DS 4400

Machine Learning and Data Mining I
Spring 2024

David Liu
Khoury College of Computer Science
Northeastern University

April 27 2022

Announcements

- Final exam grading in progress
- Released HW 4 grades
- Final Project due on Monday, May 2
  - Project video recording (5 minute presentation)
  - Project report (6-8 pages)
  - Grading
    - Presentation: 20 points
    - Exploratory data analysis: 18 points
    - ML models: 30 points
    - Metrics: 14 points
    - Interpretation of results: 15 points
    - References: 3 points

Outline

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

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:

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

- Can re-write objective function as

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

Gradient Descent

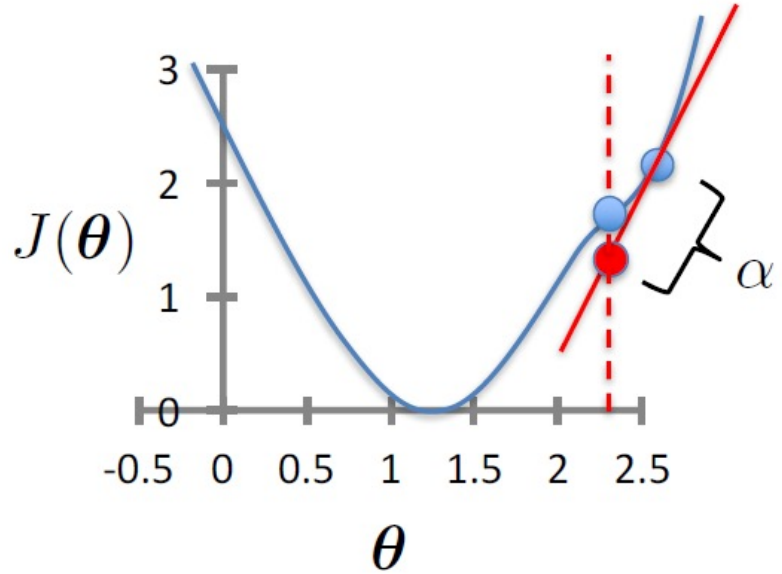
- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

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

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

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 $x_1, y_1, \ldots x_N, y_N$
- One training example $x_i = (x_{i1}, \ldots x_{id}), \text{label } y_i$
- One forward pass through the network
  - Compute prediction $\hat{y}_i = h_\theta(x_i)$
- Loss function

