Copyright Notice

deeplearning.ai

Hyperparameter tuning

Tuning process

Hyperparameters

$\rightarrow \boxed{\alpha}$

$\boxed{\beta}$ ~0.9

$\underset{0.9}{\beta_1}, \underset{0.999}{\beta_2}, \underset{10^{-8}}{\varepsilon}$

$\boxed{\text{\# layers}}$

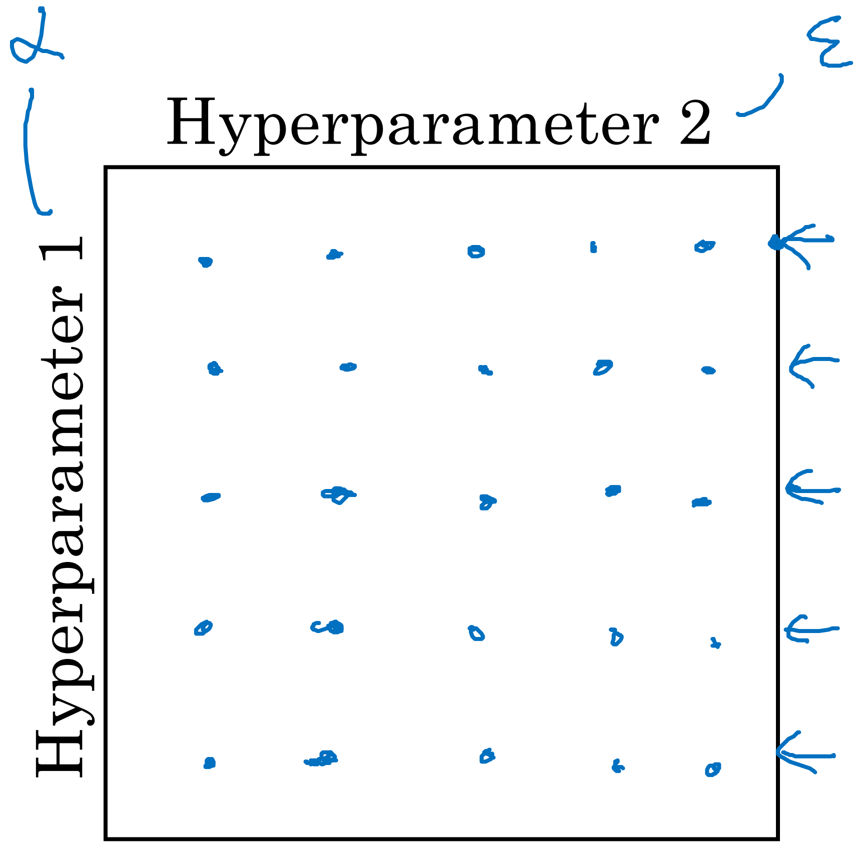
$\boxed{\text{\# hidden units}}$

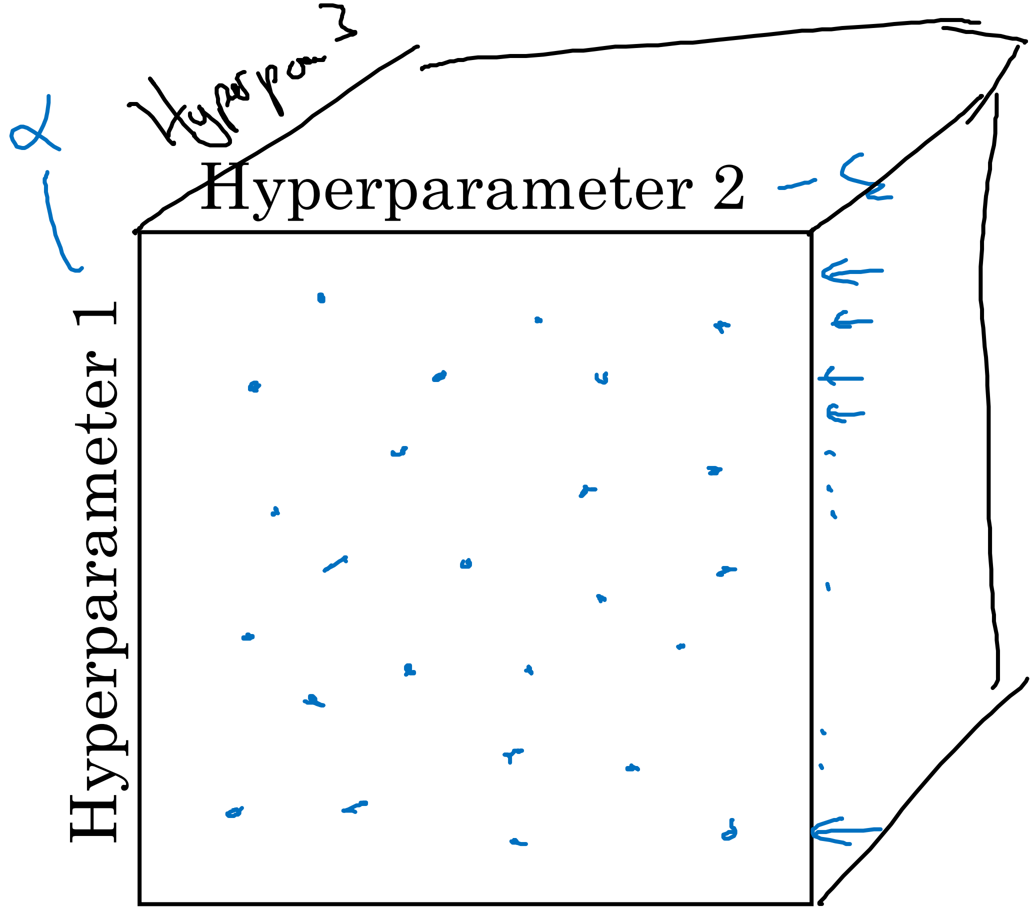
$\boxed{\text{learning rate decay}}$

$\boxed{\text{mini-batch size}}$

Andrew Ng

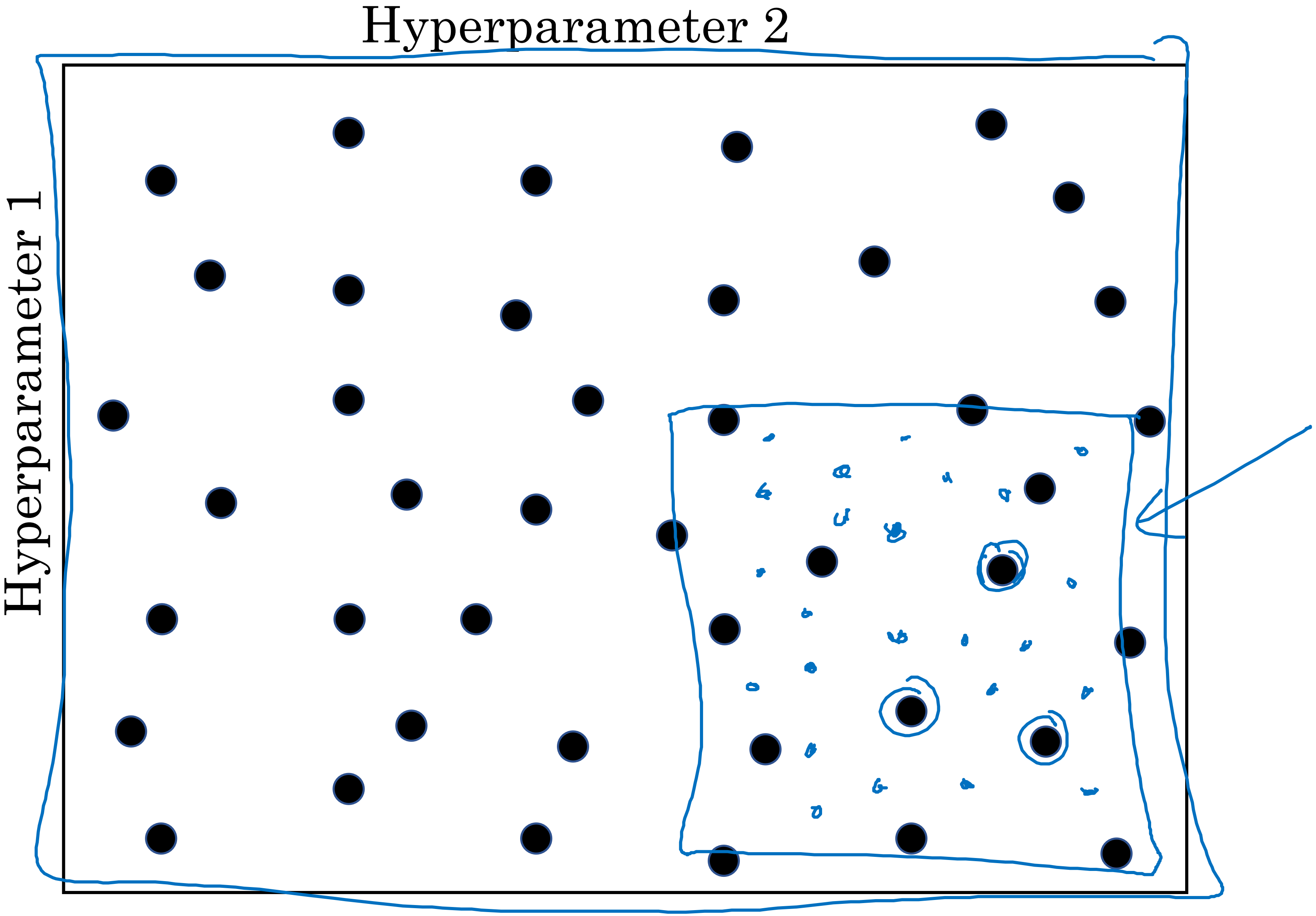
Try random values: Don't use a grid





Andrew Ng

Coarse to fine



Andrew Ng

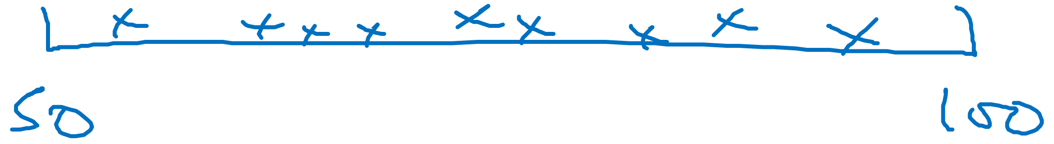
deeplearning.ai

Hyperparameter tuning

Using an appropriate scale to pick hyperparameters

Picking hyperparameters at random

→ $n^{[l]} = 50, \ldots, 100$

