Digit-Recognition Using MLP

Neural Networks Project

Fundamentals of Computational Intelligence
Ali Akbar Ahrari

Neural Networks Project

Digit-Recognition Using MLP

Course: Fundamentals of Computational Intelligence

Student Name: Ali Akbar Ahrari

Teacher Name: Mr. Tabealhojeh

Table of Contents

Note

All codes are available in this directory.

Implementation

Libraries

NumPy: For handling numerical operations and array manipulations.

Matplotlib: For visualizing data and results.

MNIST Dataset: For loading the dataset of handwritten digits.

Sequential Model: For creating a neural network model.

Layers: For defining the structure of the neural network.

to_categorical: For preprocessing the labels.

EarlyStopping: For optimizing the training process by stopping early if necessary.

Loading and Pre-processing the Data

Loading the Dataset

The MNIST dataset is loaded using TensorFlow's Keras API, which provides a convenient method to access the dataset.

Normalizing the Images

The pixel values are scaled to the range [0, 1] to facilitate faster and more effective training.

Converting Labels

The labels are converted from integers to one-hot encoded vectors, which is a common practice in classification tasks to make the labels suitable for the neural network's output layer.

Creating the Model

Layers

Different number of layers in a neural network, often referred to as the depth of the network, affects the complexity and the ability of the model to capture patterns in the data.

Shallow Networks (Few Layers):

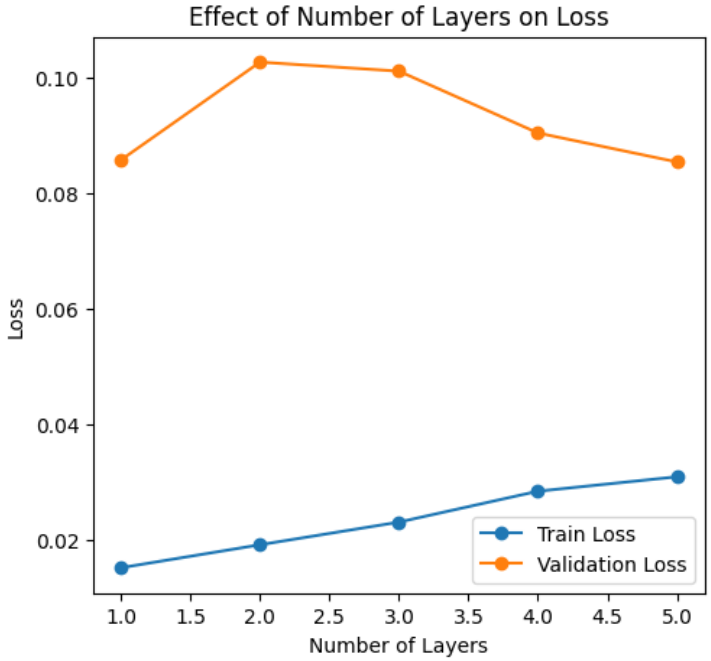
- Pros: Simpler, less prone to overfitting, faster to train.
- Cons: May not capture complex patterns, limited capacity to model non-linear relationships.

Deep Networks (Many Layers):

- Pros: Greater capacity to learn complex patterns, better at modeling non-linear relationships.

- Cons: Prone to overfitting, requires more data, computationally expensive, harder to train (risk of vanishing/exploding gradients).

In the MNIST classifier, using a deeper network may help in capturing more intricate features of the digits but also increases the risk of overfitting if not managed properly.



Neurons

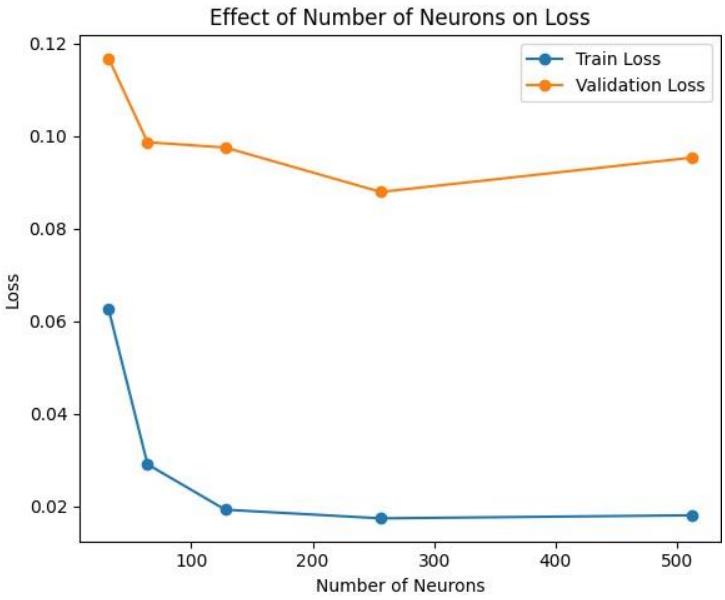
The number of neurons in each layer, often referred to as the width of the layer, determines the representational capacity of that layer.

