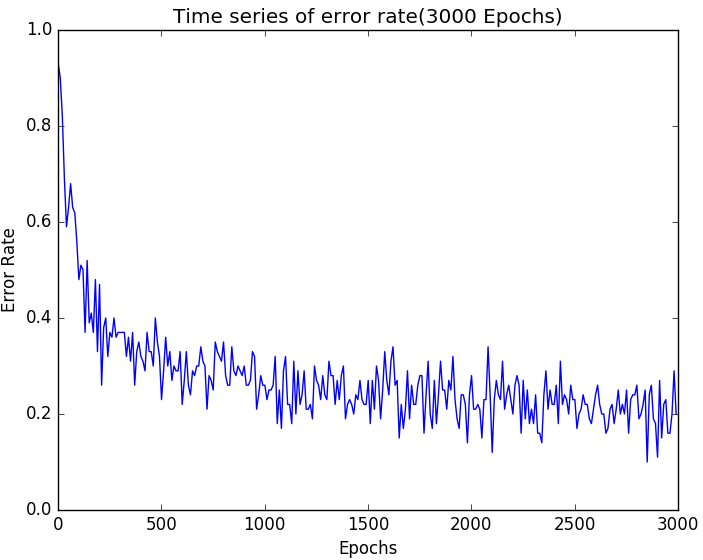


Figure 1 Time series of error rates for every 10 epochs over 3000 epochs

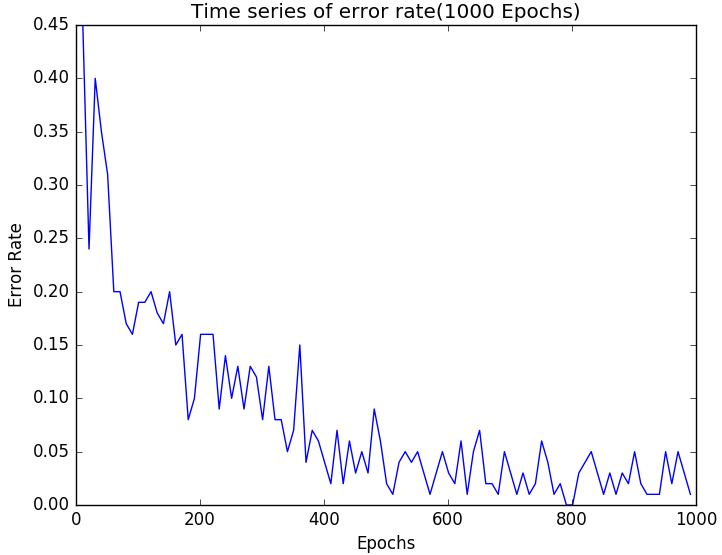


Figure 2: Time series plot of error rates for every 10 epochs over 1000 epochs (from problem 1 in HW3)

The error rates at the beginning are high and settle down at around **0.20** after 500 epochs. The average error rate observed is still **clearly higher** when compared to the best-case simulation from the earlier problem where both the hidden and output layer weights were trained (Figure 2). The final test set hit-rate (based on max-threshold approach) was **87.3%** down from **92.6 %** when both the layers were trained. **Confusion matrices** were generated for both training and testing sets to further evaluate the performance. Table 1 indicates the confusion matrix for the training set of 4000 points.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **A**  **C**  **T**  **U**  **A**  **L** |  | **PREDICTED** | | | | | | | | | |
| Digits | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 362 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 1 |
| **1** | 0 | 436 | 3 | 1 | 0 | 1 | 2 | 1 | 0 | 1 |
| **2** | 2 | 0 | 389 | 3 | 10 | 1 | 4 | 5 | 13 | 3 |
| **3** | 1 | 1 | 6 | 370 | 2 | 3 | 1 | 4 | 7 | 6 |
| **4** | 0 | 0 | 1 | 0 | 363 | 2 | 3 | 2 | 3 | 14 |
| **5** | 12 | 1 | 3 | 10 | 4 | 312 | 12 | 3 | 5 | 7 |
| **6** | 0 | 3 | 2 | 0 | 2 | 4 | 349 | 1 | 0 | 0 |
| **7** | 1 | 6 | 7 | 1 | 8 | 2 | 0 | 374 | 1 | 7 |
| **8** | 2 | 5 | 4 | 11 | 5 | 10 | 9 | 4 | 350 | 3 |
| **9** | 4 | 6 | 1 | 6 | 16 | 1 | 3 | 5 | 6 | 381 |

Table 1: Confusion matrix between the actual and predicted labels for training set (4000) points

