UNIVERSITY OF CAMBRIDGE

M1: Machine Learning Coursework Report

Yuanzhen Zhao (yz929)

Department of Physics, University of Cambridge

December 18, 2024

Contents

Word Counts: 2663

1 Introduction

This coursework addresses the challenge of accurately summing two handwritten MNIST digits using machine learning techniques. A neural network pipeline was developed to map combined digit images to their summed labels, with the hyperparameters optimised to maximise model accuracy. Multiple classifiers were implemented and compared for the benchmark. Finally, t-distributed Stochastic Neighbour Embedding (t-SNE) was employed to visualise high-dimensional feature representations of the combined digit dataset on 2D plots.

2 Dataset Preparation

2.1 MNIST Dataset Preprocessing

The MNIST database is a widely used dataset of handwritten digit images, essential for research in machine learning and deep learning. It consists of 70,000 greyscale images of digits from 0 to 9, each with a resolution of 28×28 pixels. The dataset is split into 60,000 training samples and 10,000 test samples. Derived from the NIST database, it has been preprocessed through denoising, normalisation and alignment, making it ideal for image classification and pattern recognition tasks. MNIST has significantly contributed to the development of convolutional neural networks as a benchmark for evaluating algorithm performance.

A crucial step in training a model to sum two handwritten digits involves generating a dataset by stacking images sequentially. To begin, the MNIST dataset was loaded from Keras, where the pixel values range from 0 to 255. These values were converted to floating-point numbers and normalised by dividing by 255. The training and test datasets were then concatenated to form a unified dataset of images and labels, which was subsequently split into training, validation and test sets as required.

2.2 Combining into 28×56 Images

The `numpy.random.choice` function was used to statistically bootstrap two images and corresponding labels by selecting their indices. A random seed could be set to ensure that the pseudo-random sampling process is reproducible. Combined images could be produced by stacking two 28×28 images horizontally, resulting in a single image with dimensions of 28×56, and the labels were summed accordingly. To ensure proper horizontal stacking, the `combined_images` array was reshaped into $(-1, 28, 56, 1)$ dimensions, where -1 represents a flexible size accommodating the number of images in the dataset. This procedure could be repeated several times to generate a dataset of any desired size.

Creating a dataset from a large collection of images involves chopping the data into training, validation, and test sets. The ratio of three classes was assigned as 60 : 20 : 20, as this ratio provides a sufficient amount of data for training and validating the model during the fitting process. Specifically, the first 60% of images and labels were allocated to the training set, the next 20% to the validation set and the final 20% to the test set. Labels could be set to be one-hot encoded in a customised generating function. For this experiment, the total dataset size was set to 100,000, with labels categorised appropriately.

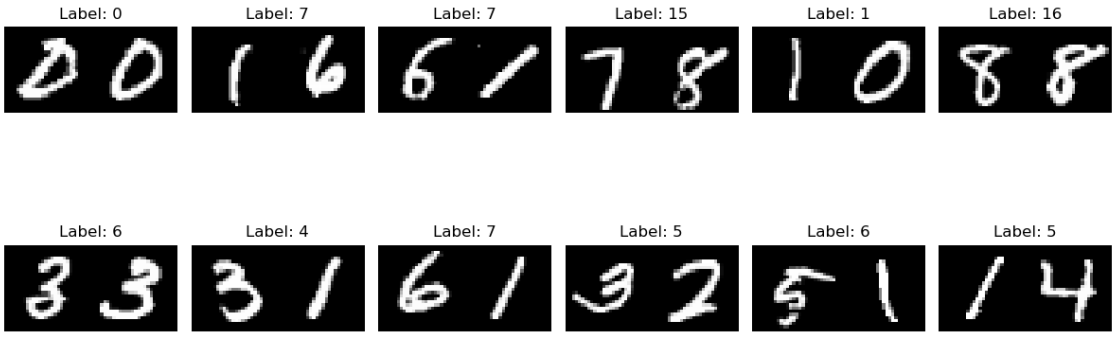

Figure 1: Examples of Horizontally Combined Images and Respective Labels

3 Neural Network Implementation

3.1 Fully-connected Neural Network

A fully connected neural network is a fundamental neural network architecture consisting of fully connected layers. Each neuron in a given layer establishes a connection with all neurons in the preceding layer. Information is transmitted through weighted sums, followed by a non-linear activation function as biases. This connection pattern is highly general-purpose and makes no assumptions about the underlying features of the data. However, it is computationally expensive, requiring significant memory for weights and resources for processing the dense connections.

