DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

April 2 2019

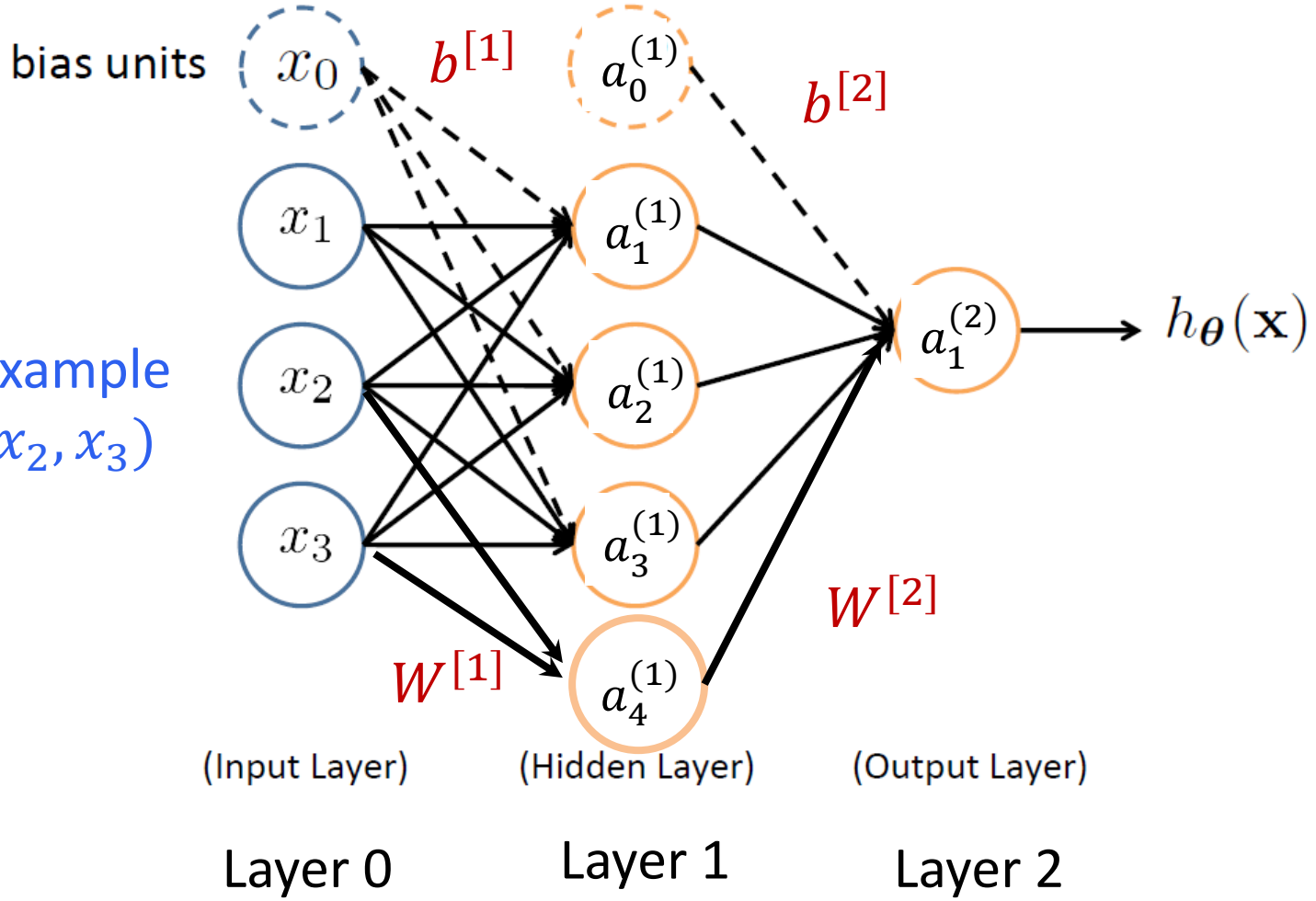
Logistics

- HW 4 is due today!
- TA office hours are today 6-7pm
- Final project presentations
  - Thursday, April 11
  - Tuesday, April 16
  - 8 minute slot – 5 min presentation and 3 min questions
- Final report due on Tuesday, April 23
  - Will prepare template

