

Topic 6

Neural Networks Cont

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

INTRODUCTION TO MACHINE LEARNING

PROF. LINDA SELLIE

Some of these slides are from Prof. Rangan

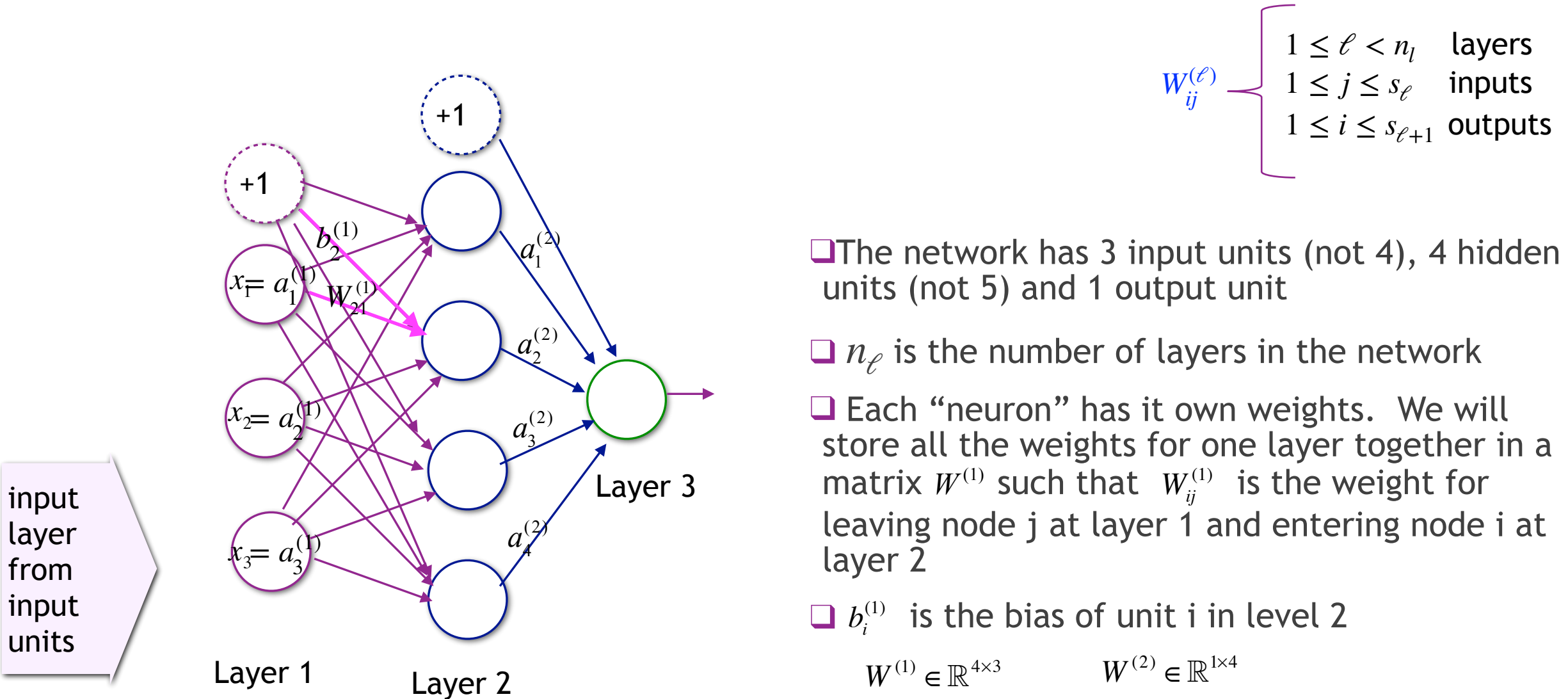
Overview

Used for both regression and classification

Neural networks *extend* logistic regression and linear regression

Neural networks are universal approximators (it is possible to approximate any bounded function)

Neural Network Model & Notation



□ The network has 3 input units (not 4), 4 hidden units (not 5) and 1 output unit

□ n_ℓ is the number of layers in the network

□ Each “neuron” has its own weights. We will store all the weights for one layer together in a matrix $W^{(1)}$ such that $W_{ij}^{(1)}$ is the weight for leaving node j at layer 1 and entering node i at layer 2

□ $b_i^{(1)}$ is the bias of unit i in level 2

$$W^{(1)} \in \mathbb{R}^{4 \times 3}$$

$$W^{(2)} \in \mathbb{R}^{1 \times 4}$$

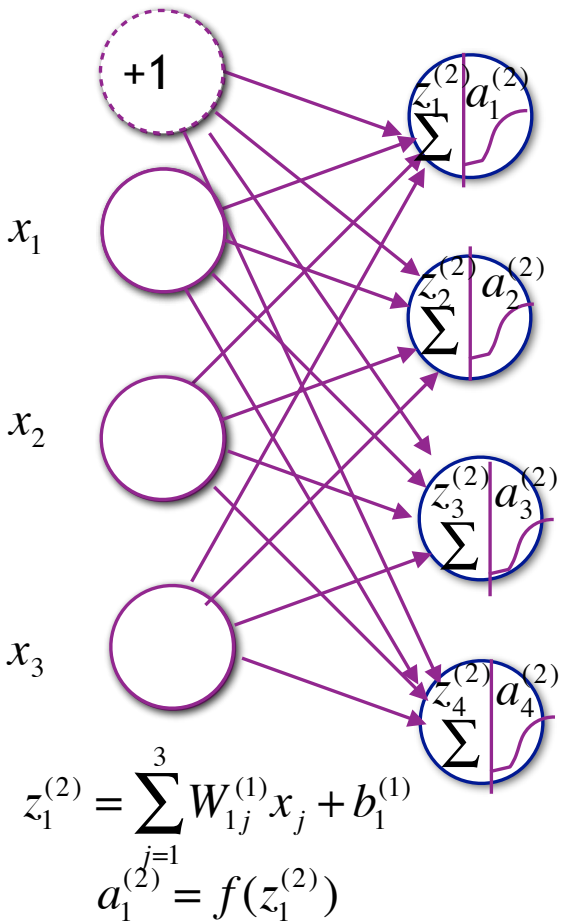
Let s_j be the number of nodes at layer j where we don't include the bias unit

