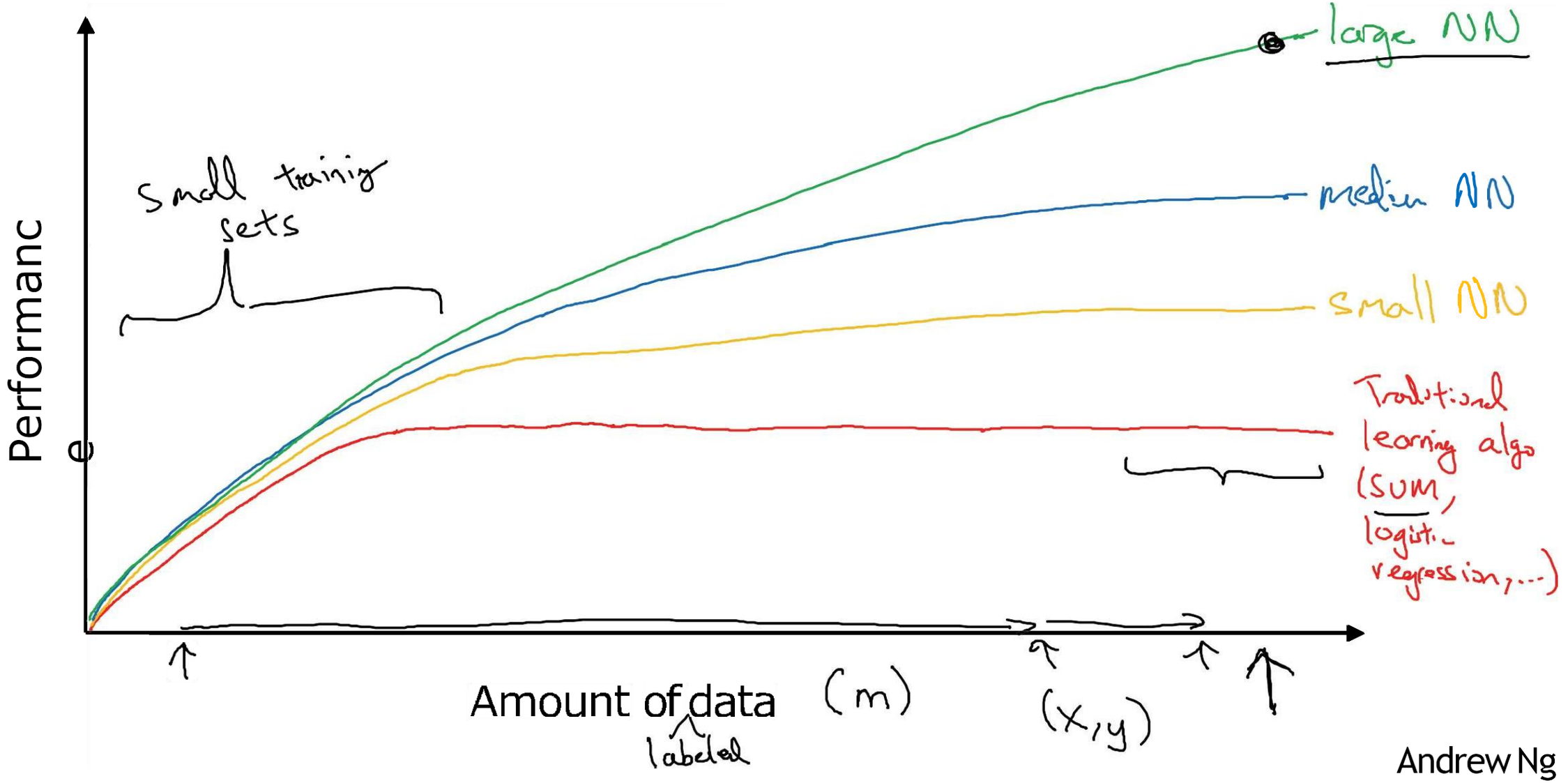
Copyright Notice

deeplearning.ai

Introduction to Neural Networks

Why is Deep Learning taking off?

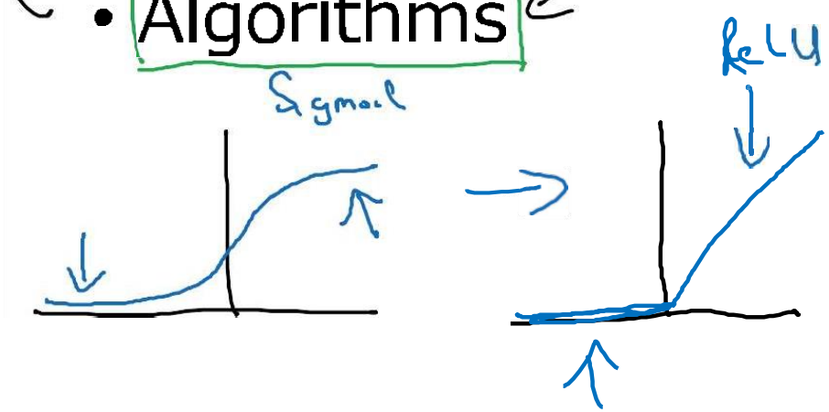
Scale drives deep learning progress



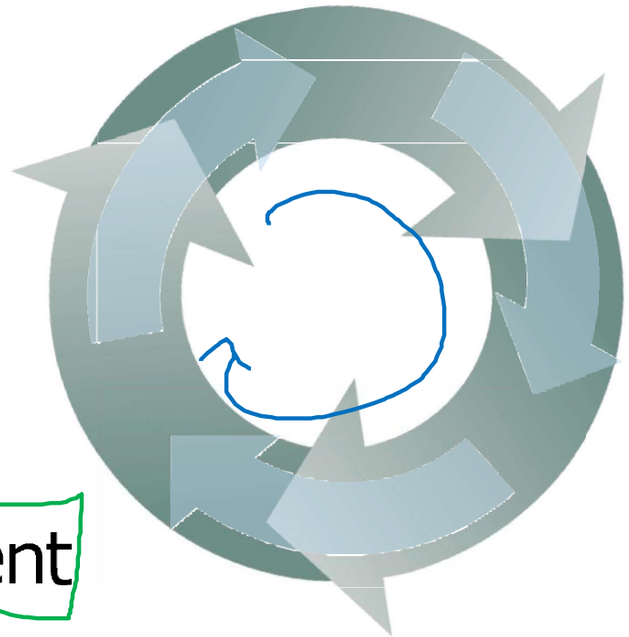
Andrew Ng

Scale drives deep learning progress

- Data
- Computation
- Algorithms



Idea



Experiment

Code

10 min
1 day
1 month

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Binary Classification

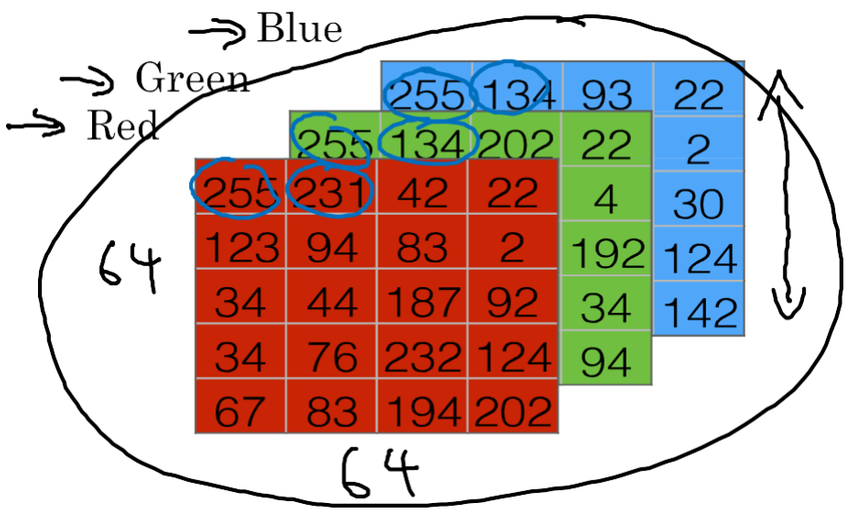
Binary Classification

64

64

1 (cat) vs 0 (non cat)

y



$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \longrightarrow y$

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$m = m_{train}$ $\qquad$ m_{test} = #test examples.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \text{ (rows)}, \quad m \text{ (columns)}$$

$X \in \mathbb{R}^{n_x \times m}$ $\qquad$ $X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$Y \in \mathbb{R}^{1 \times m}$

$Y\ shape = (1, m)$

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression

Logistic Regression

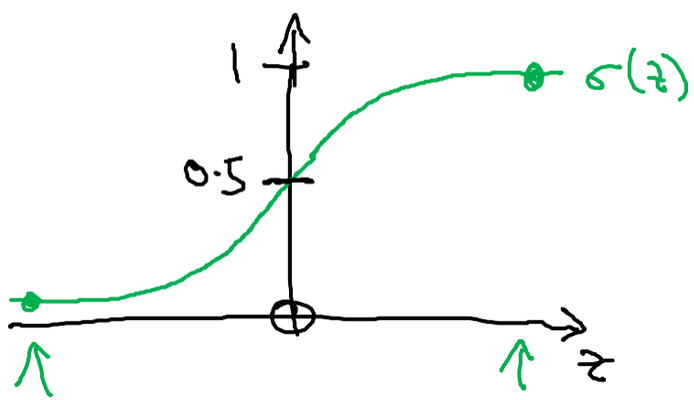
Given x, want $\hat{y} = P(y=1|x)$

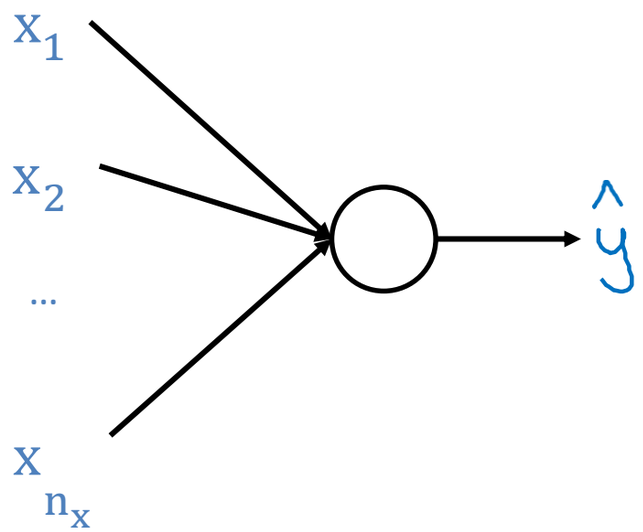
$x \in \mathbb{R}^{n_x}$

$0 \le \hat{y} \le 1$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

Output $\hat{y} = \sigma(w^T x + b)$, where $z = w^T x + b$





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

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignum}} \approx 0$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

$z^{(i)} = w^T x^{(i)} + b$

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

$x^{(i)}$, $y^{(i)}$, $z^{(i)}$ — i-th example.

Loss (error) function:

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

If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large

If $y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log 1-\hat{y}$ large ... want $\hat{y}$ small

Cost function: $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

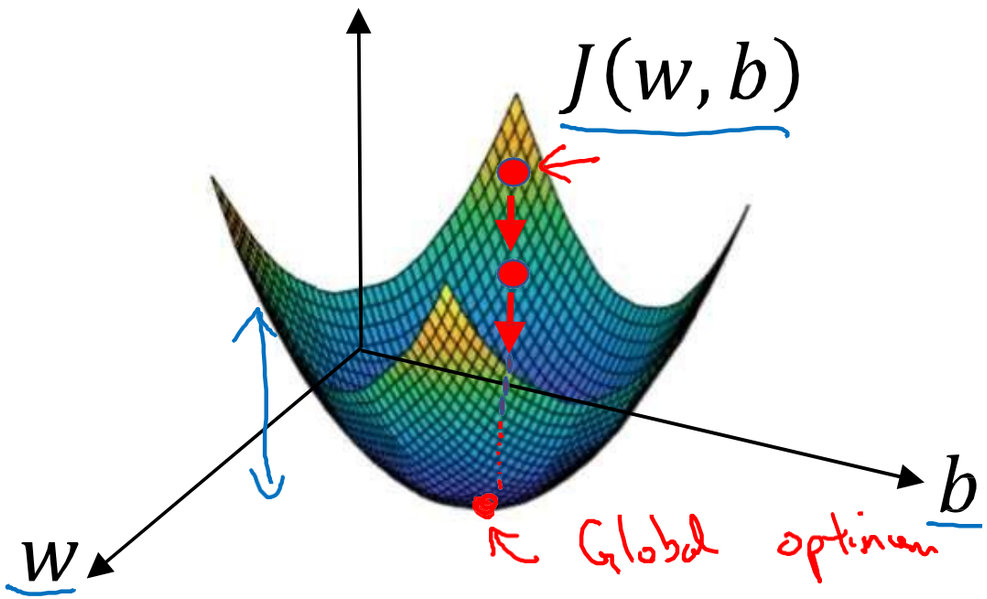
Gradient Descent

Gradient Descent

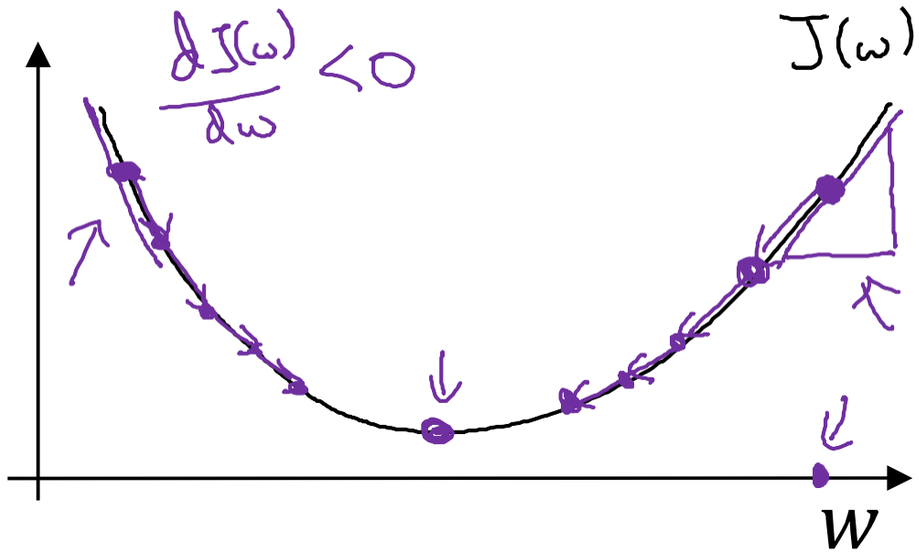
Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w,b)$



Gradient Descent



Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

}

learning rate

"dw"

$w := w - \alpha dw$

$\frac{dJ(w)}{dw} = ?$

$J(w,b)$

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

$\frac{\partial J(w,b)}{\partial w}$ → dw

$\frac{\partial J(w,b)}{\partial b}$ → db

∂ "partial derivative"

d J

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

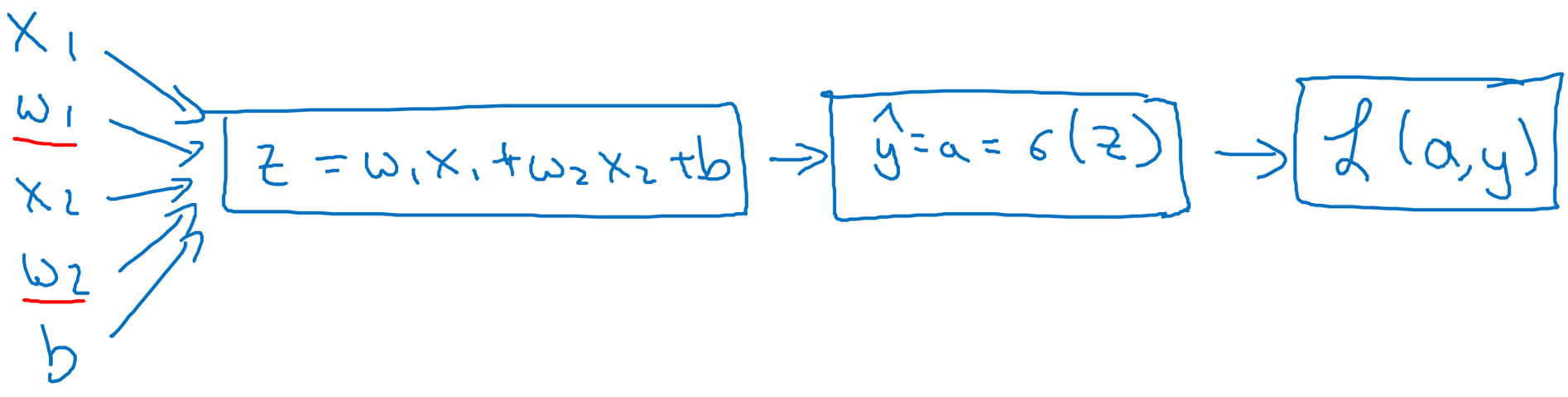
Logistic Regression Gradient descent

Logistic regression recap

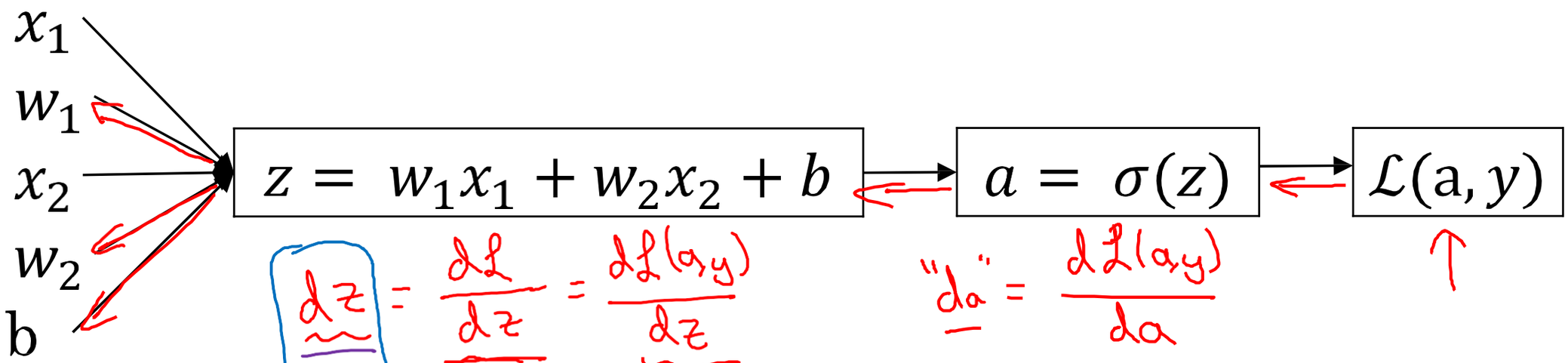
$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