Outline

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

Feed-Forward Neural Network

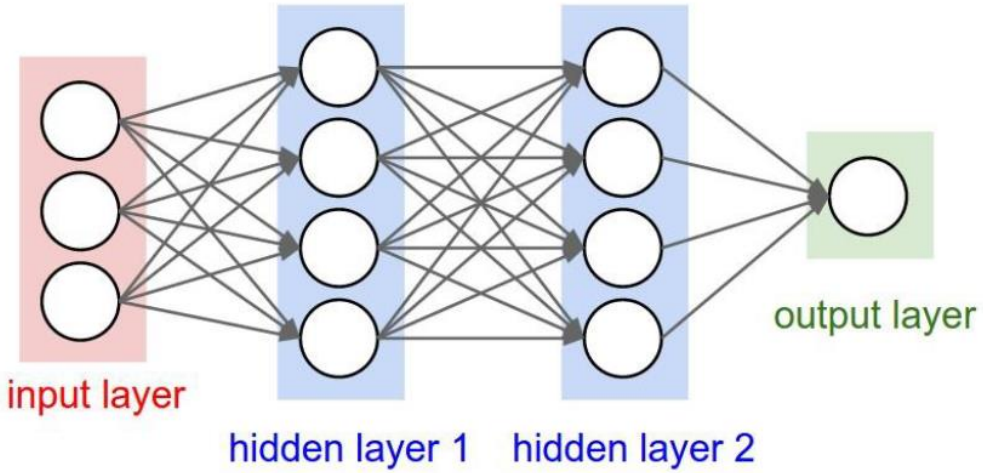


Training example
$x = (x_1, x_2, x_3)$

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.



Perceptron Learning

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(y - h(\mathbf{x}))\mathbf{x}$$

Equivalent to the intuitive rules:

- If output is correct, don't change the weights
- If output is low ($h(\mathbf{x}) = 0$, $y = 1$), increment weights for all the inputs which are 1
- If output is high ($h(\mathbf{x}) = 1$, $y = 0$), decrement weights for all inputs which are 1

Perceptron Convergence Theorem:

- If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge [Minicksy & Papert, 1969]

Batch Perceptron

Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
    Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
    for $i = 1 \ldots n$, do
        if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
            $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)}\boldsymbol{x}^{(i)}$
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}/n$ // compute average update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- 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

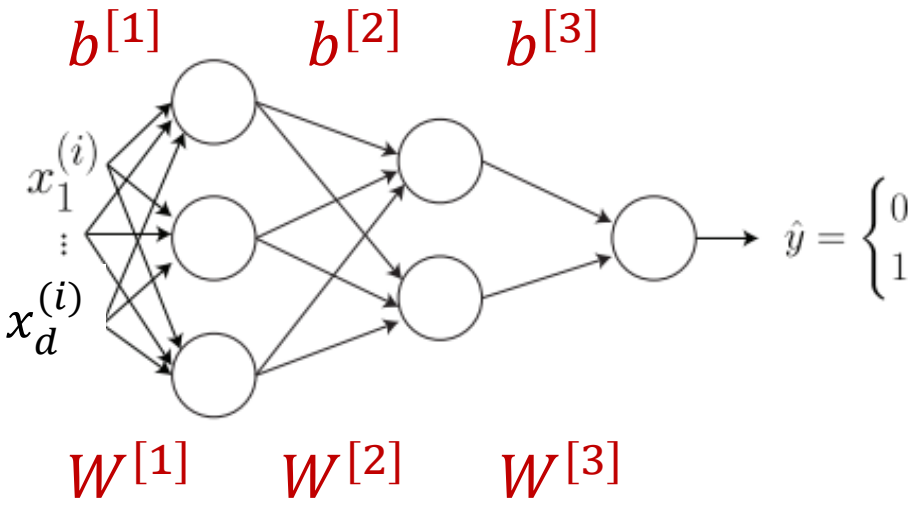
- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error