There is a significant increase in the mis-classification counts in this problem. Looking at mis-classifications, digit 8 has the **most** mis-classifications (53) followed by digit 9(48) which is consistent with the earlier problem. Also the most individual mis-classifications were observed between the digits 9 <-> 4, 2 <-> 8.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **A**  **C**  **T**  **U**  **A**  **L** |  | **PREDICTED** | | | | | | | | | |
| Digits | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 88 | 0 | 1 | 0 | 0 | 0 | 3 | 1 | 0 | 0 |
| **1** | 0 | 122 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 0 |
| **2** | 1 | 0 | 86 | 2 | 2 | 0 | 2 | 1 | 5 | 1 |
| **3** | 0 | 0 | 0 | 84 | 1 | 5 | 1 | 4 | 1 | 3 |
| **4** | 0 | 1 | 0 | 0 | 102 | 1 | 1 | 0 | 0 | 7 |
| **5** | 0 | 2 | 2 | 3 | 3 | 69 | 2 | 1 | 5 | 0 |
| **6** | 2 | 0 | 2 | 0 | 0 | 0 | 96 | 0 | 1 | 0 |
| **7** | 1 | 1 | 2 | 1 | 4 | 1 | 3 | 89 | 0 | 3 |
| **8** | 4 | 4 | 6 | 3 | 1 | 4 | 1 | 1 | 58 | 4 |
| **9** | 1 | 1 | 0 | 1 | 2 | 3 | 0 | 3 | 0 | 80 |

Table 2: Confusion matrix between the actual and predicted labels for testing set (1000) points

Table 2 is the confusion matrix for the testing set (1000) points. Once again most mis-classifications were observed for digit 8(28) with the most individual mis-classifications were observed between the digits 9 <-> 4, 2 <-> 8.

From this we can conclude that the weights trained by the “auto-encoder” were probably not of high quality and did not necessarily identify distinct features. Features from ”auto-encoder” simulation in Figure 3 still look more distinctive and clear when compared to those from the best case simulation (problem 1 from HW 3) in Figure 4, but still resulted in higher error rates.

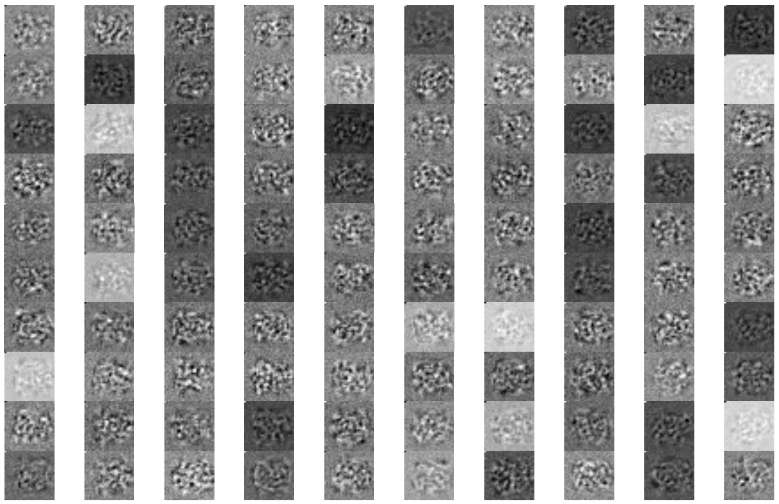


Figure 3: Features learnt by the hidden layer neurons from “auto-encoder” simulation

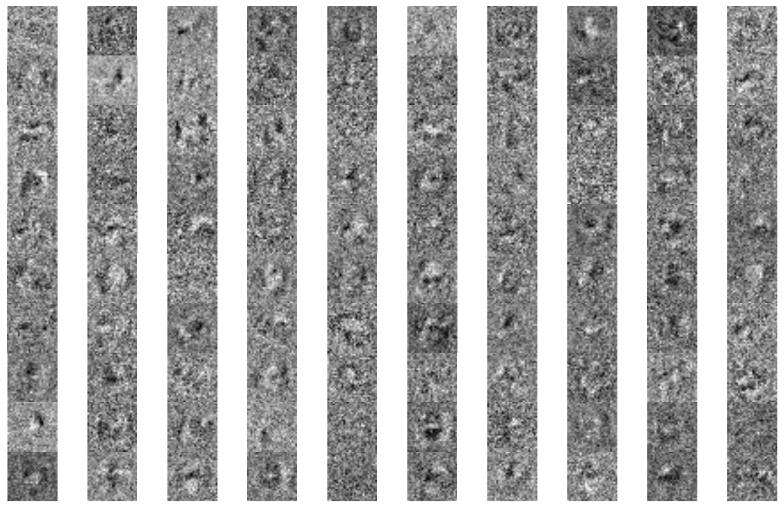


Figure 4: Features learnt by hidden layer neurons from best-case simulation (i.e. problem 1 of HW 3)

One possible explanation for this could be that the features were learnt together across both the layers in the best case simulation (problem1 from HW 3) which was missed in this problem as only hidden layer features were initialized while the weights in output layer were random.