Logistic regression derivatives



$dz = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$

$= a - y$

$= \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$ $\quad a(1-a)$

"da" $= \frac{d\mathcal{L}(a,y)}{da}$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$\frac{\partial \mathcal{L}}{\partial w_1} =$ "dw_1" $= x_1 \, dz$ $\quad dw_2 = x_2 \, dz$ $\quad db = dz$

$w_1 := w_1 - \alpha \, dw_1$

$w_2 := w_2 - \alpha \, dw_2$

$b := b - \alpha \, db$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$(x^{(i)}, y^{(i)})$

$dw_1^{(i)}$, $dw_2^{(i)}$, $db^{(i)}$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)} \text{ - } (x^{(i)}, y^{(i)})}$$

Logistic regression on m examples

$J=0$; $dw_1=0$; $dw_2=0$; $db=0$

→ For $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

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

$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

dw_3 … dw_n

$n=2$

$J /= m$ ←

$dw_1 /= m$; $dw_2 /= m$; $db /= m$. ←

dw_1 ↑ dw_2 ↑ db ↑

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha\, dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

$$b := b - \alpha\, db$$

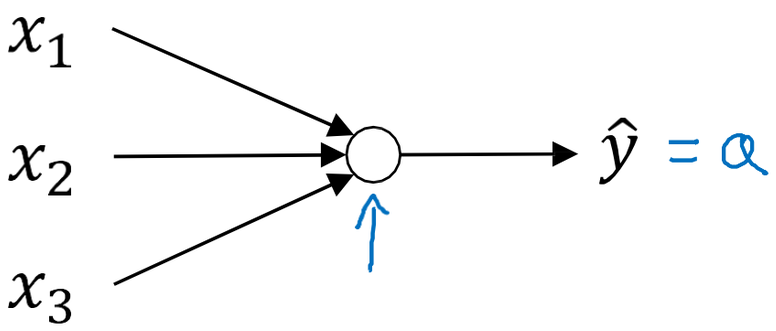
Vectorization

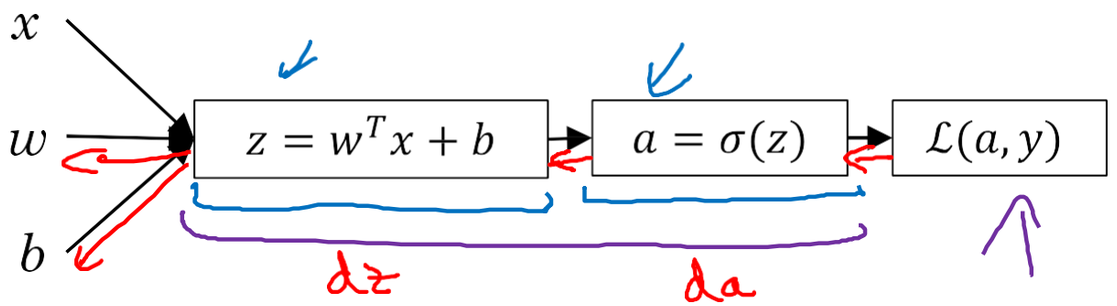
deeplearning.ai

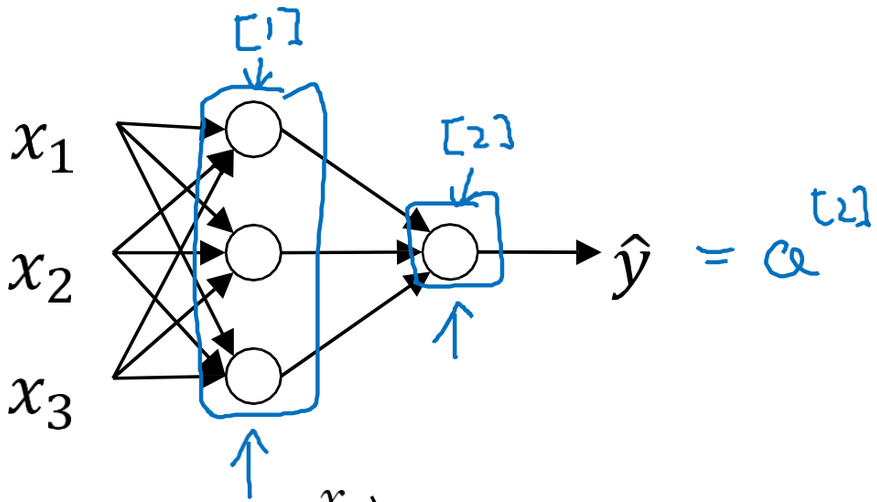
One hidden layer Neural Network

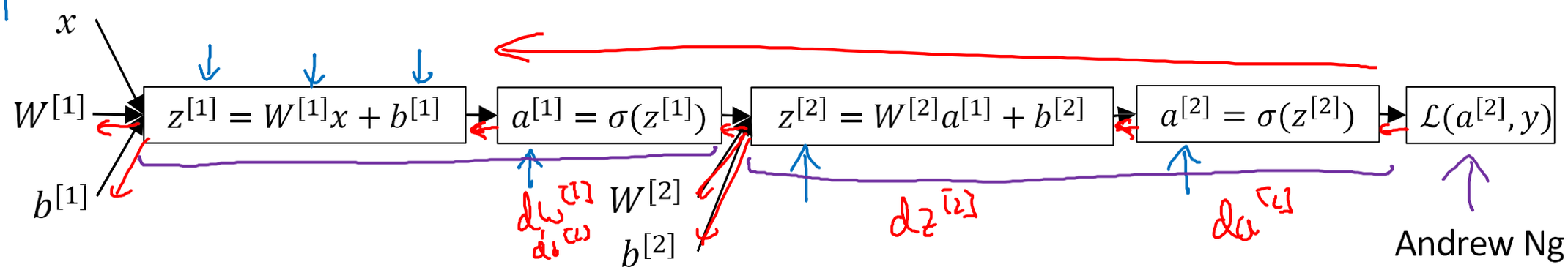
Neural Networks Overview

What is a Neural Network?







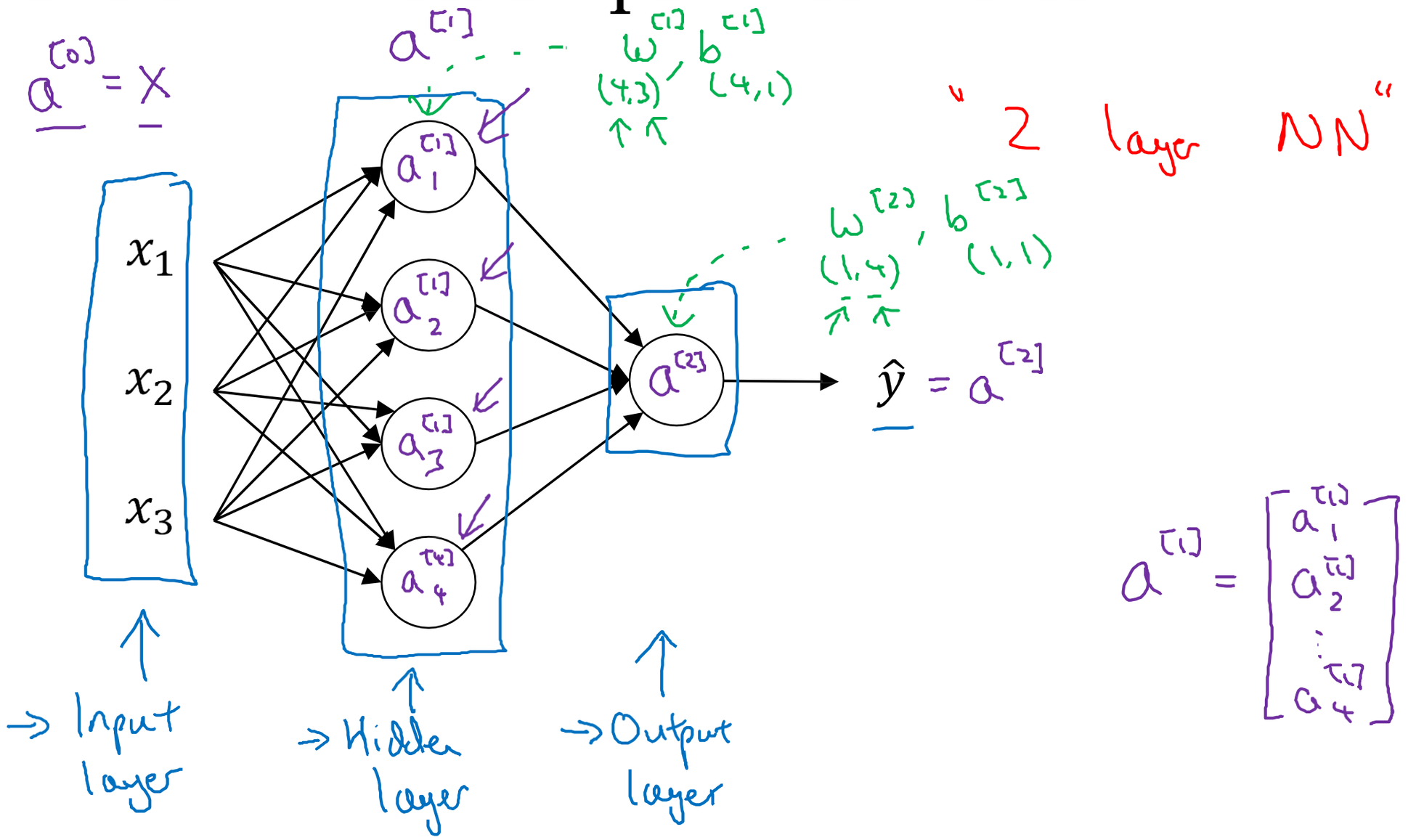


deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



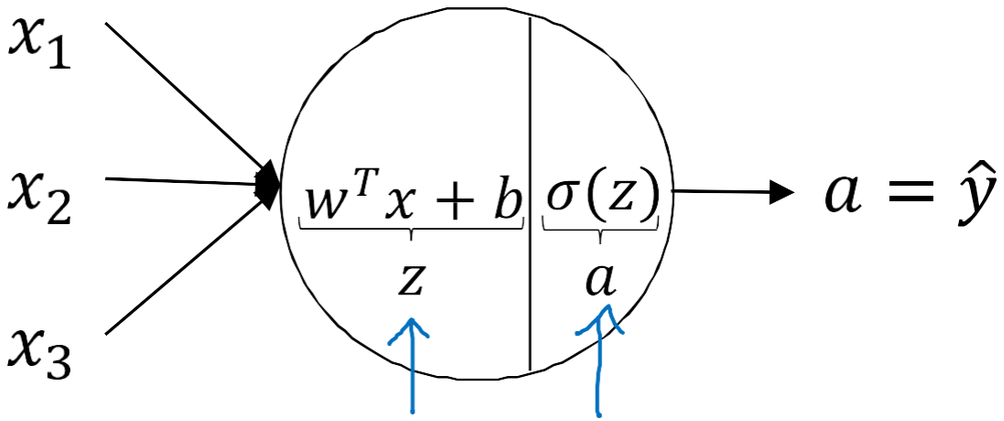
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

