http://deeplearning.stanford.edu/tutorial/ and then go to MultiLayerNeuralNetworks
http://neuralnetworksanddeeplearning.com/

Topic 5
Neural Networks

INTRODUCTION TO MACHINE LEARNING

PROF. LINDA SELLIE

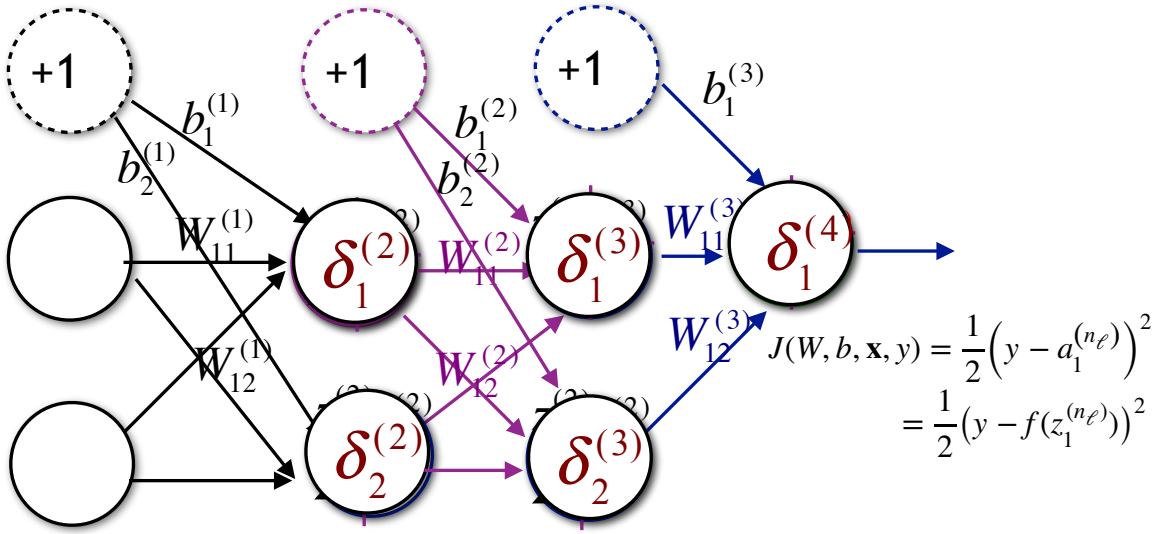
Some of these slides are from Prof. Rangan

$$\delta_i^{(\ell)} = \frac{dJ}{dz_i^{(\ell)}}$$

Chain rule from previous slide:

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{\partial J}{\partial b_i^{(\ell)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$



We first compute $\delta_1^{(4)}$, then we compute $\delta_1^{(3)}$ and $\delta_2^{(3)}$, then we compute $\delta_1^{(2)}$ and $\delta_2^{(2)}$.

What is $\delta_i^{(\ell)}$?

For the output layer:

$$\delta_j^{(n_\ell)} = \frac{\partial J}{\partial z_j^{(n_\ell)}}$$

I added a subscript in case there was more than one output neuron

$$= (\underbrace{f(z_j^{(n_\ell)})}_{\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}} - y_j) f'(z_j^{(n_\ell)})$$

For the other layers:

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}}$$

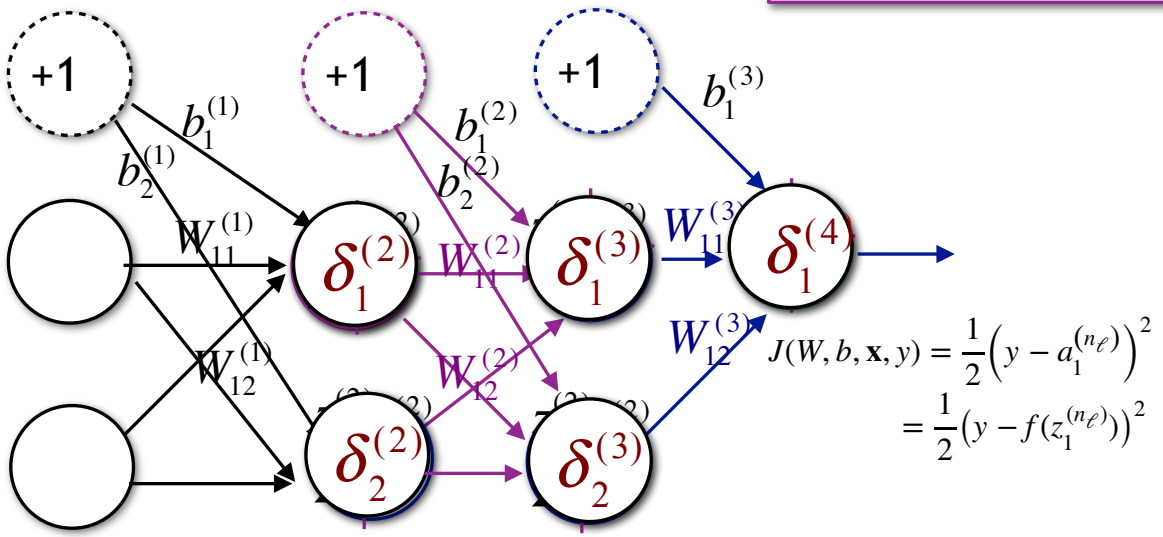
$$= \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)})$$

$$\delta_i^{(\ell)} = \frac{dJ}{dz_i^{(\ell)}}$$

Chain rule from previous slide:

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{\partial J}{\partial b_i^{(\ell)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$



We first compute $\delta_1^{(4)}$, then we compute $\delta_1^{(3)}$ and $\delta_2^{(3)}$, then we compute $\delta_1^{(2)}$ and $\delta_2^{(2)}$.

I could also have written:

$$\delta_j^{(n_\ell)} = -(y_j - f(z_j^{(n_\ell)}))f'(z_j^{(n_\ell)})$$

What is $\delta_i^{(\ell)}$?

For the output layer:

$$\delta_j^{(n_\ell)} = \frac{\partial J}{\partial z_j^{(n_\ell)}}$$

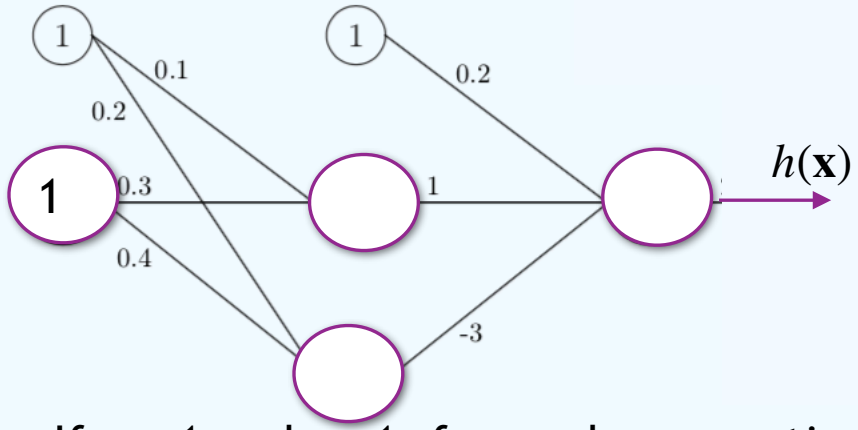
I added a subscript in case there was more than one output neuron

$$= (\underbrace{f(z_j^{(n_\ell)})}_{\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}} - y_j) f'(z_j^{(n_\ell)})$$

For the other layers:

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}}$$

$$= \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)})$$



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1*0.3+0.1 \\ 1*0.4+0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1*0.60+(-3)*0.65+0.2] \qquad a^{(3)} = [\sigma(-1.15)]$$

$$= [0.24]$$

Here we choose $f(z) = \sigma(z)$ to be the logistic function

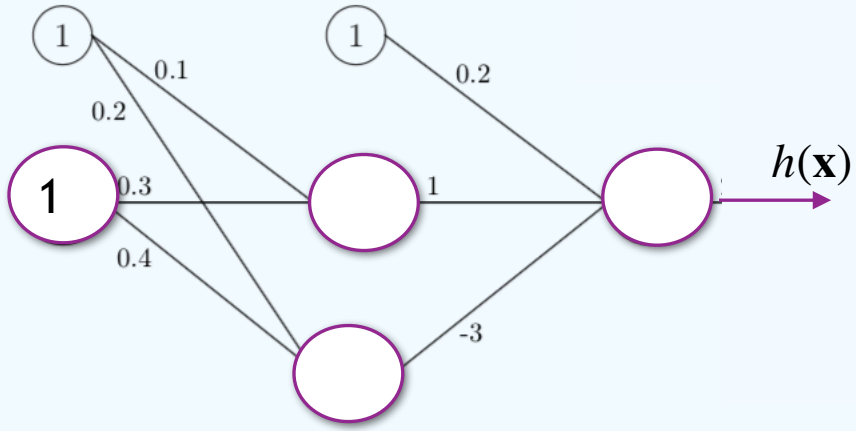
Calculations done with rounded numbers…

Similar to example in Learning from Data

$$\delta_j^{(n_\ell)} = \frac{dJ}{dz_j^{(n_\ell)}} = (f(z_j^{(n_\ell)}) - y_1) f'(z_j^{(n_\ell)})$$

$$\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}$$

$$\frac{df(z)}{dz} = f(z)(1 - f(z))$$



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1 * 0.3 + 0.1 \\ 1 * 0.4 + 0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1 * 0.60 + (-3) * 0.65 + 0.2] \qquad a^{(3)} = [\sigma(-1.15)] = [0.24]$$

