DAT405 Assignment 7 – Group 1

Filip Cederqvist - (15 hrs)
Kevin To - (15 hrs)

May 17, 2023

Problem 1

Preprocessing. In the notebook, the data is downloaded from an external server and imported into the notebook environment using the mnist.load_data() function call.

1.1. Explain the data pre-processing highlighted in the notebook

```
batch_size = 128
num_classes = 10
epochs = 10

img_rows, img_cols = 28, 28

(x_train, lbl_train), (x_test, lbl_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255

y_train = keras.utils.np_utils.to_categorical(lbl_train, num_classes)
y_test = keras.utils.np_utils.to_categorical(lbl_test, num_classes)
```

Listing 1: The Preprossesing Code.

As one can see from the code, it first set up the batch size, number of classes which are the natural numbers, and the number of epochs. The code also defines the number of rows and columns of the images. These are 28 since the images in the data set are 28x28 pixels.

Next, the code splits the MNIST datasets into training and test data, and reshapes each image in the training and test data to fit the neural network.

Then, the code normalizes the images as it divides it by 255. This is to make the pixel values to be between 0 and 1.

Problem 2

2.1. How many layers does the network in the notebook have? How many neurons does each layer have? What activation functions and why are these appropriate for this application? What is the total number of parameters for the network? Why do the input and output layers have the dimensions they have?

```
## Define model ##
model = Sequential()

model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Listing 2: The Neural Network

