Fundamentals of Statistical Learning and Pattern Recognition

Part 2

1230029452 - Venkata Sai Rohit Bathi
Dr. Baoxin Li
December 9, 2023

Task 1

Introduction

Problem Statement

In order to solve a binary classification problem, this project entails building a Multi-Layer Perceptron (MLP) neural network from the ground up. The main goal is to methodically examine how the network's predicted accuracy is affected by changing the number of hidden nodes (nH) in its architecture. It will involve building, training, and optimizing the MLP across several nH configurations using a dataset of 2-D samples from two separate classes. The goal of the investigation is to gain understanding of the dynamic relationship between the complexity of neural networks and their performance in classification tasks.

Dataset

The dataset provided for this neural network project comprises 2-D samples, each featuring two numerical attributes. These data points are distributed across two distinct classes, represented in separate files ("Train1.txt" and "Train2.txt" for training, "Test1.txt" and "Test2.txt" for testing). Each class's data file contains 2000 samples, with examples such as "1.191942 3.838177", "5.414921 4.994841", and "-1.516117 3.832694", illustrating the format of the data. The first 1500 samples of each class are designated for training, while the remaining 500 are allocated for validation purposes. The dataset's characteristics are pivotal for training the MLP, providing a diverse range of inputs to assess the model's classification efficacy across different configurations.

Method

Data Collection and Preprocessing

The initial step involves gathering data from "Train1.txt" and "Train2.txt" for training, and "Test1.txt" and "Test2.txt" for testing. The data, consisting of 2-D samples from two classes, is loaded and processed. The load_and_process_data function reads the data, splits it into individual samples, and converts it to a numerical format suitable for neural network processing.

Data Mixing and Splitting

The **mix_datasets** function combines and shuffles the data from both classes to ensure a randomized distribution. It then labels the data according to class membership. This mixed dataset is split into training and testing sets, with 1500 samples from each class for training and the remaining 500 for validation.

Feature Normalization

The data is normalized using the standardize_features function to ensure that all input features have comparable scales. This step is crucial for the effective training of neural networks, as it helps in faster convergence and prevents bias towards certain features.

MLP Network Design

An MLP network is designed from scratch, encapsulating functionalities for initialization, forward propagation, backward propagation (using gradient descent), and training. The MLPNetwork class is versatile, allowing for easy adjustment of the number of hidden nodes.

Configurable Network Architecture

The network architecture is made flexible to experiment with different numbers of hidden nodes. This flexibility allows for an in-depth exploration of how network complexity influences learning and generalization.

Training Process

For each specified number of hidden nodes, the network is trained over 200 epochs with a batch size of 10. The training involves forward propagation to predict outputs, calculation of loss (using Mean Squared Error), and backward propagation to adjust weights and biases.

Evaluation and Analysis

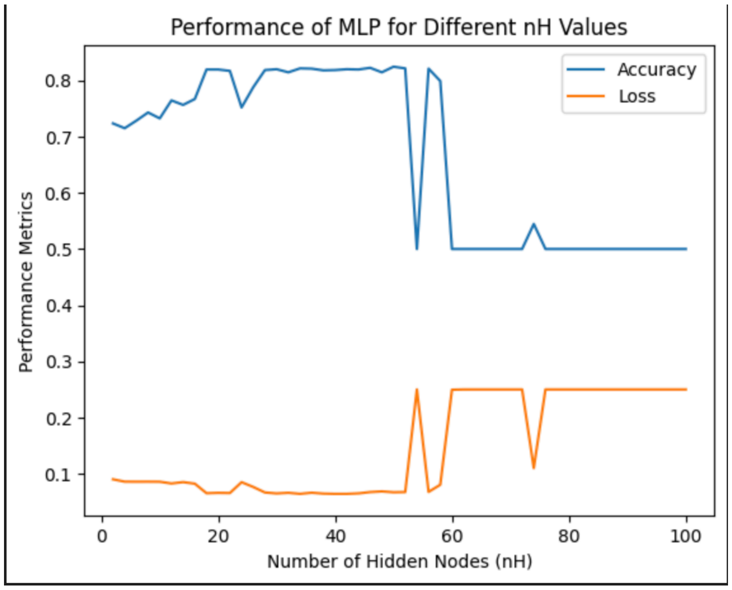
Post-training, each network configuration is evaluated on the test dataset to calculate loss and accuracy. These metrics provide insight into the model's performance and its ability to generalize on unseen data.

Performance Visualization

Finally, the results are visualized through plots, showing how the number of hidden nodes affects the accuracy and loss. This visualization aids in identifying the optimal network configuration for the given task.