- 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

Training data
Dimension d



$$z^{[1]} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$
$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}$$
$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]})$$

Parameter Initialization

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

Training

- Training data $x^{(1)}, \mathrm{y}^{(1)}, \ldots x^{(N)}, \mathrm{y}^{(\mathrm{N})}$
- One training example $x^{(i)} = \left(x_1^{(i)}, \ldots x_d^{(i)}\right)$, label $y^{(i)}$
- One forward pass through the network
  - Compute prediction $\hat{y}^{(i)}$
- 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
  - $J(W, b) = \frac{1}{N}\sum_i L\left(\hat{y}^{(i)}, y^{(i)}\right) + \lambda R(W, b)$

Reminder: Logistic Regression

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\left[y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right]$$

- Cost of a single instance:

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

- Can re-write objective function as

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \underbrace{\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}), y^{(i)}\right)}_{\text{Cross-entropy loss}}$$

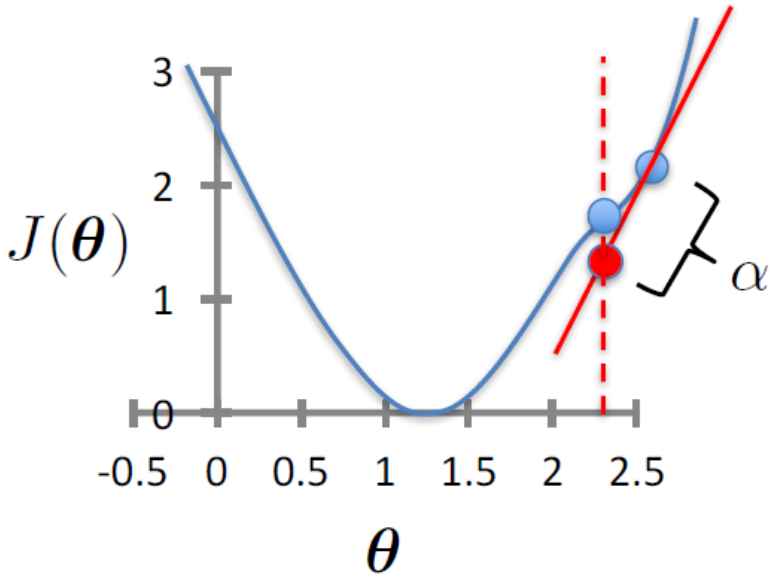
Gradient Descent

- Initialize $\boldsymbol{\theta}$ $\quad \boldsymbol{\theta} = (W, b)$
- 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