Backward propagation:

$$\delta_1^{(3)} = [(0.24 - 1)(.24)(1 - .24)] = [-0.14]$$

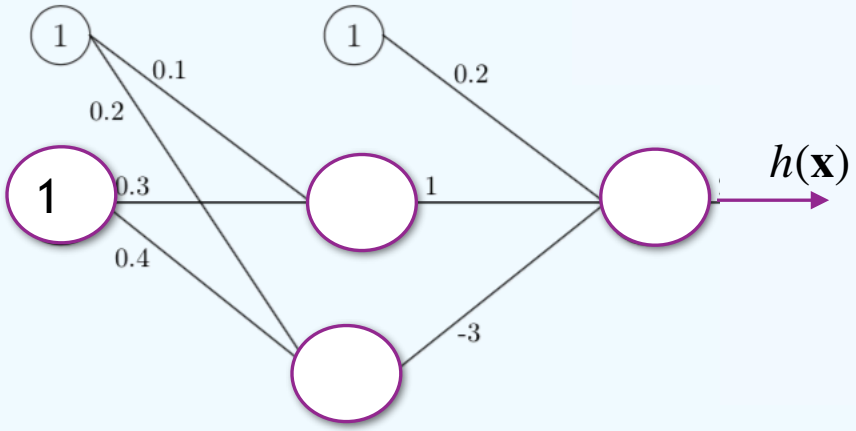
$$f(z_1^{(3)}) = \hat{y}_1 = a_1^{(3)} \qquad y_1 \qquad f'(z_1^{(3)}) = f'(-1.15)$$

Calculations done with rounded numbers…

Similar to example in Learning from Data

$$\delta_j^{(n_\ell)} = \frac{dJ}{dz_j^{(n_\ell)}} = (f(z_j^{(n_\ell)}) - y_1) f'(z_j^{(n_\ell)}) \qquad \hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}$$

$$\delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)}) \qquad \frac{df(z)}{dz} = f(z)(1 - f(z))$$



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1 * 0.3 + 0.1 \\ 1 * 0.4 + 0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1 * 0.60 + (-3) * 0.65 + 0.2] \qquad a^{(3)} = [\sigma(-1.15)] = [0.24]$$

Backward propagation:

$$\delta_1^{(3)} = [(0.24 - 1)(.24)(1 - .24)] = [-0.14]$$

$f(z_1^{(3)}) = \hat{y}_1 = a_1^{(3)}$ y_1 $f'(z_1^{(3)}) = f'(-1.15)$

$$\delta_1^{(2)} = \frac{\partial J}{\partial z_1^{(2)}} = \delta_1^{(3)} W_{11}^{(2)} f'(z_1^{(2)}) = [-0.14 * 1(0.60)(1 - 0.60)] = [-0.03]$$

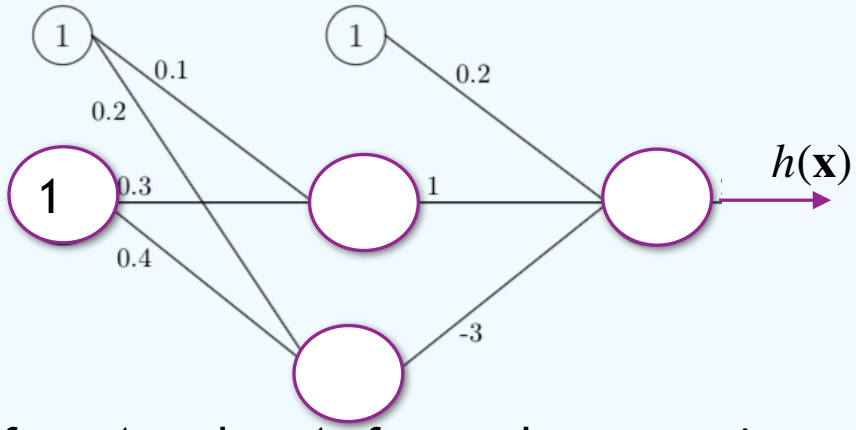
$\delta_1^{(3)}$ $W_{11}^{(2)}$ $f'(z_1^{(2)}) = f'(0.4)$

Calculations done with rounded numbers…

Similar to example in Learning from Data

$$\delta_j^{(n_\ell)} = \frac{dJ}{dz_j^{(n_\ell)}} = (f(z_j^{(n_\ell)}) - y_1) f'(z_j^{(n_\ell)}) \qquad \hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}$$

$$\delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)}) \qquad \frac{df(z)}{dz} = f(z)(1 - f(z)) \qquad \frac{dJ}{dW_{ij}^{\ell}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1 * 0.3 + 0.1 \\ 1 * 0.4 + 0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1 * 0.60 + (-3) * 0.65 + 0.2] \qquad a^{(3)} = [\sigma(-1.15)] = [0.24]$$

Backward propagation:

$$\delta_1^{(3)} = [(0.24 - 1)(.24)(1 - .24)] = [-0.14]$$

$$f(z_1^{(3)}) = \hat{y}_1 = a_1^{(3)} \qquad y_1 \qquad f'(z_1^{(3)}) = f'(-1.15)$$

$$\delta_1^{(2)} = \frac{\partial J}{\partial z_1^{(2)}} = \delta_1^{(3)} W_{11}^{(2)} f'(z_1^{(2)}) = [-0.14 * 1(0.60)(1 - 0.60)] = [-0.03]$$

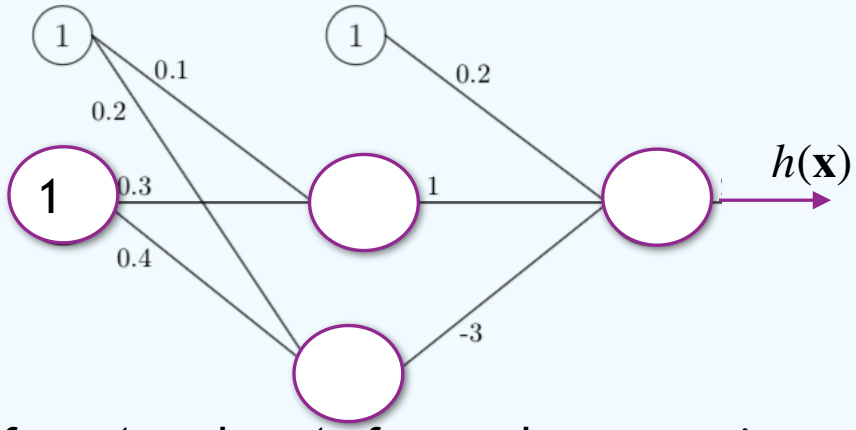
$$\delta_1^{(3)} \qquad W_{11}^{(2)} \qquad f'(z_1^{(2)}) = f'(0.4)$$

$$\frac{\partial J}{\partial W_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)} = [-0.14 * 0.60]$$

Calculations done with rounded numbers…

Similar to example in Learning from Data

$$\delta_j^{(n_\ell)} = \frac{dJ}{dz_j^{(n_\ell)}} = (\underbrace{f(z_j^{(n_\ell)})}_{\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}} - y_1) f'(z_j^{(n_\ell)}) \qquad \delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)}) \qquad \frac{df(z)}{dz} = f(z)(1-f(z)) \qquad \frac{dJ}{dW_{ij}^{\ell}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1*0.3+0.1 \\ 1*0.4+0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1*0.60+(-3)*0.65+0.2] \qquad a^{(3)} = [\sigma(-1.15)] = [0.24]$$

Backward propagation:

$$\delta_1^{(3)} = [\underbrace{(0.24}_{f(z_1^{(3)}) = \hat{y}_1 = a_1^{(3)}} \underbrace{-1)}_{y_1} \underbrace{(.24)(1-.24)}_{f'(z_1^{(3)}) = f'(-1.15)}] = [-0.14]$$

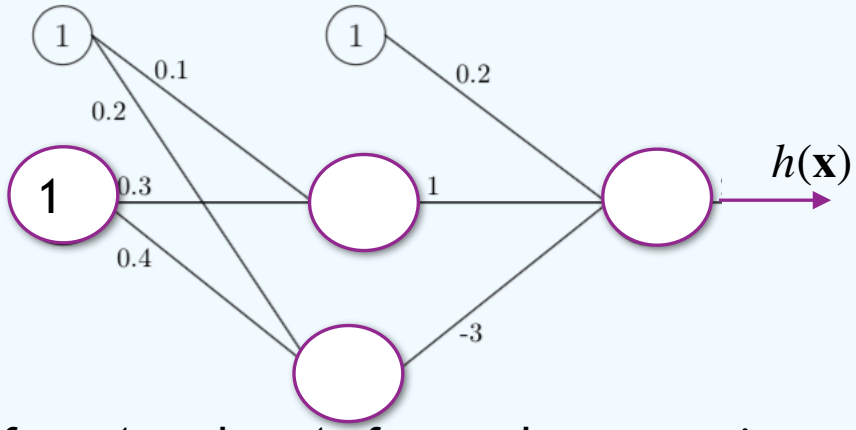
$$\delta_1^{(2)} = \frac{\partial J}{\partial z_1^{(2)}} = \delta_1^{(3)} W_{11}^{(2)} f'(z_1^{(2)}) = [\underbrace{-0.14}_{\delta_1^{(3)}} * \underbrace{1}_{W_{11}^{(2)}} \underbrace{(0.60)(1-0.60)}_{f'(z_1^{(2)}) = f'(0.4)}] = [-0.03]$$

$$\frac{\partial J}{\partial W_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)} = [-0.14 * 0.60] \qquad \frac{\partial J}{\partial W_{11}^{(1)}} = \delta_1^{(2)} a_1^{(1)} = [-0.03 * 1]$$

Calculations done with rounded numbers...

