deeplearning.ai

Hyperparameter tuning

Tuning process

Hyperparameters

$\rightarrow \boxed{\alpha}$

$\boxed{\beta}$ ~ 0.9

$\beta_1, \beta_2, \varepsilon$

$0.9 \quad 0.999 \quad 10^{-8}$

#layers

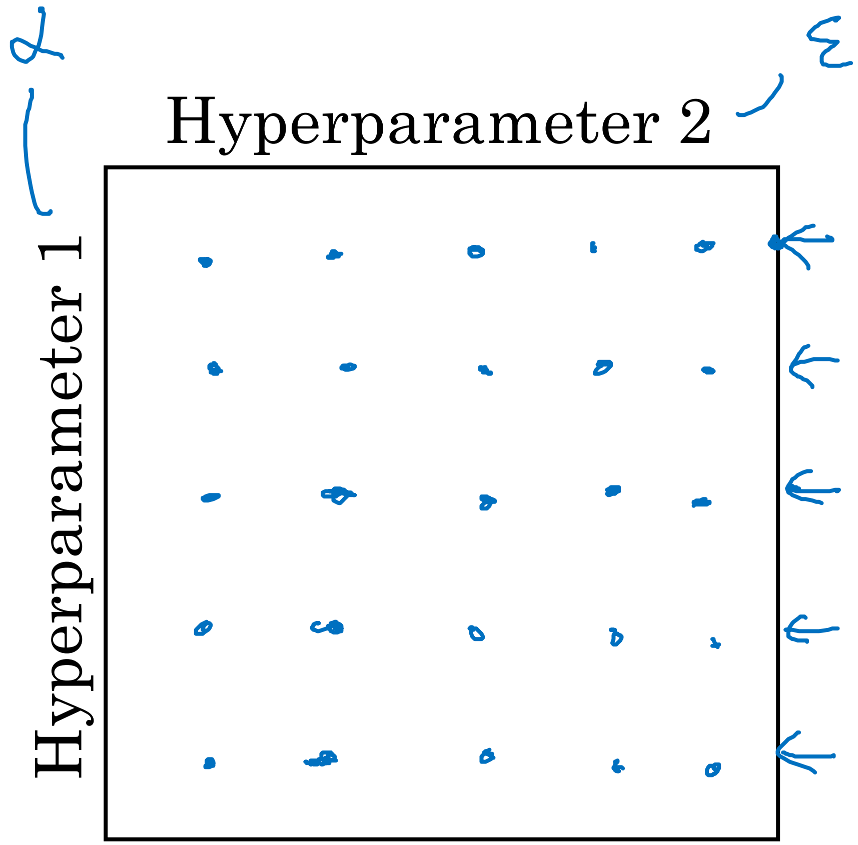
#hidden units

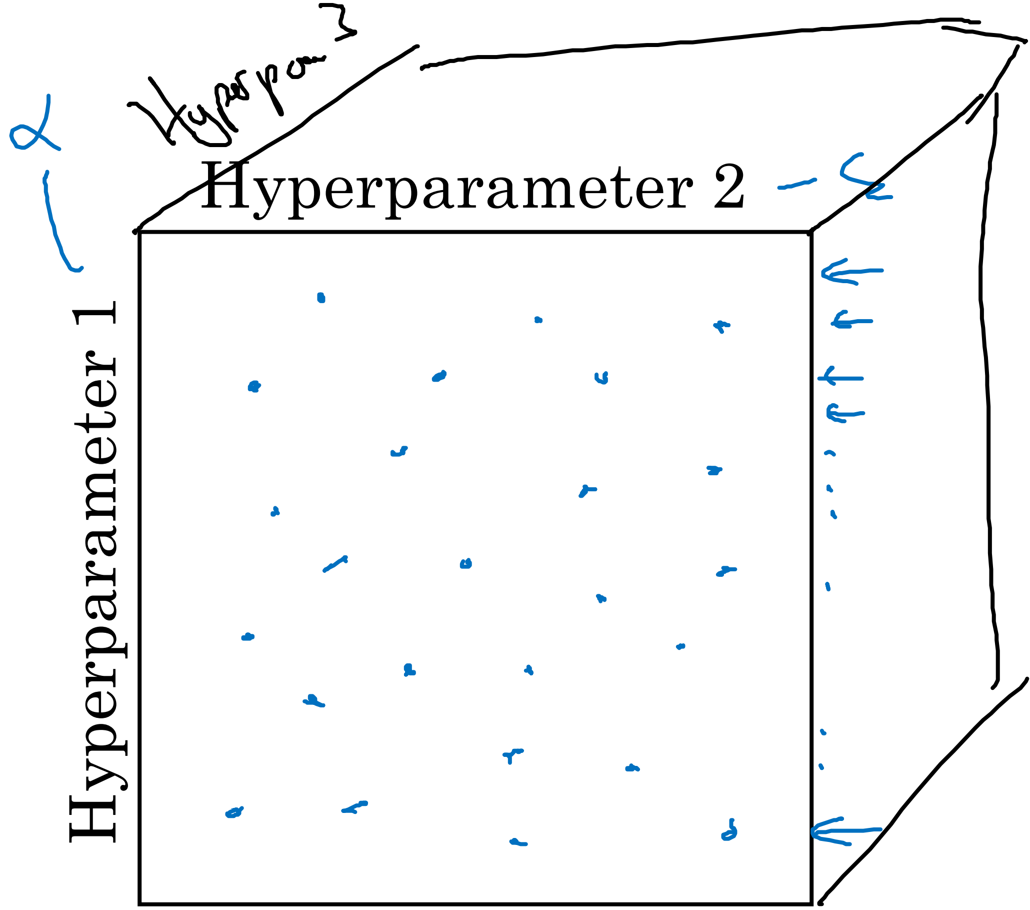
learning rate decay

mini-batch size

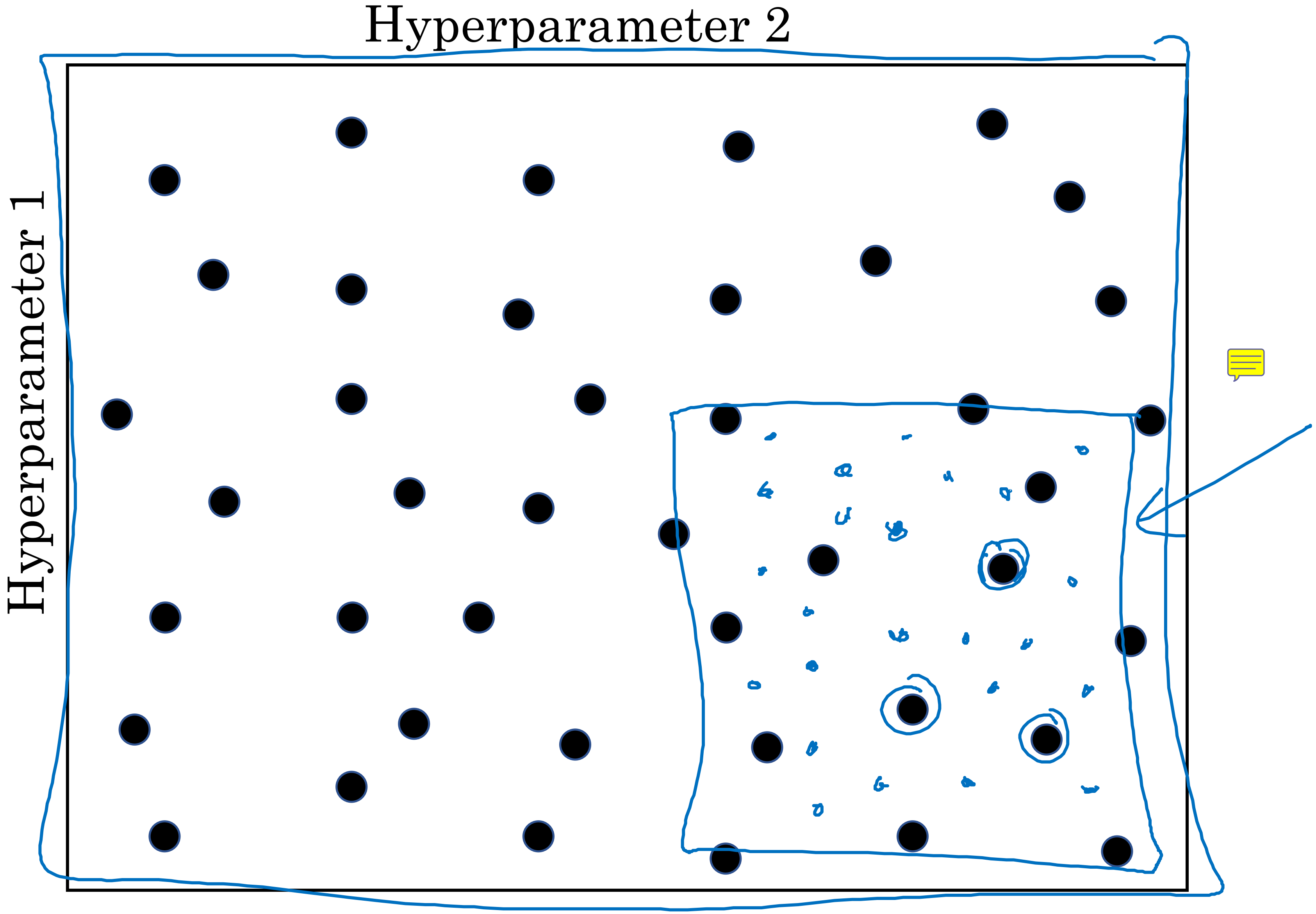
Andrew Ng

Try random values: Don't use a grid





Coarse to fine



Andrew Ng

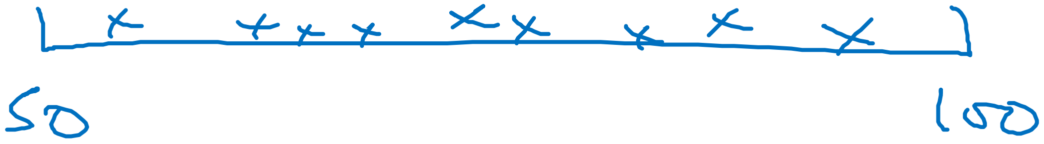
deeplearning.ai

Hyperparameter tuning

Using an appropriate scale to pick hyperparameters

