Report on MNIST Classification with Perceptrons, CNNs, and a Classical Model

1. Introduction

In this assignment I worked with the MNIST handwritten digit dataset and compared several models. I trained different multi-layer perceptron (MLP) models, two convolutional neural networks (CNNs), and I also compared these results to the best classical model from Assignment 1, which was a support vector machine (SVM) with RBF kernel. The main idea was to see how different model designs and training settings change the test accuracy and training time, while keeping the dataset and preprocessing the same.

2. Dataset and preprocessing

The dataset used in this assignment is MNIST, which contains images of handwritten digits from zero to nine. There are 60,000 training images and 10,000 test images. Each image is a 28 x 28 grayscale picture, and each label is a digit between 0 and 9.

I loaded the data from a local mnist.npz file into x_train, y_train, x_test and y_test. After loading, I converted the image pixel values from integers to floating point numbers and normalized them by dividing by 255 so that all values lie between zero and one.

Because Keras expects an explicit channel dimension, I reshaped the images from shape (28, 28) to (28, 28, 1). The labels were kept as integer class indices. After these steps, I printed the shapes to confirm that the number of samples and labels in the training and test sets matched.

3. Perceptron models (Part A)

In Part A I focused on multi-layer perceptrons and used two basic model designs: one with three hidden layers and one with five hidden layers. For each design I changed the activation functions and optimizers and then compared the results. All perceptron models used sparse categorical cross entropy as the loss function, accuracy as the metric, and were trained for five epochs. A shared training function compiled the model, fitted it, measured the running time and then evaluated the accuracy on the test set.

3.1 Model designs

The three-layer MLP (build_mlp_3 in the notebook) first flattened the 28 by 28 image into a single vector. Then it used a hidden layer with 256 units, another hidden layer with 128 units, and finally an output layer with 10 units and softmax activation for the ten classes.

The five-layer MLP (build_mlp_5) also flattened the input but used more and larger hidden layers. It had hidden layers with 512, 256, 128 and 64 units, followed by the final softmax layer with 10 units.

3.2 A.1 - Three-layer MLP with ReLU and Adam

In experiment A.1 I trained the three-layer MLP using the ReLU activation function for the hidden layers and the Adam optimizer. The model was trained for five epochs. The test accuracy for this model was about 0.9788, which is around 97.88%, and the training time was about 3.07 seconds.

This shows that even a relatively small MLP with ReLU and Adam can reach high accuracy on MNIST with short training time.

3.3 A.2 - Three-layer MLP with Sigmoid and Adam

In A.2 I kept the same three-layer model but changed the activation function in the hidden layers from ReLU to Sigmoid. The optimizer remained Adam, and the model was again trained for five epochs.

The test accuracy for this model was about 0.9637, or 96.37%, and the running time was about 3.37 seconds. Compared to A.1, the accuracy dropped by more than one percent and the training time increased slightly. This suggests that, for this task and model design, ReLU works better than Sigmoid.

3.4 A.3 - Three-layer MLP with ReLU and SGD

In A.3 I returned to ReLU activations in the hidden layers but changed the optimizer from Adam to SGD. The rest of the setup stayed the same, including the five training epochs.

This model reached a test accuracy of about 0.9197, or 91.97%, and the training time was about 2.46 seconds. The model was a little faster than the ReLU plus Adam model from A.1, but the accuracy was much lower, with a drop of around six percentage points. This shows that the choice of optimizer is very important, and in this case Adam is clearly better than plain SGD.

3.5 A.4 - Five-layer MLP with ReLU and Adam

In A.4 I increased the depth of the network by using the five-layer MLP with hidden layers of sizes 512, 256, 128 and 64 units. All hidden layers used ReLU activation and the model again used the Adam optimizer with five training epochs.

This deeper MLP achieved a test accuracy of about 0.9795, or 97.95%, with a running time of about 5.85 seconds. It is slightly more accurate than the three-layer ReLU plus Adam model from A.1, but almost twice as slow to train. This shows that increasing depth helps a little but also increases computation time.

