DS 4400

Machine Learning and Data Mining I
Spring 2021

Alina Oprea
Associate Professor
Khoury College of Computer Science
Northeastern University

April 13 2021

Announcements

- Project presentations
  – Tuesday, April 20, 11:45am-2:30pm
  – Thursday, April 22, 9am-12pm
  – 8 minutes per team (5 minute presentation + 3 min questions)

- Project report
  – Monday, April 26, at midnight
  – No late days!

Outline

- Training Neural Networks
  - Backpropagation
  - Parameter Initialization
  - Derivation for feed-forward neural network for binary classification (sigmoid activation)
- Stochastic Gradient Descent
  - Gradient descent variants

How to train Neural Networks?

- Backpropagation algorithm
- David Rumelhart, Geoffrey Hinton, Ronald Williams. "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. 1986
- Applicable to both FFNN and CNN
- Extension of Gradient Descent to multi-layer neural networks

Reminder: Logistic Regression

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

- Cost of a single instance:

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

- Can re-write objective function as

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \text{cost}\left(\; h_\theta(x_i), y_i \; \right)$$

CROSS-ENTROPY LOSS

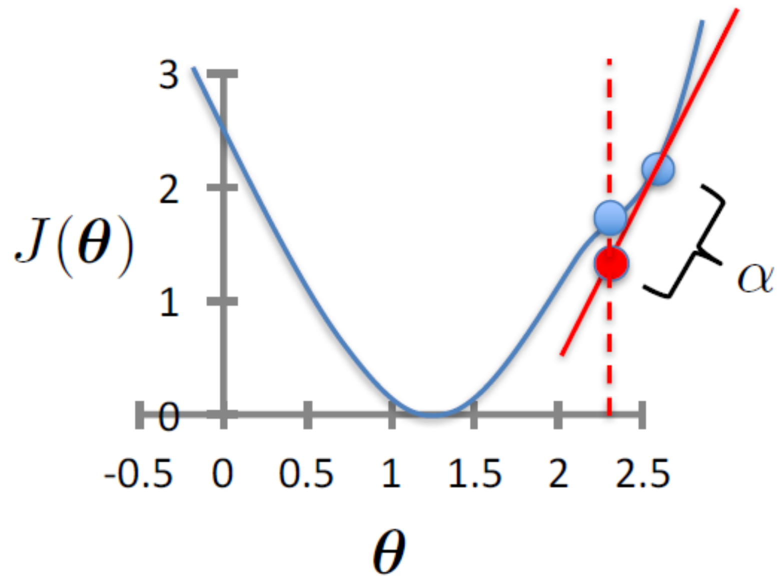
Gradient Descent

- Initialize $\boldsymbol{\theta}$ RANDOM $\boldsymbol{\theta} = (W, b)$
- Repeat until convergence

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

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



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

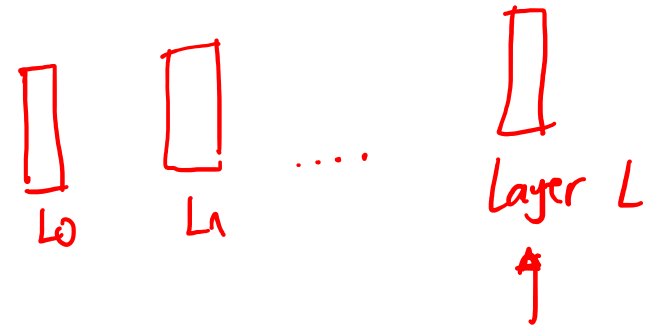
Training Neural Networks

- Training data $x_1, y_1, \dots x_N, y_N$ N training examples
- One training example $x_i = (x_{i1}, \dots x_{id})$, label y_i
- One forward pass through the network
  - Compute prediction $\hat{y}_i = h_\theta(x_i)$ $\theta = [W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}]$
- Loss function

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

$$J(\theta) = \sum_{i=1}^{N} L(y_i, \hat{y}_i)$$

GD for Neural Networks



- Initialization
  - For all layers $\ell = 1, L$
    - Initialize $W^{[\ell]}, b^{[\ell]}$ RANDOM
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ STARTING FROM LAST LAYER $L, \dots, 1$

$$W^{[\ell]} \leftarrow W^{[\ell]} - \alpha \frac{\partial J(\theta)}{\partial W^{[\ell]}}$$

$$b^{[\ell]} \leftarrow b^{[\ell]} - \alpha \frac{\partial J(\theta)}{\partial b^{[\ell]}}$$

UNTIL STOPPING CONDITION

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (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 ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (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

Online Perceptron

Let $\theta \leftarrow [0,0,\ldots,0]$
Repeat:
    Receive training example (x_i, y_i)
    If $y_i \theta^T x_i \leq 0$ // prediction is incorrect
        $\theta \leftarrow \theta + y_i \, x_i$
Until stopping condition

