INSTITUT POLYTECHNIQUE DE PARIS

DEEP LEARNING II PROJECT

Ahmed KHALIFE | Sonali PATEKAR

Dataset

In Section 3 of our analysis, we utilized the Binary AlphaDigits dataset, while in Section 4, we employed the MNIST dataset.

To run our code, we need to download two CSV files from the following link:

https://www.kaggle.com/datasets/oddrationale/mnist-in-csv.

The MNIST dataset is available in CSV format.

Additionally, please download the Binary Alpha Digits dataset from this link: https://cs.nyu.edu/~roweis/data/binaryalphadigs.mat.

Basic Functions

The elementary functions were developed and can be found in three separate files: RBM.py, DBN.py, and DNN.py. In addition, some image generation functionality was implemented in load_data.py. These files are accessed from notebook.ipynb.

Study on Binary Alpha Digits

We conducted a study on the Binary Alpha Digits dataset, testing our code with various hyperparameters. First, we experimented with the number of neurons, then, we varied the number of layers. Additionally, we adjusted the number of characters used for training.

1 Number of neurons:

To generate images with the RBM, we adjusted the parameter q and produced our images. Images with q values below 150 were significantly distorted, where the reconstruction error is also presented.

For the DBN, we utilized the sizes 50, 100, 150, 200,250. We observed that images generated with q values below 150 were blurry.

2 Number of layers:

We tested the DBN with different numbers of hidden layers ranging from one of 200 to four of 200. It should be noted that we increased the number of characters used for training for this section. We observed that images generated with four hidden layers were significantly clearer than those with fewer hidden layers

3 Characters:

We examined the impact of changing the number of characters used for training.

For the RBM, we adjusted the number of characters used for training. When we trained the model only on the digit 5, the generated images were remarkably clear, with unique variations in the writing style. As we increased the number of characters used for training, it became increasingly difficult to recognize the digits in the generated images. For example, when we trained the model on characters 1, 2, 3, 4, 5, and 6, the generated images were heavily distorted.

For the DBN, we also experimented with the number of characters used for training and generated new results. We observed that even when we trained the model on six or nine characters, some of the digits in the generated images were still recognizable. However, the images generated with characters 2, 3, 4, and 5 were the clearest.

Program and Analyses (MNIST Study)

The programs are written in Python as Jupyter Notebook.

During the production of the figures, we encountered some challenges with our Python implementation, which is slower compared to existing Python modules. As a result, the figures took longer to generate, and we had to limit the number of epochs to keep the runtime reasonable. Despite the high error rates displayed on the graphs, they are still useful for comparing different configurations.

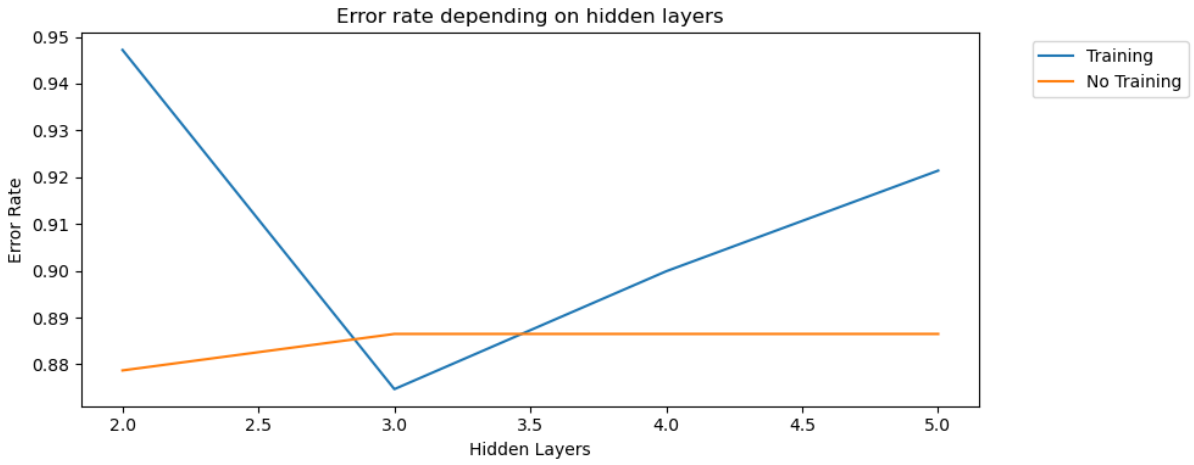

Figure 1: The results show that pre-training can be a disadvantage when there are more than 2 hidden layers. Using 2 or 3 hidden layers seems to produce good results.

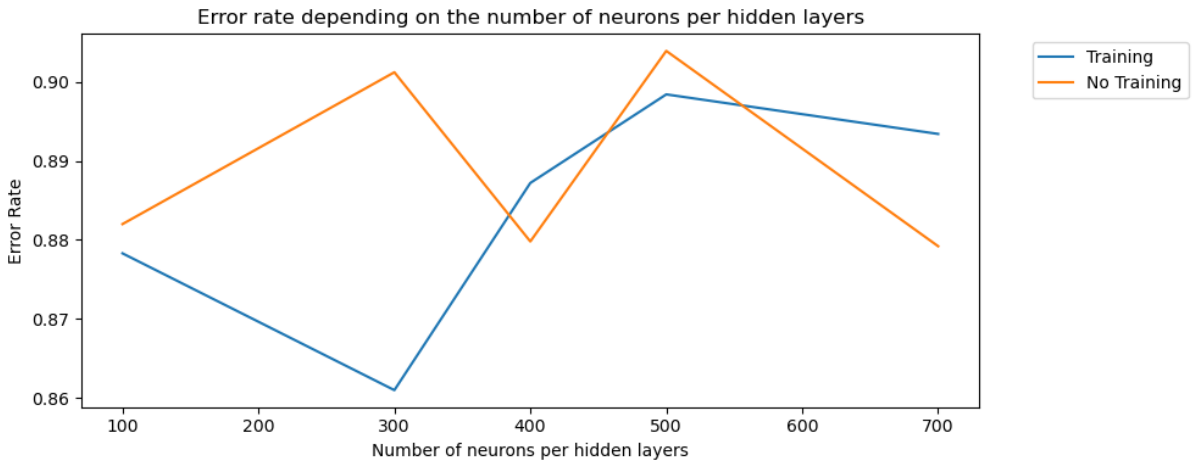

Figure 2: Generally, having more neurons per hidden layer helps to minimize the error rate. We also observed that preprocessing the data improves the error rate more effectively when there are more neurons. This is interesting because the MNIST images consist of 28x28 pixels, or 784 pixels, so the first hidden layer should be able to extract relevant information from the image.

Figure 3: As the amount of training data increases, the difference pre-training makes becomes less significant. With less than 10,000 training data, the number of epochs is sufficient for the network to "learn" the data effectively. However, the error rate increases at 10,000 training data due to a lack of epochs relative to the data volume. Increasing the amount of training data helps the network to generalize better and improve its performance.

Best Configuration

A good configuration would be to use 700 neurons per hidden layer with 2 hidden layers, and train the network on all available training data. We found that pre-training is not necessary when there is a large amount of data. With a learning rate of 0.1 and a batch size of 1000, we obtained an error rate of 14% on the test data after only 25 epochs, as shown below.