

# DS 5220

## Supervised Machine Learning and Learning Theory

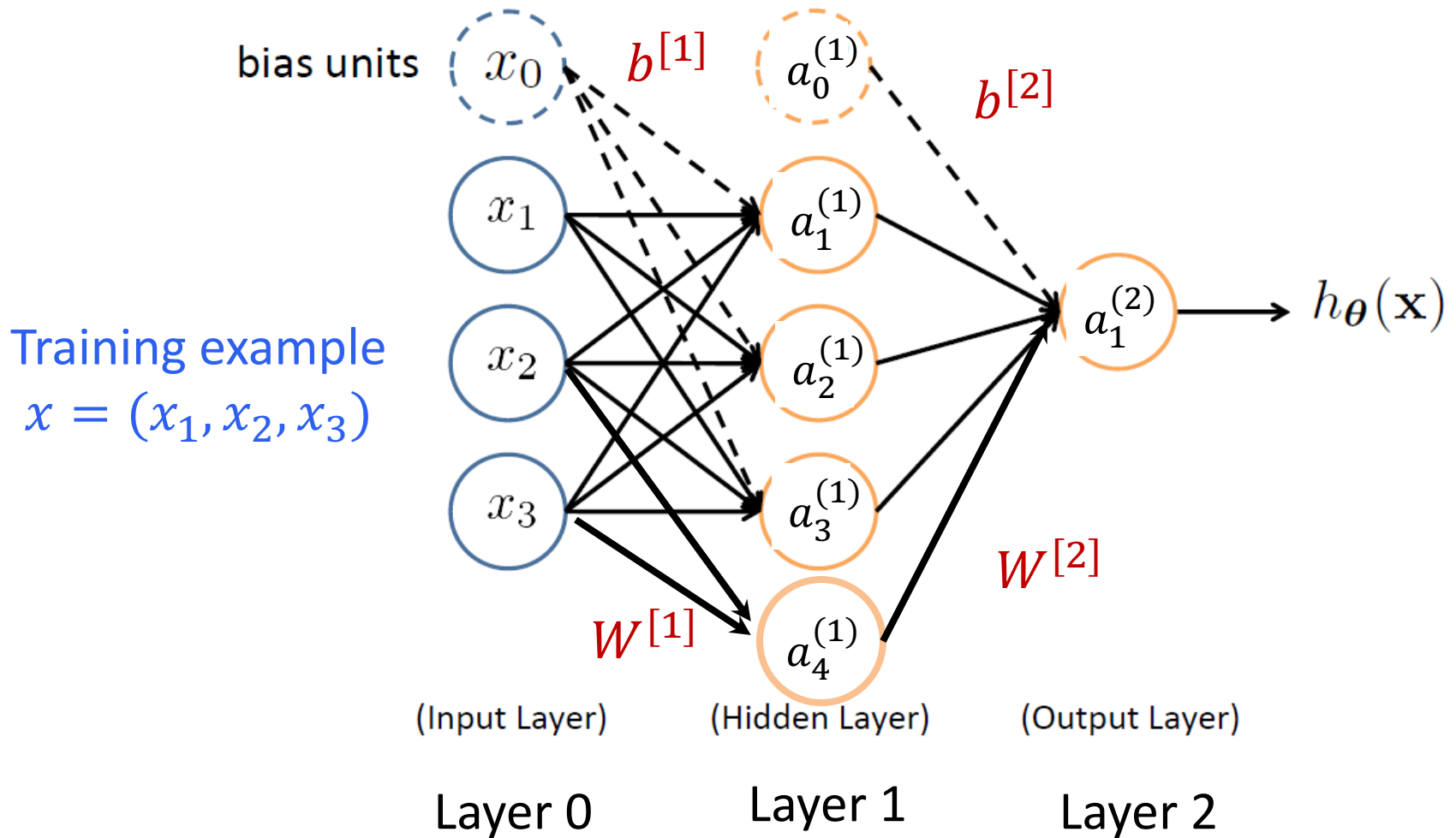
Alina Oprea  
Associate Professor, CCIS  
Northeastern University

November 20 2019

# Outline

- Training with backpropagation
  - Initialization
  - Derivation of gradients
  - Example
- Stochastic gradient descent and variants

