

# CS 189 Homework 6

Edwin Xie, John Du, Daniel Zezula

April 21, 2014

## External Sources

We used CUDA to parallelize 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  $W = [W_1 \ W_2 \ \dots \ W_{out}]$ .

Then  $\frac{\partial J}{\partial \mathbf{W}} = [\frac{\partial J}{\partial \mathbf{W}_1} \ \frac{\partial J}{\partial \mathbf{W}_2} \ \dots \ \frac{\partial J}{\partial \mathbf{W}_{out}}]$ .

We also want to find a  $n_{out} \times 1$   $\frac{\partial J}{\partial \mathbf{b}}$ , where  $\frac{\partial J}{\partial \mathbf{b}} = [\frac{\partial J}{\partial b_1} \ \frac{\partial J}{\partial b_2} \ \dots \ \frac{\partial J}{\partial b_{out}}]^T$ .

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:**

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

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

$$\frac{\partial J}{\partial \mathbf{W}_{\mathbf{k}}} = -(t_k - y_k) \frac{\partial y_k}{\partial \mathbf{W}_{\mathbf{k}}}$$

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

$$\frac{\partial y_k}{\partial \mathbf{W}_{\mathbf{k}}} = y_k(1 - y_k) \mathbf{x}$$

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{\mathbf{k}}} &= -(t_k - y_k)y_k(1 - y_k)\mathbf{x} \\
\frac{\partial J}{\partial \mathbf{b}_{\mathbf{k}}} &= -(t_k - y_k)\frac{\partial y_k}{\partial \mathbf{b}_{\mathbf{k}}} \\
\frac{\partial y_k}{\partial \mathbf{b}_{\mathbf{k}}} &= y_k(1 - y_k) \\
\frac{\partial J}{\partial \mathbf{b}_{\mathbf{k}}} &= -(t_k - y_k)y_k(1 - y_k)
\end{aligned}$$

**Derivation for cross-entropy error:**

$$\begin{aligned}
J &= - \sum_{k=1}^{n_{out}} [t_k \ln y_k + (1 - t_k) \ln (1 - y_k)] \\
\frac{\partial J}{\partial \mathbf{W}_{\mathbf{k}}} &= \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial \mathbf{W}_{\mathbf{k}}} \\
\frac{\partial J}{\partial y_k} &= -\left(\frac{t_k}{y_k} - \frac{1 - t_k}{1 - y_k}\right) = \frac{y_k - t_k}{y_k(1 - y_k)} \\
\frac{\partial y_k}{\partial \mathbf{W}_{\mathbf{k}}} &= y_k(1 - y_k)\mathbf{x} \\
\frac{\partial J}{\partial \mathbf{W}_{\mathbf{k}}} &= (y_k - t_k)\mathbf{x} \\
\frac{\partial J}{\partial \mathbf{b}_{\mathbf{k}}} &= (y_k - t_k)
\end{aligned}$$

**Forward Propagation** For one training example:

$$y_k = \sigma\left(\sum_{j=1}^{n_{features}} \mathbf{W}_{jk}x_j\right) + \mathbf{b}_{\mathbf{k}}$$

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

$$y_k = \sigma(\vec{\mathbf{W}}_{\mathbf{k}}^T \vec{x} + \mathbf{b}_{\mathbf{k}})$$

For element-wise sigmoid:

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

above we have the vector dimensions of:

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

For batch of m examples we want:

$$\begin{aligned}
Y &= [\vec{y}_1 \dots \vec{y}_m]_{n_{out}} \\
Y &= \sigma([\mathbf{W}^T x_1 + \vec{b} \dots \mathbf{W}^T x_m + \vec{b}])_{n_{out}} \\
Y &= \sigma(\mathbf{W}^T X + B)
\end{aligned}$$

where  $B = \vec{b} \times [1_1 \dots 1_m]$

**Back Propagation** For 1 training example, mean squares:

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = -\frac{\partial}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} \frac{1}{2} (t_k - y_k)^2$$

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = -(t_k - y_k) \frac{\partial}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} y_k$$

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = -(t_k - y_k) \frac{\partial}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} \sigma(\vec{\mathbf{W}}_{\mathbf{k}}^T \vec{x} + \mathbf{b}_{\mathbf{k}})$$

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

For cross entropy:

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial \vec{\mathbf{W}}_{\mathbf{k}}}$$

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = (y_k - t_k) \mathbf{x}$$

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

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = \vec{x} \cdot \delta_k^L$$

define:

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = \left[ \frac{\partial J}{\partial \vec{\mathbf{W}}_1} \dots \frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} \right]$$