Fewer Neurons

- Pros: Simpler model, less computationally expensive, faster to train.
- Cons: Limited capacity to learn features, may underfit the data.

More Neurons:

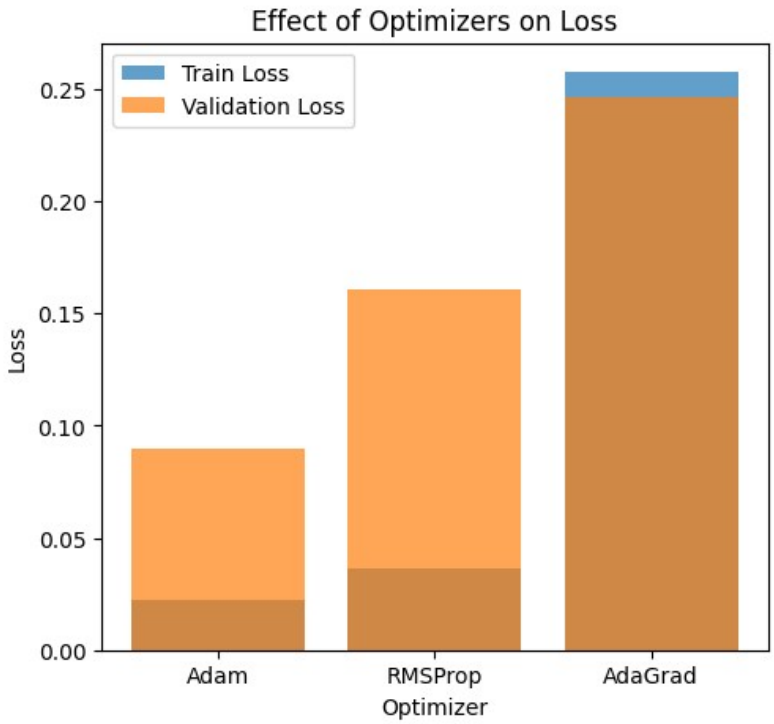
- Pros: Higher capacity to learn features, can capture more details in the data.
- Cons: Increased risk of overfitting, more computationally expensive, may require more regularization.



Optimization Algorithm

Optimization algorithms use different techniques to test and evaluate combinations of hyper-parameters, to find the optimal configurations in terms of model performance.

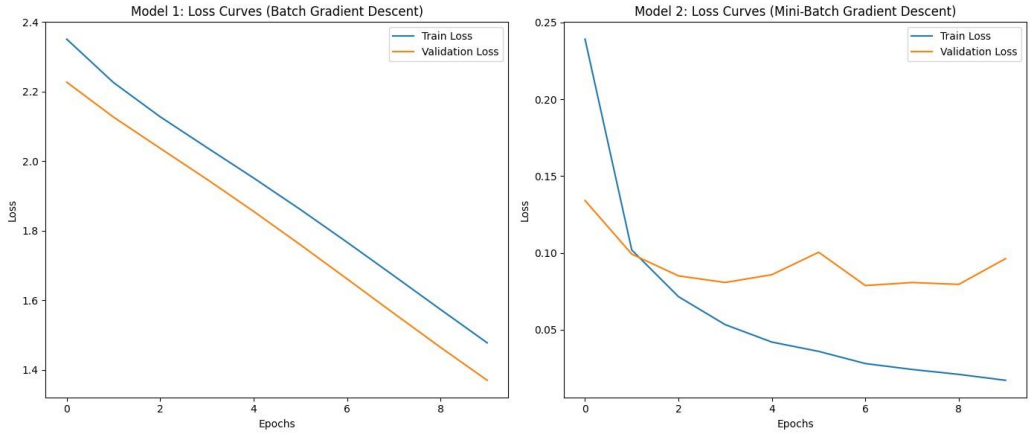
Here, we can test 3 different algorithms: Adam, RMSProp and AdaGrad. Here are the results:



Learning Algorithm

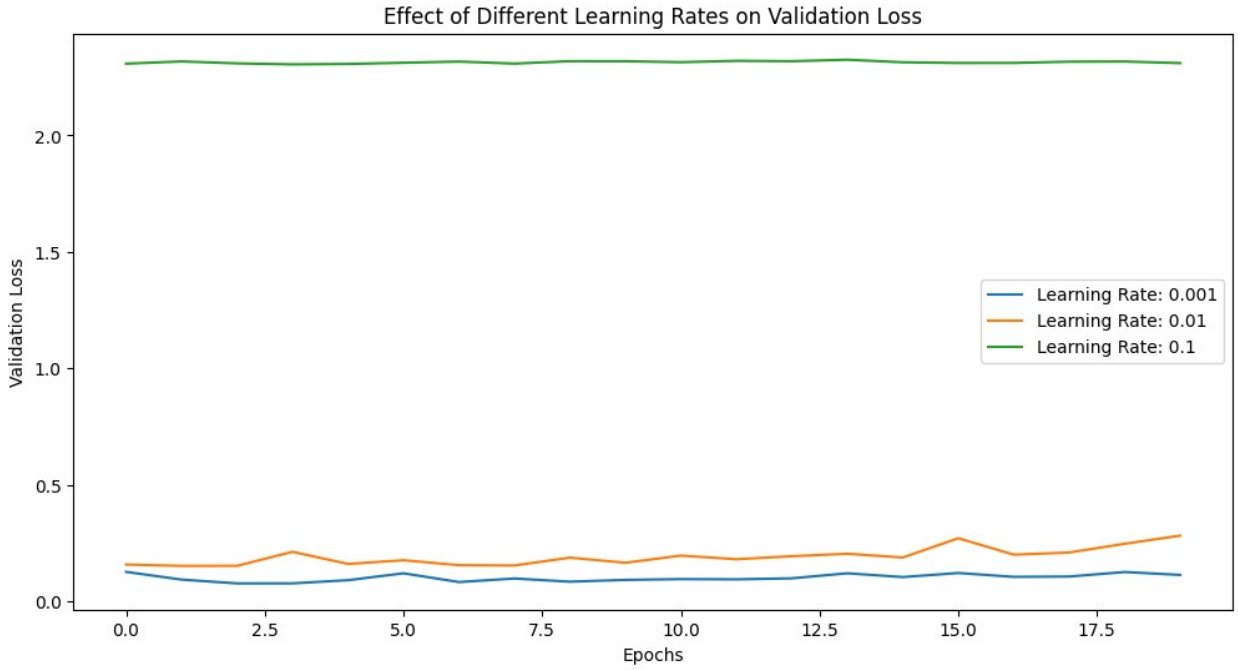
This code allows for the construction, training, evaluation, and visualization of neural network models using different learning algorithms. Batch Gradient Descent (BGD) and Mini-Batch Gradient

Descent (MBGD). The build_model function creates and compiles the model, the plot_effect_of_learning_algorithms function trains the model with different batch sizes, and the plot_results function visualizes the effect of the learning algorithms on the model's performance.



Learning Rate

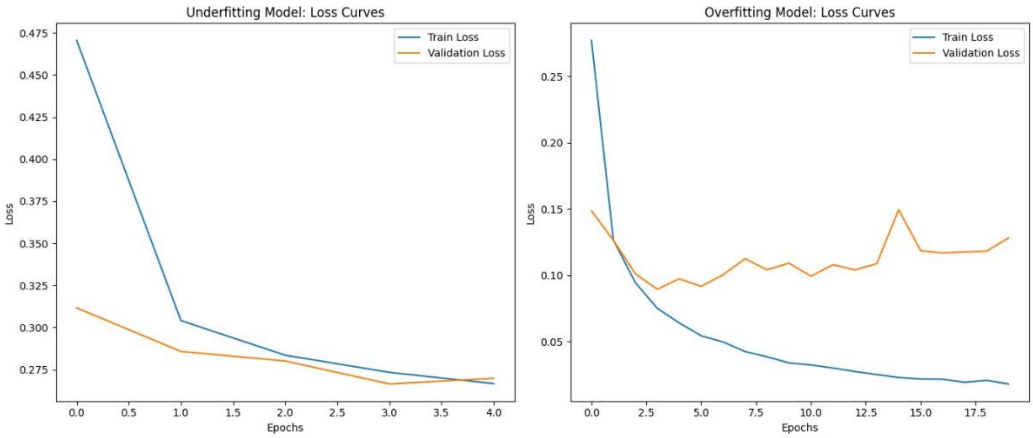
We can compare different learning rates with this picture shown below. As we could see, as learning rate increases, chance to get bigger losses increases as well.