Training Neural Networks

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

Cross-entropy loss

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

GD for Neural Networks

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

GD for Neural Networks

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

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

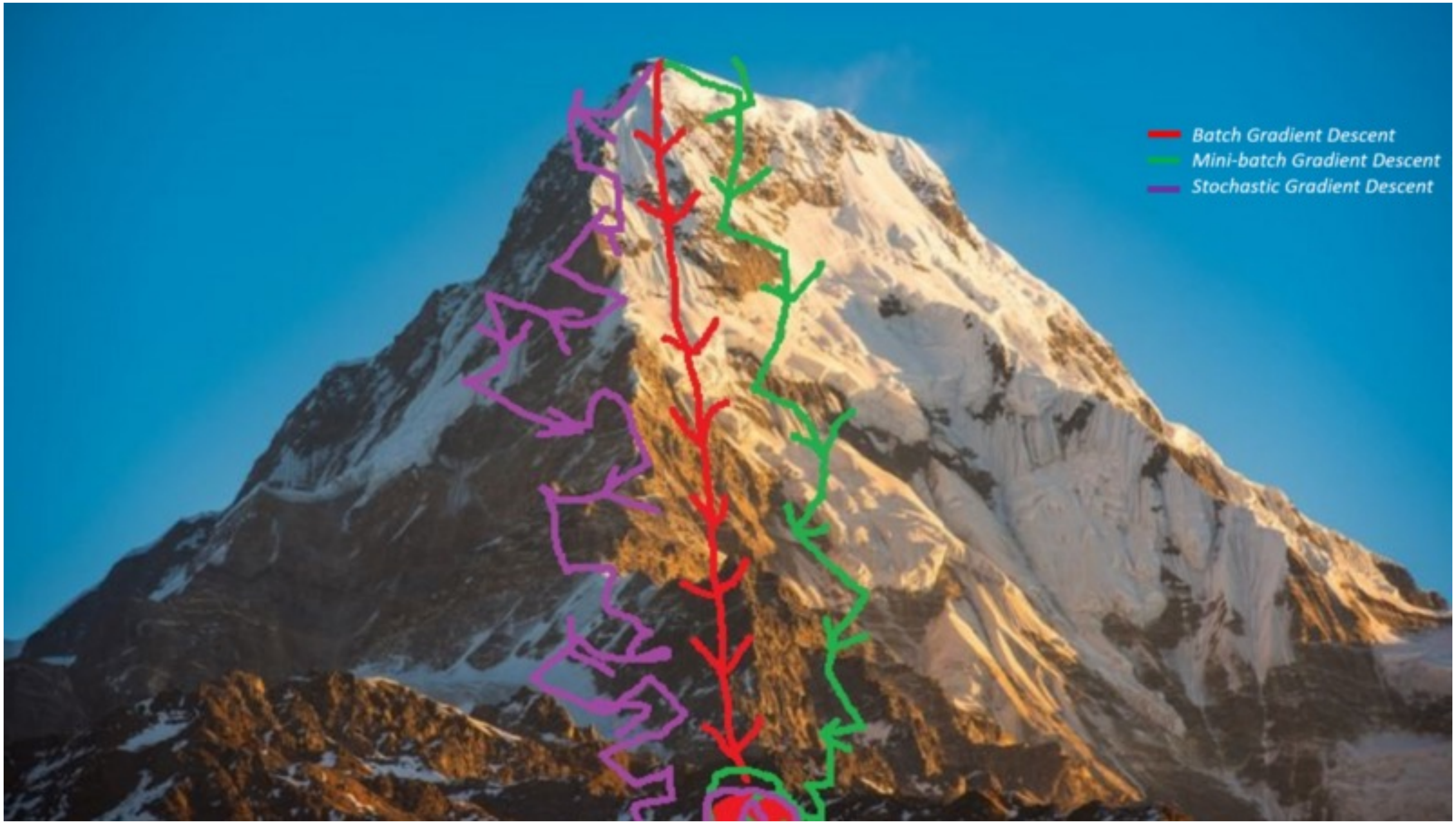
Mini-batch Gradient Descent

- 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}

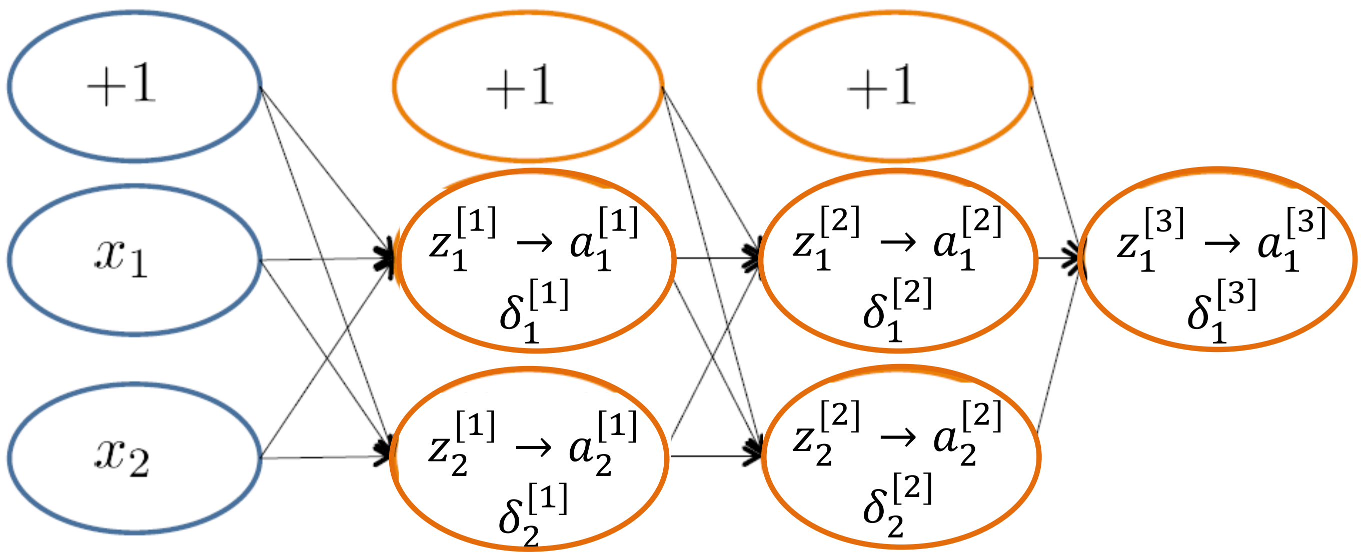
$$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$$

