Kevin Chian
23776620
CS189 Write Up
Neural Nets

Problem 1: (see scan on last page)

Problem 2:

The learning rates for MSE and cross entropy are 0.1 and 0.025 respectively. I selected points at random for each iteration and stopped training after 60,000 and 110,000 iterations for Kaggle and the plots respectively. For the weights, I initialized both using the built-in numpy function random.randn to simulate a normal distribution with mean 0 and variance 1 for every entry. Then I multiplied all entries by 0.0001 to result in very small random values. I divided all training and test images by 255 to simulate normalizing, since all entries only go up to 255, in order to avoid issues with small weight values as recommended by the GSIs.

Using a validation set of 10000 and training set of 50000, I get the following error rates on the validation set:

```
--- 718.261293173 seconds to finish training using Cross Entropy ---
Error Rate CEE:
0.0565
```

```
Kevins-MacBook-Air:hw6 kevinchian$ python neural_nets.py
--- 706.082643032 seconds to finish training using MSE ---
Error Rate MSE:
0.0784
```

For training accuracy see the plots below of iterations vs total training error. I tested on the entire training data at every 5000 iterations for 70000 iterations.

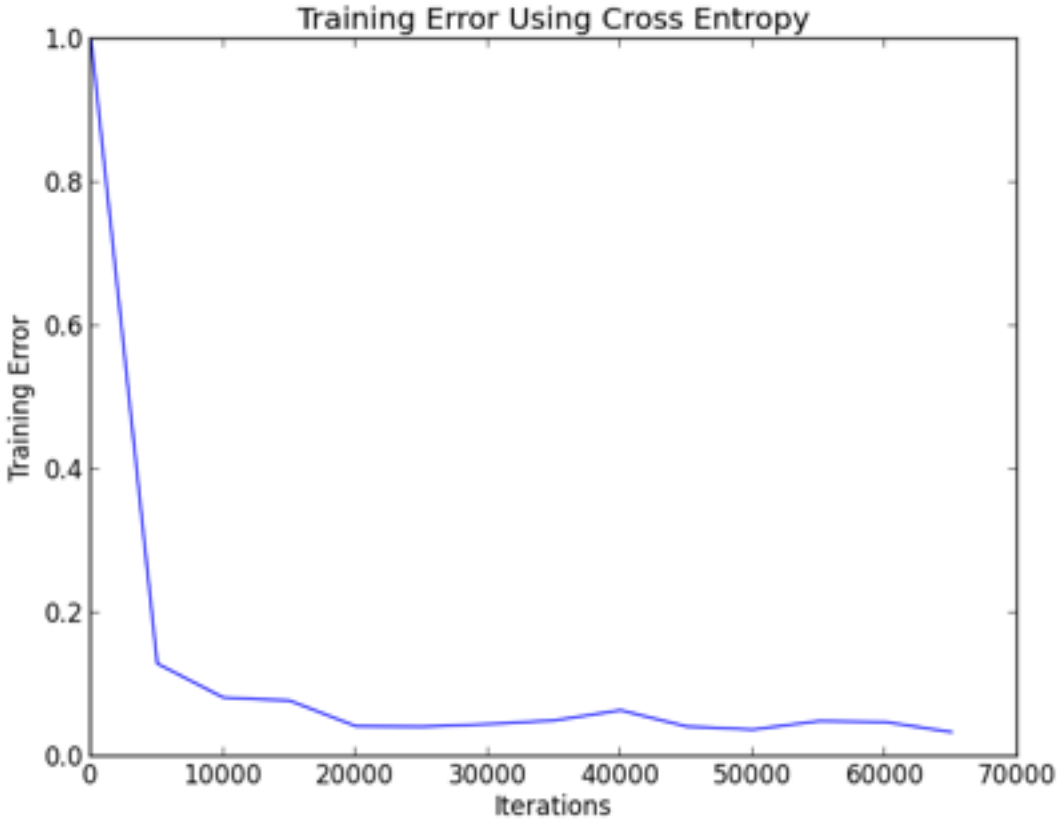
For my Kaggle submission of 60000 training points using cross entropy, I get the training time below (the error rate is the training error on 10000 random training points):

```
Kevins-MacBook-Air:hw6 kevinchian$ python -i neural_nets.py
--- 851.815284014 seconds to finish training using Cross Entropy ---
Error Rate CEE:
0.0476
```

I get a final Kaggle score of .9464 using 60000 training points.

Looking at the plots of training error for MSE and cross entropy, I find that cross entropy gets lower error rates slightly faster (see from 0 to 20000 iterations), although both eventually plateau to about the same training error given enough iterations. Using the validation set of 10000 and training of 50000, I find that cross entropy performs significantly better than MSE. The runtime of both seem to be approximately the same (~11 min) on 50000 training points.