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

Mini-batch Gradient Descent

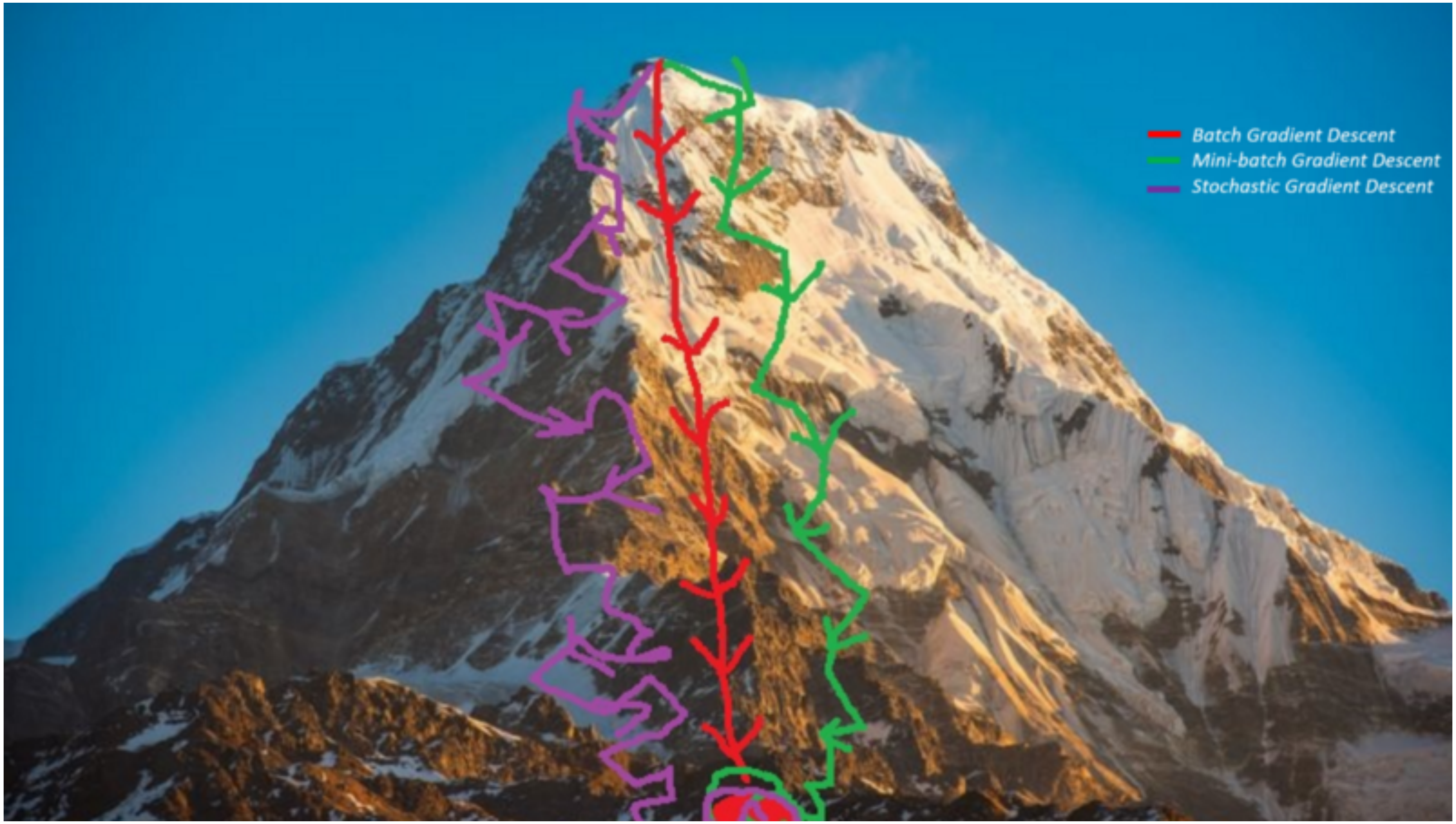
- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
      - For all batches b of size B with training examples x_{ib}, y_{ib}