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

Gradient Descent Variants



Backpropagation Intuition

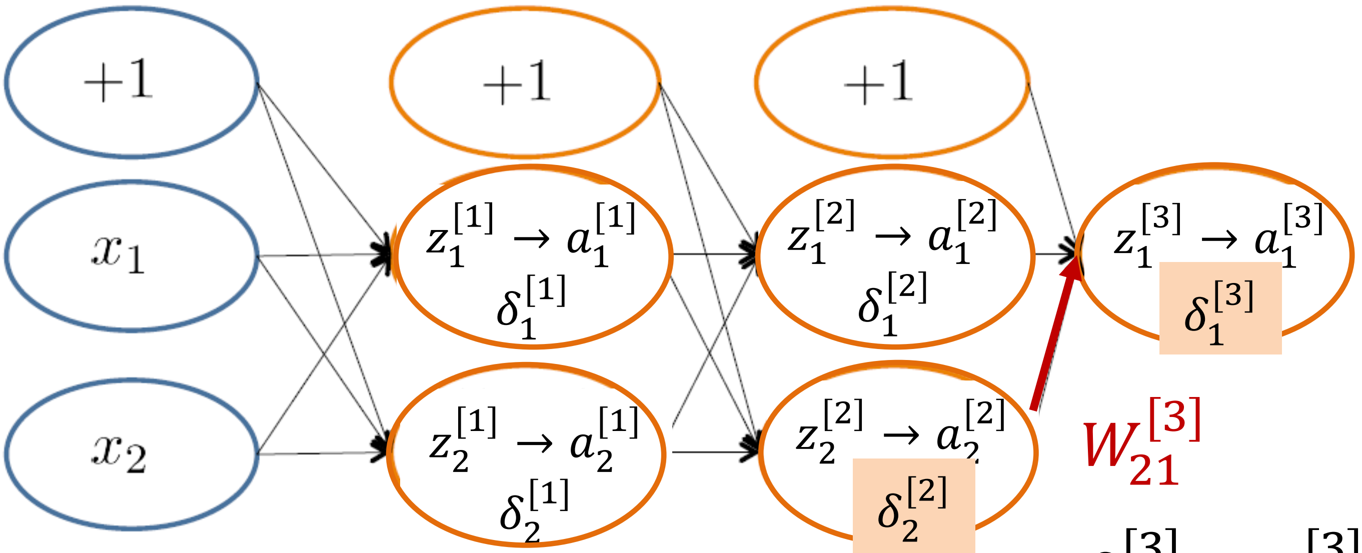


$\delta_j^{(l)}$ = "error" of node j in layer l

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

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

Backpropagation Intuition



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

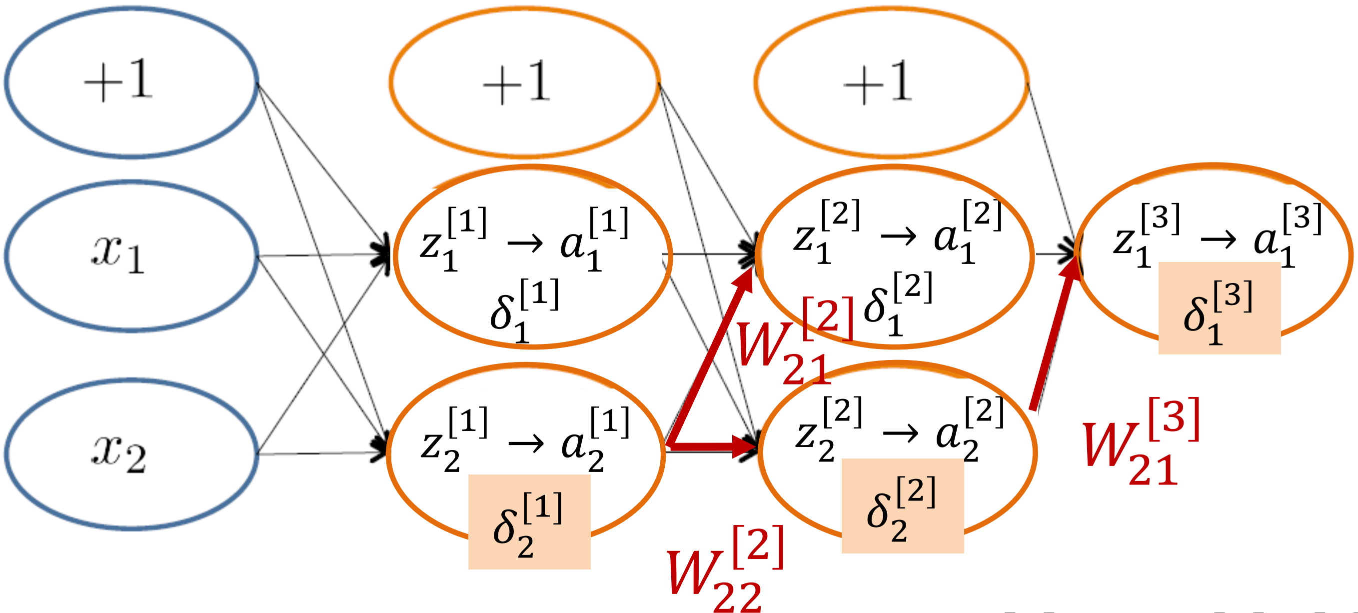
$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$

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

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

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

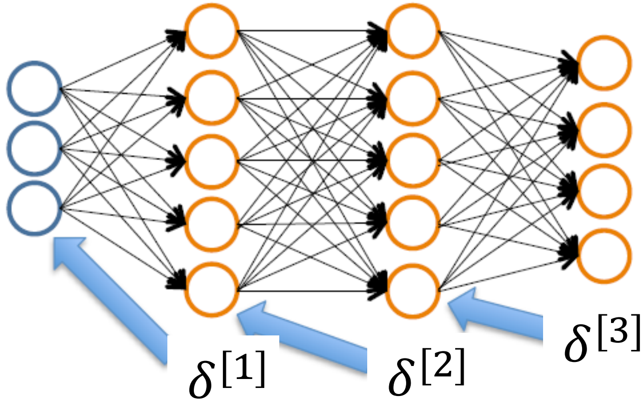
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$

