Neural Networks
Part 2

Yingyu Liang
yliang@cs.wisc.edu

Computer Sciences Department
University of Wisconsin, Madison

[Based on slides from Jerry Zhu, Mohit Gupta]
Limited power of one single neuron

- Perceptron: \( a = g(\sum_d w_d x_d) \)
- Activation function \( g \): linear, step, sigmoid
Limited power of one single neuron

- Perceptron: \( a = g(\sum_d w_d x_d) \)
- Activation function \( g \): linear, step, sigmoid
- Decision boundary linear even for nonlinear \( g \)
- XOR problem
Limited power of one single neuron

• XOR problem

• **Wait!** If one can represent AND, OR, NOT, one can represent any logic circuit (including XOR), by connecting them

Question: how to?
Multi-layer neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer
Multi-layer neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer

\[ a_1^{(2)} = g \left( \sum_d x_d w_{1d}^{(2)} \right) \]
Multi-layer neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer

\[
a_{1}^{(2)} = g \left( \sum_{d} x_{d}w_{1d}^{(2)} \right)
\]

\[
a_{2}^{(2)} = g \left( \sum_{d} x_{d}w_{2d}^{(2)} \right)
\]
Multi-layer neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer

\[
a_1^{(2)} = g \left( \sum_d x_d w_{1d}^{(2)} \right)
\]

\[
a_2^{(2)} = g \left( \sum_d x_d w_{2d}^{(2)} \right)
\]

\[
a_3^{(2)} = g \left( \sum_d x_d w_{3d}^{(2)} \right)
\]
Multi-layer neural networks

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer

\[ a_1^{(2)} = g \left( \sum_d x_d w_{1d}^{(2)} \right) \]

\[ a_2^{(2)} = g \left( \sum_d x_d w_{2d}^{(2)} \right) \]

\[ a_3^{(2)} = g \left( \sum_d x_d w_{3d}^{(2)} \right) \]

\[ a = g \left( \sum_i a_i^{(2)} w_i^{(3)} \right) \]
Neural net for $K$-way classification

- Use $K$ output units
- Training: encode a label $y$ by an indicator vector
  - class1=$(1,0,0,…,0)$, class2=$(0,1,0,…,0)$ etc.
- Test: choose the class corresponding to the largest output unit

\[
a_1 = g \left( \sum_i a_i^{(2)} w_{1i}^{(3)} \right)
\]
Neural net for $K$-way classification

- Use $K$ output units
- Training: encode a label $y$ by an indicator vector
  - class1=$(1,0,0,…,0)$, class2=$(0,1,0,…,0)$ etc.
- Test: choose the class corresponding to the largest output unit
The (unlimited) power of neural network

• In theory
  ▪ we don’t need too many layers:
  ▪ 1-hidden-layer net with *enough hidden units* can represent any continuous function of the inputs with arbitrary accuracy
  ▪ 2-hidden-layer net can even represent discontinuous functions
Learning in neural network

• Again we will minimize the error ($K$ outputs):

$$E = \frac{1}{2} \sum_{x \in D} E_x, \quad E_x = \|y - a\|^2 = \sum_{c=1}^{K} (a_c - y_c)^2$$

• $x$: one training point in the training set $D$
• $a_c$: the $c$-th output for the training point $x$
• $y_c$: the $c$-th element of the label indicator vector for $x$

```
\begin{bmatrix}
1 \\
0 \\
0 \\
\end{bmatrix} = y
```

```
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_K \\
\end{bmatrix}
```
Learning in neural network

• Again we will minimize the error ($K$ outputs):

$$E = \frac{1}{2} \sum_{x \in D} E_x, \quad E_x = \|y - a\|^2 = \sum_{c=1}^{K} (a_c - y_c)^2$$

• $x$: one training point in the training set $D$
• $a_c$: the $c$-th output for the training point $x$
• $y_c$: the $c$-th element of the label indicator vector for $x$
• Our variables are all the weights $w$ on all the edges
  ▪ Apparent difficulty: how to update the weights for the hidden units?
Learning in neural network

• Again we will minimize the error ($K$ outputs):

$$E = \frac{1}{2} \sum_{x \in D} E_x, \quad E_x = \|y - a\|^2 = \sum_{c=1}^{K} (a_c - y_c)^2$$

