CS 189 Homework 6

Edwin Xie, John Du, Daniel Zezula

April 21, 2014

External Sources

We used CUDA to paralellize the training process.

Single layer neural network

(i) Stochastic gradient descent for loss functions:
We want to find a $n_{in} \times n_{out}$ $\frac{\partial J}{\partial \mathbf{W}}$ where $(\frac{\partial J}{\partial \mathbf{W}})_{ij}$ is taken over W_{ij}.
Define W_k such that $W = \begin{bmatrix} W_1 & W_2 & ... & W_{out} \end{bmatrix}$.
Then $\frac{\partial J}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial J}{\partial \mathbf{W}_1} & \frac{\partial J}{\partial \mathbf{W}_2} & ... & \frac{\partial J}{\partial \mathbf{W}_{out}} \end{bmatrix}$.
We also want to find a $n_{out} \times 1$ $\frac{\partial J}{\partial \mathbf{b}}$, where $\frac{\partial J}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial J}{\partial \mathbf{b}_1} & \frac{\partial J}{\partial \mathbf{b}_2} & ... & \frac{\partial J}{\partial \mathbf{b}_{out}} \end{bmatrix}^T$.
The stochastic gradient descent updates are then

$$\mathbf{W} = \mathbf{W} - \eta \frac{\partial J}{\partial \mathbf{W}}$$
$$\mathbf{b} = \mathbf{b} - \eta \frac{\partial J}{\partial \mathbf{b}}$$

for every training point x.

Derivations for mean squared error:

$$J = \frac{1}{2} \sum_{k=1}^{n_{out}} (t_k - y_k)^2$$

$$J = \frac{1}{2} (\mathbf{t} - \mathbf{y})^T (\mathbf{t} - \mathbf{y})$$

$$\frac{\partial J}{\partial \mathbf{W_k}} = -(t_k - y_k) \frac{\partial y_k}{\partial \mathbf{W_k}}$$

$$\frac{\partial y_k}{\partial \mathbf{W_k}} = y_k(1 - y_k) \frac{\partial}{\partial \mathbf{W_k}} (\mathbf{W}_k^T \mathbf{x} + b_k)$$

$$\frac{\partial y_k}{\partial \mathbf{W_k}} = y_k(1 - y_k)\mathbf{x}$$

$$\frac{\partial J}{\partial \mathbf{W_k}} = -(t_k - y_k)y_k(1-y_k)\mathbf{x}$$

$$\frac{\partial J}{\partial \mathbf{b_k}} = -(t_k - y_k)\frac{\partial y_k}{\partial \mathbf{b_k}}$$

$$\frac{\partial y_k}{\partial \mathbf{b_k}} = y_k(1-y_k)$$

$$\frac{\partial J}{\partial \mathbf{b_k}} = -(t_k - y_k)y_k(1-y_k)$$

Derivation for cross-entropy error:

$$J = -\sum_{k=1}^{n_{out}} [t_k \ln y_k + (1-t_k)\ln(1-y_k)$$

$$\frac{\partial J}{\partial \mathbf{W_k}} = \frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial \mathbf{W_k}}$$

$$\frac{\partial J}{\partial y_k} = -(\frac{t_k}{y_k} - \frac{1-t_k}{1-y_k}) = \frac{y_k - t_k}{y_k(1-y_k)}$$

$$\frac{\partial y_k}{\partial \mathbf{W_k}} = y_k(1-y_k)\mathbf{x}$$

$$\frac{\partial J}{\partial \mathbf{W_k}} = (y_k - t_k)\mathbf{x}$$

$$\frac{\partial J}{\partial \mathbf{b_k}} = (y_k - t_k)$$

Derivation for vectorizing, Forward Propagation

For one training example:

$$y_k = \sigma(\sum_{j=1}^{n_{features}} \mathbf{W}_{jk}\mathbf{x}_j + \mathbf{b}_k)$$

where $1 \leq k \leq n_{out}$ and $1 \leq j \leq n_{features}$

$$y_k = \sigma({\mathbf{W_k}}^T\mathbf{x} + \mathbf{b}_k)$$

For element-wise sigmoid:

$$\mathbf{y} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

above we have the vector dimmensions of:

$\mathbf{y} = n_{out}\times 1,\ \mathbf{b} = n_{out}\times 1,\ \mathbf{W}_k = 1\times n_{features},\ \mathbf{x} = 1\times n_{features},\ \mathbf{W} = n_{features}\times n_{out}$