Neural Network Model & Notation Σ

$$z_i^{(2)} = \sum_{j=1}^3 W_{ij}^{(1)} x_j + b_i^{(1)} \quad a_i^{(2)} = f(z_i^{(2)})$$

$$z^{(2)} = W^{(1)}x + b^{(1)} \quad f(z) = \frac{1}{1 + \exp(-z)}$$

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



$$\begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} \\ W_{41}^{(1)} & W_{42}^{(1)} & W_{43}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix} = \begin{bmatrix} W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 \\ W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 \\ W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 \\ W_{41}^{(1)} x_1 + W_{42}^{(1)} x_2 + W_{43}^{(1)} x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)} \\ W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)} \\ W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)} \\ W_{41}^{(1)} x_1 + W_{42}^{(1)} x_2 + W_{43}^{(1)} x_3 + b_4^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_4^{(2)} \end{bmatrix}$$

$$a^{(2)} = \begin{bmatrix} f(z_1^{(2)}) \\ f(z_2^{(2)}) \\ f(z_3^{(2)}) \\ f(z_4^{(2)}) \end{bmatrix} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \end{bmatrix}$$

$$z_2^{(2)} = \sum_{j=1}^3 W_{2j}^{(1)} x_j + b_2^{(1)}$$

$$a_2^{(2)} = f(z_2^{(2)})$$

$$z_3^{(2)} = \sum_{j=1}^3 W_{3j}^{(1)} x_j + b_3^{(1)}$$

$$a_3^{(2)} = f(z_3^{(2)})$$

$$z_4^{(2)} = \sum_{j=1}^3 W_{4j}^{(1)} x_j + b_4^{(1)}$$

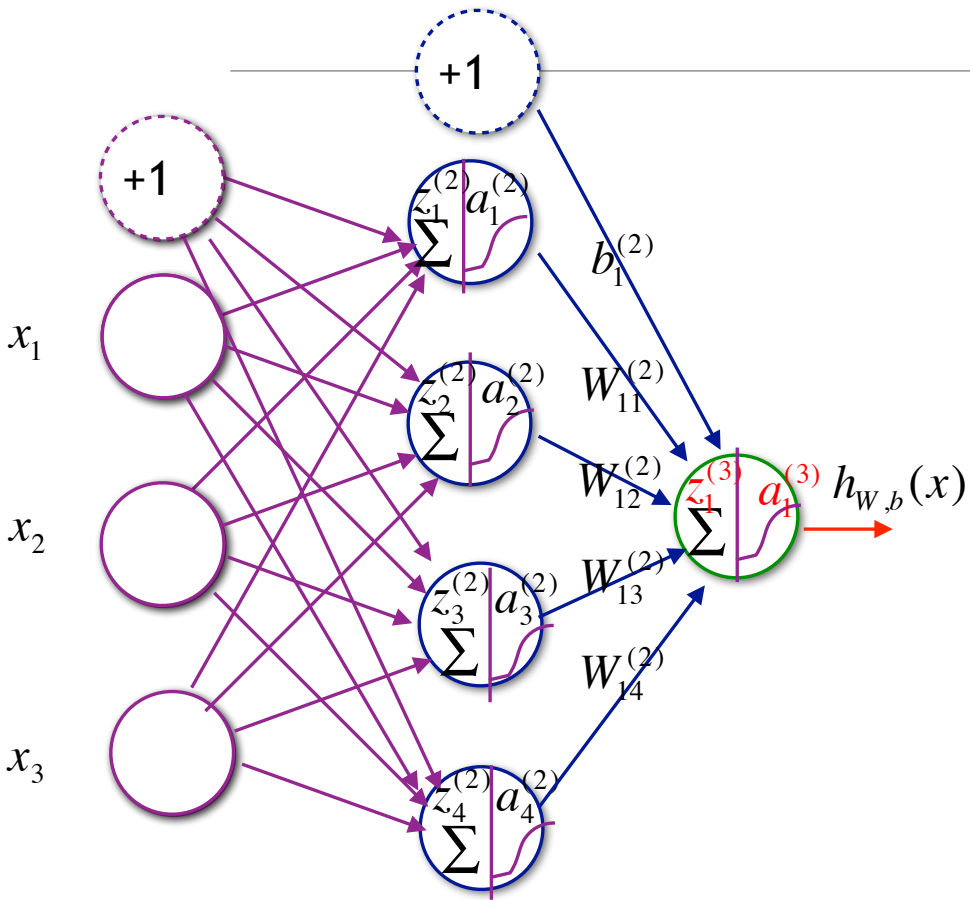
$$a_4^{(2)} = f(z_4^{(2)})$$

Neural Network Model & Notation Σ

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

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

$$f(z) = \frac{1}{1 + \exp(-z)}$$



$$\begin{bmatrix} W_{11}^{(2)} & W_{12}^{(2)} & W_{13}^{(2)} & W_{14}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \end{bmatrix} + [b_1^{(2)}] = [z_1^{(3)}]$$

