Artificial Intelligence and Machine Learning

Neural Networks

Lecture Outline

- Logistic Regression Review
- Neural Networks
  - Forward pass
  - Backward pass

Review: Logistic Regression

classification

for discrete value

Binary

Multiclass

Binary

- Sigmoid:

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

- BCE

Multiclass

- softmax

$$\sigma(z) = \frac{\exp(z)}{\sum_{i=1} \exp(z_i)}$$

- cross entropy loss

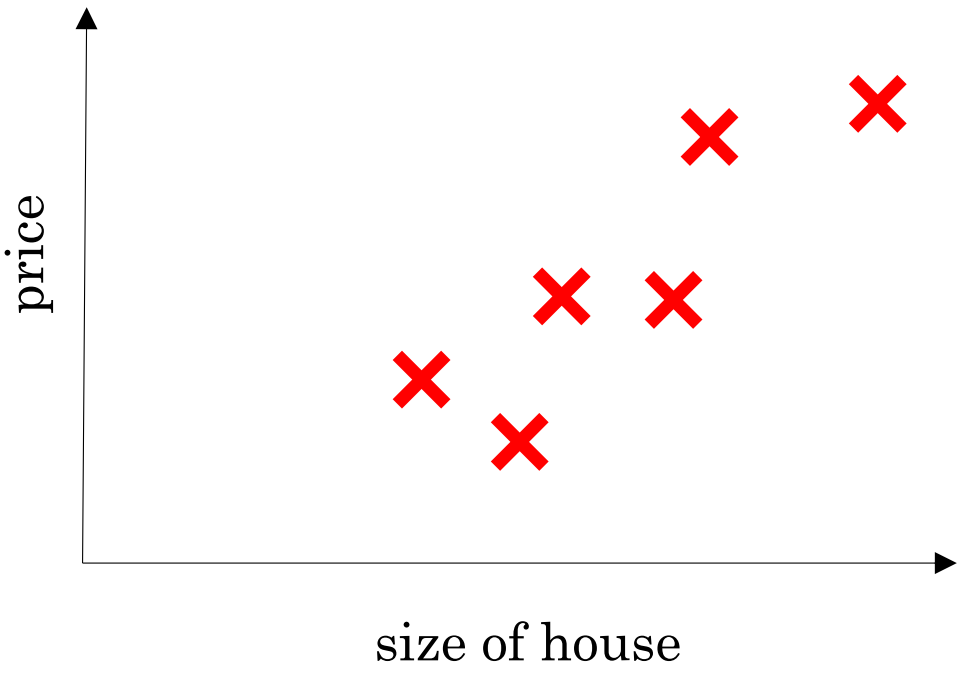
deeplearning.ai

Introduction to Deep Learning

What is a Neural Network?

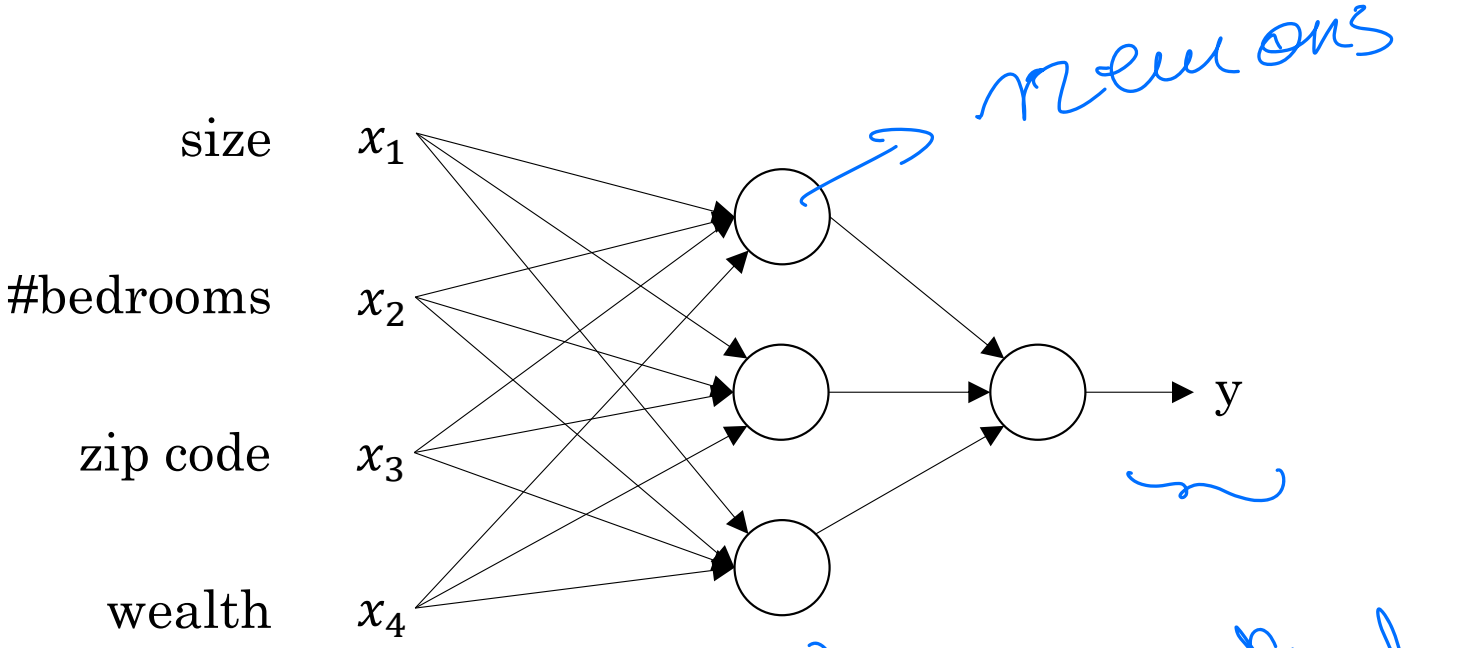
model inspired by the brain

Housing Price Prediction



Housing Price Prediction

Housing Price Prediction



deeplearning.ai

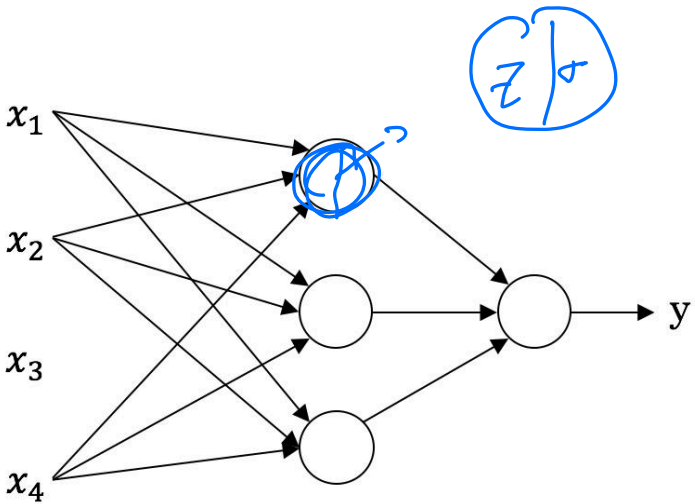
Introduction to Deep Learning

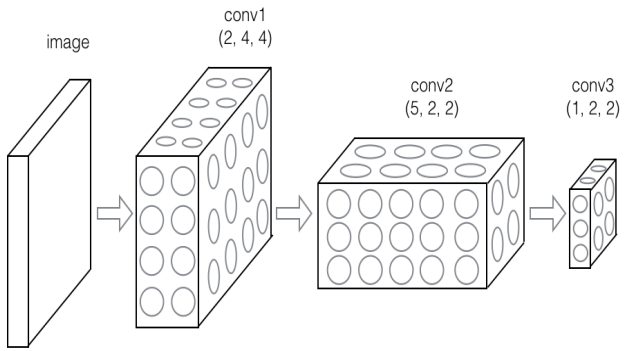
Supervised Learning with Neural Networks