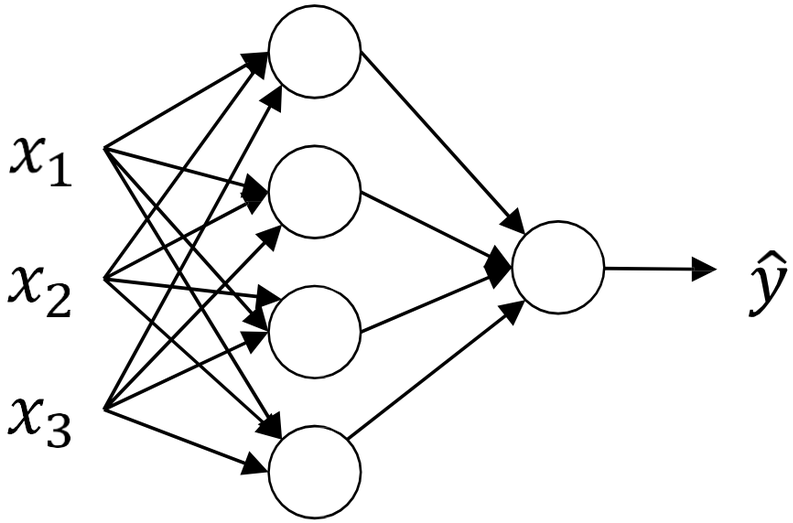
Computing a Neural Network's Output

Neural Network Representation

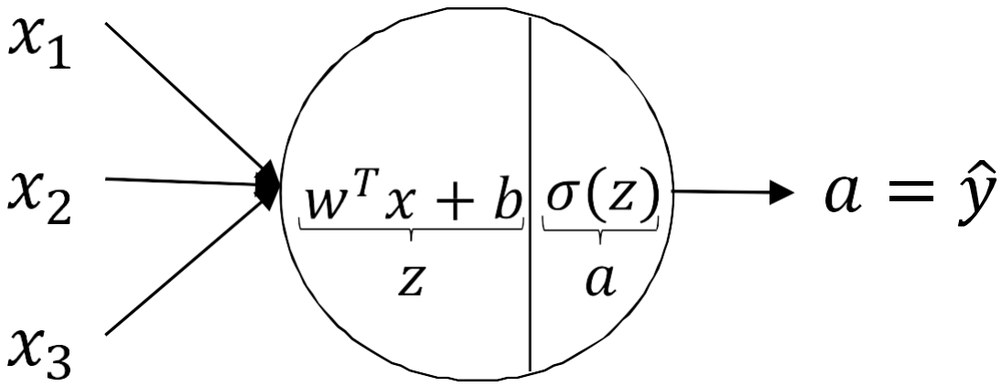


$$z = w^T x + b$$

$$a = \sigma(z)$$

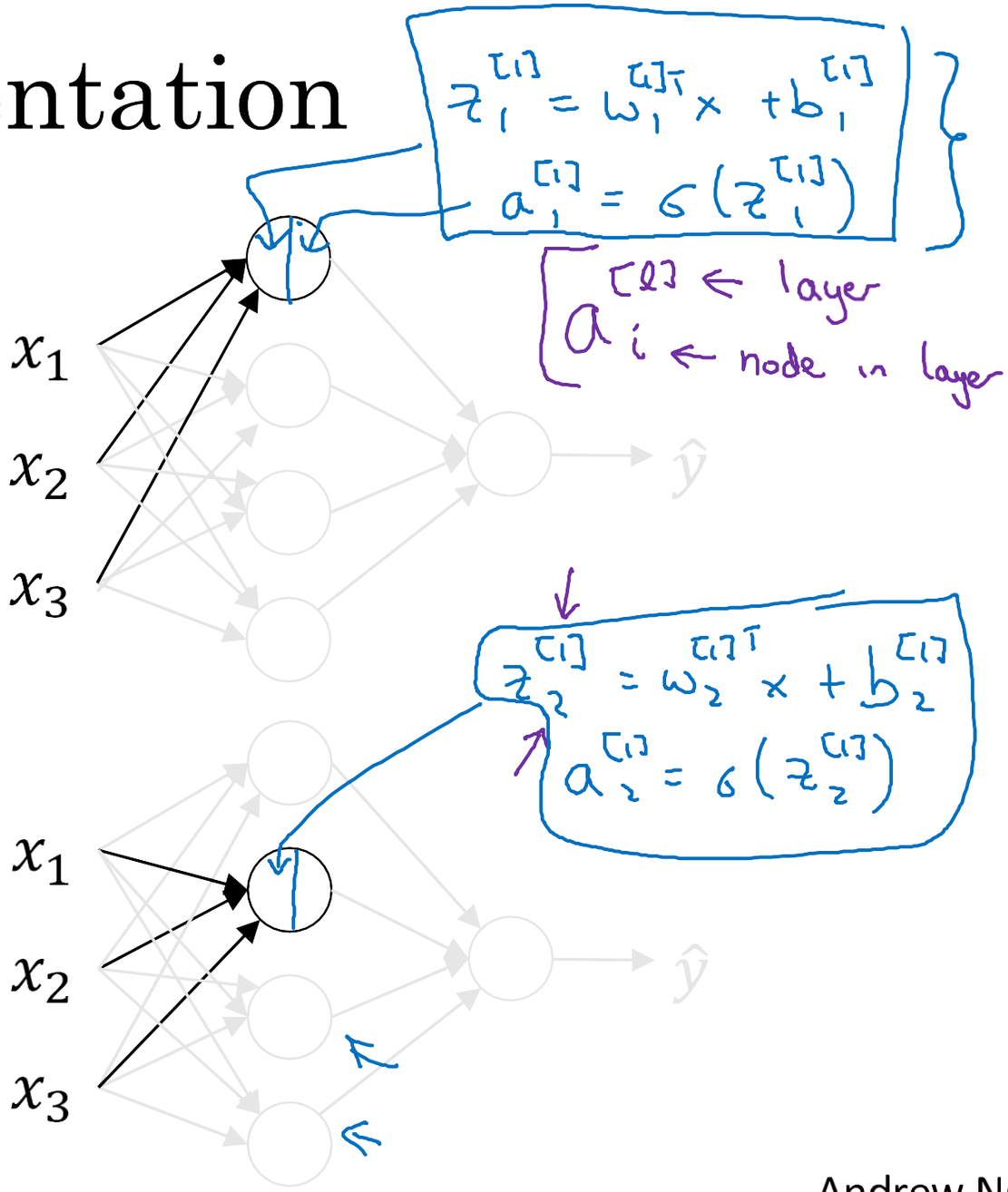


Neural Network Representation



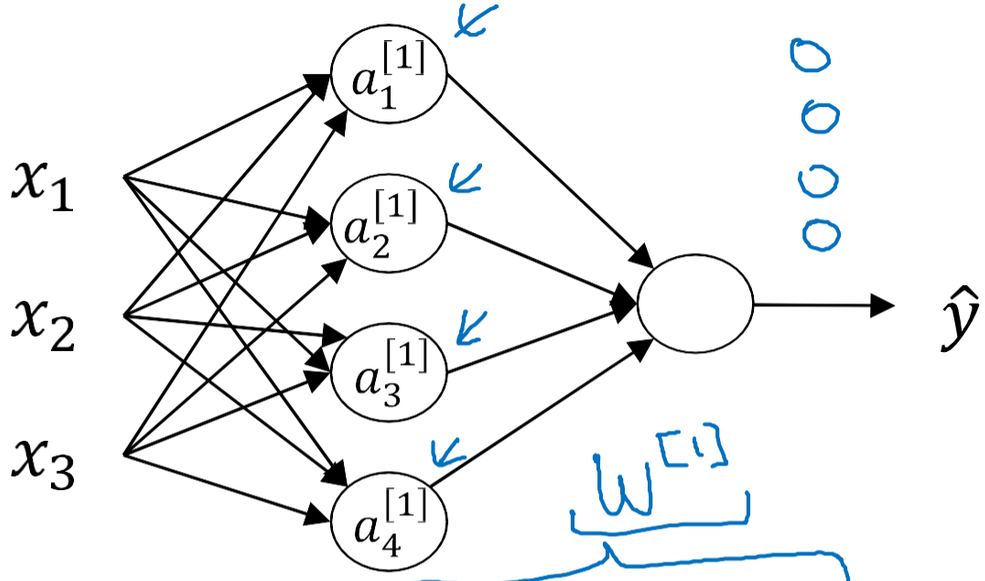
$$z = w^T x + b$$

$$a = \sigma(z)$$



Andrew Ng

Neural Network Representation



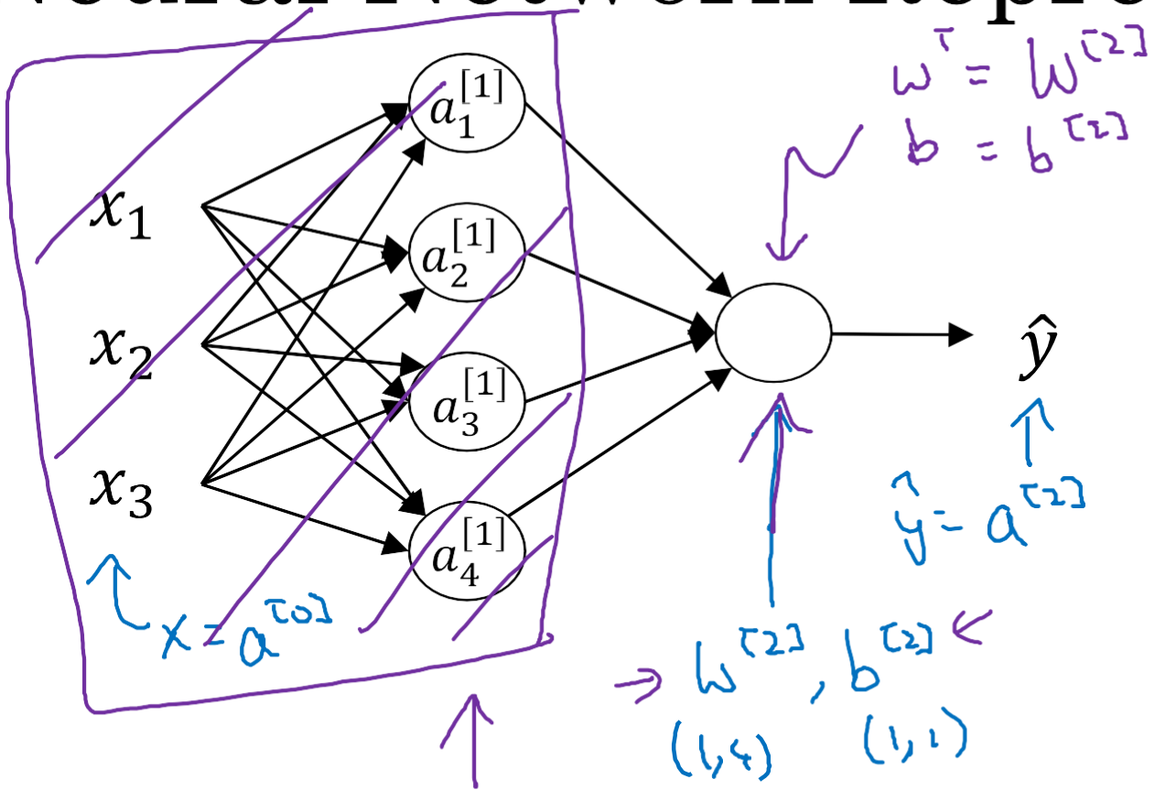
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ (4, 3) $\quad b^{[1]}$ (4, 1)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Neural Network Representation learning



Given input x:

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} a^{[0]}_{(3,1)} + b^{[1]}_{(4,1)}$$

$$a^{[1]}_{(4,1)} = \sigma(z^{[1]}_{(4,1)})$$

$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$

$$a^{[2]}_{(1,1)} = \sigma(z^{[2]}_{(1,1)})$$

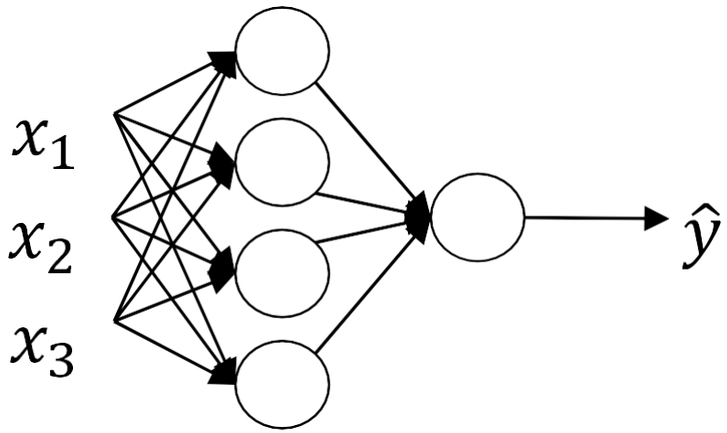
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

$x \rightarrow a^{[2]} = \hat{y}$

$x^{(1)} \rightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \rightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[2](i)}$ ← example i, ← layer 2

for $i = 1$ to m,

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

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

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

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

Vectorizing across multiple examples

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

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

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

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

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

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

$$A^{[1]} = g(Z^{[1]})$$

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

$$A^{[2]} = g(Z^{[2]})$$

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

training examples

hidden units.

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

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

#hidden units

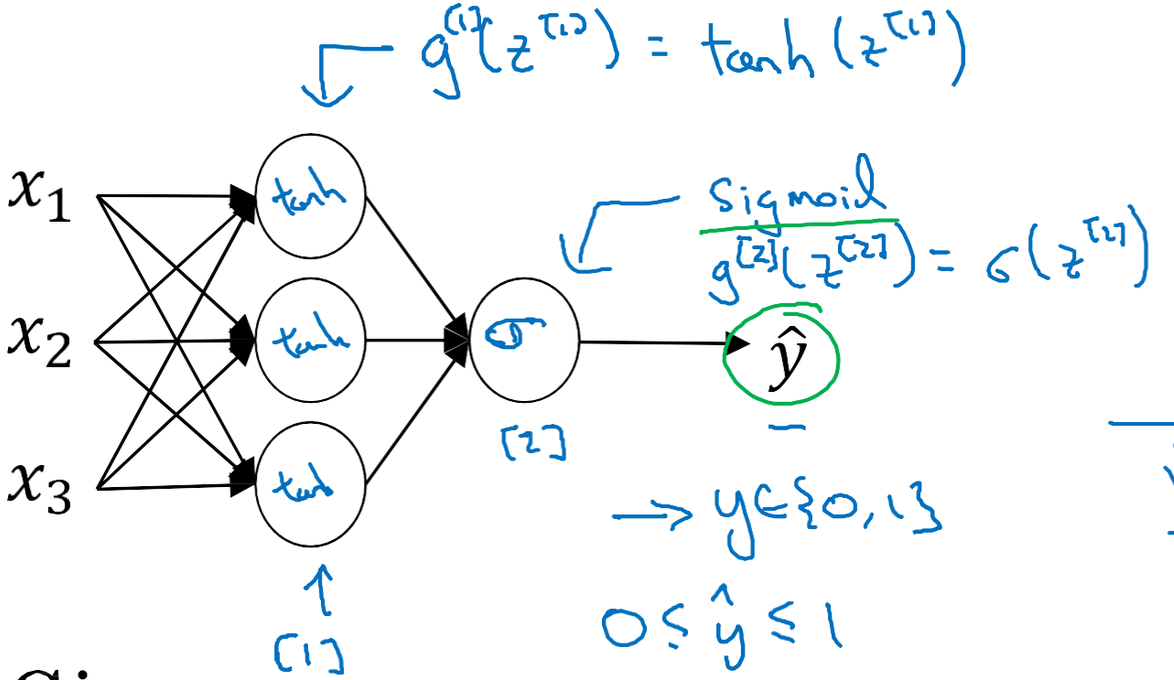
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions



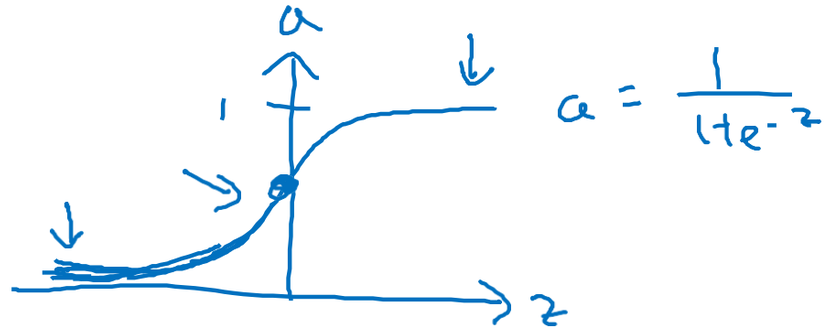
$g^{[1]}(z^{[1]}) = \tanh(z^{[1]})$

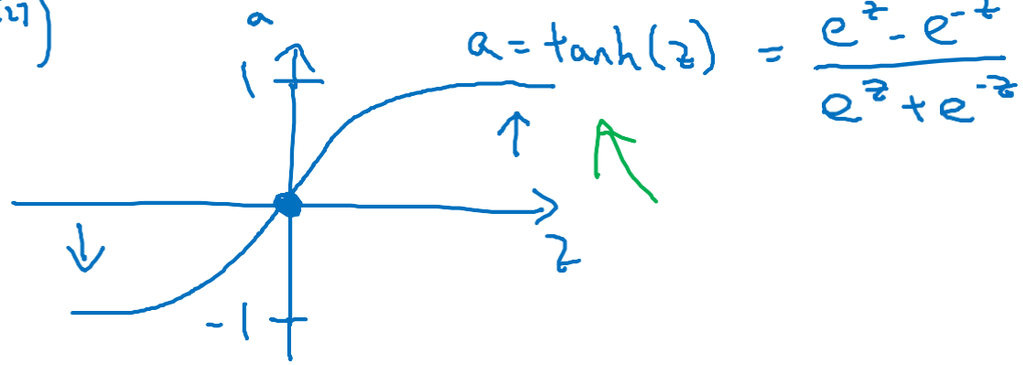
Sigmoid
$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

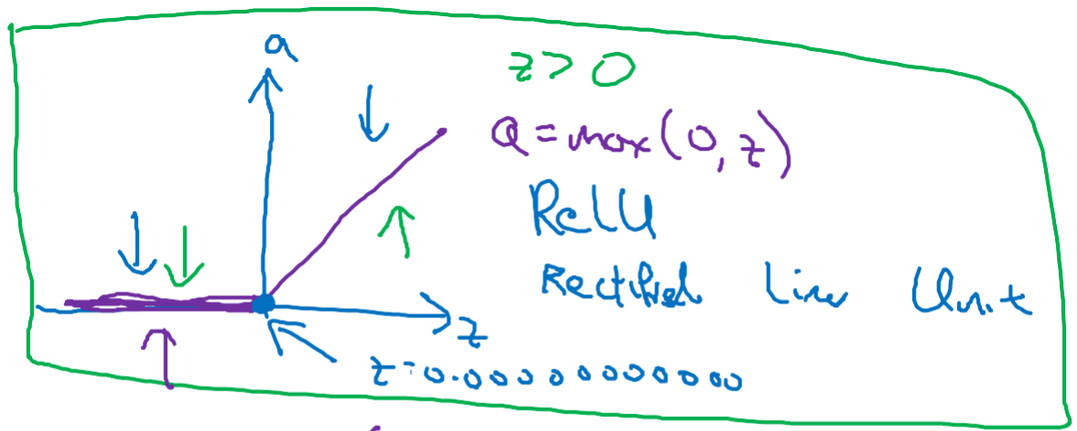
$\rightarrow y \in \{0, 1\}$

$0 \le \hat{y} \le 1$

$-1 \quad 1$







Given x:

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

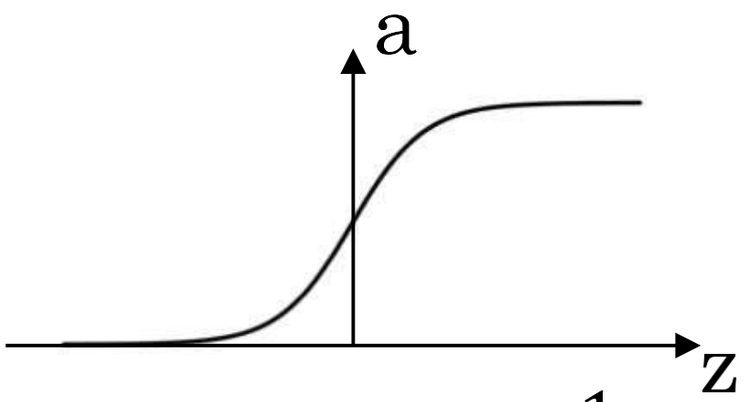
$$a^{[1]} = \sigma(z^{[1]}) \quad g^{[1]}(z^{[1]})$$

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

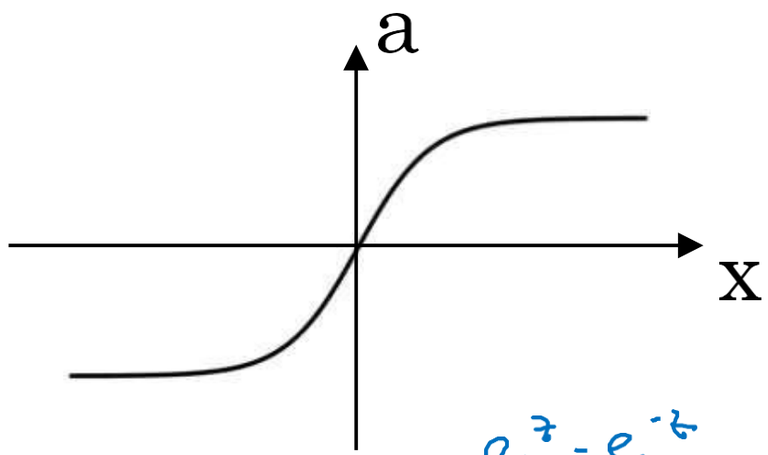
$$a^{[2]} = \sigma(z^{[2]}) \quad g^{[2]}(z^{[2]})$$

