Report on UFLDL-Part I

Yunfei WANG

[1]School of Computer Science & Technology
Huazhong University of Science & Technology

June 4th, 2013

Table of contents

Forward Propagation

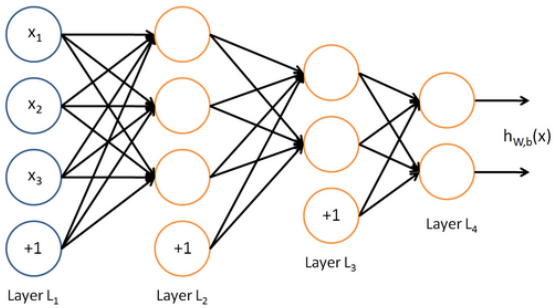

Figure: Neural Network

Forward Propagation

Compute activation of units one layer after another:

$$
\begin{aligned}
z^{l+1} &= W^l a^l + b^l \\
a^{l+1} &= f(z^{l+1})
\end{aligned}
\tag{1}
$$

where z_i^l is weighted sum of inputs to unit i in layer l,a_i^l denotes activation of unit i in layer l.

Back Propagation

Cost Function for a single sample

Measure the divergence between output and ideal targets.

$$J(W,b;x,y) = \frac{1}{2}\|h_{W,b}(x) - y\|^2 \tag{2}$$

Overall Cost Function

$$J(W,b) = \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x_i,y_i)\right]}_{Mean\ Square\ Error} + \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_l}\sum_{j=1}^{n_{l+1}}(W_{ji}^l)^2}_{Weight\ Decay} \tag{3}$$

$$\left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\|h_{W,b}(x_i) - y_i\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_l}\sum_{j=1}^{n_{l+1}}(W_{ji}^l)^2$$

Optimization Algorithm

$J(W,b)$ is a non-convex function,gradient descent is susceptible to local optima. In practice,it usually works fairly well.

$$W_{ij}^l = W_{ij}^l - \alpha \frac{\partial J(W,b)}{\partial W_{ij}^l} \tag{4}$$

$$b_i^l = b_i^l - \alpha \frac{\partial J(W,b)}{\partial b_i^l} \tag{5}$$

Derivative of overall cost function

$$\frac{\partial}{\partial W_{ij}^l} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^l} J(W,b;x_i,y_i) \right] + \lambda W_{ij}^l \tag{6}$$

$$\frac{\partial}{\partial b_i^l} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^l} J(W,b;x_i,y_i) \tag{7}$$

Processing a single sample

- Define local gradient for simplicity:

$$\delta_i^l = \frac{\partial}{\partial z_i^l} J(W, b; x, y) \tag{8}$$

- Partial derivatives with respect to weights:

$$\frac{\partial J(W, b; x, y)}{\partial W_{ij}^l} = \frac{\partial J(W, b; x, y)}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial W_{ij}^l} = \delta_{i+1} a_j^l \tag{9}$$

- Partial derivatives with respect to biases:

$$\frac{\partial J(W, b; x, y)}{\partial b_i^l} = \frac{\partial J(W, b; x, y)}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial b_i^l} = \delta_{i+1} \tag{10}$$

Single sample based gradient descend

1 Perform forward propagation to compute activations for all layers;
2 For each unit i in output layer L:

$$\delta_i^L = \frac{\partial}{\partial z_i^L}\frac{1}{2}\|y - h_{W,b}(x)\|^2 = (a_i^L - y_i)\cdot f'(z_i^L) \tag{11}$$

for $l = L-1, L-2, \cdots, 2$ **do**
3 | For each unit i:$\delta_i^l = (\sum_{j=1}^{n_{l+1}} W_{ji}^l \delta_j^{l+1}) f'(z_i^l)$;
4 **end**
5 Compute corresponding partial derivatives:

$$\frac{\partial}{\partial W_{ij}^l} J(W,b;x,y) = a_j^l \delta_i^{l+1} \tag{12}$$

$$\frac{\partial}{\partial b_i^l} J(W,b;x,y) = \delta_i^{l+1} \tag{13}$$

Vectorized Gradient Descend for single sample

Perform forward propagation to compute activations for all layers;
For output layer L:

$$\delta^L = (a^L - y) \bullet f'(z^L) \tag{14}$$

for $l = L-1, L-2, \cdots, 2$ **do**
  $\delta^l = ((W^l)^T \delta^{l+1}) \bullet f'(z^l)$;
end
Compute corresponding partial derivatives:

$$\nabla_{W^l} J(W, b; x, y) = \delta^{l+1} (a^l)^T \tag{15}$$

$$\nabla_{b^l} J(W, b; x, y) = \delta^{l+1} \tag{16}$$

;