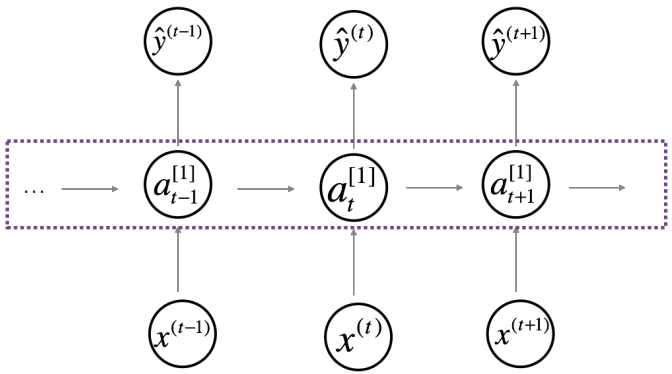
Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,…,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Neural Network examples







Standard NN

Convolutional NN

Recurrent NN

deeplearning.ai

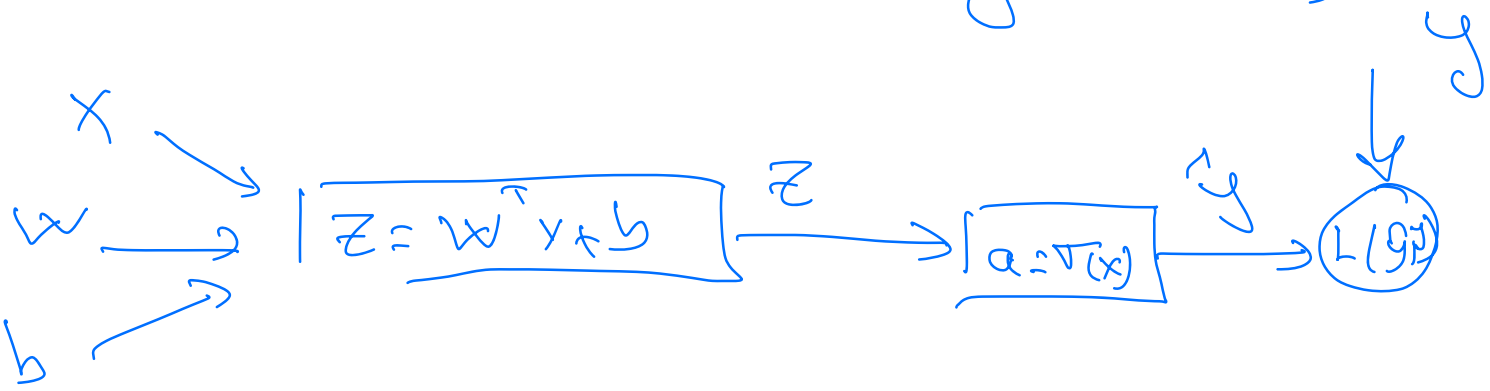
One hidden layer Neural Network

Neural Networks Overview

* what inside each neuron?

$$z = w^T x + b$$

$$y = \sigma(z)$$



* for each hidden layer & output:

$$z^{[1]} = w^{[1]T} x + b^{[1]}$$

$$a^{[1]} = \sigma[z^{[1]}]$$

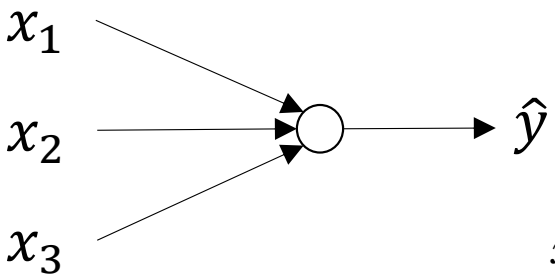
$$z^{[2]} = w^{[2]T} a^{[1]} + b^{[2]}$$

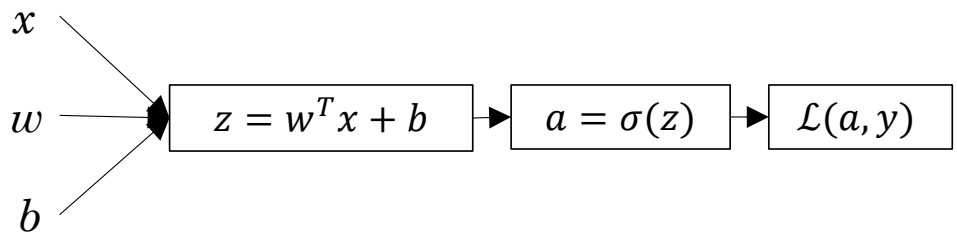
$$a^{[2]} = \sigma(z^{[2]}]$$

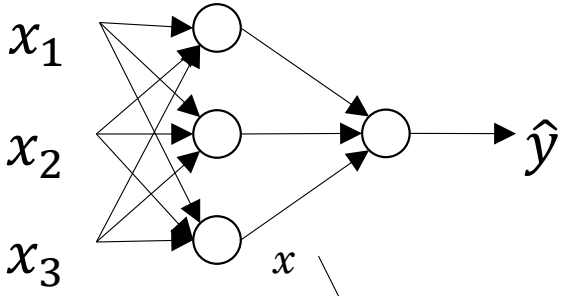
$$z^{[3]} = w^{[3]T} a^{[2]} + b^{[3]}$$

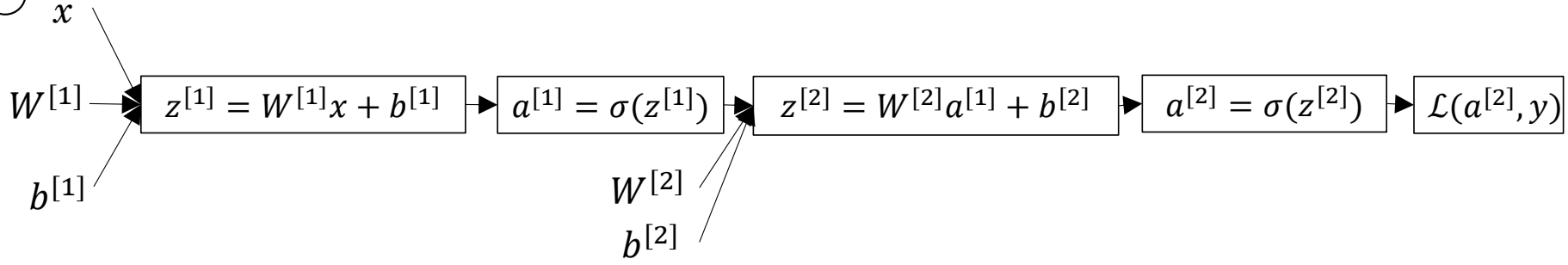
$$a^{[3]} = \sigma[z^{[3]}]$$

What is a Neural Network?