Picking hyperparameters at random

$\rightarrow \quad n^{[l]} = 50, \ldots, 100$

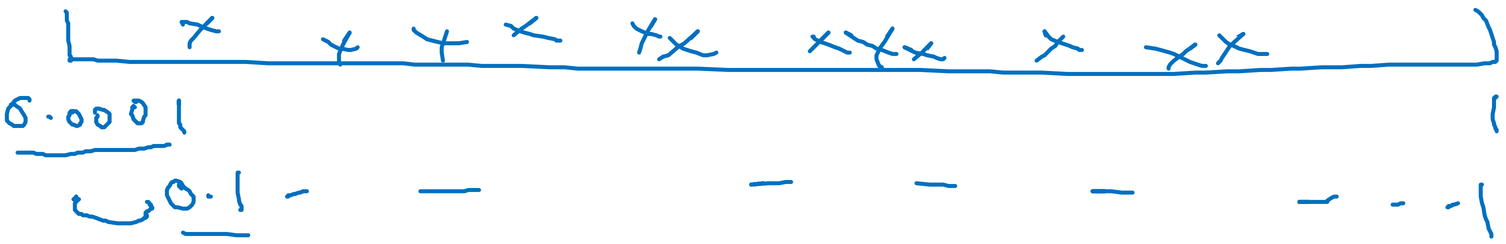


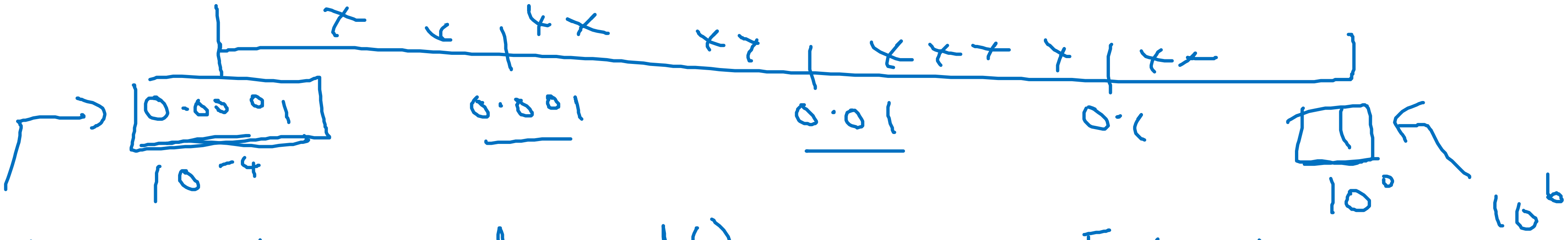
$\rightarrow$ #layers $\quad L: \; 2 - 4$

$2, \; 3, \; 4$

Appropriate scale for hyperparameters

$\alpha = 0.0001, \ldots, 1$





$a = \log_{10} 0.0001 = -4$

`r = -4 * np.random.rand()` $\leftarrow$ $r \in [-4, 0]$

$b = \log_{10} 1 = 0$

$\alpha = 10^{r}$ $\leftarrow$ $10^{-4} \ldots 10^{0}$

$10^{a} \ldots 10^{b}$ $\quad$ $r \in [a, b]$ $\quad$ $[-4, 0]$ $\quad$ $\alpha = 10^{r}$