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

Backpropagation

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

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



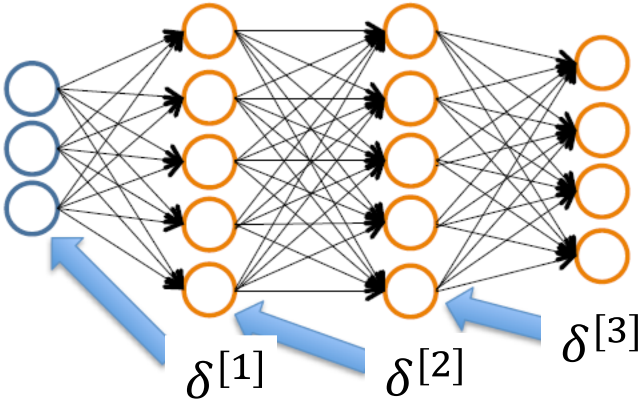
Definitions

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

Backpropagation

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

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



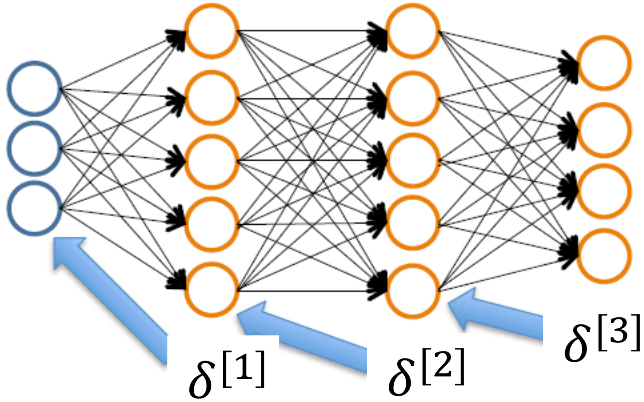
Definitions

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

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

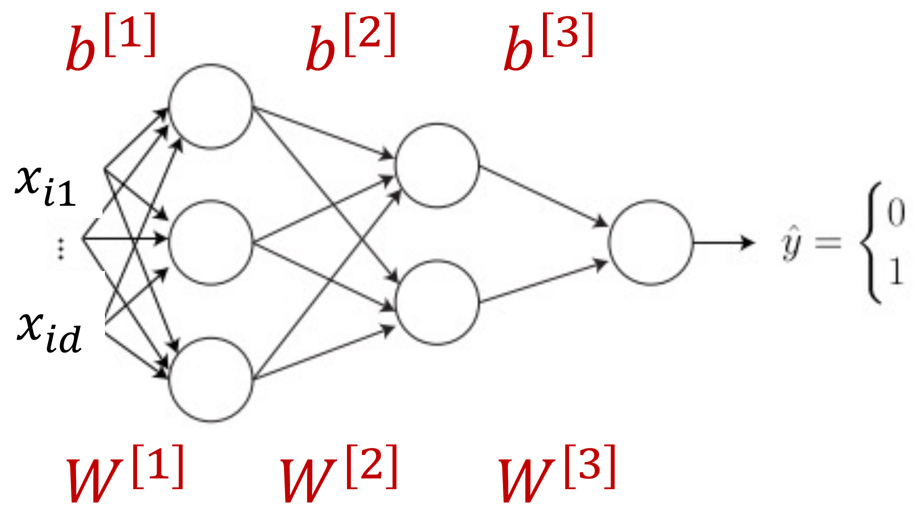
- $z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

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



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

Binary Classification Example

Binary Classification Example

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

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

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

Binary Classification Example

- $\delta^{[2]} = \frac{\partial L(\hat{y},y)}{\partial z^{[2]}} = \delta^{[3]} W^{[3]} g'(z^{[2]})$

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

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

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

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

Parameter Initialization

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

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)
  - $a^{[1]} = \mathrm{g}\left(\mathrm{z}^{[1]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Second layer
  - $z^{[2]} = \mathrm{W}^{[2]}x + \mathrm{b}^{[2]}$= (0,…0)
  - $a^{[2]} = \mathrm{g}\left(\mathrm{z}^{[2]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Third layer
  - $z^{[3]} = \mathrm{W}^{[3]}x + \mathrm{b}^{[3]}$= (0,…0)
  - $a^{[3]} = \mathrm{g}\left(\mathrm{z}^{[3]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$ does not depend on *x*
- Initialize with random values instead!