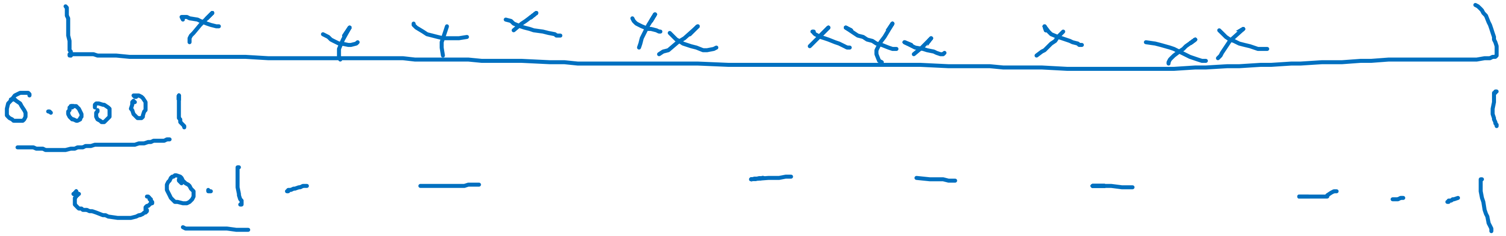


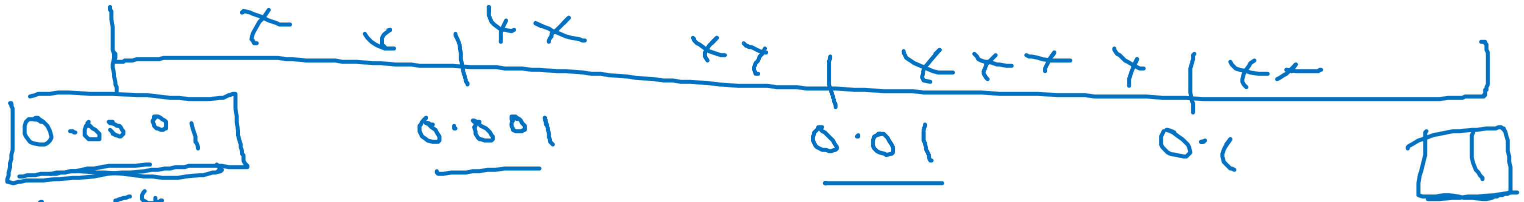
→ #layers $L: 2 - 4$

$2, 3, 4$

Appropriate scale for hyperparameters

$\alpha = 0.0001, \ldots, 1$





10^{-4} $\quad 10^a \quad 10^0 \quad 10^b$

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

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

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

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

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

Andrew Ng

Hyperparameters for exponentially weighted averages

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

$10 \qquad\qquad 1000$

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

$\left.\begin{array}{l}\beta: 0.9000 \to 0.9005\end{array}\right\} \sim 10$