Similar to example in Learning from Data

$$\delta_j^{(n_\ell)} = \frac{dJ}{dz_j^{(n_\ell)}} = (\underbrace{f(z_j^{(n_\ell)})}_{\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}} - y_1) f'(z_j^{(n_\ell)}) \qquad \delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)}) \qquad \frac{df(z)}{dz} = f(z)(1-f(z)) \qquad \frac{dJ}{dW_{ij}^{\ell}} = \delta_i^{(\ell+1)} a_j^{(\ell)} \qquad \frac{dJ}{db_i^{(\ell)}} = \delta_i^{(\ell+1)}$$



$$W_{ij}^{(\ell)} = W_{ij}^{(\ell)} - \alpha \frac{\partial J(W,b,\mathbf{x},y)}{\partial W_{ij}^{(\ell)}} \qquad b_i^{(\ell)} = b_i^{(\ell)} - \alpha \frac{\partial J(W,b,\mathbf{x},y)}{\partial b_i^{(\ell)}}$$

$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} \qquad W^{(2)} = [1 \quad -3]$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \qquad b^{(2)} = [0.2]$$

If x = 1 and y =1, forward propagation:

$$x = a^{(1)} = [1] \qquad z^{(2)} = \begin{bmatrix} 1*0.3+0.1 \\ 1*0.4+0.2 \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} 0.60 \\ 0.65 \end{bmatrix} \qquad z^{(3)} = [1*0.60+(-3)*0.65+0.2] \qquad a^{(3)} = [\sigma(-1.15)] = [0.24]$$

Backward propagation:

$$\delta_1^{(3)} = [\underbrace{(0.24}_{f(z_1^{(3)}) = \hat{y}_1 = a_1^{(3)}} \underbrace{-1)}_{y_1} \underbrace{(.24)(1-.24)}_{f'(z_1^{(3)}) = f'(-1.15)}] = [-0.14]$$

$$\delta_1^{(2)} = \frac{\partial J}{\partial z_1^{(2)}} = \delta_1^{(3)} W_{11}^{(2)} f'(z_1^{(2)}) = [\underbrace{-0.14}_{\delta_1^{(3)}} * \underbrace{1}_{W_{11}^{(2)}} \underbrace{(0.60)(1-0.60)}_{f'(z_1^{(2)}) = f'(0.4)}] = [-0.03]$$

$$\frac{\partial J}{\partial W_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)} = [-0.14 * 0.60] \qquad \frac{\partial J}{\partial W_{11}^{(1)}} = \delta_1^{(2)} a_1^{(1)} = [-0.03 * 1]$$

Calculations done with rounded numbers...

Similar to example in Learning from Data

Training a neural network

- The algorithm
- The problem
- The clever strategy
- Carrying out the strategy step by step
  - Last layer
  - Non-last layer
  - Putting the pieces together
- Partial derivative with regularization and different activation functions

Regularization

$$J(W,b,\mathbf{x},y) = \frac{1}{2}(y-\hat{y})^2 + \frac{\lambda}{2}\sum_{k=1}^{n_\ell-1}\sum_{j=1}^{s_\ell}\sum_{i=1}^{s_{\ell+1}}(W_{ij}^{(k)})^2$$

Adding L2 regularization

$$\frac{\partial J(W,b,\mathbf{x},y)}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)} + \lambda(W_{ij}^{(\ell)})$$

$$\frac{\partial J(W,b,\mathbf{x},y)}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

Activation functions

If we use a ***sigmoid*** activation: $f(z) = \frac{1}{1+e^{-z}}$ then $f'(z) = \frac{df(z)}{dz} = f(z)(1-f(z))$

If we use a ***relu*** activation: $f(z) = \max(0,z)$ then $f'(z) = \frac{df(z)}{dz} = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$

Choose a sub-gradient of 0

$$\frac{\partial J(W,b,\mathbf{x},y)}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)} + \lambda(W_{ij}^{(\ell)})$$

$$\delta_j^{(n_\ell)} = -(y_j - f(z_j^{(n_\ell)}))f'(z_j^{(n_\ell)})$$

$$\frac{\partial J(W,b,\mathbf{x},y)}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

$$\delta_j^{(\ell)} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)})$$

Pair share: think about how to write the update rule for different activation functions

How would you store all the partial derivatives?

$$W^{(\ell)} = \begin{bmatrix} W_{11}^{(\ell)} & W_{21}^{(\ell)} & \cdots & W_{s_{\ell+1}1}^{(\ell)} \\ W_{12}^{(\ell)} & W_{22}^{(\ell)} & & W_{s_{\ell+1}2}^{(\ell)} \\ \vdots & \ddots & & \vdots \\ W_{1s_\ell}^{(\ell)} & W_{2s_\ell}^{(\ell)} & \cdots & W_{s_{\ell+1}s_\ell}^{(\ell)} \end{bmatrix}$$

$$\mathbf{b}^{(\ell)} = \begin{bmatrix} b_1^{(\ell)} \\ b_2^{(\ell)} \\ \vdots \\ b_{s_\ell+1}^{(\ell)} \end{bmatrix}$$

$$\nabla_{W^{(\ell)}} J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)}) = \begin{bmatrix} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{11}^{(\ell)}} & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{12}^{(\ell)}} & \cdots & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{1s_\ell}^{(\ell)}} \\ \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{21}^{(\ell)}} & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{22}^{(\ell)}} & & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{2s_\ell}^{(\ell)}} \\ \vdots & \ddots & & \vdots \\ \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}1}^{(\ell)}} & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}2}^{(\ell)}} & \cdots & \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}s_\ell}^{(\ell)}} \end{bmatrix}$$

$$\nabla_{\mathbf{b}^{(\ell)}} J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)}) = \begin{bmatrix} \frac{\partial J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{\partial b_1^{(\ell)}} \\ \frac{\partial J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{\partial b_2^{(\ell)}} \\ \vdots \\ \frac{\partial J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{\partial b_{s_\ell+1}^{(\ell)}} \end{bmatrix}$$

Previous version of this slide had a typo.

Outline

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
  - Pseudocode
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

Implementing Gradient Descent!

Outline

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
  - Pseudocode
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

To efficiently implement the algorithm we use vectorization

Matrix-Vector Notation: $\delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}}$

Hadamard product: element-wise product operator

$a = b \bullet c \quad a_i = b_i c_i$

$f'([z_1, z_2, z_3]^T) = [f'(z_1), f'(z_2), f'(z_3)]^T$

Our formulas

The output layer:

$$\delta_j^{(n_\ell)} = \frac{\partial J}{\partial z_j^{(n_\ell)}} = -(y_j - f(z_j^{(n_\ell)}))f'(z_j^{(n_\ell)})$$

The other layers:

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}} = \sum_{i=1}^{s_{\ell+1}} \delta_i^{(\ell+1)} W_{ij}^{(\ell)} f'(z_j^{(\ell)})$$

Vectorized formulas

$$\delta^{(n_\ell)} = -(\mathbf{y} - f(\mathbf{z}^{(n_\ell)})) \bullet f'(\mathbf{z}^{(n_\ell)})$$

$$\delta^{(\ell)} = \left(\left(W^{(\ell)}\right)^T \delta^{(\ell+1)}\right) \bullet f'(z^{(\ell)})$$

$$W^{(\ell)} = \begin{bmatrix} W_{11}^{(\ell)} & W_{12}^{(\ell)} & \cdots & W_{1s_\ell}^{(\ell)} \\ W_{21}^{(\ell)} & W_{22}^{(\ell)} & & W_{2s_\ell}^{(\ell)} \\ \vdots & & & \vdots \\ W_{s_{\ell+1}1}^{(\ell)} & W_{s_{\ell+1}2}^{(\ell)} & \cdots & W_{s_{\ell+1}s_\ell}^{(\ell)} \end{bmatrix}$$

$$\left(W^{(\ell)}\right)^T = \begin{bmatrix} W_{11}^{(\ell)} & W_{21}^{(\ell)} & \cdots & W_{s_{\ell+1}1}^{(\ell)} \\ W_{12}^{(\ell)} & W_{22}^{(\ell)} & & W_{s_{\ell+1}2}^{(\ell)} \\ \vdots & & & \vdots \\ W_{1s_\ell}^{(\ell)} & W_{2s_\ell}^{(\ell)} & \cdots & W_{s_{\ell+1}s_\ell}^{(\ell)} \end{bmatrix}$$

