hw6

April 23, 2015

1 Homework 6

- Name: Austin Chen
- SID: 23826762
- Repro: Open up hw6.ipynb in IPython Notebook.

1.1 1. Gradient Descent Updates

Square brackets denote element-wise operations.

1.1.1 Mean-squared Error

W1

$$J = 1/2 \sum (y_k - h_k(x))^2$$

$$dJ/dW_2 = -\sum (y_k - h_k(x)) * d/dW_2(h_k(x))$$

Note that $h(x) = g(tanh(xW_1)W_2)$, where x is a row vector. Consider $d/dW_2(h(x))$:

$$d/dW_2(h(x)) = d/dW_2(tanh(xW_1)W_2) * g'(tanh(xW_1)W_2)$$
$$= tanh(xW_1)^T * [g(tanh(xW_1)W_2)(1 - g(tanh(xW_1)W_2))]$$
$$= tanh(xW_1)^T * [h(x)(1 - h(x))]$$

Thus:

$$dJ/dW_2 = -tanh(xW_1)^T * [h(x)(1 - h(x))(y - h(x))]$$

W2

$$dJ/dW_1 = -\sum (y_k - h_k(x)) * d/dW_1(h_k(x))$$

Consider $d/dW_1(h(x))$:

$$d/dW_1(h(x)) = d/dW_1(tanh(xW_1)W_2) * g'(tanh(xW_1)W_2)$$
$$d/dW_1(tanh(xW_1)W_2) = W_2^T tanh'(xW_1)x^T$$

Thus:

$$dJ/dW_1 = -x^T * [[h(x)(1 - h(x))(y - h(x))] * W_2^T(tanh'(xW_1))]$$

1.1.2 Cross-entropy Error

$$J = \sum y_k ln(h_k(x)) + (1 - y_k) ln(1 - h_k(x))$$

$$dJ/dW = \sum d/dW y_k ln(h_k(x)) + d/dW(1 - y_k) ln(1 - h_k(x))$$

$$dJ/dW = \sum y_k d/dW ln(h_k(x)) + (1 - y_k) d/dW ln(1 - h_k(x))$$

$$dJ/dW = \sum y_k / h_k(x) d/dW h_k(x) + (1 - y_k)/(1 - h_k(x)) d/dW(-h_k(x))$$

$$dJ/dW = [y/h(x) - (1 - y)/(1 - h(x))] d/dW(h(x))$$

Using the results from the previous part, we get:

$$dJ/dW2 = -tanh(xW_1)^T * [y/h(x) - (1 - y)/(1 - h(x))][h(x)(1 - h(x))]$$

$$dJ/dW_1 = -x^T * [[h(x)(1 - h(x))[y/h(x) - (1 - y)/(1 - h(x))]] * W_2^T(tanh'(xW_1))]$$

1.2 2. Parameters

I used a learning rate of 0.01, with a pseudo-simulated annealing: every 5k iterations, if the validation accuracy hasn't increased, decrease the rate by a multiplicative factor of 0.8, resetting back to 0.01 every 100k iterations.

Weights were initialized to the normal distribution with mean 0 and variance 10^{-5}.

I got a training accuracy of 99.2%, and a Kaggle score of 96.4%.

The running time was about 10k iterations per minute, for a total of about 30-40 minutes for my final result.

Plots of accuracy over time can be found below; cross-entropy error performed much better than mean-squared error, as a loss function.

```
In [1]: import numpy as np
        import scipy.io
        import math
        import random
        from sklearn import preprocessing
        from __future__ import division

        # Load the data
        mat = scipy.io.loadmat('digit-dataset/train.mat')
        images = np.reshape(mat["train_images"], (1, -1, 60000))[0].T
        labels = mat['train_labels']
        labels = np.squeeze(np.asarray(labels))

        # Preprocess X
        X = preprocessing.scale(images.astype(float))
        X = np.c_[X, np.ones(len(X))]

        # Form y row vectors
        Y = [[1 if label == i else 0 for i in range(10)] for label in labels]
        Y = np.array(Y)
```