**PROBLEM 2**

**Problem Summary:** As part of this problem we were supposed to simulate a “fine-tuned” multi-layer feed-forward network with back-propagation including momentum step. Initial weights to the hidden layer are initiated using the weights from an “auto-encoder” network trained earlier but instead of training only output layer (as in problem 1), we should train both the layers to see if it makes any differences with respect to the error rates observed.

**Parameter Settings:** Since the objective of this problem is to check if training weights from two layers instead of one (starting with the same initial weights for hidden layer in both cases), same parameters were selected as in problem 1. The learning process was again repeated by using a smaller learning rate (ηh = 0.05 instead of 0.1) for hidden layer weights as they have initialized using the weights from “auto-encoder” simulation with some learning already done on them.

**Results:** Using the above parameters, the final network was trained by Back-propagation using **stochastic gradient descent** method. Both the hidden and output weights were ‘learnt’ over **1000** epochs (same number of epochs in problem1 from HW3) with 100 random points in each epoch. The error rates were plotted for every 10 epochs as shown in Figure 1.

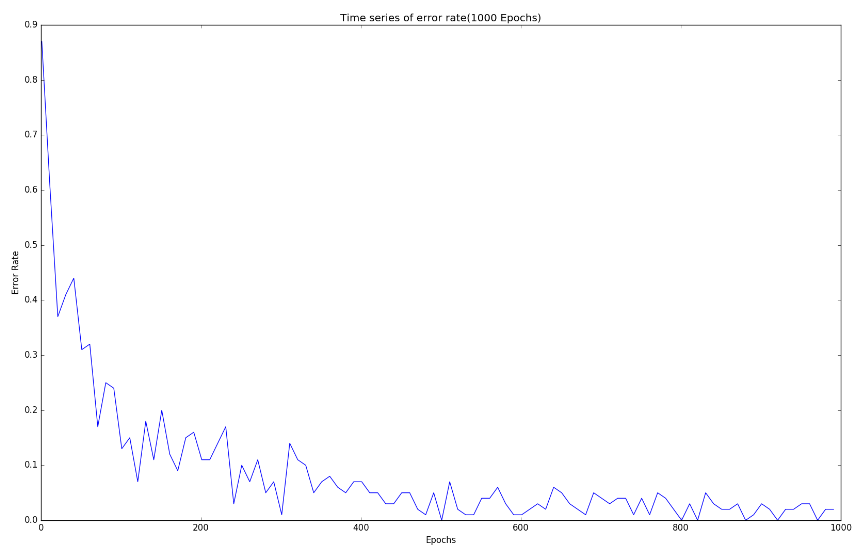


Figure 5 Time series of error rates for every 10 epochs over 1000 epochs

The error as could be seen decreases sharply, from the first epoch itself and completely flat-lines by 250 epochs which is a lot faster than in previous case (i.e. problem 1 from HW3) in Figure 1 which flat-lines after 400 epochs. This clearly shows that the **fine-tuning** process followed by using hidden layer weights from auto-encoding had a positive effect on the speed of learning.

The final test set hit-rate was **92.8%** which is also very close to the best-case simulation rate of **92.6 %.** **Confusion matrices** were generated for both training and testing sets to further evaluate the performance. Table 1 indicates the confusion matrix for the training set of 4000 points.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **A**  **C**  **T**  **U**  **A**  **L** |  | **PREDICTED** | | | | | | | | | |
| Digits | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 377 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| **1** | 0 | 457 | 2 | 2 | 0 | 0 | 1 | 0 | 0 | 0 |
| **2** | 0 | 0 | 417 | 2 | 4 | 0 | 2 | 2 | 2 | 2 |
| **3** | 0 | 0 | 3 | 382 | 0 | 0 | 0 | 4 | 2 | 1 |
| **4** | 0 | 0 | 0 | 0 | 391 | 0 | 0 | 1 | 1 | 3 |
| **5** | 2 | 2 | 3 | 2 | 0 | 329 | 3 | 1 | 2 | 4 |
| **6** | 2 | 2 | 0 | 0 | 0 | 0 | 356 | 0 | 0 | 0 |
| **7** | 0 | 2 | 2 | 0 | 1 | 1 | 0 | 403 | 1 | 0 |
| **8** | 1 | 1 | 1 | 4 | 1 | 1 | 1 | 1 | 390 | 0 |
| **9** | 1 | 1 | 0 | 3 | 2 | 1 | 1 | 2 | 3 | 406 |

