RECOGNISING IMAGE ORIENTATION OF HANDWRITTEN DIGITS USING A FULLY CONNECTED NEURAL NETWORK

Alex Koh

Victoria University of Wellington, NZ

1. INTRODUCTION

The MNIST dataset is a benchmark in the field of machine learning, widely used for training and testing various image classification algorithms. This paper addresses the problem of classifying images from the MNIST dataset, specifically focusing on recognising the orientation of the images. The task involves preprocessing the dataset to include images rotated by 90 degrees to the left and right, alongside non-rotated images. The objective is to train a neural network to accurately classify these orientations.

The contribution of this paper is the implementation and analysis of a neural network model designed to classify rotated and non-rotated images from the MNIST dataset. A supervised learning approach with a stochastic gradient descent (SGD) optimiser was utilised to explicitly show the iterations and learning process. The model's performance is evaluated through training and validation accuracy metrics over multiple epochs.

In the subsequent sections, we first describe the theoretical background related to image classification using neural networks, including relevant equations and concepts. Following this, the experimental setup is detailed, outlining the data preprocessing steps, network architecture, training procedure, and evaluation metrics. Finally, we present the results and discuss the model's performance, concluding with potential future improvements and applications.

2. THEORY

In this section, we discuss the theoretical background relevant to the task of classifying rotated images from the MNIST dataset using neural networks.

For a classification problem, a naive objective function would be a binary indicator of whether the prediction is correct or incorrect. However, such a function is not differentiable, making it unsuitable for optimisation using gradient-based methods. Instead, we use a probabilistic formulation that provides a continuous measure of confidence in the prediction, leading to a differentiable objective function.

In a probabilistic setting, our model estimates the probability distribution of the classes given the input. For a multi-class classification problem, this can be written as $q_\theta(y_i = 1|x)$ with $y_i \in 0,1$, where y is the class label and x is the input. The objective is to maximise the likelihood of the correct class, which is equivalent to minimising the cross-entropy loss between the predicted probabilities and the true class labels.

The cross-entropy loss for a multi-class classification problem is given by:

$$L(\theta) = -\sum_i y_i \, log q_\theta(y_i|x), \quad (1)$$

where y_i is the true label, and $q_\theta(y_i|x)$ is the predicted probability of class i given the input x.

Multi-Class Classification and Softmax

In multi-class classification, each class gets an output unit (neuron) in the neural network. The network's output is a vector of estimated probabilities for each class. To ensure these probabilities sum to one and are non-negative, we use the softmax function:

$$\mathrm{q}(y_i = 1|x) = \frac{exp(f_i(x))}{\sum_{j \in C} exp(f_j(x))}, \quad (1)$$

where:

- $\mathrm{q}(y_i = 1|x)$ represents the probability that the input x belongs to class i
- $f_i(x)$ is the output of the neural network for class i given input x
- $f_i(x)$ is the output of the neural network for class j given input x
- C is the set of all possible classes

3. EXPERIMENTS

3.1 Experimental Setup

The neural network model was configured with an input layer of 784 neurons to accommodate the flattened 28x28 pixel MNIST images. This was followed by two hidden layers with 128 and 64 neurons, respectively, both employing the ReLU activation function. The output layer consisted of 3 neurons

corresponding to the three classes: rotated left, rotated right, and not rotated.

The software versions used for this experiment were JAX 0.2.12 and Python 3.8.10, running in a Google Colab environment. TensorFlow 2.4.1 was utilised for data loading.

3.2 Data Description

The MNIST dataset, containing handwritten digit images, was used for this experiment. The training set contains 60,000 images and 60,000 labels, and the test set contains 10,000 images and 10,000 labels. Each image was of size 28x28 pixels with grayscale values ranging from 0 to 255, and each label contains values ranging from 0 to 9, representing the digit classes.

To prepare the data, the images were normalised to the range [0, 1] by dividing the pixel values by 255. 1/3 of the images were rotated 90 degrees clockwise while 1/3 of the images were rotated 90 degrees counterclockwise. The remaining 1/3 images were left unrotated. The rotated training data was then split into 85% training and 15% validation sets, ensuring that the required distribution was maintained.