- Input layer: receives input data for the neural network
- Hidden layers: transform input signals using weight matrices, with each neuron applying a non-linear activation function (e.g., sigmoid or ReLU) to generate outputs for the next layer
- Output layer: yields the prediction results of neural networks, e.g. the class probabilities in a classification problem

Various hyperparameters can be determined to optimise the performance of neural networks, including the learning rate, the number of neurons in hidden layers, activation functions, batch size, dropout rate and so on. Different combinations have arguably different performances on the final accuracy and the training time. To mitigate overfitting, regularisation techniques such as L2 regularisation and dropout could be utilised. L2 regularisation adds a penalty term to the loss function, while dropout would randomly deactivate neurons during the training process. Training in batches is another effective strategy for accelerating the training process, though it potentially leads to unstable convergence or causes the model stuck in local minima.

3.2 Activation Functions

In neural networks, activation functions are the key components for introducing non-linearity, enabling the model to learn intricate patterns in the dataset. A variety of activation functions are available, each suited for different scenarios. In this architecture, I selected two activation

functions for better performance.

3.2.1 ReLU

Rectified Linear Unit (ReLU) is mathematically defined as $\max(0, x)$. It addresses the vanishing gradient issue within the positive domain and strongly supports rapid convergence. However, ReLU can encounter the 'dying neuron' problem, where neurons become inactive for negative inputs as their gradients are zero. This issue could be mitigated by using improved versions such as Leaky ReLU, which allows small gradient propagation in a negative domain.

3.2.2 Softmax

Softmax is an activation function designed for multiclass classification tasks. It normalises the input vectors into a probability distribution across classes, mathematically defined as $\exp(x_i)/\sum_i \exp(x_i)$. Unlike a standard max function, Softmax does not discard smaller values but assigns them small probabilities, making it a 'soft' version of max or arg max. Therefore, it is typically used in the output layer rather than hidden layers.

3.3 Model Architecture

The sequential neural network model starts with an **Input layer**, where the shape should be $(28, 56, 1)$ for horizontally combined images. This is followed by a **Flatten layer**, which decreases the dimensionality of data. The third layer would be a **Dense layer** with a ReLU activation function. Next, a **Dropout layer** with a flexible dropout rate is included to avoid overfitting. The fifth layer is another **Dense layer** with a ReLU activation function. The final layer is a **Dense layer** with a Softmax activation function, having the same number of neurons as the number of output classes. The architecture is illustrated in the following instruction diagram.

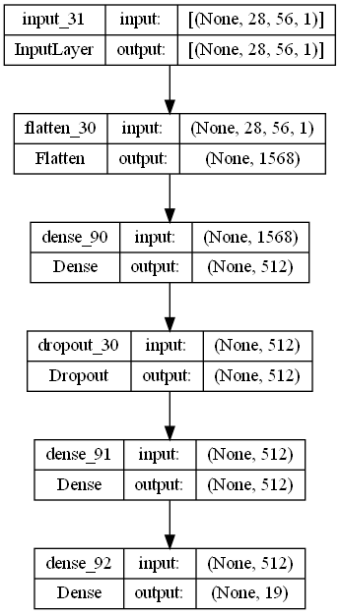

Figure 2: Architecture of My Best Neural Network Plotted by `plot_model` Function

3.4 Hyperparameter Optimisation

To optimise the model, five hyperparameters were tuned using OPTUNA:

- **number of units** in Dense layer 1, ranging from 256 to 512, to explore moderate to high model complexity.
- **number of units** in Dense layer 2, also ranging from 256 to 512 to complement the complexity of the model.
- **dropout rate** in the Dropout layer, ranging from 0.1 to 0.5 to reduce overfitting.
- **learning rate** of the ADAM optimiser, from 0.001 to 0.01 to control the step size of weight updates.
- **batch size** in the training process tested with values of 128, 256 and 512, to balance training speed and robustness.

This configuration ensures a balance between complexity, regularisation and computational efficiency, allowing the model to perform well without overfitting or excessive training time.

3.5 Training and Validation

To train the neural network effectively, the combined dataset was split into training, validation and test sets. The training set was used to optimise the model weights within each neuron, whereas the validation set aided in tuning hyperparameters and mitigating overfitting. The test set was set aside exclusively for the final evaluation of performance. The training process involved an ADAM optimiser with a non-fixed learning rate. The loss function was categorical cross-entropy, chosen for its suitability in multiclass classification tasks. Model performance was measured using accuracy as the evaluation metric.

To expedite hyperparameter optimisation, the number of training epochs was initially set to 20. The validation accuracy was monitored throughout, and the model with the highest validation accuracy was saved as the best model. After determining the optimal hyperparameters, the model was retrained with 50 epochs to ensure a robust fit to the data. Early stopping was applied with a patience of 3 epochs, terminating training if the validation accuracy stabilised.