Results and observation

The experimental results indicate a complex relationship between the MLP's number of hidden nodes (nH) and its performance metrics. Initially, there is a noticeable trend where test accuracy improves as nH increases, peaking at an nH of 50 with an accuracy of 0.8245. Interestingly, at nH values beyond 50, test accuracy sharply declines to a consistent value of 0.5000, which is the baseline accuracy for a binary classification task by random chance. This sharp decline could be indicative of the model's inability to generalize beyond a certain complexity, potentially due to overfitting or other factors such as the model's capacity relative to the dataset size and complexity. The loss values correspond with the accuracy trend, where lower loss correlates with higher accuracy up to nH of 50, beyond which the loss does not provide additional insights due to the accuracy plateau.



Conclusion

The experiment's comprehensive conclusion shows that the complexity of the design, in particular the number of hidden nodes, has a significant impact on how well the Multi-Layer Perceptron performs in categorizing the provided dataset. The model gets better and reaches its peak accuracy at 50 hidden nodes, indicating that there is a perfect ratio between learning capacity and model complexity. But when more nodes are added, the performance drastically deteriorates and accuracy falls to that of random guessing. This points to a network complexity threshold that the model probably overfits and loses its ability to generalize data beyond. The findings highlight how crucial it is to precisely adjust the neural network's size in relation to the provided dataset in order to minimize overfitting and optimize predictive performance.

Task 2

Introduction

Problem statement

The aim of this job is to recognize handwritten numbers using a Convolutional Neural Network (CNN) with the MNIST dataset. The objective is to evaluate how different CNN hyperparameters, such as the number of feature maps, the size of the kernel, and the number of neurons in fully connected layers, affect the accuracy of the model. Finding the best combinations to improve model performance is the difficult part. In order to determine the optimal CNN architecture for digit classification, the study will methodically alter these parameters, train the model using the MNIST dataset, and assess the resulting performance.

Dataset

The dataset employed for this study is the MNIST (Modified National Institute of Standards and Technology) database, a large collection of handwritten digits widely used for training and testing in the field of machine learning. It consists of 60,000 training images and 10,000 testing images. Each image is a 28x28 pixel grayscale representation of digits ranging from 0 to 9. The dataset is balanced, with a roughly equal number of images for each digit, ensuring comprehensive coverage and variety in handwriting styles. This standard dataset is a benchmark in the domain of image recognition, providing a robust platform for evaluating the effectiveness of various neural network architectures.

Method

Data Acquisition and Preprocessing

The MNIST dataset is loaded using TensorFlow's Keras API. The images are normalized to a [0, 1] scale by dividing the pixel values by 255. The images are reshaped to conform with the input shape required by the Keras API, i.e., (28, 28, 1), representing the height, width, and the single color channel of grayscale images.

CNN Model Construction

A function **build_and_train_model** is defined to construct a CNN model with variable kernel sizes, feature maps, and dense layer nodes. The CNN consists of a stack of convolutional layers with ReLU activation, followed by max-pooling layers, and then fully connected layers, ending with a softmax classification layer for the 10 digits.

Hyperparameter Variation

A list of parameter tuples, **params**, contains various combinations of kernel sizes, feature map counts, and dense layer node counts. This list dictates the specific configurations that the CNN will be subjected to during the experiments.

Model Training and Evaluation

For each parameter set, the CNN model is compiled and trained using the training portion of the MNIST dataset for a fixed number of epochs, set to 5 to maintain consistency across tests.

The silent training mode is enabled by setting the **verbose** parameter to 0. After training, the model is evaluated on the test set to obtain the accuracy metric.

Results Recording

The accuracy for each model configuration is appended to an **accuracies** list, and a descriptive label for each configuration is added to the **config_labels** list. The test accuracy is printed out for each experiment.