$$\delta^{(\ell)} = \begin{bmatrix} \delta_1^{(\ell)} \\ \delta_2^{(\ell)} \\ \vdots \\ \delta_{s_\ell}^{(\ell)} \end{bmatrix} = \left[\begin{bmatrix} W_{11}^{(\ell)} & W_{21}^{(\ell)} & \cdots & W_{s_{\ell+1}1}^{(\ell)} \\ W_{12}^{(\ell)} & W_{22}^{(\ell)} & & W_{s_{\ell+1}2}^{(\ell)} \\ \vdots & & & \vdots \\ W_{1s_\ell}^{(\ell)} & W_{2s_\ell}^{(\ell)} & \cdots & W_{s_{\ell+1}s_\ell}^{(\ell)} \end{bmatrix} \begin{bmatrix} \delta_1^{(\ell+1)} \\ \delta_2^{(\ell+1)} \\ \vdots \\ \delta_{s_{\ell+1}}^{(\ell+1)} \end{bmatrix} \right] \bullet \begin{bmatrix} f'(z_1^{(\ell)}) \\ f'(z_2^{(\ell)}) \\ \vdots \\ f'(z_{s_\ell}^{(\ell)}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{s_{\ell+1}} W_{i1}^{(\ell)} \delta_i^{(\ell+1)} \\ \sum_{i=1}^{s_{\ell+1}} W_{i2}^{(\ell)} \delta_i^{(\ell+1)} \\ \vdots \\ \sum_{i=1}^{s_{\ell+1}} W_{is_{\ell+1}}^{(\ell)} \delta_i^{(\ell+1)} \end{bmatrix} \bullet \begin{bmatrix} f'(z_1^{(\ell)}) \\ f'(z_2^{(\ell)}) \\ \vdots \\ f'(z_{s_\ell}^{(\ell)}) \end{bmatrix}$$

Matrix-Vector Notation:

Definition: $\delta_j^{(\ell)} = \frac{dJ}{dz_j^{(\ell)}}$ $\frac{dJ}{dW_{ij}^{\ell}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$ $\frac{dJ}{db_i^{\ell}} = \delta_i^{(\ell+1)}$

Hadamard product: element-wise product operator $a = b \bullet c$ $a_i = b_i c_i$ $f'([z_1, z_2, z_3]^T) = [f'(z_1), f'(z_2), f'(z_3)]^T$

The partial derivatives:

Our formulas

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{dJ}{db_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

Vectorized formulas

$$\frac{\partial J}{\partial W^{(\ell)}} = \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\frac{dJ}{db^{(\ell)}} = \delta^{(\ell+1)}$$

$$a^{(\ell)} = \begin{bmatrix} a_1^{(\ell)} \\ a_2^{(\ell)} \\ \vdots \\ a_{s_\ell}^{(\ell)} \end{bmatrix} \quad \delta^{(\ell+1)} = \begin{bmatrix} \delta_1^{(\ell+1)} \\ \delta_2^{(\ell+1)} \\ \vdots \\ \delta_{s_{\ell+1}}^{(\ell+1)} \end{bmatrix} \quad W^{(\ell)} = \begin{bmatrix} W_{11}^{(\ell)} & W_{12}^{(\ell)} & \cdots & W_{1s_\ell}^{(\ell)} \\ W_{21}^{(\ell)} & W_{22}^{(\ell)} & & W_{2s_\ell}^{(\ell)} \\ \vdots & & & \vdots \\ W_{s_{\ell+1}1}^{(\ell)} & W_{s_{\ell+1}2}^{(\ell)} & \cdots & W_{s_{\ell+1}s_\ell}^{(\ell)} \end{bmatrix}$$

$$\frac{\partial J}{\partial W^{(\ell)}} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}^{(\ell)}} & \frac{\partial J}{\partial W_{12}^{(\ell)}} & \cdots & \frac{\partial J}{\partial W_{1s_\ell}^{(\ell)}} \\ \frac{\partial J}{\partial W_{21}^{(\ell)}} & \frac{\partial J}{\partial W_{22}^{(\ell)}} & & \frac{\partial J}{\partial W_{2s_\ell}^{(\ell)}} \\ \vdots & & & \vdots \\ \frac{\partial J}{\partial W_{s_{\ell+1}1}^{(\ell)}} & \frac{\partial J}{\partial W_{s_{\ell+1}2}^{(\ell)}} & \cdots & \frac{\partial J}{\partial W_{s_{\ell+1}s_\ell}^{(\ell)}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(\ell+1)} \\ \delta_2^{(\ell+1)} \\ \vdots \\ \delta_{s_{\ell+1}}^{(\ell+1)} \end{bmatrix} \begin{bmatrix} a_1^{(\ell)} & a_2^{(\ell)} & \cdots & a_{s_\ell}^{(\ell)} \end{bmatrix}$$

Batch Gradient Descent with regularization

$$J(W,b) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N} J(W,b,\mathbf{x}^{(i)},\mathbf{y}^{(i)}) + \frac{\lambda}{2}\sum_{k=1}^{n_\ell - 1}\sum_{j=1}^{s_\ell}\sum_{i=1}^{s_{\ell+1}}(W_{ij}^{(k)})^2$$

$$\frac{\partial J(W,\mathbf{b})}{\partial W_{ij}^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial J(W,b,\mathbf{x}^{(i)},\mathbf{y}^{(i)})}{\partial W_{ij}^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \frac{1}{2}\|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\|_2^2}{W_{ij}^{(\ell)}}$$

$$\frac{\partial J(W,b)}{\partial b_i^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial J(W,b,\mathbf{x}^{(i)},\mathbf{y}^{(i)})}{\partial b_i^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \frac{1}{2}\|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\|_2^2}{b_i^{(\ell)}}$$

How do we go from stochastic to batch gradient descent?

Our update rules: $W_{ij}^{(\ell)} = W_{ij}^{(\ell)} - \alpha \cdot \frac{\partial J(W,b)}{\partial W_{ij}^{(\ell)}}$ and $b_i^{(\ell)} = b_i^{(\ell)} - \alpha \cdot \frac{\partial J(W,b)}{\partial b_i^{(\ell)}}$

$$\frac{\partial J(W,b)}{\partial W_{ij}^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial J(W,b,\mathbf{x}^{(i)},\mathbf{y}^{(i)})}{\partial W_{ij}^{(\ell)}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \frac{1}{2}\|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\|_2^2}{\partial W_{ij}^{(\ell)}}$$

$$\Delta W^{(\ell)} = \sum \frac{\partial J(W,b,\boldsymbol{x},y)}{\partial W^{(\ell)}} = \begin{bmatrix} \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{11}^{(\ell)}} & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{12}^{(\ell)}} & \cdots & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{1s_\ell}^{(\ell)}} \\ \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{21}^{(\ell)}} & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{22}^{(\ell)}} & & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{2s_\ell}^{(\ell)}} \\ \vdots & & & \vdots \\ \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}1}^{(\ell)}} & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}2}^{(\ell)}} & \cdots & \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial W_{s_{\ell+1}s_\ell}^{(\ell)}} \end{bmatrix}$$

$$\Delta b^{(\ell)} = \sum \frac{\partial J(W,b,\boldsymbol{x},y)}{\partial b^{(\ell)}} = \begin{bmatrix} \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial z_1^{(\ell)}} \\ \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial z_2^{(\ell)}} \\ \vdots \\ \sum_{i=1}^{N} \frac{\partial J(W,b,\boldsymbol{x}^{(i)},y^{(i)})}{\partial z_{s_{\ell+1}}^{(\ell)}} \end{bmatrix}$$

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

We just need to choose the notation $\Delta W^{(\ell)}$, $\Delta b^{(\ell)}$

The batch gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$\Delta W^{(\ell)} = 0$

$\Delta b^{(\ell)} = 0$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level $\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)}\left(a^{(\ell)}\right)^T$

$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Outline

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
  - Pseudocode
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

The batch gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$\Delta W^{(\ell)} = 0$

$\Delta b^{(\ell)} = 0$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level $\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)}\left(a^{(\ell)}\right)^T$

$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Random Initialization of Parameters

```
nn_structure = [3, 4, 3]
```

What are the dimensions of $W^{(1)}$?