Andrew Ng

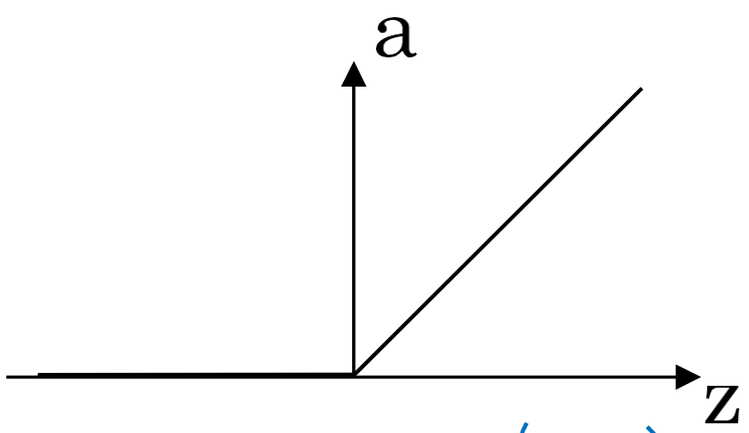
Pros and cons of activation functions



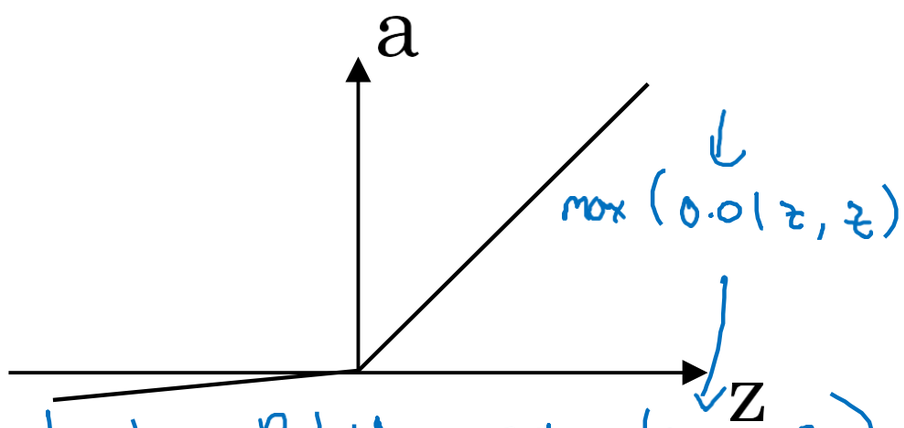
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



tanh: $a = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{z}}$



ReLU $a = \max(0, z)$



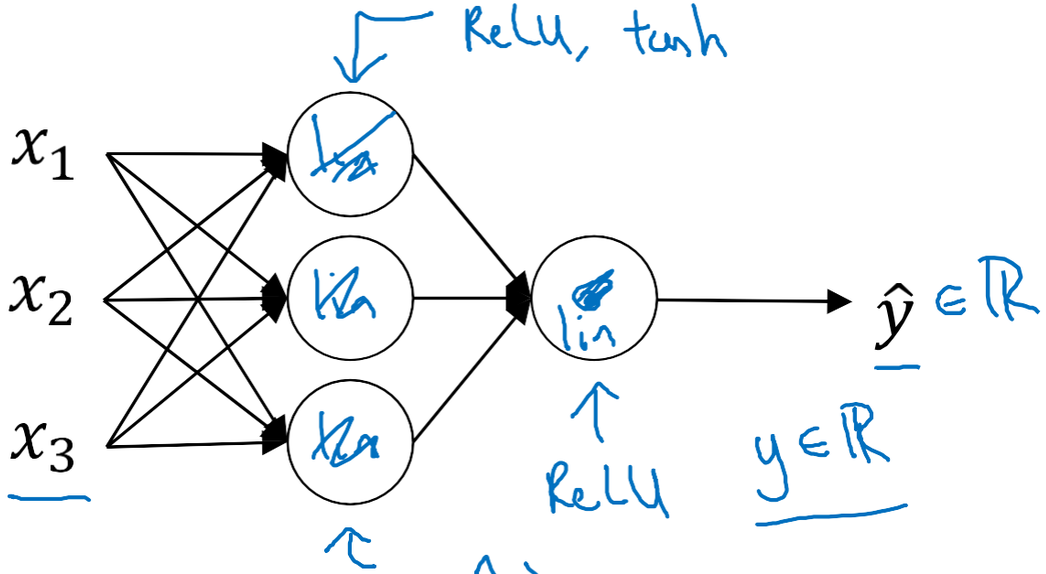
Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



Given x:

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

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

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

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

$g(z) = z$ "linear activation function"

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

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

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

$$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$$

$$= W'x + b'$$

$g(z) = z$

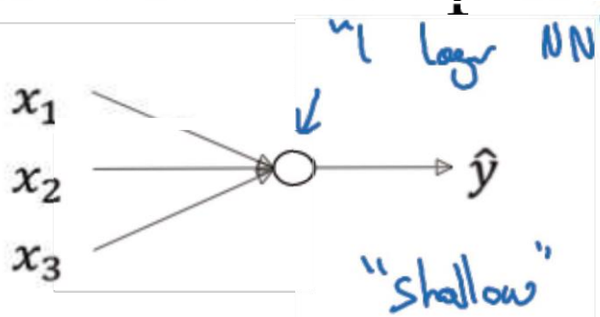
Andrew Ng

deeplearning.ai

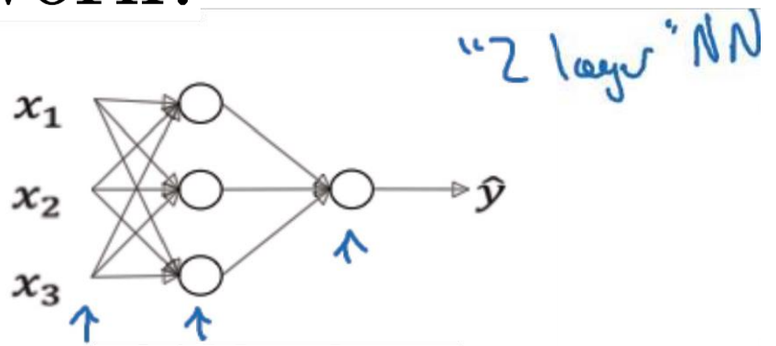
Deep Neural Networks

Deep L-layer Neural network

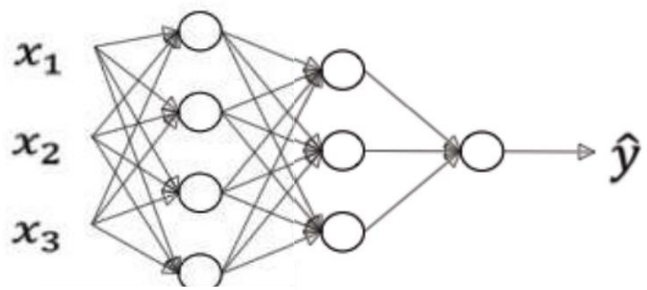
What is a deep neural network?



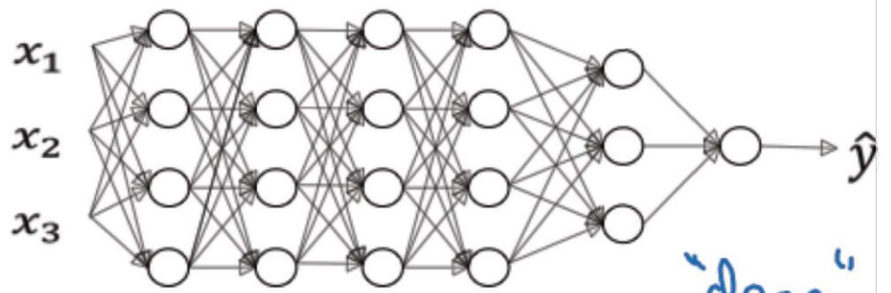
logistic regression



1 hidden layer



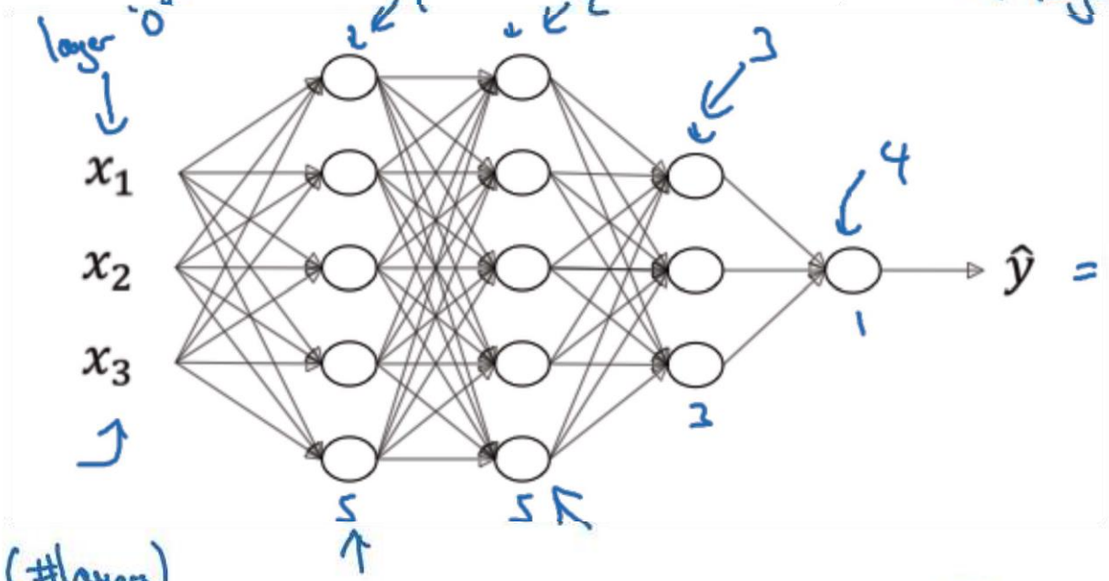
2 hidden layers



5 hidden layers

Andrew Ng

Deep neural network notation



$l = 4$ (#layers)

$n^{[l]}$ = #units in layer l

$n^{[1]} = 5$, $n^{[2]} = 5$, $n^{[3]} = 3$, $n^{[4]} = n^{[L]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activations in layer l

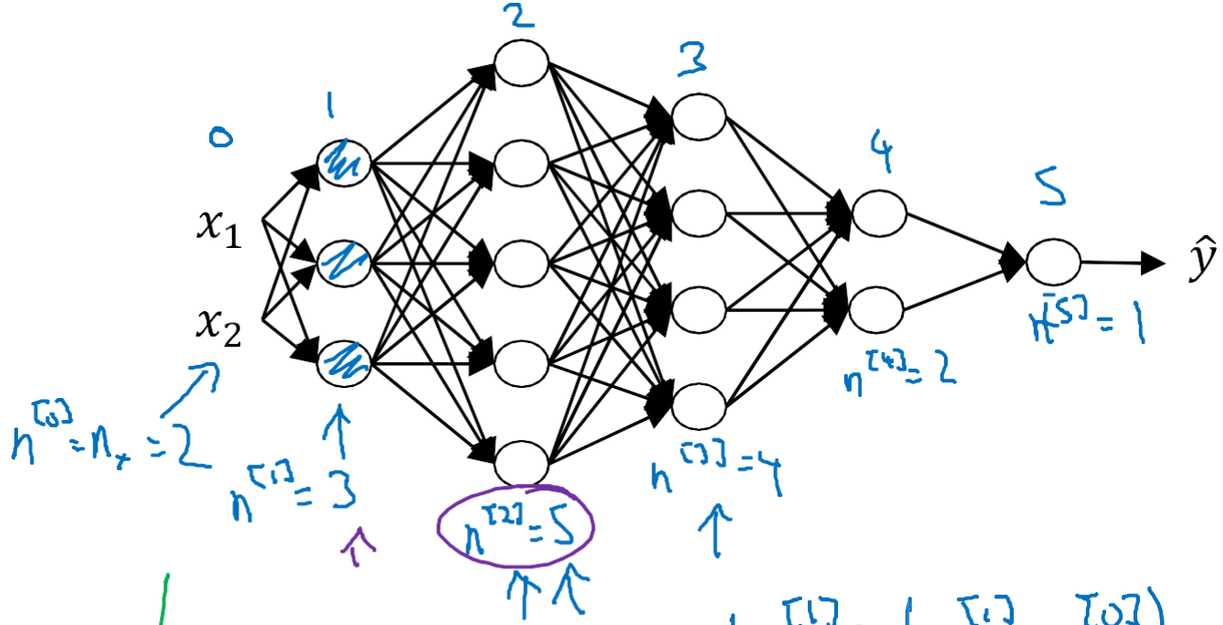
$a^{[l]} = g^{[l]}(z^{[l]})$. $w^{[l]}$, $b^{[l]}$ = weights for $z^{[l]}$

deeplearning.ai

Deep Neural Networks

Getting your matrix dimensions right

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



$L = 5$

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

$a^{[l]}$

- $w^{[l]} : (n^{[l]}, n^{[l-1]})$
- $b^{[l]} : (n^{[l]}, 1)$
- $dw^{[l]} : (n^{[l]}, n^{[l-1]})$
- $db^{[l]} : (n^{[l]}, 1)$

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

$(3,1) \leftarrow (3,2) \quad (2,1) \quad (3,1)$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, 1) \quad (n^{[1]}, 1)$

$w^{[1]} : (n^{[1]}, n^{[0]})$

$w^{[2]} : (5, 3) \quad (n^{[2]}, n^{[1]})$

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

$\rightarrow (5,1) \quad (5,3) \quad (3,1) \quad (5,1)$

$(n^{[2]}, 1)$

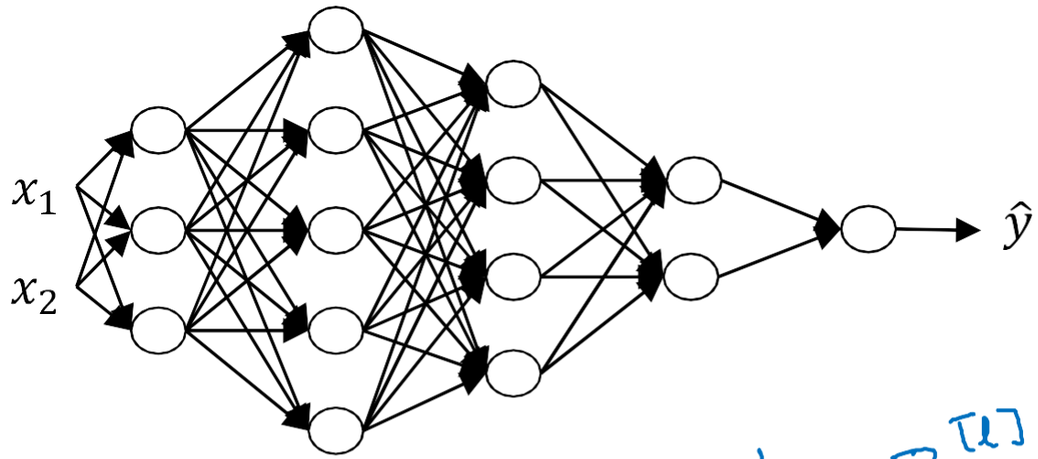
$w^{[3]} : (4, 5)$

$w^{[4]} : (2, 4) \quad , \quad w^{[5]} : (1, 2)$

$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$

Andrew Ng

Vectorized implementation



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

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, 1) \quad (n^{[1]}, 1)$

$$\begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \end{bmatrix}$$

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

$(n^{[1]}, m) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, m) \quad (n^{[1]}, 1) \rightarrow (n^{[1]}, m)$

$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l = 0 \quad A^{[0]} = X = (n^{[0]}, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

What are hyperparameters?

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyperparameters: learning rate α

#iterations

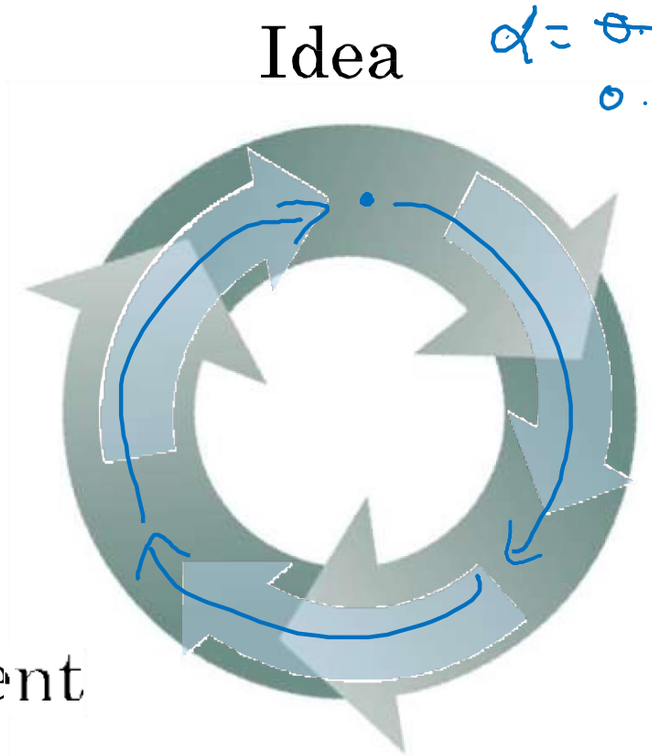
hidden layers L

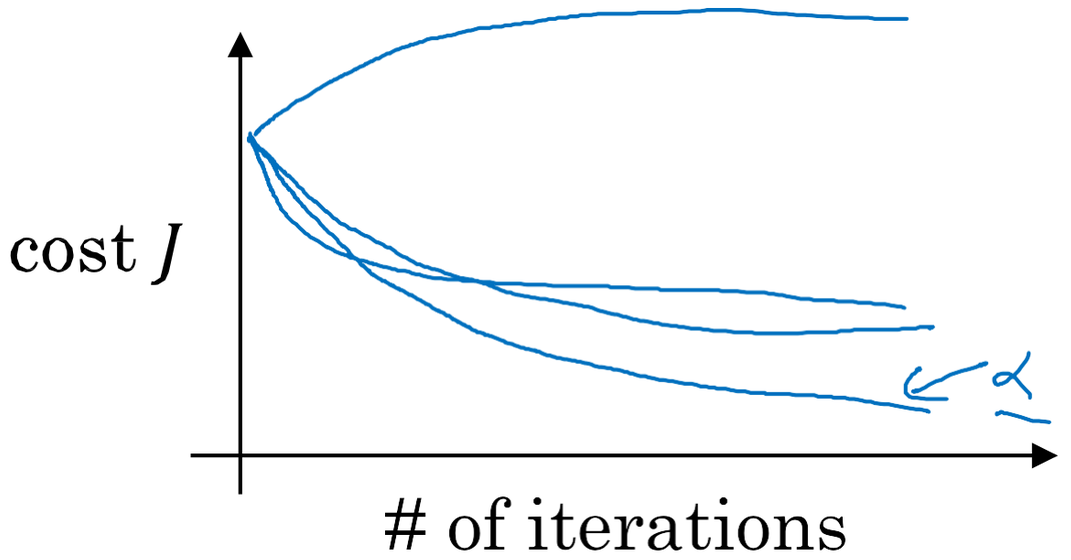
hidden units $n^{[1]}, n^{[2]}, \ldots$

Choice of activation function

Later: Momentum, mini-batch size, regularizations, ...