$$\mathbf{a}^{(3)} = [f(z_1^{(2)})] = [a_1^{(3)}]$$

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

$$\mathbf{a}^{(3)} = f(\mathbf{z}^{(3)}) = f(W^{(2)}\mathbf{x} + b^{(2)})$$

$$z_1^{(3)} = \sum_{j=1}^4 W_{1j}^{(2)} a_j^{(2)} + b_1^{(2)}$$

$$a_1^{(3)} = f(z_1^{(3)})$$

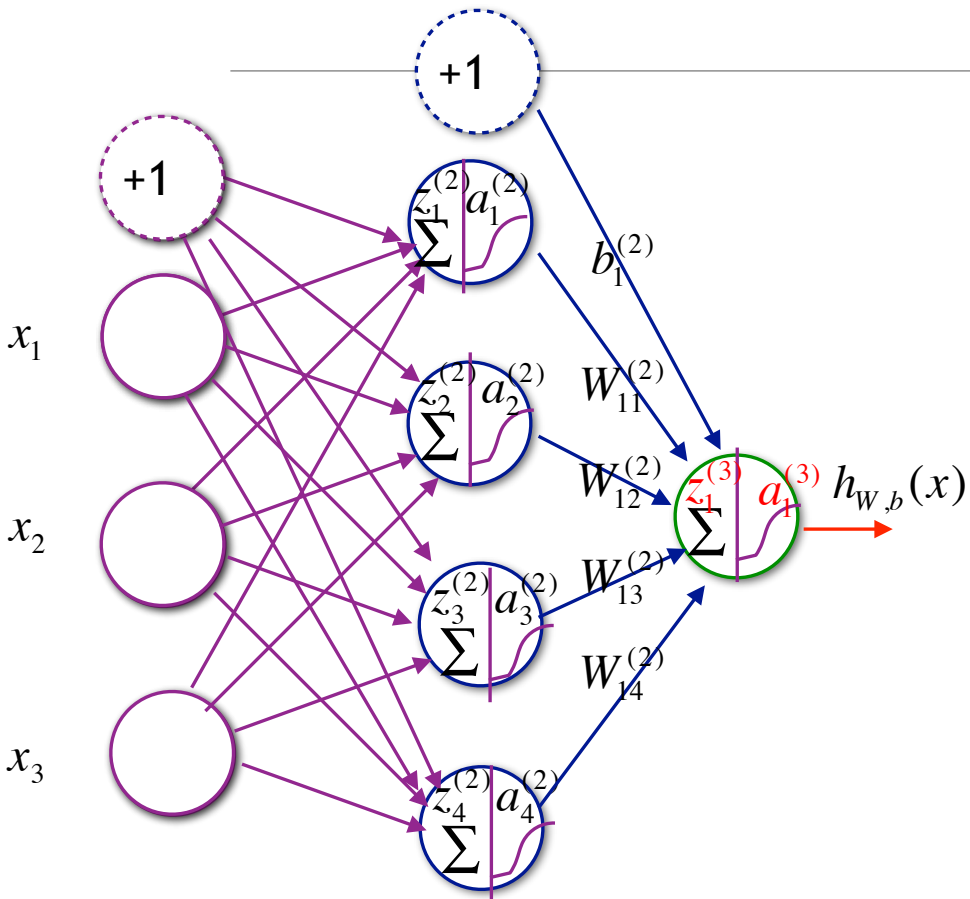
$$h_{W,b} = \mathbf{a}^{(3)} = f(\mathbf{z}^{(3)})$$

Neural Network Model & Notation Σ

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

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

$$f(z) = \frac{1}{1 + \exp(-z)}$$



What is $a_3^{(2)}$?

- a) One of the inputs to neurons on layer 2
- b) One of the inputs to neurons on layer 3
- c) activation value of 3rd neuron on layer 2
- d) activation value of 2nd neuron on layer 3
- e) activation value of 3rd neuron on layer 3
- f) activation value of 2nd neuron on layer 3

$$h_{W,b} = \mathbf{a}^{(3)} = f(\mathbf{z}^{(3)})$$

Artificial Neural Networks

Each node (we call the nodes **neurons**) represents a linear function of its input, similar to logistic regression

$$z_1^{(2)} = W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}$$

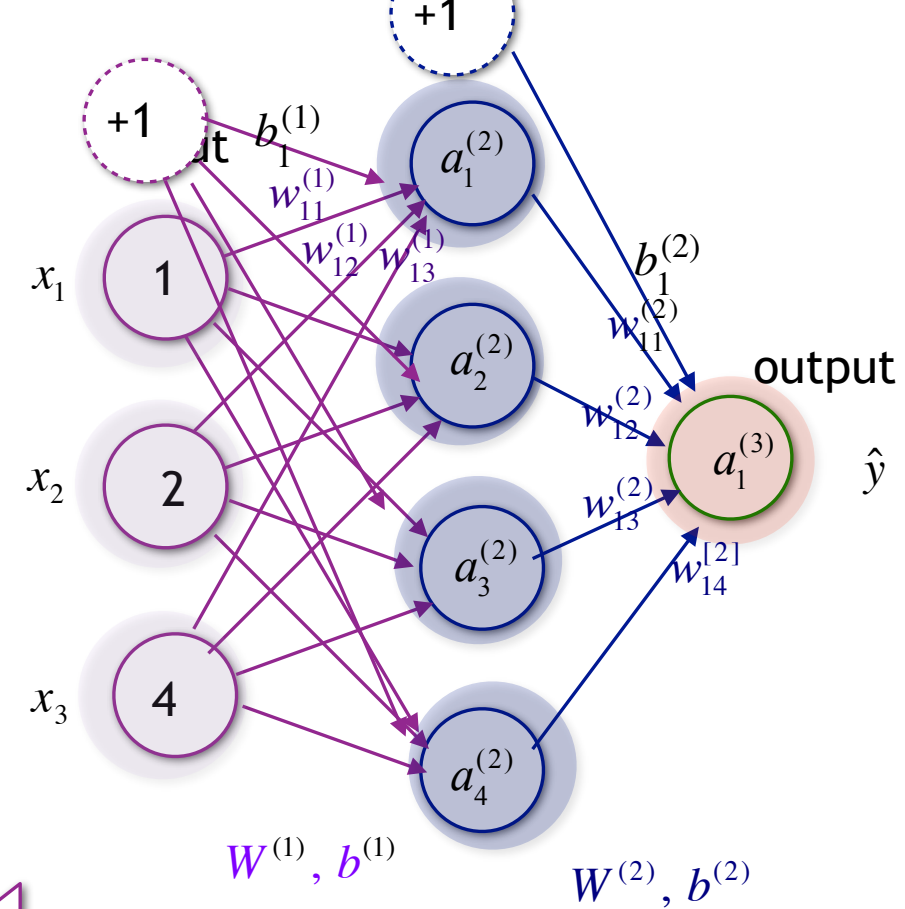
$$a_1^{(2)} = f(z_1^{(2)})$$

- $W_{11}^{(1)} W_{12}^{(1)} W_{13}^{(1)} b_1^{(1)}$ are the **weights** (aka parameters)
- $W^{(1)}$ is the matrix of weights that map values from input layer 1 to the first hidden layer (layer 2) ($W^{(\ell)}$ is the matrix that maps layer ℓ to layer $\ell+1$)
- $b^{(1)}$ is the **vector** of bias values for the neurons on layer 1

$f^{(1)}$ is the “activation function” for layer 1. ($f^{(\ell)}$ is the activation function for layer ℓ)

$a_j^{(2)}$ is the activation value for the j^{th} neuron of layer 2
 $a_j^{(\ell)}$ is the activation value for the j^{th} neuron of layer the ℓ

Let s_j is the number of nodes at layer j where we don't include the bias unit




One example of an activation function is the logistic sigmoid

$$\begin{bmatrix} W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} \\ W_{41}^{(1)} & W_{42}^{(1)} & W_{43}^{(1)} \end{bmatrix} \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} W_{11}^{(2)} & W_{12}^{(2)} & W_{13}^{(2)} & W_{14}^{(2)} \end{bmatrix} \quad \mathbf{b}^{(2)} = \begin{bmatrix} b_1^{(2)} \end{bmatrix}$$

