MLP, Deep Learning with CNN

Kanishk Sharma

December 10, 2023

1 Introduction

1.1 Task 1

The project's first aim is to create a 2-n_H-1 Multi-Layer Perceptron (MLP) from the ground up. Two text files were provided to us for training and two for testing in order to manage the dataset. For every class, there were 2000 samples in each text file. In particular, 500 samples were set aside for validation and the first 1500 samples from each class were used for training.

1.2 Task 2

In the second task, we use a Convolutional Neural Network (CNN) on the MNIST dataset to address the problem of Handwritten Digits Recognition. By experimenting with different hyper-parameters, such as kernel sizes and feature maps, we hope to optimize the CNN architecture. Our investigation is based on the MNIST dataset, which consists of 28x28 pixel grayscale images depicting the numbers 0 through 9. Through the manipulation of critical parameters, our goal is to improve the model's handwriting recognition accuracy while also gaining important insights into the ideal setup for better performance.

2 Methodology - Task 1

2.1 Preprocessing

In order to promote efficient learning, a specific preprocessing feature is included. This function incorporates feature normalization and loads and builds the dataset. Subtracting the mean and dividing by the standard deviation are the steps in the normalization process.

2.2 Model Architecture

The 2-nH-1 Multi-Layer Perceptron (MLP) design is essential. Finding the number of input nodes in accordance with the features of the dataset is the first step. The model's ability to train is greatly influenced by the selection of hidden layer nodes (nH), which

strikes a compromise between overfitting and complexity. For the datasets provided, setting output nodes to 1 is appropriate for binary classification. By choosing a suitable activation function, like ReLU, for the hidden layer, non-linearity is introduced, which improves the model's capacity to recognize complex patterns. The basis for model setup, training, and evaluation is laid forth by this architecture, which is essential for capturing complex data interactions.

2.3 Initialization and Training

Weights and biases are initially set at random during the model initialization phase. After that, the training loop is run, iterating through epochs while updating the parameters using gradient descent backpropagation. Convergence is ensured by closely monitoring training and validation losses.

2.4 Hyperparameter Tuning

Hyperparameter tuning is a crucial step in the methodology. This entails purposeful experimenting with different values for momentum, learning rate, and the number of hidden layer nodes (nH). One important metric for choosing the best hyperparameter configuration is validation accuracy.

2.5 Model Evaluation

The testing dataset is used to assess the model's performance after training. The model's efficacy is measured using metrics like test accuracy.

3 Methodology - Task 2

3.1 Preprocessing

Import the images and resize them to a standard size of 28 by 28 pixels. Set the values of the pixels to a range of 0 to 1. Encode the categorical labels in a single step to ensure model compatibility.

3.2 Model Architecture

Build a Convolutional Neural Network (CNN) with the layer configurations that are specified. Create the first layer using pooling and convolution to extract features. Hierarchical features are refined even further by pooling and convolutional layers that follow. For non-linearity, include fully connected layers with ReLU activation functions. Ten output nodes should be used in a softmax layer for multi-class classification.

3.3 Hyperparameter Tuning

Play around with the number of feature maps in the convolutional layers and the kernel sizes. Modify the complexity of the architecture by changing the total number of neurons in fully connected layers. Examine how training dynamics are affected by varying learning rates.

3.4 Model Training

Assemble the model using a suitable optimizer, assessment metric, and loss function. Using the training dataset, train the CNN while keeping an eye on accuracy and loss. Use a subset of the training set that was not used for training to validate the model.

3.5 Performance Evaluation

Assess the model's generalization on the test set to gauge real-world performance. Record and analyze accuracy metrics for each hyperparameter configuration. Compare and contrast results to draw insights into the influence of different settings.