Applied deep learning is a very empirical process





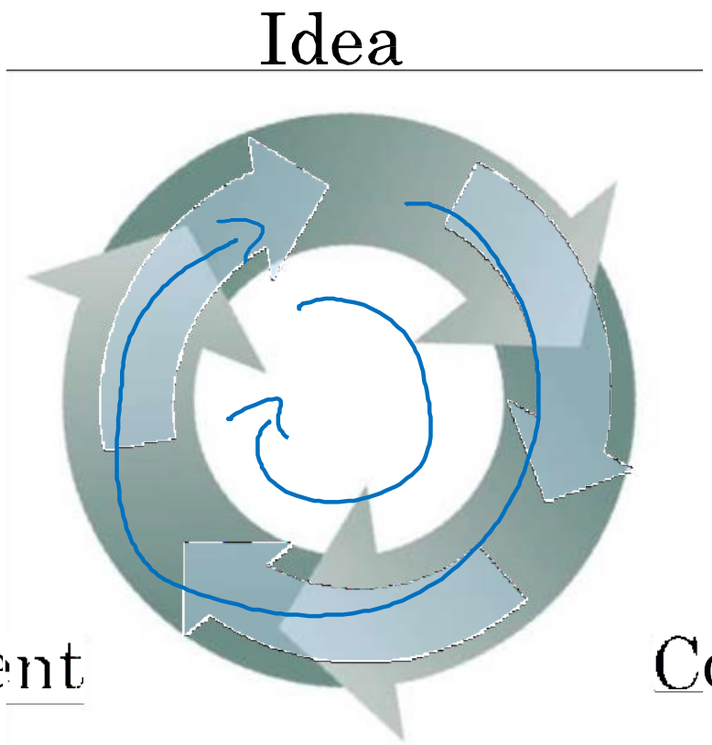
deeplearning.ai

Setting up your ML application

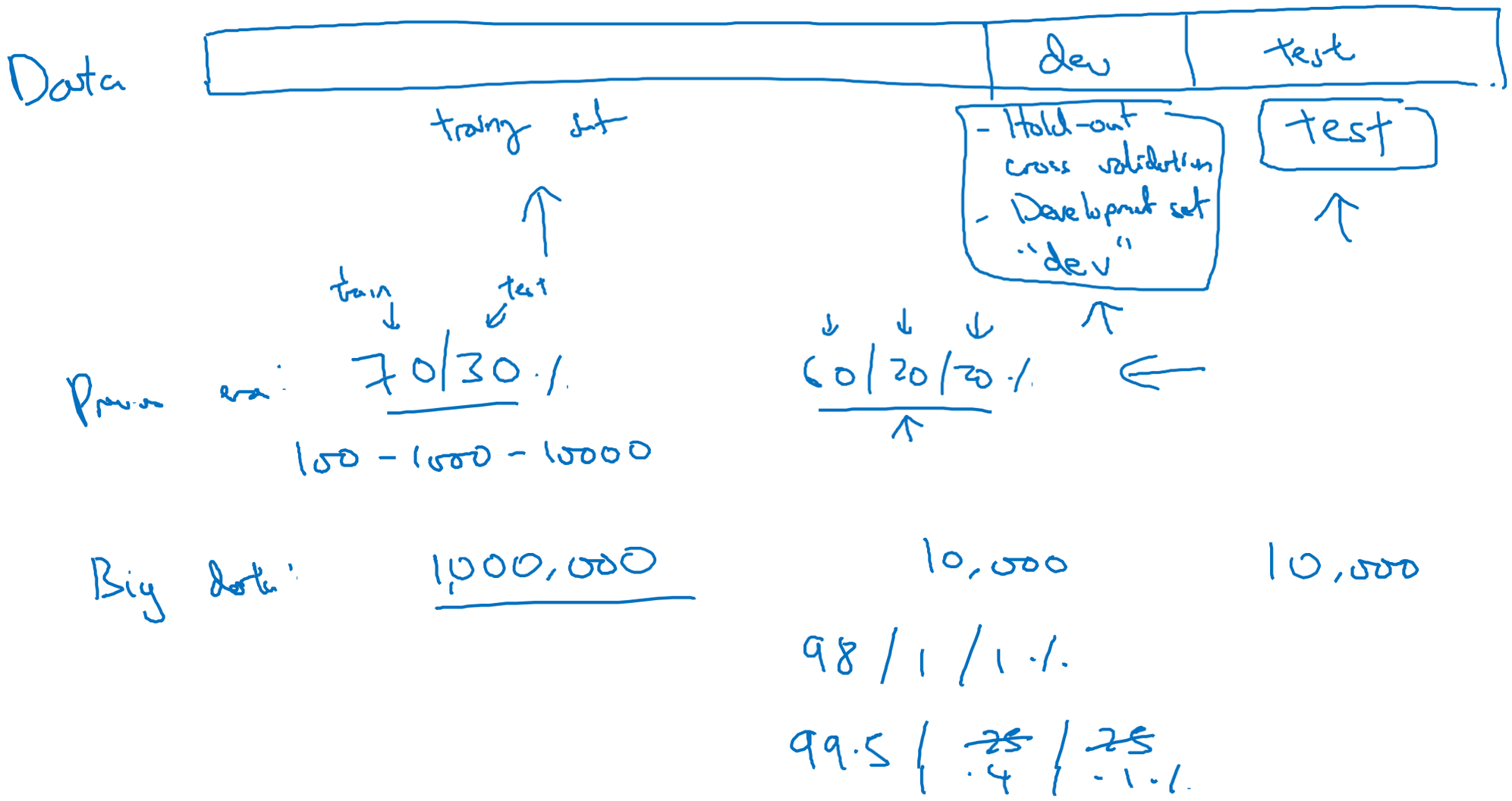
Train/dev/test sets

Applied ML is a highly iterative process

- # layers
- # hidden units
- learning rates
- activation functions
- ...



Train/dev/test sets



Mismatched train/test distribution

Cats

Training set:
Cat pictures from webpages

Dev/test sets:
Cat pictures from users using your app

→ Make sure dev and test come from same distribution.

"test"

train / dev

train / test

→ train / dev

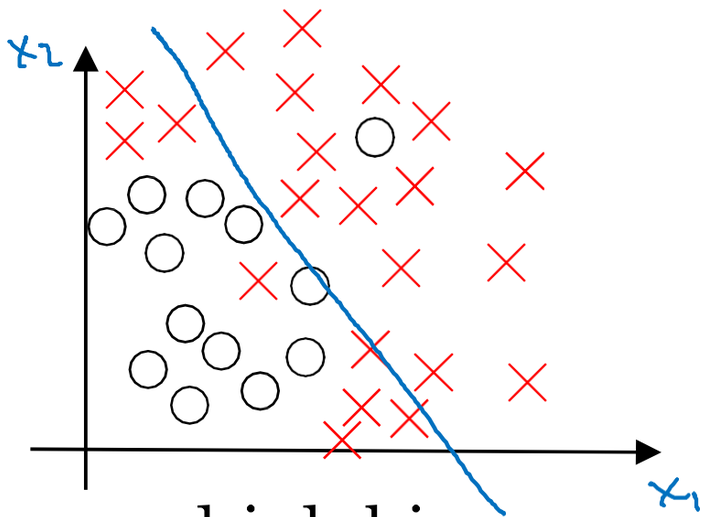
Not having a test set might be okay. (Only dev set.)

deeplearning.ai

Setting up your ML application

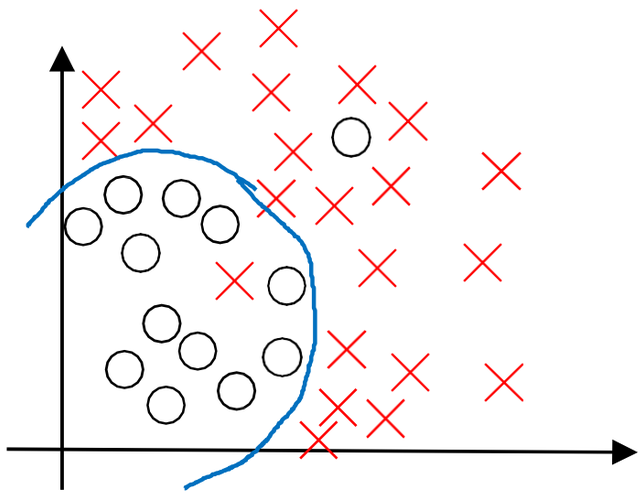
Bias/Variance

Bias and Variance

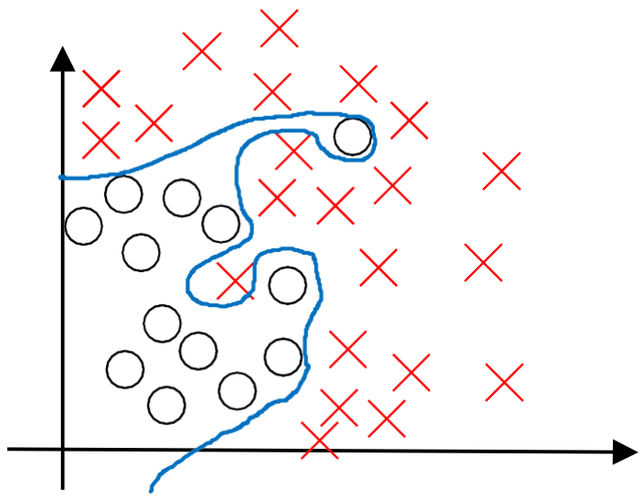


high bias

Underfitting



→ "just right"



high variance

Overfitting

Bias and Variance

Cat classification

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human ≈0%

Optimal (Bayes) error: ~~0%~~ 15%

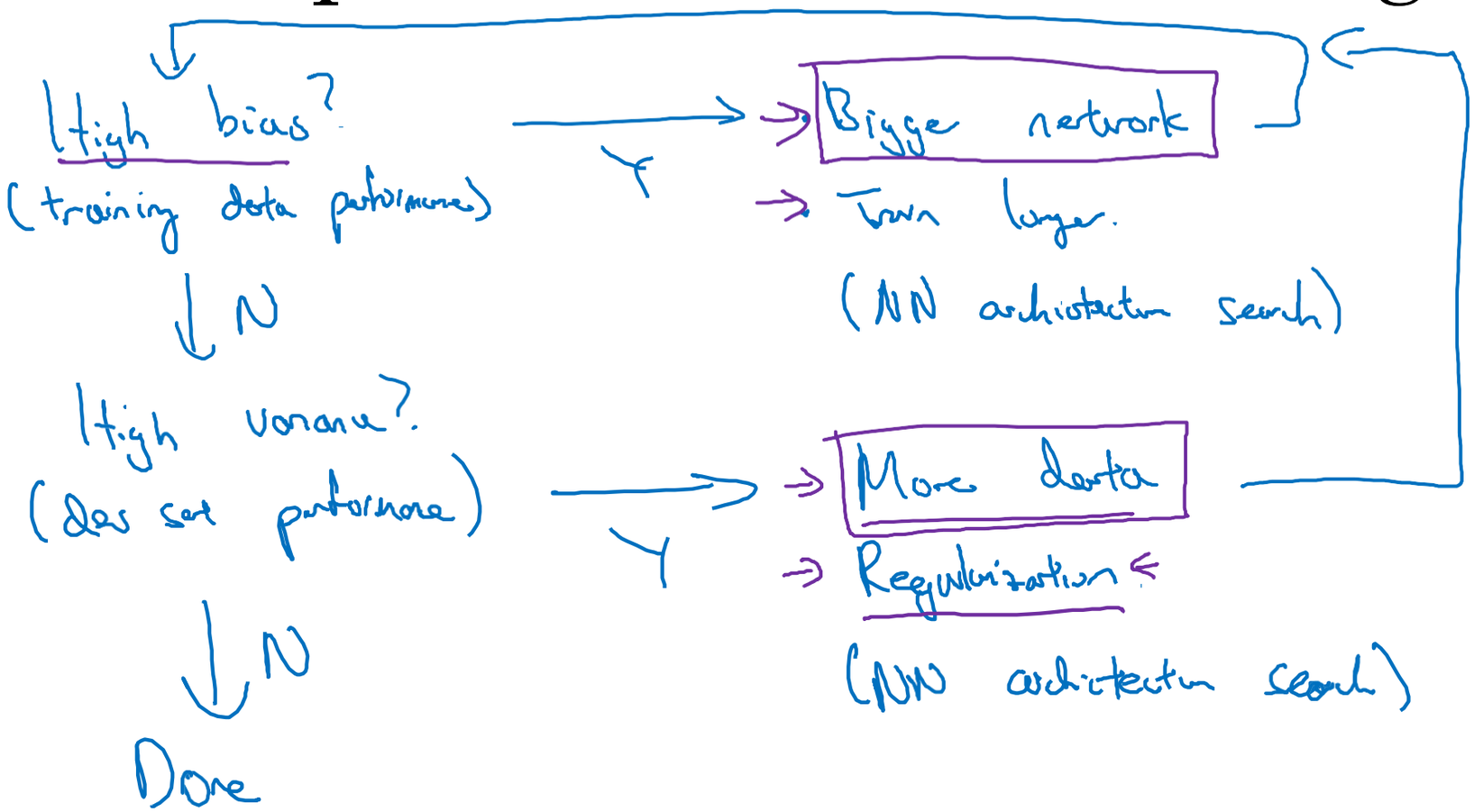
Blurry images

deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning



deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

$\min_{w,b} J(w,b)$

$w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

λ = regularization parameter

lambda

lambd

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

L_2 regularization $\quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

Neural network

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

$w^{[l]} : (n^{[l]}, n^{[l-1]})$

"Frobenius norm" $\|\cdot\|_2^2$ $\|\cdot\|_F^2$

$$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

$$\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}$$

"Weight decay"

$$w^{[l]} := w^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \right]$$

$$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha\, (\text{from backprop})$$

$$= \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{<1} w^{[l]} - \alpha\, (\text{from backprop})$$

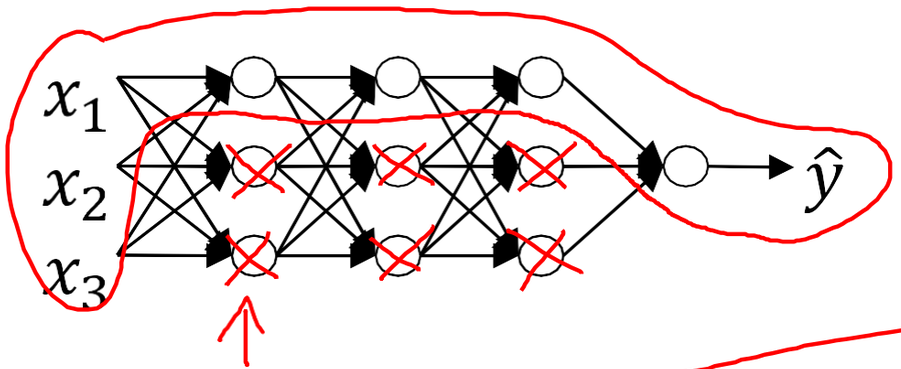
Andrew Ng

deeplearning.ai

Regularizing your neural network

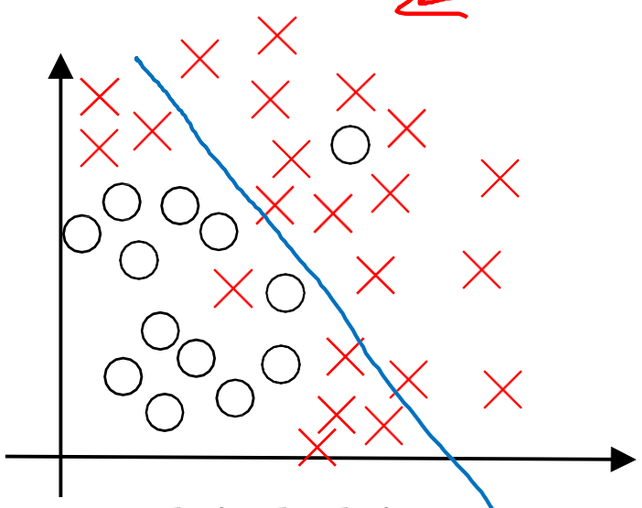
Why regularization reduces overfitting

How does regularization prevent overfitting?