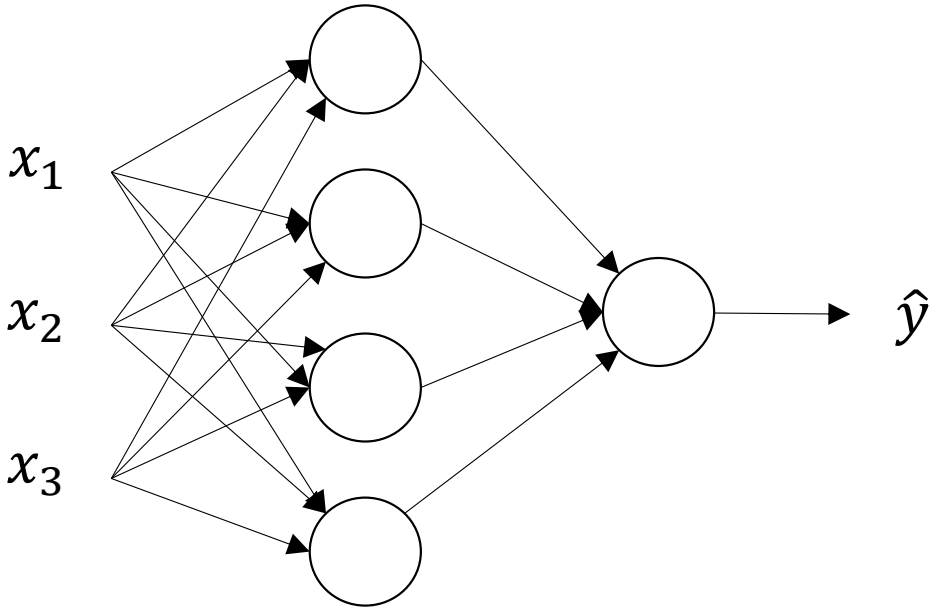


deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation

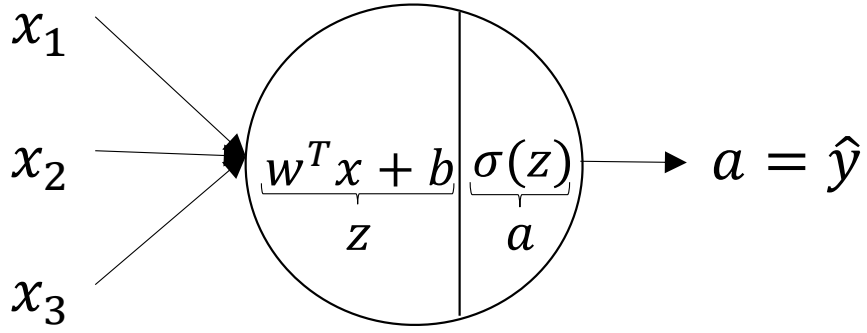


deeplearning.ai

One hidden layer Neural Network

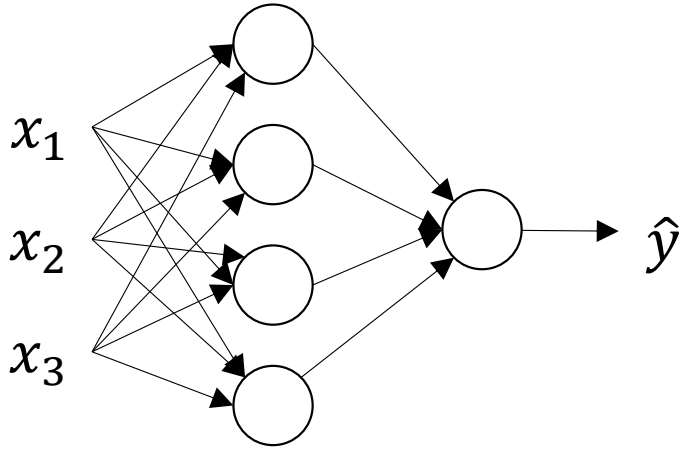
Computing a Neural Network's Output

Neural Network Representation

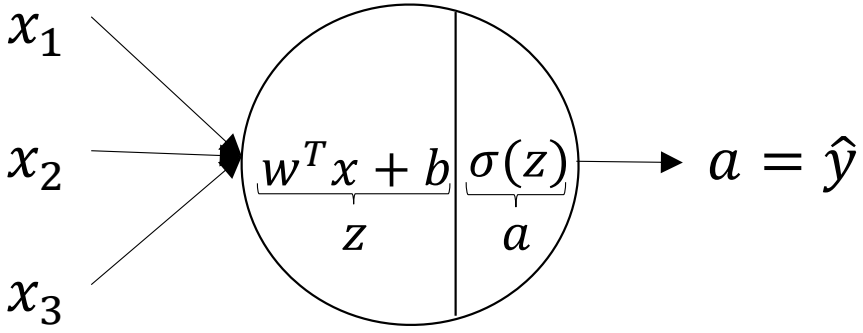


$$z = w^T x + b$$

$$a = \sigma(z)$$

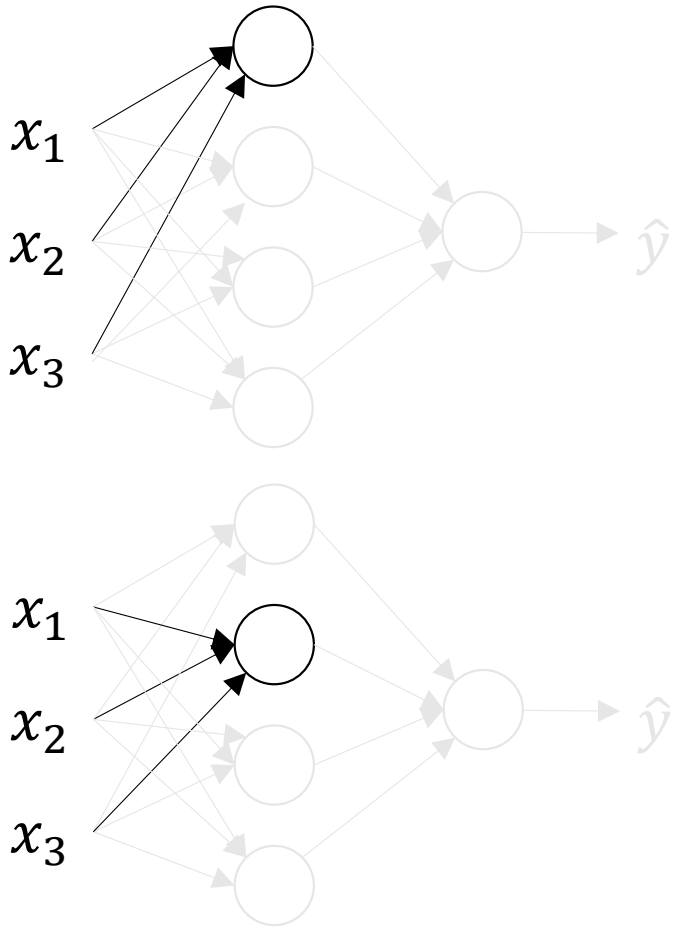


Neural Network Representation

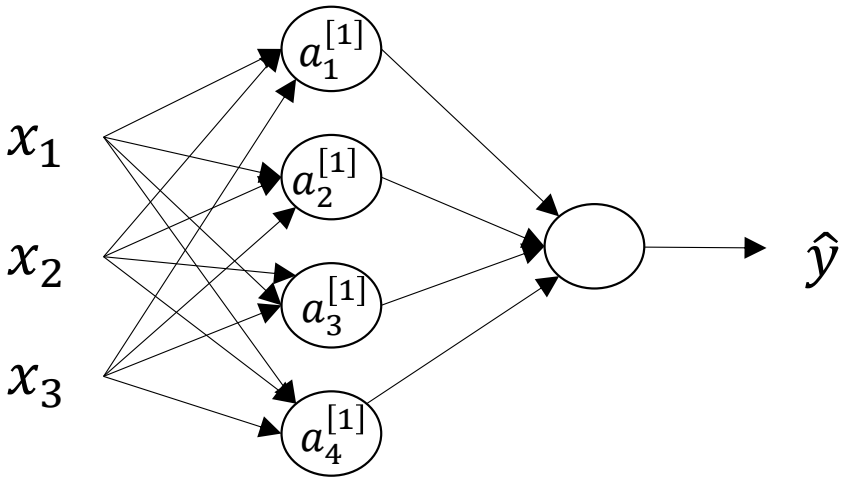


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

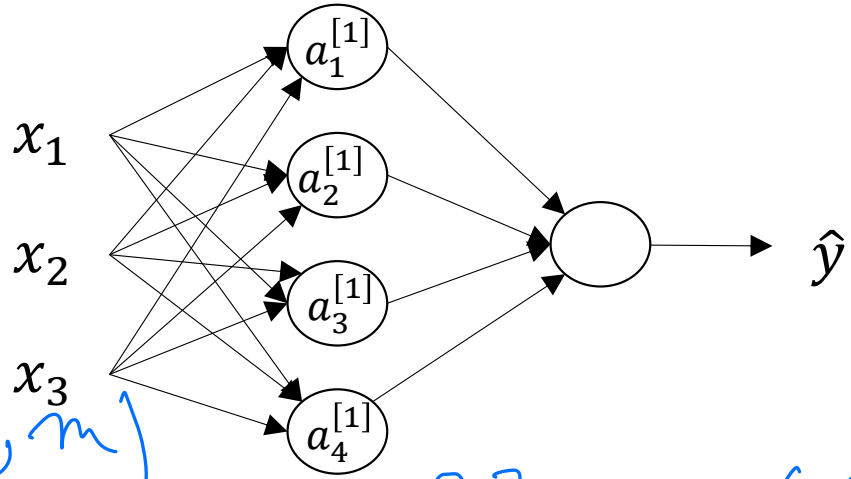
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

$$\Rightarrow Z^{[1]} = W^{[1]T} X + b^{[1]}$$

Neural Network Representation learning



Given input x:

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

$$a^{[1]} = \sigma(z^{[1]})$$

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

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

$(4, m)$

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$(4, m)$

$(4, n)$ (n, m): n features, m data points

$a^{[1]} = \sigma(z^{[1]})$ → same size $(4, m)$.

