Neural Networks II

Chen Gao
Virginia Tech
Neural Networks

• Origins: Algorithms that try to mimic the brain.

What is this?
A single neuron in the brain

- Input
- Output

- Dendrite
- Axon
- Cell body
- Axon terminal
- Schwann cell
- Node of Ranvier
- Myelin sheath
- Nucleus

Slide credit: Andrew Ng
An artificial neuron: Logistic unit

\[ x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \]

\[ h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \]

- Sigmoid (logistic) activation function

Slide credit: Andrew Ng
Visualization of weights, bias, activation function

bias $b$ only change the position of the hyperplane

The range is determined by $g(.)$
Activation - sigmoid

- Squashes the neuron’s pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

\[ g(x) = \frac{1}{1 + e^{-x}} \]

Slide credit: Hugo Larochelle
Activation - hyperbolic tangent (tanh)

- Squashes the neuron’s pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Slide credit: Hugo Larochelle
Activation - rectified linear (relu)

- Bounded below by 0
- always non-negative
- Not upper bounded
- Tends to give neurons with sparse activities

\[ g(x) = \text{relu}(x) = \max(0, x) \]
Activation - softmax

• For multi-class classification:
  • we need multiple outputs (1 output per class)
  • we would like to estimate the conditional probability \( p(y = c \mid x) \)

• We use the softmax activation function at the output:

\[
g(x) = \text{softmax}(x) = \left[ \frac{e^{x_1}}{\sum_c e^{x_c}} \ldots \frac{e^{x_c}}{\sum_c e^{x_c}} \right]
\]
Universal approximation theorem

“a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

Hornik, 1991
Neural network – Multilayer

Layer 1

Layer 2 (hidden)

Layer 3

$h_{θ}(x)$

Slide credit: Andrew Ng
Neural network

\[ a_i^{(j)} = \text{“activation” of unit } i \text{ in layer } j \]

\[ \theta^{(j)} = \text{matrix of weights controlling function mapping from layer } j \text{ to layer } j + 1 \]

Size of \( \theta^{(j)} \)?

\[ s_j \text{ unit in layer } j \]

\[ s_{j+1} \text{ units in layer } j + 1 \]

\[ a_1^{(2)} = g \left( \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 \right) \]

\[ a_2^{(2)} = g \left( \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 \right) \]

\[ a_3^{(2)} = g \left( \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 \right) \]

\[ h_\theta(x) = g \left( \theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(1)} a_1^{(2)} + \theta_{12}^{(1)} a_2^{(2)} + \theta_{13}^{(1)} a_3^{(2)} \right) \]
Neural network

Why do we need $g(.)$?

Slide credit: Andrew Ng

$$a_1^{(2)} = g \left( \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right) = g(z_1^{(2)})$$

$$a_2^{(2)} = g \left( \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right) = g(z_2^{(2)})$$

$$a_3^{(2)} = g \left( \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right) = g(z_3^{(2)})$$

$$h_\Theta(x) = g \left( \Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(1)} a_1^{(2)} + \Theta_{12}^{(1)} a_2^{(2)} + \Theta_{13}^{(1)} a_3^{(2)} \right) = g(z^{(3)})$$
Neural network

\[ h_\Theta(x) = g(z^{(3)}) \]

\[
\begin{align*}
a^{(2)}_1 &= g(z^{(2)}_1) \\
a^{(2)}_2 &= g(z^{(2)}_2) \\
a^{(2)}_3 &= g(z^{(2)}_3) \\
h_\Theta(x) &= g(z^{(3)})
\end{align*}
\]

\[
x = \begin{bmatrix} x_0 \\
x_1 \\
x_2 \\
x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z^{(2)}_1 \\
z^{(2)}_2 \\
z^{(2)}_3 \end{bmatrix}
\]

\[
\begin{align*}
z^{(2)} &= \Theta^{(1)} x = \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \\
\text{Add } a^{(2)}_0 &= 1 \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
h_\Theta(x) &= a^{(3)} = g(z^{(3)})
\end{align*}
\]
Flow graph - Forward propagation

\[ z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g(z^{(2)}) \]
Add \( a_0^{(2)} = 1 \)
\[ z^{(3)} = \Theta^{(2)} a^{(2)} \]
\[ h_\Theta(x) = a^{(3)} = g(z^{(3)}) \]

