# Classification of Handwritten digits

Pseudocode

## Pre-Requisites:

- **Pandas**: A powerful and flexible library for data manipulation and analysis, providing high-performance data structures like the DataFrame.
- **NumPy**: The fundamental library for scientific computing in Python, providing a high-performance multidimensional array object and tools for working with these arrays.
- **Matplotlib.pyplot**: A plotting library that provides a MATLAB-like interface for creating a wide variety of static, animated, and interactive visualizations

## Function Definition:

Model Definition (DNN Class)

- **Initialize DNN:**
  - Store layer sizes.
  - Create weight matrices for each layer with random values, including bias weights.
- **Sigmoid Function:**
  - Calculate 1 / (1 + exp(-input * 0.01)).
- **Predict Function:**
  - Take input data.
  - For each layer:
    - Add a bias term to the input.
    - Calculate the weighted sum of inputs.
    - Apply the sigmoid function.
  - Return the output of the last layer.

- **Train Function:**
    - Take input data and true label.
    - Perform a forward pass (using steps similar to predict) to get the output and intermediate layer outputs.
    - Calculate the error at the output layer.
    - Perform backpropagation to calculate gradients for each weight matrix.
    - Update weights using gradients and a learning rate.
    - Return the calculated error.

## Main Code Execution:

Data Loading and Preprocessing

- Load training data (images and labels) from mnist_train_small.csv.
- Normalize image pixel values to be between 0 and 1.
- One-hot encode the labels.
- Load test data (images and labels) from mnist_test.csv.
- Normalize test image pixel values.

3. Model Training

- Create a DNN model instance with specified layer sizes.
- Initialize a queue to track recent errors.
- Loop for a specified number of training steps:
    - Randomly select a training sample and its corresponding encoded label.
    - Train the model using the selected sample and label.
    - Add the calculated error to the error queue.
    - Periodically (e.g., every 1000 steps):
        - Print the current step and average error from the queue.
        - Display the image of the training sample.
        - Print the model's prediction for the training sample.

4. Model Testing

- Initialize a counter for correct predictions.
- Loop through each sample in the test data:
    - Get the model's prediction for the test image.
    - Compare the prediction to the true label.
    - If the prediction is correct, increment the counter.
- Calculate the percentage of correct predictions.
- Print the percentage of correct predictions.