RANDOMIZED

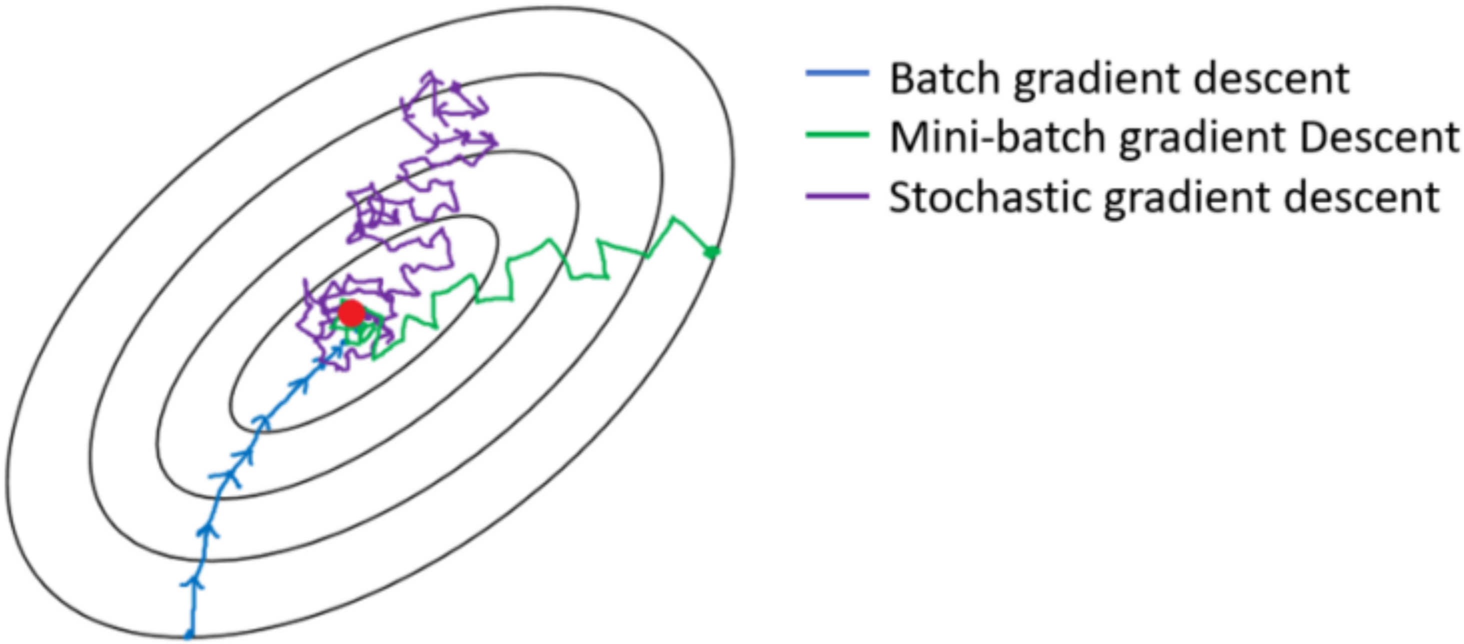
$$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$$