For batch of m examples we want:
Y has columns of $\mathbf{y_1}$ ... $\mathbf{y_m}$

$$Y = \sigma([\mathbf{W}^T\mathbf{x}_1 + \mathbf{b} \ \ ... \ \ \mathbf{W}^T\mathbf{x}_m + \mathbf{b}])$$

$$Y = \sigma(\mathbf{W}^T\mathbf{X} + \mathbf{B})$$

B has m columns equal to **b**

$$X = m \times n_{features}$$

Derivation for vectorizing, Back Propagation

For 1 training example, mean squares:

$$\frac{\partial J}{\partial \mathbf{W_k}} = -(t_k - y_k)y_k(1 - y_k)\mathbf{x}$$

$$\delta_k^L = (y_k - t_k)y_k(1 - y_k)$$

For cross entropy:

$$\frac{\partial J}{\partial \mathbf{W_k}} = \frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial \mathbf{W_k}}$$

$$\frac{\partial J}{\partial \mathbf{W_k}} = (y_k - t_k)\mathbf{x}$$

$$\delta_k^L = (y_k - t_k)$$

In both cases continue with:

$$\frac{\partial J}{\partial \mathbf{W_k}} = \mathbf{x} \times \delta_k^L$$

define:

$$\frac{\partial J}{\partial \mathbf{W}} = [\frac{\partial J}{\partial \mathbf{W}_1} ... \frac{\partial J}{\partial \mathbf{W}_k}]$$

where $k = 1, \ ... \ n_{features}$ then:

$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{x} \times \delta^{L^T}$$

For the bias:

$$\frac{\partial J}{\partial \mathbf{b}_k} = \delta_k^L$$

define:
The rows of $\frac{\partial J}{\partial \mathbf{b}}$ are $\frac{\partial J}{\partial b_1} ... \frac{\partial J}{\partial b_{out}}$

$$\frac{\partial J}{\partial \mathbf{b}} = \delta^L$$

$$\frac{\partial J}{\partial \mathbf{W}_k} = n_{features} \times 1$$

$$\frac{\partial J}{\partial \mathbf{W}} = n_{features} \times n_{out}$$

where the columns of $\frac{\partial J}{\partial \mathbf{W}}$ are:

$$\frac{\partial J}{\partial \mathbf{W}_1} ... \frac{\partial J}{\partial \mathbf{W}_k}$$

$$\delta^L = n_{out} \times 1$$

where the columns of Δ^L are δ_1^L ... δ_m^L
for batch of m examples:

$$\frac{\partial J}{\partial \mathbf{W}_m} = X_m {\delta_m^L}^T$$

$$X \times \Delta^L = \sum_{i=1}^{m} (x_i {\delta_i^L}^T)$$

$$X \times \Delta^L = \sum_{i=1}^{m} (\frac{\partial J}{\partial \mathbf{W_i}})$$

$$X \times \Delta^L = \frac{\partial J}{\partial \mathbf{W}}$$

Which is what we want for batch gradient descent. Similarly:

$$\frac{\partial J}{\partial \mathbf{b}} = \sum_{i=1}^{m} \delta_i^L$$

$$\frac{\partial J}{\partial \mathbf{b}} = \Delta^L [1_1 ... 1_m]$$

Where Δ^L has dimensions $n_{out} \times m$

Multilayer feed forward neural network

(i) Parameter update equations for loss functions at output layer:

$$\frac{\partial J}{\mathbf{W}_{ij}^l} = \frac{\partial J}{\mathbf{y}_j^l} \cdot \frac{\partial \mathbf{y}_j^l}{\mathbf{s}_j^l} \cdot \frac{\partial \mathbf{s}_j^l}{\mathbf{W}_{ij}^l}$$

For mean squares:

$$\frac{\partial J}{\mathbf{y}_j^l} \cdot \frac{\partial \mathbf{y}_j^l}{\mathbf{s}_j^l} = (\mathbf{y}_i^L - t_j)\mathbf{y}_j^L(1 - \mathbf{y}_j^L) = \delta_j^L$$

For cross entropy:

$$\frac{\partial J}{\mathbf{y}_j^l} \cdot \frac{\partial \mathbf{y}_j^l}{\mathbf{s}_j^l} = \frac{\mathbf{y}_j^L - t_k}{\mathbf{y}_j^L(1-\mathbf{y}_j^L)} \mathbf{y}_j^L(1-\mathbf{y}_j^L) = (\mathbf{y}_i^L - t_j) = \delta_j^L$$