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

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

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

define:

$$\frac{\partial J}{\partial \vec{b}} = \left[ \frac{\partial J}{\partial b_1} \dots \frac{\partial J}{\partial b_{out}} \right]'$$

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

$$\sigma' = (y_k)(1 - y_k) \quad \tanh' = 1 - \tanh^2$$

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

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_{\mathbf{k}}} = n_{features} \times 1$$

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

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

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

$$\frac{\partial J}{\partial \vec{\mathbf{W}}_1} \cdots \frac{\partial J}{\partial \vec{\mathbf{W}}_k}$$

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

$$\Delta^L = n_{out} \times m$$

where the columns of  $\Delta^L$  are  $\delta_1^L \dots \delta_m^L$  for batch of m examples:

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

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

$$X \Delta^L = \sum_{i=1}^m \left( \frac{\partial J}{\partial \mathbf{W}_i} \right)$$

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

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

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

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

## Multilayer feed forward neural network

(i) Parameter update equations for loss functions:

$$\frac{\partial J}{\partial W_{ij}^l} = \frac{\partial J}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial s_j^l} \cdot \frac{\partial s_j^l}{\partial W_{ij}^l}$$

For mean squares:

$$\frac{\partial J}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial s_j^l} = (y_j^L - t_j) y_j^L (1 - y_j^L) = \delta_j^L$$

For cross entropy:

$$\frac{\partial J}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial s_j^l} = \frac{y_j^L - t_k}{y_j^L (1 - y_j^L)} y_j^L (1 - y_j^L) = (y_j^L - t_j) = \delta_j^L$$

$$\begin{aligned}
\frac{\partial s_j^l}{W_{ij}^l} &= y_i^{l-1} \\
\delta_i^{l-1} &= \frac{\partial J}{\partial s_i^{l-1}} = \sum_{j=1}^{d(l)} \frac{\partial J}{\partial s_j^l} \cdot \frac{\partial s_j^l}{\partial y_i^{l-1}} \cdot \frac{\partial y_i^{l-1}}{\partial s_i^{l-1}} \\
&= \sum_{j=1}^{d(l)} (\delta_j^l \cdot W_{ij}^l \cdot y_i^{l-1} (1 - y_i^{l-1})) \\
&= y_i^l (1 - y_i^l) \sum_{j=1}^{d(l)} (\delta_j^l \cdot W_{ij}^l) \\
\delta^L &= (y^L - t) \circ (y^L - y^{L^2})
\end{aligned}$$

Where  $\circ$  is the Hadamard product.

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

Dimension check:

$$d(l-1) \times n = (d(l-1) \times n) \cdot (d(l-1) \times d(l)) \cdot (d(l) \times n)$$

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

Dimension check:

$$(d(l-1) \times d(l)) = (d(l-1) \times n) \cdot (n \times d(l))$$

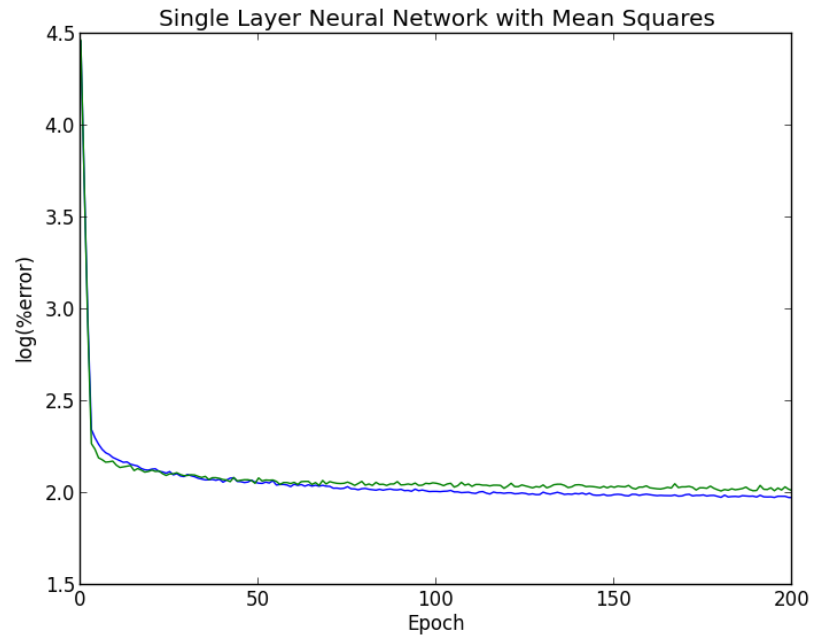
$$\frac{\partial J}{b^l} = \delta^l$$

$$y^0 = x$$

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

## Results

We implemented a single layer neural network with both mean squares and cross entropy as error functions. We also 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