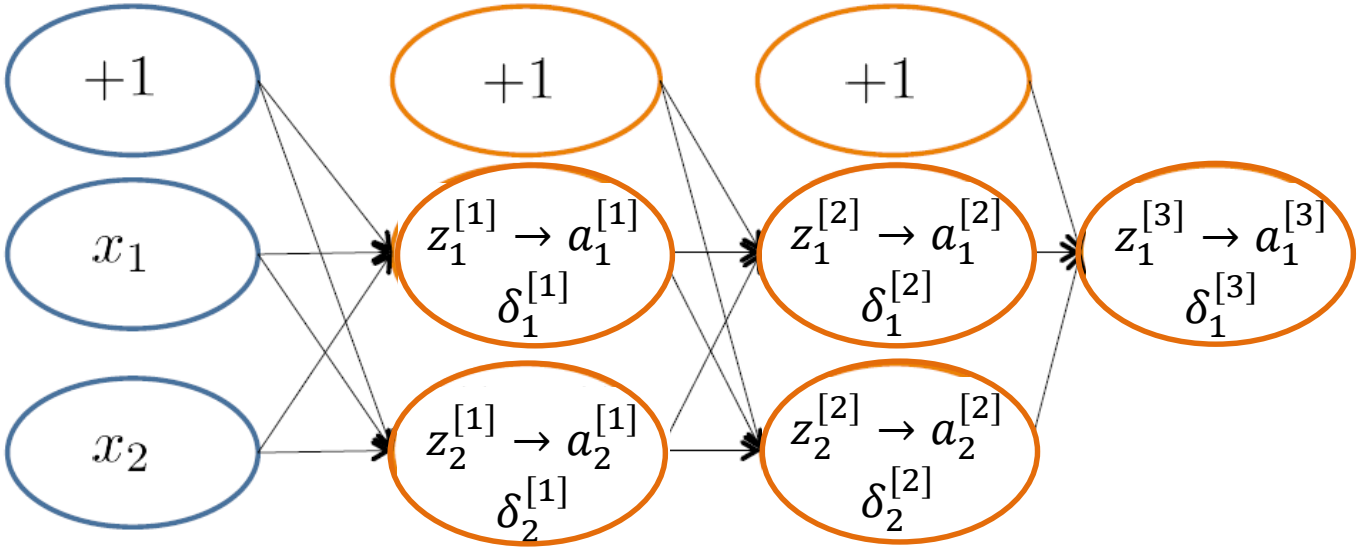


- Converges for convex objective
- Could get stuck in local minimum for non-convex objectives

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}^{(i)}, y^{(i)})}{\partial W^{[\ell]}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}^{(i)}, y^{(i)})}{\partial b^{[\ell]}}$

Backpropagation Intuition

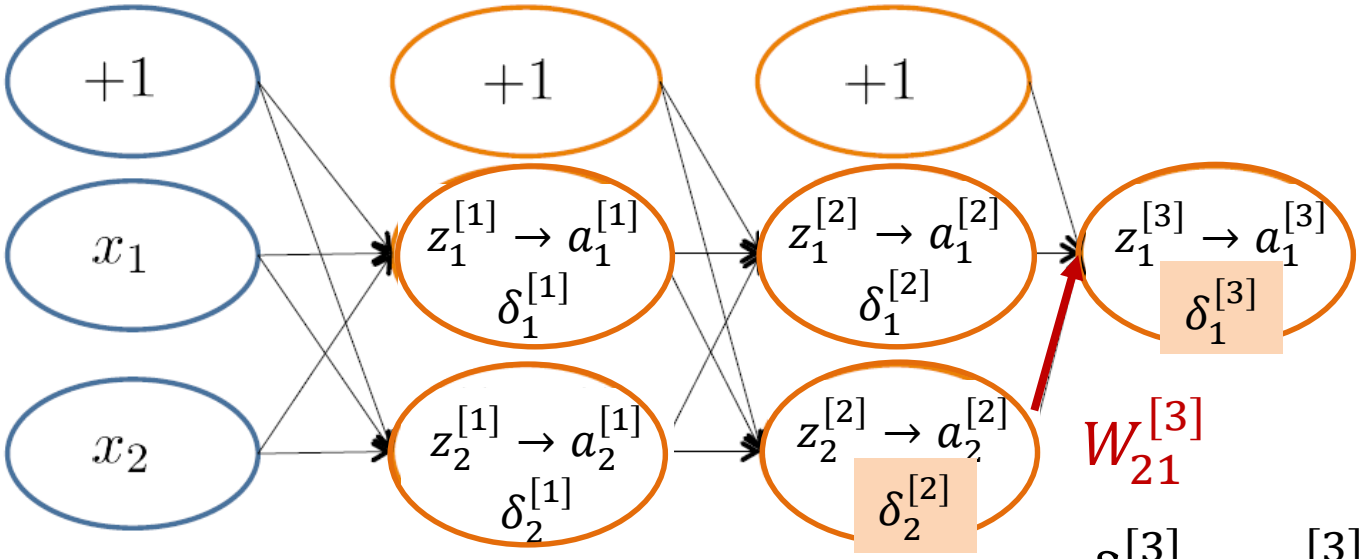


$\delta_j^{(l)}$ = "error" of node j in layer l

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

$$cost\left(x^{(i)}\right) = y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta(x^{(i)})\right)$$

