Assignment 2 Report

Li Xiaowei
matric: A0142325R
NUSNET ID: E0006340

Part 1 Description of Problem

The MNIST database is a large database of handwritten digits commonly used for digit recognition. The database contains 60,000 training images and 10,000 testing images. The images are grayscale with dimension 28 × 28. We apply 90%-10% train-validation-split.

Part 2 Design of the Neural Network Architecture

The best neural network architecture is shown in Figure 1 with key processes and implementations described below:

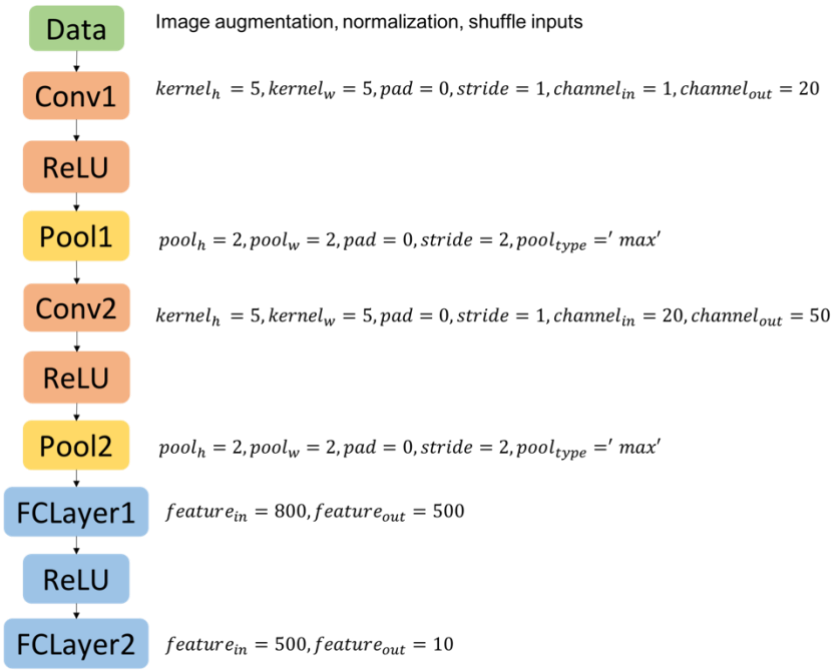

Figure 1 design architecture

2.1 Data Preprocessing

We first apply **image augmentation** to the 60,000 training images to produce 120,000 new images. We randomly apply blotches, rotation, translation or add random noise to the input images (see *codes/dataAugmentation.py*). We append the augmented 120,000 images to the original training images, apply **normalization** to both training and testing images and apply train-validation split to the total 180,000 training images. The training images *X_train* has dimension (162000, 1, 28, 28) and validation images has dimension *X_val* of (18000, 1, 28, 28). We feed *X_train* to Conv1 layer.

2.2 Forward and Backward of Convolutional Layer

2.2.1 Forward Process

The convolutional layer performs the convolution process between the filters and receptive field on the image. Input to Conv1 has dimension $(n, c_{in}, x_{h,}x_w)$ where $n, c_{in}, x_{h,}x_w$ denote batch, number of in_channel, input height, input width respectively. Output of Conv1 has dimension $(n, c_{out}, o_{h,}o_w)$ where $n, c_{out}, o_{h,}o_w$ denote out_channel, output height, output width respectively. We can calculate o_h *and* o_w as: $o\_h = floor\left(\frac{l_h+2\times p_h-k_h}{s_h}\right)+1$ and $o\_h = floor\left(\frac{l_h+2\times p_h-k_h}{s_h}\right)+1$. We apply img2col operation to achieve faster computation, input images are converted into 2D matrix of shape $(c_{in}\times k_h\times k_w, n\times o_h\times o_w)$ and kernels are converted into 2D matrix of shape $(c_{out}, c_{in}\times k_h\times k_w)$. We apply matrix multiplication to converted input images and kernels to obtain output of shape $(c_{out}, n\times o_h\times o_w)$ and add in the bias. We reshape the output to dimension of $(n, c_{out}, o_{h,}o_w)$.

2.2.2 Backward Process

For backward process, we have $\frac{\partial L}{\partial Y}$ of shape $(n, c_{out}, x_{h,}x_w)$ and inputs of shape $(n, c_{in}, x_{h,}x_w)$. We convert $\frac{\partial L}{\partial Y}$ to $\frac{\partial L}{\partial Y'}$ of shape ($c_{out}, n\times x_h\times x_w$). We first apply img2col operation to obtain 2D input X of shape $(c_{in}\times k_h\times k_w, n\times o_h\times o_w)$ and 2D kernels W of shape $(c_{out}, c_{in}\times k_h\times k_w)$. By Y' = WX + b, $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y'}\times X.T$, $\frac{\partial L}{\partial X} = W.T\times\frac{\partial L}{\partial Y'}$, $\frac{\partial L}{\partial b} = sum\ of\ original\frac{\partial L}{\partial Y}\ along\ axis(0,2,3)$.