Andrew Ng

Hyperparameters for exponentially weighted averages

$\beta = 0.9 \quad \ldots \quad 0.999$

$\downarrow \quad\quad \downarrow$

$10 \quad\quad 1000$

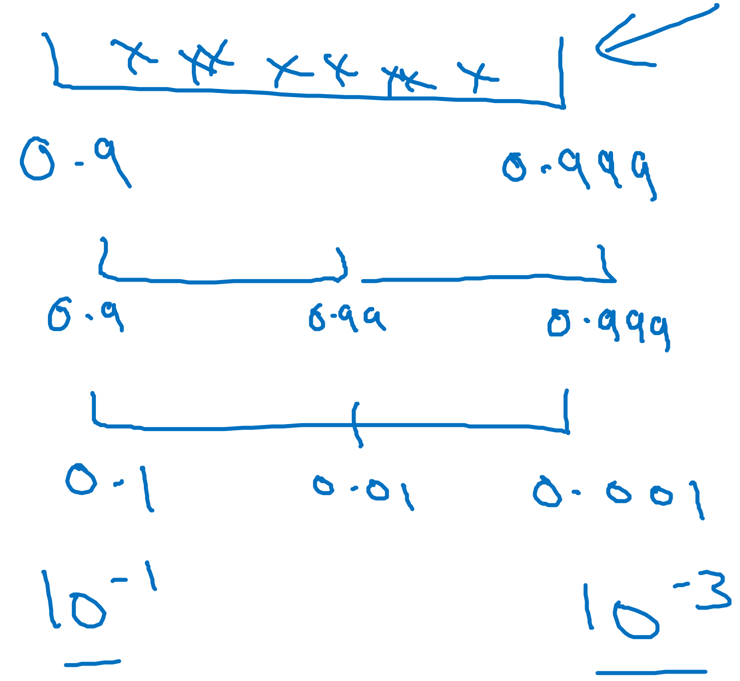
$1-\beta = 0.1 \quad \ldots \quad 0.001$

$\beta: 0.9000 \rightarrow 0.9005 \} \sim 10$

$\beta: 0.999 \rightarrow 0.9995$

$\sim 1000 \quad\quad \sim 2000$

$\frac{1}{1-\beta}$



$r \in [-3, -1]$

$1-\beta = 10^{r}$

$\beta = 1 - 10^{r}$

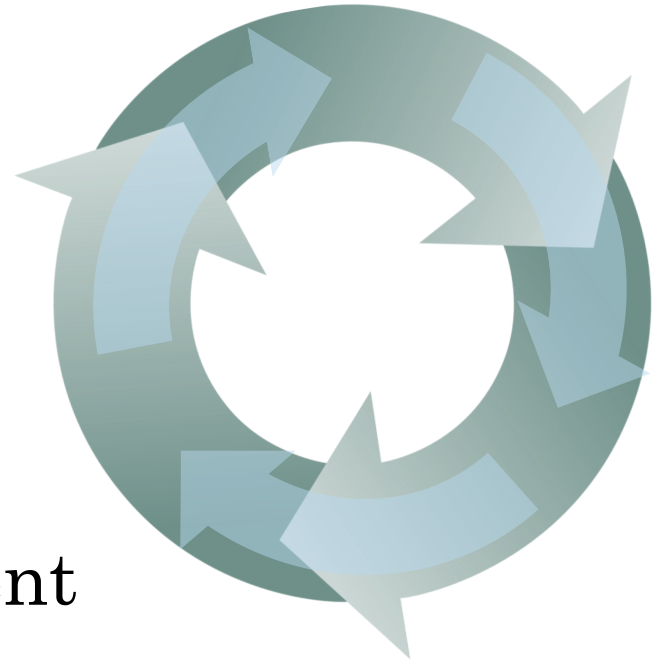
Andrew Ng

deeplearning.ai

Hyperparameters tuning

Hyperparameters tuning in practice: Pandas vs. Caviar

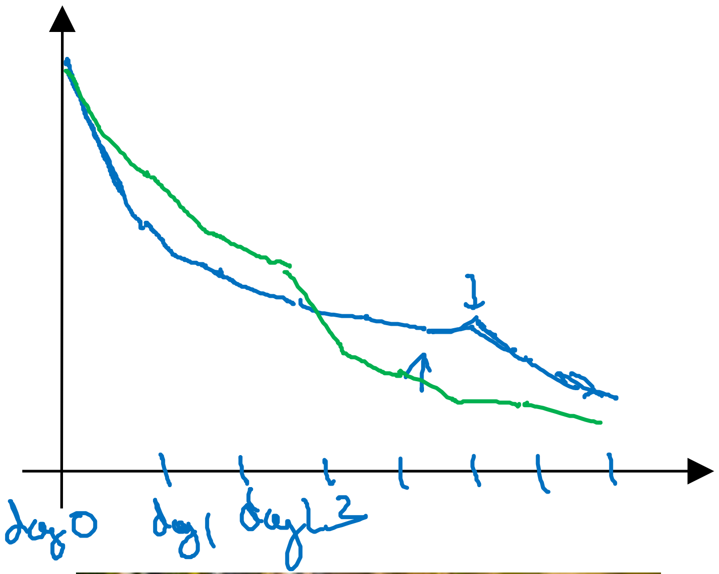
Re-test hyperparameters occasionally

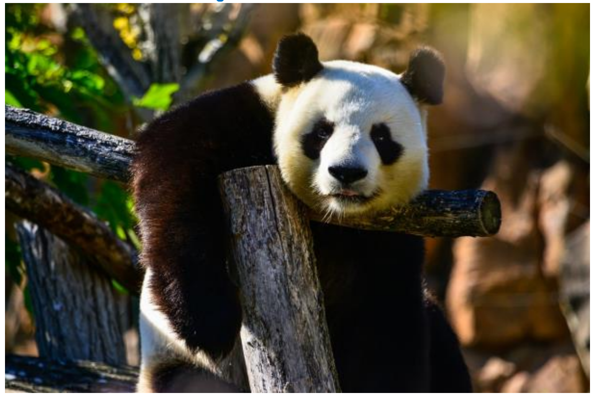


- NLP, Vision, Speech, Ads, logistics, ...

- Intuitions do get stale. Re-evaluate occasionally.