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

$(1, 4)\ (4, m)\ (1, m)$

$(1, m)$

Reason: 3 hidden layers

$z^{[1]} = w^{[1]} x + b^{[1]}$

$(4, 4)\ (4, m)\ (4, m)$

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

$(3, 4)$

$(3, m)$

$$Z = \begin{bmatrix} ? \\ 0 \end{bmatrix}^{[1]} X + b$$

$(3, 4)$ → 4

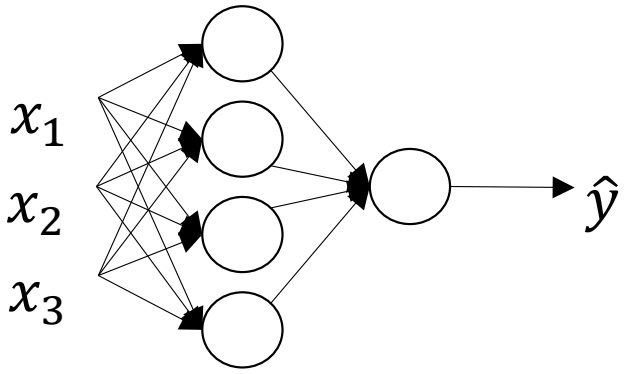
4×1

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Vectorizing across multiple examples


```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

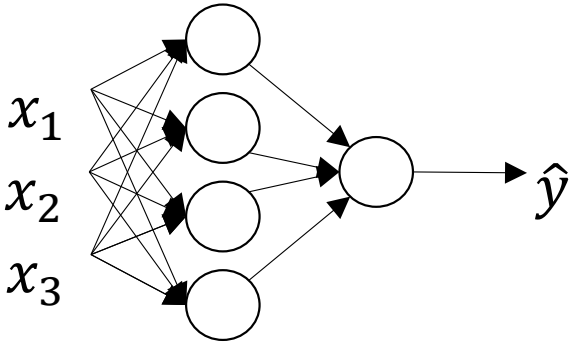
deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

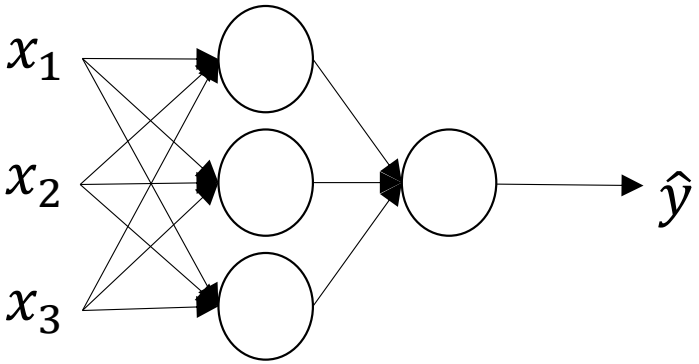
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

deeplearning.ai

One hidden layer Neural Network

Activation functions

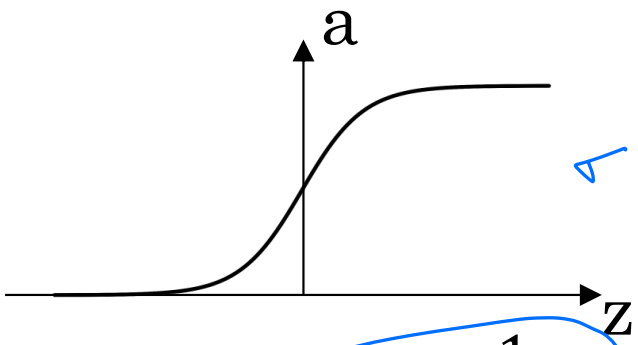
Activation functions



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

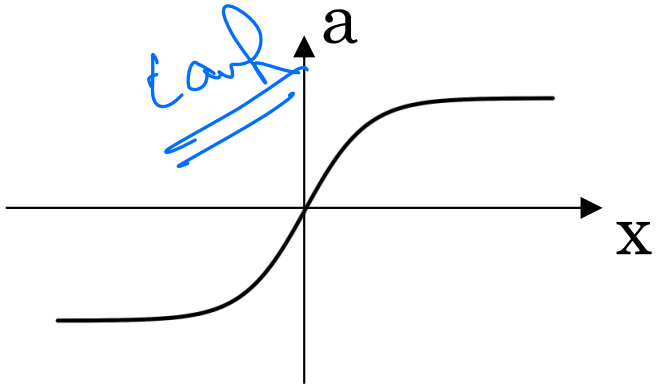
Pros and cons of activation functions



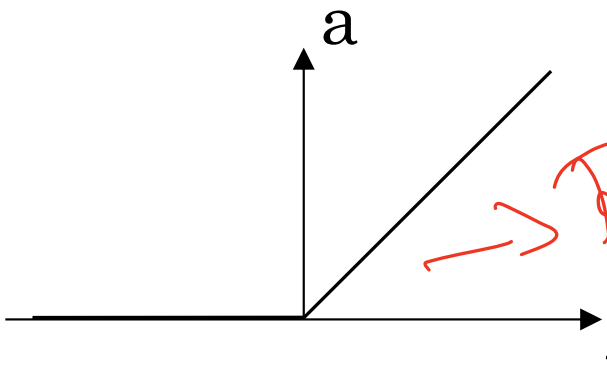
$\sigma(0) = 0.5$

sigmoid: $a = \frac{1}{1+e^{-z}}$

$\frac{1}{1+e^{0}} = \frac{1}{2}$

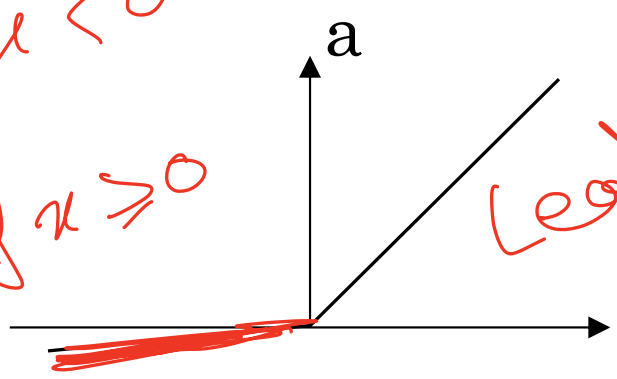


tanh



→ Relu : $\begin{cases} 0 & x < 0 \\ x \text{ if } x \geq 0 \end{cases}$

Relu : $\max(0, x)$



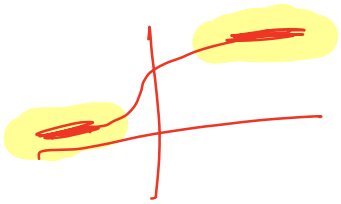
Leaky Relu

- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$.

  between (0,1) → like probability

  suitable for binary classification

  derivative: $\sigma(x) \cdot (1 - \sigma(x))$.