Backpropagation Intuition



$W_{21}^{[3]}$

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

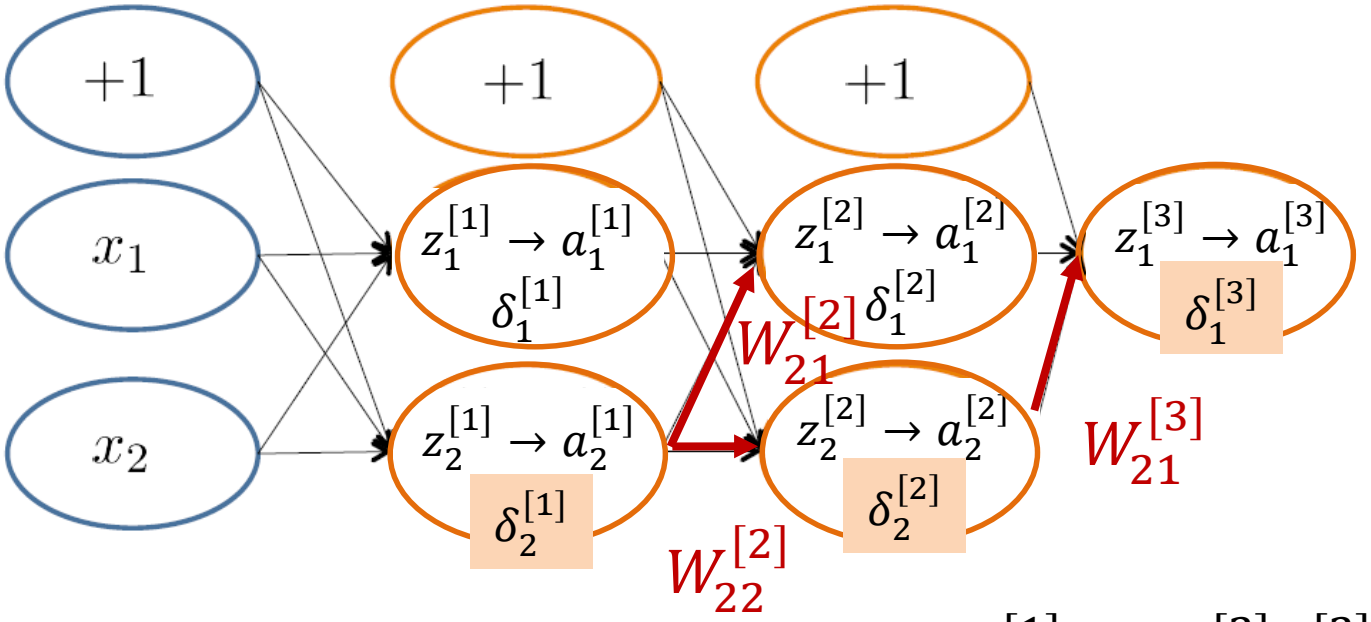
$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$

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

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

$$cost\left(x^{(i)}\right) = y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)$$

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

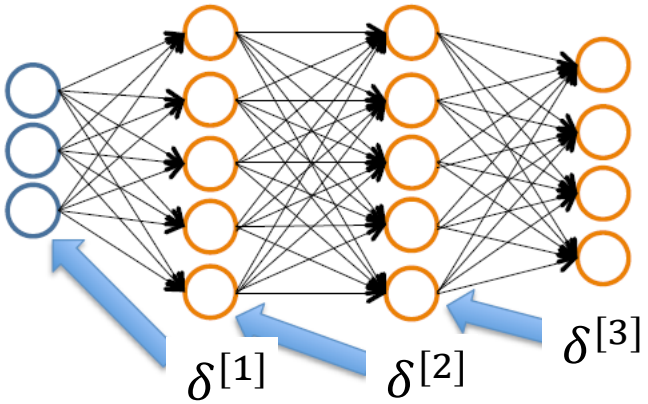
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}\ (x^{(i)})$

$$cost\left(x^{(i)}\right) = y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - h_\theta(x^{(i)}))$$