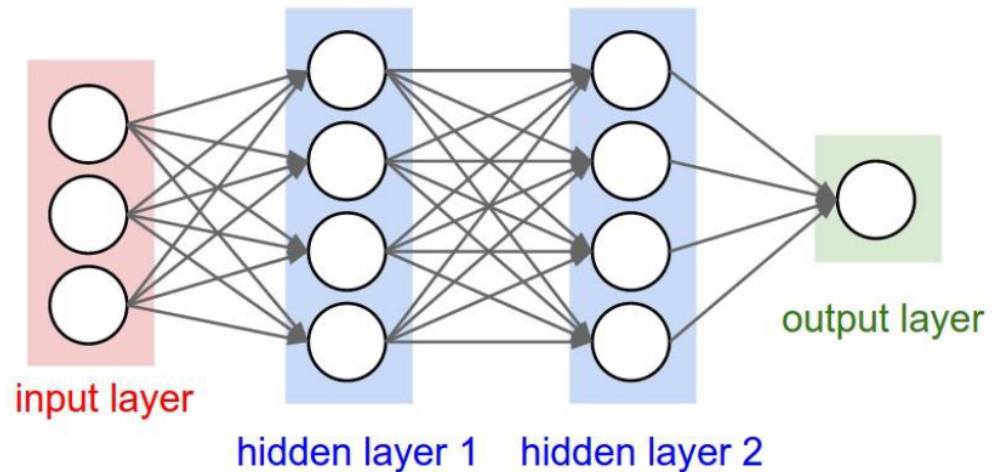
# Feed-Forward Neural Network



No cycles

# Forward Propagation

- The input neurons first receive the data features of the object. After processing the data, they send their output to the first hidden layer.
- The hidden layer processes this output and sends the results to the next hidden layer.
- This continues until the data reaches the final output layer, where the output value determines the object's classification.
- This entire process is known as **Forward Propagation**, or **Forward prop.**



# The Perceptron

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance  $(x_i, y_i)$ ,

$$\theta_j \leftarrow \theta_j - \frac{1}{2} (h_{\theta}(x_i) - y_i)x_{ij}$$

either 2 or -2

- If the prediction matches the label, make no change
- Otherwise, adjust  $\theta$

# Online Perceptron

Let  $\theta \leftarrow [0,0,\dots,0]$

Repeat:

Receive training example  $(x_i, y_i)$

If  $y_i \theta^T x_i \leq 0$  // prediction is incorrect

$\theta \leftarrow \theta + y_i x_i$

**Online learning** – the learning mode where the model update is performed each time a single observation is received

**Batch learning** – the learning mode where the model update is performed after observing the entire training set

# Batch Perceptron

Given training data  $\{(x_i, y_i)\}_{i=1}^n$

Let  $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

Let  $\Delta \leftarrow [0, 0, \dots, 0]$

for  $i = 1 \dots n$ , do

if  $y_i \theta^T x_i \leq 0$  // prediction for  $i^{th}$  instance is incorrect

$\Delta \leftarrow \Delta + y_i x_i$

$\Delta \leftarrow \Delta / n$  // compute average update

$\theta \leftarrow \theta + \alpha \Delta$

Until  $\|\Delta\|_2 < \epsilon$

- Simplest case:  $\alpha = 1$  and don't normalize, yields the fixed increment perceptron
- Each increment of outer loop is called an **epoch**
- An epoch includes a pass over the entire training data