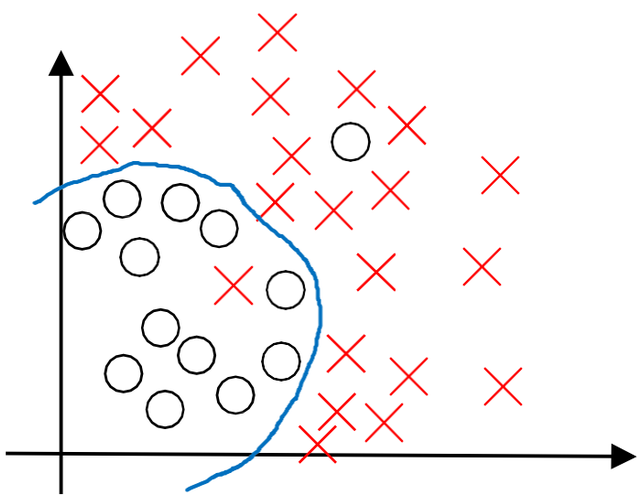


$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

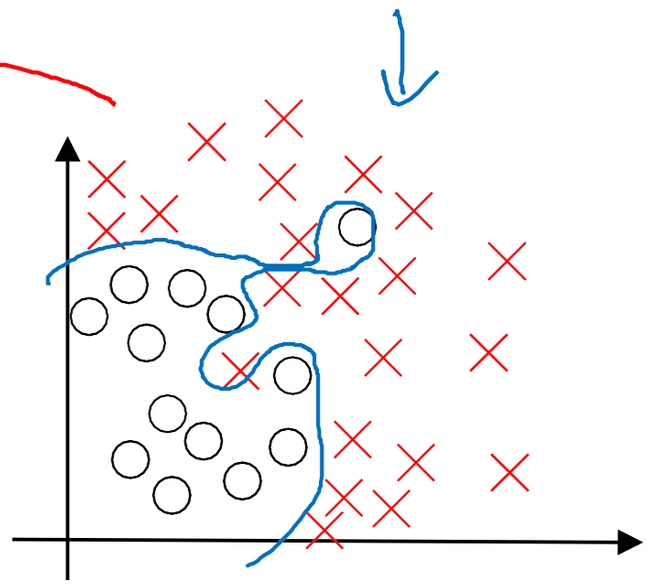
$w^{[l]} \approx 0$



high bias



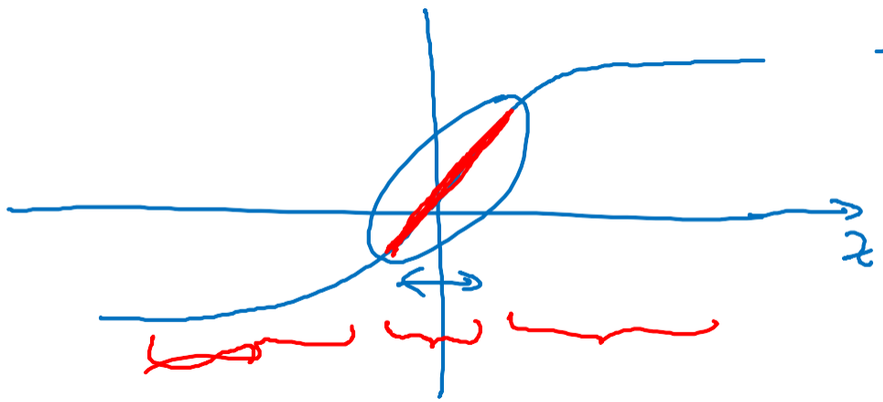
"just right"



high variance

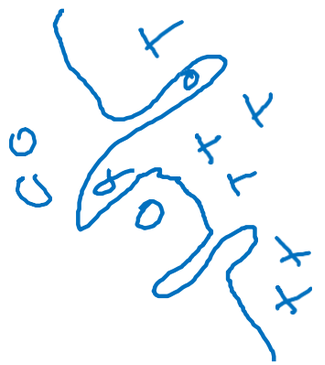
Andrew Ng

How does regularization prevent overfitting?



$g(z) = \tanh(z)$

$\lambda \uparrow \quad W^{[l]} \downarrow \quad z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$



Every layer $\approx$ linear

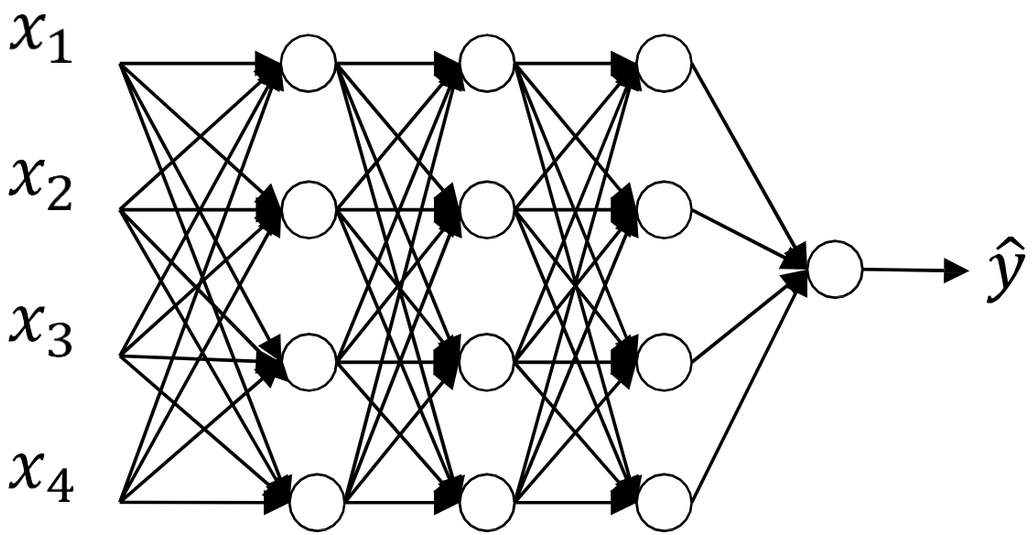
$$J(\dots) = \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_l \|W^{[l]}\|_F^2$$

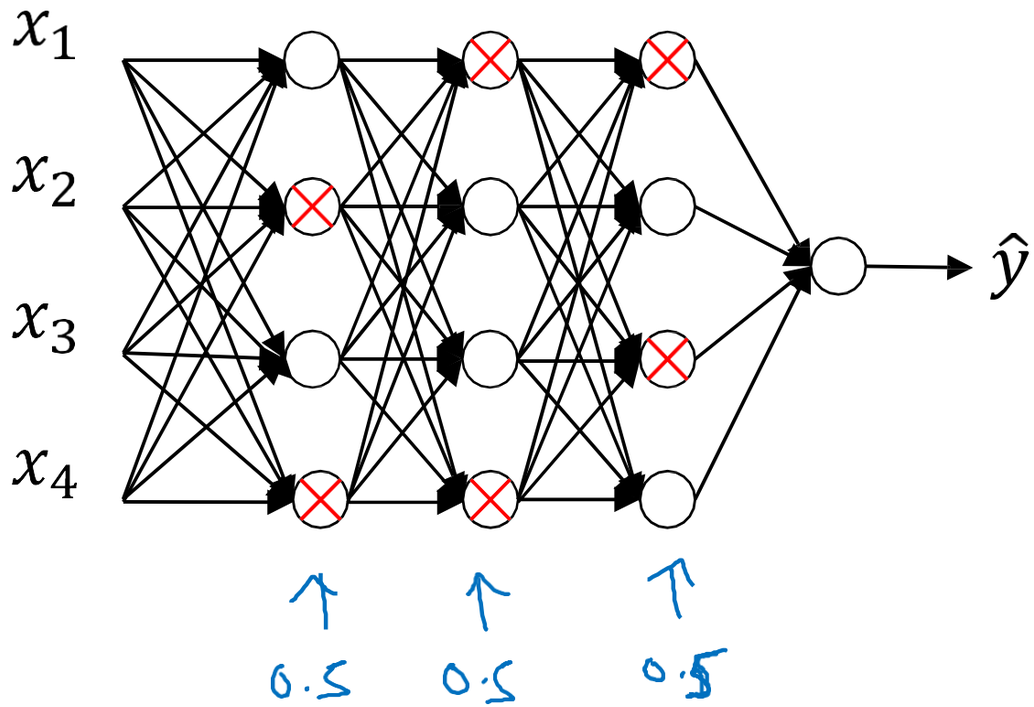
deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization





Implementing dropout ("Inverted dropout")

Illustrate with layer $l = 3$. keep-prob = 0.8 (0.2)

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\text{-}prob$

$a3 = np.multiply(a3, d3)$ # $a3 *= d3$.

$a3 /= keep\text{-}prob$ (~~0.8~~)

50 units → 10 units shut off

$z^{[4]} = w^{[4]} \cdot a^{[3]} + b^{[4]}$

$a^{[3]}$ reduced by 20%.

$/= 0.8$

Test

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

No drop out.

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

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

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

$a^{[3]} = \ldots$

$\hat{y}$

$/= keep\text{-}prob$

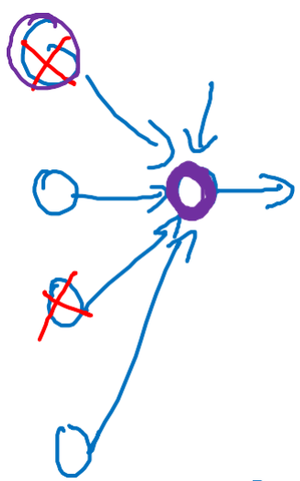
deeplearning.ai

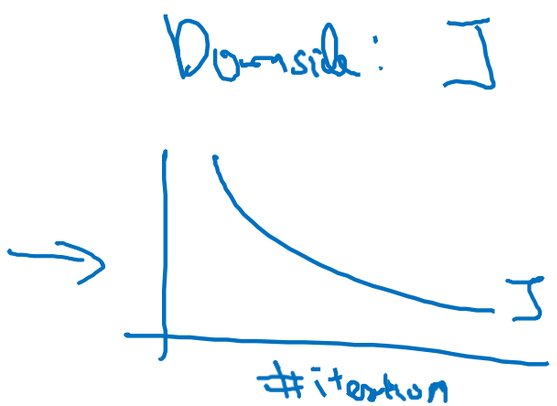
Regularizing your neural network

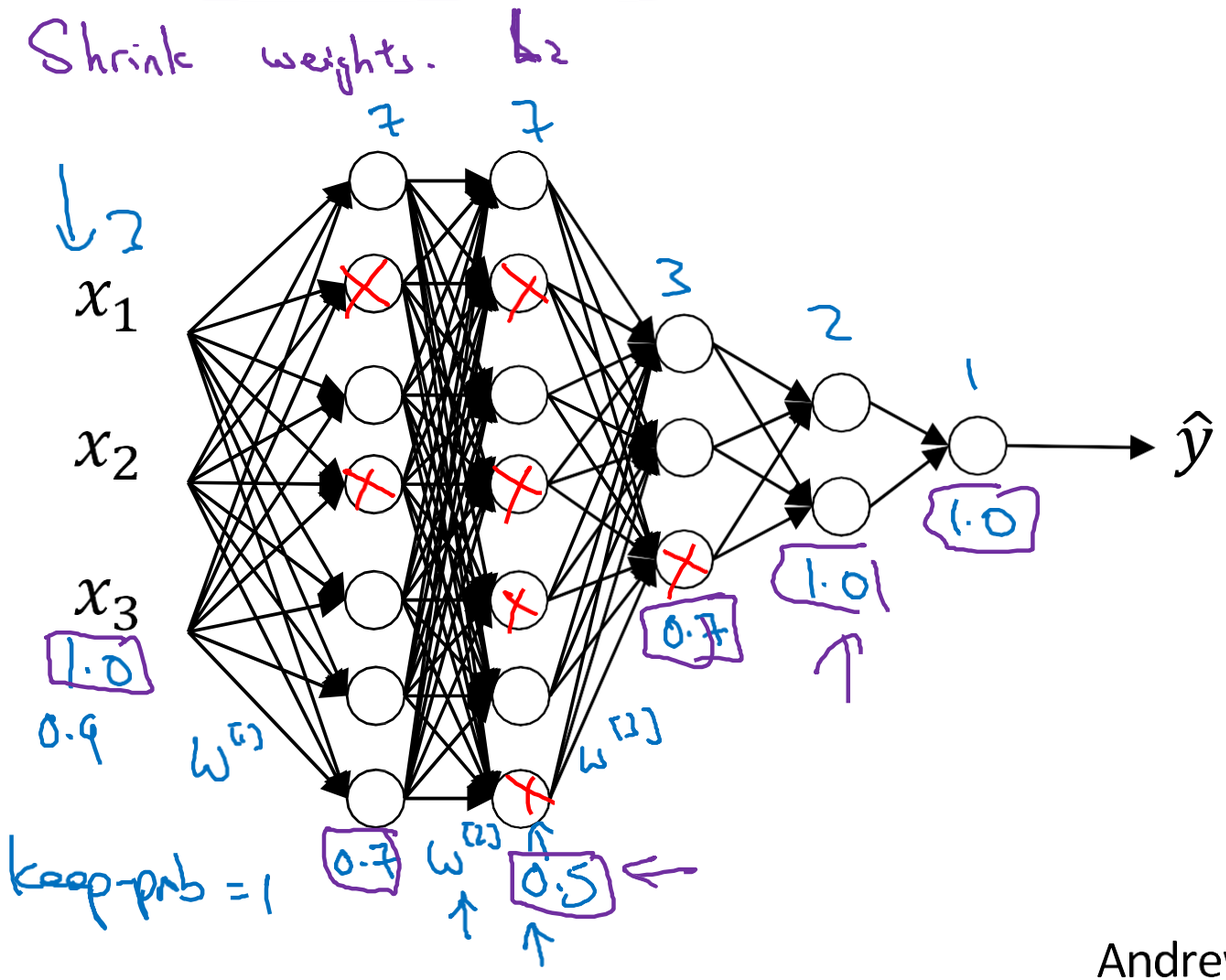
Understanding dropout

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights.







