Project-1 Report of "Neural Network and Deep Learning"

Jialun Shen
16307110030

April 30, 2021

1 Neural Network

1.1 Questions

1. Change the network structure: the vector *nHidden* specifies the number of hidden units in each layer.

   (code in `q1.m`)

   Figure 1 shows model performances with different number of hidden units and a single hidden layer. We can see that test accuracy increases as *nHidden* increases, but a larger moel requires longer training time. Figure 2 shows model performance with two hidden layers. The two-layer model's performance is even worse than that of a single-layer model, and it also takes a longer training time. We may infer that, in this simple digit classification problem, model with a single hidden layer is good enough to represent the features of the images.

2. Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables. That momentum uses the update

   $$w^{t+1} = w^t - \alpha_t \nabla f(w^t) + \beta^t (w^t - w^{t-1})$$

   where α_t is the learning rate (step size) and β_t is the momentum strength. A common value of β_t is a constant 0.9.

   (code in `q2.m`)

   Figure 3, Figure 4, Figure 5 shows model perfromances of stochastic gradient descend (SGD) with constant learning rate (lr), SGD with exponential decay lr, and momentum method, all with $nHidden = 10$. We can see from Figure 3 that, with a constant lr of 0.01 or 0.001, the model successfully learned something; but with a large constant lr of 0.1, the model failed to converge. Besides, by comparing Figure 3, Figure 4 and Figure 5, we can see that both SGD with exponential decay lr and momentum have higher accuracy than SGD with constant lr.

3. You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations in a reasonable amount of time.

   (code in `q3.m` and `q3_vectirized_MLPLoss.m`)

   Figure 6 and Figure 7 shows model results with vectorized loss function (lr=0.001). In this way, training time can be saved without loss of model performance.

4. Add l_2-regularization (or l_1-regularization) of the weights to your loss function. For neural networks this is called *weight decay*. An alternate form of regularization that is sometimes used is *early stopping*, which is stopping training when the error on a validation set stops decreasing.

   (code in `q4.m`)

   Results are shown in Figure 8 and Figure 9 (lr=0.001). We can see that l_2-regularization improves model performance. The reason may be that over-parameterization can be avoided through this approach.

```
######
nHidden = 5
Training iteration = 0, validation error = 0.870800
Training iteration = 20000, validation error = 0.696400
Training iteration = 40000, validation error = 0.660600
Training iteration = 60000, validation error = 0.623800
Training iteration = 80000, validation error = 0.582200
时间已过 6.088888 秒。
Test error with final model = 0.533000
######
nHidden = 10
Training iteration = 0, validation error = 0.881800
Training iteration = 20000, validation error = 0.565000
Training iteration = 40000, validation error = 0.536000
Training iteration = 60000, validation error = 0.558600
Training iteration = 80000, validation error = 0.500200
时间已过 7.264565 秒。
Test error with final model = 0.454000
######
nHidden = 20
Training iteration = 0, validation error = 0.868000
Training iteration = 20000, validation error = 0.464800
Training iteration = 40000, validation error = 0.447800
Training iteration = 60000, validation error = 0.418200
Training iteration = 80000, validation error = 0.402800
时间已过 10.714467 秒。
Test error with final model = 0.409000
######
nHidden = 50
Training iteration = 0, validation error = 0.878600
Training iteration = 20000, validation error = 0.303200
Training iteration = 40000, validation error = 0.316800
Training iteration = 60000, validation error = 0.279800
Training iteration = 80000, validation error = 0.289400
时间已过 28.767429 秒。
Test error with final model = 0.274000
######
nHidden = 100
Training iteration = 0, validation error = 0.903400
Training iteration = 20000, validation error = 0.252800
Training iteration = 40000, validation error = 0.251400
Training iteration = 60000, validation error = 0.247800
Training iteration = 80000, validation error = 0.247000
时间已过 59.395910 秒。
Test error with final model = 0.228000
```