In the initial tuning phase, the model achieved a validation accuracy of approximately 64%. After retraining with the best hyperparameters, the model reached a peak validation accuracy of 93%, demonstrating the effectiveness of the optimised architecture and training strategy. The weights of the best model after training have been saved as a `.weights.h5` file, which could be loaded by the `load_weights` method.

3.6 Superiority of Neural Networks

Multi-layer neural networks are highly effective for multiclass classification due to their capacity to model complex, non-linear relationships in data. Each hidden layer in neural networks applies a linear transformation, followed by a non-linear activation function. This combination allows the network to approximate highly non-linear functions. By sequentially projecting input features into higher-dimensional spaces, the layers transform the data into representations where classes become more linearly separable.

4 Comparative Study of Classifiers

To reduce runtime, a smaller dataset of 60,000 combined images with regular labels was created. The training and test images were reshaped into two-dimensional arrays, preserving the first di-

mension. For classification tasks, the `RandomForestClassifier` and `SVC` from SCIKIT-LEARN were imported.

4.1 Random Forest Classifier

To evaluate the performance of Random Forest Classifiers (RFCs), I used the `RandomForestClassifier` module from SCIKIT-LEARN. This investigation focused on two key hyperparameters: **n_estimators**, which determines the number of trees in the forest, and **max_depth**, which limits the depth of each tree. These parameters were tuned to balance computational efficiency and prediction accuracy. The classifier was trained on flattened training images and labels and then tested on flattened test images, with performance evaluated using `accuracy_score`.

The optimal hyperparameters identified were **n_estimators** set to 500 and **max_depth** set to 50, yielding an accuracy of 0.74725. These numbers implied that the number of trees has approximately reached a certain point where it might diminish the result beyond the point, and the maximum depth explored significant complexity without becoming excessively deep.

RFC gradually partitioned the feature space to derive simple rules of classification. Nevertheless, in multiclass classification tasks, these rules often fail to capture the complex boundaries between classes, especially in high-dimensional feature spaces. While RFCs performed quite robustly in high-dimensional features, their strategies became less competitive when some classes exhibited non-linear relationships, limiting their effectiveness in such scenarios.

4.2 Support Vector Machine

For the Support Vector Classifier (SVC), I used the `SVC` module from SCIKIT-LEARN with the radial basis function (RBF) kernel. The classifier was trained on the flattened training images. Prediction accuracy was evaluated using `accuracy_score` by comparing predictions with test labels.

While SVC achieved reasonable accuracy on smaller datasets, its performance deteriorated as the dataset size increased. The runtime for training and prediction grew significantly, and the model struggled to converge efficiently. This computational overhead and reduced accuracy on larger datasets make SVC less ideal for high-dimensional or large-scale tasks without significant parameter tuning.

The primary advantage of SVC lies in maximising the margin between classes to build robust decision boundaries. Yet, in multiclass tasks, decision boundaries are often non-linear, making it challenging to find universal kernel parameters that work effectively for all classes. Although the RBF kernel helps handle non-linear relationships, it may still fail to capture the full complexity of multiclass problems without significant tuning. Additionally, the computational complexity of SVC increases substantially for high-dimensional data, as it requires rigorous feature normalisation and scaling to perform effectively.

The model achieved an accuracy of 0.72, which is relatively low compared to the neural network.

4.3 Comprehension on classifiers

The results demonstrated that RFC was computationally efficient, with shorter training and prediction times compared to SVC. Furthermore, RFC consistently delivered higher accuracy on this dataset, making it a reliable choice for this task. Its ability to handle large datasets efficiently and achieve robust performance highlights its strength for image classification tasks.

The efficiency of RFC arises from its ability to reduce feature space complexity, assisting tree-based models to effectively divide categories. Performance improvements for SVC could be achieved by tuning kernel parameters or using a linear SVC to accelerate training, though this may come at the cost of accuracy in non-linear cases.

5 Experiment: Single Image Input vs Sequential Images Input

The **single-layer perceptron** is a weak linear classifier for its linear separability. While the output in a classic single-layer perceptron is binary (0 or 1), this investigation adapted it for multiclass classification with 19 classes in the Dense layer, corresponding to the combined image dataset. Instead of a step function, a Softmax activation function was used to handle the multiclass output effectively.

This experiment classified both a combined image dataset and sequentially separated image datasets using a single-layer perceptron. For the combined dataset, the model predicted one of 19 possible sums, while for the sequential datasets, it predicted one of 10 possible outcomes per image.