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

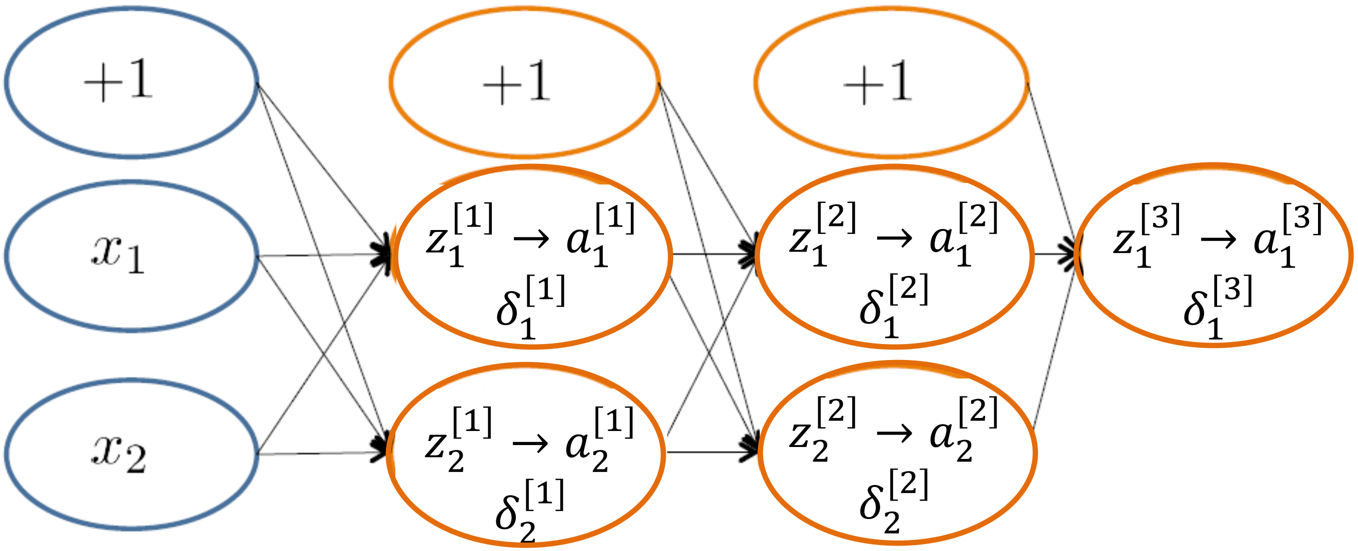
Gradient Descent Variants



Gradient Descent Variants



Backpropagation Intuition

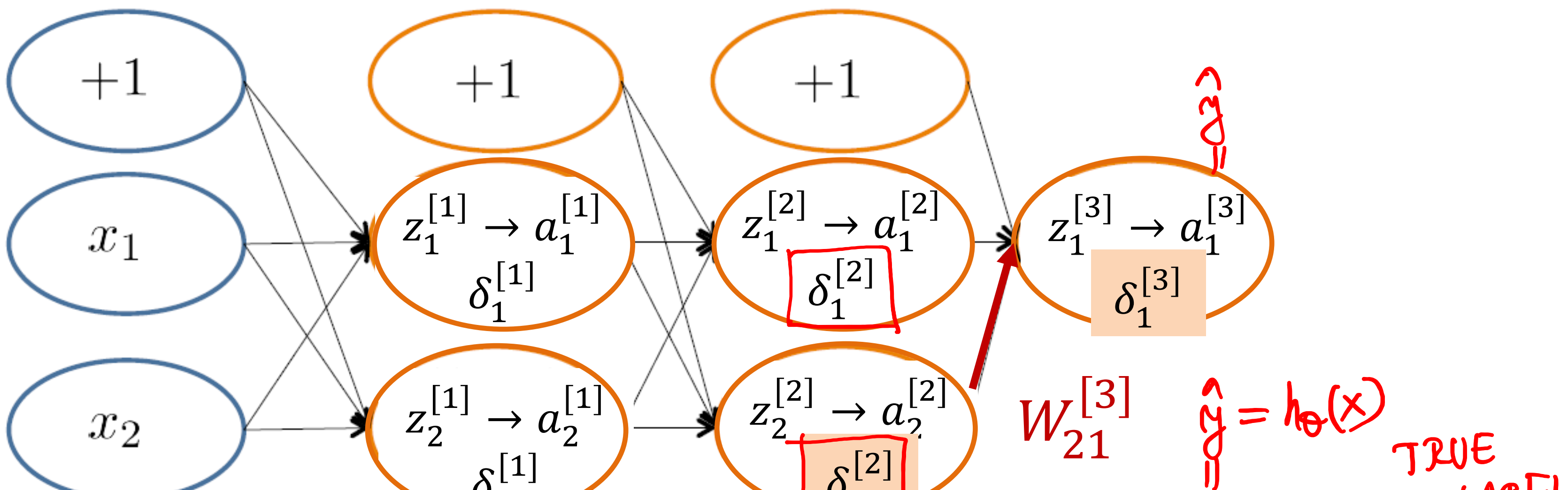


$\delta_j^{(l)}$ = "error" of node j in layer l

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

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

Backpropagation Intuition



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

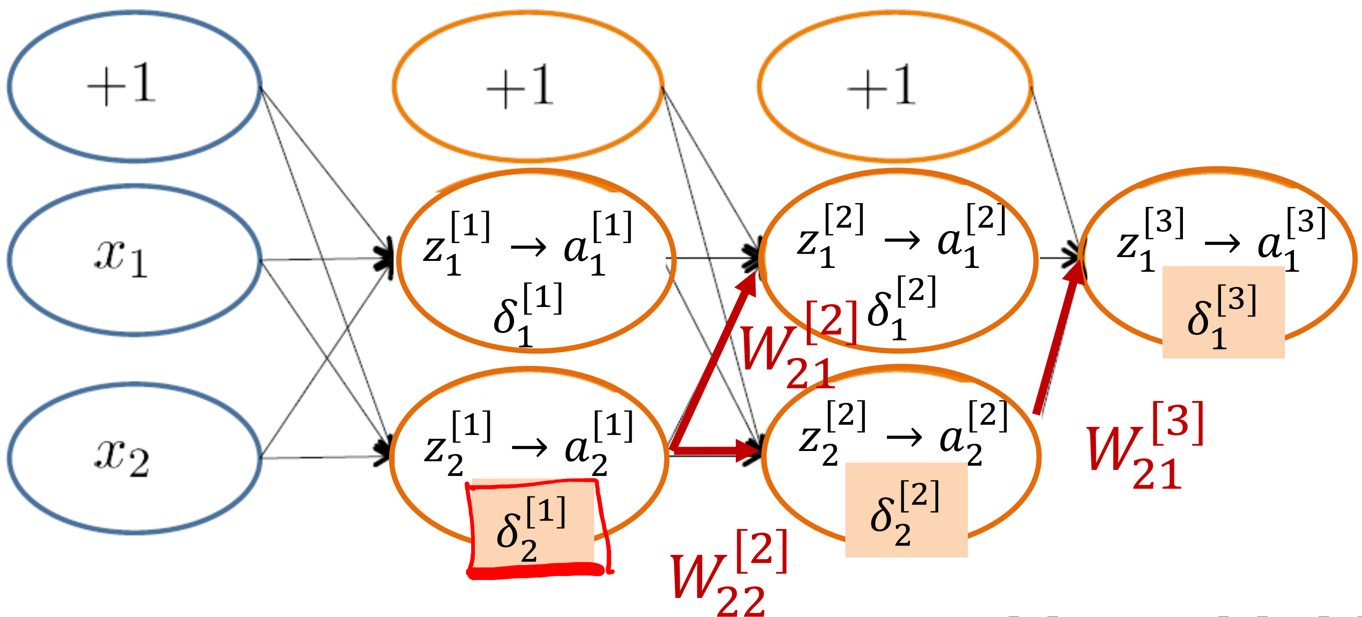
$$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$$

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

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

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

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

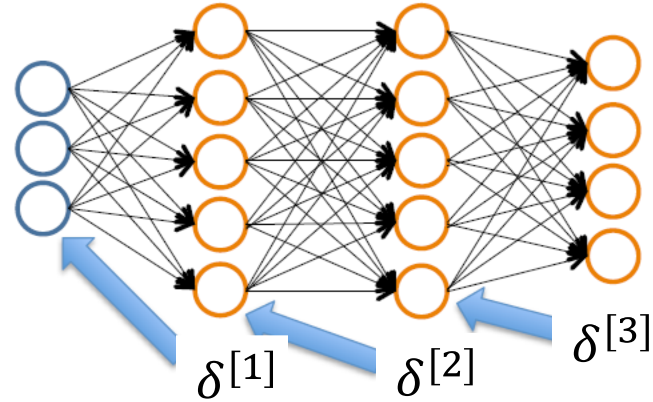
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