Figure 1: Different *nHidden*, single hidden layer

```
######
nHidden =     10     5

Training iteration = 0, validation error = 0.870800
Training iteration = 20000, validation error = 0.729800
Training iteration = 40000, validation error = 0.699400
Training iteration = 60000, validation error = 0.672800
Training iteration = 80000, validation error = 0.647000
时间已过 30.687436 秒。
Test error with final model = 0.615000
```

Figure 2: Two hidden layers

```
######
SGD with constant lr = 0.001000
Training iteration = 0, validation error = 0.870400
Training iteration = 20000, validation error = 0.631400
Training iteration = 40000, validation error = 0.615000
Training iteration = 60000, validation error = 0.580400
Training iteration = 80000, validation error = 0.559400
时间已过 7.826398 秒。
Test error with final model = 0.484000
######
SGD with constant lr = 0.010000
Training iteration = 0, validation error = 0.928600
Training iteration = 20000, validation error = 0.603400
Training iteration = 40000, validation error = 0.385800
Training iteration = 60000, validation error = 0.314200
Training iteration = 80000, validation error = 0.257600
时间已过 7.028325 秒。
Test error with final model = 0.305000
######
SGD with constant lr = 0.100000
Training iteration = 0, validation error = 0.884000
Training iteration = 20000, validation error = 0.907400
Training iteration = 40000, validation error = 0.896600
Training iteration = 60000, validation error = 0.911400
Training iteration = 80000, validation error = 0.894600
时间已过 6.901854 秒。
Test error with final model = 0.894000
```

Figure 3: SGD with different constant learning rates

```
######
Exponential Decay, lr = 0.001000, decay rate = 0.900000
Training iteration = 0, validation error = 0.914800
Training iteration = 20000, validation error = 0.582200
Training iteration = 40000, validation error = 0.547600
Training iteration = 60000, validation error = 0.524600
Training iteration = 80000, validation error = 0.462600
时间已过 7.754717 秒。
Test error with final model = 0.415000
```

Figure 4: Exponential decay learning rate

```
######
Momentum, lr = 0.001000, momentum strength = 0.900000
Training iteration = 0, validation error = 0.934000
Training iteration = 20000, validation error = 0.604600
Training iteration = 40000, validation error = 0.556400
Training iteration = 60000, validation error = 0.499400
Training iteration = 80000, validation error = 0.478400
时间已过 7.922191 秒。
Test error with final model = 0.434000
```

Figure 5: Momentum

```
######
Use vectorized loss, nHidden = 10
Training iteration = 0, validation error = 0.898600
Training iteration = 20000, validation error = 0.598800
Training iteration = 40000, validation error = 0.586200
Training iteration = 60000, validation error = 0.553000
Training iteration = 80000, validation error = 0.519800
时间已过 3.574077 秒。
Test error with final model = 0.478000
```

Figure 6: Vectorized loss, single hidden layer

```
######
Use vectorized loss, nHidden =     10     5

Training iteration = 0, validation error = 0.891200
Training iteration = 20000, validation error = 0.748400
Training iteration = 40000, validation error = 0.689200
Training iteration = 60000, validation error = 0.637000
Training iteration = 80000, validation error = 0.610000
时间已过 24.640204 秒。
Test error with final model = 0.580000
```

Figure 7: Vectorized loss, two hidden layers

```
######
Use vectorized loss and l2 regularization, lambda = 0.01, nHidden = 10
Training iteration = 0, validation error = 0.901000
Training iteration = 20000, validation error = 0.555600
Training iteration = 40000, validation error = 0.544400
Training iteration = 60000, validation error = 0.505800
Training iteration = 80000, validation error = 0.372600
时间已过 3.900651 秒。
Test error with final model = 0.269000
```

Figure 8: l_2-regularization, single hidden layer

5. Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class. Recall that the softmax function is

$$p(y_i) = \frac{\exp(z_i)}{\sum_{j=1}^{J} \exp(z_j)}$$

you can replace squared error with the negative log-likelihood of the true label under this loss, $-\log p(y_i)$.

