Department of Electrical and Electronics Engineering

EEE 443: Neural Networks

Class Project V Report

Name and Surname: Ali Aral Takak

Student ID: 22001758

Department: EEE

Instructor: Erdem Koyuncu

Introduction

In this project, we design and implement a neural network to classify handwritten digits from the MNIST dataset. The problem involves recognizing images of digits (0–9) and assigning them to the correct class using a supervised learning approach. Our design employs a fully connected feedforward architecture with one or more hidden layers, and we optimize the network parameters using the backpropagation algorithm. To improve generalization and convergence, we incorporate techniques such as dynamic learning rate adaptation, dropout regularization, and momentum-based updates. Through systematic experimentation with hyperparameters—including the number of hidden neurons, learning rate, dropout probability, and momentum coefficient—we aim to achieve a model with test accuracy exceeding 95%.

Design Process

In our design process, we started with a basic fully connected neural network for MNIST digit classification and then systematically experimented with various architectural and training parameters to enhance performance. Initially, a single hidden layer network using sigmoid activations was implemented, but it showed slow convergence and limited accuracy. We then replaced the hidden layer's sigmoid with the tanh activation, which improved gradient flow due to its zero-centered output, and modified the output layer to use softmax with cross-entropy loss for more appropriate probability modeling in multi-class classification. We kept the number of hidden layers the same at every trial, as one, due to the fact that while more hidden layers could provide more performance and accuracy, a single hiden layer is sufficient for MNIST dataset.

Next, we focused on hyperparameter tuning, experimenting with the number of hidden neurons, initial learning rate, dropout probability, and momentum coefficient. We introduced a dynamic learning rate adaptation mechanism that reduces the learning rate when the training loss increases between epochs, ensuring more stable convergence. Dropout was incorporated in the hidden layer to randomly deactivate a fraction of neurons during training, thereby mitigating overfitting, while momentum was added to the gradient updates to smooth the optimization trajectory and accelerate convergence.

Final Model

The final neural network model that we have used is designed to classify handwritten digits from the MNIST dataset. The network is a fully connected feedforward architecture that consists of an input layer, one hidden layer, and an output layer. The input layer corresponds to 784 inputs which is derived from flattened 28×28 pixel images. Our hidden layer contains 175 neurons, and the output layer contains 10 neurons, each representing one of the digits from zero to nine, via one-hot encoding. Our initial learning rate is set as 0.25, and all the parameters were chosen by trial-and-error method. While lower and higher learning rates resulted with converge to lower accuracy rates, our choice of 0.25 resulted with the best performance. We use 250 epoch for the training process, where we chose it by trial-and-error again. While epoch numbers higher than 250 were overkill and resulted with similar accuracy results and convergences, lower epoch numbers were satisfactory too, however, in order to guarantee our results, we chose 250 epochs.

In the hidden layer, we use the hyperbolic tangent activation function, which is defined as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

The hyperbolic tangent activation function produces outputs in the range from -1 to 1, which is beneficial due to its zero-centered nature and in general provides better gradient flow during the training. The output layer uses the softmax activation function, which converts the raw output scores into a probability distribution. The SoftMax activation is formulated as:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{10} e^{x_j}}$$

SoftMax activation is beneficial for multiclass classification tasks, due to the fact that its formulation outputs the sum to one and can be interpreted as probabilities.

Our network's parameters, in this case weights, are initialized with small random values, and zeroes for biases. Our loss function is the cross-entropy loss, which is formulated as:

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{10} Y_{ji}\log\left(A2_{ji}\right)$$

Where m is the number of training examples, Y is the one-hot encoded target, and $A2$ is the output of the SoftMax function.