Continue in either case with:

$$\frac{\partial \mathbf{s}_j^l}{\mathbf{W}_{ij}^l} = \mathbf{y}_i^{l-1}$$

Inductive update for the hidden layers:

$$\delta_i^{l-1} = \frac{\partial J}{\partial \mathbf{s}_i^{l-1}} = \sum_{j=1}^{d(l)} \frac{\partial J}{\partial \mathbf{s}_j^l} \cdot \frac{\partial \mathbf{s}_j^l}{\partial \mathbf{y}_i^{l-1}} \cdot \frac{\partial \mathbf{y}_i^{l-1}}{\partial \mathbf{s}_i^{l-1}}$$

$$= \sum_{j=1}^{d(l)} (\delta_j^l \cdot \mathbf{W}_{ij}^l \cdot (1 - \mathbf{y}_i^{l-1^2}))$$

$$= (1 - \mathbf{y}_i^{l-1^2}) \sum_{j=1}^{d(l)} (\delta_j^l \cdot \mathbf{W}_{ij}^l)$$

For mean squares:

$$\delta^L = (\mathbf{y}^L - \mathbf{t}) \circ (\mathbf{y}^L - \mathbf{y}^{L^2})$$

For cross entropy:

$$\delta^L = (\mathbf{y}^L - \mathbf{t})$$

Where $\circ$ is the Hadamard product.

$$\delta^{l-1} = (\mathbf{1} - \mathbf{y}^{l-1^2}) \circ (\mathbf{W}^l \delta^l)$$

$\mathbf{y}$ has dimensions $d^{l-1} \times n$
$\delta^{\mathbf{L^T}}$ has dimensions $d^l \times n$

$$\frac{\partial J}{\mathbf{W}^l} = \mathbf{y}^{l-1} \cdot \delta^{l^T}$$

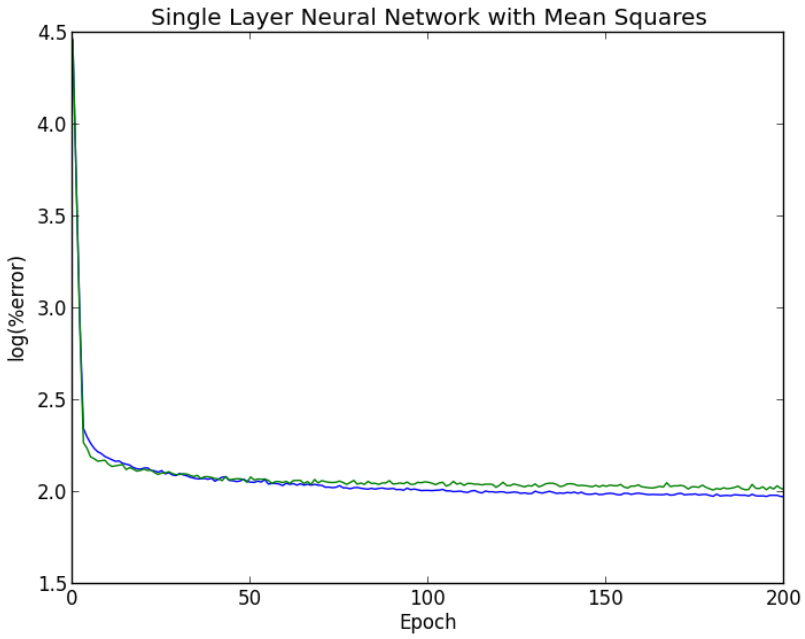
$$\frac{\partial J}{\mathbf{b}^l} = \delta^l$$

$$\mathbf{y}^0 = \mathbf{x}$$

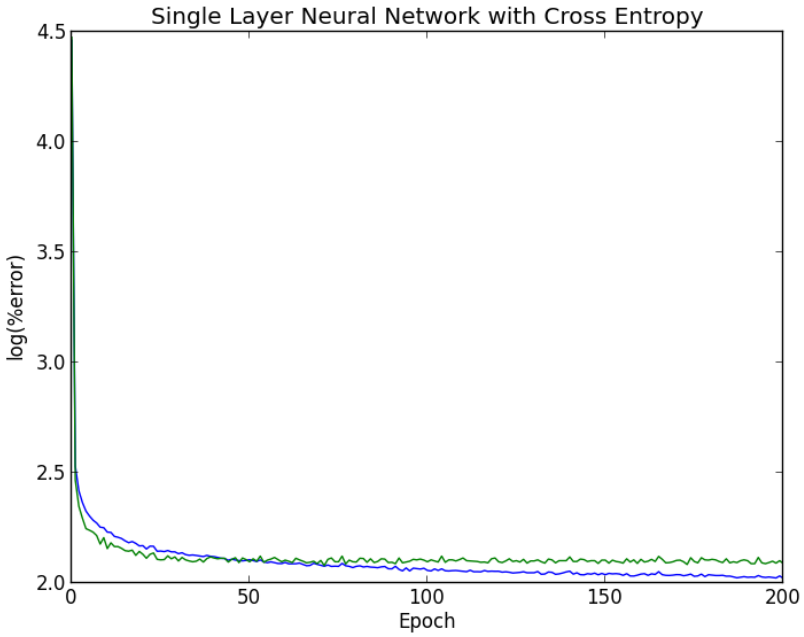
$$\delta^{l-1} = \frac{\partial \mathbf{y}^{l-1}}{\partial \mathbf{s}^{l-1}} \mathbf{W}^l \delta^l$$