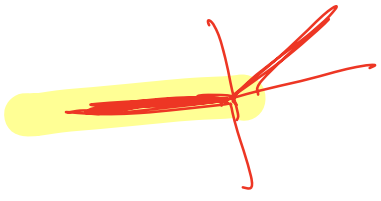


gradient become very small for extreme value which slow down the learning process

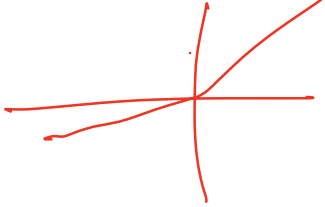
- ReLu: $\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ ⇒ allow fast training (-fast convergence)

derivative: $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

Dying problem: Neurons can become inactive during training.



* Leaky ReLu:



$$\begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$$

$$\max(0.001x, x)$$

α

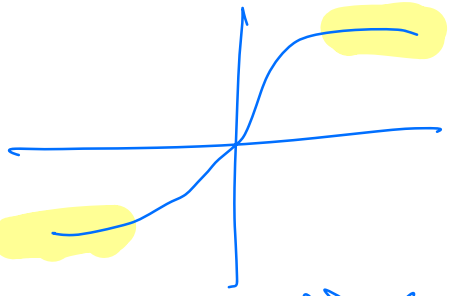
derivative:

$$\begin{cases} \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$$

⇒ solve the dying problem.

* Tanh: $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

range between $(-1, 1)$



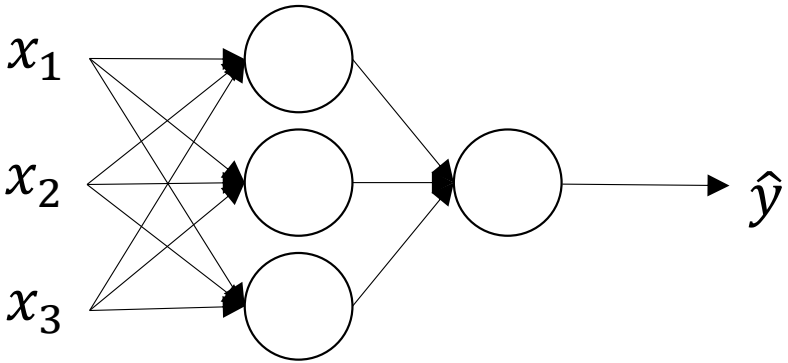
Similar to the sigmoid, vanishing gradient problem at extreme values.

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



Given x:

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

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

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

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

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

* Review:

optimization algorithm used to find the weights that minimize the cost (loss) function

$$W_{new} = W_{old} - \eta \cdot \nabla L(W)$$

gradient $\frac{\partial J}{\partial W}$

Formulas for computing derivatives

$$\Rightarrow \frac{\partial L}{\partial w} = ?? \quad \& \quad \frac{\partial L}{\partial b} = ??$$