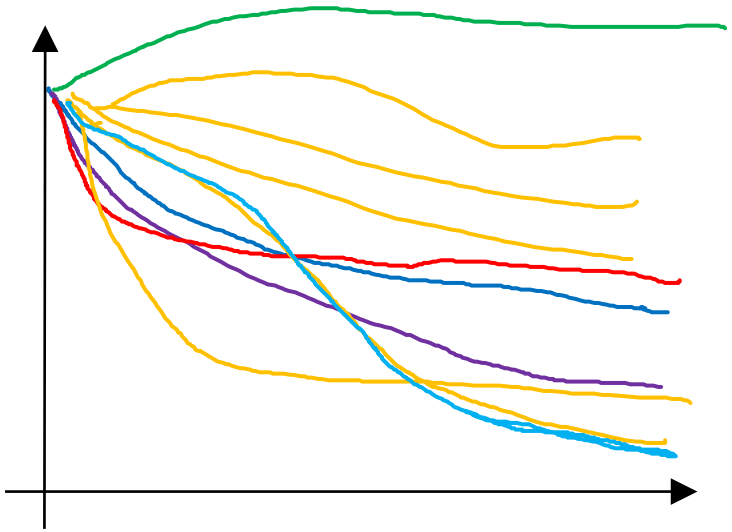
Babysitting one model





Panda

Training many models in parallel



Caviar

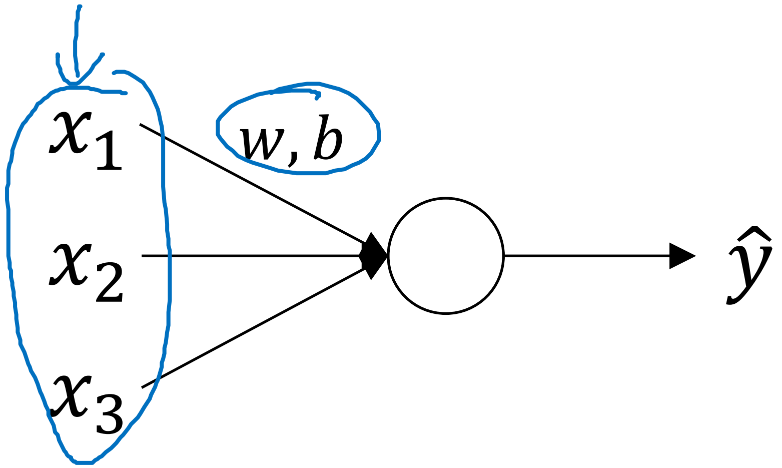
Andrew Ng

deeplearning.ai

Batch Normalization

Normalizing activations in a network

Normalizing inputs to speed up learning

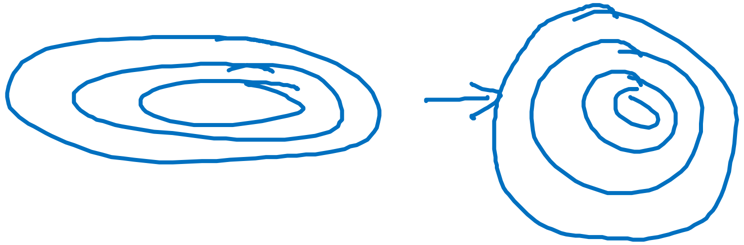


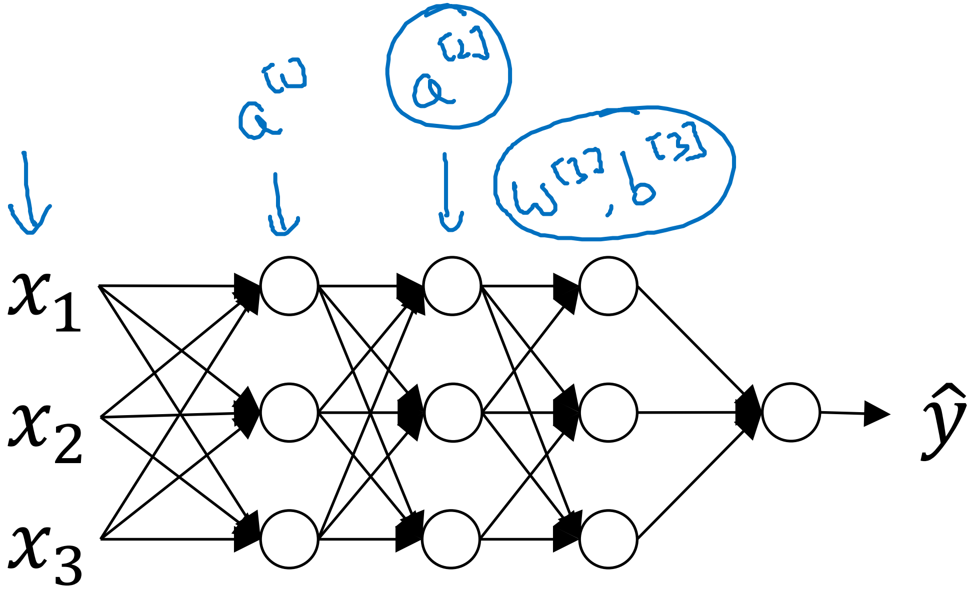
$\mu = \frac{1}{m} \sum_i x^{(i)}$

$X = X - \mu$ ← element-wise

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

$X = X / \sigma^2$





Can we normalize $\underline{a^{[2]}}$ so as to train $W^{[3]}, b^{[3]}$ faster

Normalize $\underline{z^{[2]}}$ ↑

Andrew Ng

Implementing Batch Norm

Given some intermediate values in NN $\underbrace{z^{(1)}, \ldots, z^{(m)}}_{z^{[l](i)}}$

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z_i-\mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)}-\boxed{\mu}}{\boxed{\sqrt{\sigma^2+\varepsilon}}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

learnable parameters of model.

If

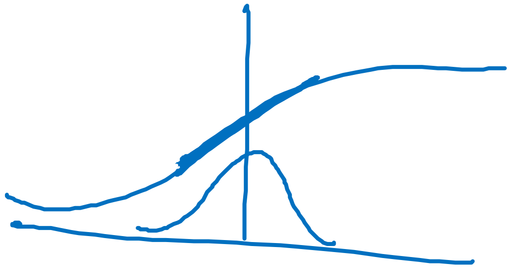
$$\gamma = \boxed{\sqrt{\sigma^2+\varepsilon}}$$