How do we evaluate our prediction?
Cost function

Logistic regression:

\[
J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2
\]

Neural network:

\[
h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}
\]

\[
J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right]
\]

\[
+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{s_l=1}^{s_{l+1}} \sum_{j=1}^{(s_{l+1} - s_l)} (\Theta_{j}^{(l)})^2
\]
Gradient computation

\[
J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} \sum_{k=1}^{K} y_{ik} \log(h_\Theta(x^{(i)}))_k + (1 - y_{ik}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] \\
+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{s_l}^{s_{l+1}} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2
\]

\[\min_{\Theta} J(\Theta)\]

Need to compute:

\[
\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)
\]
Gradient computation

Given one training example \((x, y)\)

\[
\begin{align*}
    a^{(1)} &= x \\
    z^{(2)} &= \Theta^{(1)} a^{(1)} \\
    a^{(2)} &= g(z^{(2)}) \text{ (add } a_0^{(2)} \text{)} \\
    z^{(3)} &= \Theta^{(2)} a^{(2)} \\
    a^{(3)} &= g(z^{(3)}) \text{ (add } a_0^{(3)} \text{)} \\
    z^{(4)} &= \Theta^{(3)} a^{(3)} \\
    a^{(4)} &= g(z^{(4)}) = h_{\Theta}(x)
\end{align*}
\]
Gradient computation: Backpropagation

Intuition: $\delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l$

For each output unit (layer $L = 4$)

$$
\delta^{(4)} = a^{(4)} - y
$$

$$
\delta^{(3)} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial z^{(3)}} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}}
$$

$$
= 1 * \Theta^{(3)^T} \delta^{(4)} .* g'(z^{(4)}) .* g'(z^{(3)})
$$

$$
\begin{align*}
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= g(z^{(4)})
\end{align*}
$$

Slide credit: Andrew Ng
Backpropagation algorithm

Training set \{ (x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)}) \}

Set \( \Theta^{(1)} = 0 \)

For \( i = 1 \) to \( m \)

Set \( a^{(1)} = x \)

Perform forward propagation to compute \( a^{(l)} \) for \( l = 2 \ldots L \)

use \( y^{(i)} \) to compute \( \delta^{(L)} = a^{(L)} - y^{(i)} \)

Compute \( \delta^{(L-1)}, \delta^{(L-2)} \ldots \delta^{(2)} \)

\( \Theta^{(l)} = \Theta^{(l)} - a^{(l)} \delta^{(l+1)} \)
Activation - sigmoid

• Partial derivative

\[ g'(x) = g(x) (1 - g(x)) \]

\[ g(x) = \frac{1}{1 + e^{-x}} \]
Activation - hyperbolic tangent (tanh)

• Partial derivative

$$g'(x) = 1 - g(x)^2$$

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
Activation - rectified linear (relu)

- Partial derivative

\[ g'(x) = 1_{x > 0} \]

\[ g(x) = \text{relu}(x) = \max(0, x) \]
Initialization

• For bias
  • Initialize all to 0

• For weights
  • Can’t initialize all weights to the same value
    • we can show that all hidden units in a layer will always behave the same
    • need to break symmetry
  • Recipe: $U[-b, b]$
    • the idea is to sample around 0 but break symmetry
Putting it together

Pick a network architecture

• No. of input units: Dimension of features
• No. output units: Number of classes
• Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)
• Grid search

Slide credit: Hugo Larochelle
Putting it together

Early stopping

• Use a validation set performance to select the best configuration
• To select the number of epochs, stop training when validation set error increases
Other tricks of the trade

• Normalizing your (real-valued) data
• Decaying the learning rate
  • as we get closer to the optimum, makes sense to take smaller update steps
• mini-batch
  • can give a more accurate estimate of the risk gradient
• Momentum
  • can use an exponential average of previous gradients
Dropout

• Idea: «cripple» neural network by removing hidden units
  • each hidden unit is set to 0 with probability 0.5
  • hidden units cannot co-adapt to other units
  • hidden units must be more generally useful