3.6 Summary of Part A

To summarize the perceptron models: the three-layer MLP with ReLU and Adam (A.1) gave about 97.88% accuracy in about 3.07 seconds. The same model with Sigmoid in the hidden layers (A.2) dropped to about 96.37 percent accuracy and took about 3.37 seconds. When I used ReLU with SGD (A.3), the accuracy dropped further to about 91.97 percent, while the time was about 2.46 seconds. The five-layer MLP with ReLU and Adam (A.4) reached about 97.95 percent accuracy with around 5.85 seconds of training time.

From these results I learned three main things. First, **ReLU is a better activation than Sigmoid** for this setup. Second, **Adam is a much better optimizer than SGD** here. Third, making the network deeper gives a small improvement in accuracy but also makes training slower.

4. Convolutional neural networks (Part B)

In Part B I used convolutional neural networks, which are more suitable for image data because they use convolution layers and pooling to capture local patterns. I trained two CNN models: a simple four-layer CNN and a LeNet-5 style CNN. Both were trained with the Adam optimizer and the same loss and metric as in Part A, again for five epochs, using a helper function to measure accuracy and running time.

4.1 B.1 - Simple four-layer CNN with Adam

The first CNN was a simple model with two convolution and pooling blocks followed by dense layers. The design was as follows. The input was an image of shape 28 x 28 x 1. The first convolution layer had 32 filters with kernel size 3 and used ReLU activation, followed by a max pooling layer. The second convolution layer had 64 filters with kernel size 3 and ReLU activation, followed by another max pooling layer. After that, there was a flatten layer, a dense layer with 64 units and ReLU activation, and finally a dense layer with 10 units and softmax activation for the digit classes.

Trained with Adam for five epochs, this model achieved a test accuracy of about 0.9872, or 98.72%. The running time was about 21.56 seconds. This accuracy is higher than any of the MLPs from Part A, but the training is clearly slower.

4.2 B.2 - LeNet-5 style CNN with Adam

The second CNN was based on the classic LeNet-5 structure. This model also took images of shape 28 x 28 x 1 as input, then applied a convolution layer, an average pooling layer, another convolution layer, another average pooling layer, then a flatten layer, followed by dense layers with 120 and 84 units, and finally the output dense layer with 10 units and softmax activation.

With the Adam optimizer and five training epochs, this LeNet-5 style CNN achieved a test accuracy of about 0.9858, or 98.58%, and took about 13.69 seconds to train. Its accuracy is slightly lower than the simple four-layer CNN, but it is faster.

4.3 Summary of Part B

The simple four-layer CNN reached about 98.72% accuracy with a training time of about 21.56 seconds. The LeNet-5 model reached about 98.58 percent accuracy with a training time of about 13.69 seconds.

Both CNNs performed better than the perceptron models in terms of accuracy, which matches the expectation that CNNs are better at handling image data. The simple CNN is the most accurate model overall, while LeNet-5 is slightly less accurate but faster.

5. Comparison with the classical model from Assignment 1

In Assignment 1, the best classical model for the MNIST dataset was a support vector machine with RBF kernel. That model achieved a test accuracy of about 0.9792, or 97.92%.

When I compare this SVM to the deep learning models, I see the following. The five-layer MLP with ReLU and Adam from Part A achieved about 97.95 percent accuracy in about 5.85 seconds, which is very close to the SVM's accuracy. The simple four-layer CNN from Part B reached about 98.72 percent accuracy in about 21.56 seconds, and the LeNet-5 CNN reached about 98.58 percent accuracy in about 13.69 seconds.

So the best CNNs clearly outperform the SVM and the MLPs in terms of accuracy. However, they do require more training time and are more complex.

6. Conclusions

This assignment helped me understand how different models behave on the same dataset when the preprocessing is fixed. Using simple normalization and reshaping of the MNIST images was enough to get strong results across all models.

For the perceptron models, the experiments showed that the choice of activation function and optimizer is crucial. ReLU with Adam consistently worked better than Sigmoid or SGD in both accuracy and training behavior. Adding more layers improved accuracy slightly but also increased the training time.

The convolutional neural networks performed the best overall. Both the simple four-layer CNN and the LeNet-5 model reached accuracies above 98.5 percent, which is higher than the perceptron models and the SVM from Assignment 1. At the same time, they took longer to train, so there is a clear balance between accuracy and computation time.