Project Report

CSC 578 HW1 Siyu Nan

Introduction

This project implemented deep neural network (DNN) for the task of handwritten digit recognition. Main parts written by the author are in layer.py, loss.py and optimization.py file. Layers are built by forward and backward computation of derivative. Back propagation algorithm is implemented with stochastic gradient descent as the optimizing algorithm.

To evaluate the performance with different learning rate and different weight initialization, I built a timer in the program. In this report, performance is evaluated on accuracy rate of testing set and training time. We randomly pick sample #800 to show the predicted label, real label.

Analysis of Method

1. Learning rate: Stochastic gradient descent update with every sample. High learning rate may make the whole results affected by a specific sample whereas low learning rate may cause the loss function stuck in a not satisfactory local minimum. Therefore, an appropriate learning rate is worthy of discussion.
2. In our initial setting, Gaussian initialization is used for weight assignment. With the idea that, with data normalization, about half of the weight should

be negative and half should be positive. Setting all of them equal to zero is setting them to be their expected value, which is quite reasonable. Another try is to set bias initiation 0, keeping Gaussian initialization for the weight. Reason for the guess is similar as for the weight as expectation of bias is zero.

Result

Learning Rate exploration

Learning rate is usually set from 0.0001 to 1. In the code given, it is set to 3. Therefore, in my code, I tried from 0.0001 to 1. Pros and cons of a too large/small learning rate is analyzed above. As the result shown, a small leaning rate didn't work well in our case. The accuracy is less than 50% for learning rate less than or equal to 10^(-2). It improved with learning rate increases, and went to over 90% accuracy in the range of 0.5 to 1.2, then decreased again. So the best learning rate range, which lead us to over 90% accuracy is from 0.5 to 1.2. Time cost was very close and didn't show a trend high/lower learning rate necessarily made faster/slower.

Learning rate=0.0001, accuracy 0.101:

```
optimizer = SGD(0.0001, 10)  # learning rate 0.0001, batch size 10
```

```
Epoch 0: 1013 / 10000
The image is 8. Your prediction is 8.
--- 9.71921396255 seconds ---
```

Learning rate=0.001, accuracy 0.123:

```
optimizer = SGD(0.001, 10)  # learning rate 0.001, batch size 10
```

```
Epoch 0: 1230 / 10000
The image is 8. Your prediction is 7.
--- 9.31782603264 seconds ---
```

Learning rate=0.01, accuracy 0.489:

```
optimizer = SGD(0.01, 10)  # learning rate 0.01, batch size 10
```

```
Epoch 0: 4894 / 10000
The image is 8. Your prediction is 3.
--- 9.38849687576 seconds ---
```

Learning rate=0.1, accuracy 0.870:

```
optimizer = SGD(0.1, 10)  # learning rate 0.1, batch size 10
```

```
Epoch 0: 8701 / 10000
The image is 8. Your prediction is 5.
--- 10.031055212 seconds ---
```

Learning rate=0.5, accuracy 0.917:

```
optimizer = SGD(0.5, 10)  # learning rate 0.5, batch size 10
```

```
Epoch 0: 9173 / 10000
The image is 8. Your prediction is 8.
--- 8.93536186218 seconds ---
```

Learning rate=0.8, accuracy 0.915:

```
optimizer = SGD(0.8, 10)  # learning rate 0.8, batch size 10
```

```
----
Epoch 0: 9149 / 10000
The image is 8. Your prediction is 8.
--- 9.83679509163 seconds ---
```

Learning rate=1.2, accuracy 0.908:

```
optimizer = SGD(1.2, 10)  # learning rate 1.2, batch size 10
```

```
Epoch 0: 9077 / 10000
The image is 8. Your prediction is 5.
--- 9.20657896996 seconds ---
```

Learning rate=1.6, accuracy 0.813:

```
optimizer = SGD(1.6, 10)  # learning rate 1.6, batch size 10
```

```
Epoch 0: 8130 / 10000
The image is 8. Your prediction is 8.
--- 10.1159770489 seconds ---
```