a) 1x1
b) 3x4
c) 4x3
d) 3x3

Random Initialization of Parameters

```
nn_structure = [3, 4, 3]
```

$W^{(1)}$

```
[[0.65, 0.18, 0.89],
 [0.06, 0.74, 0.72],
 [0.92, 0.29, 0.71],
 [0.39, 0.41, 0.10]]
```

$b^{(1)}$

```
0.07,
0.35,
0.60,
0.50
```

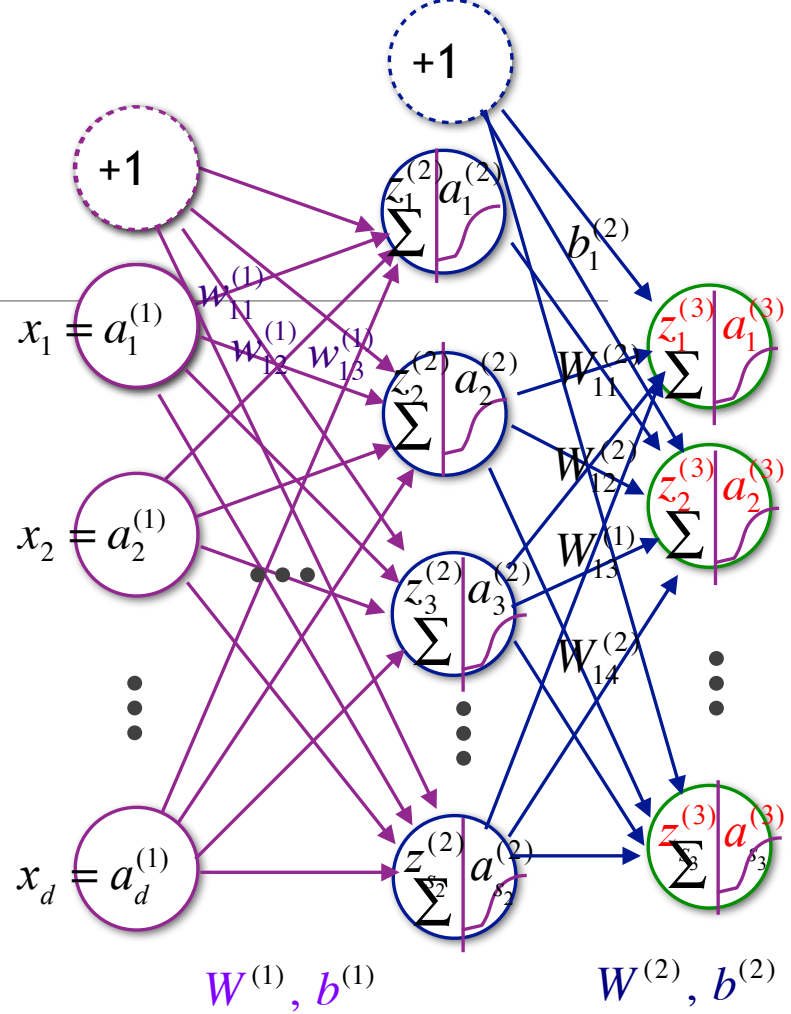
$W^{(2)}$

```
[[0.45, 0.64, 0.10, 0.82],
 [0.30, 0.25, 0.39, 0.36],
 [0.78, 0.60, 0.16, 0.07]]
```

$b^{(2)}$

```
0.14,
0.64,
0.62
```

```
def setup_and_init_weights(nn_structure):
    W = {}
    b = {}
    for l in range(1, len(nn_structure)):
        W[l] = r.random_sample((nn_structure[l], nn_structure[l-1]))
        b[l] = r.random_sample((nn_structure[l],))
    return W, b
```



Array - we represent it as column
Numpy represents it as a row

The gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$$\Delta W^{(\ell)} = 0$$
$$\Delta b^{(\ell)} = 0$$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$
run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ
Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level $\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$

$$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Initialization of △W and △b to zero

nn_structure = [3, 4, 3]

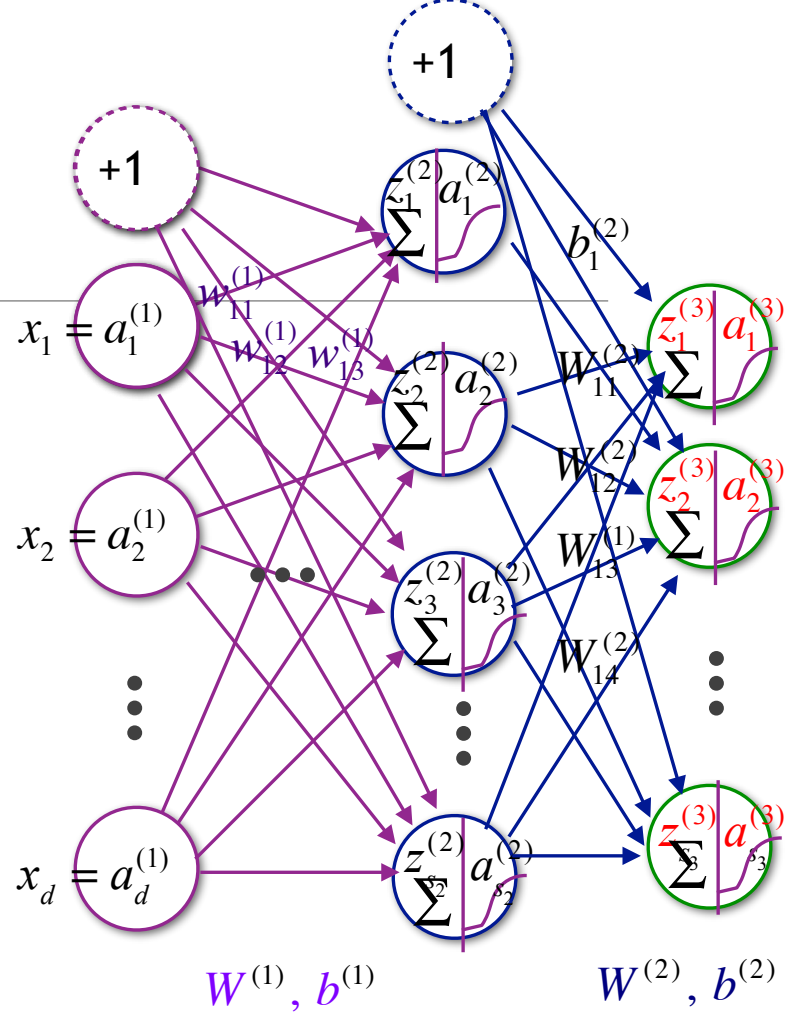
```
[[0., 0., 0.],   [0.,
 [0., 0., 0.],    0.,
 [0., 0., 0.],    0.,
 [0., 0., 0.]]    0.]
```

$\triangle W^{(1)}$ $\triangle b^{(1)}$

```
[[0., 0., 0., 0.],   [0.,
 [0., 0., 0., 0.],    0.,
 [0., 0., 0., 0.]]    0.]
```

$\triangle W^{(2)}$ $\triangle b^{(2)}$

```
def init_tri_values(nn_structure):
    tri_W = {}
    tri_b = {}
    for l in range(1, len(nn_structure)):
        tri_W[l] = np.zeros((nn_structure[l], nn_structure[l-1]))
        tri_b[l] = np.zeros((nn_structure[l],))
    return tri_W, tri_b
```



The gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$\Delta W^{(\ell)} = 0$

$\Delta b^{(\ell)} = 0$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level

$$\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Forward Propagation

nn_structure = [3, 4, 3]

- pseudocode:

$a^{(1)} = x$

for ℓ = 1 to L do

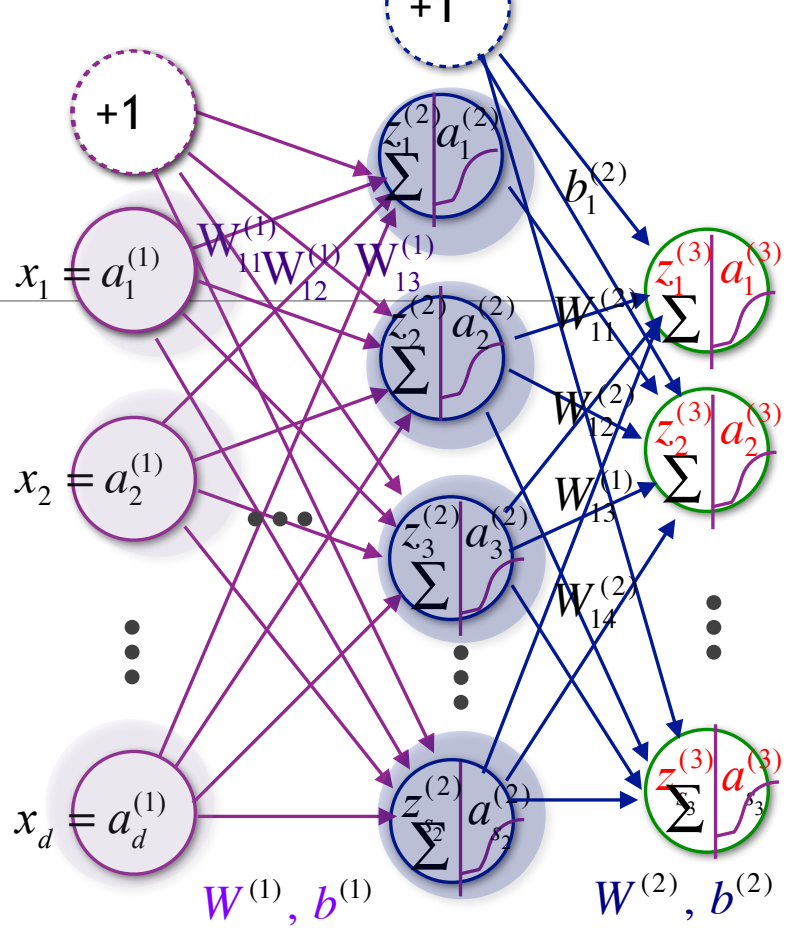
$z^{(\ell+1)} = W^{(\ell)} a^{(\ell)} + b^{(\ell)}$

$a^{(\ell+1)} = f(z^{(\ell+1)})$

$\hat{y} = a^{(n_L)}$

What is the dimension of $a^{(2)}$?