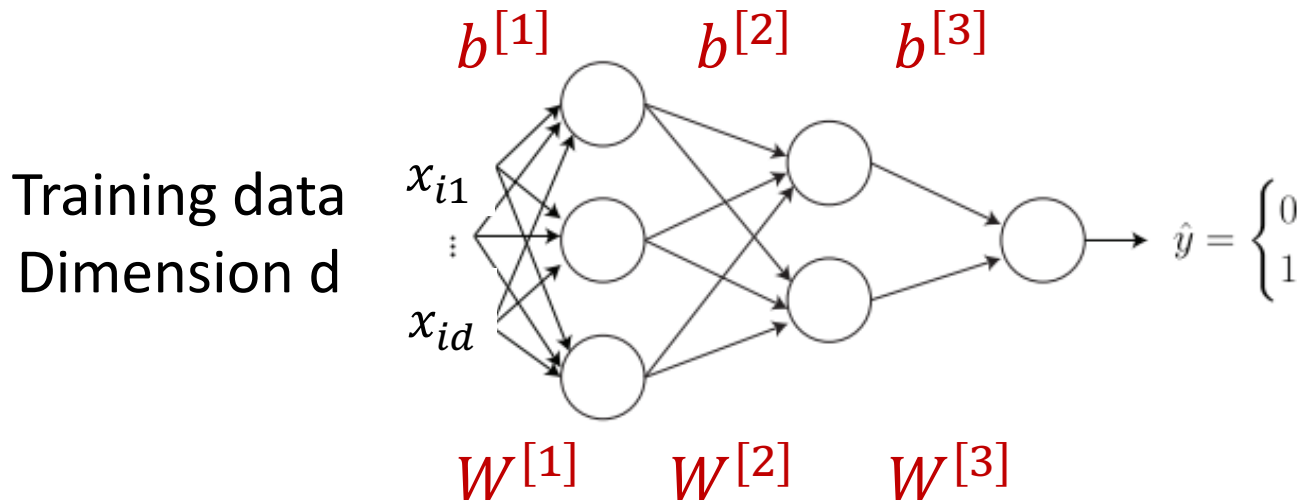
# Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

Error at last layer can be measured, but it is challenging to determine error at intermediate hidden layers



# Example



$$z^{[1]} = W^{[1]} x_i + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = g(z^{[2]})$$

$$z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$$

$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]})$$

# Parameter Initialization

- How about we set all  $W$  and  $b$  to 0?
- First layer
  - $z^{[1]} = W^{[1]}x + b^{[1]} = (0, \dots, 0)$
  - $a^{[1]} = g(z^{[1]}) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Second layer
  - $z^{[2]} = W^{[2]}x + b^{[2]} = (0, \dots, 0)$
  - $a^{[2]} = g(z^{[2]}) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Third layer
  - $z^{[3]} = W^{[3]}x + b^{[3]} = (0, \dots, 0)$
  - $a^{[3]} = g(z^{[3]}) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$  does not depend on  $x$
- **Initialize with random values instead!**

# Training

- Training data  $x_1, y_1, \dots, x_N, y_N$
- One training example  $x_i = (x_{i1}, \dots, x_{id})$ , label  $y_i$
- One forward pass through the network
  - Compute prediction  $\hat{y}_i = h(x_i)$
- Loss function for one example
  - $L(\hat{y}, y) = -[(1 - y) \log(1 - \hat{y}) + y \log \hat{y}]$

Cross-entropy loss

- Loss function for training data
  - $J(W, b) = \frac{1}{N} \sum_i L(\hat{y}_i, y_i) + \lambda R(W, b)$

# Reminder: Logistic Regression

$$J(\theta) = - \sum_{i=1}^N [y_i \log h_{\theta}(x_i) + (1 - y_i) \log (1 - h_{\theta}(x_i))]$$

- Cost of a single instance:

$$\text{cost}(h_{\theta}(\mathbf{x}), y) = \begin{cases} -\log(h_{\theta}(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(\mathbf{x})) & \text{if } y = 0 \end{cases}$$

- Can re-write objective function as

$$J(\theta) = \sum_{i=1}^n \underbrace{\text{cost}(h_{\theta}(x_i), y_i)}_{\text{Cross-entropy loss}}$$

