ENGR 510 Neural Networks Homework

Anthony Su

November 8, 2024

1

`X` contains the digit image examples and `y` contains the labels.

2

70000 example images are loaded. They are each 28 pixels by 28 pixels in size.

3

A 784-dimensional input layer is used because that is the dimension of the input data, which consists of 784-pixel images. A 10-dimensional output layer is used because that is the dimension of the output data, which consists of the likelihood of the digits 0-9.

4

The network has two hidden layers.

5

There is a weight between each node in a layer and each node in the adjacent layer. There are $784 \times 64 = 50176$ weights between the first and second layers. There are $64 \times 32 = 2048$ weights between the second and third layers. There are $32 \times 10 = 320$ weights between the third and fourth layers. The total number of weights is $50176 + 2048 + 320 = 52544$

6

`loss.backward()` computes the gradient and `optimizer.step()` proceeds one step in the model optimization. These are the commands which train the model in cell 6. In cell 7, the model is being tested, not trained, so the model is only evaluated, not updated.

7

$X =$ `images`

$Y =$ `pred`

$f =$ `model`

`model` also contains β internally.

8

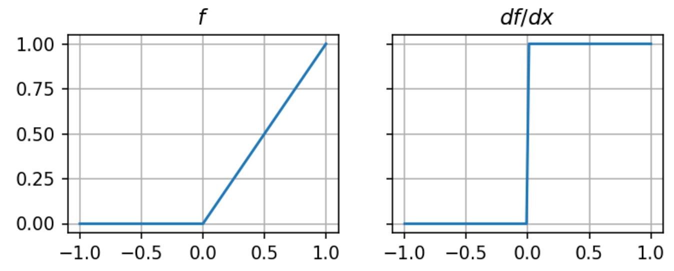

Figure 1: ReLU activation function

The derivative is queried many times when computing the gradient, and the gradient is computed many times when fitting the model. Thus, an activation function whose derivative is simple (piece-wise constant in this case) makes fitting the model much less expensive.

9

The model's prediction is compared to the truth using the cross-entropy loss function in line 37. Then, the gradient of the loss is computed using the backpropagation algorithm in line 39. The model weights are then updated such as to descend the gradient in line 40.

10

The model is trained to accurately reproduce the outputs of the training data set. However, the accuracy it achieves on the training data set does not necessarily match the accuracy it achieves in general because the model may have over-fitted to the training data set. Thus, data that was not used to train the model is used for testing how the model generalizes.

11

The model is slightly over-fit to the training data.

12

Listing 1: Original Code

```python
def my_model() -> nn.Module:
model = nn.Sequential(
    nn.Linear(784, 64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Linear(32, 10),
)
return model
```

Listing 2: Modified Code

```python
def my_model() -> nn.Module:
```

```python
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10),
)
return model
```

The original model took 43 seconds to train on my CPU. The shallower model took 58 seconds to train on my CPU.

The original model has 52544 weights. The shallower model has 203264 weights.

Since the single-hidden-layer architecture has a slightly higher accuracy on the training data set and the increase in computational cost is small, I would use that approach in the future. However, if the computational costs were orders of magnitude higher, I would prefer the double-hidden-layer architecture for its lower computational cost at similar levels of accuracy.

13

Note: the model changes in Section 12 were reverted before proceeding with this section.

Listing 3: Original Code

```python
optimizer = Adam(model.parameters(), lr=0.001)
```

Listing 4: Modified Code

```python
optimizer = SGD(model.parameters(), lr=0.001)
```

I also imported the `SGD` class from the `torch.optim` modeule.

The Adam optimization algorithm achieved a minimized loss of 0.12079 and an accuracy of 96.918% on the training data set. Its convergence is shown in Figure 2.

The SGD optimization algorithm achieved a minimized loss of 0.63998 and an accuracy of 82.224% on the training data set. Its convergence is shown in Figure 3.