Batch Gradient Descendent

1 Initialize weights and biases;

2 **repeat** // Batch Processing

3 Set $\triangle W^l = 0, \triangle b^l = 0$;

4 **for** $i = 1; i \leq m; i = i + 1$ **do** // Batch Gradient Descendent

5 a. Using algorithm above to compute $\nabla_{W^l} J(W, b; x, y), \nabla_{b^l} J(W, b; x, y)$;

6 b. Set $\triangle W^l = \triangle W^l + \nabla_{W^l} J(W, b; x, y)$;

7 c. Set $\triangle b^l = \triangle b^l + \nabla_{b^l} J(W, b; x, y)$;

8 **end**

9 Parameter Modification Quantity:

$$\triangle W^l = -\alpha \left[(\frac{1}{m} \triangle W^l) + \lambda W^l \right] \tag{2.22}$$

$$\triangle b^l = -\alpha \left[\frac{1}{m} \triangle b^l \right] \tag{2.23}$$

;

10 Update Parameters:

$$W^l = W^l + \triangle W^l \tag{2.24}$$

$$b^l = b^l + \triangle b^l \tag{2.25}$$

;

11 **until** *Convergence*;

Momentum

Momentum is used to accelerate learning procedure:

$$\triangle W^l(t+1) = \tau \triangle W^l(t) - \alpha \nabla_{W^l} J(W, b; x, y) \tag{17}$$

$\nabla_{W^l} J(W, b; x, y)$ is fixed here,a brief explanation for acceleration of momentum is provided below($\tau \in [0.9 \sim 1)$):

Sparsity Constraint

Average activation of hidden unit j in layer l:

$$\rho_j^l = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^l(x_i) \right] \tag{18}$$

KL-divergence is used to force $\rho_j^l = \rho$(ρ is a small sparsity parameter):

$$KL(\rho || \rho_j^l) = \rho \log \frac{\rho}{\rho_j^l} + (1-\rho) \log \frac{1-\rho}{1-\rho_j^l} \tag{19}$$

Add extra sparsity penalty term to achieve sparsity constraint:

$$\sum_{j=1}^{s_2} KL(\rho || \rho_j^l) = \sum_{j=1}^{s_l} \rho \log \frac{\rho}{\rho_j^l} + (1-\rho) \log \frac{1-\rho}{1-\rho_j^l} \tag{20}$$

Objective Function of Autoencoder

Our overall objective function is:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{l=2}^{L-1} \sum_{i=1}^{s_l} KL(\rho||\rho_i^l) \tag{21}$$

where $J(W,b)$ is defined as previous,β controls weight of sparsity penalty term,ρ_j^l implicitly depends on W,b.

Make small change to the algorithm above to incorporate KL-divergence term into derivative calculation:

$$\delta_i^l = \left(\left(\sum_{j=1}^{s_l} W_{ji}^l \delta_j^{l+1} \right) + \beta \left(-\frac{\rho}{\rho_i^l} + \frac{1-\rho}{1-\rho_i^l} \right) \right) f'(z_i^l) \tag{22}$$

Simple Derivation I

For simplicity,we divide the sparsity penalty term into small items:

$$S = \beta \sum_{l=2}^{L-1} S^l \tag{23}$$

where S^l is sparsity penalty term for layer l:

$$\begin{aligned} S^l &= \sum_{i=1}^{s_l} KL(\rho||\rho_i^l) \\ &= \sum_{i=1}^{s_l} \left[\rho(\log\rho - \log\rho_i^l) + (1-\rho)(\log(1-\rho) - \log(1-\rho_i^l))\right] \end{aligned} \tag{24}$$

It's apparent that $J_{sparse}(W,b) = J(W,b) + S$.

Simple Derivation II

Next,we have the following partial derivatives:

$$\frac{\partial S^l}{\partial \rho_i^l} = \frac{KL(\rho||\rho_i^l)}{\partial \rho_i^l} = -\frac{\rho}{\rho_i^l} + \frac{1-\rho}{1-\rho_i^l} \tag{25}$$

$$\frac{\partial \rho_i^l}{\partial z_i^l} = \frac{1}{m}\frac{\sum_{j=1}^m a_i^l(x_j)}{\partial z_i^l} = \frac{1}{m}\sum_{j=1}^m f'(z_i^l(x_j)) \tag{26}$$

$$\begin{aligned} \frac{\partial S}{\partial z_i^l} &= \frac{\partial S}{\partial S^l}\frac{\partial S^l}{\partial \rho_i^l}\frac{\partial \rho_i^l}{\partial z_i^l} \\ &= \beta\left(-\frac{\rho}{\rho_i^l} + \frac{1-\rho}{1-\rho_i^l}\right) f'(z_i^l) \end{aligned} \tag{27}$$

Simple Derivation III

Based on the derivations above,we have the following equation:

$$\begin{aligned}\frac{\partial J_{sparse}(W,b)}{\partial z_i^l} &= \frac{\partial J(W,b)}{\partial z_i^l} + \frac{\partial S}{\partial z_i^l} \\ &= \left(\left(\sum_{j=1}^{s_l} W_{ji}^l \delta_j^{l+1}\right) + \beta\left(-\frac{\rho}{\rho_i^l} + \frac{1-\rho}{1-\rho_i^l}\right)\right) f'(z_i^l)\end{aligned} \tag{28}$$

What does Autoencoder learn?