a) Vector of size 1
b) Vector of size 2
c) Vector of size 3
d) Vector of size 4
e) Vector of size 5
f) Vector of size 6



Forward Propagation

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)} \quad x = [1,\ 1,\ 1]^T$$

$$a^{(l+1)} = f(z^{(l+1)}) = f(W^{(l)} a^{(l)} + b^{(l)})$$

- pseudocode:

$a^{(1)} = x$

for ℓ = 1 to L do

$z^{(\ell+1)} = W^{(\ell)} a^{(\ell)} + b^{(\ell)}$

$a^{(\ell+1)} = f(z^{(\ell+1)})$

$\hat{y} = a^{(n_L)}$

$W^{(1)}\ a^{(1)} + b^{(1)} = z^{(2)}$

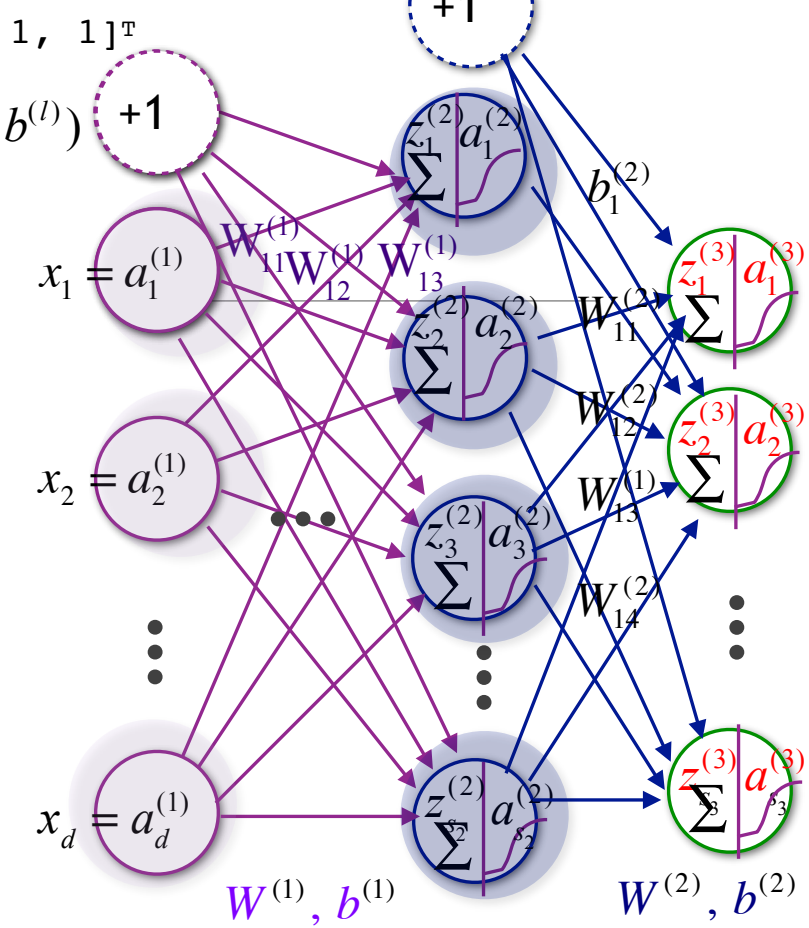
```
[[0.65, 0.18, 0.89],  [1      [0.07,    [1.80,
 [0.06, 0.74, 0.72],   1   +   0.35,  =   1.88,
 [0.92, 0.29, 0.71],   1]      0.60,      2.53,
 [0.39, 0.41, 0.10]]           0.50]      1.39]
```

$f(z^{(2)}) = a^{(2)}$

```
f( [1.80,    ) = [0.86,
     1.88,        0.87,
     2.53,        0.93,
     1.39]        0.80]
```

```
def f(x):
    return 1 / (1 + np.exp(-x))
```

```
def feed_forward(x, W, b):
    a = {1: x}
    z = { }
    for l in range(1, len(W) + 1): # for each layer
        node_in = a[l]
        z[l+1] = W[l].dot(node_in) + b[l]  #  z^(l+1) = W^(l) a^(l) + b^(l)
        a[l+1] = f(z[l+1])  #  a^(l+1) = f(z^(l+1)) = f(W^(l) a^(l) + b^(l))
    return a, z
```



Forward Propagation

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)}) = f(W^{(l)} a^{(l)} + b^{(l)})$$

- pseudocode:

$a^{(1)} = x$

for ℓ = 1 to L do

$z^{(\ell+1)} = W^{(\ell)} a^{(\ell)} + b^{(\ell)}$

$a^{(\ell+1)} = f(z^{(\ell+1)})$

$\hat{y} = a^{(n_L)}$

```
W(2)
 [[0.45, 0.64, 0.10, 0.82],
  [0.30, 0.25, 0.39, 0.36],
  [0.78, 0.60, 0.16, 0.07]]
```

$W^{(2)} a^{(2)} + b^{(2)} = z^{(3)}$

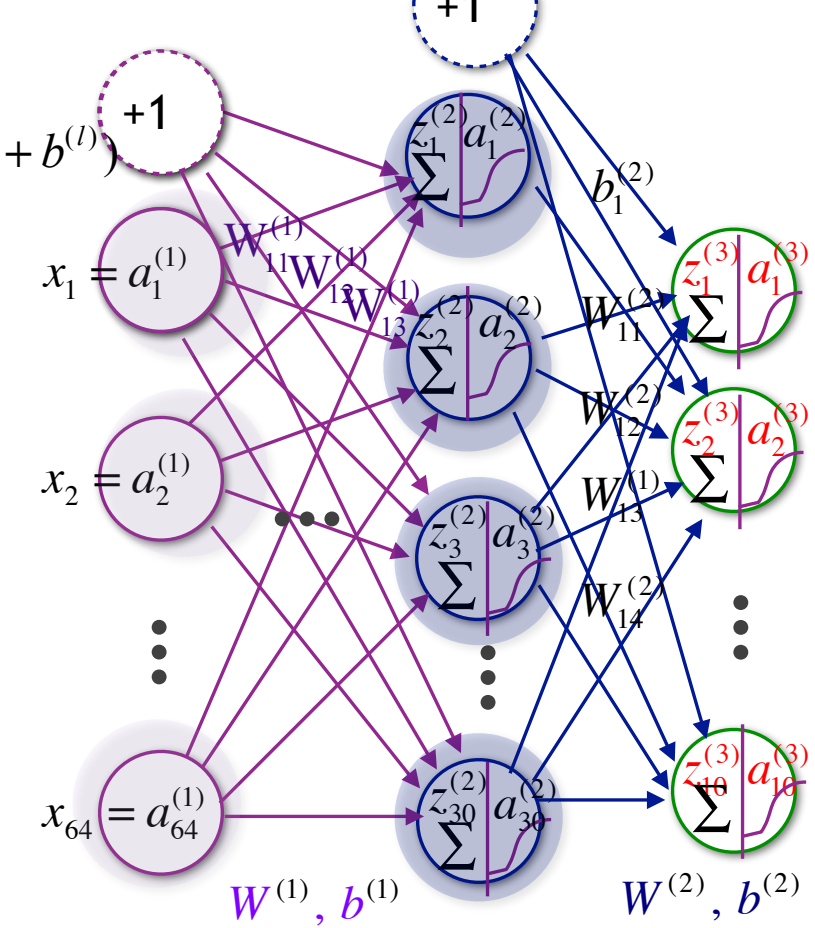
```
[0.86,   0.14,     [2.66,
 0.87,   0.64,  =   1.76,
 0.93,   0.62       2.01]
 0.80]
```

$f(z^{(3)}) = a^{(3)}$

```
f( [2.66,  )  =  [0.93
   1.76,          0.85
   2.01]          0.88 ]
```

```
def f(x):
    return 1 / (1 + np.exp(-x))
```

```
def feed_forward(x, W, b):
    a = {1: x}
    z = { }
    for l in range(1, len(W) + 1): # for each layer
        node_in = a[l]
        z[l+1] = W[l].dot(node_in) + b[l]  #  z^(l+1) = W^(l) a^(l) + b^(l)
        a[l+1] = f(z[l+1])  #  a^(l+1) = f(z^(l+1)) = f(W^(l) a^(l) + b^(l))
    return a, z
```



The gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$$\Delta W^{(\ell)} = 0$$

$$\Delta b^{(\ell)} = 0$$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through nℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level $\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)}\left(a^{(\ell)}\right)^T$

$$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Definitions:

$$\frac{\partial J}{\partial W^{(\ell)}} = \delta^{(\ell+1)}\left(a^{(\ell)}\right)^T$$

$$\frac{\partial J}{\partial b^{(\ell)}} = \delta^{(\ell+1)}$$

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}}$$

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{\partial J}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