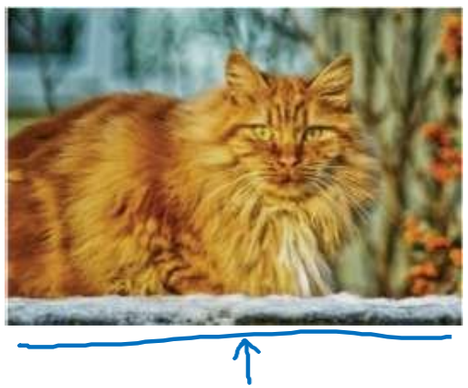
Andrew Ng

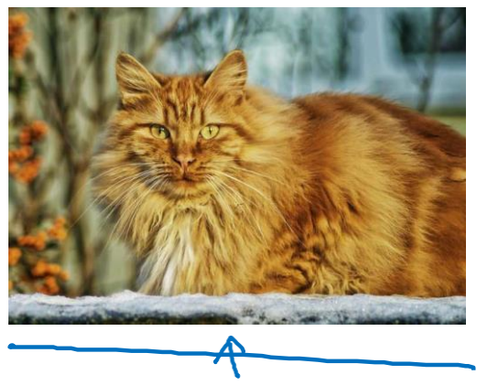
deeplearning.ai

Regularizing your neural network

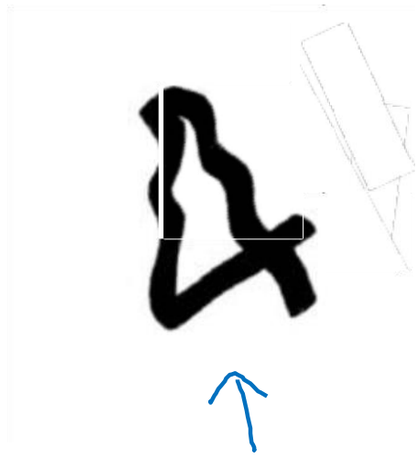
Other regularization methods

Data augmentation



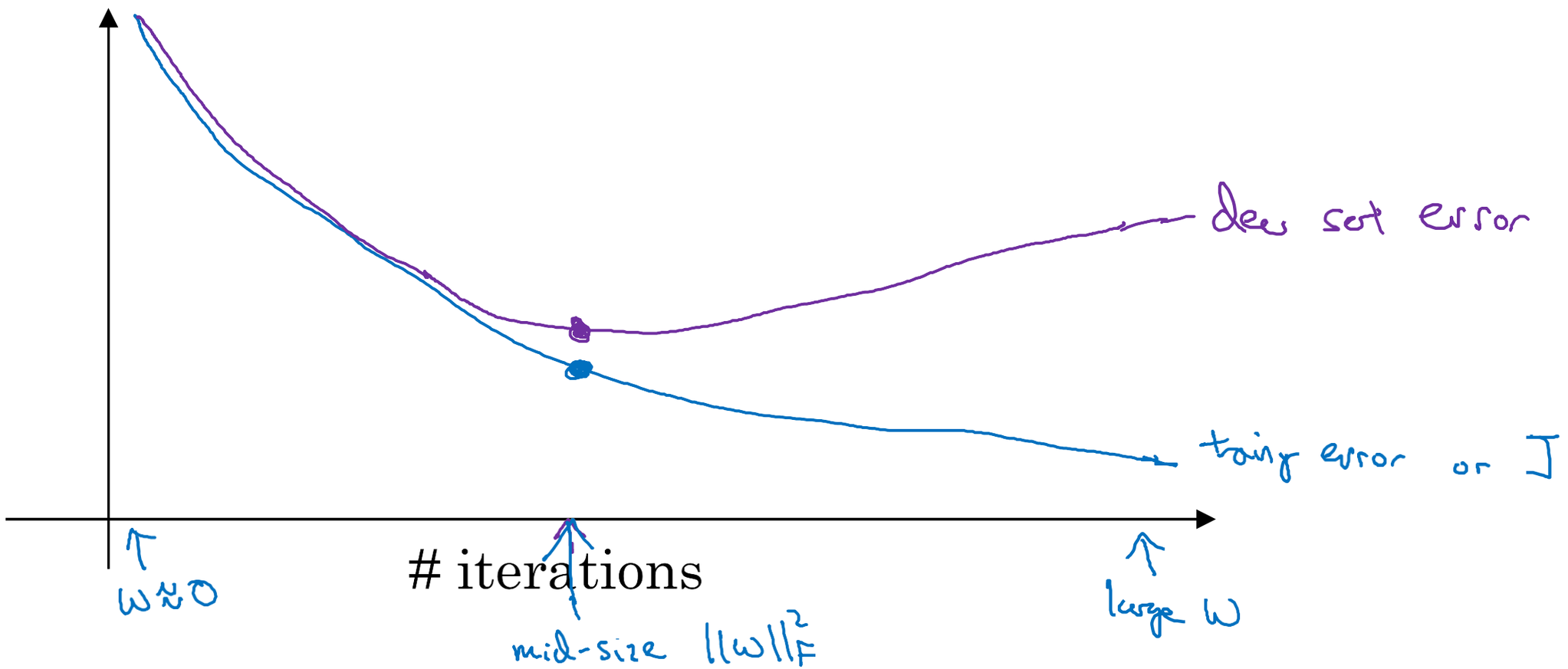


4



Andrew Ng

Early stopping



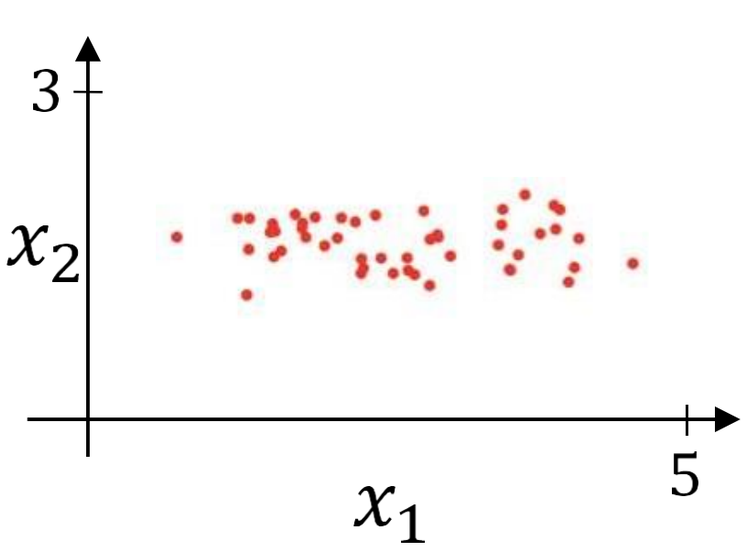
deeplearning.ai

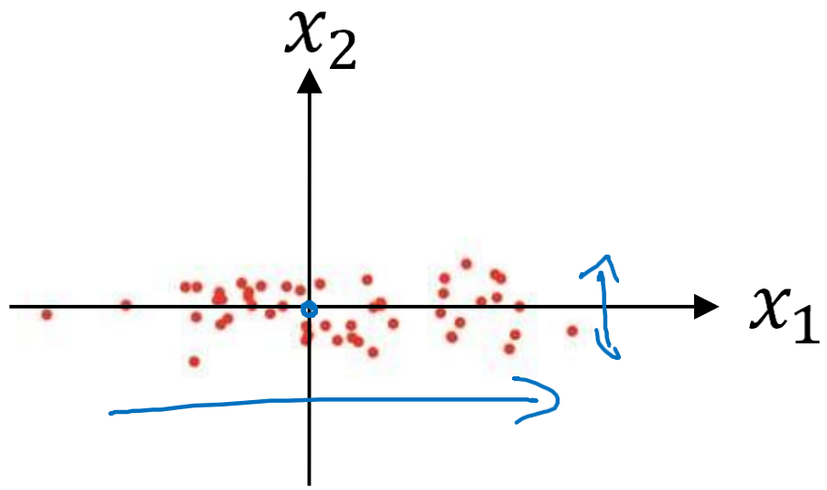
Setting up your optimization problem

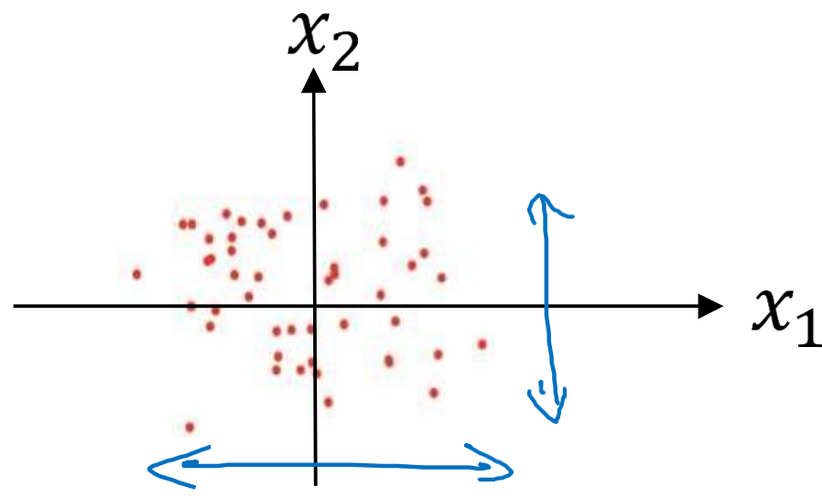
Normalizing inputs

Normalizing training sets

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$







Subtract mean:

$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$

$x := x - \mu$

Normalize variance

$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} **2$

↑ element-wise

$x /= \sigma$

Use same μ, σ to normalize test set.

Andrew Ng

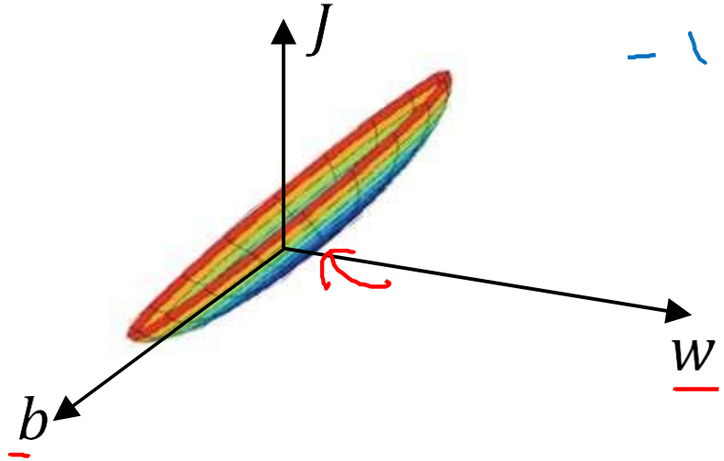
Why normalize inputs?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

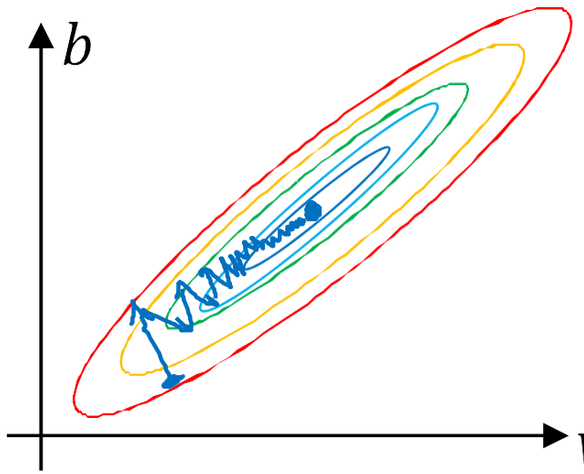
w_1 x_1 : 1 ... 1000 ←

w_2 x_2 : 0 ... 1 ←

Unnormalized:



−1 ... 1

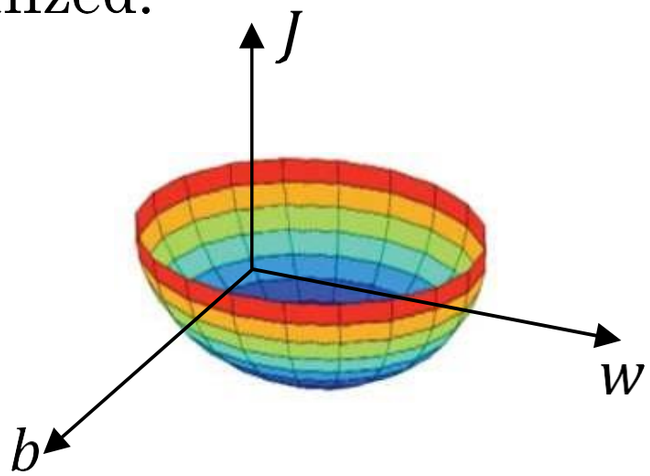


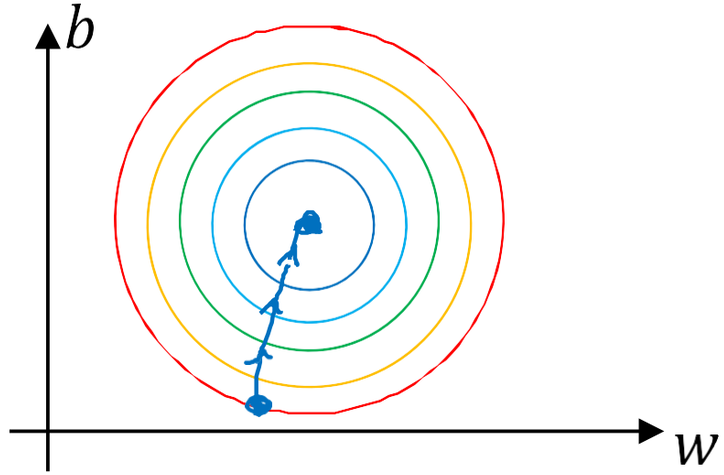
x_1 : 0 ... 1

x_2 : −1 ... 1

x_3 : 1 ... 2

Normalized:





Andrew Ng

deeplearning.ai

Optimization Algorithms

Mini-batch gradient descent

Batch vs. mini-batch gradient descent

X, Y $\quad$ $X^{\{t\}}, Y^{\{t\}}$.

Vectorization allows you to efficiently compute on m examples.

$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)}\ \cdots\ x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)}\ \cdots\ x^{(2000)}}_{X^{\{2\}}\ (n_x,1000)} \mid \cdots \mid \underbrace{\cdots\ x^{(m)}}_{X^{\{5,000\}}\ (n_x,1000)}]$

(n_x, m)

$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)}\ \cdots\ y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)}\ \cdots\ y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \cdots \mid \underbrace{\cdots\ y^{(m)}}_{Y^{\{5,000\}}\ (1,1000)}]$

$(1, m)$

What if $m = 5{,}000{,}000$?

5,000 mini-batches of 1,000 each

Mini-batch t: $X^{\{t\}}, Y^{\{t\}}$

$x^{(i)}$

$z^{[l]}$

$X^{\{t\}}, Y^{\{t\}}$.

Mini-batch gradient descent

1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$.
(as if $m = 1000$)

Repeat {
  for $t = 1, \ldots, 5000$ {

    Forward prop on $X^{\{t\}}$.

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Vectorized implementation (1000 examples)

X, Y

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$ from $X^{\{t\}}, Y^{\{t\}}$.

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

  }
}

"1 epoch"
└ pass through training set.

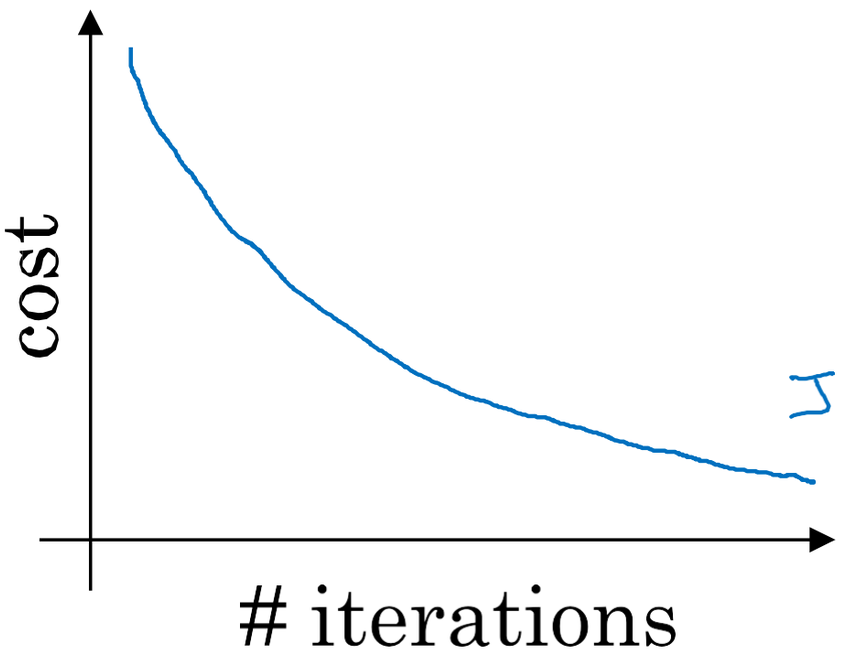
deeplearning.ai

Optimization Algorithms

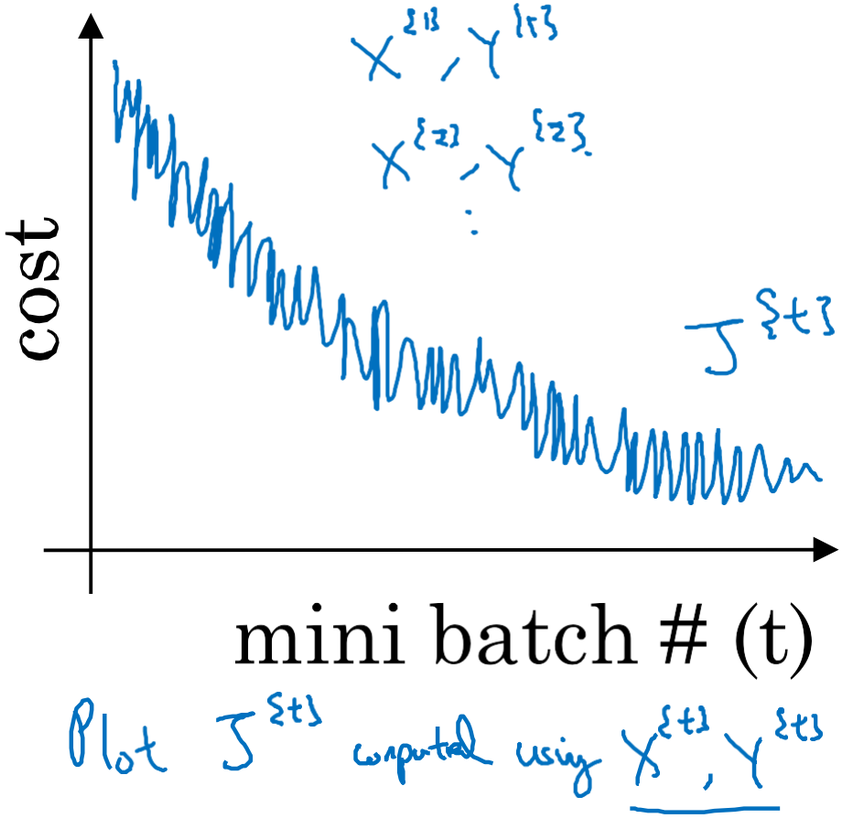
Understanding mini-batch gradient descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent

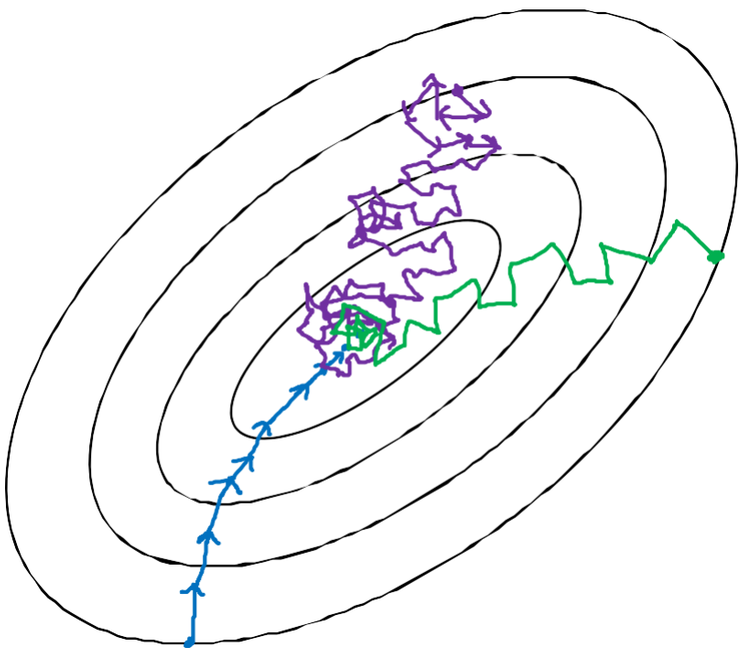


Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own mini-batch. $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ .. $(x^{(2)}, y^{(1)})$

In practice: Somewhere in-between 1 and m



Stochastic gradient descent

{

Lose speedup from vectorization

In-between (mini-batch size not too big/small)

{

Fastest learning.

- Vectorization. (n 1000)
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)

{

Too long per iteration

Choosing your mini-batch size

If small train set: Use batch gradient descent.
($m \leq 2000$)

Typical mini-batch sizes:

→ 64, 128, 256, 512 1024
2^6 2^7 2^8 2^9 2^{10}

Make sure mini-batch fit in CPU/GPU memory.