To enhance the model's generalization and accelerated convergence, we have also integrated two additional techniques, which are known as dropout regularization and momentum. Dropout regularization is a technique which involves stochastically dropping out some layer outputs during training, in order to prevent overfitting and improve model performance [1]. During training, dropout randomly deactivates a chosen proportion of neurons within a layer. To account for the dropped neurons, the outputs of the active neurons are scaled up by a factor which is equal to the probability of keeping a neuron active [1]. Our second technique, momentum, is a technique used to accelerate the optimization process during training by considering the previous updates made to the model parameters [2]. Instead of relying solely on the current gradient to update the parameters, momentum considers the accumulated history of gradients and adjusts the update direction and magnitude accordingly [2]. Additionally, we also use dynamic learning rate adaptation. After each epoch of training, if the training loss increases compared to the previous epoch, the learning rate is scaled by 0.9. Early stopping is also used: The training phase is terminated if the training error rate falls below 1%, hence preventing overfitting.

Pseudocode of the Architecture

The pseudocode of the whole Python script for the MNIST dataset classification is as follows:

1. **Data Loading and Preprocessing:** The data is loaded from idx files. The images are converted to PyTorch tensors, normalized by dividing the pixel values by 255, and flattened to create 784-dimensional vectors. Labels are then converted to one-hot encoded vectors.
2. **Definition of Helper Functions:** We have defined various helper functions, such as one-hot encoding, tanh activation and its derivative, softmax activation, sigmoid activation and its derivative, parameter initialization function, cross-entropy loss function etc.
3. **Forward Propagation with Dropout:** The forward propagation with dropout function computes the hidden layer pre-activation, which is mathematically represented as $Z_1 = W_1 x + B_1$, and its activation, which is mathematically defined as $A_1 = \tanh(Z_1)$. If dropout is randomly applied, a dropout mask D is generated such that each element of A_1 is retained with probability p and scaled by $1/p$. Then, the output layer computes $Z_2 = W_2 A_1 + B_2$, and applies the softmax activation $A_2 = softmax(Z_2)$.
4. **Backpropagation:** Gradients are computed using the chain rule. For the output layer, the gradients are computed as:
   a. $dZ_2 = A_2 - Y$
   b. $dW_2 = \frac{1}{m} dZ_2 A_1^T$
   c. $dB_2 = \frac{1}{m} \sum dZ_2$

   For the output layer, the gradients are computed as:

   d. $dZ_1 = (W_2^T dZ_2) \odot (1 - tanh^2(Z_1))$
   e. $dW_1 = \frac{1}{m} dZ_1 x^T$
   f. $dB_1 = \frac{1}{m} \sum dZ_1$
5. **Parameter Update with Momentum:** Velocity terms for each parameter are updated as:
   a. $v_{W_1} = \alpha v_{W_1} + \eta dW_1$
   b. $W_1 = W_1 - v_{W_1}$.

Similar updates are also performed for B_1, W_2, B_2, where η is the learning rate and α is the momentum coefficient.

6. **Training Loop:** For each epoch:
    a. Perform forward propagation with dropout on the training data.
    b. Compute the training loss and performance metrics.
    c. Execute backpropagation to compute gradients.
    d. Update parameters using momentum.
    e. Evaluate the network on the training and test sets without dropout.
    f. Append performance metrics for later plotting.
    g. If the training loss increases compared to the previous epoch, reduce the learning rate by ten percent.
    h. If the training error rate falls below one percent, stop training early.