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

EPOCH

For each training instance (x_k, y_k):

Compute $a^{[1]}, a^{[2]}, \dots, a^{[L]}$ via forward propagation

Compute errors $\delta^{[L]} = a^{[L]} - y_k, \delta^{[L-1]}, \dots \delta^{[1]}$

Compute gradients $\Delta_{ij}^{[l]} = \Delta_{ij}^{[l]} + a_j^{[l-1]} \delta_i^{[l]}$

Update weights via gradient step

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

Until weights converge or maximum number of epochs is reached

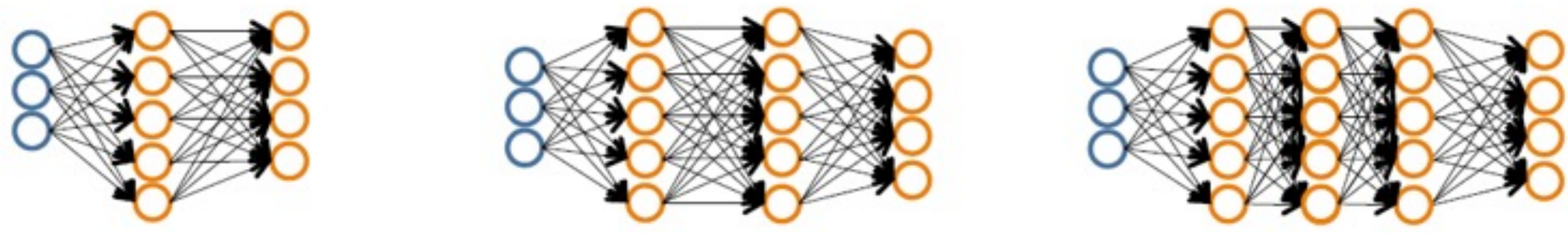
Training Neural Networks

- 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

Overfitting



- The larger the network, the higher the capacity (more model parameters)
- But also more prone to overfitting!

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

<u>L2 regularization:</u> $R(W) = \sum_k \sum_l W_{k,l}^2$ → Weight decay

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- When computing gradients of loss function, regularization term needs to be taken into account

Dropout

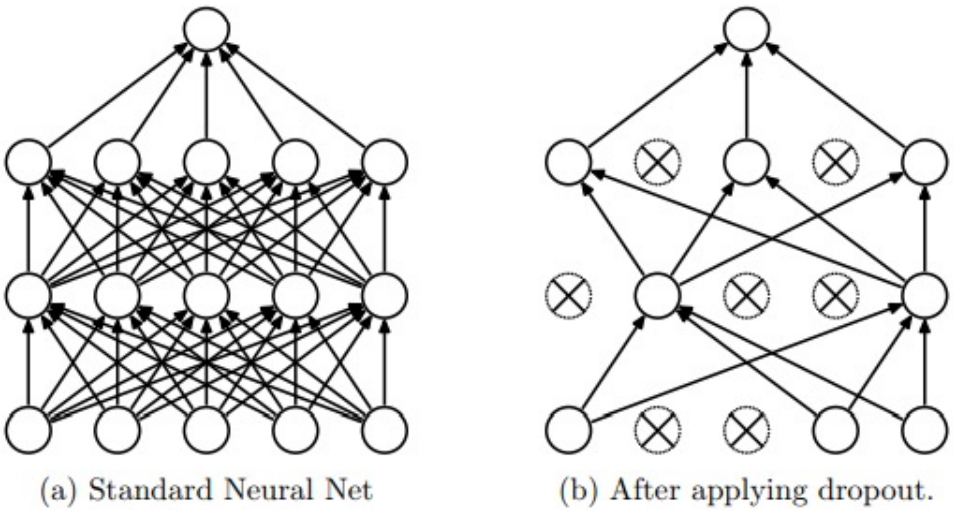


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- Regularization technique that has proven very effective for deep learning
- Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014