Over-fitting and Under-fitting

The under-fitting model is too simple to capture the underlying patterns, while the overfitting model is too complex and learns noise in the training data. The loss curves comparison helps visualize the trade-off between bias and variance in model performance. I tested two different models for showing this error: one with only one layer with 10 neurons and the other with 5 layers and too many neurons. As we can see, in the under-fitting model, at the start of both the test and the train set, the error is decreasing.

But in the over-fitted model, the noise in training set decreases by each epoch. But in the test set, it does not follow any constant rule.

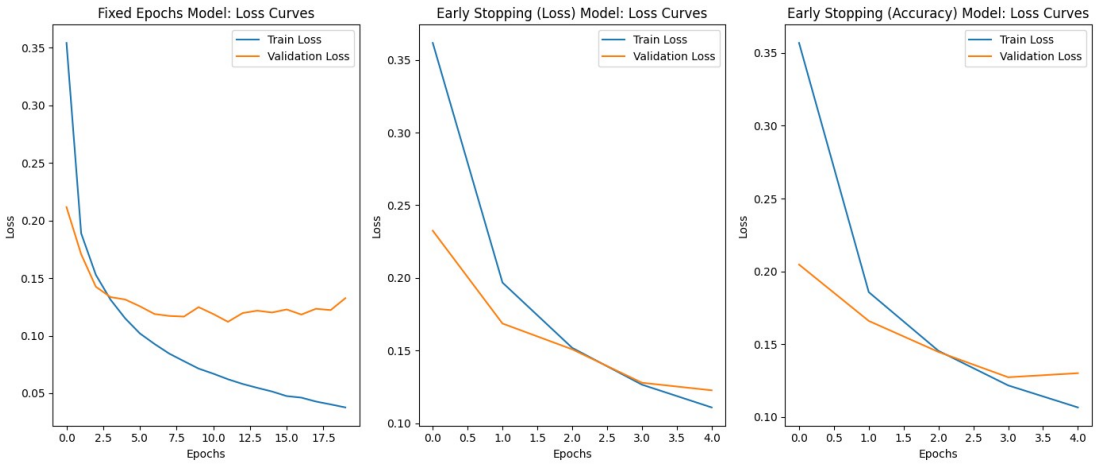


Stopping Criteria

Using appropriate stopping criteria is crucial for training models on the MNIST dataset effectively. The most common methods include early stopping based on validation performance, setting a fixed number of epochs, monitoring convergence, and limiting training time. These methods help in preventing overfitting, saving computational resources, and ensuring the model's performance is optimal.

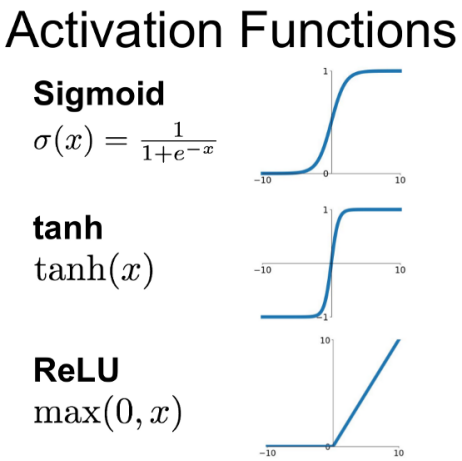
To compare models using different stopping criteria, we can train three models with the following criteria:

- Fixed number of epochs: Train for a fixed number of epochs.
- Early stopping based on validation loss: Stop training when the validation loss stops improving.
- Early stopping based on validation accuracy: Stop training when the validation accuracy stops improving.