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

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

LINEAR ACT

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

DEF

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

• LAYER L: $\delta^{[L]} = \frac{\partial L(y,\hat{y})}{\partial z^{[L]}} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot g'(z^{[L]})$

• LAYER $\ell < L$: $\delta^{[\ell]} = \frac{\partial L(y,\hat{y})}{\partial z^{[\ell]}} = \frac{\partial L(y,\hat{y})}{\partial z^{[\ell+1]}} \cdot \frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \cdot \frac{\partial a^{[\ell]}}{\partial z^{[\ell]}}$

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

$\frac{\partial L(y,\hat{y})}{\partial z^{[\ell+1]}} = \delta^{[\ell+1]}$, $\frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \to W^{[\ell+1]}$, $\frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = g'(z^{[\ell]})$

$= \delta^{[\ell+1]} W^{[\ell+1]} g'(z^{[\ell]})$

Backpropagation

GD UPDATE

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

  where $\dfrac{\partial L(y,\hat{y})}{\partial z^{[l]}} = \delta^{[l]}$ and $\dfrac{\partial z^{[l]}}{\partial W^{[l]}} = a^{[l-1]T}$

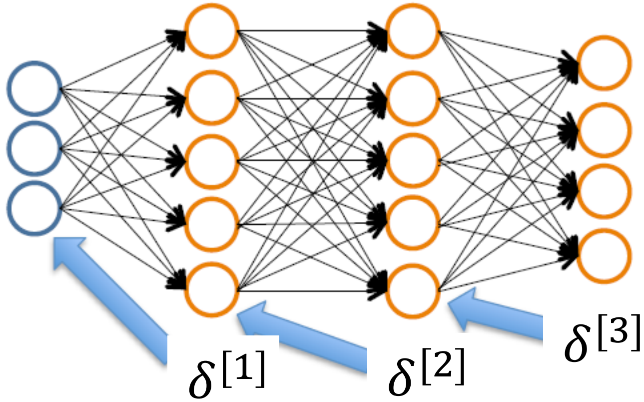
$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

- $\dfrac{\partial L(y,\hat{y})}{\partial b^{[l]}} = \dfrac{\partial L(y,\hat{y})}{\partial z^{[l]}} \cdot \dfrac{\partial z^{[l]}}{\partial b^{[l]}} = \delta^{[l]}$

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

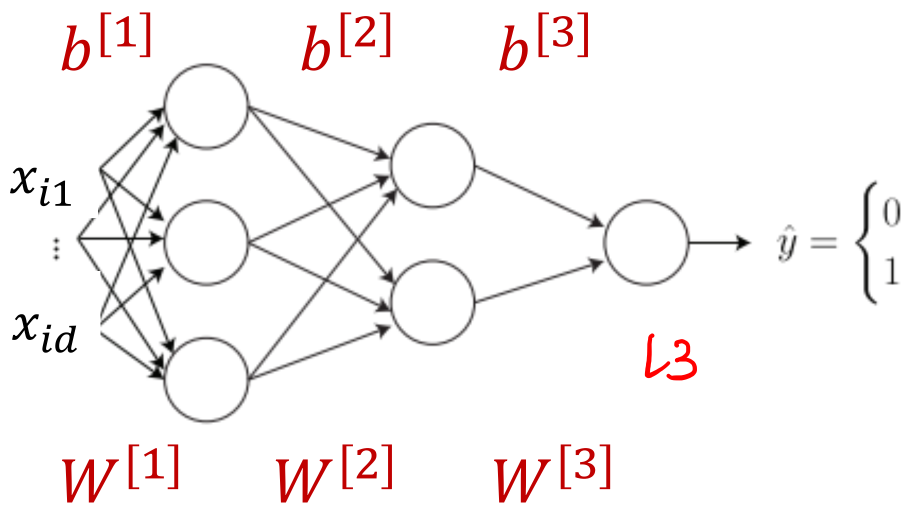
- $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

1. For last layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{z}^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} g'\left(z^{[L]}\right)$
2. For layer ℓ: $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell+1]}} \frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = \delta^{[\ell+1]} W^{[\ell+1]} g'(z^{[\ell]})$
3. Compute parameter gradients

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

Example 2 Hidden Layers

Training data
Dimension d



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

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

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

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

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

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

g=SIGMOID

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

Binary Classification Example

$$\delta^{[3]} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot g'(z^{[3]}) = \frac{\hat{y}-y}{(1-\hat{y})\hat{y}} \cdot \underbrace{g(z^{[3]})}_{a^{[3]}=\hat{y}} \underbrace{(1-g(z^{[3]}))}_{1-\hat{y}} = \hat{y}-y$$

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

$$\frac{\partial L(y,\hat{y})}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} = \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{(1-\hat{y})\hat{y}} = \frac{\hat{y}-y}{(1-\hat{y})\hat{y}}$$

$$g(z) = \frac{1}{1+e^{-z}}, \quad g'(z) = g(z)(1-g(z))$$

$$\delta^{[3]} = \hat{y} - y = a^{[3]} - y$$

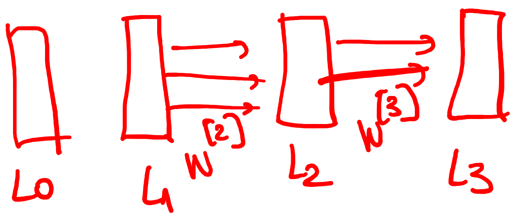
→ ERROR (ONLY FOR SIGMOID ACTIVATION)