(code in `q5.m` and `q5_CE_MLPLoss.m`)

Results are shown in Figure 10 and Figure 11. With a softmax layer, the model performance improves with a small increase in training time. Besides, the learning rate should be adjusted to fit the new model.

6. Instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant, so that each layer has a bias.

(code in `q6.m`, `q6_MLP_with_bias.m`, and `q6_Predict_with_bias.m`)

Results are shown in Figure 12 and Figure 13. The model performances are improved with bias, with increased training time. The lr also needs to be tuned.

7. Implement "dropout", in which hidden units are dropped out with probability p during training. A common choice is $p = 0.5$.

(code in `q7.m` and `q7_MLP_with_dropout.m`)

```
######
Use vectorized loss and l2 regularization, lambda = 0.01, nHidden =     10     5

Training iteration = 0, validation error = 0.898800
Training iteration = 20000, validation error = 0.741600
Training iteration = 40000, validation error = 0.680200
Training iteration = 60000, validation error = 0.621800
Training iteration = 80000, validation error = 0.567600
时间已过 23.204474 秒。
Test error with final model = 0.437000
```

Figure 9: l_2-regularization, two hidden layers

```
######
Use softmax, cross entropy loss, lr = 0.001000,nHidden = 10
Training iteration = 0, validation error = 0.915000
Training iteration = 20000, validation error = 0.838600
Training iteration = 40000, validation error = 0.761400
Training iteration = 60000, validation error = 0.718400
Training iteration = 80000, validation error = 0.683800
时间已过 5.884562 秒。
Test error with final model = 0.620000
######
Use softmax, cross entropy loss, lr = 0.010000,nHidden = 10
Training iteration = 0, validation error = 0.886400
Training iteration = 20000, validation error = 0.480200
Training iteration = 40000, validation error = 0.373200
Training iteration = 60000, validation error = 0.319400
Training iteration = 80000, validation error = 0.287000
时间已过 5.747954 秒。
Test error with final model = 0.279000
```

Figure 10: softmax, single hidden layer

```
######
Use softmax, cross entropy loss, lr = 0.001000, nHidden =     10     5

Training iteration = 0, validation error = 0.908600
Training iteration = 20000, validation error = 0.893200
Training iteration = 40000, validation error = 0.865200
Training iteration = 60000, validation error = 0.849400
Training iteration = 80000, validation error = 0.835000
时间已过 27.650347 秒。
Test error with final model = 0.803000
######
Use softmax, cross entropy loss, lr = 0.010000, nHidden =     10     5

Training iteration = 0, validation error = 0.888200
Training iteration = 20000, validation error = 0.683600
Training iteration = 40000, validation error = 0.593800
Training iteration = 60000, validation error = 0.518400
Training iteration = 80000, validation error = 0.474600
时间已过 26.883035 秒。
Test error with final model = 0.426000
```

Figure 11: softmax, two hidden layers

```
######
With bias, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.901600
Training iteration = 20000, validation error = 0.540400
Training iteration = 40000, validation error = 0.503600
Training iteration = 60000, validation error = 0.465600
Training iteration = 80000, validation error = 0.430000
时间已过 9.907717 秒。
Test error with final model = 0.398000
######
With bias, lr = 0.010000, nHidden = 10
Training iteration = 0, validation error = 0.909200
Training iteration = 20000, validation error = 0.381000
Training iteration = 40000, validation error = 0.283800
Training iteration = 60000, validation error = 0.254200
Training iteration = 80000, validation error = 0.236000
时间已过 9.490441 秒。
Test error with final model = 0.214000
```

Figure 12: with bias, single hidden layer

```
######
With bias, lr = 0.001000, nHidden =     10     5

Training iteration = 0, validation error = 0.939800
Training iteration = 20000, validation error = 0.799800
Training iteration = 40000, validation error = 0.787400
Training iteration = 60000, validation error = 0.772000
Training iteration = 80000, validation error = 0.756200
时间已过 34.420852 秒。
Test error with final model = 0.680000
######
With bias, lr = 0.010000, nHidden =     10     5

Training iteration = 0, validation error = 0.920400
Training iteration = 20000, validation error = 0.480400
Training iteration = 40000, validation error = 0.399200
Training iteration = 60000, validation error = 0.410200
Training iteration = 80000, validation error = 0.510200
时间已过 33.911632 秒。
Test error with final model = 0.453000
```