$\beta: 0.999 \to 0.9995$

$\sim 1000 \qquad \sim 2000$

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

$0.9 \qquad\qquad 0.999$

$0.9 \qquad 0.99 \qquad 0.999$

$0.1 \qquad 0.01 \qquad 0.001$

$10^{-1} \qquad\qquad 10^{-3}$

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

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

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

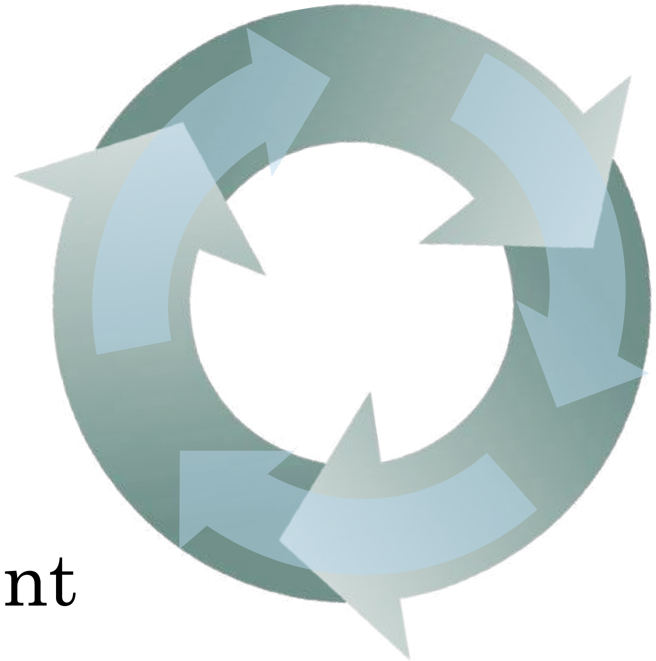
Andrew Ng

deeplearning.ai

Hyperparameters tuning

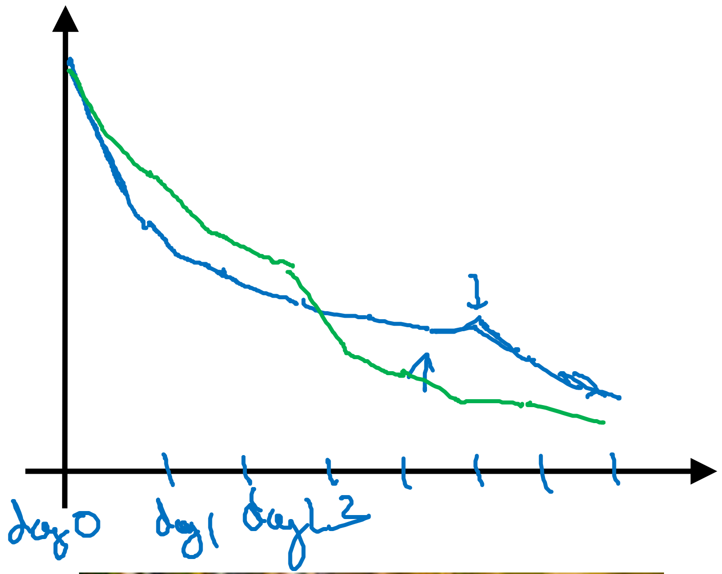
Hyperparameters tuning in practice: Pandas vs. Caviar

Re-test hyperparameters occasionally



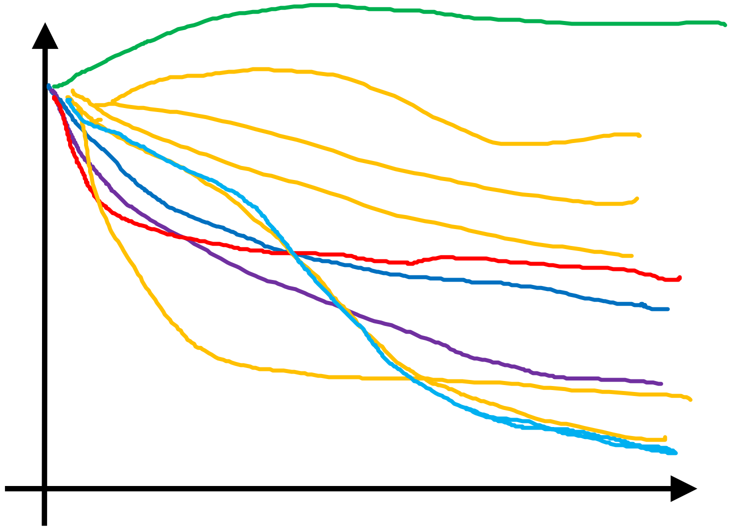
- NLP, Vision, Speech, Ads, logistics, ...
- Intuitions do get stale. Re-evaluate occasionally.