$$\beta = \boxed{\mu}$$

then $\tilde{z}^{(i)} = z^{(i)}$

X

$z^{(i)}$



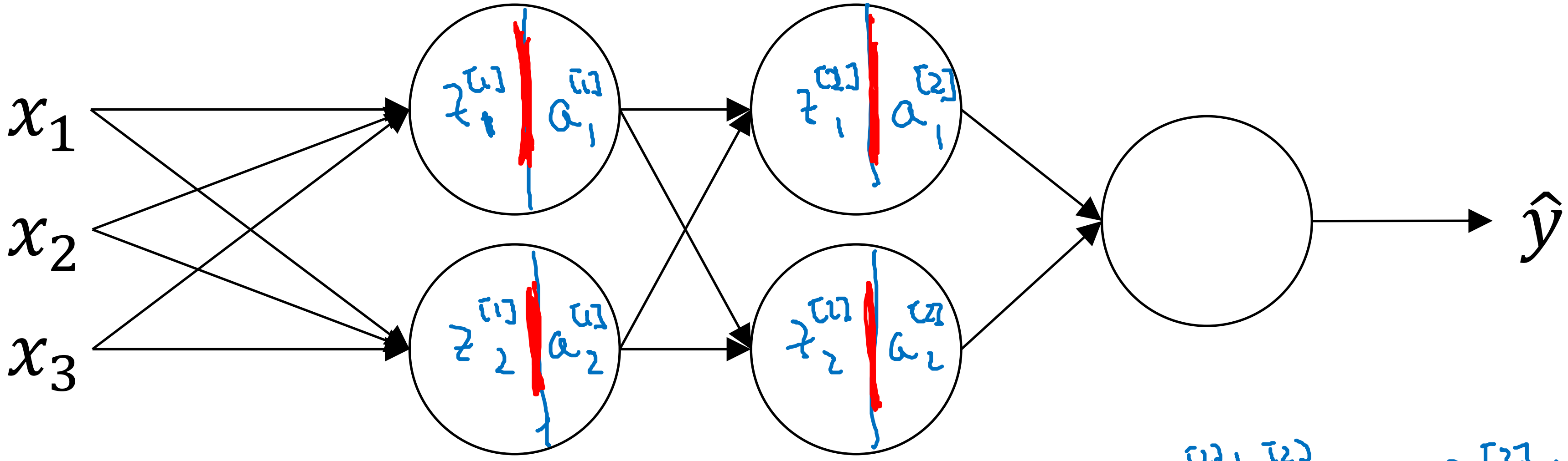
Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

Batch Normalization

deeplearning.ai

Fitting Batch Norm into a neural network

Adding Batch Norm to a network



$$X \xrightarrow{W^{[1]},\, b^{[1]}} z^{[1]} \xrightarrow[\boxed{\text{Batch Norm (BN)}}]{\beta^{[1]},\, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{W^{[2]},\, b^{[2]}} z^{[2]} \xrightarrow[\text{BN}]{\beta^{[2]},\, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \rightarrow \cdots$$

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots, W^{[L]}, b^{[L]},$
$\rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots, \beta^{[L]}, \gamma^{[L]}$

$\rightarrow \beta$

$d\beta^{[l]}$ $\qquad \beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$

tf.nn.batch_normalization $\Leftarrow$

Working with mini-batches

$X^{\{1\}} \xrightarrow{W^{[1]},\, b^{[1]}} z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]},\, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \xrightarrow{W^{[2]},\, b^{[2]}} z^{[2]} \rightarrow \cdots$

$\boxed{X^{\{2\}}} \longrightarrow z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]},\, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow \cdots$

$X^{\{3\}} \longrightarrow \cdots$

Parameters: $W^{[l]}$, $\cancel{b^{[l]}}$, $\beta^{[l]}$, $\gamma^{[l]}$.

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

$z^{[l]}$

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

$z^{[l]} = W^{[l]} a^{[l-1]} + \boxed{\cancel{b^{[l]}}}$

$z^{[l]} = W^{[l]} a^{[l-1]}$

$z^{[l]}_{norm}$

$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]}_{norm} + \boxed{\beta^{[l]}}$

Andrew Ng

Implementing gradient descent

for $t = 1 \ldots$ numMiniBatches

Compute forward prop on $X^{\{t\}}$.

In each hidden layer, use BN to replace $\underline{z^{[l]}}$ with $\underline{\tilde{z}^{[l]}}$.

Use backprop to compute $\underline{dW^{[l]}}$, $\cancel{db^{[l]}}$, $\underline{d\beta^{[l]}}$, $\underline{d\gamma^{[l]}}$

Update params

$$\left.\begin{aligned} W^{[l]} &:= W^{[l]} - \alpha\, dW^{[l]} \\ \beta^{[l]} &:= \beta^{[l]} - \alpha\, d\beta^{[l]} \\ \gamma^{[l]} &:= \ldots \end{aligned}\right\} \leftarrow$$

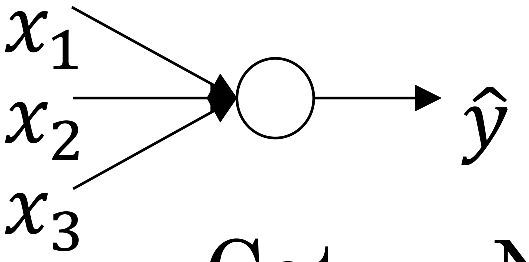
Works w/ momentum, RMSprop, Adam.

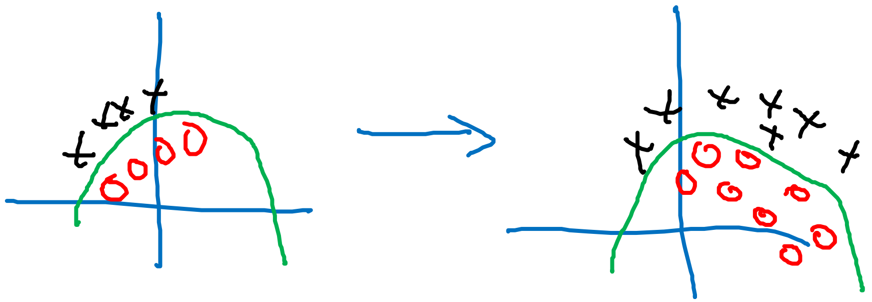
deeplearning.ai

Batch Normalization

Why does Batch Norm work?

Learning on shifting input distribution





Cat

$y = 1$

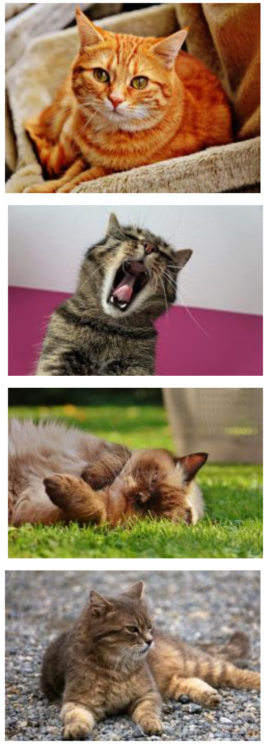
Non-Cat

$y = 0$

"Covariate shift"