Figure 13: with bias, two hidden layers

Results are shown in Figure 14, with $p \in (0.1, 0.2, 0.3, 0.4, 0.5)$ tested. We can see that the model performance decreases as p increases. The reason may be that the model is relatively small (with $nHidden = 10$), so every neuron is somehow important in the hidden layer.

8. You can do 'fine-tuning' of the last layer. Fix the parameters of all the layers except the last one, and solve for the parameters of the last layer exactly as a convex optimization problem. E.g., treat the input to the last layer as the features and use techniques from earlier in the course (this is particularly fast if you use the squared error, since it has a closed-form solution).

   (code in `q8.m` and `q8_Predict_finetune.m`)

   Results are shown in Figure 15, with $nHidden \in (10, 15, 20)$. We can see that the model performance slightly improves if we use the finetuning method.

9. You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.

   (code in `q9.m`, `data_augmentation.m` and `visualize_image.m`)

   I implemented data augmentation by applying small rotations and moves to the original images. An example is shown in Figure 16 (from left to right: original, rotate 15° anti-clockwise, rotate 15° clockwise, move 2 pixels to the left, move 2 pixels to the right).

   Model result trained with data augmentation is shown in Figure 17. Surprisingly, there is no observable difference between augmented or not-augmented model. The reason may be that the task is relatively simple (with a small image), so the original data is enough for training.

10. Replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The Matlab conv2 function implements 2D convolutions. Filters of size 5 by 5 are a common choice.

    (code in `q10.m`, `ConvNet.m` and `Conv_Predict.m`)

    Figure 18 shows the result of model with a convolution layer, with kernal size 5. Training time increases dramatically, but the model performance also improves greatly.

1.2 Final Model

(code in `final_model.m`)

```
######
With dropout p = 0.10, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.919600
Training iteration = 20000, validation error = 0.565000
Training iteration = 40000, validation error = 0.545000
Training iteration = 60000, validation error = 0.524200
Training iteration = 80000, validation error = 0.508000
时间已过 4.815554 秒。
Test error with final model = 0.464000
######
With dropout p = 0.20, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.890000
Training iteration = 20000, validation error = 0.612600
Training iteration = 40000, validation error = 0.570600
Training iteration = 60000, validation error = 0.562600
Training iteration = 80000, validation error = 0.563800
时间已过 3.863756 秒。
Test error with final model = 0.497000
######
With dropout p = 0.30, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.920200
Training iteration = 20000, validation error = 0.589600
Training iteration = 40000, validation error = 0.564800
Training iteration = 60000, validation error = 0.540800
Training iteration = 80000, validation error = 0.507400
时间已过 3.865295 秒。
Test error with final model = 0.497000
######
With dropout p = 0.40, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.936200
Training iteration = 20000, validation error = 0.622800
Training iteration = 40000, validation error = 0.583200
Training iteration = 60000, validation error = 0.612600
Training iteration = 80000, validation error = 0.573000
时间已过 3.917726 秒。
Test error with final model = 0.585000
######
With dropout p = 0.50, lr = 0.001000, nHidden = 10
Training iteration = 0, validation error = 0.894200
Training iteration = 20000, validation error = 0.580000
Training iteration = 40000, validation error = 0.576200
Training iteration = 60000, validation error = 0.583000
Training iteration = 80000, validation error = 0.590000
时间已过 3.876638 秒。
Test error with final model = 0.608000
```