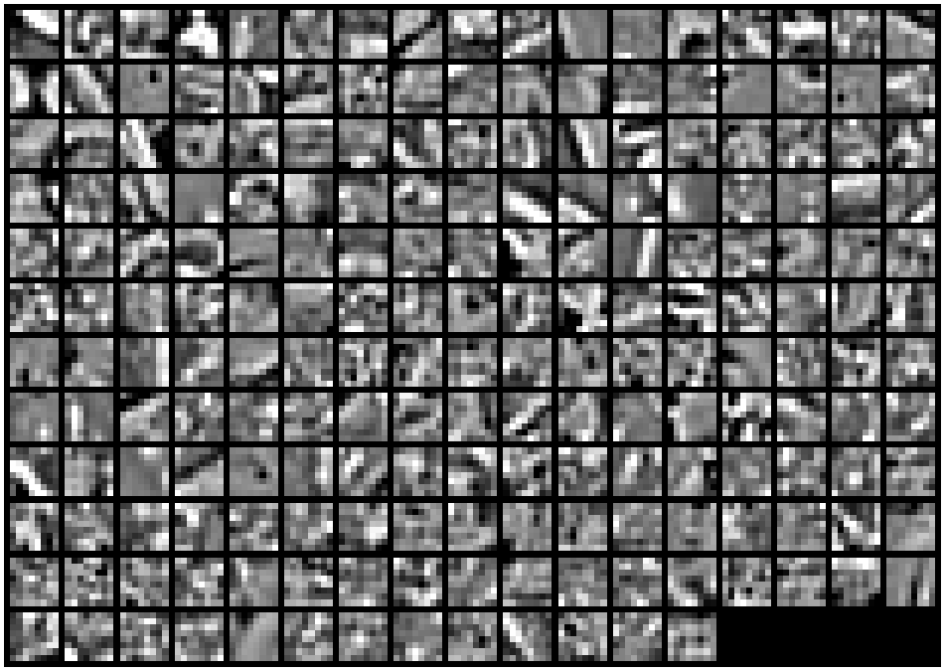

Figure: Training Examples

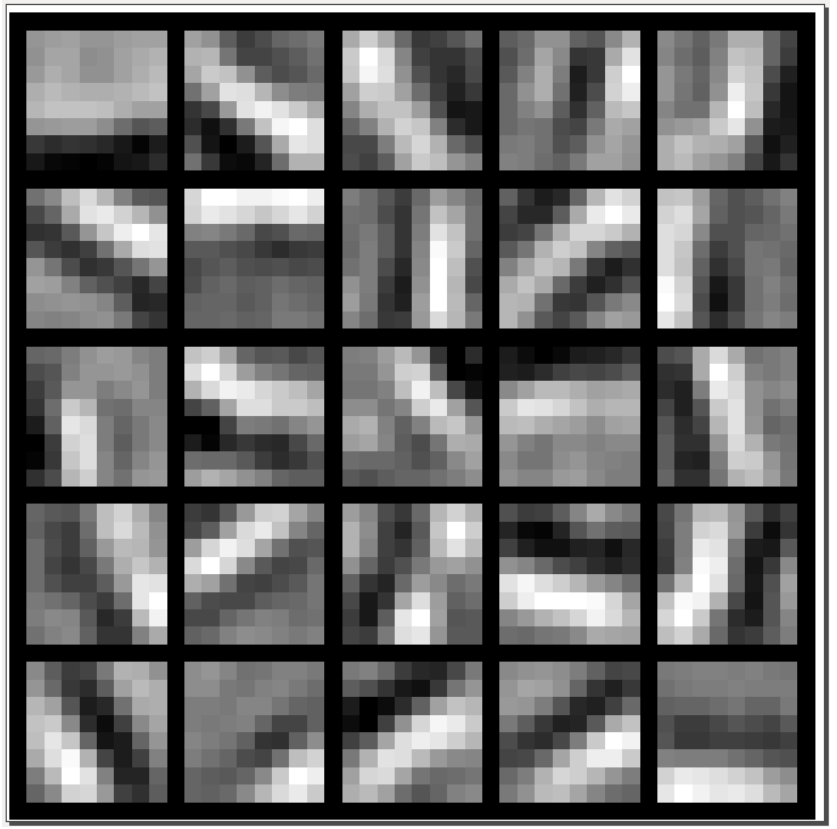

Figure: Learned Edges

Different hidden units have learned to detect edges at different positions and orientations in the images,which are quite useful for object recognition and other vision tasks.