$X \longrightarrow y$

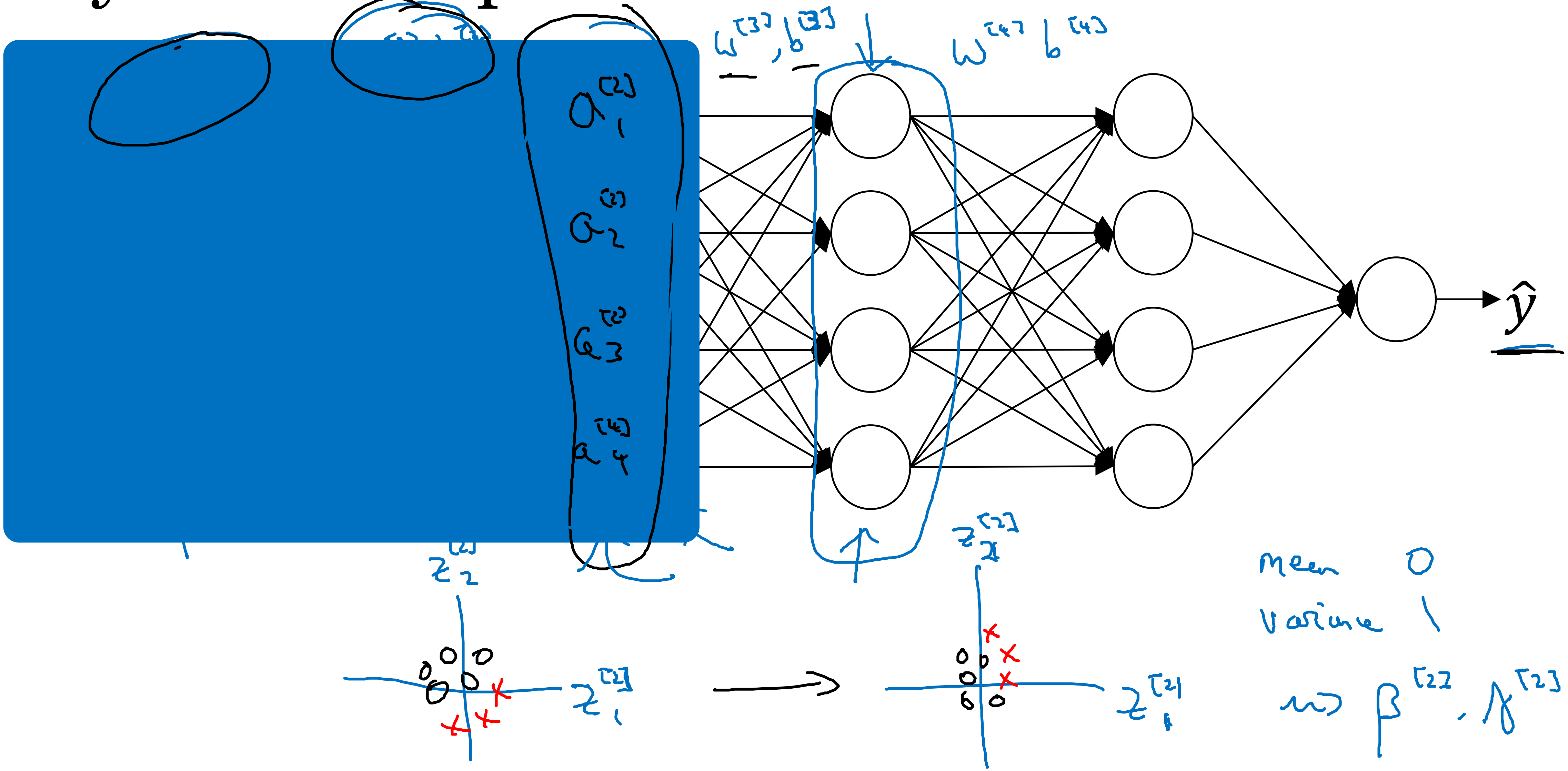
$y = 1$



$y = 0$

Andrew Ng

Why this is a problem with neural networks?



Andrew Ng

Batch Norm as regularization

X

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.

$X^{\{t\}}$

$\tilde{z}^{[l]}$ 64, 128 $z^{[l]}$

- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.

μ, σ^2

- This has a slight regularization effect.

mini-batch : 64 $\longrightarrow$ 512

Andrew Ng

Batch Normalization

Batch Norm at test time

deeplearning.ai

Batch Norm at test time

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

μ, σ^2: estimate using exponentially weighted average (across mini-batches).

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}, \ldots$

$\cancel{\mu^{[l]}}$

$\mu^{\{1\}[l]}$, $\mu^{\{2\}[l]}$, $\mu^{\{3\}[l]}$ $\longrightarrow \mu$

θ_1, θ_2, θ_3

$\sigma^{2\{1\}[l]}$, $\sigma^{2\{2\}[l]}$ .... σ^2

$$z_{\text{norm}} = \frac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}} \qquad \tilde{z} = \gamma z_{\text{norm}} + \beta$$

Andrew Ng

deeplearning.ai

Multi-class classification

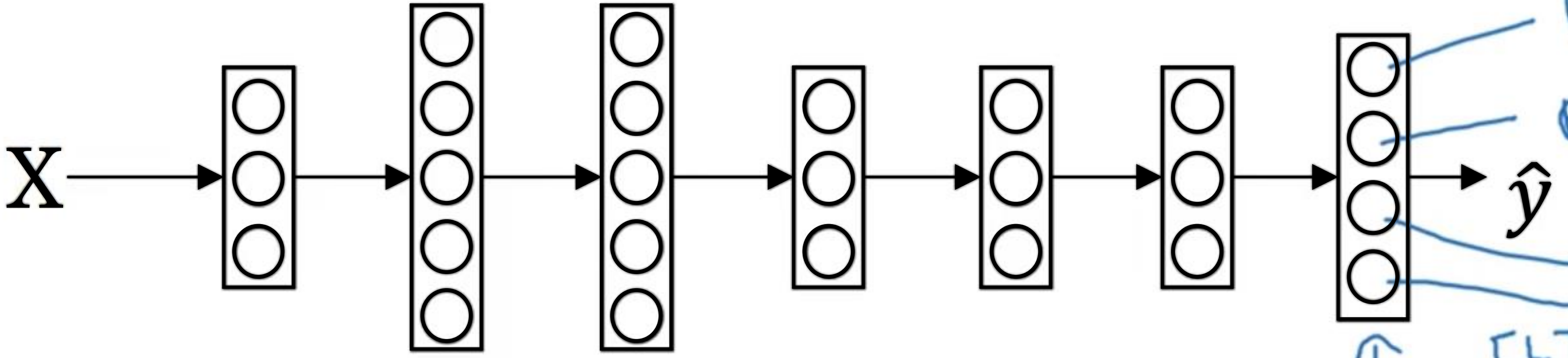
Softmax regression

Recognizing cats, dogs, and baby chicks, other

1 2 3 0

3 1 2 0 3 2 0 1

C = #classes = 4   (0, ..., 3)