Figure 14: with dropout, different p

```
######
Use vectorized loss, nHidden = 10
Training iteration = 0, validation error = 0.866200
Training iteration = 20000, validation error = 0.549800
Training iteration = 40000, validation error = 0.525600
Training iteration = 60000, validation error = 0.507000
Training iteration = 80000, validation error = 0.493400
时间已过 3.795337 秒。
Test error with final model = 0.474000
Test error with finetuned model = 0.460000
######
Use vectorized loss, nHidden = 15
Training iteration = 0, validation error = 0.880200
Training iteration = 20000, validation error = 0.501200
Training iteration = 40000, validation error = 0.471800
Training iteration = 60000, validation error = 0.448600
Training iteration = 80000, validation error = 0.440400
时间已过 4.456293 秒。
Test error with final model = 0.413000
Test error with finetuned model = 0.404000
######
Use vectorized loss, nHidden = 20
Training iteration = 0, validation error = 0.925400
Training iteration = 20000, validation error = 0.500200
Training iteration = 40000, validation error = 0.496000
Training iteration = 60000, validation error = 0.448800
Training iteration = 80000, validation error = 0.444200
时间已过 5.470547 秒。
Test error with final model = 0.384000
Test error with finetuned model = 0.343000
```

Figure 15: with fine-tuning

Figure 16: data augmentation example

```
######
With data augmentation, vectorized loss, nHidden = 10
Training iteration = 0, validation error = 0.913600
Training iteration = 20000, validation error = 0.647200
Training iteration = 40000, validation error = 0.636200
Training iteration = 60000, validation error = 0.588000
Training iteration = 80000, validation error = 0.531200
时间已过 4.282588 秒。
Test error with final model = 0.499000
```

Figure 17: with data augmentation

```
######
Use convolution layer, nHidden1 = 10, kernal size = 5, nhidden2 = 1
Training iteration = 0, validation error = 0.883400
Training iteration = 20000, validation error = 0.302200
Training iteration = 40000, validation error = 0.281600
Training iteration = 60000, validation error = 0.247000
Training iteration = 80000, validation error = 0.283000
时间已过 33.011187 秒。
Test error with final model = 0.246000
```

Figure 18: with convolution layer

The final model uses 4 convolution layers with kernel size 3, followed by three full-connected layers of size $[128, 64, 32]$, model result is shown in Figure 19.

The final model uses the trick of Xavier initialization [1], which can greatly accelerate the training process. As a comparison, if we train the model without Xavier initialization, the model converges much slower and fails to achieve a good performance after the same number of iterations, as shown in Figure 20 and Figure 21.

If the FC layers are removed, training time reduces by half, but model performances become worse, and Xavier initialization trick still helps, as shown in Figure 22, Figure 23 and Figure 24.

```
######
Use Xavier initialization
Use momentum, momentum strength = 0.9
lr = 0.000100
Use convolution layer, kernal size = 3
Number of convolution layers = 4
nHidden =    128    64    32

时间已过 288.312631 秒。
Test error with final model = 0.043000
```

Figure 19: the final model

```
######
Do not use Xavier initialization
Use momentum, momentum strength = 0.9
lr = 0.000100
Use convolution layer, kernal size = 3
Number of convolution layers = 4
nHidden =    128    64    32

时间已过 241.376076 秒。
Test error with final model = 0.600000
```