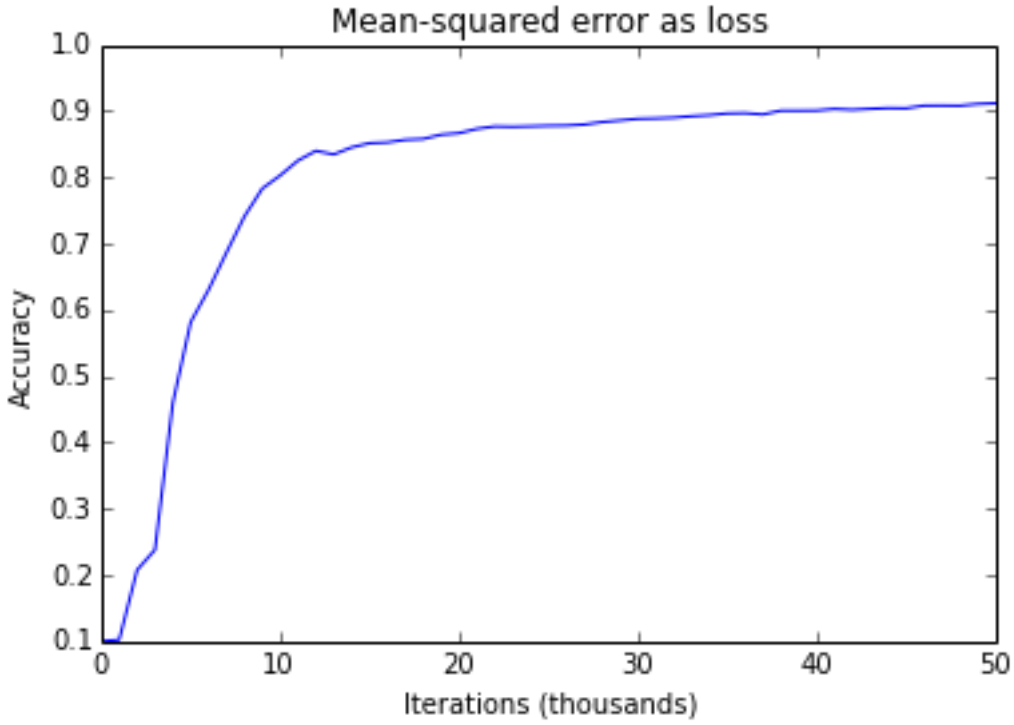
```
In [2]: class NeuralNet(object):
            def __init__(self):
                e = 10**-5
                self.W1 = np.random.normal(0, e, (785, 200))
                self.W2 = np.random.normal(0, e, (201, 10))
```

```
        def forward(self, x):
            a2 = np.tanh(x.dot(self.W1))
            a2 = np.c_[a2, np.ones(len(a2))]
            hx = sigmoid(a2.dot(self.W2))
            return hx

        def train(self, X, Y, iterations, step, error):
            for i in range(iterations):
                i = random.randrange(0, len(X))
                x, y = X[i], Y[i] # x and y are both row vectors

                a2 = np.tanh(x.dot(self.W1))
                a2 = np.append(a2, 1)
                hx = sigmoid(a2.dot(self.W2))
                d3 = np.multiply(np.multiply(hx, 1 - hx), error(y, hx))

                dJdW2 = - np.outer(a2.T, d3)
                dJdW1 = - np.outer(x.T, d3.dot(self.W2.T) * (1 - a2**2))

                self.W2 -= step * dJdW2
                self.W1 -= step * dJdW1[:,:-1]

        def predict(self, images):
            output = self.forward(images)
            return [max((v, i) for i, v in enumerate(o))[1] for o in output]

        def score(self, data, labels):
            predictions = self.predict(data)
            return sum(d == l for d, l in zip(predictions, labels)) / len(labels)

        def save(self, filename):
            np.savez(filename, self.W1, self.W2)

        def load(self, filename):
            arrs = np.load(filename)
            self.W1, self.W2 = arrs['arr_0'], arrs['arr_1']

    def sigmoid(z):
        return 1 / (1 + np.exp(-z))

    def mean_squared_error(y, hx):
        return y - hx

    def cross_entropy_error(y, hx):
        return np.divide(y, hx) - np.divide(1 - y, 1 - hx)
```

In [8]:
```
import matplotlib.pyplot as plt
%matplotlib inline

scores = [0.10]
nn = NeuralNet()
for _ in range(50):
    nn.train(X, Y, 1000, 0.01, mean_squared_error)
    scores.append(nn.score(X, labels))
```