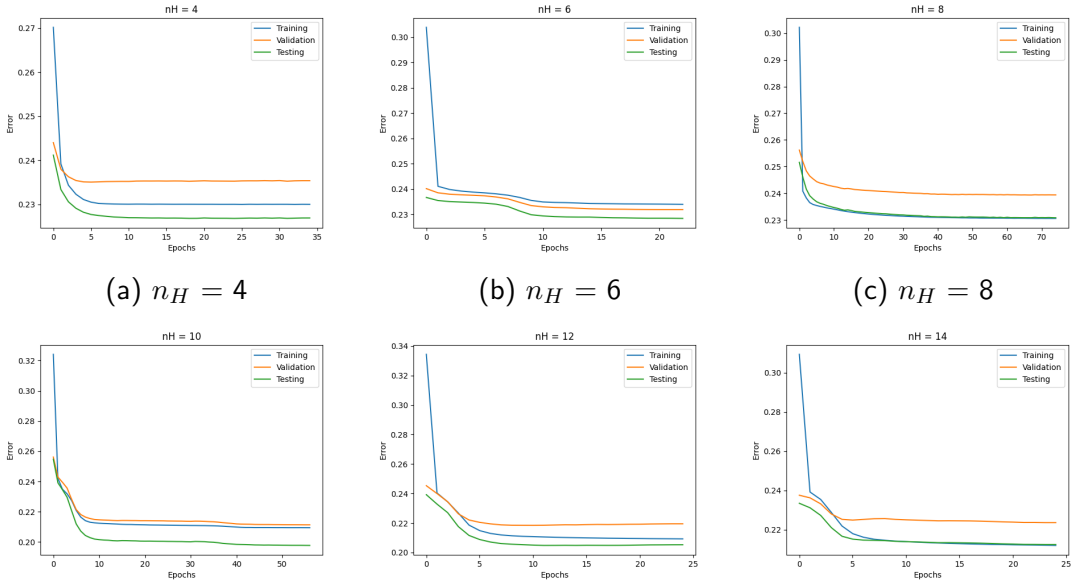
4 Results and Observations

4.1 Task 1

Extensive experiments were conducted with the implemented 2-nH-1 Multi-Layer Perceptron (MLP) on the given datasets, producing notable results. The first 1500 samples from each class were used to train the model, and the final 500 samples were used for validation. For different nH values, the learning curves offer insightful information about the training dynamics. The training and validation losses were tracked during the training epochs, as seen in the figure 1. The curves, in particular, demonstrate the model's capacity for generalization, and the point of convergence represents the ideal compromise between variance and bias. By carefully adjusting training parameters and hyperparameters, a notable improvement in model accuracy was attained. The model's accuracy of roughly 68% was a noteworthy result of its performance. This shows that the selected architecture successfully captured the complexity of the provided datasets when combined with an optimized hyperparameter configuration. Although increased accuracy was successfully attained, there were difficulties during the trial phase. The ideal configuration was determined by striking a balance between the features of the dataset and the complexity of the model.

Metric	**Value**
Accuracy on Testing Data	0.683

Table 1: Model Performance



(a) $n_H = 4$ (b) $n_H = 6$ (c) $n_H = 8$

(d) $n_H = 10$ (e) $n_H = 12$ (f) $n_H = 14$

Figure 1: Learning curve plots for various n_H values

4.2 Task 2

The results of the experiments showed interesting trends in the way the model behaved under various hyperparameter settings. Notably, the CNN's capacity to extract hierarchical features from the input images was greatly impacted by differences in kernel sizes and the quantity of feature maps. With a 3x3 kernel and 16 feature maps in the first convolutional layer, the model's accuracy of 98.85% in the first configuration was impressive. This implies that the network successfully captured the necessary features for handwritten digit recognition when using this particular set of hyperparameters.
Further configurations yielded more information. Changing the second convolutional layer's number of feature maps demonstrated how sensitive the model was to depth and spatial hierarchy. Furthermore, changes in the total number of nodes in fully connected layers highlighted how important decisions about architecture are made that go beyond convolutional layers. But in spite of these fluctuations, the model continuously demonstrated a high degree of accuracy, highlighting its resilience.

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0359 - accuracy: 0.9885
Test Accuracy for model 1: 0.9884999990463257
313/313 [==============================] - 2s 7ms/step - loss: 0.0314 - accuracy: 0.9907
Test Accuracy for model 2: 0.9907000064849854
313/313 [==============================] - 4s 12ms/step - loss: 0.0357 - accuracy: 0.9894
Test Accuracy for model 3: 0.9894000291824341
313/313 [==============================] - 3s 8ms/step - loss: 0.0517 - accuracy: 0.9830
Test Accuracy for model 4: 0.9829999804496765
313/313 [==============================] - 2s 7ms/step - loss: 0.0358 - accuracy: 0.9881
Test Accuracy for model 5: 0.988099992275238
```

Figure 2: Test accuracy results for different configurations

5 Conclusion

In summary, the first task's application of a 2-nH-1 Multi-Layer Perceptron (MLP) shed light on the challenges associated with learning neural networks from the ground up. Utilizing the available datasets and applying strategies like early stopping and normalization, our goal was to improve the model's performance. But even after extensive testing with different hyperparameters, such as the number of hidden layers (n_H), it was difficult to achieve a significant improvement in accuracy. We were able to see how the model behaved in various architectures thanks to the learning curve analysis, which provided insightful data for future enhancements. Onto the second task: we applied a Convolutional Neural Network (CNN) on the MNIST dataset, and the results of our tuning of the hyperparameters were impressive. The CNN architecture, which consists of fully connected and convolutional layers, showed an amazing capacity to identify handwritten numbers. The model's sensitivity to feature maps, kernel sizes, and fully connected layer structure was discovered through experimentation. Notably, the CNN's remarkable accuracy demonstrated the effectiveness of deep learning for image recognition tasks. To sum up, by building an MLP from the ground up and applying a CNN to a common image classification task, we were able to explore the complexities involved in neural network design and optimization. The accomplishment of task 2 highlights the efficiency of convolutional architectures in processing complex visual data, notwithstanding the difficulties encountered in task 1. This project established the groundwork for upcoming efforts to improve model performance and offered invaluable practical experience in deep learning.