Babysitting one model



Panda

Training many models in parallel



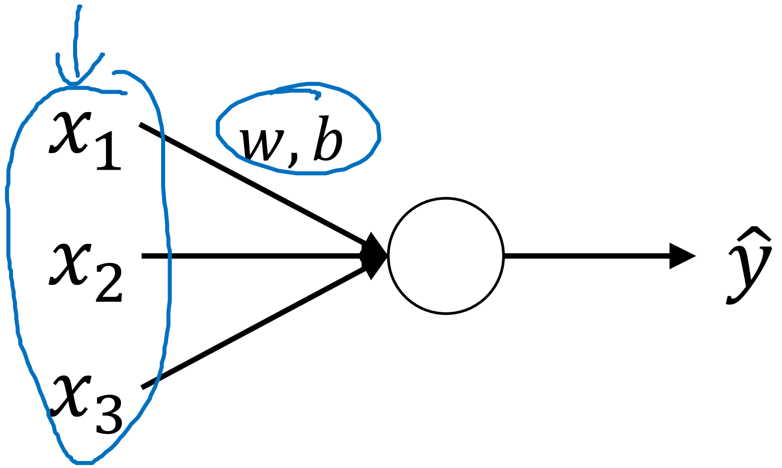
Caviar

deeplearning.ai

Batch Normalization

Normalizing activations in a network

Normalizing inputs to speed up learning

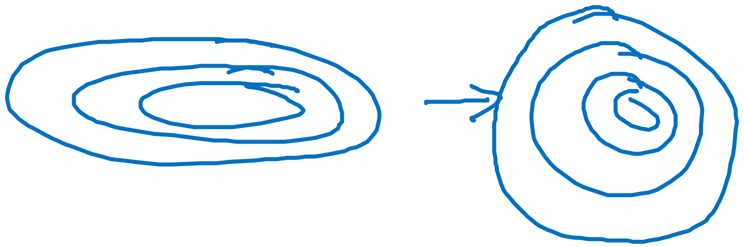


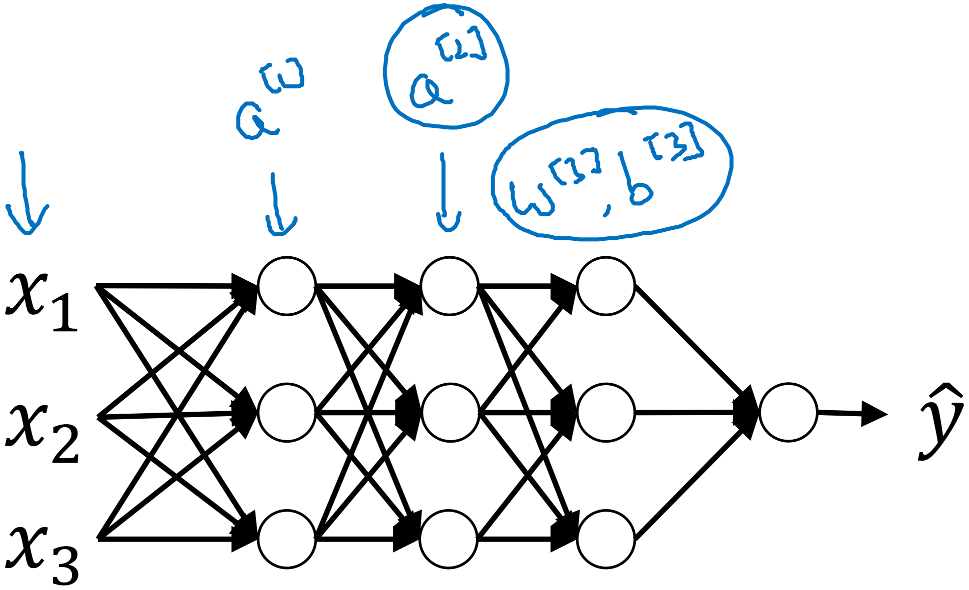
$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu \qquad \leftarrow \text{element-wise}$$

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

$$X = X / \sigma^2$$





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

Normalize $z^{[2]}$

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

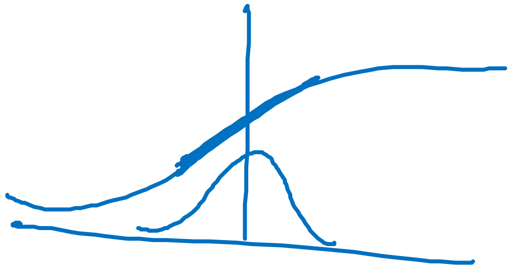
$$\gamma = \boxed{\sqrt{\sigma^2 + \varepsilon}}$$

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

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

X

$z^{(i)}$



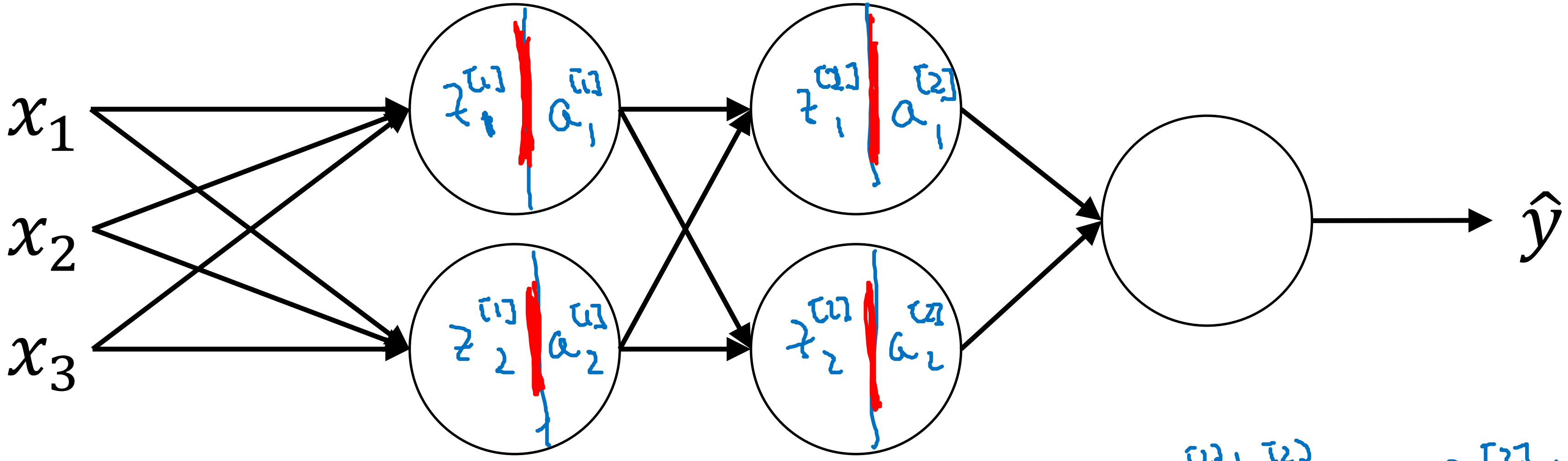
Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

deeplearning.ai

Batch Normalization

Fitting Batch Norm into a neural network

Adding Batch Norm to a network



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

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

$\rightarrow \beta$

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

tf.nn.batch_normalization $\Leftarrow$

Andrew Ng

Working with mini-batches

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

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

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

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

$b^{[l]}$: $(n^{[l]}, 1)$ $\quad$ $\beta^{[l]}$: $(n^{[l]}, 1)$ $\quad$ $\gamma^{[l]}$: $(n^{[l]}, 1)$

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

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

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

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

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

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

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

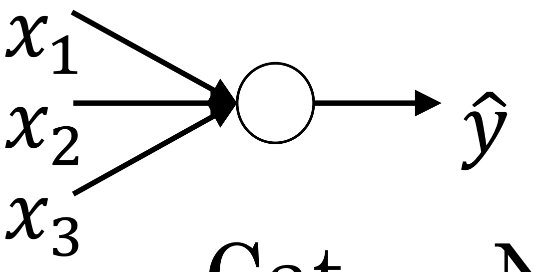
Works w/ momentum, RMSprop, Adam.