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

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

1. For last layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial\, \hat{y}} \frac{\partial \hat{y}}{\partial\, z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial\, \hat{y}} g'(z^{[L]})$
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
   - $\frac{\partial L(\hat{y},y)}{\partial\, W^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial\, z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial\, W^{[\ell]}} = \delta^{[\ell]} a^{[\ell-1]T}$
   - $\frac{\partial L(\hat{y},y)}{\partial\, b^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial\, z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial\, b^{[\ell]}} = \delta^{[\ell]}$

Binary Classification Example

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

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

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

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

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

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

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 $(x_1, y_1), \ldots, (x_N, y_N)$
Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ
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 $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$
Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

EPOCH

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

GD for Neural Networks

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

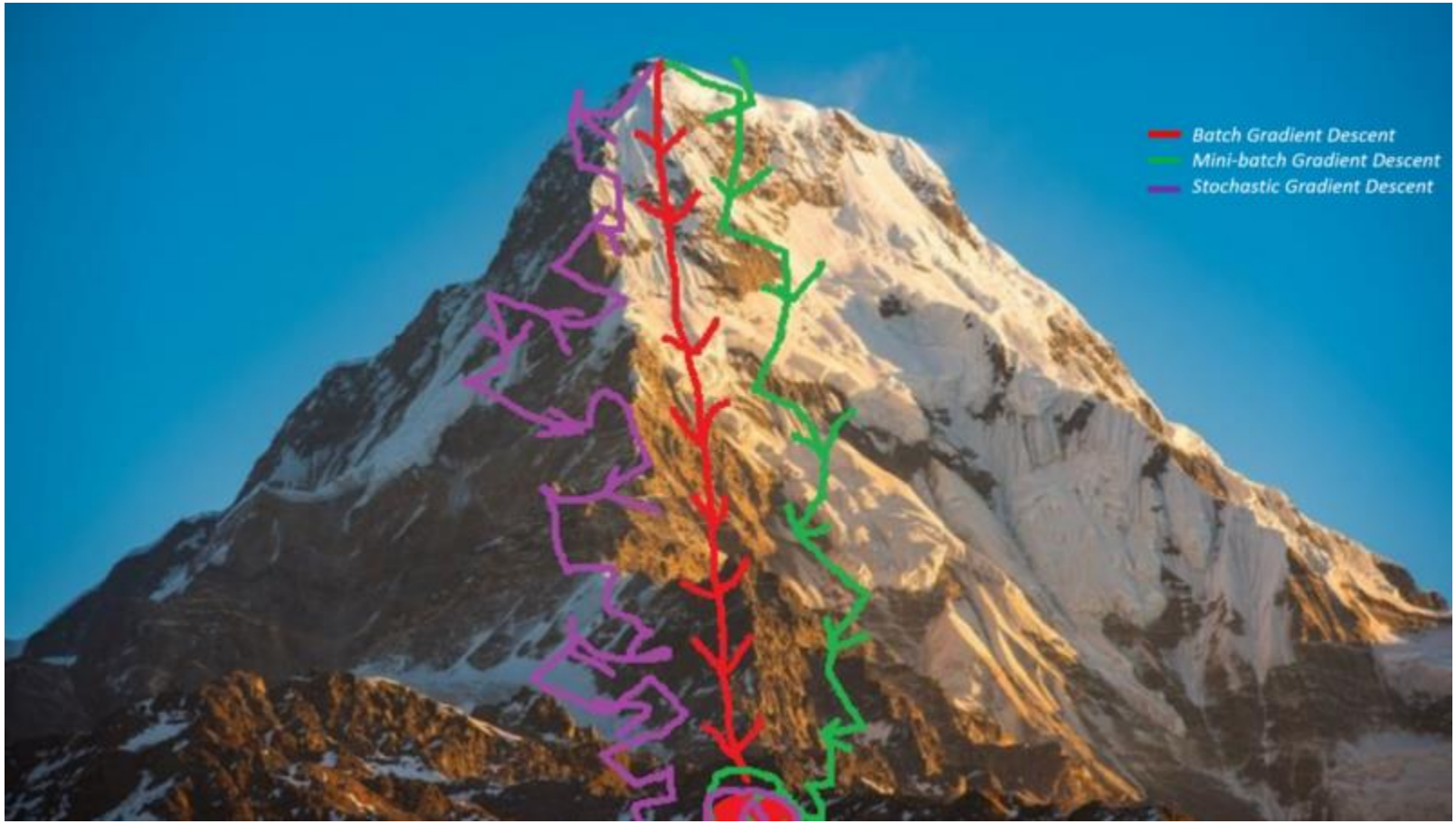
Mini-batch Gradient Descent

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

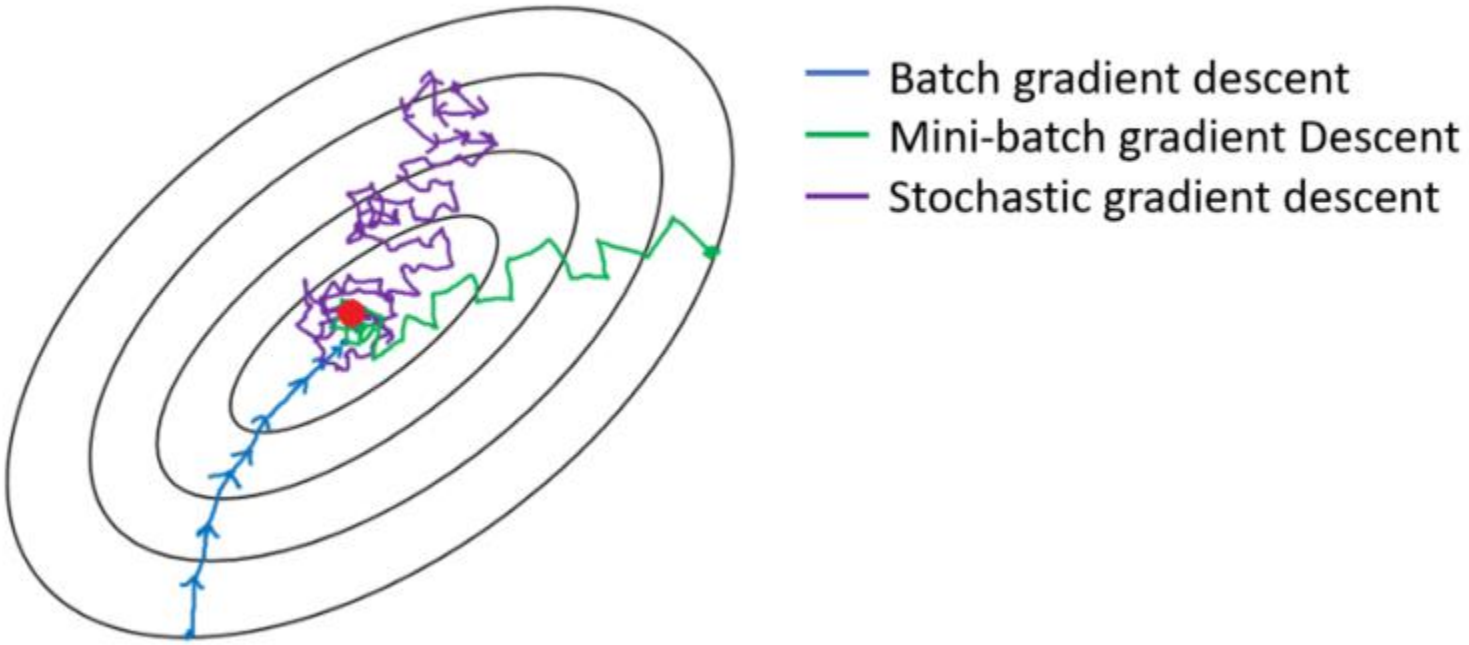
$$- W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}^{(ib)}, y^{(ib)})}{\partial W^{[\ell]}}$$

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

Gradient Descent Variants



Gradient Descent Variants



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:
  - Yann LeCun
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
- Thanks!