The 28x28 images were flattened into one-dimensional vectors with 784 dimensions to serve as input for the neural network. The images were also converted to floating-point numbers for efficient computation during gradient descent and backpropagation.

3.3 Training Procedure

The neural network was trained using stochastic gradient descent (SGD) with the following parameters: epochs (10), batch size (32), and learning rate (0.01). The training process began with initialising the network weights and biases using He initialisation to ensure proper scaling.

During each epoch, the input data was passed through the network, with ReLU activation applied to the hidden layers. The cross-entropy loss was computed using the softmax function. The gradients of the loss with respect to the network parameters were calculated using JAX's grad function, and the network parameters were updated using a manually implemented SGD update rule. Training loss and accuracy, as well as validation accuracy, were recorded at the end of each epoch.

3.4 Evaluation Metrics

The model's performance was evaluated using accuracy metrics on the training, validation, and test datasets. The accuracy was computed as the proportion of correctly classified images out of the total number of images. The final test accuracy achieved was approximately 98.91%. The training and validation accuracy over the epochs is depicted in Figure 1 on the Appendix. The steady improvement and convergence of both training and validation accuracies indicate effective learning and generalisation by the neural network.

4. CONCLUSION

Our results demonstrate that the fully connected neural network effectively classifies the orientation of MNIST images, achieving a high level of accuracy. The network was trained to recognise three distinct orientations: non-rotated, rotated 90 degrees to the left, and rotated 90 degrees to the right. The implementation of stochastic gradient descent (SGD) with explicit gradient calculations and manual updates facilitated a transparent and efficient learning process.

We also experimented with different depths of the neural network by varying the number of hidden layers. Our results indicate that a deeper network with more hidden layers did not significantly outperform the shallower configuration used in the final model. This suggests that for the task of classifying rotated MNIST images, the additional complexity introduced by deeper networks may not be necessary. The two hidden layers with 128 and 64 neurons, respectively, provided a good balance between model complexity and performance.

The final test accuracy achieved by our model was approximately 98.91%, underscoring the robustness and generalisation capability of the neural network. The steady convergence of training and validation accuracies, as shown in the performance curves, indicates that the network learned the task well without significant overfitting.

Future work could explore several avenues to enhance performance further. Incorporating more complex network architectures, such as convolutional neural networks (CNNs), could leverage spatial information more effectively. Additionally, the application of regularisation techniques, such as dropout or L2 regularisation, could help in preventing overfitting. Experimenting with different activation functions and optimisation algorithms might also yield improvements in accuracy and training efficiency.

5. STATEMENT OF TOOLS AND AUTHORSHIP

This project was conducted using JAX in Google Colab. The code can be accessed via this link. I confirm that this work is my own and all sources have been appropriately cited.

6. REFERENCES

[1] W. B. Kleijn. (2024). Basic Deep Learning Regressor [Lecture notes]. Victoria University of Wellington. https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/lect4ObjF.pdf

7. APPENDIX

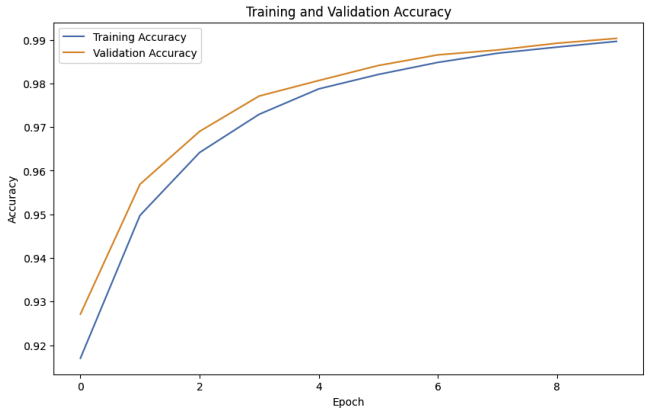

Fig 1: Neural network's training and validation accuracy across epochs.