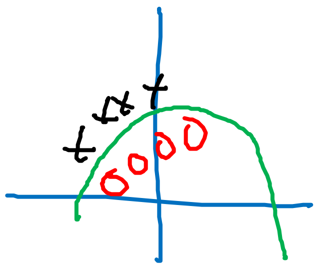
deeplearning.ai

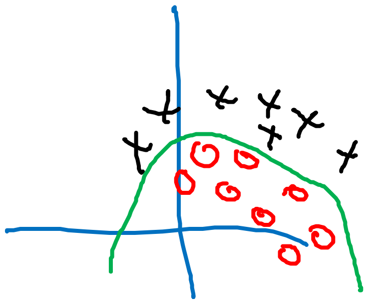
Batch Normalization

Why does Batch Norm work?

Learning on shifting input distribution

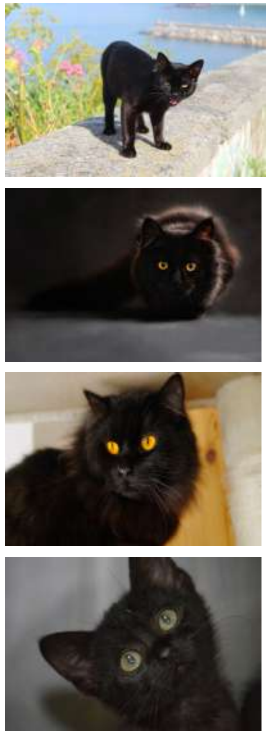






Cat

$y = 1$



Non-Cat

$y = 0$

"Covariate shift"

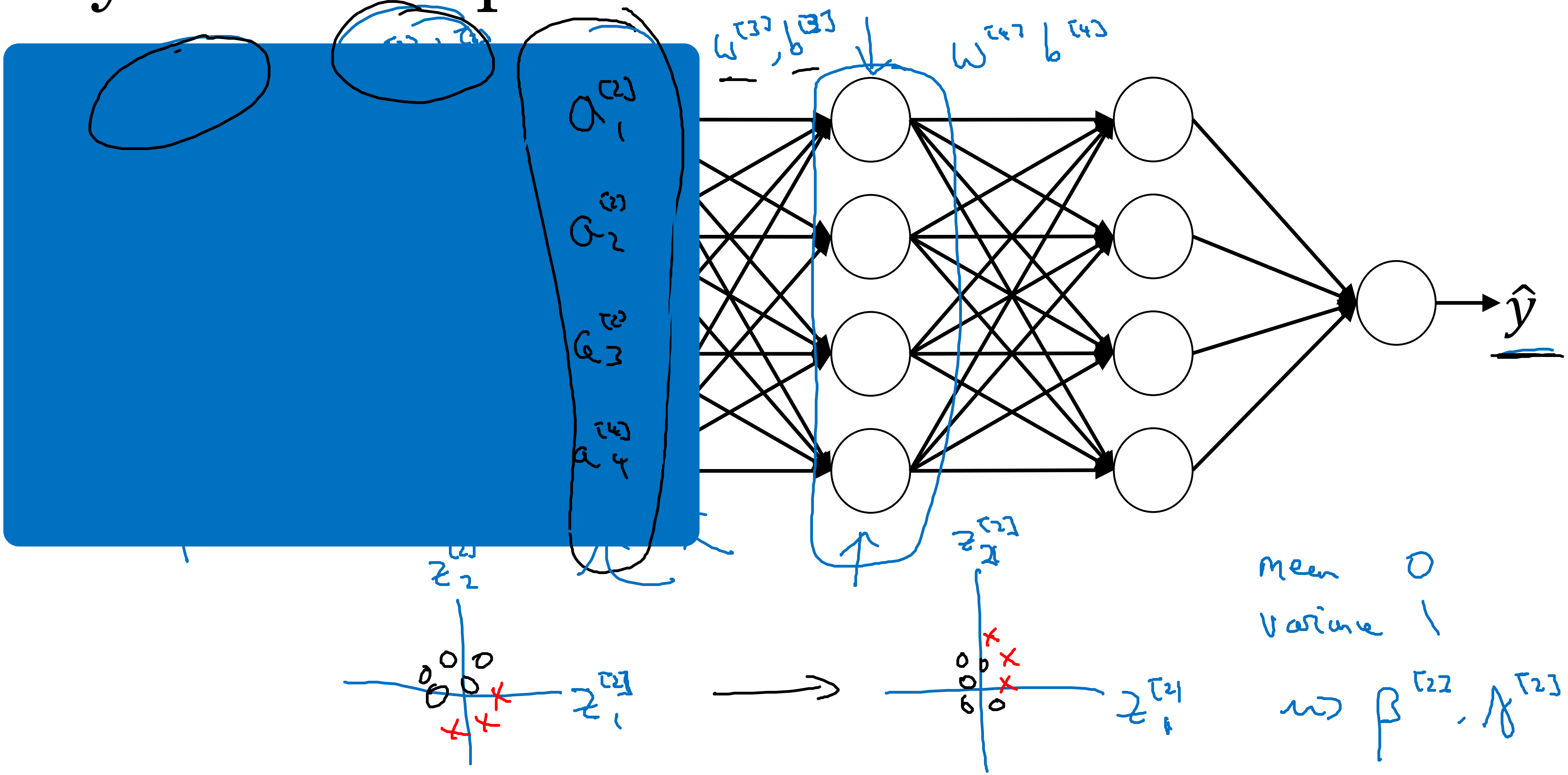
$X \rightarrow y$

$y = 1$

$y = 0$

Andrew Ng

Why this is a problem with neural networks?