Training Error Using MSE
Training Error
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000


Training Error Using Cross Entropy
Training Error
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
Iterations

External References:

Backpropagation by J.G. Makin https://inst.eecs.berkeley.edu/~cs182/sp06/notes/backprop.pdf

Neural network tutorial by Ryan Harris https://www.youtube.com/watch?v=aVId8KMsdUU

Problem 1

$\nabla J(w) = \frac{\partial J(w)}{\partial w_{ij}^{(l)}}$

$= \frac{\partial J(w)}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$

$= \delta_j^{(l)} \cdot x_i^{(l-1)}$

x_i^l = output @ layer l

s^l = input @ layer l

$\frac{\partial}{\partial z}\tanh(z) = 1 - \tanh^2(z)$

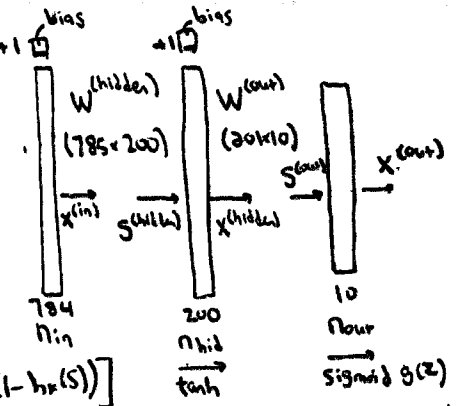
$\frac{\partial}{\partial z} g(z) = g(z)(1-g(z))$

$s_j^{(l)} = \sum_i x_i^{(l-1)} w_{ij}^{(l)}$

$\text{MSE} = J = \frac{1}{2}\sum_{k=1}^{10}(y_k - h_k(s))^2$

Cross entropy:

$J = -\sum_{k=1}^{10}\left[y_k \ln h_k(s) + (1-y_k)\ln(1-h_k(s))\right]$



* Terms used in derivation defined above

[output layer]

$\delta_j^{(out)} = \frac{\partial J(w)}{\partial s_j^{(out)}} = \frac{\partial J(w)}{\partial x_j^{(out)}} \cdot \frac{\partial x_j^{(out)}}{\partial s_j^{(out)}}$

$\frac{\partial x_j^{(out)}}{\partial s_j^{(out)}} = \frac{\partial g(s_j^{(out)})}{\partial s_j^{(out)}} = g'(s_j^{(out)})$

For MSE:

$\frac{\partial J(w)}{\partial x_j^{(out)}} = \frac{\partial}{\partial x_j^{(out)}} \frac{1}{2}\sum_j (y_j - x_j)^2$

$= 2\left(\frac{1}{2}\right)(y_j - x_j)\frac{\partial}{\partial x_j}(-x_j)$

$= -(y_j - x_j)$

$\delta_j^{(out)} = \frac{\partial J(w)}{\partial x_j^{(out)}} \cdot \frac{\partial x_j^{(out)}}{\partial s_j^{(out)}}$

$= -(y_j - x_j^{(out)})\, g'(s_j^{(out)})$

$= -(y_j - x_j^{(out)})\left(g(s_j^{(out)})(1 - g(s_j^{(out)}))\right)$

$= -(y_j - x_j^{(out)})\left(x_j^{(out)}(1 - x_j^{(out)})\right)$

For Cross Entropy:

$\frac{\partial J(w)}{\partial x_j^{(out)}} = \frac{\partial}{\partial x_j} - \sum_j \left[y_j \ln(x_j) + (1-y_j)\ln(1-x_j)\right]$

$= -\left[\frac{y_j}{x_j} + \frac{1-y_j}{1-x_j}(-1)\right]$

$= -\left[\frac{y_j(1-x_j) + (y_j - 1)x_j}{x_j(1-x_j)}\right]$

$= -\left[\frac{y_j - x_j}{x_j(1-x_j)}\right] = \frac{x_j - y_j}{x_j(1-x_j)}$

$\delta_j^{(out)} = \frac{\partial J(w)}{\partial x_j^{(out)}} \cdot \frac{\partial x_j^{(out)}}{\partial s_j^{(out)}}$

$= \frac{x_j - y_j}{x_j(1-x_j)}(x_j)(1-x_j)$

$= x_j^{(out)} - y_j$

[hidden layer]

$\delta_i^{(hidden)} = \frac{\partial J(w)}{\partial s_i^{(hidden)}} = \sum_j \frac{\partial J(w)}{\partial s_j^{(out)}} \cdot \frac{\partial s_j^{(out)}}{\partial x_i^{(hidden)}} \cdot \frac{\partial x_i^{(hidden)}}{\partial s_i^{(hidden)}}$

$= \sum_j \delta_j^{(out)} \cdot w_{ij}^{(out)} \cdot \tanh'(s_i^{(hidden)})$

$= (1 - \tanh^2(s_i^{(hidden)}))\sum_j \delta_j^{(out)} \cdot w_{ij}^{(out)}$

$= (1 - (x_i^{(hidden)})^2)\sum_j \delta_j^{(out)} w_{ij}^{(out)}$

Update Equation: $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$

Using the above inputs, we can use stochastic gradient descent to find $w^{(hidden)}$ and $w^{(out)}$