Table 3: Confusion matrix between the actual and predicted labels for training set (4000) points

Large diagonal values indicate that most points were correctly classified with a hit-rate of **96.8%.** Most mis-classifications are observed in digits 5(19), 9(14) and 2(14) consistent with earlier case (problem 1 in HW3). Similarly confusion matrix is also generated for test set as in Table 4.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **A**  **C**  **T**  **U**  **A**  **L** |  | **PREDICTED** | | | | | | | | | |
| Digits | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 77 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 |
| **1** | 0 | 106 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| **2** | 1 | 0 | 88 | 3 | 1 | 0 | 3 | 1 | 2 | 0 |
| **3** | 0 | 0 | 1 | 101 | 0 | 3 | 0 | 1 | 1 | 1 |
| **4** | 0 | 0 | 0 | 0 | 96 | 0 | 1 | 1 | 1 | 5 |
| **5** | 3 | 0 | 1 | 5 | 2 | 91 | 2 | 0 | 3 | 1 |
| **6** | 3 | 0 | 1 | 0 | 2 | 1 | 95 | 0 | 0 | 0 |
| **7** | 1 | 1 | 2 | 0 | 2 | 2 | 0 | 92 | 0 | 2 |
| **8** | 0 | 1 | 2 | 2 | 1 | 3 | 0 | 2 | 76 | 1 |
| **9** | 4 | 0 | 0 | 1 | 1 | 3 | 0 | 3 | 2 | 86 |

Table 2: Confusion matrix between the actual and predicted labels for testing set (1000) points

Test set mostly mimics the training set in terms of mis-classifications seen. Most mis-classifications could be seen in digits 5(17), 9(14) and 8(12) respectively. Final features from hidden layer were plotted in Figure 6 to check if the features learnt are in anyway ‘better’ than the one used on earlier problem (which are the same features trained in “auto-encoder” simulation) in Figure 3. Weights/features in Figure 6 are very similar to the auto-encoded weights in terms of the shapes/structures. In terms of pixel sharpness, the new weights are little sharper than the initial weights indicating the extra learning that has happened over the 1000 epochs of back-propagation learning. Also, the differences observed are not major which could have been the reason the error rate stabilized so quickly (after around 200 epochs) as the weight changes are very minimal

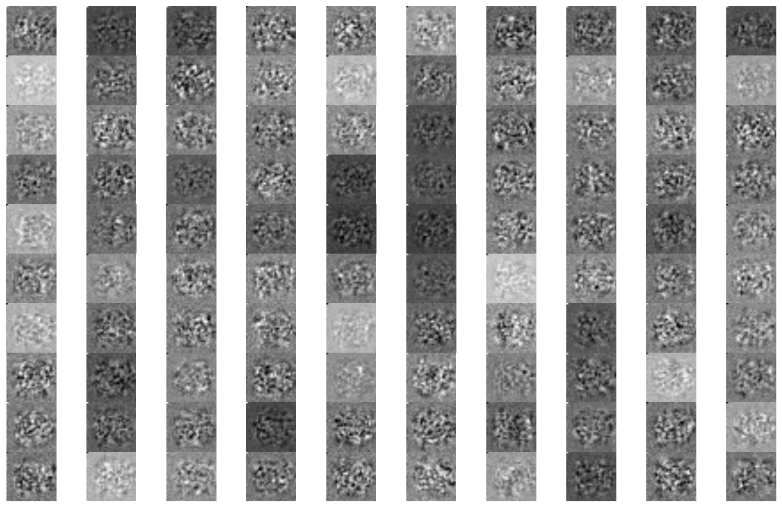


Figure 6: Features learnt by the hidden layer neurons with initial weights initiated from auto-encoder simulation

Also the similarity among each of these weights in terms of shapes of the features encoded (which is the case in both the figures) indicate that these weights co-learnt/co-adapted i.e. they have learned very similar features. To force the hidden layer features to be more diverse and disrupt co-adaptation, we can follow **regularization** techniques like “**drop-out”** or **“sparse-coding”** where we allow a hidden neuron to be active only for a small fraction of data points.