Figure 20: compare model A

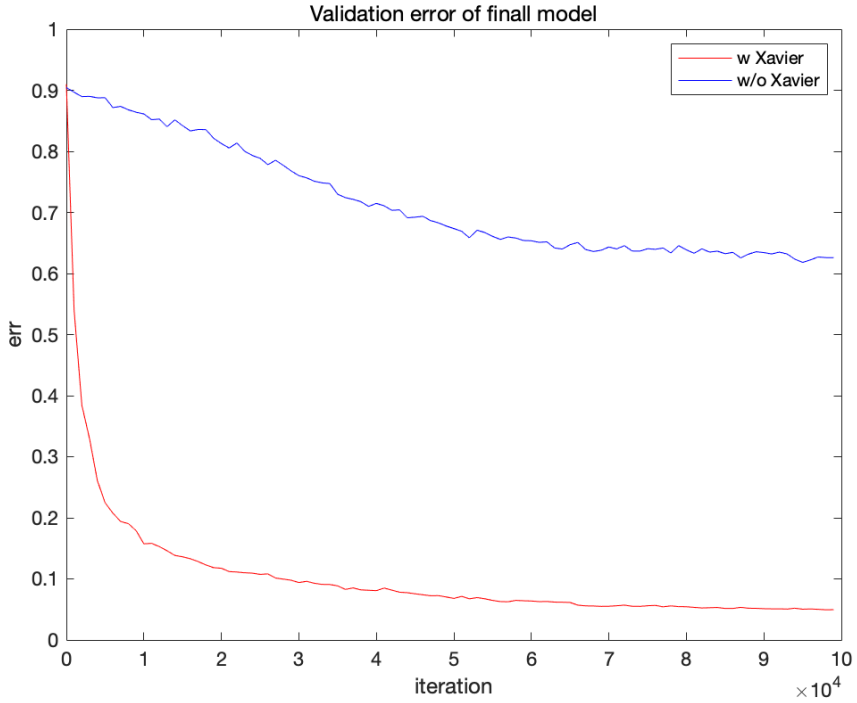

Figure 21: comparison w and w/o Xavier, with FC layers

[1] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.

```
######
Use Xavier initialization
Use momentum, momentum strength = 0.9
lr = 0.000100
Use convolution layer, kernal size = 3
Number of convolution layers = 4
no hidden layer
时间已过 128.310327 秒。
Test error with final model = 0.101000
```

Figure 22: compare model B

```
######
Do not use Xavier initialization
Use momentum, momentum strength = 0.9
lr = 0.000100
Use convolution layer, kernal size = 3
Number of convolution layers = 4
no hidden layer
时间已过 123.244838 秒。
Test error with final model = 0.751000
```

Figure 23: compare model C

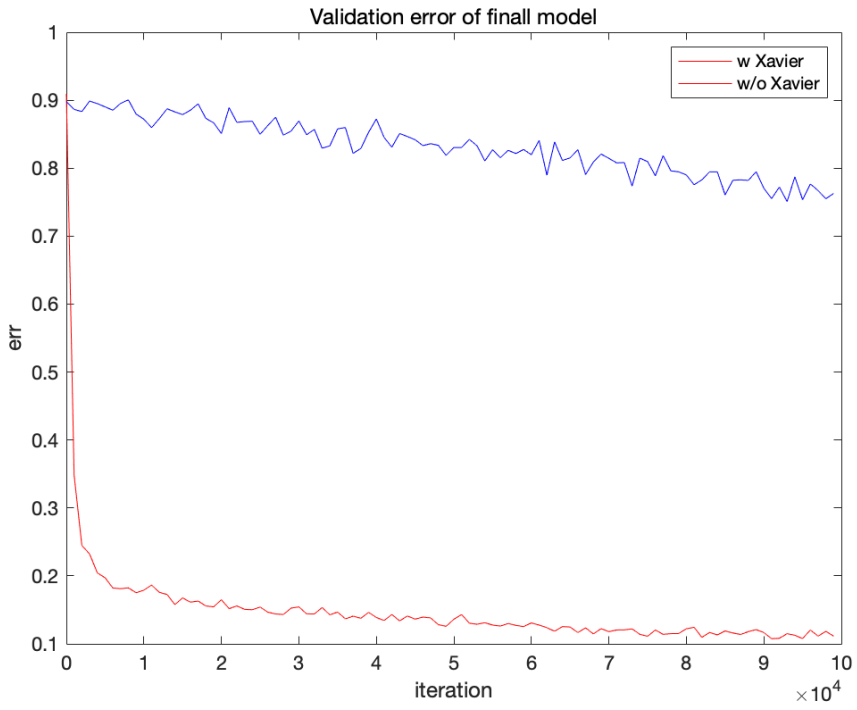

Figure 24: comparison w and w/o Xavier, w/o FC layers