The network has four layers which are: Input, two hidden (dense) layers, and output. The input layer has $28 * 28 = 784$ neurons, as this is the number of pixel of one image, while the two hidden layer have 64 neurons each. The output layer has 10 neurons which represent all natural numbers.

The activation function used in the hidden layers is the Rectified Linear Unit activation function (ReLU), see figure 1. This is a good activation function to uses here because the values cannot be negative. As the model aims to predict one of the natural numbers, this function suits the model well.

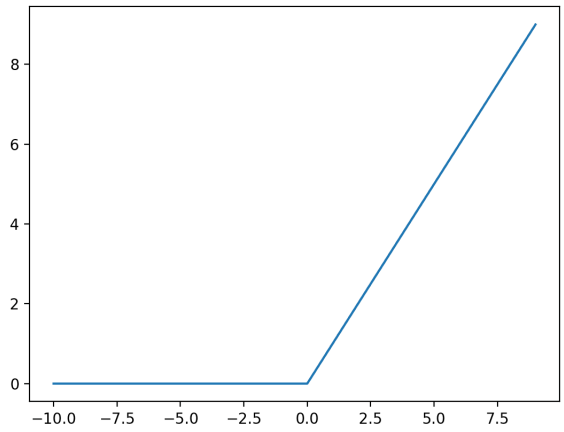

Figure 1: The Graph of the ReLU Activation Function.

The activation function used in the output layer is Softmax function. This is a good function to have in the last layer, as it assign each natural number a probability in decimal form. These probabilities must add up to 1.0.

The parameters of a neural network is the sum of the weight and biases. In this model we have 784 inputs, 64 nodes in 2 hidden layers and 10 nodes in the output layer. This gives the weight: $784 * 64 + 64 * 64 + 64 * 10 = 54912$. The amount of bias are: $64 + 64 + 10 = 138$. As such, there are $54912 + 138 =$ **55 050** parameters.

2.2 What loss function is used to train the network? What is the functional form (a mathematical expression) of the loss function? and how should we interpret it? Why is it appropriate for the problem at hand?

The loss function used to train this model is the Cross-entropy loss function. It has the form

$$CE = -\sum_{n=i}^{C} t_i log(f(s)_i),$$