Cross-entropy loss

# Gradient Descent

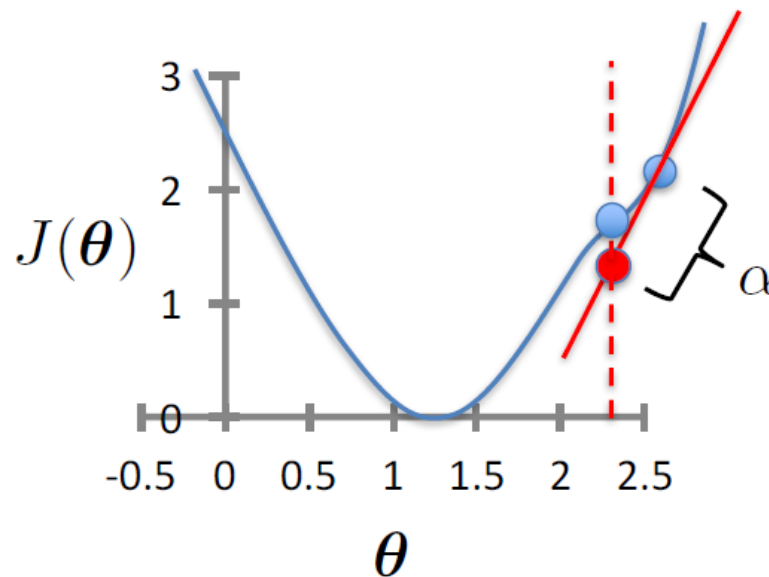
- Initialize  $\theta$
- Repeat until convergence

$$\theta = (W, b)$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

simultaneous update  
for  $j = 0 \dots d$

learning rate (small)  
e.g.,  $\alpha = 0.05$



- Converges for convex objective
- Could get stuck in local minimum for non-convex objectives

# GD for Neural Networks

- Initialization

- For all layers  $\ell$

- Set  $W^{[\ell]}, b^{[\ell]}$  at random

- Backpropagation

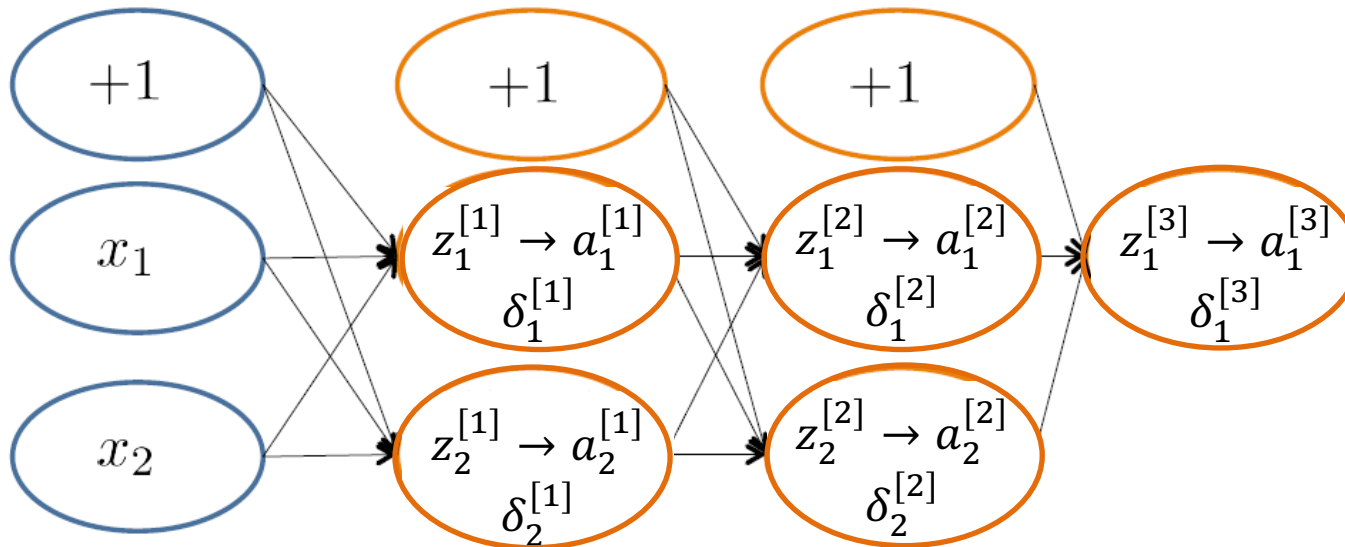
- Fix learning rate  $\alpha$

- For all layers  $\ell$  (starting backwards)

- $$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^N \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$$

- $$b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^N \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$$