$a^{[3]}$: PREDICTION $= h_\theta(x)$

y: TRUE LABEL

Binary Classification Example

- $$\frac{\partial L(y, \hat{y})}{\partial W^{[3]}} = \delta^{[3]} a^{[2]T} = (a^{[3]} - y) a^{[2]T}$$
- $$\frac{\partial L(y, \hat{y})}{\partial b^{[3]}} = \delta^{[3]} = a^{[3]} - y$$



Binary Classification Example

LAYER 2: $\delta^{[2]} = \delta^{[3]} \cdot W^{[3]} g'(z^{[2]})$

$= (a^{[3]} - y) W^{[3]} \cdot g(z^{[2]})(1 - g(z^{[2]})) = (a^{[3]} - y) W^{[3]} a^{[2]} (1 - a^{[2]})$

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

$$\frac{\partial L(y, \hat{y})}{\partial W^{[2]}} = \delta^{[2]} a^{[1]T} = (a^{[3]} - y) W^{[3]} a^{[2]} (1 - a^{[2]}) a^{[1]T}$$

$$\frac{\partial L(y, \hat{y})}{\partial b^{[2]}} = \delta^{[2]} = (a^{[3]} - y) W^{[3]} a^{[2]} (1 - a^{[2]})$$

EXERCISE: COMPUTE $\delta^{[1]}$

$$\frac{\partial L(y, \hat{y})}{\partial W^{[1]}}, \quad \frac{\partial L(y, \hat{y})}{\partial b^{[1]}}$$

Parameter Initialization

- How about we set all W and b to 0?

$$z^{[1]} = W^{[1]}x + b^{[1]} = [0, 0, \dots, 0]$$

$$a^{[1]} = g(z^{[1]}) = [\tfrac{1}{2}, \dots, \tfrac{1}{2}]$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = [0, \dots, 0]$$

$$a^{[2]} = [\tfrac{1}{2}, \dots, \tfrac{1}{2}]$$

$$\vdots$$

$$a^{[L]} = [\tfrac{1}{2}, \dots, \tfrac{1}{2}]$$

$$(x, y) \to (\tfrac{1}{2}, \dots, \tfrac{1}{2})$$

INIT PARAMS AT RANDOM

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 ℓ
Loop

Set $\Delta_{ij}^{[l]} = 0$, for all layers l and indices i, j
For each training instance (x_k, y_k):

EPOCH

BATCH

- Compute $a^{[1]}, a^{[2]}, \dots, a^{[L]}$ via forward propagation
- Compute errors $\delta^{[L]} = a^{[L]} - y_k, \delta^{[L-1]}, \dots \delta^{[1]}$ BACKWARD
- Compute gradients $\Delta_{ij}^{[l]} = \Delta_{ij}^{[l]} + a_j^{[l-1]} \delta_i^{[l]}$ $\delta^{[l]} a^{[l-1]T}$

Update weights via gradient step
- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \Delta_{ij}^{[\ell]}$ $\delta^{[l]} a^{[l-1]T}$
- Similar for $b_{ij}^{[\ell]}$ $b^{[l]} \leftarrow b^{[l]} - \alpha \delta^{[l]}$

Until weights converge or maximum number of epochs is reached

Training Neural Networks

- Randomly initialize weights
- Implement forward propagation to get prediction $\widehat{y_i}$ 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

Materials

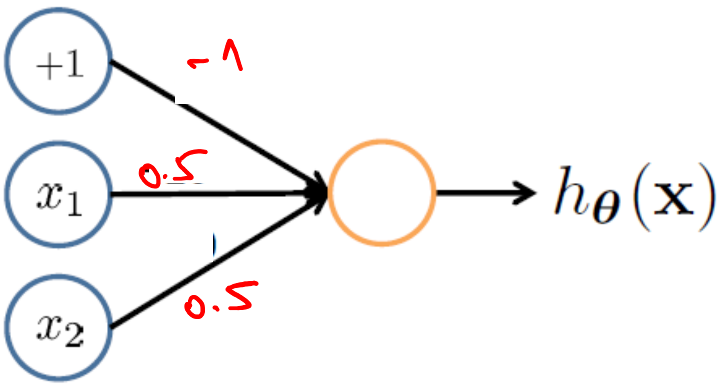
- Stanford tutorial on training Multi-Layer Neural Networks
  - http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
- Notes on backpropagation by Andrew Ng
  - http://cs229.stanford.edu/notes-spring2019/backprop.pdf
- Deep learning notes by Andrew Ng
  - http://cs229.stanford.edu/notes2020spring/cs229-notes-deep_learning.pdf