Trying to run faster

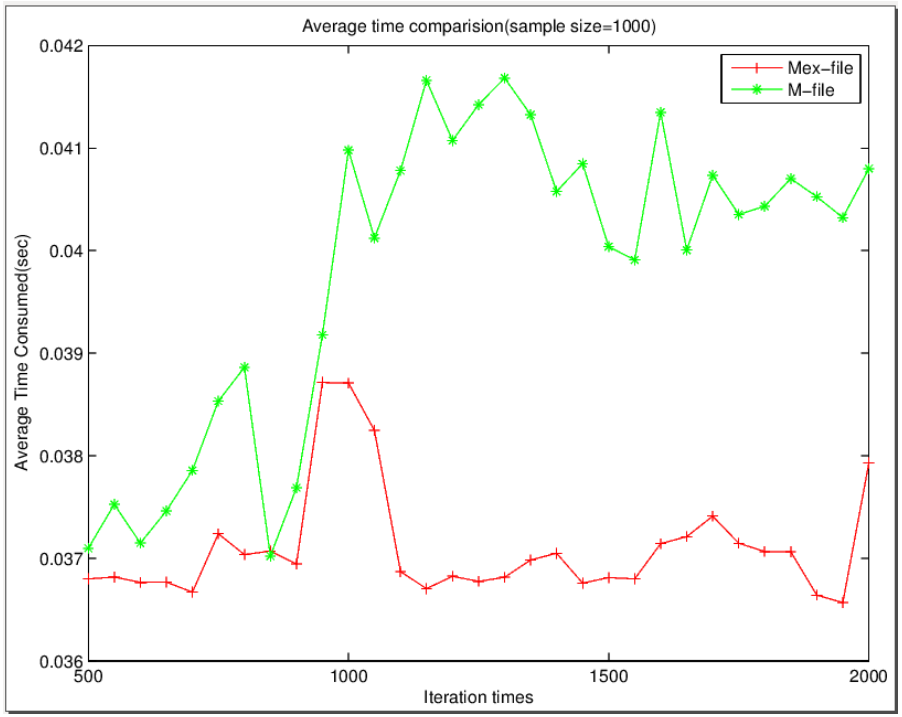

Figure: Average Time Consumed

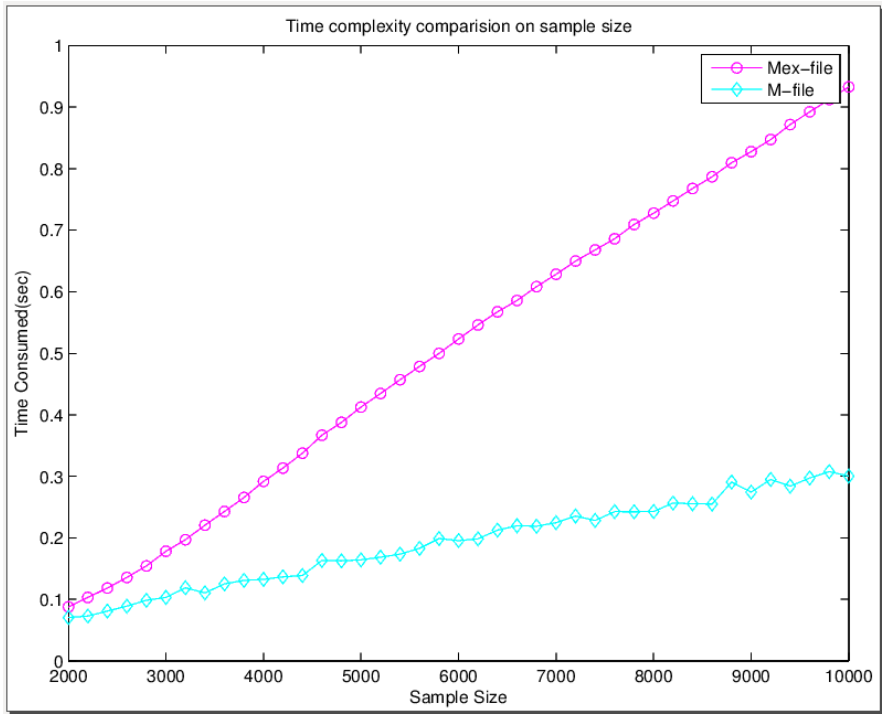

Figure: Time Consumed VS Sample Size