Dropout

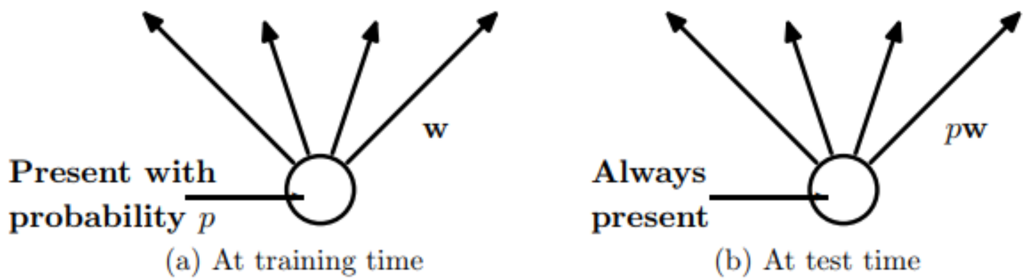


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

- At training time, sample a sub-network per epoch (batch) and learn weights
  - Keep each neuron with probability p
- At testing time, all neurons are there, but multiply weight by a factor of p

Results on MNIST

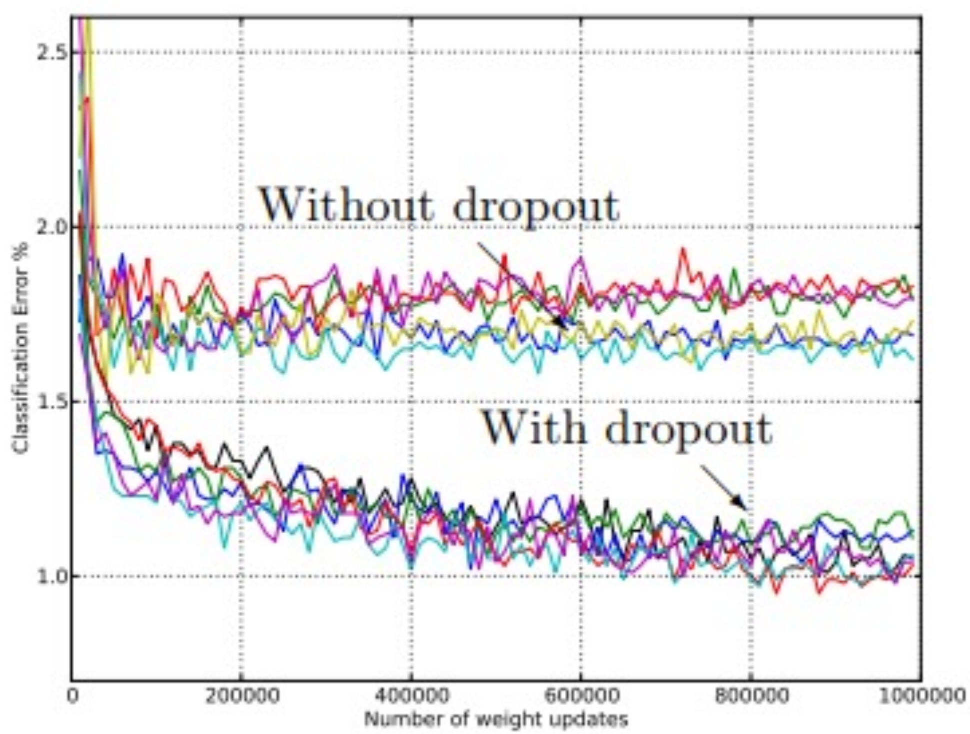


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Review

- Backpropagation is the standard method to train neural networks
  - Applicable to many architectures (FFNNs, CNNs, RNNs)
  - Mini batch gradient descent
  - Parameter updates are done from last layer backwards
  - Deep learning packages perform automatic differentiation (no need to compute gradients by hand)
- Neural networks tend to overfit due to over-parameterization
  - Use regularization (weight decay, dropout)