Results

Our final model has achieved considerably good results. The figure given below displays the results for the last 10 epochs:

```
Epoch 240: Train Loss = 0.1166, Train Errors = 2046, Train Accuracy = 0.9659, Test Loss = 0.1273, Test Errors = 382, Test Accuracy = 0.9618
Epoch 241: Train Loss = 0.1162, Train Errors = 2046, Train Accuracy = 0.9659, Test Loss = 0.1270, Test Errors = 382, Test Accuracy = 0.9618
Epoch 242: Train Loss = 0.1158, Train Errors = 2039, Train Accuracy = 0.9660, Test Loss = 0.1266, Test Errors = 381, Test Accuracy = 0.9619
Epoch 243: Train Loss = 0.1155, Train Errors = 2031, Train Accuracy = 0.9661, Test Loss = 0.1263, Test Errors = 379, Test Accuracy = 0.9621
Epoch 244: Train Loss = 0.1151, Train Errors = 2022, Train Accuracy = 0.9663, Test Loss = 0.1261, Test Errors = 379, Test Accuracy = 0.9621
Epoch 245: Train Loss = 0.1148, Train Errors = 2018, Train Accuracy = 0.9664, Test Loss = 0.1258, Test Errors = 378, Test Accuracy = 0.9622
Epoch 246: Train Loss = 0.1145, Train Errors = 2019, Train Accuracy = 0.9664, Test Loss = 0.1256, Test Errors = 377, Test Accuracy = 0.9623
Epoch 247: Train Loss = 0.1141, Train Errors = 2015, Train Accuracy = 0.9664, Test Loss = 0.1254, Test Errors = 377, Test Accuracy = 0.9623
Epoch 248: Train Loss = 0.1138, Train Errors = 2005, Train Accuracy = 0.9666, Test Loss = 0.1251, Test Errors = 377, Test Accuracy = 0.9623
Epoch 249: Train Loss = 0.1135, Train Errors = 1997, Train Accuracy = 0.9667, Test Loss = 0.1249, Test Errors = 375, Test Accuracy = 0.9625
```

Figure 1: *Results of the model for the last 10 epochs.*

The figure given below displays the plots displaying epoch versus the number of classification errors, epoch versus energy (or loss), and epoch versus accuracy:

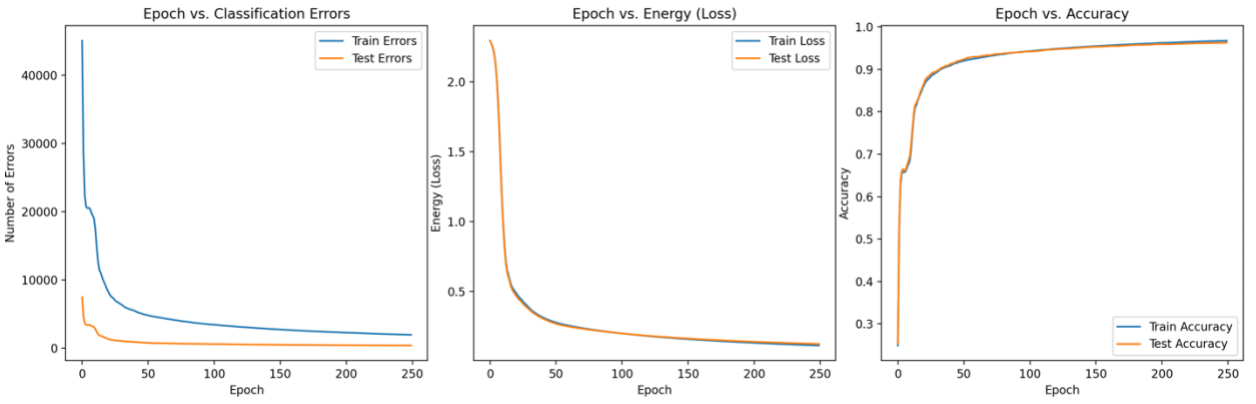

Figure 2: *Plots displaying classification errors, energy, and accuracy per epoch.*

Conclusion

In this project, we developed a neural network from scratch to classify handwritten digits from the MNIST dataset. We designed a network with 784 input neurons, a hidden layer of 150 neurons with tanh activation, and an output layer of 10 neurons using softmax activation. To improve performance, we applied techniques such as dropout, momentum-based updates, and dynamic learning rate adaptation. Our experiments showed that these methods helped the network achieve high accuracy, with training and test accuracies exceeding 96%. Overall, our approach successfully handled the classification task and produced robust results.

References

[1] GeeksforGeeks, “Dropout regularization in Deep learning,” GeeksforGeeks, https://www.geeksforgeeks.org/dropout-regularization-in-deep-learning/ (accessed Mar. 23, 2025).

[2] GeeksforGeeks, “What is momentum in neural network?,” GeeksforGeeks, https://www.geeksforgeeks.org/what-is-momentum-in-neural-network/ (accessed Mar. 23, 2025).