**Problem 1**

1. In one-dimensional case, the cost is some function of a single variable .

When the derivative is positive, choose to be negative (i.e., decrease ). When the derivative is negative, choose to be positive (i.e., increase ). This way, it is guaranteed that is always negative, in other words, the cost C always decreases.

2. Advantage: More time and space efficient per update, since computing and storing gradients of one sample take much less time and space than with the entire dataset. This computational advantage allows for more iterations of SGD (e.g., more epochs).

Disadvantage: A mini-batch of size 1 captures much less information than the entire dataset, thus the gradients are less precise (i.e., noisier). When weights and biases are updated, they move less directly towards the optimal solution, thus are more likely to deviate from what they ought to be.

In practice, a mini-batch size somewhere between 1 and n is likely both to take advantage of the computational efficiency of SGD and to update the parameters in a more accurate manner.

3. prob1.py, mnist\_loader.py, network.py

(venv) (base) Fans-MacBook-Pro:PS2 fanfeng$ python3 prob1.py

Epoch 0: 6714 / 10000

Epoch 1: 7372 / 10000

Epoch 2: 7622 / 10000

Epoch 3: 7660 / 10000

Epoch 4: 7661 / 10000

Epoch 5: 8193 / 10000

Epoch 6: 8330 / 10000

Epoch 7: 8357 / 10000

Epoch 8: 8374 / 10000

Epoch 9: 8371 / 10000

Epoch 10: 8361 / 10000

Epoch 11: 8338 / 10000

Epoch 12: 8353 / 10000

Epoch 13: 8377 / 10000

Epoch 14: 8352 / 10000

Epoch 15: 8384 / 10000

Epoch 16: 8384 / 10000

Epoch 17: 8370 / 10000

Epoch 18: 8391 / 10000

Epoch 19: 8381 / 10000

Epoch 20: 8384 / 10000

Epoch 21: 8392 / 10000

Epoch 22: 8359 / 10000

Epoch 23: 8384 / 10000

Epoch 24: 8375 / 10000

Epoch 25: 8391 / 10000

Epoch 26: 8373 / 10000

Epoch 27: 8378 / 10000

Epoch 28: 8342 / 10000

Epoch 29: 8385 / 10000

Accuracy is 83.85%, although different runs yield different results.

**Problem 2**

1. The Hadamard product denotes the elementwise product of two vectors.

Now, let be a diagonal matrix where the j-th element along the diagonal is . Then, by **matrix multiplication**, has the dimension as , and the j-th element equals .

Therefore, .

2. Similar to problem 2.1,

Now, let be a diagonal matrix where the j-th element along the diagonal is . Then, by matrix multiplication, has the dimension as , and the j-th element equals .

Therefore, .

3. Applying results from problem 2.2 and 2.1,

4. When , , thus . Therefore,

Note that one can chain in a similar way as in problem 2.3.

Finally, and remain the same.

**Problem 3**

1. When , .

When , .

In contrast, when , , which is undefined.

When , , which is undefined.

The problem with the second expression doesn’t afflict the first expression.

2. The cross-entropy cost is,

Take the derivative of with respect to a,

When , , thus the cross-entropy cost is minimized.

When ,

3.

The derivatives of the cost with respect to weights of the second layer are,

The derivative of the cost with respect to biases of the second layer is (0.3707, -0.1086).

The derivative of the cost with respect to weights of the first layer are,

The derivative of the cost with respect to biases of the first layer are,

(0.0489, 0.0545).

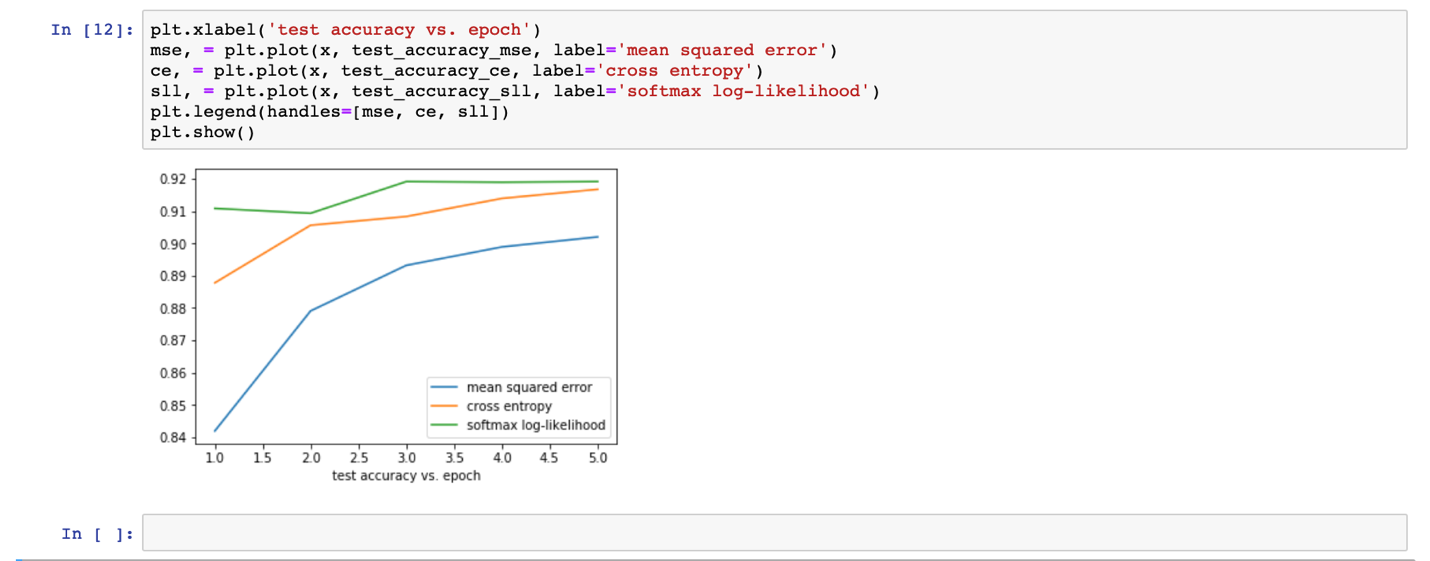
**Problem 4**

1. See **prob4\_1.py** and **ff242.ipynb**.

From the plots, it seems that mean squared error loss and cross entropy loss converge faster than softmax log-likelihood loss.

Besides, with 5 epochs, softmax log-likelihood loss and cross entropy loss achieve higher test accuracy than mean squared error loss.





2. Note that softmax cross-entropy loss is used for this problem.

|  |  |  |
| --- | --- | --- |
|  | L1 test accuracy | L2 test accuracy |
| **0 (no regularization)** | **0.9177** | **0.9177** |
| 0.0001 | 0.9180 | 0.9167 |
| 0.0005 | 0.9101 | 0.9182 |
| 0.001 | 0.8909 | 0.9186 |
| 0.005 | 0.8549 | 0.9129 |
| 0.01 | 0.1135 | 0.9123 |
| 0.1 | 0.1135 | 0.8175 |

|  |  |
| --- | --- |
|  | test accuracy |
| **1.0 (no dropout)** | **0.9177** |
| 0.2 | 0.4825 |
| 0.4 | 0.7582 |
| 0.5 | 0.8225 |
| 0.6 | 0.8545 |
| 0.8 | 0.8945 |
| 0.9 | 0.9068 |

The final results are sensitive to the parameters.

Neural networks with regularizations learn nothing from the training data when becomes too large. L2 regularization is more tolerant on large then L1 regularization.

Dropout doesn’t improve test accuracy because the neural networks don’t overfit the data.