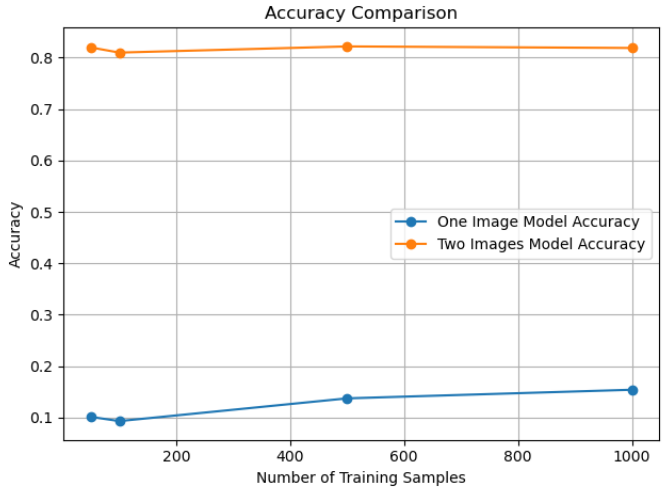

Figure 3: Comparison of Single Combined Image Input with Sequential Images Input

Accuracy results demonstrate that fitting the combined 28×56 images achieves only about 15%, regardless of the number of training samples, significantly lower than fitting two images separately, which yields over 80% accuracy.

The poor performance of single-layer perceptrons in classifying combined images could be attributed to several factors. First, combining two images as a single input introduces a class imbalance in the input, where some classes are overrepresented while others are underrepresented. This imbalance would lead the model to overfit dominant classes with ignorance of less common ones. Second, combining images would significantly increase feature complexity, as the input shape raises from 28×28 to 28×56, resulting in a high-dimensional feature space with sparser data distribution. This increased complexity makes the model harder to find effective decision boundaries. As a simple linear model, the single-layer perceptron cannot capture these complex, high-dimensional relationships effectively.

In contrast, when images are classified separately and sequentially, the correlation between their

features and labels is more direct, making the model less susceptible to bias and improving accuracy remarkably. These results highlight the limitations of single-layer perceptrons in processing complex, high-dimensional and imbalanced datasets.

6 Visualisation of High-Dimensional Features

t-SNE is a non-linear algorithm for dimensionality reduction, designed for high-dimensional data to project into a low-dimensional space (commonly 2D or 3D). It preserves the local structure of the data by retaining pairwise similarities. In high-dimensional space, similarities are modelled using **Gaussian distributions**, while in the low-dimensional projection, **t-distributions** are used to avoid the 'crowding problem' and separate distant clusters better.

While t-SNE cannot perfectly maintain global distances, it excels at identifying local neighbourhoods and forming clusters in low-dimensional space. This makes it ideal for visualising high-dimensional feature spaces, especially noisy or complex data. A list of perplexities had been examined to find an optimised clustering for embedding layers and input data.

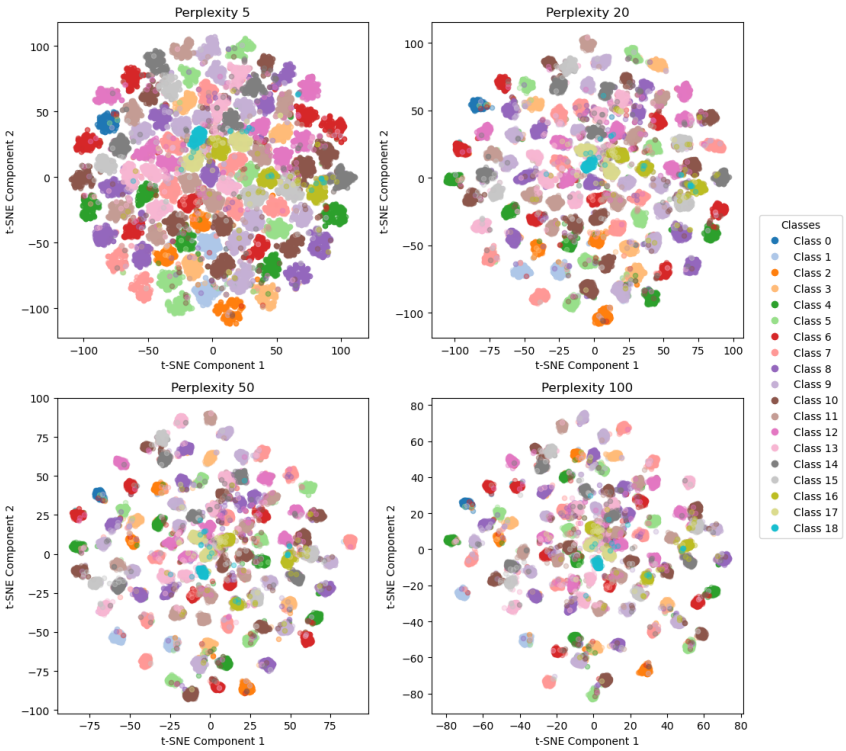

Figure 4: t-SNE Distribution of Embedding Layer

The figure above illustrates that classes 8 to 10 dominate the clustering process due to their consistent input distribution, while classes 0 and 18 each form a single cluster.