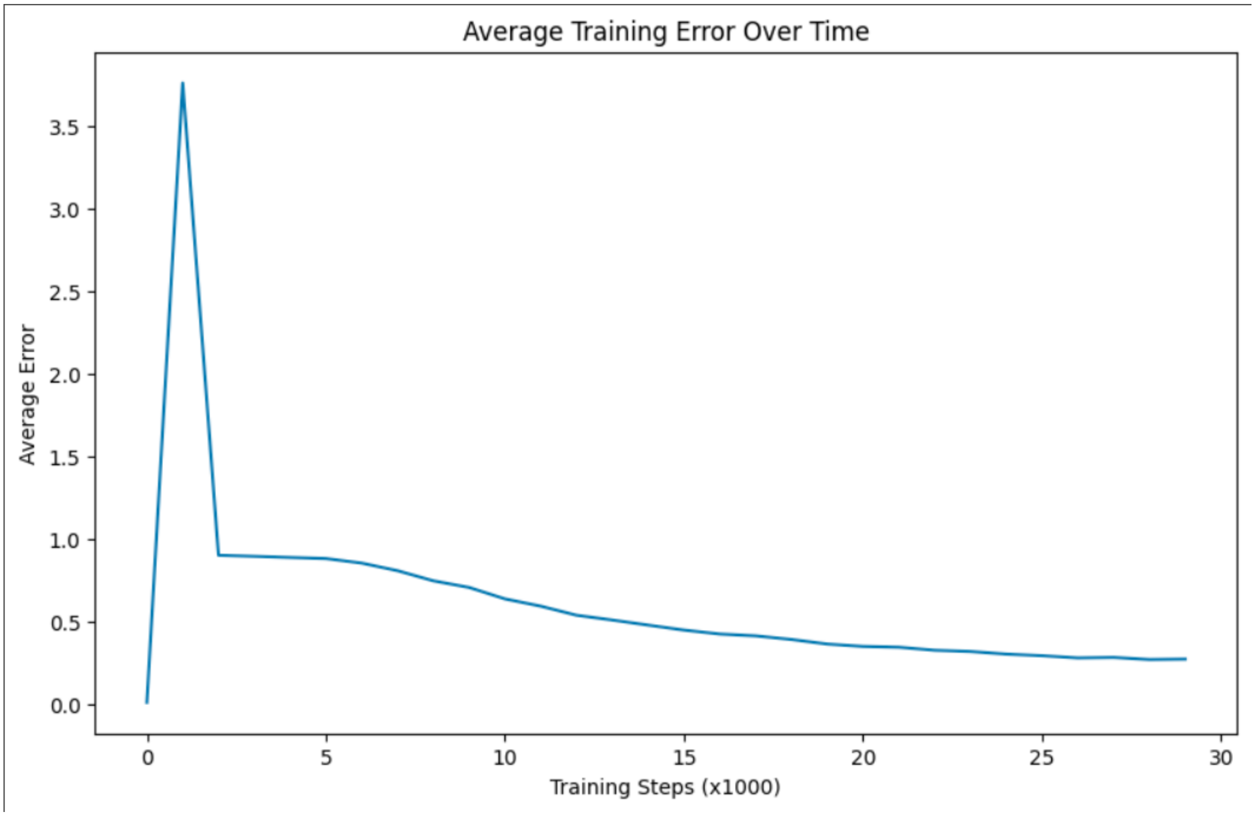
## Comparision [Increase in Neurons]:

[C] - Code Execution as-is from the article: Here is the error percentage for every 1000 steps:

| Step | Average Error |
|---:|---:|
| 0 | 0.0085422 |
| 1000 | 1.407 |
| 2000 | 0.898235 |
| 3000 | 0.897382 |
| 4000 | 0.892662 |
| 5000 | 0.88387 |
| 6000 | 0.867767 |
| 7000 | 0.8378 |
| 8000 | 0.779796 |
| 9000 | 0.743291 |
| 10000 | 0.692835 |
| 11000 | 0.653548 |
| 12000 | 0.583941 |
| 13000 | 0.56098 |
| 14000 | 0.509175 |
| 15000 | 0.478133 |

| | |
|---|---|
| 16000 | 0.469609 |
| 17000 | 0.455497 |
| 18000 | 0.426183 |
| 19000 | 0.382638 |
| 20000 | 0.382546 |
| 21000 | 0.340882 |
| 22000 | 0.34964 |
| 23000 | 0.335699 |
| 24000 | 0.335802 |
| 25000 | 0.335712 |
| 26000 | 0.314881 |
| 27000 | 0.289233 |
| 28000 | 0.27962 |
| 29000 | 0.270752 |