# Backpropagation Intuition

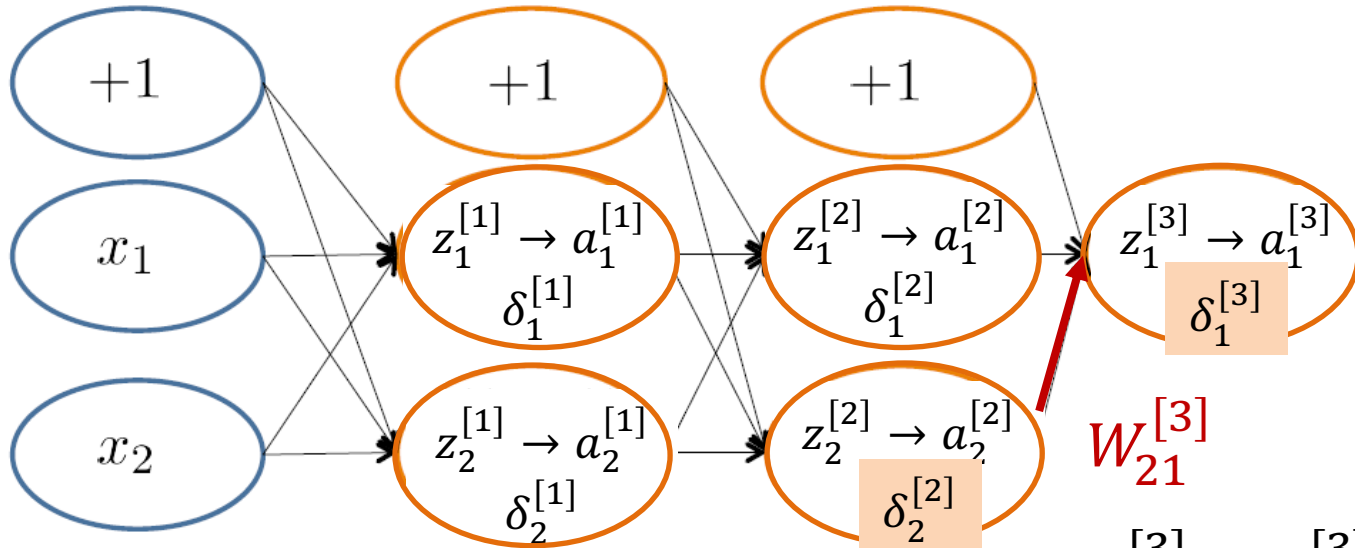


$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

Formally, 
$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$$

where  $\text{cost}(\mathbf{x}_i) = y_i \log h_{\Theta}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\Theta}(\mathbf{x}_i))$

# Backpropagation Intuition



$$\delta_1^{[3]} \approx a_1^{[3]} - y$$

$$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$$

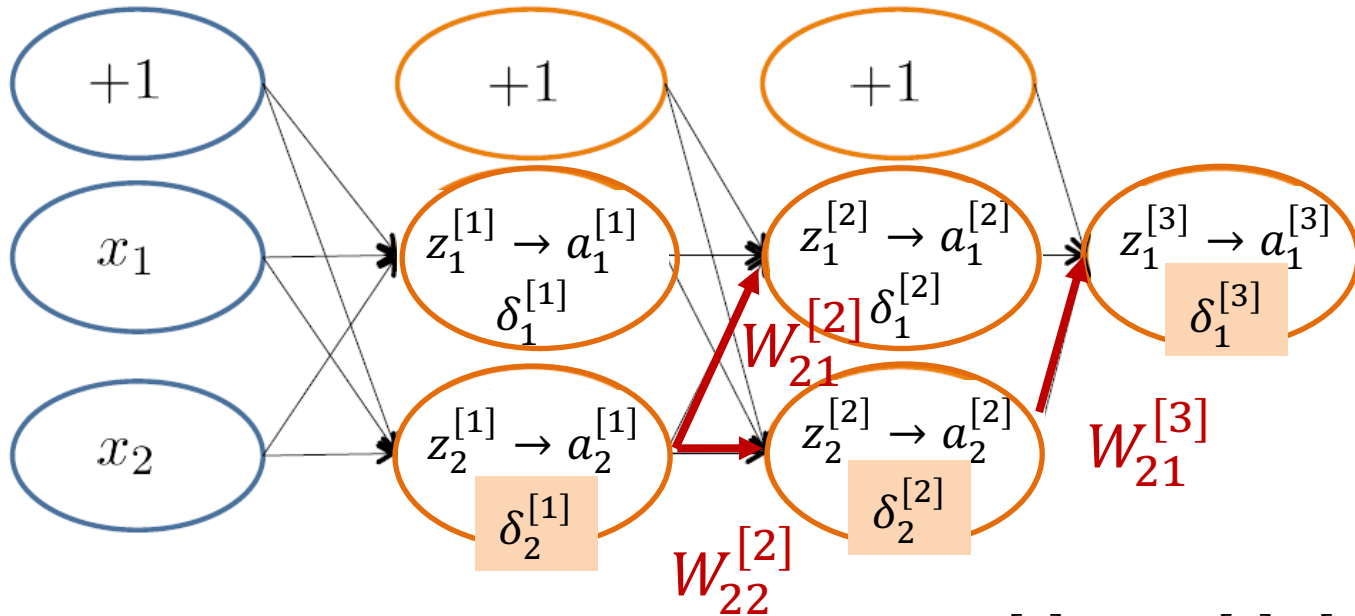
$\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