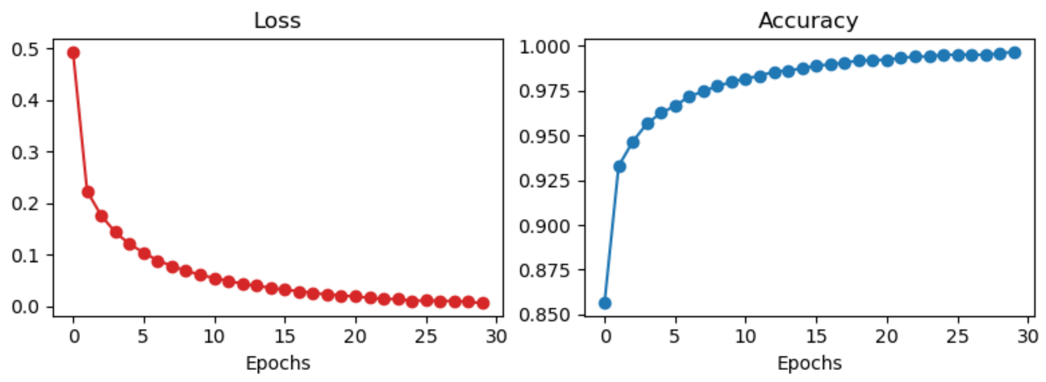

Figure 2: Convergence using Adam optimization algorithm

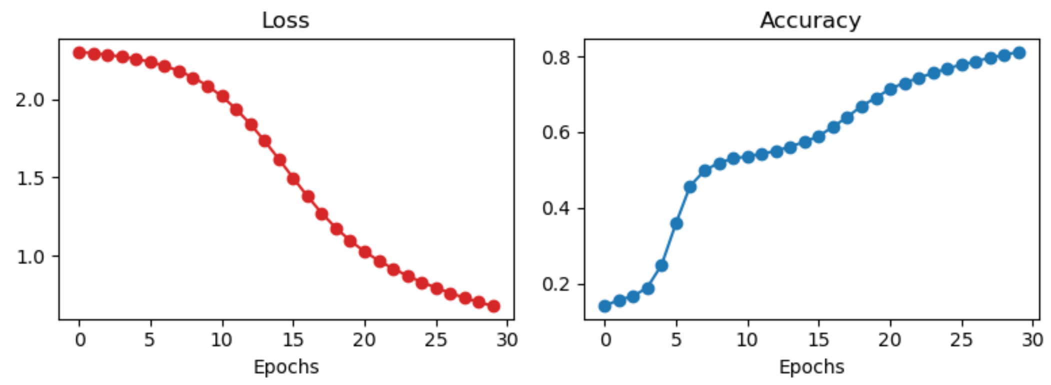

Figure 3: Convergence using SGD optimization algorithm

Adam converged with fewer epochs than SGD. In the context of iterative algorithms, convergence refers to the state where further iterations do not (significantly) further improve the quality of the answer.

14

Listing 5: Original Code

```
optimizer = SGD(model.parameters(), lr=0.001)
```

Listing 6: ModifiedCode

```
optimizer = SGD(model.parameters(), lr=0.01)
```

With a learning rate of 0.001, the model achieved a minimized loss of 0.63998 and an accuracy of 82.224% on the training data set. Its convergence is shown in Figure 4.

With a learning rate of 0.01, the model achieved a minimized loss of 0.18402 and an accuracy of 94.247% on the training datat set. Its covergence is shown in Figure 4.

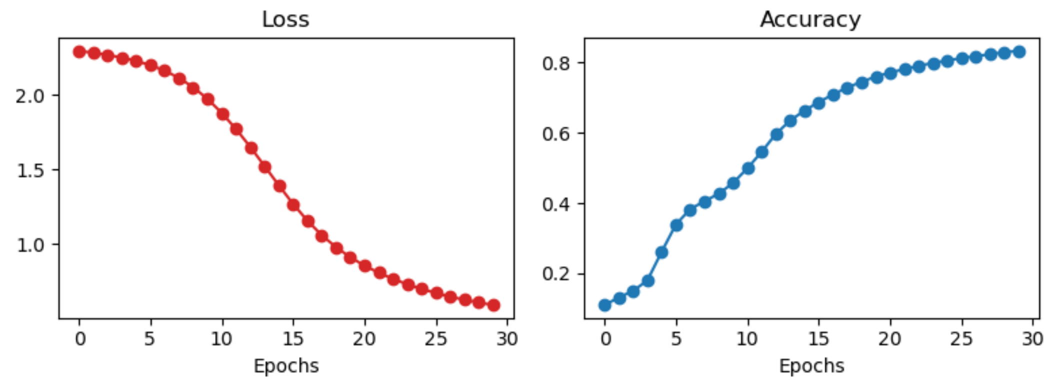

Figure 4: Convergence using learning rate of 0.001

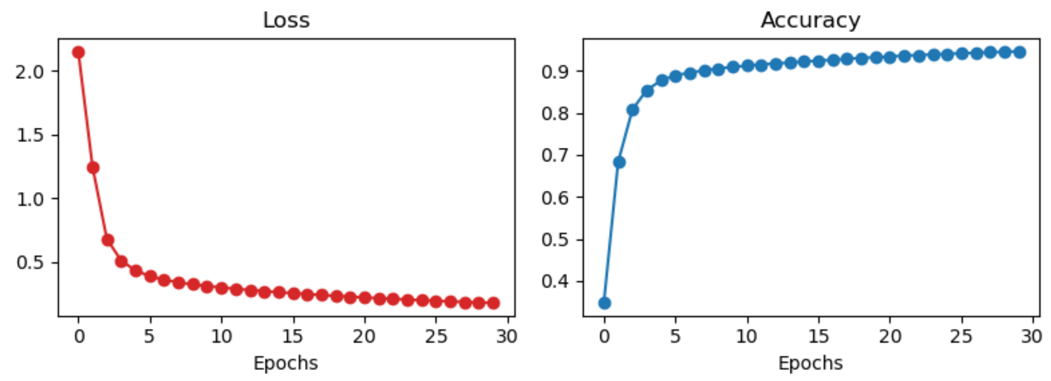

Figure 5: Convergence using learning rate of 0.01

The learning rate reduces convergence time, which increases the accuracy for a given fixed training time or number of epochs. The possible pitfall of a higher learning rate is the possibility of entirely skipping over minima.

15

The purpose of this model is to predict which of ten classes an input most probably belongs in. The cross-entropy loss function is defined in terms of probability distributions and is well-suited to tuning the model to improve this predictions of probability. In contrast, the mean squared error is defined in for general quantities and does not accurately capture the meaning and constraints inherent to probability.