Percentage of Correct Prediction: 88.06%

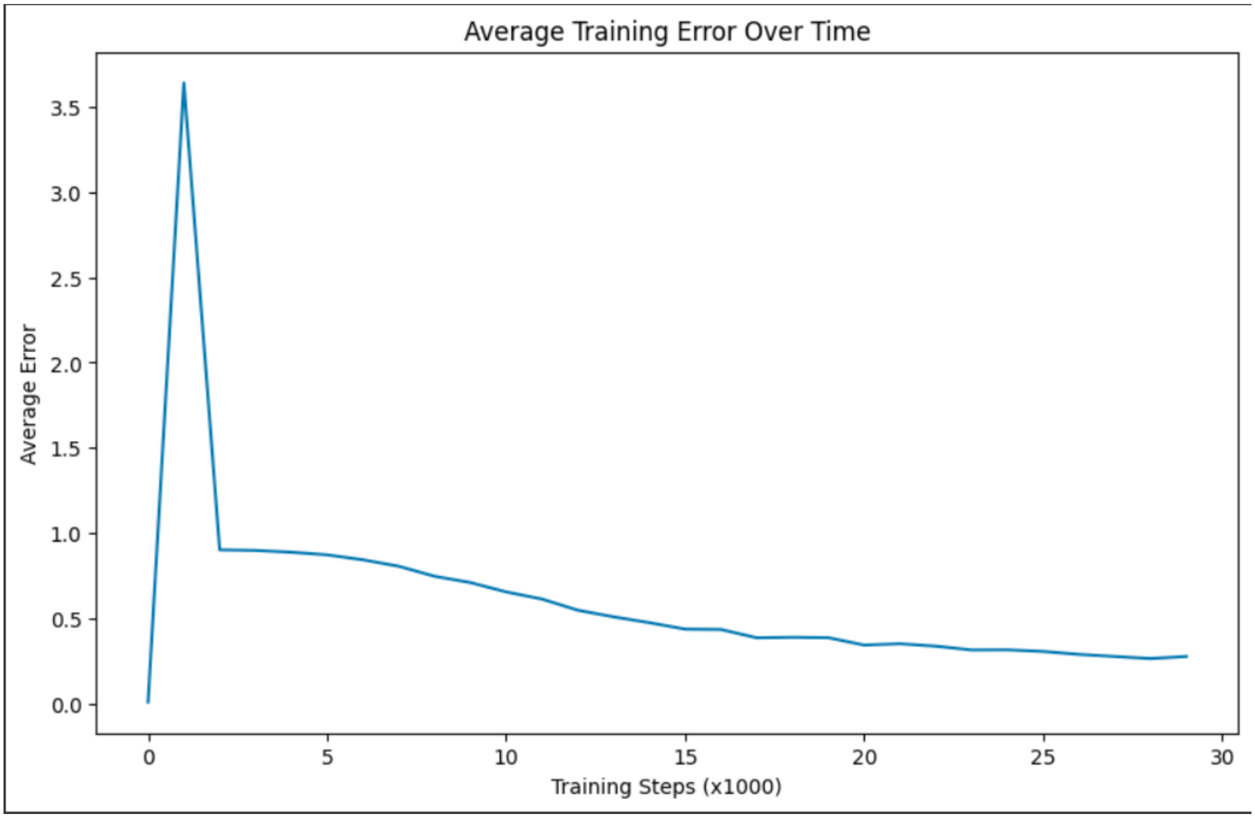



[D] - Neurons increased from 1250 to 2000 and below is the error rate per 1000 steps:

| Step | Average Error |
|---|---|
| 0 | 0.008959 |
| 1000 | 3.639592 |
| 2000 | 0.901995 |
| 3000 | 0.898259 |
| 4000 | 0.887977 |
| 5000 | 0.872487 |
| 6000 | 0.843606 |
| 7000 | 0.805622 |
| 8000 | 0.746954 |
| 9000 | 0.710006 |
| 10000 | 0.655241 |
| 11000 | 0.613281 |
| 12000 | 0.547981 |
| 13000 | 0.509071 |

| | |
|---|---|
| 14000 | 0.474802 |
| 15000 | 0.437017 |
| 16000 | 0.43482 |
| 17000 | 0.386118 |
| 18000 | 0.389009 |
| 19000 | 0.386681 |
| 20000 | 0.343399 |
| 21000 | 0.351219 |
| 22000 | 0.337011 |
| 23000 | 0.314855 |
| 24000 | 0.315485 |
| 25000 | 0.306064 |
| 26000 | 0.288852 |
| 27000 | 0.276523 |
| 28000 | 0.264435 |
| 29000 | 0.276673 |

Percentage of Correct Prediction: 87.59%




Conclusion:

Increasing the number of neurons in the hidden layers from 1250 to 2000 did not improve the network's performance. The observation is that performance degraded, going from 88.06% to 87.59%.

## References:


1. **Kaggle.** (2018). **MNIST in CSV**. Retrieved from https://www.kaggle.com/datasets/oddrationale/mnist-in-csv