Forward path:

$$z^1 = w^1 x + b^1$$

$$A^1 = \sigma(z^1)$$

$$z^2 = w^{2T} A^1 + b^2$$

$$A^2 = \sigma(z^2) \rightarrow \mathcal{L}$$

$L(A^2, y)$

Binary cross entropy: $L(A^2, y)$

$$\frac{\partial L}{\partial A_2} = \frac{A_2 - y}{A_2(1 - A_2)}$$

$$\frac{\partial L}{\partial W_2}, \quad \frac{\partial L}{\partial b_2}, \quad \frac{\partial L}{\partial W_1}, \quad \frac{\partial L}{\partial b_1}$$

$$\frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} = \frac{\partial L}{\partial W_2}$$

$$\frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2} = \frac{\partial L}{\partial b_2}$$

$$\frac{\partial L}{\partial Z_2} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} = \frac{A_2 - y}{A_2(1 - A_2)} \cdot A_2(1 - A_2) = A_2 - y$$

$\frac{\partial L}{\partial A_2} = \frac{A_2 - y}{A_2(1 - A_2)}$

$\frac{\partial A_2}{\partial Z_2} = A_2(1 - A_2)$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2} \cdot A_1^T$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2}$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2} \cdot 1$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1}$$

$$W_2^T \quad g^{[1]'}(Z^{[1]}) \quad X^T$$

$$\frac{\partial L}{\partial Z_1} = W_2^T \frac{\partial L}{\partial Z_2} * g^{[1]'} \cdot [Z^{[1]}]$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot x^T$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot 1$$

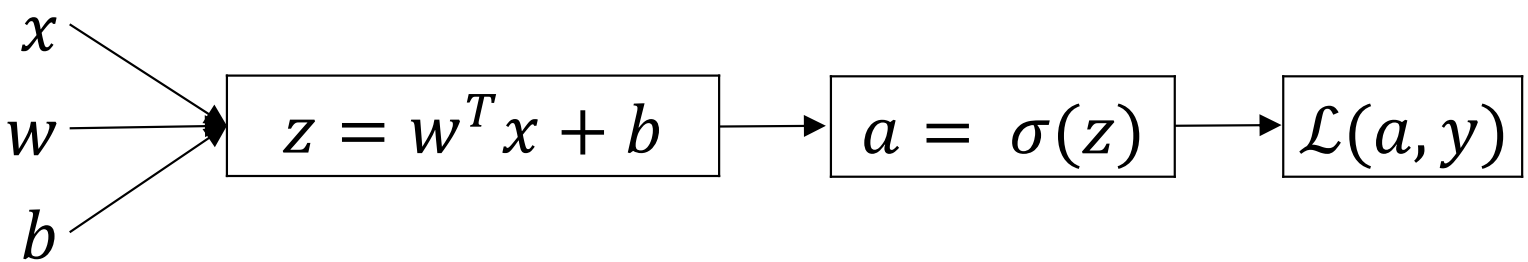
deeplearning.ai

One hidden layer Neural Network

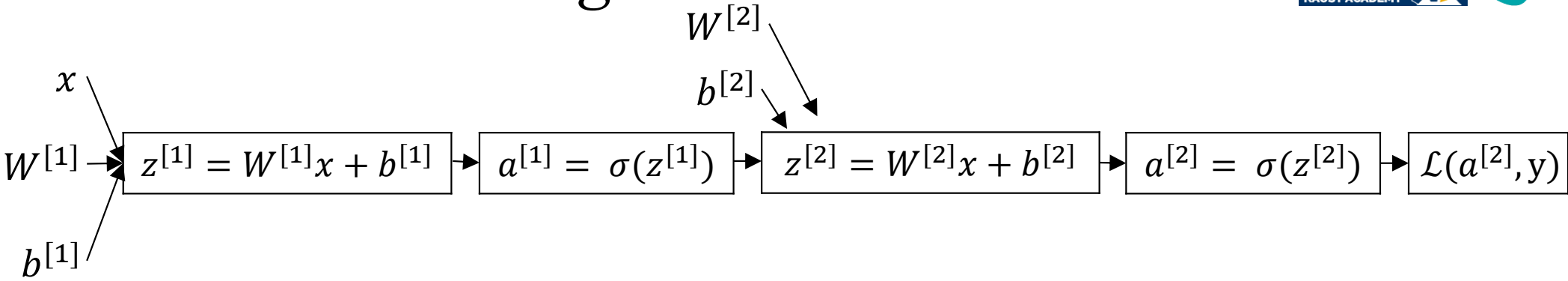