2.3 Forward and Backward of ReLU Layer

Rectified Linear Unit(ReLU) is a common activation function of the form $f(x) = \max(0, x)$. It maintains a good balance of preventing gradient vanishing and introducing non-linearity to the layers. For the forward pass, we maintain a 2D mask which has 1 if mask[i][j] >= 0 and otherwise 0. We then do element-wise multiplication between

the mask and vectorized input X. During backward pass, we do element-wise multiplication between the mask and the incoming gradient.

2.4 Forward and Backward of Max and Average Pooling Layer

2.4.1 Forward Pass of Max and Average Pooling

For the forward pass, we apply im2col to convert input to 2D matrix X of shape $(pool_h \times pool_w, c_{in} \times n \times o_h \times o_w)$. For max pooling, we obtain the argmax for each column of X and keep a copy of argmax index matrix (1 for max, 0 otherwise). For average pooling, we obtain the average for each column of X. We then convert X to output shape of (n, c_{in}, o_h, o_w).

2.4.3 Backward Pass of Max and Average Pooling

For Backward pass, we convert $\frac{\partial L}{\partial Y}$ to $\frac{\partial L}{\partial Y'}$ of shape $(pool_h \times pool_w, c_{in} \times n \times o_h \times o_w)$. For max pooling, we perform element-wise multiplication between $\frac{\partial L}{\partial Y'}$ and argmax indices matrix (1 for max, 0 otherwise). For average pooling, we divide $\frac{\partial L}{\partial Y'}$ by $(pool_h \times pool_w)$.

2.5 Forward and Backward of Dropout Layer

Dropout is a technique to prevent overfitting by randomly setting some neurons to zero. During training, we sample a binary mask according to a threshold p with 1 means keep the neuron and 0 otherwise. After element-wise multiplication between mask and input, we scale input by a factor $\frac{1}{1-p}$. During backward propagation, we perform element-wise multiplication between in_grads and mask and scale factor. During testing, we do nothing.

2.6 Forward and Backward of Fully Connected Layer

FC layer perform Y = W.T.dot(X) + b operation where X has shape (batch, in_feature) and W has dimension (batch, out_feature). During backward pass, $\frac{\partial L}{\partial W} = X.T \times \frac{\partial L}{\partial Y}$, $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \times W.T$, $\frac{\partial L}{\partial b} = sum\ of\ original \frac{\partial L}{\partial Y}\ along\ axis(0)$.

2.7 Forward and Backward of Softmax Layer

Softmax loss has equation $L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$ where s = WX, e^{sy_i} is the score for correct class and e^{s_j} is the score for class j. Softmax is commonly used for multiclass classification task where the final output class for a particular instance is given by the highest probability ($p_i = \frac{e^{sy_i}}{\sum_j e^{s_j}}$) among all classes.

Part 3 Report of Accuracy and Loss

I run the entire dataset for 2 epochs at learning rate of 0.001 and achieved the best testing accuracy of **0.98480** (see *codes/CS5242_Assignment_1.ipynb*). Validation accuracy is always lower than test accuracy probably due to randomized image augmentation which produces many unseen images in both the training and validation set.

```
Test accuracy=0.98400, loss=0.05694
Validation accuracy: 0.83383, loss: 0.57922
Iteration 4500: accuracy=0.83333, loss=0.36134, regularization loss= 0.27350797658175463
Validation accuracy: 0.83417, loss: 0.57837
Iteration 4600: accuracy=1.00000, loss=0.06101, regularization loss= 0.2735570188746937
Validation accuracy: 0.83356, loss: 0.57880
Iteration 4700: accuracy=0.93333, loss=0.09931, regularization loss= 0.27357271135438493
Test accuracy=0.98380, loss=0.05707
Validation accuracy: 0.83361, loss: 0.57861
Iteration 4800: accuracy=0.96667, loss=0.08588, regularization loss= 0.2735943321650813
Validation accuracy: 0.83422, loss: 0.57835
Iteration 4900: accuracy=0.96667, loss=0.10163, regularization loss= 0.2736193690142717
Validation accuracy: 0.83350, loss: 0.57935
Iteration 5000: accuracy=0.90000, loss=0.42141, regularization loss= 0.2736421263482395
Test accuracy=0.98390, loss=0.05691
Validation accuracy: 0.83372, loss: 0.57914
Iteration 5100: accuracy=0.86667, loss=0.42931, regularization loss= 0.27366129108457793
Validation accuracy: 0.83439, loss: 0.57798
Iteration 5200: accuracy=0.96667, loss=0.21212, regularization loss= 0.27368737923296804
```

Figure 2 Screenshot of training, validation and testing accuracy and loss over iterations