Outline

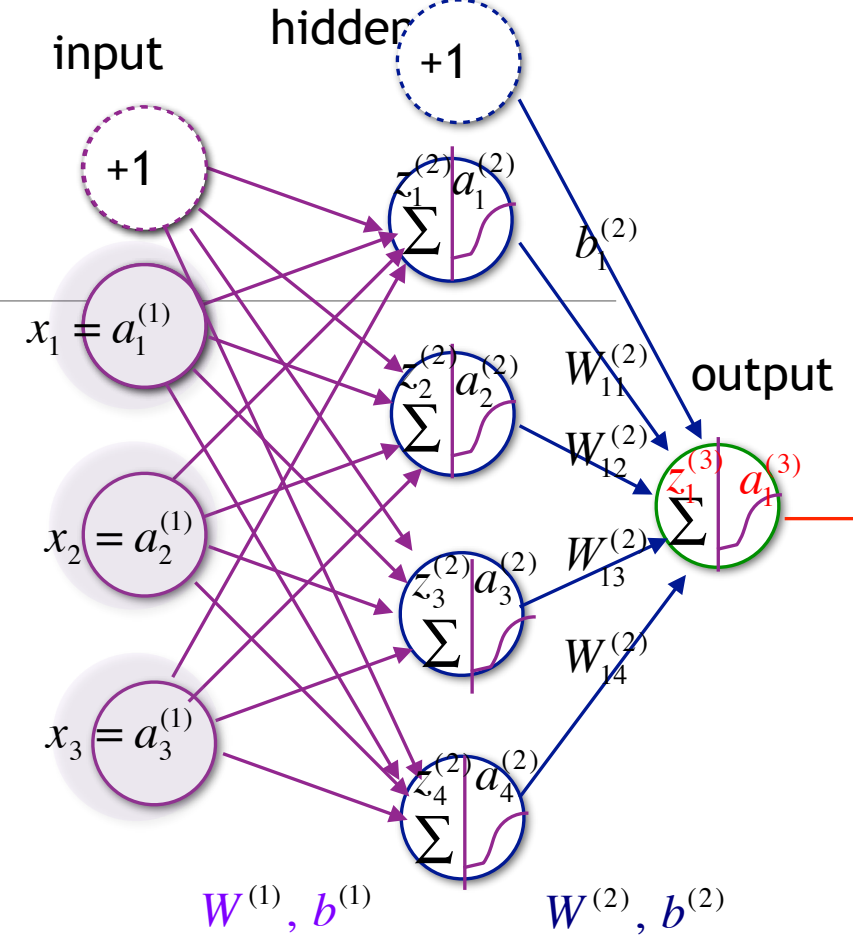
- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
 - Vectorization
 - Pseudocode
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

$f(z) = \frac{1}{1 + \exp(-z)}$ Forward Propagation

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

□ Forward propagation:

$\mathbf{a}^{(1)} = \mathbf{x}$
for $\ell = 1$ to $n_\ell - 1$ do
 $\mathbf{z}^{(\ell+1)} = \mathbf{W}^{(\ell)} \mathbf{a}^{(\ell)} + \mathbf{b}^{(\ell)}$
 $\mathbf{a}^{(\ell+1)} = f(\mathbf{z}^{(\ell+1)})$
 $\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}$



Push weighted predictions through the network.

$f(z) = \frac{1}{1 + \exp(-z)}$ Forward Propagation

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

Generalizing for an arbitrary neural network. Forward propagation:

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

for $\ell = 1$ to $n_\ell - 1$ do

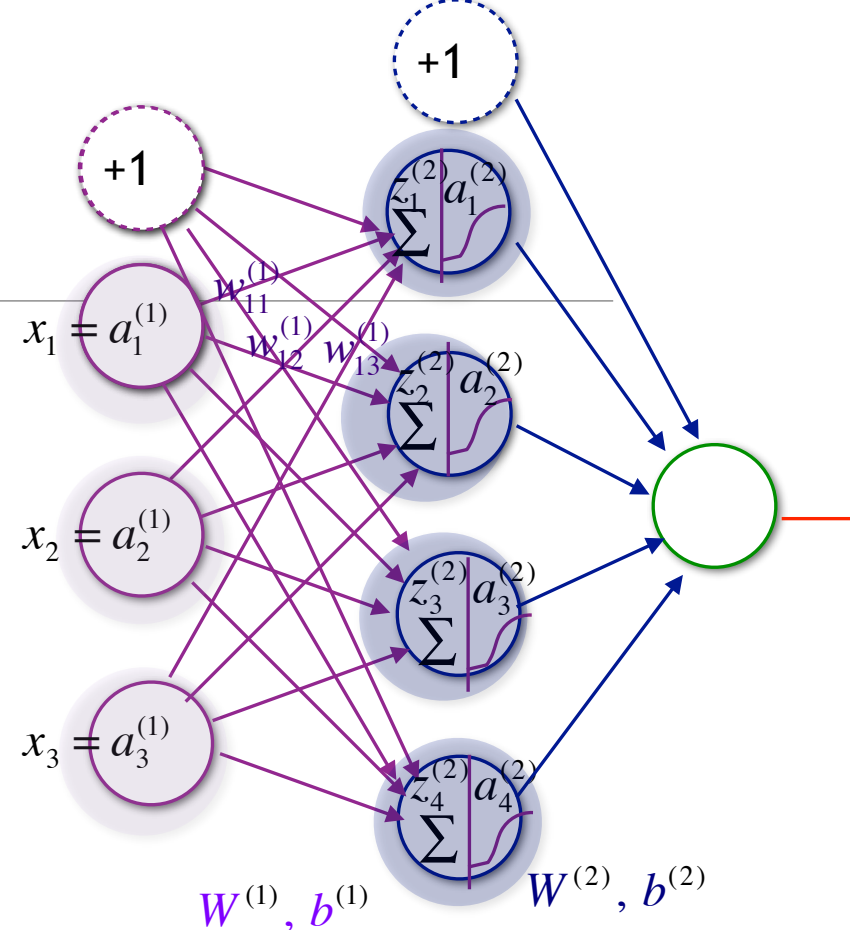
$$\mathbf{z}^{(\ell+1)} = W^{(\ell)}\mathbf{a}^{(\ell)} + \mathbf{b}^{(\ell)} = \mathbf{z}^{(2)}$$