where C is the number of classes (natural numbers), t_i is the target vector, $f(s)_i$ is the predicted value after applying the Softmax function. Since there will only be one target value, the summation can be ommited, resulting in the following expression:

$$CE = -log(f(s)).$$

The loss function is used in order to determine new weights and biases in the network. The aim is to reduce the loss as much as possible. The Cross-entropy loss function fits our network well, as it is logarithmic. This means if the model makes an unlikely prediction, the loss will be huge. Similarly, if the model makes a good prediction, the loss will not be that great. This can be illustrated with the graph of the function. This quickly makes it so that the network improves fast.

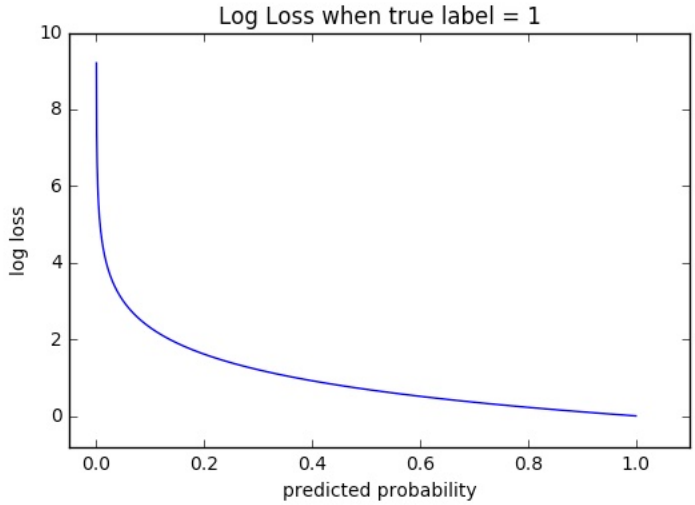

Figure 2: Graph for the Cross-entropy Loss Function.

2.3 Train the network for 10 epochs and plot the training and validation accuracy for each epoch.

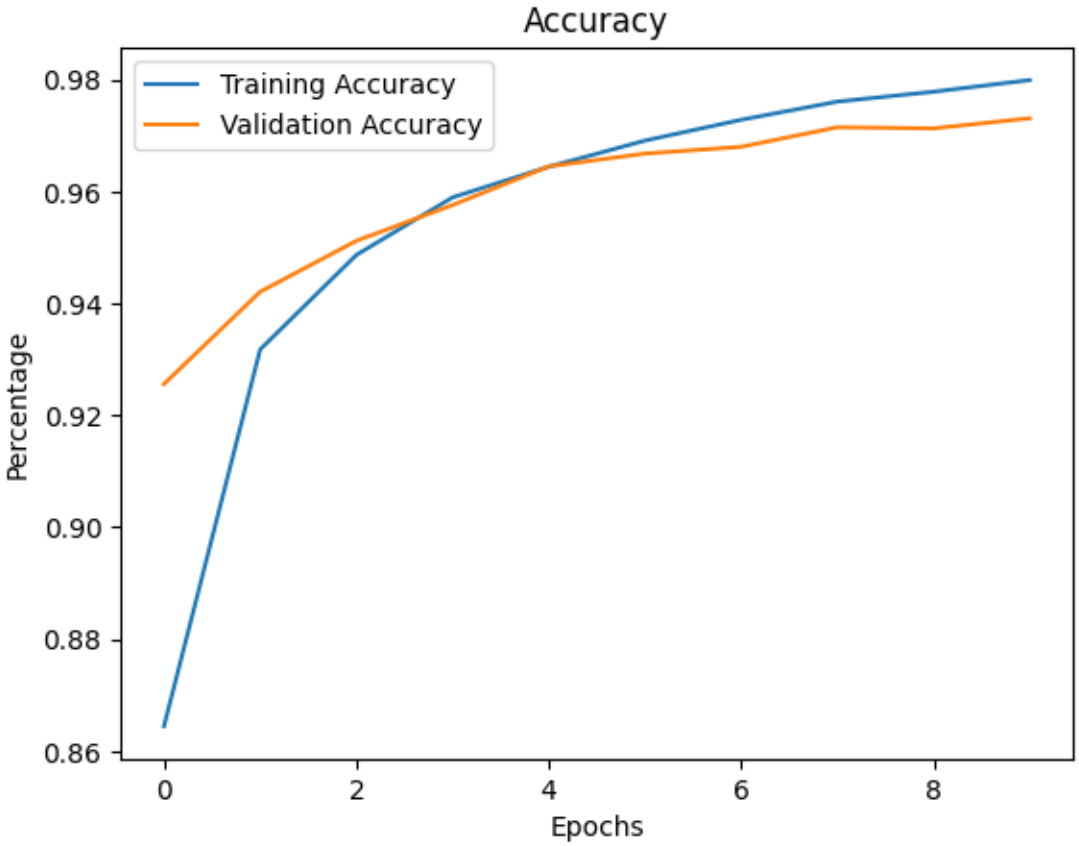

Figure 3: The Training and Validation Accuracy for each Epoch

2.4 Update the model to implement a three-layer neural network where the hidden layers have 500 and 300 hidden units respectively. Train for 40 epochs. What is the best validation accuracy you can achieve? – Geoff Hinton (a co-pioneer of Deep learning) claimed this network could reach a validation accuracy of 0.9847 using weight decay.

We choose decay values of 0.000001, 0.00001, 0.0001, 0.0005, 0.001 and trained three models with these. The results can be visible in the table and graphs below:

Decays	Final Test Accuracy	Mean Test Accuracy	Std Test Accuracy