Calculating δ

$$\delta^{(n_\ell)} = -(y - a^{(n_\ell)}) \bullet f'(z^{(n_\ell)})$$

$$\delta^{(\ell)} = \left(\left(W^{(\ell)} \right)^T \delta^{(\ell+1)} \right) \bullet f'(z^{(\ell)})$$

```
def f_deriv(x):
    return f(x) * (1 - f(x))
```

```
def calculate_out_layer_delta(y, a_out, z_out):
    return -(y-a_out) * f_deriv(z_out)
```

$$-\left(y - a^{(3)}\right) \bullet f'(z^{(3)})$$

$$-\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0.935 \\ 0.854 \\ 0.882 \end{pmatrix} \bullet f'\begin{pmatrix} 2.66, \\ 1.76, \\ 2.01 \end{pmatrix} = -\begin{pmatrix} 0.065 \\ 0.146 \\ 0.118 \end{pmatrix} \bullet \begin{pmatrix} 0.06087702 \\ 0.12500965 \\ 0.10390214 \end{pmatrix} = \delta^{(3)} \begin{matrix} -0.00396415 \\ -0.01830894 \\ -0.01223682 \end{matrix}$$

Calculating δ

$$\delta^{(n_\ell)} = -(y - a^{(n_\ell)}) \bullet f'(z^{(n_\ell)})$$

$$\delta^{(\ell)} = \left(\left(W^{(\ell)} \right)^T \delta^{(\ell+1)} \right) \bullet f'(z^{(\ell)})$$

```
def f_deriv(x):
    return f(x) * (1 - f(x))
```

```
def calculate_hidden_delta(delta_plus_1, w_l, z_l):
   return np.dot(np.transpose(w_l), delta_plus_1) * f_deriv(z_l)
```

$$\left(W^{(2)T} \quad \delta^{(3)} \right) \bullet \quad f'(z^{(2)}) \quad = \left(W^{(2)T} \delta^{(3)} \right) \quad f'(z^{(2)}) \quad = \quad \delta^{(2)}$$

$$\left(\left(\begin{bmatrix} 0.453 & 0.304 & 0.776 \\ 0.637 & 0.246 & 0.600 \\ 0.996 & 0.390 & 0.164 \\ 0.820 & 0.361 & 0.070 \end{bmatrix} \begin{pmatrix} -0.004 \\ -0.018 \\ -0.012 \end{pmatrix} \right) \bullet f' \begin{pmatrix} 1.798 \\ 1.875 \\ 2.530 \\ 1.389 \end{pmatrix} = \begin{pmatrix} -0.017 \\ -0.014 \\ -0.013 \\ -0.011 \end{pmatrix} \bullet \begin{pmatrix} 0.122 \\ 0.115 \\ 0.068 \\ 0.160 \end{pmatrix} = \begin{pmatrix} -0.002 \\ -0.002 \\ -0.001 \\ -0.002 \end{pmatrix}$$

The gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$\Delta W^{(\ell)} = 0$

$\Delta b^{(\ell)} = 0$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level

$$\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Definitions:

$$\frac{\partial J}{\partial W^{(\ell)}} = \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\frac{\partial J}{\partial b^{(\ell)}} = \delta^{(\ell+1)}$$

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}}$$

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{\partial J}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

Computing

$$\delta^{(n_\ell)} = -(y - a^{(n_\ell)}) \bullet f'(z^{(n_\ell)}) \qquad \Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

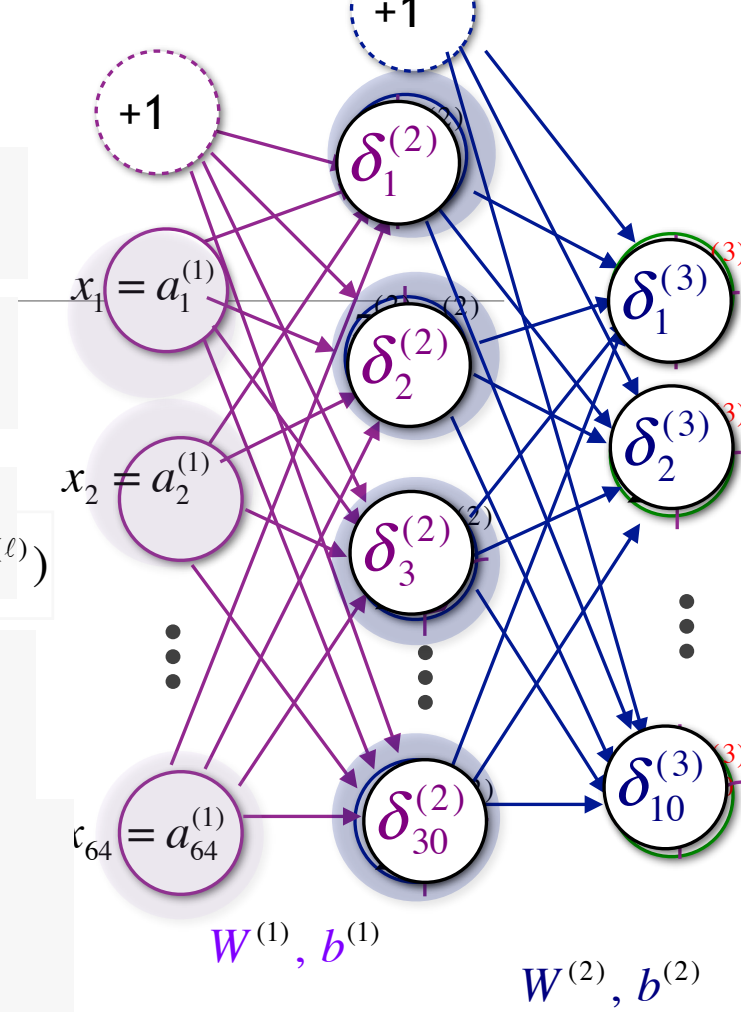
$$\delta^{(\ell)} = \left(\left(W^{(\ell)}\right)^T \delta^{(\ell+1)}\right) \bullet f'(z^{(\ell)}) \qquad \Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

```
def f_deriv(x):
    return f(x) * (1 - f(x))
```

```
def calculate_out_layer_delta(y, a_out, z_out):
    return -(y-a_out) * f_deriv(z_out)   # = -(y - a^{n_ℓ}) • f'(z^{n_ℓ})
```

```
def calculate_hidden_delta(delta_plus_1, w_l, z_l):
    return np.dot(np.transpose(w_l), delta_plus_1) * f_deriv(z_l)   # = ((W^(ℓ))^T δ^(ℓ+1)) • f'(z^(ℓ))
```

```
for i in range(len(y)):
    delta = {}
    a, z = feed_forward(X[i, :], W, b)
    for l in range(len(nn_structure), 0, -1):
        if l == len(nn_structure):
            delta[l] = calculate_out_layer_delta(y[i,:], a[l], z[l])
        else:
            if l > 1:
                delta[l] = calculate_hidden_delta(delta[l+1], W[l], z[l])
    tri_W[l] += np.dot(delta[l+1][:,np.newaxis], np.transpose(a[l][:,np.newaxis]))   # ΔW^(ℓ) + δ^(ℓ+1) (a^(ℓ))^T
    tri_b[l] += delta[l+1]   # Δb^(ℓ) + δ^(ℓ+1)
```



The gradient descent algorithm

Algorithm and code adapted from http://adventuresinmachinelearning.com/neural-networks-tutorial/

Randomly initialize the weights for each layer: $W^{(\ell)}$

While iterations < iteration limit:

$\Delta W^{(\ell)} = 0$

$\Delta b^{(\ell)} = 0$

For i = 1 to N:

run forward propagation and save for each level, the values $a^{(\ell)}$ $\mathbf{z}^{(\ell)}$

run back-propagation to calculate $\delta^{(\ell)}$ values for levels 2 through n_ℓ

Update $\Delta W^{(\ell)}$ $\Delta b^{(\ell)}$ for each level

$$\Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

Perform a gradient descent step

$$W^{(\ell)} = W^{(\ell)} - \alpha \frac{1}{N} \Delta W^{(\ell)}$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \frac{1}{N} \Delta b^{(\ell)}$$

Definitions:

$$\frac{\partial J}{\partial W^{(\ell)}} = \delta^{(\ell+1)} \left(a^{(\ell)}\right)^T$$

$$\frac{\partial J}{\partial b^{(\ell)}} = \delta^{(\ell+1)}$$

$$\delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}}$$

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell+1)} a_j^{(\ell)}$$

$$\frac{\partial J}{\partial b_i^{(\ell)}} = \delta_i^{(\ell+1)}$$

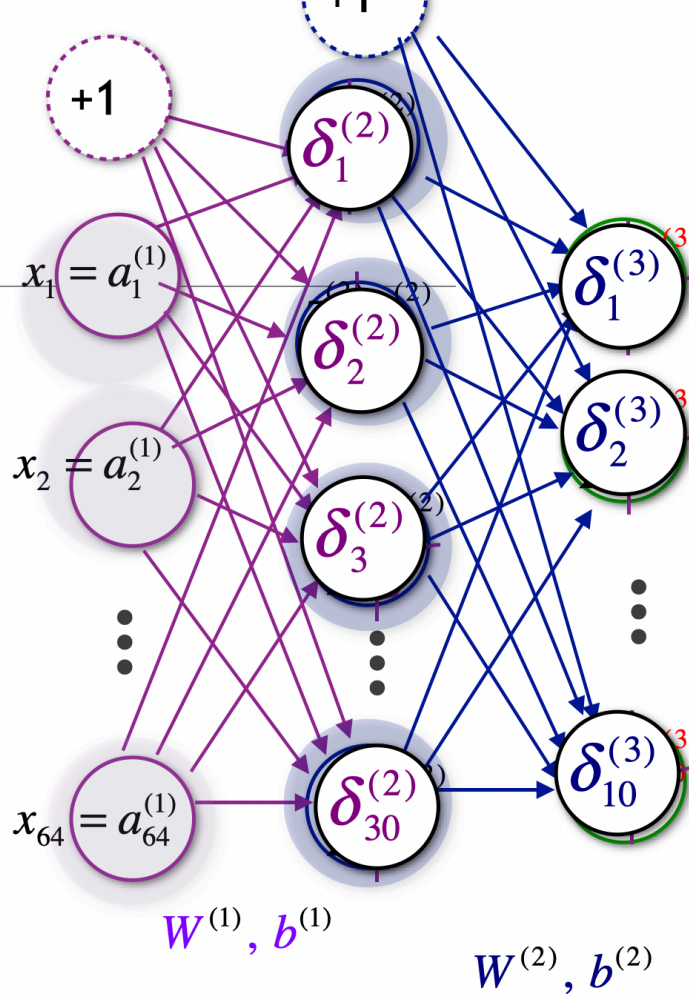
Computing

$$\delta^{(n_\ell)} = -(y - a^{(n_\ell)}) \bullet f'(z^{(n_\ell)}) \qquad \Delta W^{(\ell)} = \Delta W^{(\ell)} + \delta^{(\ell+1)} \left(a^{(\ell)}\right)$$