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

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

$$\begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} \\ W_{41}^{(1)} & W_{42}^{(1)} & W_{43}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_4^{(2)} \end{bmatrix}$$

$$\mathbf{a}^{(2)} = \begin{bmatrix} f(z_1^{(2)}) \\ f(z_2^{(2)}) \\ f(z_3^{(2)}) \\ f(z_4^{(2)}) \end{bmatrix} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \end{bmatrix}$$



$f(z) = \frac{1}{1 + \exp(-z)}$ Forward Propagation

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

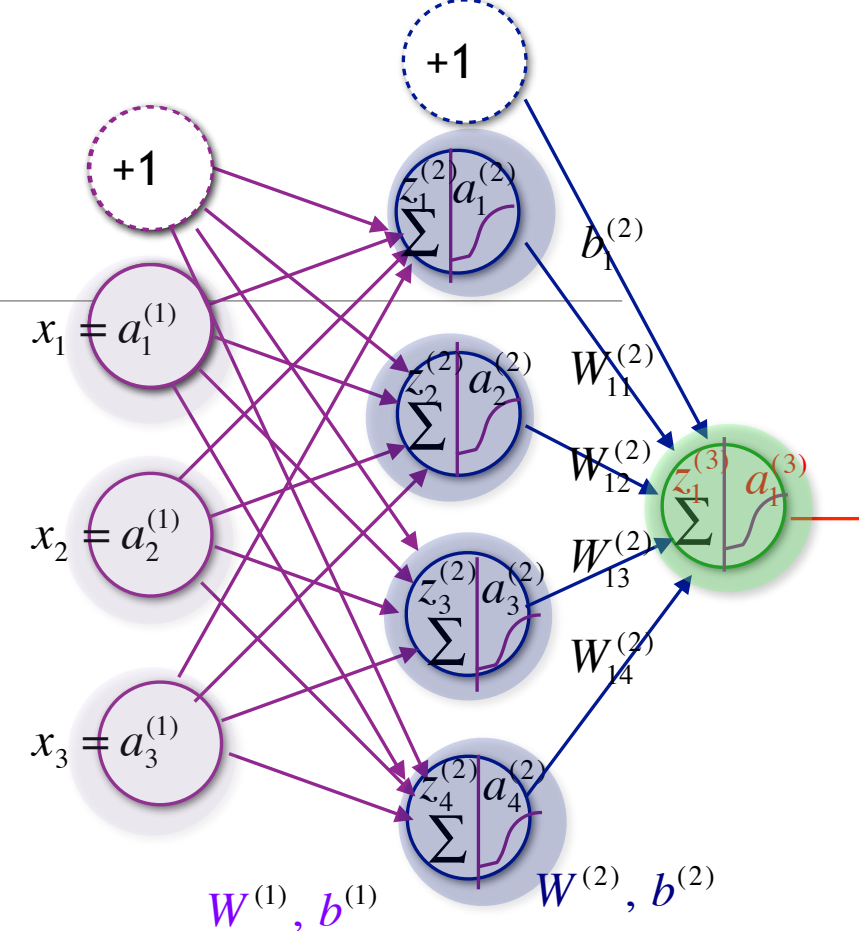
Generalizing for an arbitrary neural network. Forward propagation:

$\mathbf{a}^{(1)} = \mathbf{x}$
 for $\ell = 1$ to $n_\ell - 1$ do
 $\mathbf{z}^{(\ell+1)} = W^{(\ell)} \mathbf{a}^{(\ell)} + \mathbf{b}^{(\ell)}$
 $\mathbf{a}^{(\ell+1)} = f(\mathbf{z}^{(\ell+1)})$
 $\hat{\mathbf{y}} = \mathbf{a}^{(n_\ell)}$

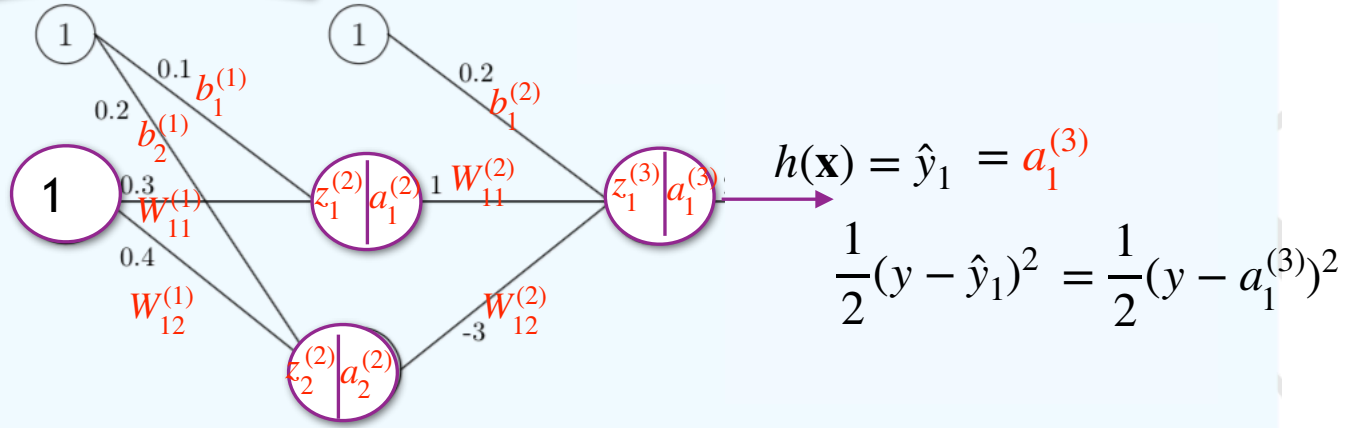
$$\mathbf{z}^{(3)} = \begin{bmatrix} W_{11}^{(2)} & W_{12}^{(2)} & W_{13}^{(2)} & W_{14}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \end{bmatrix} + [b_1^{(2)}] = [z_1^{(3)}]$$

$$\mathbf{a}^{(3)} = [f(\mathbf{z}^{(2)})] = [a_1^{(3)}]$$

$$a^{(3)} = f(W^{(2)} f(W^{(1)} a^{(1)} + b^{(1)}) + b^{(2)})$$



Calculations
done with rounded
numbers...



$$W^{(1)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} 1 & -3 \end{bmatrix}$$

$$b^{(2)} = \begin{bmatrix} 0.2 \end{bmatrix}$$

If $\mathbf{x} = 1$ and $y = 1$, forward propagation:

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

$\underbrace{\quad \quad \quad}_{W^{(1)}a^{(1)} + b^{(1)}} \quad \underbrace{\quad \quad \quad}_{f(z^{(2)})} \quad \underbrace{\quad \quad \quad}_{W^{(2)}a^{(2)} + b^{(2)}} \quad \underbrace{\quad \quad \quad}_{f(z^{(3)})}$

$\underbrace{\quad \quad \quad}_{f(W^{(1)}a^{(1)} + b^{(1)})} \quad \underbrace{\quad \quad \quad}_{f(W^{(2)}a^{(2)} + b^{(2)})}$

$\underbrace{\quad \quad \quad}_{f(W^{(2)}f(W^{(1)}a^{(1)} + b^{(1)}) + b^{(2)})}$

$$J(W, b, \mathbf{x}, y) = \frac{1}{2}(y - a_1^{(3)})^2 = \frac{1}{2} \left(y - f \left(W^{(2)} f(W^{(1)} a^{(1)} + b^{(1)}) + b^{(2)} \right) \right)^2$$

What if we didn't use an activation function?