Batch Norm as regularization

X

$X^{\{t\}}$

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

$\tilde{z}^{[l]}$ 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 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).

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

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

$\theta_1 \quad \theta_2 \quad \theta_3 \qquad \sigma^2$

$\sigma^{2\{1\}[l]} \quad \sigma^{2\{2\}[l]} \quad \ldots$

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

Andrew Ng

Batch Norm at test time

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

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

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

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

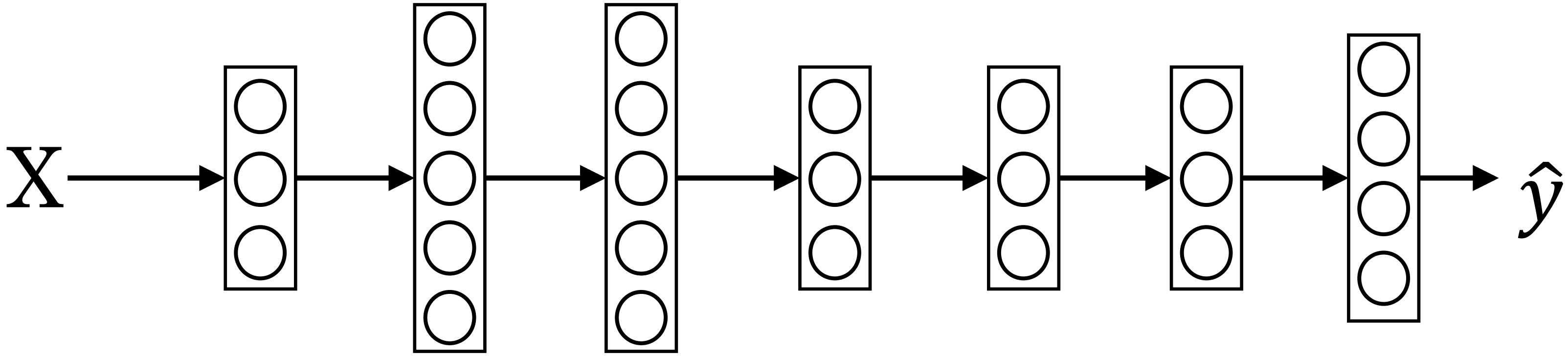
deeplearning.ai

Multi-class classification

Softmax regression

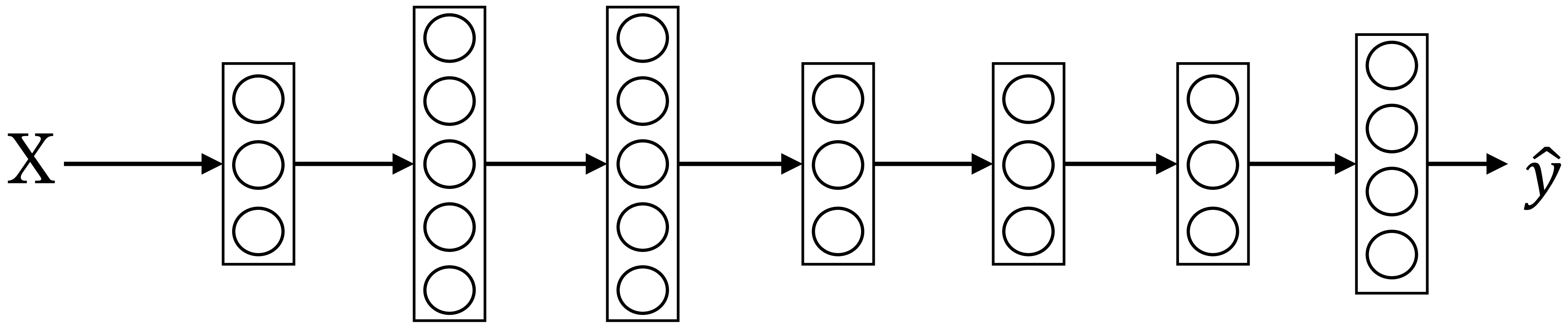
Recognizing cats, dogs, and baby chicks

3 1 2 0 3 2 0 1



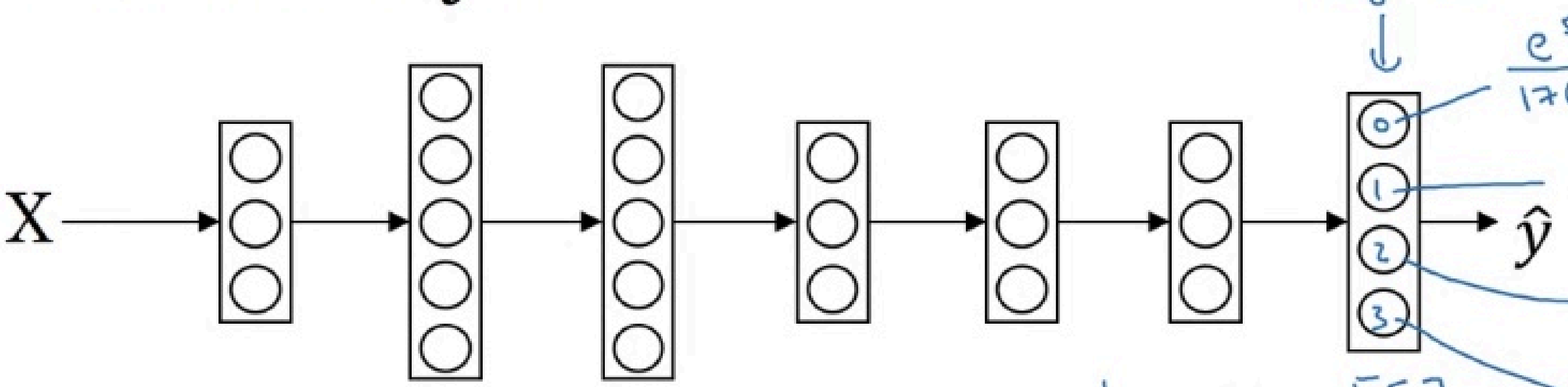
