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

Layer 1 (input)

Layer 2 (hidden)

Layer 3 (output)

\[ x_1 \]

\[ x_2 \]
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 $x$
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: we don’t know the ‘correct’ output of 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: we don’t know the ‘correct’ output of 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)

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

$$\nabla E_x$$

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

\[
\begin{align*}
\tilde{E}_x &= \| y - a \|^2 \\
&= \frac{1}{2} (y - a)^T (y - a) \\
&= \frac{1}{2} y^T y - ay - ya + a^T a \\
&= \frac{1}{2} y^T y - 2ay + a^T a \\
&= \frac{1}{2} y^T y - 2ay + a^T a \\
&= \frac{1}{2} y^T y - 2ay + a^T a \\
&= \frac{1}{2} y^T y - 2ay + a^T a \\
&= \frac{1}{2} y^T y - 2ay + a^T a
\end{align*}
\]

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

\[
\begin{align*}
x_1 &\rightarrow w_{11} \rightarrow a_1 \\
x_2 &\rightarrow w_{11} \rightarrow a_2 \\
\end{align*}
\]
Gradient (on one data point)

\[ x_1^2 x_1^1 = y - a_2^2 a_2^1 \]

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

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

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

\[ x_1 \]  \[ \rightarrow \]  \[ \rightarrow \]  \[ \rightarrow \]  \[ a_1 \]

\[ x_2 \]  \[ \rightarrow \]  \[ \rightarrow \]  \[ \rightarrow \]  \[ a_2 \]

\[ z_1^{(4)} \]  \[ \rightarrow \]  \[ g(z_1^{(4)}) \]  \[ \rightarrow \]  \[ \| y - a \|^2 \]  \[ \rightarrow \]  \[ E_x \]
Gradient (on one data point)

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

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

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

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

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

\[ x_1 \rightarrow w_{11}^{(4)} \rightarrow z_1^{(4)} \rightarrow a_1 \]
\[ x_2 \rightarrow \]

\[ w_{11}^{(4)} a_1^{(3)} \rightarrow + \rightarrow z_1^{(4)} \rightarrow g \left( z_1^{(4)} \right) \rightarrow a_1 \rightarrow \]

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

\[ a_2 \rightarrow \]

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

\[ by Chain Rule: \quad \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1)g' \left( z_1^{(4)} \right) \frac{\partial z_1^{(4)}}{\partial w_{11}^{(4)}} \]
**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:

\[ \frac{\partial E_x}{\partial w_{11}^{(4)}} = 2(a_1 - y_1)g'(z_1^{(4)})a_1^{(3)} \]
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:

\[ \frac{\partial E_{x}}{\partial w_{11}^{(4)}} = 2(a_{1} - y_{1})g(z_{1}^{(4)}) \left( 1 - g(z_{1}^{(4)}) \right) a_{1}^{(3)} \]
Gradient (on one data point)

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

\[ x_1 \]
\[ x_2 \]

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

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

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

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)

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

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

By Chain Rule:

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

Can be computed by network activation
### Backpropagation

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

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

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

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

By Chain Rule:

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

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

\[ x_1 \]
\[ x_2 \]

\[ w^{(4)}_{11} a_1^{(3)} \]
\[ w^{(4)}_{12} a_2^{(3)} \]

\[ z^{(4)}_1 = w^{(4)}_{11} a_1^{(3)} + w^{(4)}_{12} a_2^{(3)} \]

\[ a_1 \]
\[ a_2 \]

\[ w^{(4)}_{11} \]

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

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

By Chain Rule:

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

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

By Chain Rule:

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

\[ x_2 \]

\[ x_1 \]

Layer (1) \hspace{1cm} Layer (2) \hspace{1cm} Layer (3) \hspace{1cm} Layer (4)

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

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

By Chain Rule:

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

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

\[
\begin{align*}
E_x &= \|y - a\|^2 \\
\delta_2^{(4)} &= \frac{\partial E_x}{\partial z_2^{(4)}} = 2(a_2 - y_2)g'(z_2^{(4)}) \\
\frac{\partial E_x}{\partial w_{21}^{(4)}} &= \delta_2^{(4)}a_1^{(3)} \\
\frac{\partial E_x}{\partial w_{22}^{(4)}} &= \delta_2^{(4)}a_2^{(3)}
\end{align*}
\]

By Chain Rule:
Backpropagation

Thus, for any weight in the network:

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

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

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

\(w_{jk}^{(l)}\): Weight from \(k^{th}\) neuron in Layer \(l - 1\) to \(j^{th}\) 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., 
\[z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\]

\[E_x = \|y - a\|^2\]
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 $l + 1$
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
      \[
      \delta_j^{(l)} = \sum_k \delta_k^{(l+1)}w_{kj}^{(l+1)}a_j^{(l)}(1 - a_j^{(l)})
      \]
   d. Compute Gradient of Cost w.r.t each Weight and Bias for the Training Image using $\delta$
      \[
      \frac{\partial E_x}{\partial w_{jk}^{(l)}} = \delta_j^{(l)}a_k^{(l-1)} \\
      \frac{\partial E_x}{\partial b_j^{(l)}} = \delta_j^{(l)}
      \]
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