$X^{\{t\}}, Y^{\{t\}}$

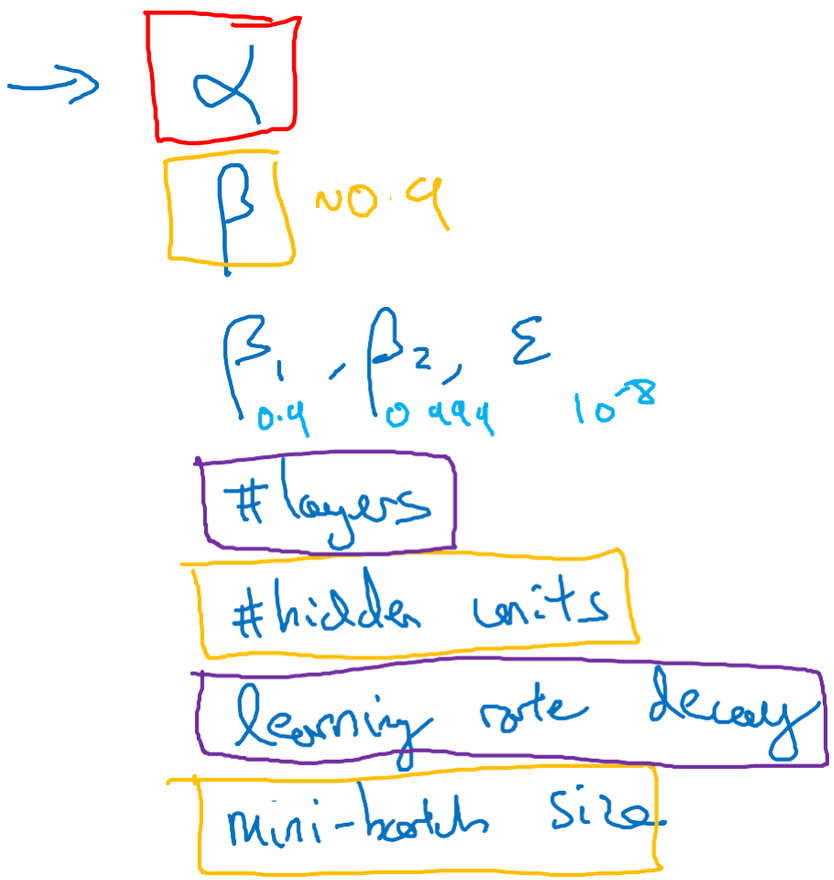
here

deeplearning.ai

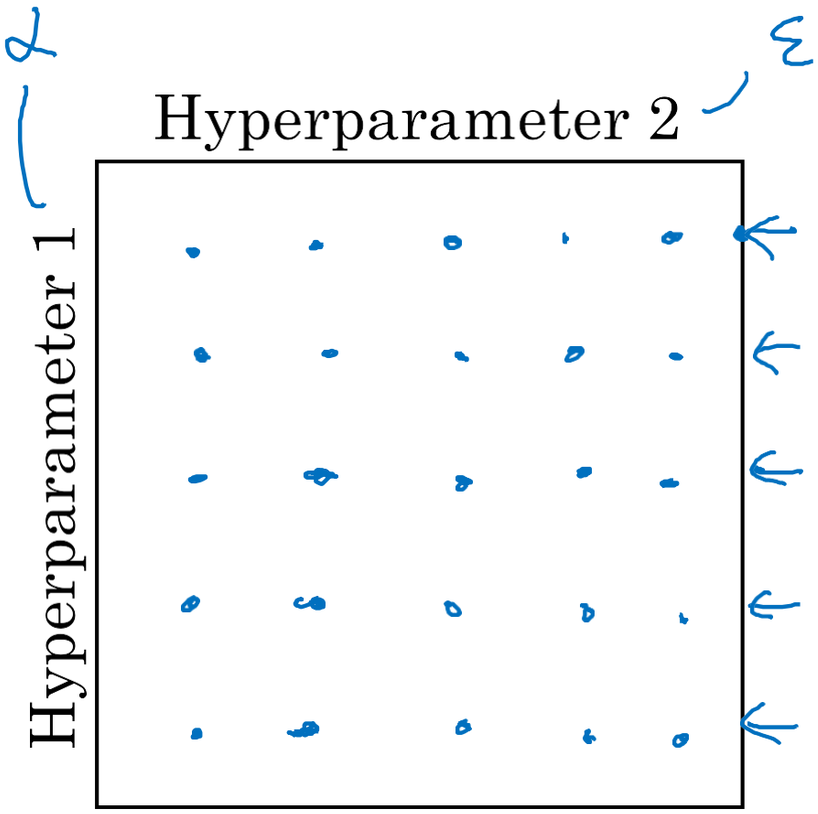
Hyperparameter tuning

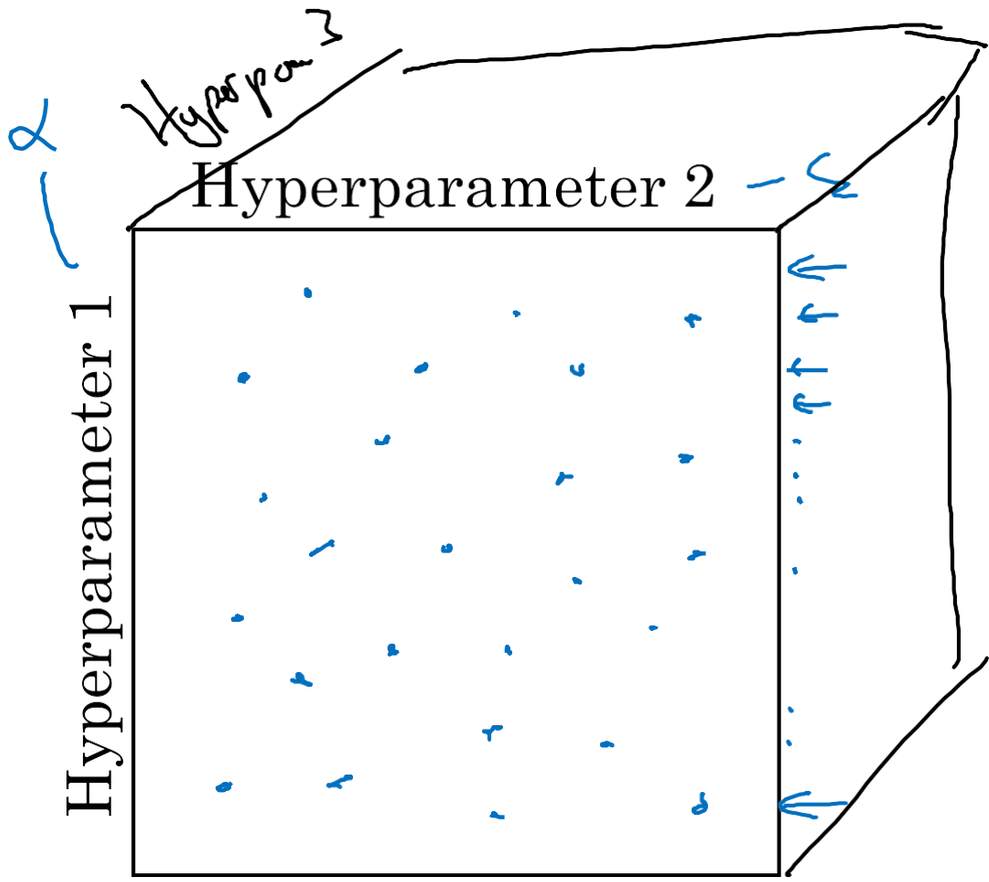
Tuning process

Hyperparameters

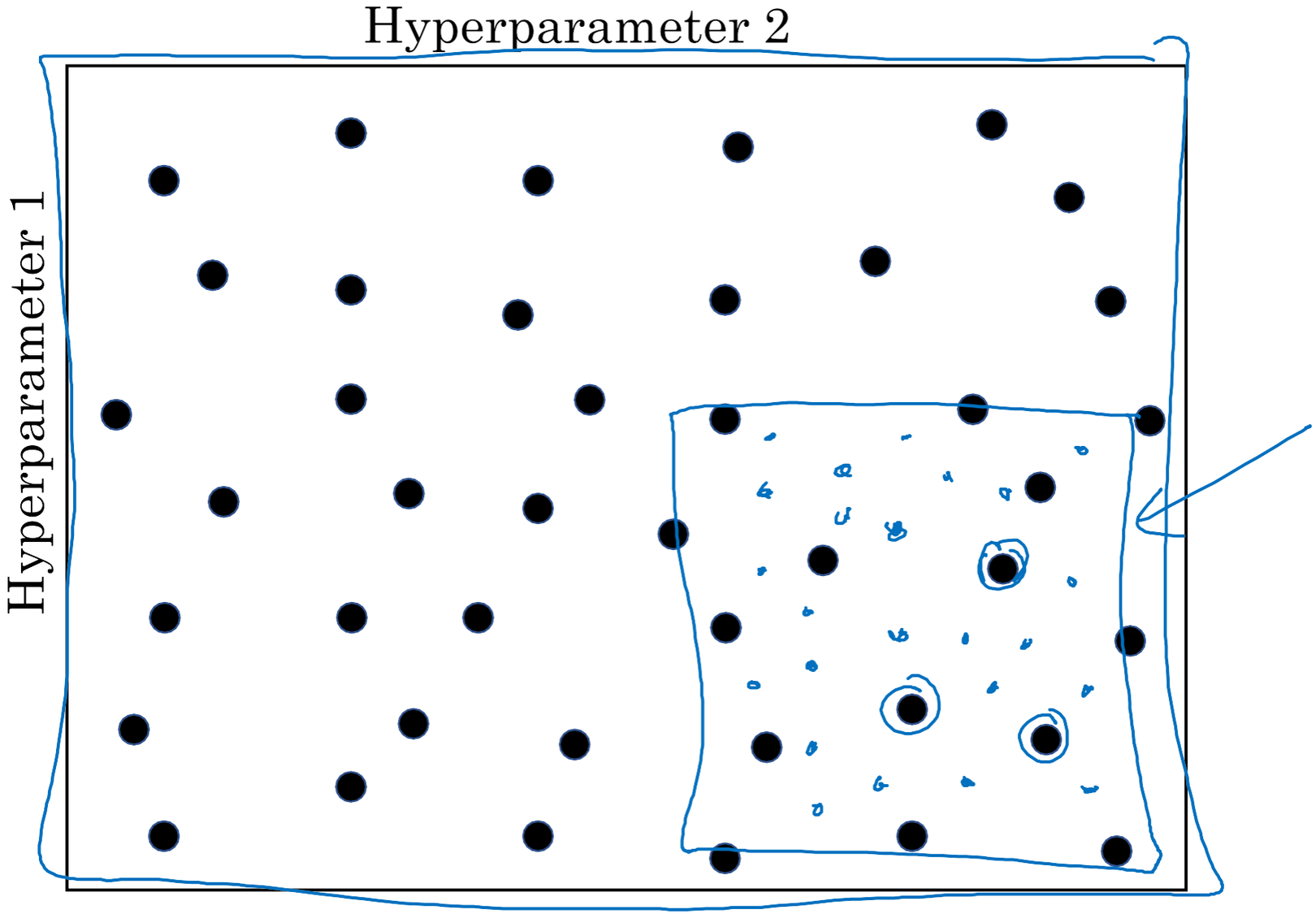


Try random values: Don't use a grid





Coarse to fine



deeplearning.ai

Multi-class classification

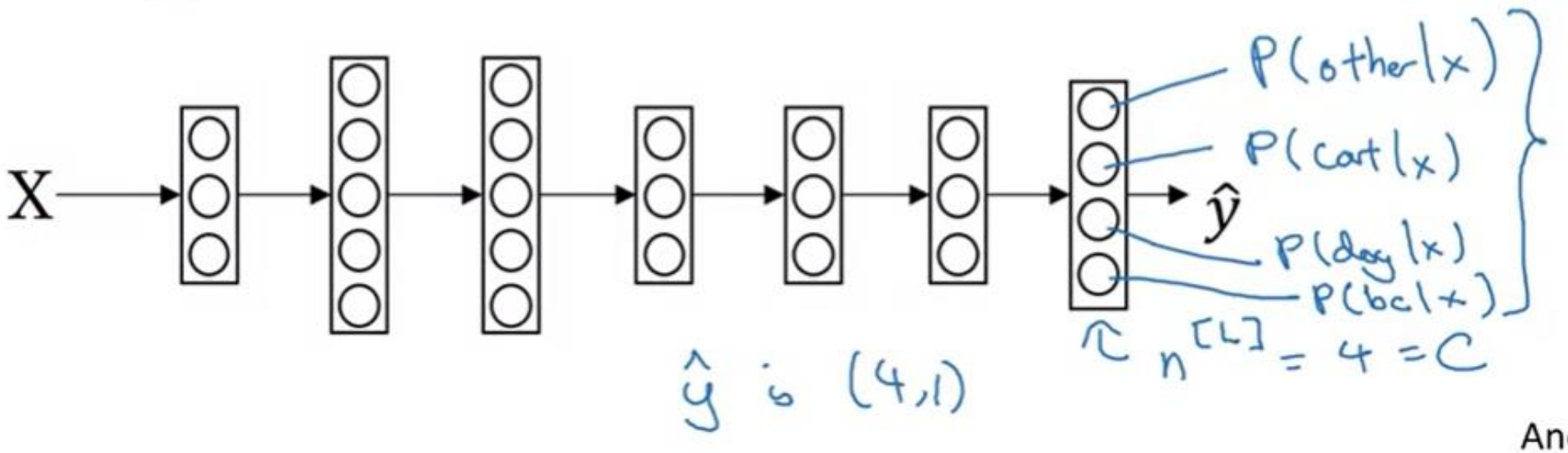
Softmax regression

Recognizing cats, dogs, and baby chicks, other

1 2 3 0

3 1 2 0 3 2 0 1

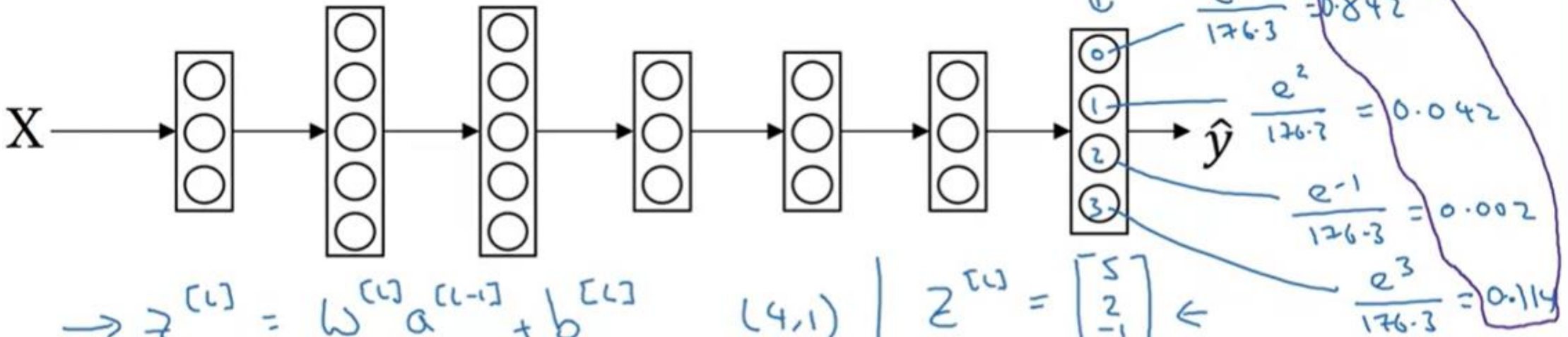
C = #classes = 4 $(0, \ldots, 3)$



$\hat{y}$ is (4,1)

Andrew Ng

Softmax layer



$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ (4,1)

Activation function:

$t = e^{(z^{[l]})}$ (4,1)

$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i}$, $a_i^{[l]} = \frac{t_i}{\sum_{j=1}^{4} t_i}$

(4,1) $\hat{y}$

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

(4,1) (4,1)

$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$ $\sum_{j=1}^{4} t_j = 176.3$

$a^{[l]} = \frac{t}{176.3}$

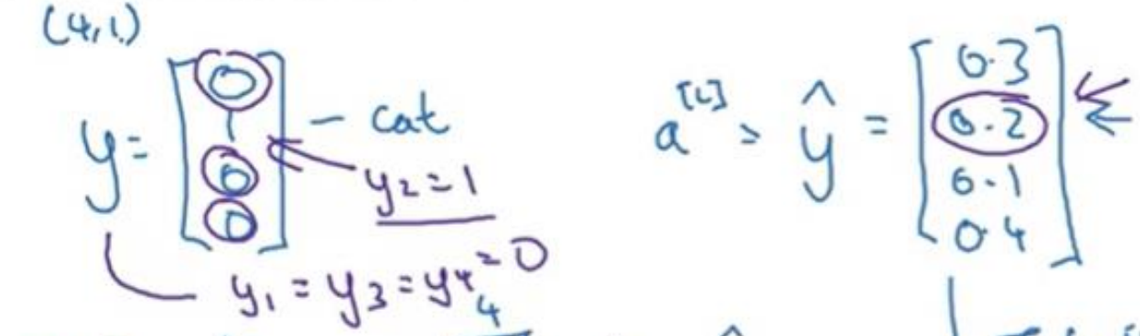
Andrew Ng

deeplearning.ai

Multi-class classification

Training a softmax classifier

Loss function



$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$$

small

$$J(w^{[1]}, b^{[1]}, \ldots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$-y_2 \log \hat{y}_2 = -\log \hat{y}_2.$$

Make $\hat{y}_2$ big.

Andrew Ng

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

This is the input matrix in 17 th slide of CMPE362_W13_CNN-part1.pdf

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

This is the input matrix in 33 rd slide of CMPE362_W13_CNN-part1.pdf