□ Linear model... $z^{(3)} = a^{(3)} = \cancel{W^{(2)}} \cancel{W^{(1)}} a^{(1)} + b^{(1)} + b^{(2)}$

To simplify the notation: Assume $b^{(i)} = 0$

$$z^{(2)} = W^{(1)} \mathbf{x}$$

$$z^{(3)} = W^{(2)} z^{(2)} = W^{(2)} W^{(1)} \mathbf{x}$$

If we added another layer


$$z^{(4)} = a^{(4)} = \cancel{W^{(3)}} \cancel{W^{(2)}} \cancel{W^{(1)}} a^{(1)} + b^{(1)} + b^{(2)} + b^{(3)}$$

$$\tilde{W} = W^{(3)} W^{(2)} W^{(1)}$$

$$z^{(4)} = W^{(3)} a^{(3)} = W^{(3)} W^{(2)} a^{(2)} = W^{(3)} W^{(2)} W^{(1)} \mathbf{x} = \tilde{W} \mathbf{x}$$



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

How do we train a neural network?

We will use the values we computed in the forward pass

Alternatively, choose a different activation function on the last layer. We are keeping things simple in this lecture.

ERROR BACK-PROPAGATION:

FIRST WE DO A FORWARD PROPAGATION AND COMPUTE THE z AND a VALUES THEN WE WORK BACKWARD AND COMPUTE THE GRADIENT OF THE LOSS WITH RESPECT TO THE WEIGHTS

SINCE ARE ARE USING THE SIGMOID ACTIVATION ON THE LAST LAYER, PREPROCESS THE TARGET VALUES TO BE IN $[0,1]$

Error/loss function

This lecture:

Squared error for single output: $J(W, b, \mathbf{x}, y) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - a^{(n_\ell)})^2$

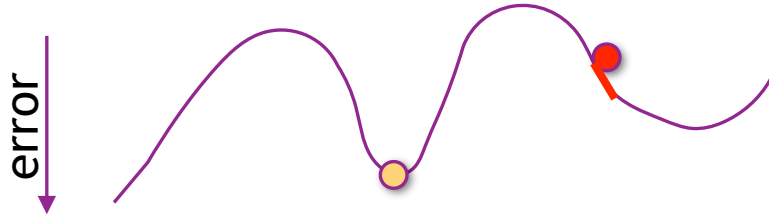
Many Alternatives, e.g:

Squared error for multiple outputs $J(W, b, \mathbf{x}, y) = \frac{1}{2} \sum_{k \in K} (y_k - \hat{y}_k)^2 = \frac{1}{2} \sum_{k \in K} (y_k - a_k^{(n_\ell)})^2$

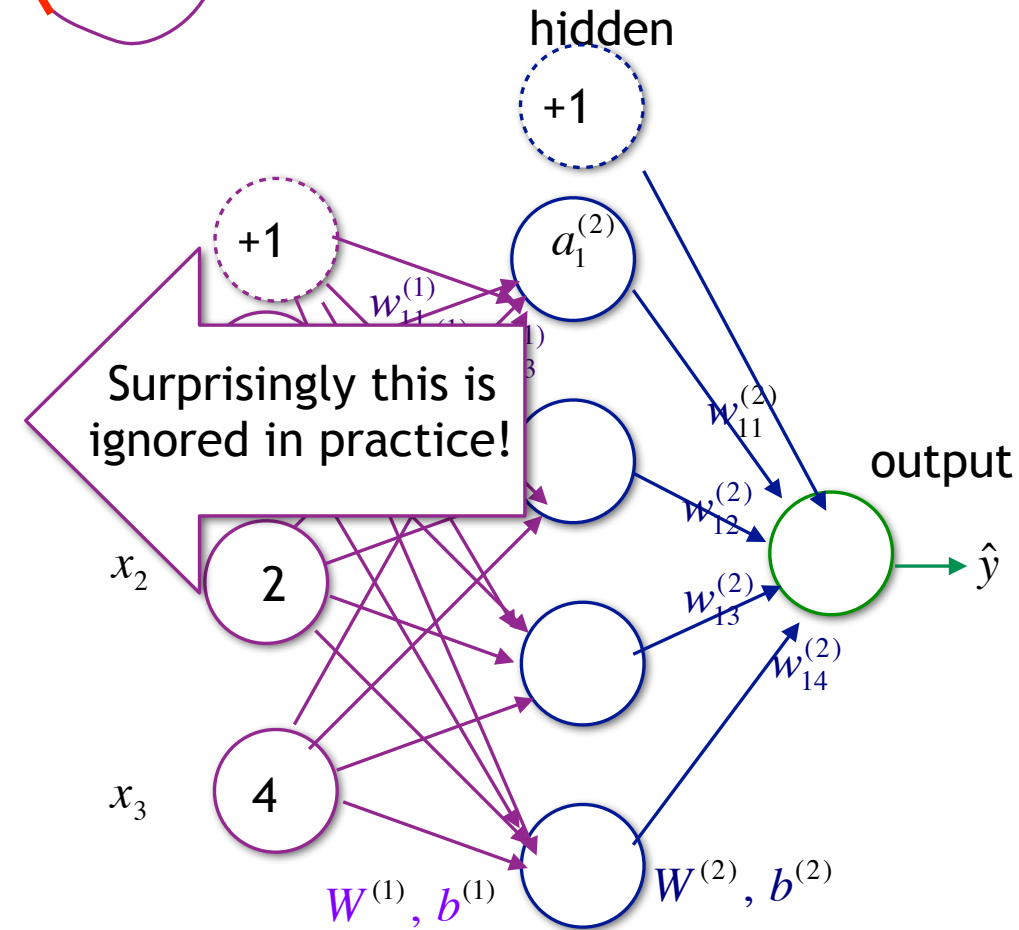
Cross entropy cost for single output: $J(W, b, \mathbf{x}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$

Cross entropy cost for multiple outputs: $J(W, b, \mathbf{x}, y) = - \sum_{k \in K} y_k \log(\hat{y}_k)$

Problems!