Backpropagation intuition

Computing gradients

Logistic regression



Neural network gradients



Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^{T}$$

$$db^{[1]} = dz^{[1]}$$

Backpropagation.

$$\frac{\partial L}{\partial A_2}, \quad \frac{\partial A_2}{\partial z_2} \rightarrow \frac{\partial L}{\partial z_2} = A^{[2]} - y$$

$$\frac{\partial z_2}{\partial W_2} = A^{[1]T}$$

$$\frac{\partial z_2}{\partial b_2} = 1$$

Andrew Ng

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

w^2

$g^{(1)'}(z^{(1)})$

$$\frac{\partial L}{\partial z_1}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

x^T

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$\rightarrow 1$

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]^T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

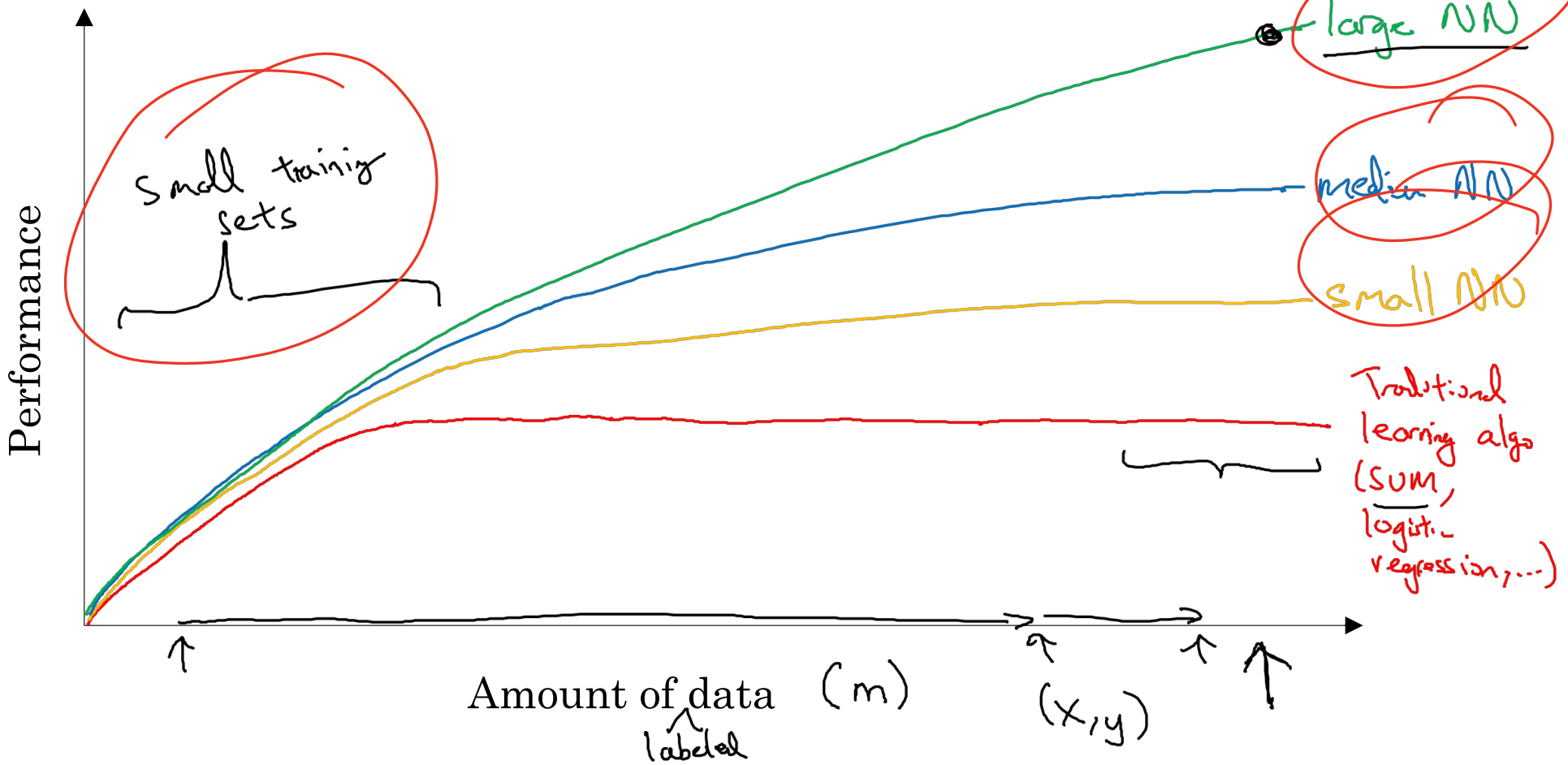
Andrew Ng

deeplearning.ai

Introduction to Neural Networks

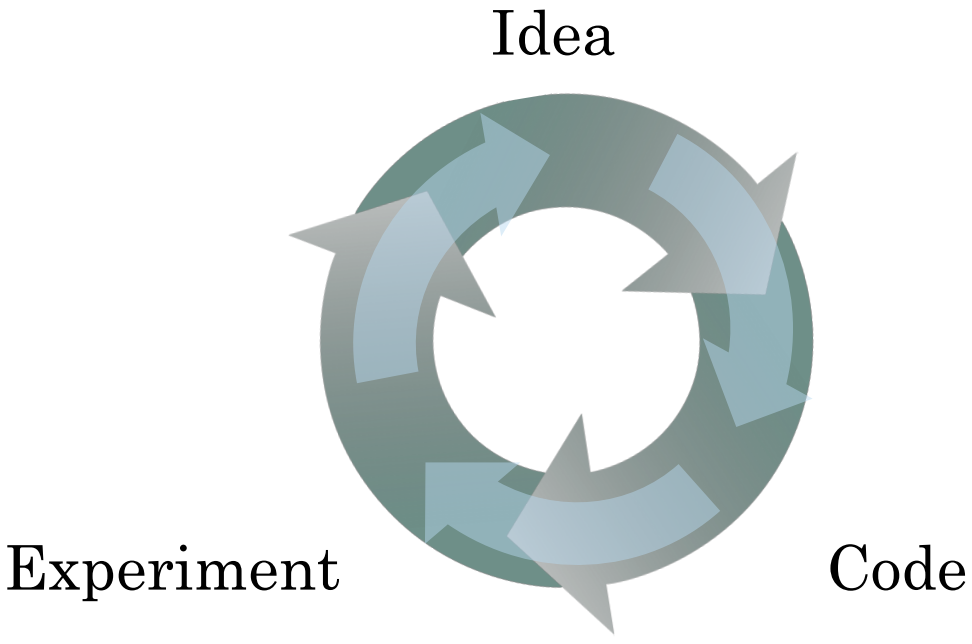
Why is Deep Learning taking off?

Scale drives deep learning progress



Scale drives deep learning progress

- Data
- Computation
- Algorithms

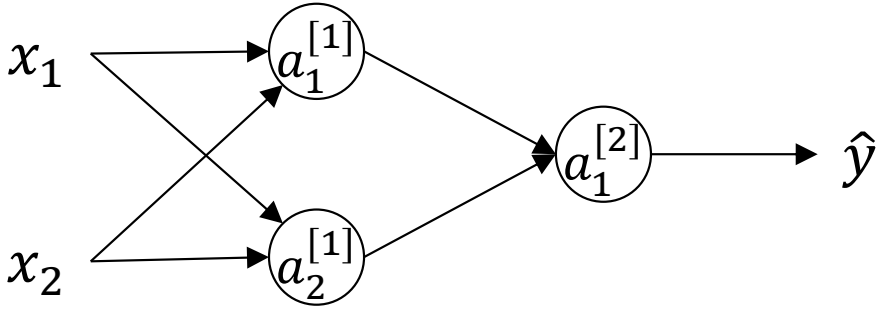


deeplearning.ai

One hidden layer Neural Network

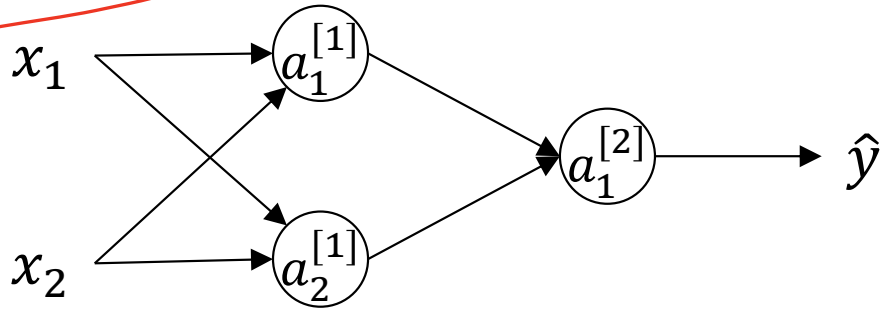
Random Initialization

What happens if you initialize weights to zero?



Zero is not generally recommended
Random initialization is preferred
to allow the network to learn

Random initialization



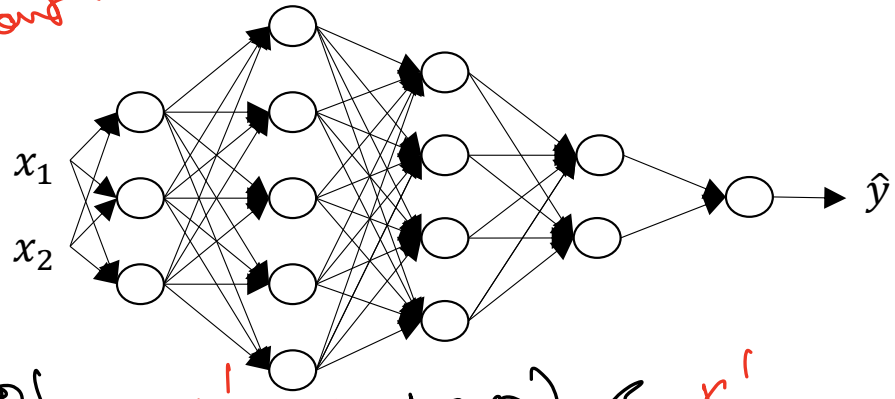
unique features
and avoid the