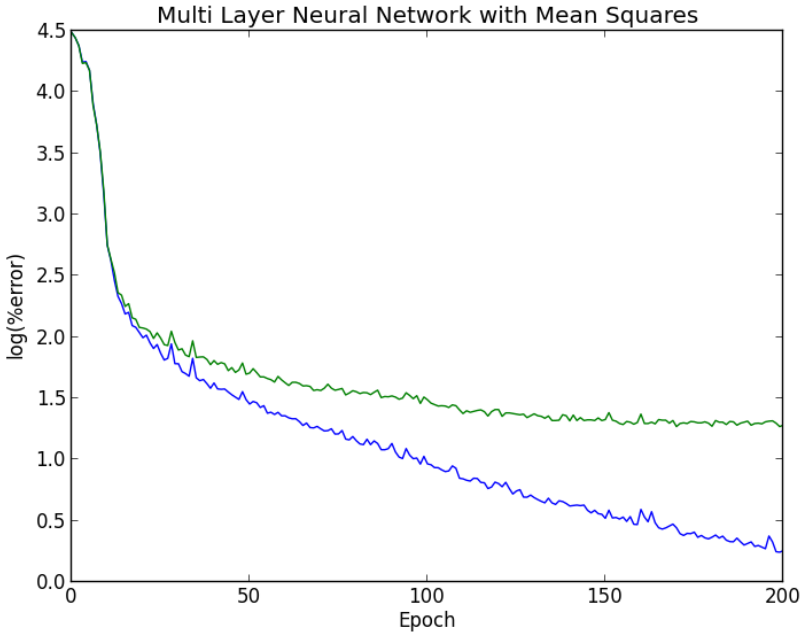
Results

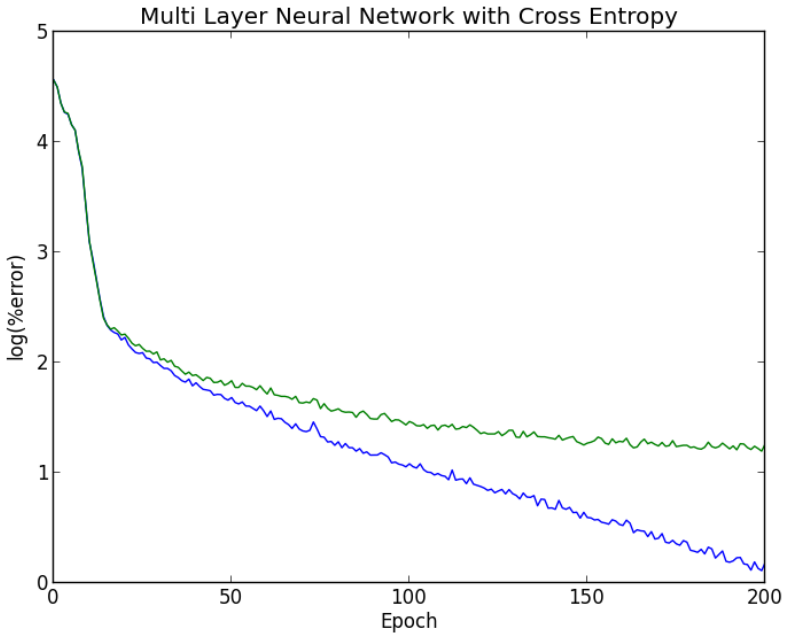
We implemented a single layer neural network with both mean squares and cross entropy as error functions. We alson implemented a two hidden layer multi layer neural network with both mean squares and cross entropy loss functions. The graphs of these can be seen below, where the green line is the test set, and blue line is the training set. The lowest error we received is with the multi layer neural network with the cross entropy error function, which resulted in a



1.26


Single Layer Neural Network with Cross Entropy
log(%error)
Epoch
4.5
4.0
3.5
3.0
2.5
2.0
0
50
100
150
200


Multi Layer Neural Network with Mean Squares
log(%error)
Epoch
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200


Multi Layer Neural Network with Cross Entropy
log(%error)
Epoch
0
1
2
3
4
5
0
50
100
150
200