At a perplexity of 5, the t-SNE visualisation forms tight and localised clusters with overly fine separations. As perplexity increases from 20 to 100, clusters become more compact and class boundaries clearer, along with smaller clusters overlapping with larger clusters.

Higher perplexity causes sparse classes to converge towards the central area. At lower perplexity, t-SNE emphasises local structures and the similarity of the nearest neighbours for points in high-dimensional space. This would cause widely spreading data points and crude class boundaries. In

contrast, higher perplexity prioritises global structures and smoothens minimal local discrepancies, eliminating local variations and aggregating classes centrally with reduced local detail.

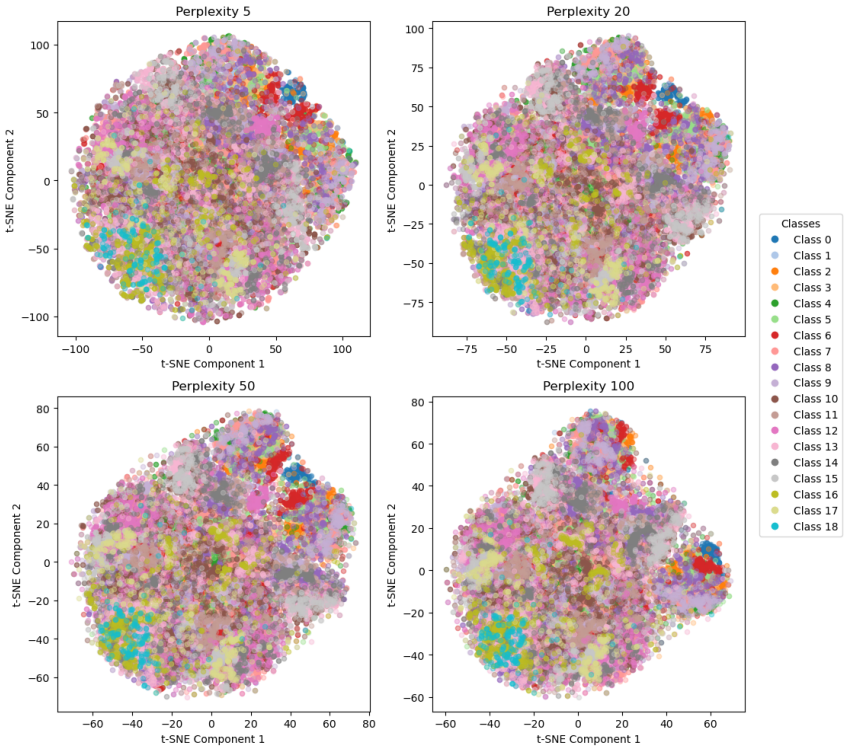

Figure 5: t-SNE Distribution of the Input Dataset

For the t-SNE distributions of the input dataset, it is observable that the distribution is noisy with scattered points for a denser dataset at lower perplexity values. As perplexity rises, clusters start to form visible patterns. At the perplexity of 100, some globally coherent clusters emerge, indicating that t-SNE is capturing broader patterns in higher dimensions. Due to the complexity of the input dataset, class overlap consistently remains dramatic, which means it struggles to fully separate classes when the data contains significant noise or overlapping features.

7 Conclusion

In this coursework, multi-layer fully connected neural networks were proven well-suited for multiclass classification, offering superior performance compared to simpler models. Random Forest Classifiers and Support Vector Classifiers, while functional, are computationally inefficient and show degenerating performance with larger datasets. In contrast, single-layer neural networks, acting as weak linear classifiers, cannot handle the complexity of combined image datasets. The t-SNE visualisation highlights clear feature aggregation in the classified data compared to the disorderly input dataset, validating the existence of higher-dimensional hyperplanes capable of successful classification.

A Appendix

A.1 Use of Auto-Generation Tools

I used ChatGPT to assist with the proofreading process, focusing on reviewing grammar and ensuring sentence clarity.