Symmetry problem

deeplearning.ai

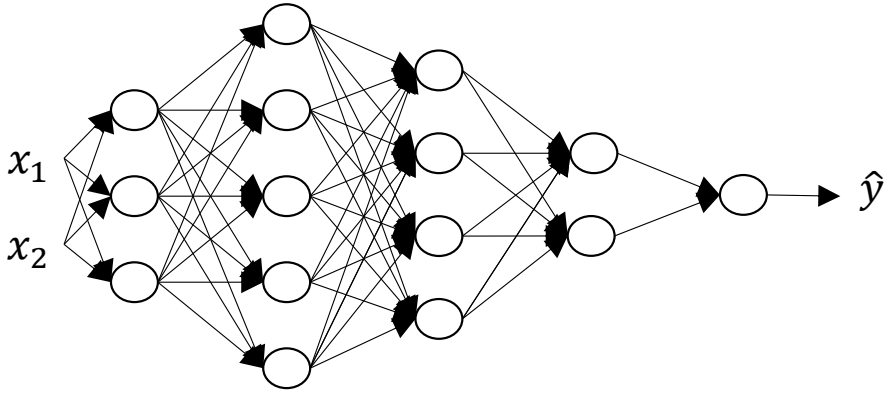
Deep Neural Networks

Getting your matrix dimensions right

Parameters $W^{[l]}$ and $b^{[l]}$



Vectorized implementation



deeplearning.ai

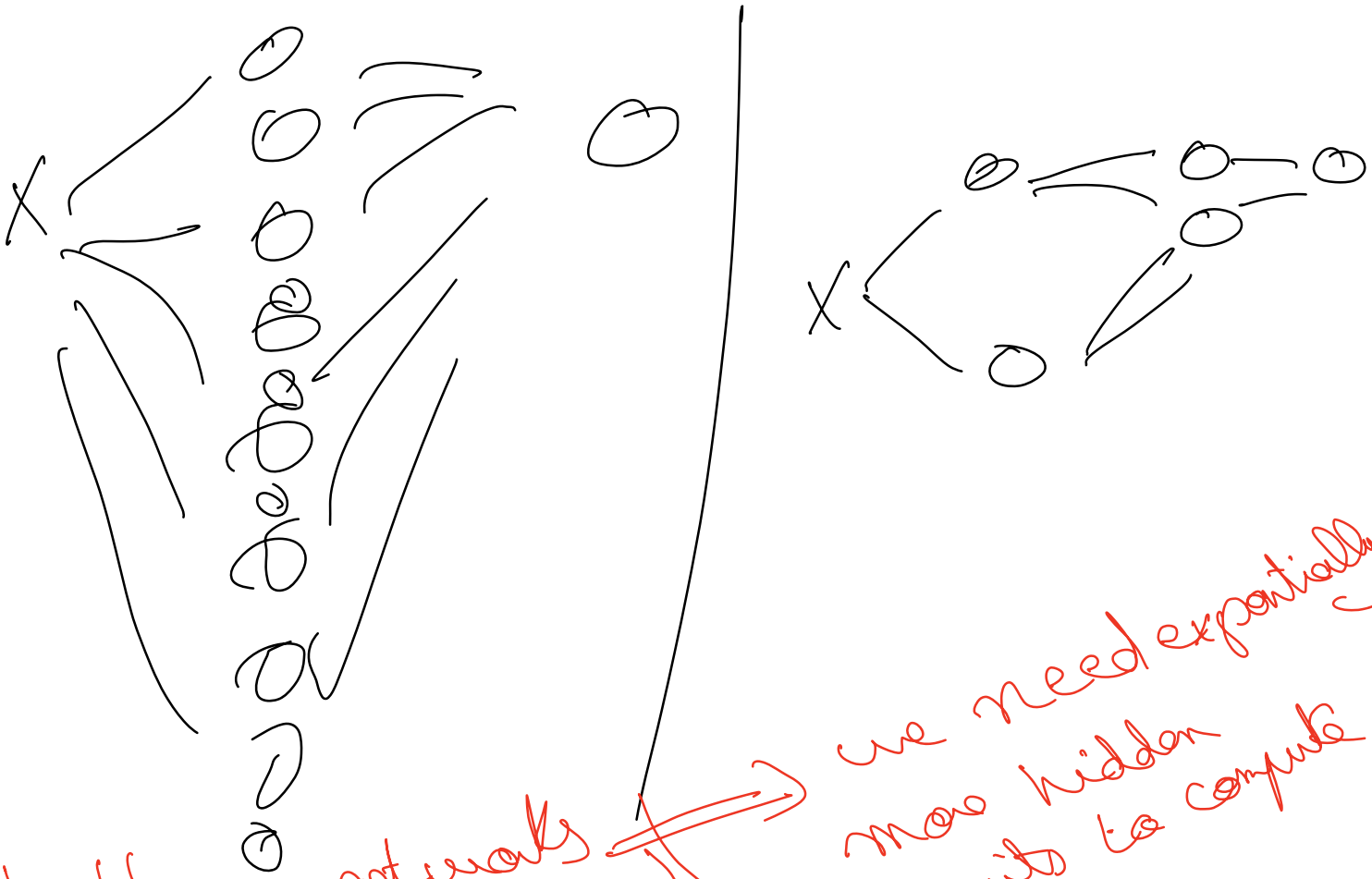
Deep Neural Networks

the number of layers ⇒ increase

Why deep representations?

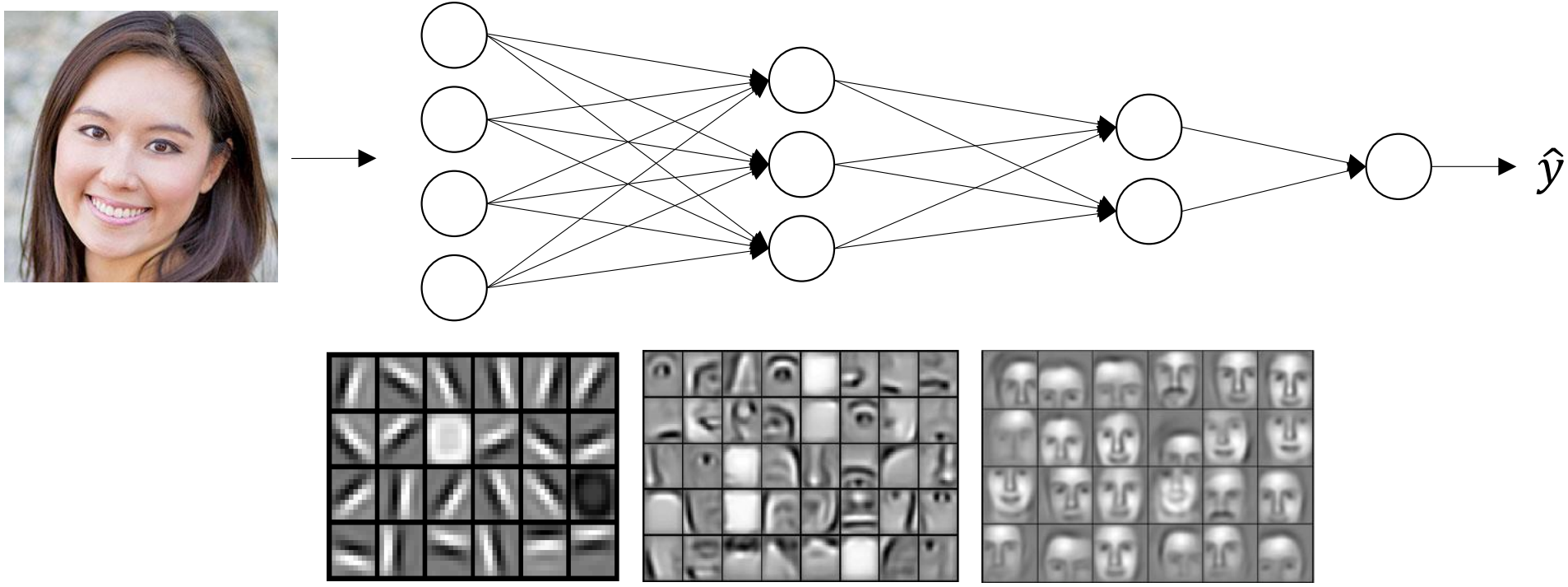
⇒ increase the # of layers can potentially capture and compute more complex functions with a reasonable number of hidden units in each layer.

Increasing the # of layers can potentially capture and compute complex function with a reasonable number of hidden units in each layer



Shallow networks (less hidden layers) → we need exponentially more hidden units to compute

Intuition about deep representation



Circuit theory and deep learning

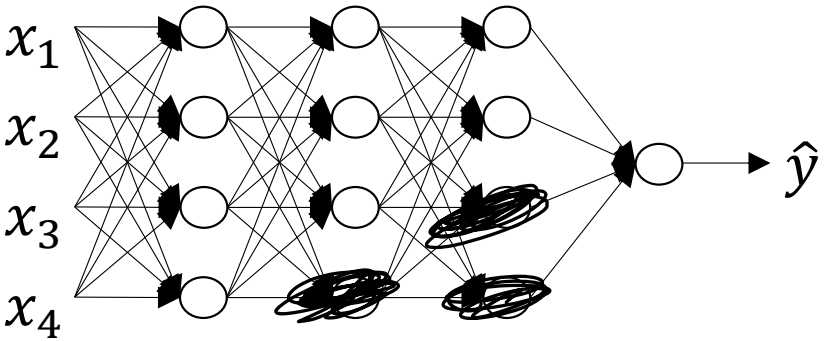
Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

deeplearning.ai

Deep Neural Networks

Building blocks of deep neural networks

Forward and backward functions



$X(n, m)$

$n = 4$

$W_1(4, 4)$ $W_2(3, 4)$

$b_1(4, m)$ $b_2(3, m)$

$W_3(2, 3)$ $W_4(1, 2)$

$b_3(2, m)$ $b_4(1, m)$

Forward and backward functions

deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Forward propagation for layer *l*

Input $a^{[l-1]}$

Output $a^{[l]}$, cache $(z^{[l]})$

Backward propagation for layer *l*

Input $da^{[l]}$

Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

Summary

Steps:

1- Forward path

2- Backward pass

3- optimizer