Formally, 
$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$$

where  $\text{cost}(\mathbf{x}_i) = y_i \log h_{\Theta}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\Theta}(\mathbf{x}_i))$



# Backpropagation Intuition



$$\delta_2^{[1]} \approx W_{21}^{[2]} \delta_1^{[2]} + W_{22}^{[2]} \delta_2^{[2]}$$

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

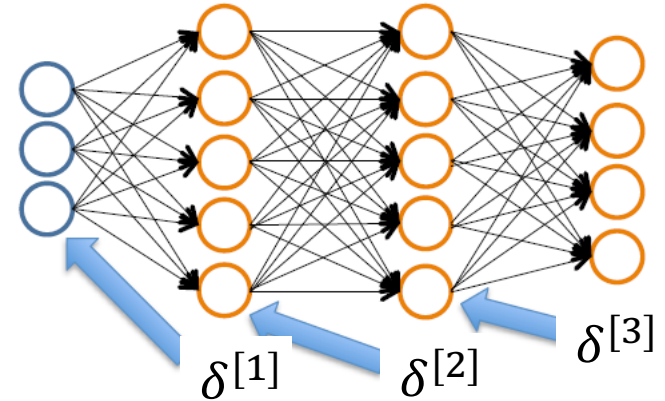
Formally, 
$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$$

where  $\text{cost}(\mathbf{x}_i) = y_i \log h_{\Theta}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\Theta}(\mathbf{x}_i))$

# Backpropagation

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

$$L(y, \hat{y}) = -[(1 - y) \log(1 - \hat{y}) + y \log \hat{y}]$$



## Definitions

$$- z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g(z^{[\ell]})$$

$$- \delta^{[\ell]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell]}}; \text{ Output } \hat{y} = a^{[L]} = g(z^{[L]})$$

$$1. \text{ For last layer } L: \delta^{[L]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} g'(z^{[L]})$$

$$2. \text{ For layer } \ell: \delta^{[\ell]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell]}} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell+1]}} \frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = \delta^{[\ell+1]} W^{[\ell+1]} g'(z^{[\ell]})$$

3. Compute parameter gradients

$$- \frac{\partial L(\hat{y}, y)}{\partial W^{[\ell]}} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial W^{[\ell]}} = \delta^{[\ell]} a^{[\ell-1]T}$$

$$- \frac{\partial L(\hat{y}, y)}{\partial b^{[\ell]}} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = \delta^{[\ell]}$$

# Binary Classification Example

- $\delta^{[3]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[3]}} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} g'(z^{[3]}); \hat{y} = g(z^{[3]}) = a^{[3]}$
- $\frac{\partial L(\hat{y}, y)}{\partial \hat{y}} = - \frac{\partial [(1-y) \log(1-\hat{y}) + y \log \hat{y}]}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}$
- $\delta^{[3]} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} g'(z^{[3]})$   
 $= \frac{a^{[3]}-y}{g(z^{[3]})(1-g(z^{[3]}))} g(z^{[3]}) (1 - g(z^{[3]})) = a^{[3]} - y$
- $\frac{\partial L(\hat{y}, y)}{\partial w^{[3]}} = \delta^{[3]} a^{[2]T} = (a^{[3]} - y) a^{[2]T}$
- $\frac{\partial L(\hat{y}, y)}{\partial b^{[3]}} = a^{[3]} - y$

$$g(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$
$$g'(x) = \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