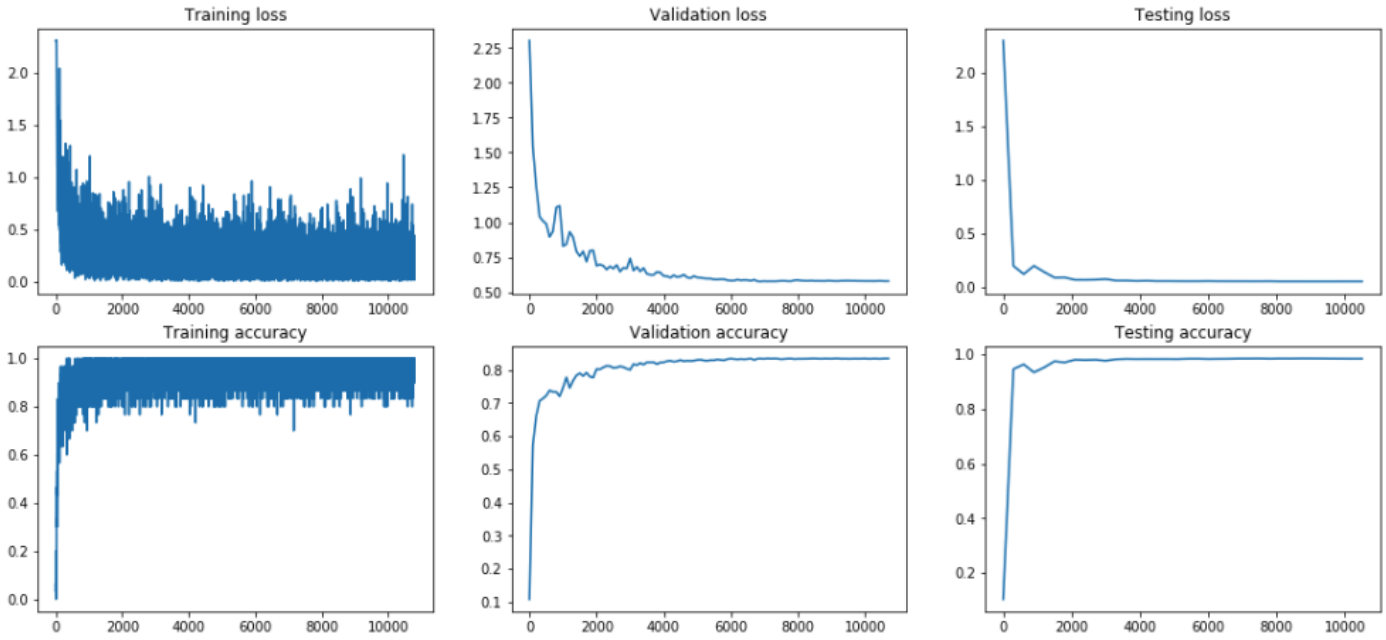

Figure 3 Training, validation and testing loss and accuracy over iterations

Part 4 Additional Insights about Parameter Tuning

4.1 Importance of data augmentation

Having applied blotching, rotation, translation and adding random noise to input images to generate more datasets, the network can better learn the important features and less susceptible to variations in input images. After augmenting 12,000 new images to the original 6,000 images, the accuracy improved by 0.5%. However, since rotation operation is random, and digits like 9 will become 6 after rotating 180 degrees, but the correct label is still 9. This added noise to the training process. I believe that the overall result would still be better since the likelihood of rotating 180 degrees for digit 9 is still relatively low.

4.2 Importance of normalizing input

Normalization keeps input data to have zero mean and unit variance at each dimension. This can prevent gradient exploding by keeping the value at each iteration to be relatively small after it is multiplied by input X. Moreover, a unit variance ensures the update of gradient in the steepest direction during the training process. Due to time constraint, I did not implement batch normalization layer, I believe that having a batch normalization layer before pooling layer will improve the result significantly.

4.3 Effect of learning rate on stability of convergence

Too small a learning rate can make convergence slow, while too big a learning rate can make convergence slower, unstable and easily overshot. A comparison of learning rate of 0.001 and 0.1 is demonstrated below. We chose learning rate of 0.001 to be the value used in final neural network.

4.4 Effect of optimizers on rate of convergence

I trained neural network with optimizers including SGD with momentum, Adam, Adagrad and RMSprop. Adam generates the fastest rate of convergence.

4.5 Effect of Dropout on preventing overfitting

The effect of dropout in regularizing the neural network does not seem significant, that is probably because of relatively small number of hidden layers. Together with L2 regularization, the network did not overfit the training set as observed by training accuracy and testing accuracy being very close. The final neural network architecture does not include dropout layer.

4.6 Importance of Shuffling Input Datasets

Since the dataset is split to training and validation set according to a 9-1 ratio, it is important to shuffle the dataset before splitting so as to prevent any fixed pattern in the imported dataset. For example, most of the small numbers (e.g. 1-5) appear early, and large numbers appear at the back. Shuffling the dataset can prevent uneven distribution data in the splitted training dataset. After applying random shuffling, the accuracy improved by 0.3%.