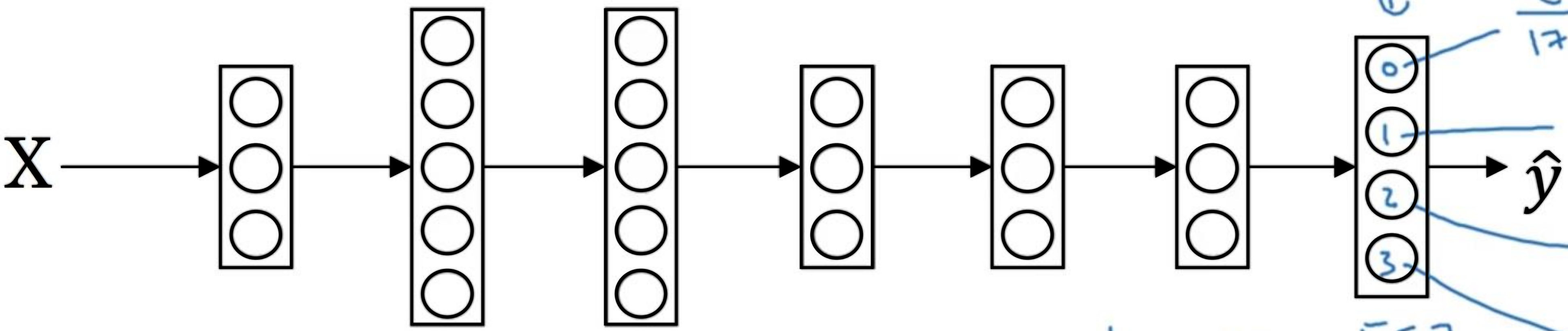
P(other|x)
P(cat|x)
P(dog|x)
P(bc|x)

$n^{[L]} = 4 = C$

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

Andrew Ng

Softmax layer



layer L

$\frac{e^5}{176.3} = 0.842$

$\frac{e^2}{176.3} = 0.042$

$\frac{e^{-1}}{176.3} = 0.002$

$\frac{e^3}{176.3} = 0.114$

$$\rightarrow z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]} \qquad (4,1)$$

Activation function:

$$t = e^{(z^{[L]})} \qquad (4,1)$$

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

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

$$\underset{(4,1)}{a^{[L]}} = g^{[L]}(\underset{(4,1)}{z^{[L]}})$$

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

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

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

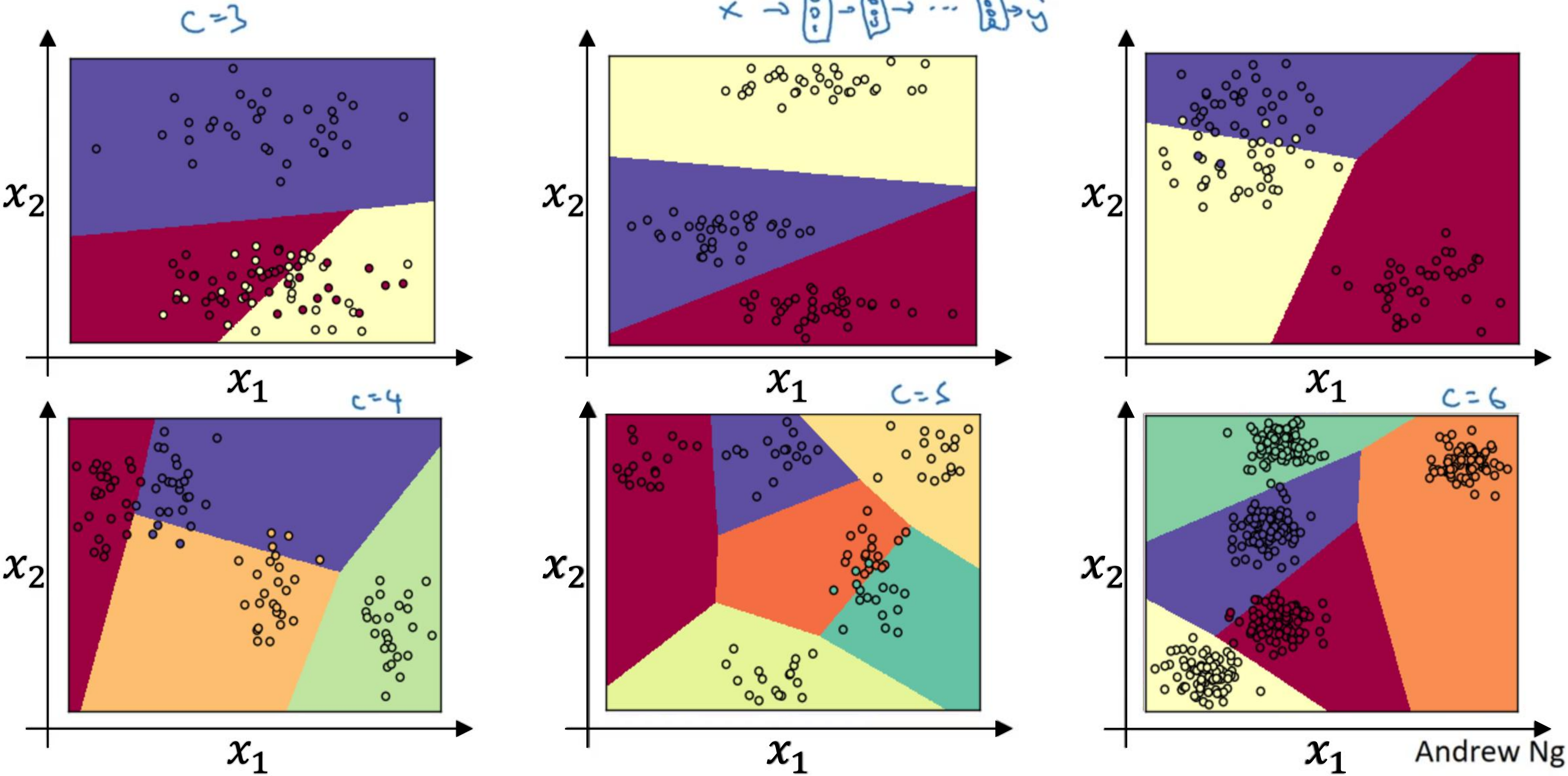
Softmax examples

$x_1, x_2 \rightarrow \boxed{\circ\circ\circ} \rightarrow \hat{y}$

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

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

$x \rightarrow \boxed{\circ\circ\circ} \rightarrow \boxed{\circ\circ\circ} \rightarrow \cdots \boxed{\circ\circ\circ} \rightarrow \hat{y}$



Andrew Ng

deeplearning.ai

Multi-class classification

Trying a softmax classifier

Understanding softmax

(4,1)

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

C = 4 $\quad g^{[L]}(\cdot)$

"soft max"

"hard max"

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Softmax regression generalizes logistic regression to C classes.