# Binary Classification Example

- $\delta^{[2]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[2]}} = \delta^{[3]} W^{[3]} g'(z^{[2]})$
- $\frac{\partial L(\hat{y}, y)}{\partial W^{[2]}} = \delta^{[2]} a^{[1]T} = \delta^{[3]} W^{[3]} g'(z^{[2]}) a^{[1]T} =$   
 $= [a^{[3]} - y] W^{[3]} g(z^{[2]}) (1 - g(z^{[2]})) a^{[1]T}$
- $\frac{\partial L(\hat{y}, y)}{\partial b^{[2]}} = [a^{[3]} - y] W^{[3]} g(z^{[2]}) (1 - g(z^{[2]}))$

$$g(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$
$$g'(x) = \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

# Backpropagation

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$  (Used to accumulate gradient)

For each training instance  $(\mathbf{x}_i, y_i)$ :

Set  $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation

Compute  $\delta^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\delta^{(L-1)}, \dots, \delta^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Average gradient is  $\frac{\Delta_{ij}^{[\ell]}}{N}$

# Training NN with Backpropagation

Given training set  $(x_1, y_1), \dots, (x_N, y_N)$

Initialize all parameters  $W^{[\ell]}, b^{[\ell]}$  randomly, for all layers  $\ell$

Loop

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$  (Used to accumulate gradient)

For each training instance  $(\mathbf{x}_i, y_i)$ :

Set  $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation **EPOCH**

Compute  $\delta^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\delta^{(L-1)}, \dots, \delta^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \frac{\Delta_{ij}^{[\ell]}}{N}$
- Similar for  $b_{ij}^{[\ell]}$

Until weights converge or maximum number of epochs is reached

# Materials

- Stanford tutorial on training Multi-Layer Neural Networks
  - <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>
- Notes on backpropagation by Andrew Ng
  - <http://cs229.stanford.edu/notes/cs229-notes-backprop.pdf>
- Deep learning notes by Andrew Ng
  - [http://cs229.stanford.edu/notes/cs229-notes-deep\\_learning.pdf](http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf)

# GD for Neural Networks

- Initialization

- For all layers  $\ell$ 
  - Set  $W^{[\ell]}, b^{[\ell]}$  at random

- Backpropagation

- Fix learning rate  $\alpha$
- For all layers  $\ell$  (starting backwards)

- $$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^N \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$$

- $$b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^N \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$$

This is  
expensive!



# Stochastic Gradient Descent

- Initialization

- For all layers  $\ell$ 
  - Set  $W^{[\ell]}, b^{[\ell]}$  at random

- Backpropagation

- Fix learning rate  $\alpha$
- For all layers  $\ell$  (starting backwards)
  - For all training examples  $x_i, y_i$

- $W^{[\ell]} = W^{[\ell]} - \alpha \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$

- $b^{[\ell]} = b^{[\ell]} - \alpha \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

Incremental  
version of GD

# Mini-batch Gradient Descent

- Initialization

- For all layers  $\ell$ 
  - Set  $W^{[\ell]}, b^{[\ell]}$  at random

- Backpropagation

- Fix learning rate  $\alpha$
- For all layers  $\ell$  (starting backwards)
  - For all batches  $b$  of size  $B$  with training examples  $x_{ib}, y_{ib}$

- $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^B \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$

- $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^B \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial b^{[\ell]}}$

# Gradient Descent Variants



# Gradient Descent Variants



- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent

# Training Neural Networks

- Randomly initialize weights
- Implement forward propagation to get prediction  $\hat{y}_i$  for any training instance  $x_i$
- Compute loss function  $L(\hat{y}_i, y_i)$
- Implement backpropagation to compute partial derivatives  $\frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$  and  $\frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$
- Use gradient descent with backpropagation to compute parameter values that optimize loss
- Can be applied to both feed-forward and convolutional nets

# Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
  - Yann Lecun
- Thanks!