• $x$: one training point in the training set $D$
• $a_c$: the $c$-th output for the training point $x$
• $y_c$: the $c$-th element of the label indicator vector for $x$
• Our variables are all the weights $w$ on all the edges
  ▪ Apparent difficulty: how to update the weights for the hidden units?
  ▪ It turns out to be OK: we can still do gradient descent. The trick you need is the chain rule
  ▪ The algorithm is known as back-propagation
Gradient (on one data point)

want to compute \( \frac{\partial E_x}{\partial w_{11}^{(4)}} \)
Gradient (on one data point)

\[ \mathcal{E}_x = \|y - a\|^2 \]

\[ a_1 \xrightarrow{\text{softmax}} a_1 \]

\[ a_2 \xrightarrow{\text{softmax}} a_2 \]

\[ x_1 \rightarrow \text{Layer (1)} \rightarrow \text{Layer (2)} \rightarrow \text{Layer (3)} \rightarrow \text{Layer (4)} \]

\[ x_2 \rightarrow \text{Layer (1)} \rightarrow \text{Layer (2)} \rightarrow \text{Layer (3)} \rightarrow \text{Layer (4)} \]

\[ w_{11}^{(4)} \]

\[ y \rightarrow \text{Layer (1)} \rightarrow \text{Layer (2)} \rightarrow \text{Layer (3)} \rightarrow \text{Layer (4)} \]

\[ y \rightarrow \text{Layer (1)} \rightarrow \text{Layer (2)} \rightarrow \text{Layer (3)} \rightarrow \text{Layer (4)} \]
Gradient (on one data point)

\[ z_1^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

\[ E_x = \| y - a \|^2 \]
Gradient (on one data point)

\[ x_1, x_2 \]

Layer (1) \quad Layer (2) \quad Layer (3) \quad Layer (4)

\[ z_1^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

\[ E_x = \| y - a \|^2 \]
Gradient (on one data point)

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ z^{(4)} = w^{(4)}_{11} a^{(3)}_1 + w^{(4)}_{12} a^{(3)}_2 \]

\[ E_x = \|y - a\|^2 \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w^{(4)}_{11}} = \frac{\partial E_x}{\partial a_1} \frac{\partial a_1}{\partial z^{(4)}_1} \frac{\partial z^{(4)}_1}{\partial w^{(4)}_{11}} \]
Gradient (on one data point)

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ z^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

\[ E_x = \|y - a\|^2 \]

\[ \frac{\partial E_x}{\partial a_1} = 2(a_1 - y_1) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1) g'(z_1^{(4)}) \frac{\partial z_1^{(4)}}{\partial w_{11}^{(4)}} \]
Gradient (on one data point)

\[ E_x = \|y - a\|^2 \]

\[ z_1^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1)g'(z_1^{(4)})a_1^{(3)} \]
Gradient (on one data point)

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ x_1 \]
\[ x_2 \]

\[ z_1^{(4)} = w_1^{(4)} a_1^{(3)} + w_2^{(4)} a_2^{(3)} \]

\[ a_1 \]

\[ E_x = \| y - a \|^2 \]

\[ a_2 \]

\[ g \left( z_1^{(4)} \right) \]

\[ \frac{\partial a_1}{\partial z_1^{(4)}} = g'(z_1^{(4)}) \]

\[ \frac{\partial E_x}{\partial a_1} = 2(a_1 - y_1) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1)g \left( z_1^{(4)} \right) \left( 1 - g \left( z_1^{(4)} \right) \right) a_1^{(3)} \]
Gradient (on one data point)

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ z_1^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

\[ E_x = \|y - a\|^2 \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1) a_1 (1 - a_1) a_1^{(3)} \]
Gradient (on one data point)

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ z^{(4)} = w_{11}^{(4)} a_1^{(3)} + w_{12}^{(4)} a_2^{(3)} \]

\[ E_x = \| y - a \|^2 \]

\[
\frac{\partial a_1}{\partial z_1^{(4)}} = g'(z_1^{(4)})
\]

\[
\frac{\partial E_x}{\partial a_1} = 2(a_1 - y_1)
\]

By Chain Rule:

\[
\frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1)a_1(1 - a_1)a_1^{(3)}
\]

Can be computed by network activation
Backpropagation

Layer (1)  Layer (2)  Layer (3)  Layer (4)

$z^{(4)} = w^{(4)}_{11}a^{(3)} + w^{(4)}_{12}a^{(3)}$

$E_x = \|y - a\|^2$

$\frac{\partial E_x}{\partial z^{(4)}_1} = 2(a_1 - y_1)g'(z^{(4)}_1)$

By Chain Rule:

$\frac{\partial E_x}{\partial w^{(4)}_{11}} = 2(a_1 - y_1)a_1(1 - a_1)a^{(3)}_1$
Backpropagation

Layer (1) Layer (2) Layer (3) Layer (4)

\[ z^{(4)} = w^{(4)}_{11} a^{(3)}_1 + w^{(4)}_{12} a^{(3)}_2 \]

\[ E_x = \|y - a\|^2 \]

\[ \delta_1^{(4)} = \frac{\partial E_x}{\partial z_1^{(4)}} = 2(a_1 - y_1)g'(z_1^{(4)}) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w^{(4)}_{11}} = 2(a_1 - y_1)a_1(1 - a_1)a^{(3)}_1 \]
Backpropagation

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ y = \sum (w_{11} a_1 + w_{12} a_2) \]

\[ \delta_1^{(4)} = \frac{\partial E_x}{\partial z_1^{(4)}} = 2(a_1 - y_1) g'(z_1^{(4)}) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}} = \delta_1^{(4)} a_1^{(3)} \]
Backpropagation

\[ x_2 \]

\[ x_1 \]