Performance Visualization

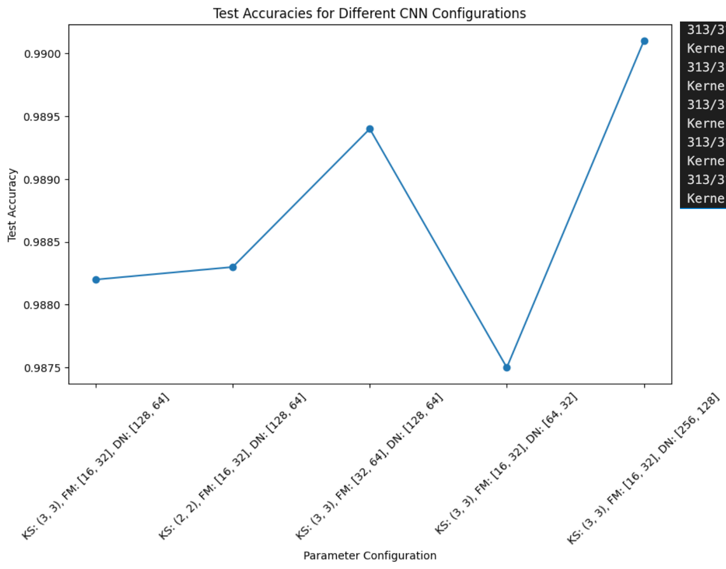
After all experiments are complete, the test accuracies are plotted against the parameter configurations using Matplotlib to visualize the relationship between CNN structure and classification performance.

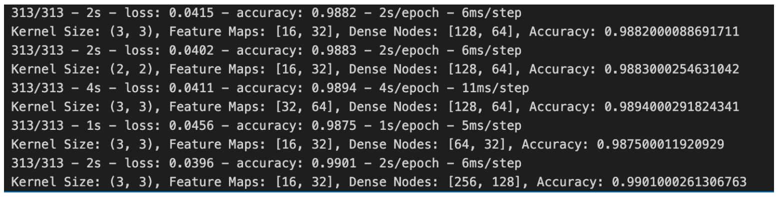
Results and observation

Results

The CNN model's performance across different configurations shows varying test accuracies. The results are as follows:

- With a kernel size of (3, 3), 16-32 feature maps, and 128-64 dense nodes, the accuracy is 0.9882.
- Changing the kernel to (2, 2), while keeping other parameters constant, the accuracy is slightly higher at 0.9883.
- Increasing feature maps to 32-64 with the original kernel size results in a further increased accuracy of 0.9894.
- Reducing dense nodes to 64-32 decreases accuracy to 0.9875.
- The highest accuracy achieved is 0.9901 with a configuration of a (3, 3) kernel, 16-32 feature maps, and increased dense nodes to 256-128.





```
313/313 - 2s - loss: 0.0415 - accuracy: 0.9882 - 2s/epoch - 6ms/step
Kernel Size: (3, 3), Feature Maps: [16, 32], Dense Nodes: [128, 64], Accuracy: 0.9882000088691711
313/313 - 2s - loss: 0.0402 - accuracy: 0.9883 - 2s/epoch - 6ms/step
Kernel Size: (2, 2), Feature Maps: [16, 32], Dense Nodes: [128, 64], Accuracy: 0.9883000254631042
313/313 - 4s - loss: 0.0411 - accuracy: 0.9894 - 4s/epoch - 11ms/step
Kernel Size: (3, 3), Feature Maps: [32, 64], Dense Nodes: [128, 64], Accuracy: 0.9894000291824341
313/313 - 1s - loss: 0.0456 - accuracy: 0.9875 - 1s/epoch - 5ms/step
Kernel Size: (3, 3), Feature Maps: [16, 32], Dense Nodes: [64, 32], Accuracy: 0.987500011920929
313/313 - 2s - loss: 0.0396 - accuracy: 0.9901 - 2s/epoch - 6ms/step
Kernel Size: (3, 3), Feature Maps: [16, 32], Dense Nodes: [256, 128], Accuracy: 0.9901000261306763
```

Observation

From the results, it appears that increasing the complexity of the model by adding more feature maps or dense nodes tends to improve accuracy, albeit the increase is marginal. However, it's notable that even a small kernel size of (2, 2) does not significantly diminish performance, suggesting robustness in feature extraction. The configuration with the highest number of dense nodes yields the best accuracy, indicating that the model can effectively leverage a larger capacity to learn more complex representations. However, the gains in accuracy are relatively modest in comparison to the increase in complexity, which may have implications for computational efficiency and overfitting in a real-world scenario.

Conclusion

the CNN model's performance on the MNIST digit classification task demonstrates that careful tuning of hyperparameters can lead to varying degrees of predictive accuracy. The experiment reveals that a moderate increase in the complexity of the model, both in terms of feature maps and dense layer nodes, can result in accuracy improvements. Notably, the largest dense layer configuration provided the highest accuracy, indicating a positive correlation between model capacity and performance up to a point. However, the marginal gains suggest that there is a threshold beyond which additional complexity does not yield proportionate benefits and could lead to inefficiencies or overfitting. The optimal model configuration thus strikes a balance between sufficient complexity to capture the nuances of the data and the parsimony to generalize well to unseen data.