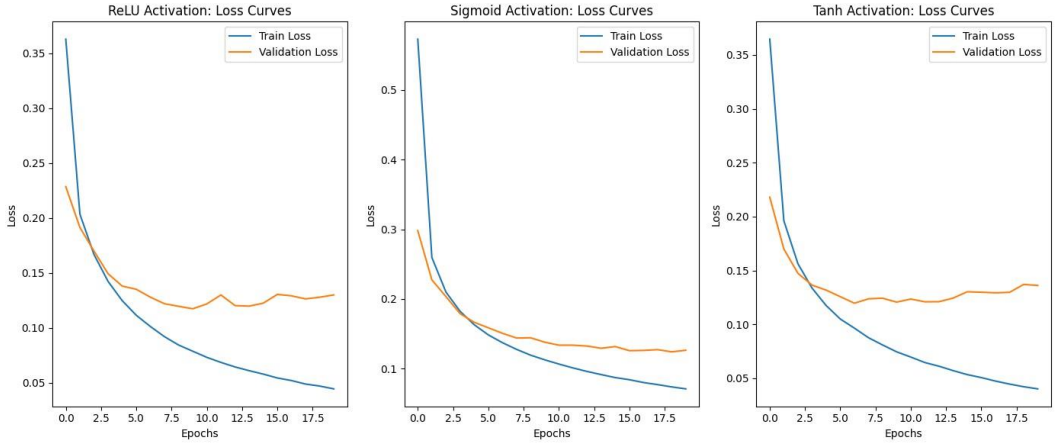


Activation Function

For the MNIST dataset, ReLU is commonly used in hidden layers due to its efficiency, while Softmax is used in the output layer for multi-class classification. Other activation functions like Sigmoid, Tanh, Leaky ReLU, and ELU can also be used based on specific requirements and to address issues like vanishing gradients and neuron saturation.



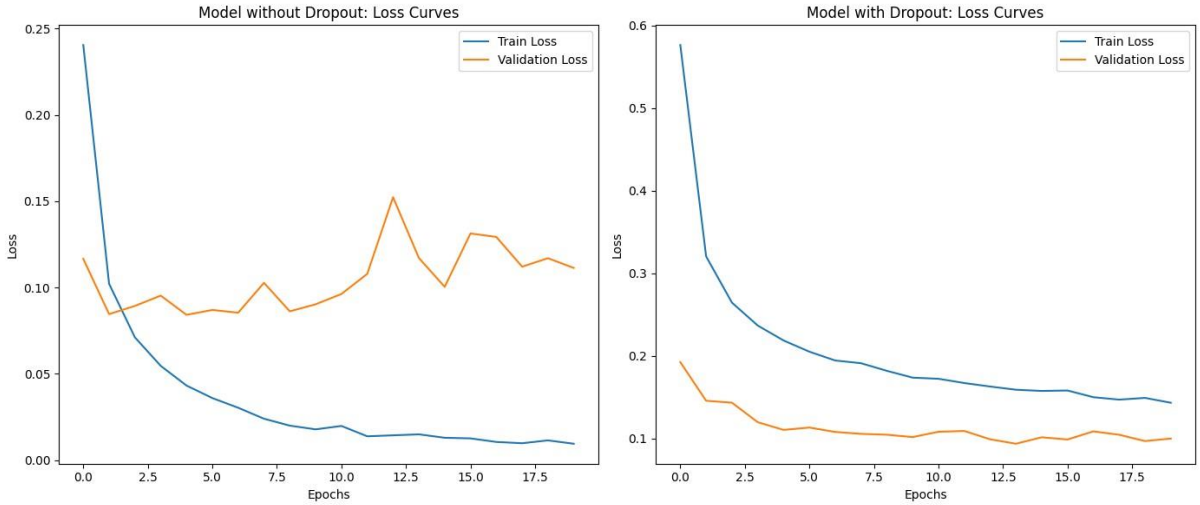
- ReLU (Rectified Linear Unit): Commonly used activation function for hidden layers.
- Sigmoid: Typically used in the output layer for binary classification, but we can also test its effect in hidden layers.
- Tanh (Hyperbolic Tangent): Similar to Sigmoid but output ranges from -1 to 1.



Dropout

Dropout refers to the practice of disregarding certain nodes in a layer at random during training. A dropout is a regularization approach that prevents overfitting by ensuring that no units are codependent with one another.

As we can see in this picture, using dropout could cause less noise and loss in the model. Note that this would be possible when using many layers and neurons.



Batch Normalization

Batch normalization is a technique to improve the training and performance of neural networks. It works by normalizing the inputs of each layer to have a mean of zero and a variance of one, thus stabilizing the learning process and allowing for faster convergence.

Batch normalization has a significant positive impact on training neural networks for the MNIST dataset by:

- Accelerating training speed.
- Enhancing generalization.
- Providing greater training stability.
- Reducing sensitivity to hyperparameters.

Incorporating batch normalization in your neural network can lead to better and more efficient training, resulting in improved performance on both training and test data.

The picture below shows the effect of using this normalization. As we can see, noise and error of the model has decreased by epochs and risk of overfitting has almost been solved.