If C=2, softmax reduces to logistic regression. $a^{[L]} = \begin{bmatrix} 0.842 \\ \cancel{0.158} \end{bmatrix}$

Andrew Ng

Loss function

$(4,1)$

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ — cat}$$

$y_2 = 1$

$y_1 = y_3 = y_4 = 0$

$(4,1)$

$$a^{[L](i)} = \hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$C = 4$

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

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

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

Make $\hat{y}_2$ big.

$$Y = [y^{(1)}\ y^{(2)} \dots, y^{(m)}] = \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$$

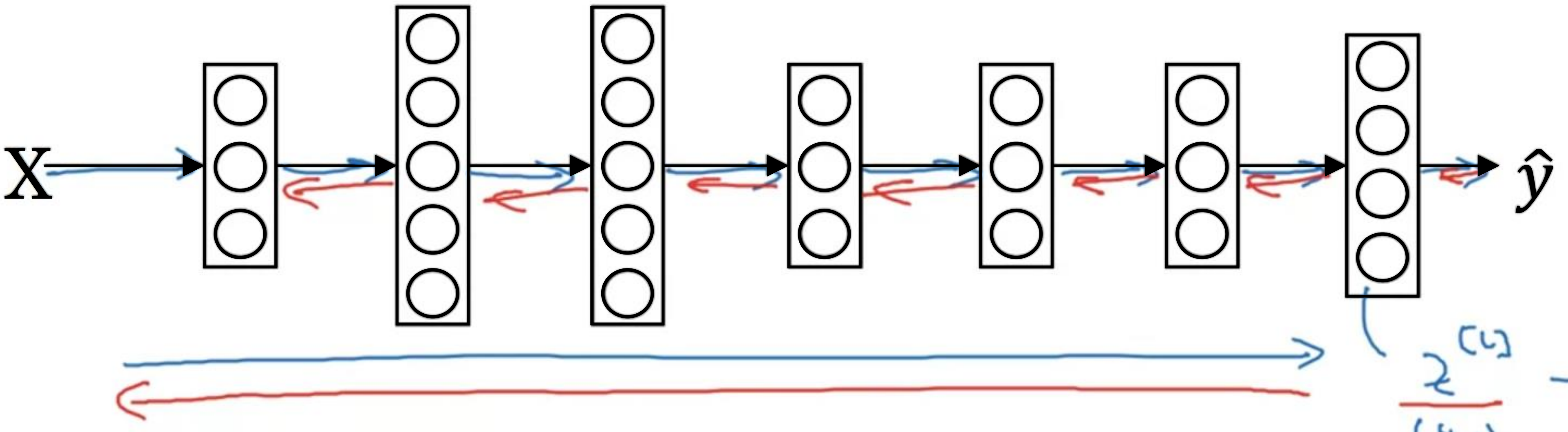
$(4, m)$

$$\hat{Y} = [\hat{y}^{(1)}, \dots, \hat{y}^{(m)}] = \begin{bmatrix} 0.3 & \dots \\ 0.2 & \\ 0.1 & \\ 0.4 & \end{bmatrix}$$

$(4, m)$

Andrew Ng

Gradient descent with softmax



$\underset{(4,1)}{z^{[L]}} \longrightarrow a^{[L]} = \hat{y} \to \mathcal{L}(\hat{y}, y)$

Backprop: $\underset{(4,1)}{dz^{[L]}} = \hat{y} - y$

$\frac{\partial J}{\partial z^{[L]}}$

Andrew Ng

deeplearning.ai

Programming Frameworks

Deep Learning frameworks

Deep learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

Choosing deep learning frameworks

- Ease of programming (development and deployment)
- Running speed
- Truly open (open source with good governance)

deeplearning.ai

Programming Frameworks

TensorFlow

Motivating problem

$J(w) = \boxed{w^2 - 10w + 25}$

(cost)

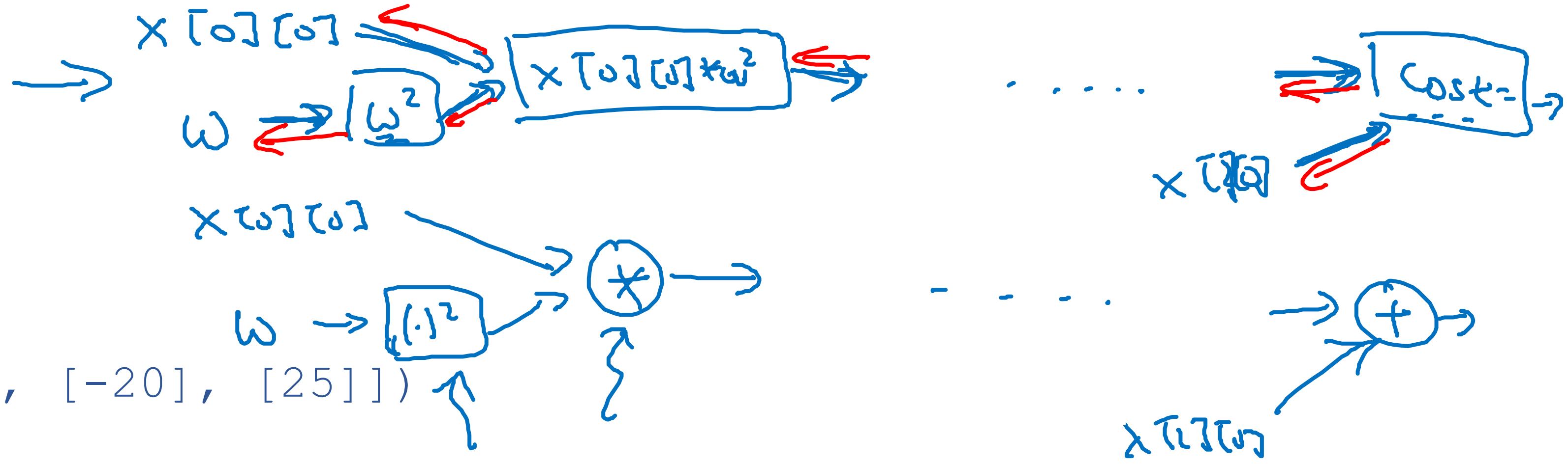
$(w-5)^2$

$w = 5$

$J(w, b)$

$\uparrow \quad \uparrow$

Code example




```
import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0],dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]     # (w-5)**2
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))
```

```
with tf.Session() as session:
    session.run(init)
    print(session.run(w))
```

```
for i in range(1000):
     session.run(train, feed_dict={x:coefficients})
print(session.run(w))
```

Andrew Ng