- ❑ We would like to use gradient descent to update the weights...
- ❑ Gradient descent worked before because we had a **convex** function... we no longer can assume our function is convex
- ❑ Solutions?
 - We can find the partial derivative for one level when the other levels are fixed
 - We can try different initial weights and retrain with different initial configurations



Training a neural network

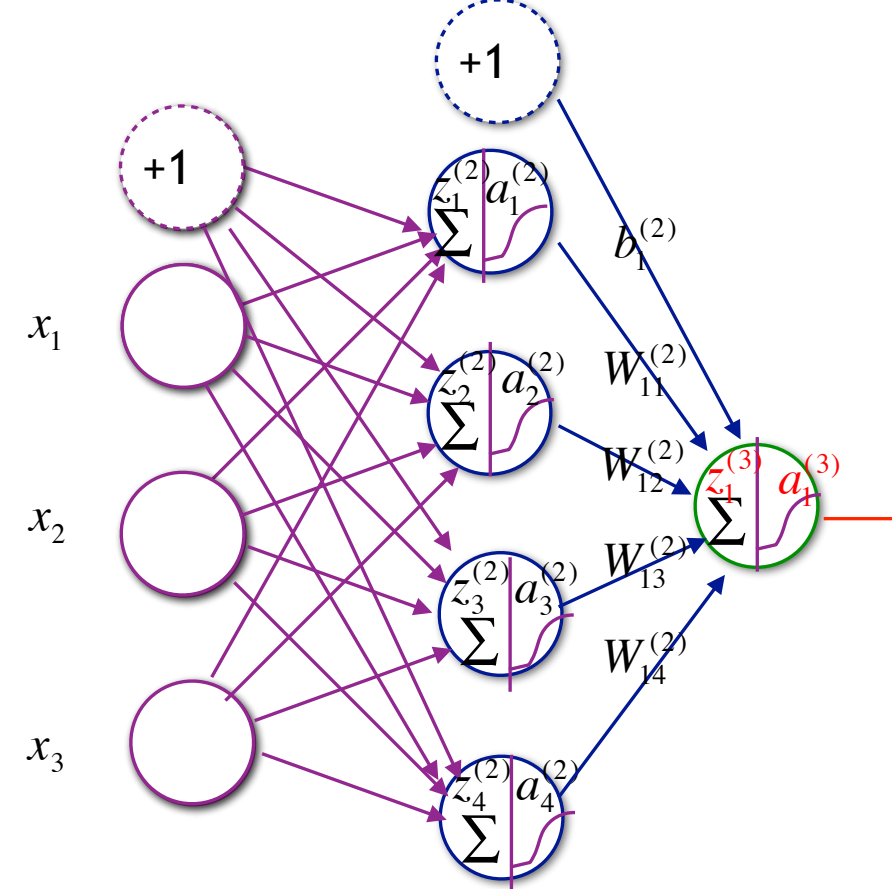
Local optimization:

Our approach:

- 1) create a ***stochastic algorithm*** where we update the weights after observing only *one training example*
 - If the neural network correctly identifies (predicts) the example - no changes are made to the weights
 - If the neural network incorrectly identifies (predicts) the example - the weights are updated to reduce the error

How do you think we can update the weights?

- 2) we turn our stochastic algorithm into batch gradient descent.

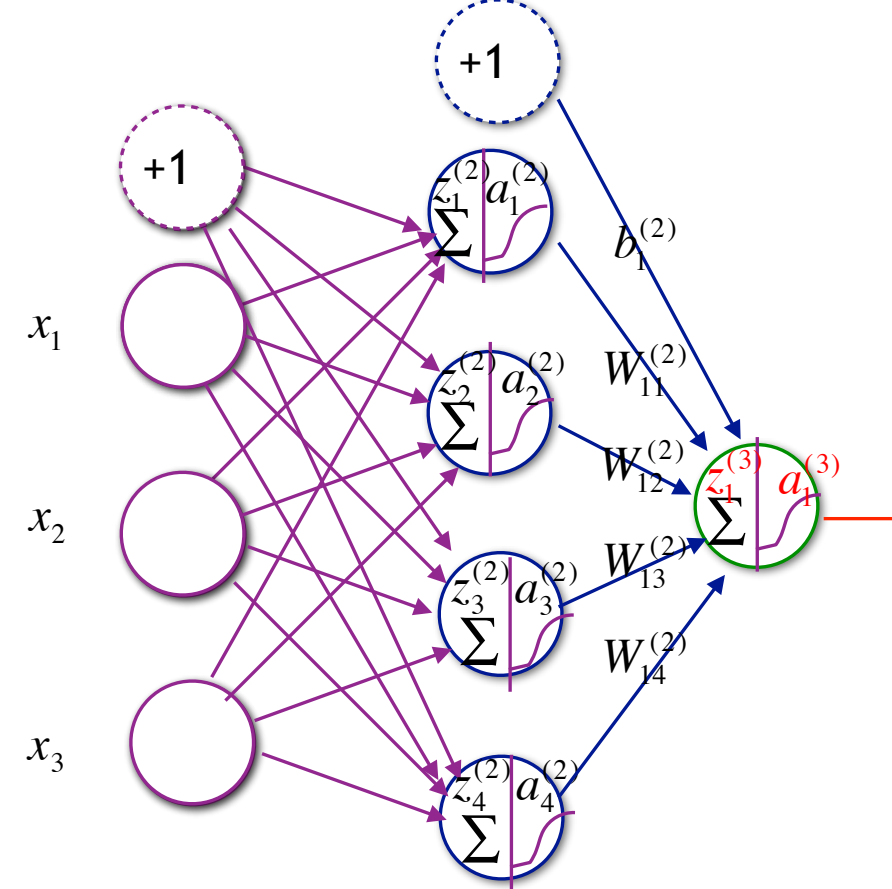


Local Optimization

$$W_{\text{new}} = W_{\text{old}} - \alpha \nabla \text{error}$$

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

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



Stochastic gradient descent algorithm

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

While iterations < iteration limit:

Randomly choose a training example: (\mathbf{x}, y)