$$\delta^{(\ell)} = \left(\left(W^{(\ell)}\right)^T \delta^{(\ell+1)}\right) \bullet f'(z^{(\ell)}) \qquad \Delta b^{(\ell)} = \Delta b^{(\ell)} + \delta^{(\ell+1)}$$

```
while cnt < iter_num:
    tri_W, tri_b = init_tri_values(nn_structure)

    for i in range(len(y)):
        delta = {}
        a, z = feed_forward(X[i, :], W, b)
        for l in range(len(nn_structure), 0, -1):
            if l == len(nn_structure):
                delta[l] = calculate_out_layer_delta(y[i,:], a[l], z[l])
            else:
                if l > 1:
                    delta[l] = calculate_hidden_delta(delta[l+1], W[l], z[l])
            tri_W[l] += np.dot(delta[l+1][:,np.newaxis], np.transpose(a[l][:,np.newaxis]))
            tri_b[l] += delta[l+1] # Δb^(ℓ) + δ^(ℓ+1)
```

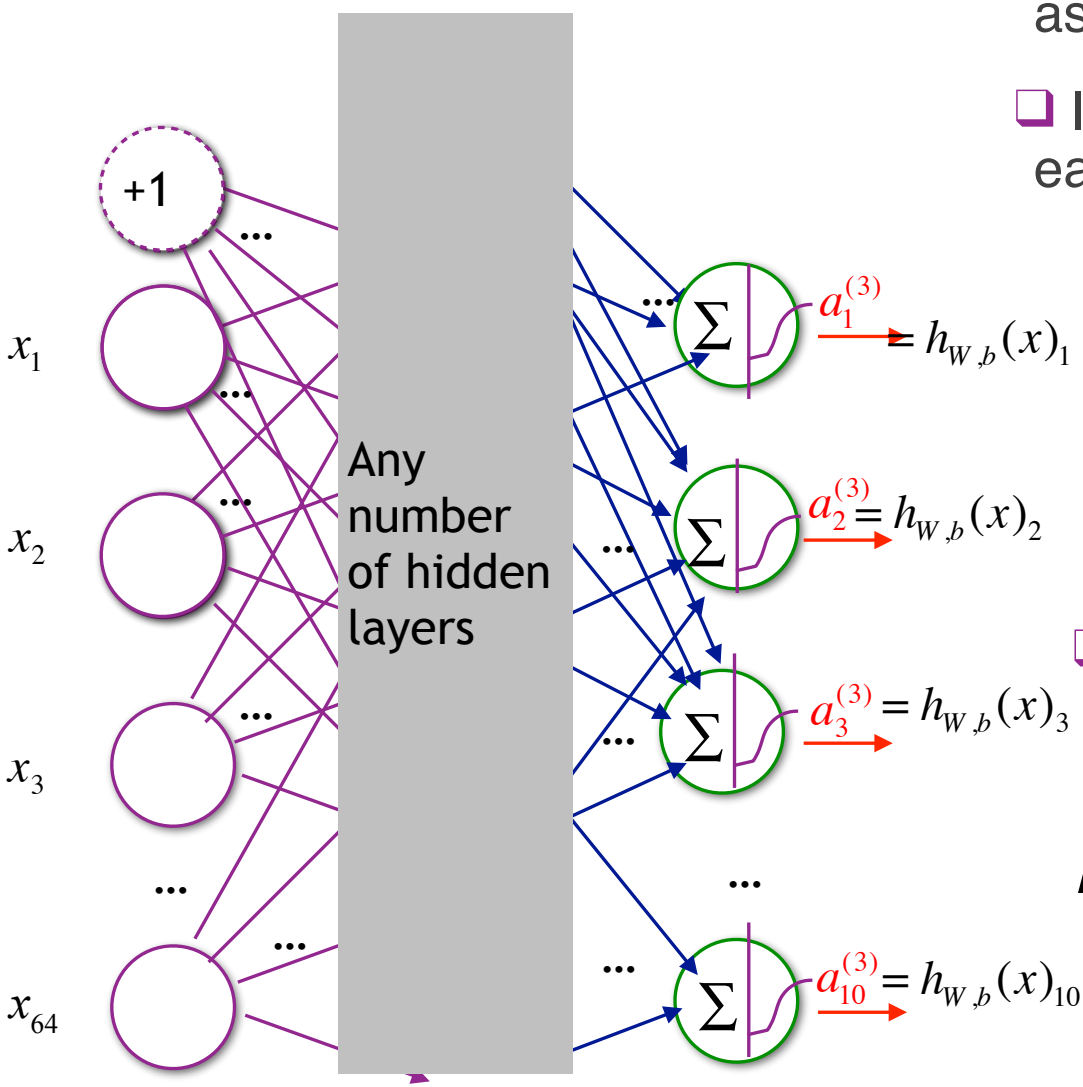
```
# perform the gradient descent step for the weights in each layer
for l in range(len(nn_structure) - 1, 0, -1):
    W[l] += -alpha * (1.0/N * tri_W[l])
    b[l] += -alpha * (1.0/N * tri_b[l])


cnt += 1
```



Outline

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

Multiple Output Units: One-vs-All



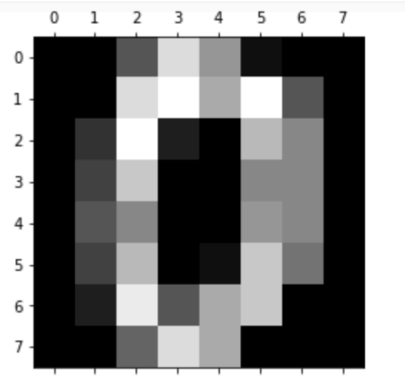
- Instead of just computing one output, we can compute as many as we would like
- If we were classifying digits, we could have ten outputs, one for each digit - using a 1 of k representation (aka one-hot encoding)

$$y = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ If } y = 0 \qquad y = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \text{ If } y = 1 \quad \cdots \qquad y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \text{ If } y = 9$$

- For our hypothesis $h_{W,b}(x) \in \mathbb{R}^K$ we want:

$$h_{W,b}(x) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad h_{W,b}(x) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad h_{W,b}(x) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Pre-processing steps



- One-vs-all

```
def convert_y_to_vect(y):
    y_vect = np.zeros((len(y), 10))
    for i in range(len(y)):
        y_vect[i, y[i]] = 1
    return y_vect
```

```
[9
 9
 4
 3
 …]

[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
                                …]]
```

For each feature. Find the mean and standard deviation of the feature and then subtract the mean and divide by the standard deviation.
x'=(x – mean) / standard_deviation

Mean of feature 1 is 0
Mean of feature 2 is 3

- Scale the date set

```
X_scale = StandardScaler()
X = X_scale.fit_transform(digits.data)
```

```
0.   0.   5. 13.   9.   1.   0.   0.
0.   0. 13. 15. 10. 15.   5.   0.
0.   3. 15.   2.   0. 11.   8.   0.
0.   4. 12.   0.   0.   8.   8.   0.
0.   5.   8.   0.   0.   9.   8.   0.
0.   4. 11.   0.   1. 12.   7.   0.
0.   2. 14.   5. 10. 12.   0.   0.
0.   0.   6. 13. 10.   0.   0.   0.

 0.    -0.335 -0.043  0.274 -0.664 -0.844 -0.41  -0.125
-0.059 -0.624  0.483  0.76  -0.058  1.128  0.88  -0.13
-0.045  0.111  0.896 -0.861 -1.15   0.515  1.906 -0.114
-0.033  0.486  0.47  -1.5   -1.614  0.076  1.542 -0.047
 0.     0.765  0.053 -1.448 -1.737  0.044  1.44   0.
-0.061  0.811  0.63  -1.122 -1.066  0.661  0.818 -0.089
-0.035  0.742  1.151 -0.869  0.11   0.538 -0.757 -0.21
-0.024 -0.299  0.087  0.208 -0.367 -1.147 -0.506 -0.196
```