Layer (1) Layer (2) Layer (3) Layer (4)

\[ w_{11}(4) a_1^{(3)} \]

\[ w_{12}(4) a_2^{(3)} \]

\[ a_1 \]

\[ a_2 \]

\[ w_1^{(4)} \]

\[ w_2^{(4)} \]

\[ E_x = ||y - a||^2 \]

\[ \delta_1^{(4)}(a_1^{(3)}) = \frac{\partial E_x}{\partial a_1^{(3)}} = 2(a_1 - y_1) g'(z_1^{(4)}) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = \delta_1^{(4)} a_1^{(3)} , \quad \frac{\partial E_x}{\partial w_{12}^{(4)}} = \delta_1^{(4)} a_2^{(3)} \]
Backpropagation

Layer (1)  Layer (2)  Layer (3)  Layer (4)

\[ \mathbf{x}_1 \]
\[ \mathbf{x}_2 \]

\[ \mathbf{a}_1^{(3)} \]
\[ \mathbf{a}_2^{(3)} \]

\[ \mathbf{w}_{11} \mathbf{a}_1^{(3)} \]
\[ \mathbf{w}_{12} \mathbf{a}_2^{(3)} \]

\[ \mathbf{z}_1^{(4)} \]

\[ \delta_2^{(4)} = \frac{\partial E_x}{\partial \mathbf{z}_2^{(4)}} = 2(\mathbf{a}_2 - \mathbf{y}_2)g'(\mathbf{z}_2^{(4)}) \]

By Chain Rule:

\[ \frac{\partial E_x}{\partial \mathbf{w}_{21}^{(4)}} = \delta_2^{(4)} \mathbf{a}_1^{(3)} \]
\[ \frac{\partial E_x}{\partial \mathbf{w}_{22}^{(4)}} = \delta_2^{(4)} \mathbf{a}_2^{(3)} \]

\[ E_x = \| \mathbf{y} - \mathbf{a} \|^2 \]
Backpropagation

Thus, for any weight in the network:

$$\frac{\partial E_x}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}$$

$$\delta_j^{(l)} : \delta \text{ of } j^{th} \text{ neuron in Layer } l$$

$$a_k^{(l-1)} : \text{Activation of } k^{th} \text{ neuron in Layer } l - 1$$

$$w_{jk}^{(l)} : \text{Weight from } k^{th} \text{ neuron in Layer } l - 1 \text{ to } j^{th} \text{ neuron in Layer } l$$

$$E_x = \|y - a\|^2$$
Exercise

Show that for any bias in the network:

\[
\frac{\partial E_x}{\partial b_j^{(l)}} = \delta_j^{(l)}
\]

\(\delta_j^{(l)}\) : \(\delta\) of \(j^{th}\) neuron in Layer \(l\)

\(b_j^{(l)}\) : bias for the \(j^{th}\) neuron in Layer \(l\), i.e., 
\[
z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}
\]
Backpropagation of $\delta$

Thus, for any neuron in the network:

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} g'(z_j^{(l)})$$

- $\delta_j^{(l)}$: $\delta$ of $j^{th}$ Neuron in Layer $l$
- $\delta_k^{(l+1)}$: $\delta$ of $k^{th}$ Neuron in Layer $l + 1$
- $g'(z_j^{(l)})$: derivative of $j^{th}$ Neuron in Layer $l$ w.r.t. its linear combination input
- $w_{kj}^{(l+1)}$: Weight from $j^{th}$ Neuron in Layer $l$ to $k^{th}$ Neuron in Layer $l + 1$

$E_x = \|y - a\|^2$
Gradient descent with Backpropagation

1. Initialize Network with Random Weights and Biases

2. For each Training Image:
   a. Compute Activations for the Entire Network
   b. Compute $\delta$ for Neurons in the Output Layer using Network Activation and Desired Activation
      \[ \delta_j^{(L)} = 2(y_j - a_j)a_j(1 - a_j) \]
   c. Compute $\delta$ for all Neurons in the previous Layers
      \[ \delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} a_k^{(l)} (1 - a_j^{(l)}) \]
   d. Compute Gradient of Cost w.r.t each Weight and Bias for the Training Image using $\delta$
      \[ \frac{\partial E_x}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)} \quad \frac{\partial E_x}{\partial b_j^{(l)}} = \delta_j^{(l)} \]
Gradient descent with Backpropagation

3. Average the Gradient w.r.t. each Weight and Bias over the Entire Training Set

\[ \frac{\partial E}{\partial w_{jk}^{(l)}} = \frac{1}{n} \sum \frac{\partial E_x}{\partial w_{jk}^{(l)}} \quad \frac{\partial E}{\partial b_j^{(l)}} = \frac{1}{n} \sum \frac{\partial E_x}{\partial b_j^{(l)}} \]

4. Update the Weights and Biases using Gradient Descent

\[ w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{\partial E}{\partial w_{jk}^{(l)}} \quad b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial E}{\partial b_j^{(l)}} \]

5. Repeat Steps 2-4 till Cost reduces below an acceptable level