Run forward propagation and save for each level, $a^{(\ell)}$ and $z^{(\ell)}$

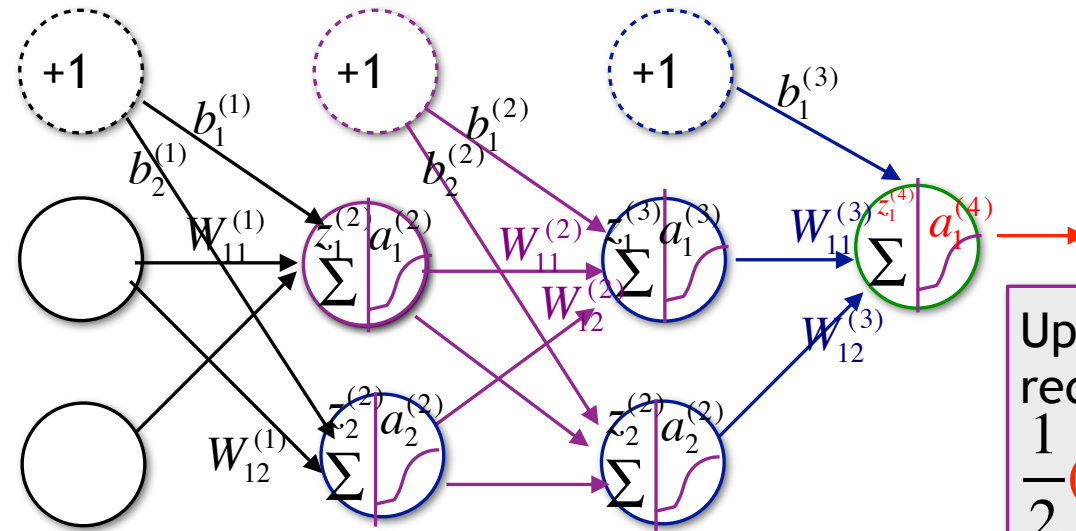
Run back propagation to compute the partial derivatives: $\nabla_{b^{(\ell)}} J(W, b, \mathbf{x}, y)$, $\nabla_{W^{(\ell)}} J(W, b, \mathbf{x}, y)$

Perform a gradient descent step for $\ell \in \{1 \dots n_\ell\}$:

$$W^{(\ell)} = W^{(\ell)} - \alpha \nabla_{W^{(\ell)}} J(W, b, \mathbf{x}, y)$$

$$b^{(\ell)} = b^{(\ell)} - \alpha \nabla_{b^{(\ell)}} J(W, b, \mathbf{x}, y)$$

Gradient descent algorithm depends on the error function and the structure of the network



Update weight to reduce error:
 $\frac{1}{2}(a_1^{(4)} - y)^2$

How would you store all the partial derivatives?

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

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

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

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

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

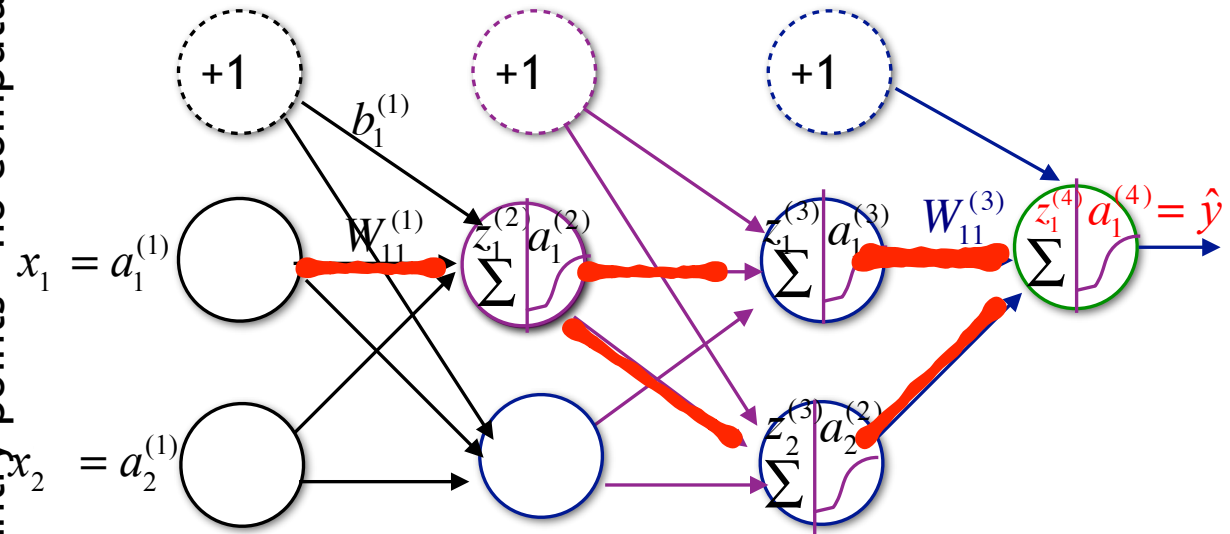
$$\nabla_{\mathbf{b}^{(\ell)}} J(W, \mathbf{b}) = \begin{bmatrix} \sum_{i=1}^N \frac{J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{b_1^{(\ell)}} \\ \sum_{i=1}^N \frac{J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{b_2^{(\ell)}} \\ \vdots \\ \sum_{i=1}^N \frac{J(W, \mathbf{b}, \mathbf{x}^{(i)}, y^{(i)})}{b_{s_{\ell}+1}^{(\ell)}}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

Partial derivatives for neural networks

It is a mess

For this lecture, all activations are: $f(z) = \frac{1}{1 + e^{-z}}$

Entry points - no computation



$$J = \frac{1}{2}(a_1^{(4)} - y)^2$$

Chain Rule

$$\begin{aligned} \frac{\partial J}{\partial W_{11}^{(1)}} &= \frac{\partial J}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} = \frac{\partial J}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} + \frac{\partial J}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} \\ &= \frac{\partial J}{\partial z_1^{(4)}} \frac{\partial z_1^{(4)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} + \frac{\partial J}{\partial z_1^{(4)}} \frac{\partial z_1^{(4)}}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} = -(y - f(z_1^{(4)}))f'(z_1^{(4)}) \frac{\partial z_1^{(4)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} + -(y - f(z_1^{(4)}))f'(z_1^{(4)}) \frac{\partial z_1^{(4)}}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} \end{aligned}$$

Its complicated and we will end up recomputing the same values again and again

Can we find a pattern to
make this much easier?