Representing Boolean Functions

Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



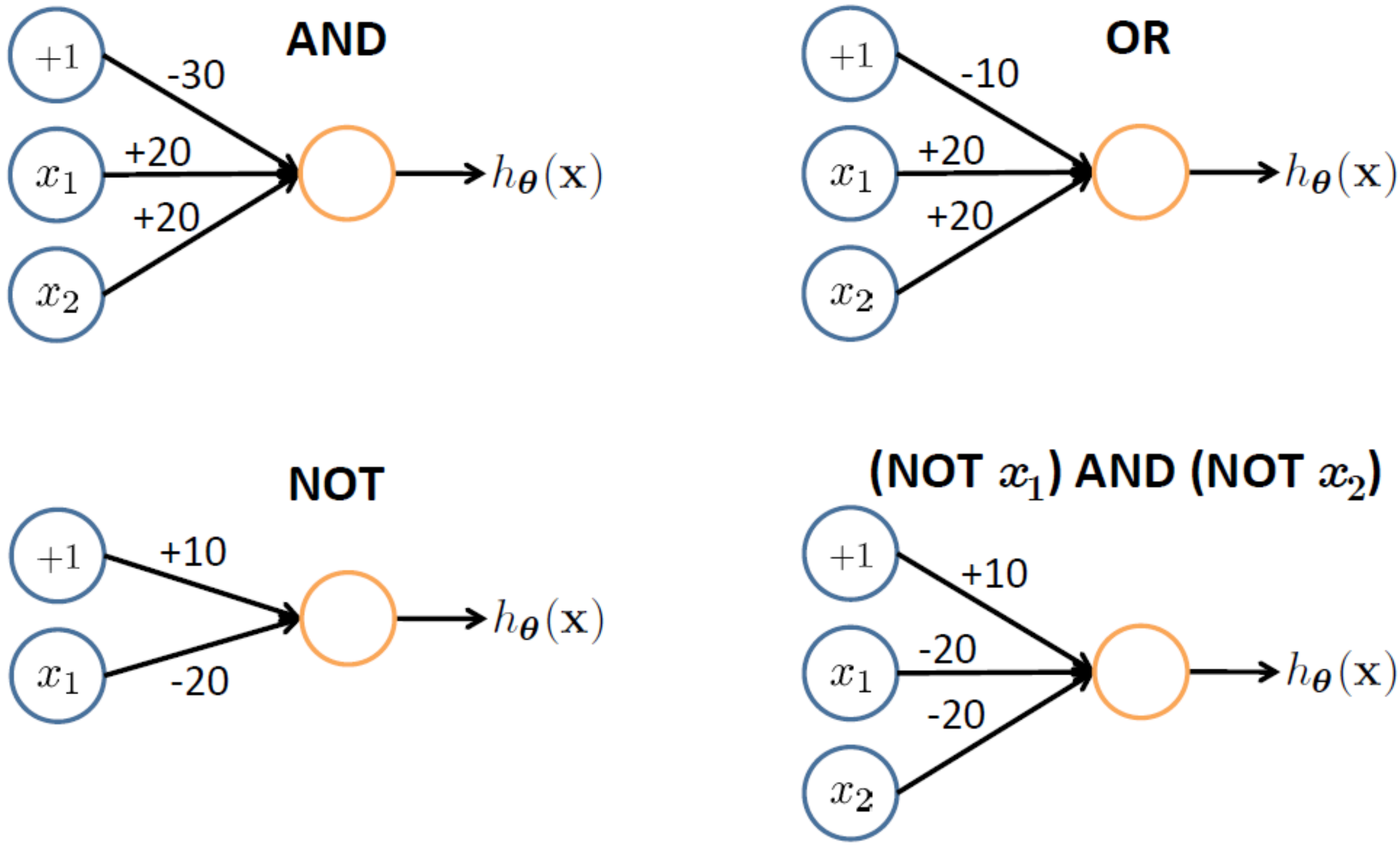
$h_{\Theta}(\mathbf{x}) = g(\ ? \ + \ ? \ x_1 + \ ? \ x_2)$

Activation function

$$g(z) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

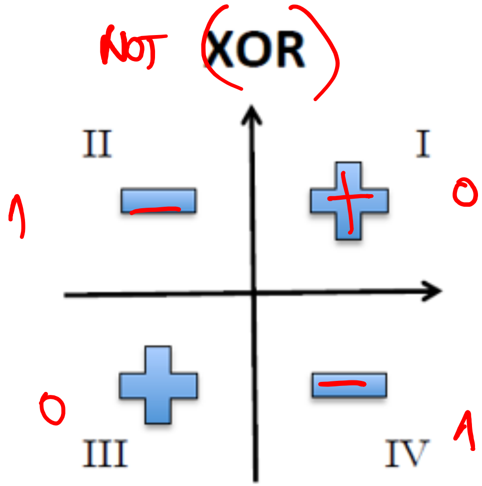
x_1	x_2	$h_{\Theta}(\mathbf{x})$
0	0	0
0	1	0
1	0	0
1	1	1

Representing Boolean Functions



XOR

- Need at least one hidden layer to compute XOR!