Learning rate=2, accuracy 0.865:

```
optimizer = SGD(2, 10)  # learning rate 2, batch size 10
```

```
Epoch 0: 8645 / 10000
The image is 8. Your prediction is 8.
--- 9.87142491341 seconds ---
```

Learning rate=3, accuracy 0.788:

```
optimizer = SGD(3, 10)  # learning rate 3, batch size 10
```

```
Epoch 0: 7880 / 10000
The image is 8. Your prediction is 5.
--- 9.3538620472 seconds ---
```

Weight initialization exploration

Initial setting of w is shown in layer.py function.

```
self.W = np.random.randn(*shape)
self.b = np.random.randn(shape[0], 1)
```

We try to initialize it in a different way, setting W to be 0.

```
  def __init__(self, shape):
      self.W = [[0]*shape[1]]*shape[0]
      self.b = np.random.randn(shape[0], 1)
```

By doing this, we get the result below. In best learning rate region, [0.5,1.2], all result is better than the initial Gaussian setting, although taking a bit more time. This is very surprising. Intuitively, I think setting all of them to 0 would be a bad choice since they are all assigned with the same weight initially. That is to say, all neurons are going to compute the same output and do the same parameter update. This is against the aim of using deep neural network—to train a complex model for accurate result. I don't know how to explain this. Hopefully this could be answered by knowledge learnt in latter class.

```
optimizer = SGD(0.5, 10)  # learning rate 0.5, batch size 10

Epoch 0: 9307 / 10000
The image is 8. Your prediction is 8.
--- 10.1499068737 seconds ---

optimizer = SGD(0.8, 10)  # learning rate 0.8, batch size 10

Epoch 0: 9290 / 10000
The image is 8. Your prediction is 8.
--- 9.97492003441 seconds ---

optimizer = SGD(1.2, 10)  # learning rate 1.2, batch size 10

Epoch 0: 9117 / 10000
The image is 8. Your prediction is 5.
--- 10.0330500603 seconds ---
```

For the trail of setting initial b being 0 vector, we got results listed below. Here we used the best performing region of learning rate, [0.5,1.2]. Comparing the performance with Gaussian distribution, all except for learning rate = 0.1 perform worse than set initial to be Gaussian distribution. I tried to explain and prove this in

a mathematic way, but didn't make it. Intuitively, I would guess this is because 0 was a stage that is harder to leave, thus affected the accuracy. Further study is needed to confirm/disprove this assumption.

```
optimizer = SGD(1.2, 10)  # learning rate 1,2, batch size 10

Epoch 0: 8997 / 10000
The image is 8. Your prediction is 3.
--- 9.33703303337 seconds ---
```

```
optimizer = SGD(0.8, 10)  # learning rate 0.8, batch size 10

Epoch 0: 8888 / 10000
The image is 8. Your prediction is 8.
--- 9.56837415695 seconds ---
```

```
optimizer = SGD(0.5, 10)  # learning rate 0.5, batch size 10

Epoch 0: 9073 / 10000
The image is 8. Your prediction is 8.
--- 9.05287718773 seconds ---
```

```
optimizer = SGD(0.1, 10)  # learning rate 0.1, batch size 10

Epoch 0: 8762 / 10000
The image is 8. Your prediction is 8.
--- 9.14085102081 seconds ---
```

Conclusion

This report analyzes effects of different setting of initial weight, different setting of initial bias and different choice of learning rate. Effects are analyzed based on time consumed and accuracy rate. Since time consumed are very close, analysis is mainly focused on accuracy rate. It is found that [0.5,1.2] is best learning rate region for this specific problem. Initially setting bias vector to 0 drag down accuracy rate while

setting weight matrix to 0 increases it. The author gave her explanation in the main part but further studies are definitely needed to make general statements.