0.000001	0.9824	0.9773	0.0092021
0.00001	0.9828	0.9782	0.0087519
0.0001	0.9816	0.9772	0.0089336
0.0005	0.9833	0.9773	0.0092406
0.001	0.9803	0.9752	0.0093075

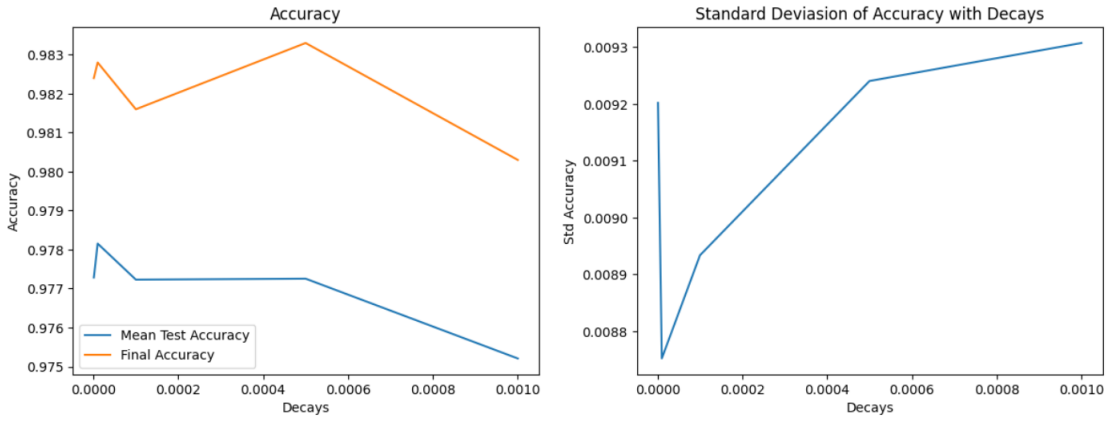

Figure 4: Plot over the change in accuracy with different regularization factors.

From the result, we can see that we didn't reach quite as high of an accuracy as Geoff Hinton managed since his best accuracy was 0.9847 compared to ours of 0.9833. However, the accuracy was not far off.

The reasons why we didn't reach as high of an accuracy could be because of the epochs and learning rate. Though, increasing the number of epochs is not likely to increase the accuracy as this converges quickly in the beginning.

The learning rate, on the other hand, could play a factor. If we lowered this, the model would become more hesitant toward change and thus maybe give greater results.

Lastly, we don't know what regularization factors Hilton used in his example. Our graph shows that the accuracy changes up and down depending on the regularization factor. If we had time to test multiple factors, we might have one that gives a better result.

Problem 3

3.1 Design a model that makes use of at least one convolutional layer – how performant a model can you get?

By incorporating two convolutional layers with 32 filters in the first layer and 64 filters in the second layer, both utilizing a 7x7 kernel size, Keeping the first hidden layer we obtained the following outcome (see: figure 5).

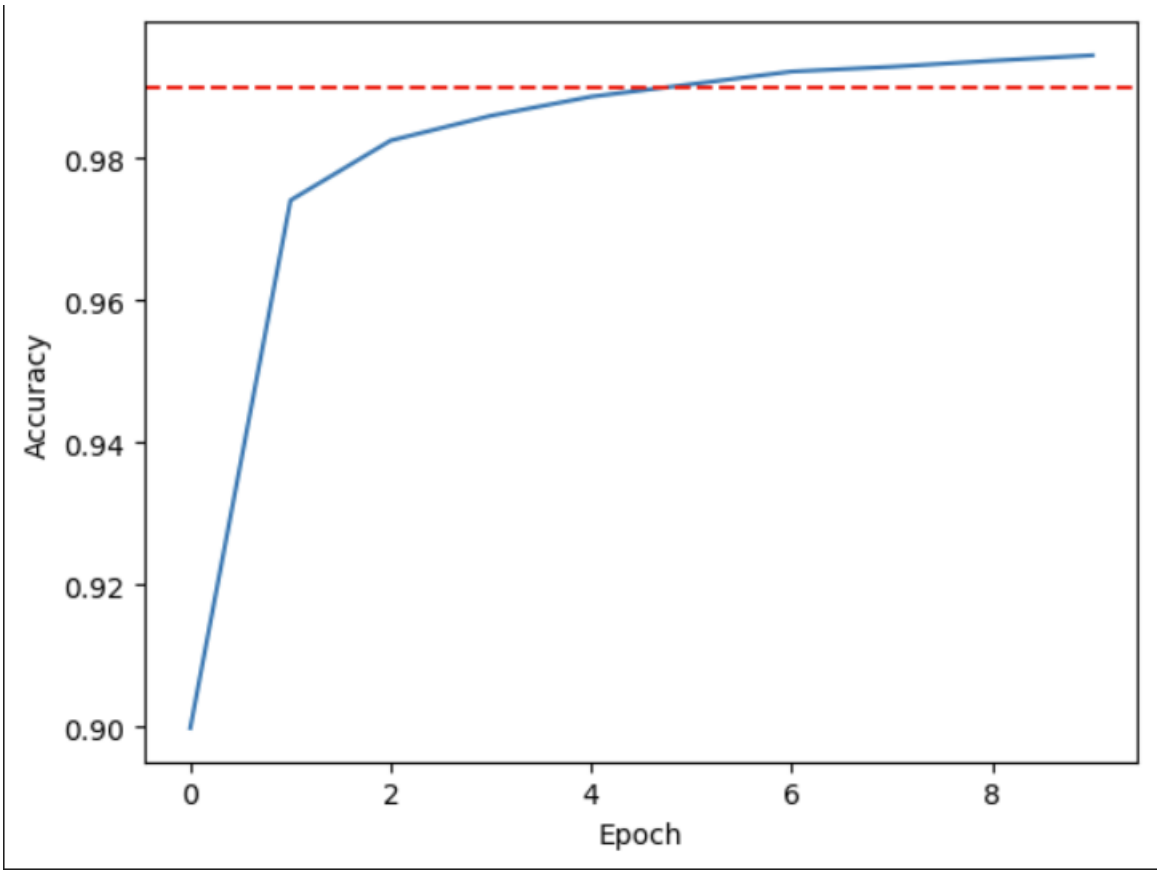

Figure 5: Results from first try with two convolutional layers