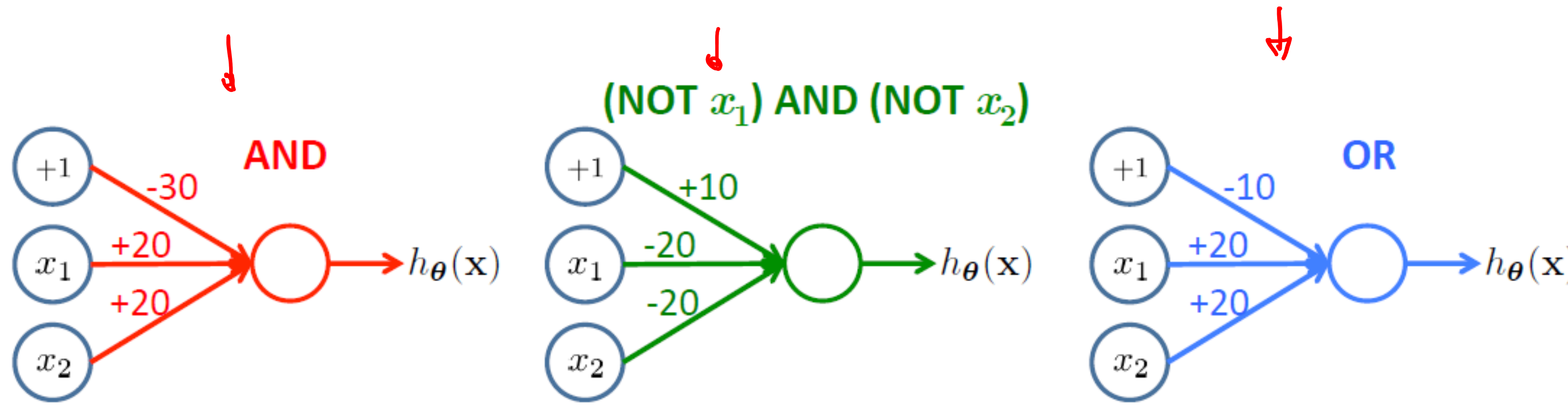


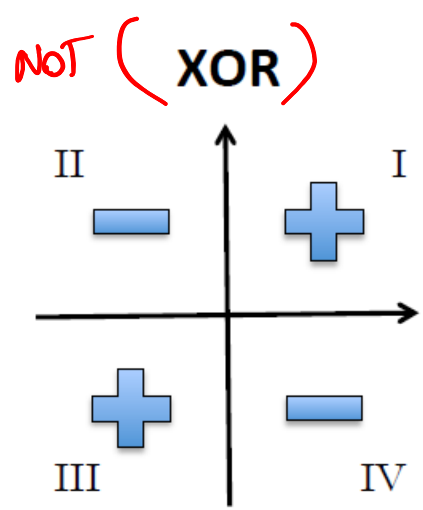
Non-linearly separable

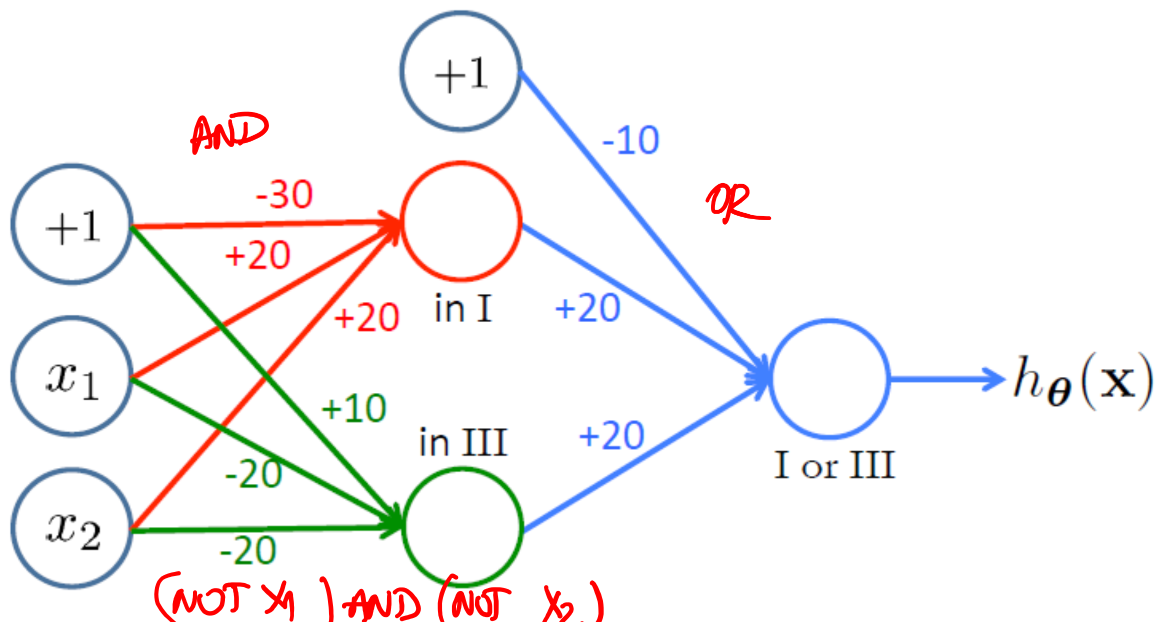
NOT (X1 XOR X2) = (X1 AND X2) OR (NOT X1 AND NOT X2))

		NOT(XOR)
0	0	1
1	0	0
0	1	0
1	1	1

Combining Representations







XOR is non-linear!

Acknowledgements

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