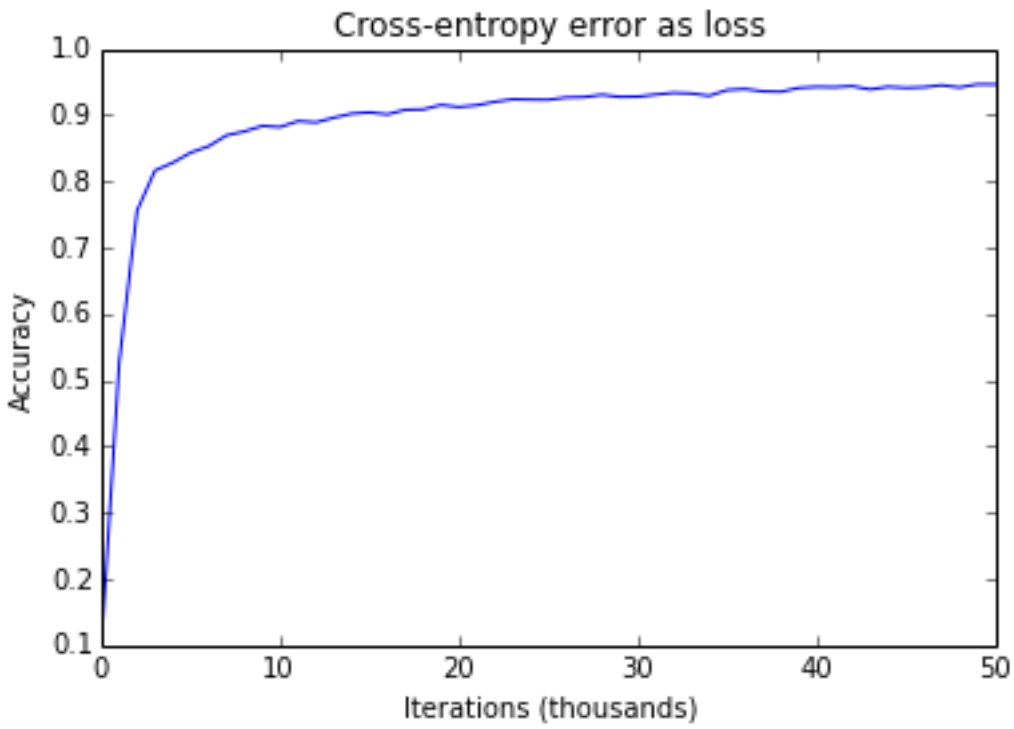
```
        plt.plot(scores)
        plt.title("Mean-squared error as loss")
        plt.ylabel("Accuracy")
        plt.xlabel("Iterations (thousands)")
```

```
Out[8]: <matplotlib.text.Text at 0x10eb18790>
```




```
In [10]: scores = [0.10]
         nn = NeuralNet()
         for _ in range(50):
             nn.train(X, Y, 1000, 0.01, cross_entropy_error)
             scores.append(nn.score(X, labels))
         plt.plot(scores)
         plt.title("Cross-entropy error as loss")
         plt.ylabel("Accuracy")
         plt.xlabel("Iterations (thousands)")
```

```
Out[10]: <matplotlib.text.Text at 0x10ec8e710>
```




```
In []: nn = NeuralNet()
       nn.load('backup.npz')
       step = 0.001
       # prev_score = 0.98

       for _ in range(40):
           nn.train(X, Y, 5000, step, cross_entropy_error)
           score = nn.score(X, labels)
           print(score)
       nn.save('backup.npz')
       #     if score < prev_score:
       #         step *= 0.8
       #         nn.load('backup.npz')
       #     else:
       #         prev_score = score
       #         nn.save('backup.npz')
```

```
In [18]: # Load and preprocess test data
         mat = scipy.io.loadmat('test.mat')
         test_images = np.reshape(mat["test_images"], (1, -1, 10000))[0].T
         test_X = preprocessing.scale(test_images.astype(float))
         test_X = np.c_[test_X, np.ones(len(test_X))]
```

```
In [38]: # Load Neural Net weights from backup
         nn = NeuralNet()
         nn.load('backup.npz')
         result = nn.predict(test_X)
```

```
# Write the results to a csv
f = open('digits6.csv', 'w')
f.write('Id,Category\n')
for i in range(len(result)):
    f.write("{0},{1}\n".format(i + 1, result[i]))
f.close()
```