We achieved further enhancement in the model's performance by introducing a third convolutional layer with 128 filters (see . However, it should be noted that this addition noticeably increased the runtime of the model. Due to the substantial increase in computational time, we decided not to explore any additional improvements beyond this point.

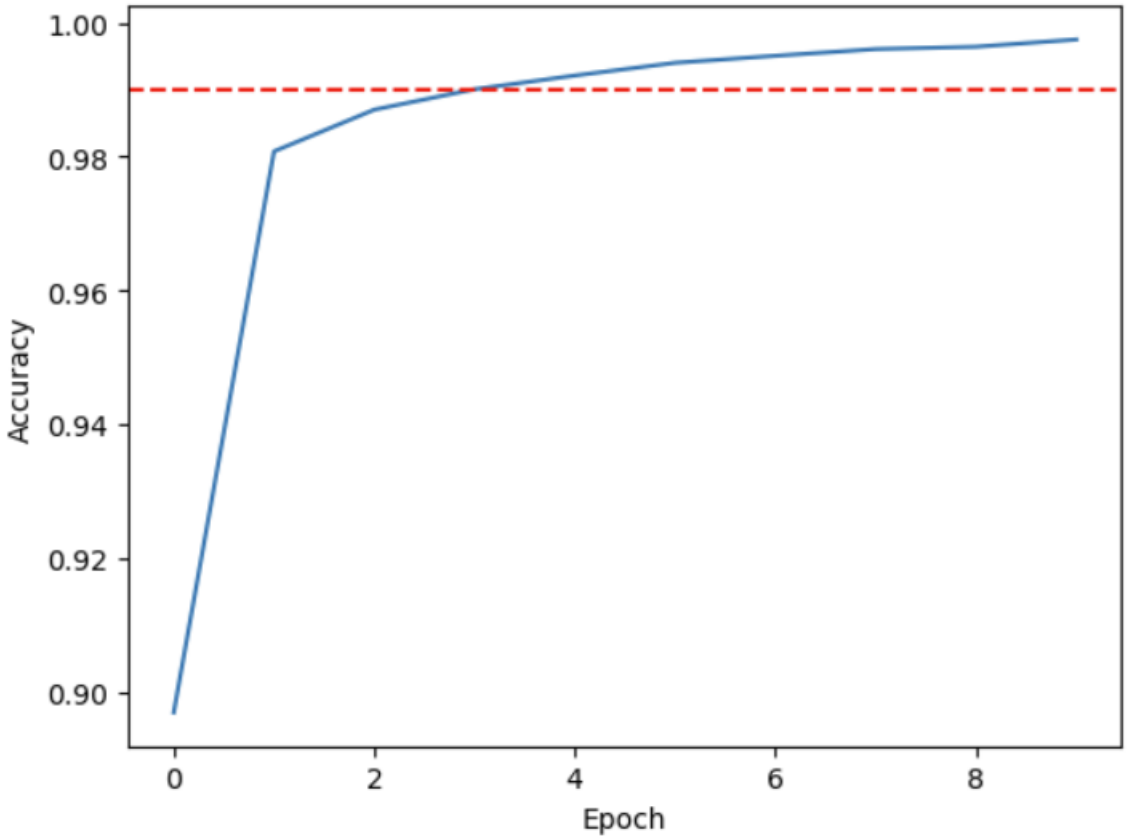

Figure 6: Results from second try with three convolutional layers

3.2 Discuss the differences and potential benefits of using convolutional layers over fully connected ones for the application?

Using convolutional layers instead of fully connected layers has significant differences. Firstly, with fully connected layers, the number of parameters increases a lot as the image size gets larger or when we add more layers, like in our application with hidden layers having more neurons. On the other hand, with convolutional layers, the parameters are shared, which reduces the size and improves computational efficiency.

Another difference occurs when the image is slightly shifted in a random direction. In a fully connected network, this shift can make it unable to detect and classify the image accurately. However, convolutional neural networks (CNNs) excel in this aspect. CNNs identify image features by looking at neighboring pixels, instead of treating the whole image as a single entity. This allows CNNs to classify images even when there are shifts. The only thing that can affect a CNN's classification is changes to the pixels themselves.

In our specific application, these features offer advantages in terms of computational efficiency and overall accuracy.