Andrew Ng

Softmax layer



Softmax layer

layer L



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

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

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

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

$$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]} \quad (4,1)$$

Activation function:

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

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

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

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

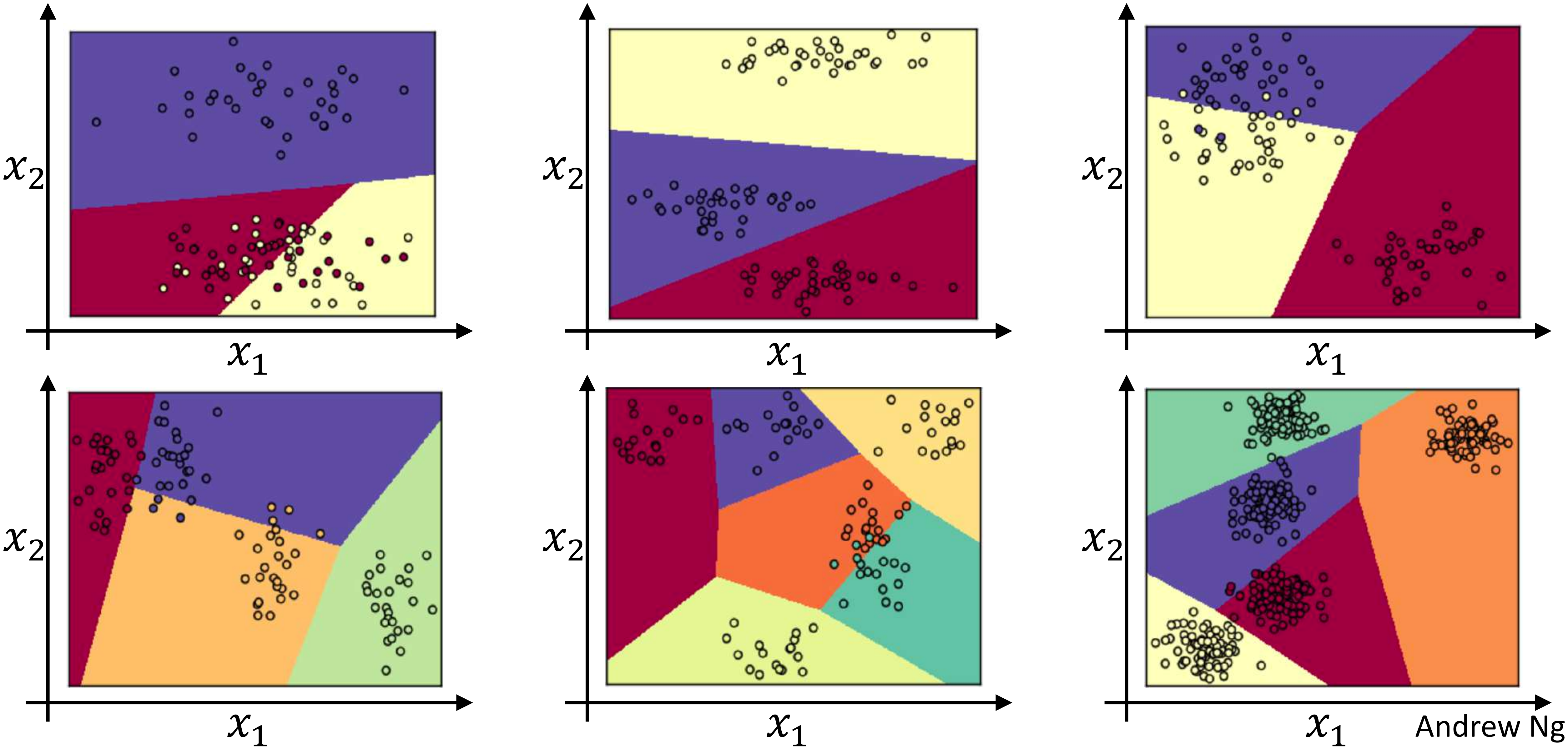
$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

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

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

Andrew Ng

Softmax examples



Andrew Ng

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

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad (4,1)$$

– cat

$\underline{y_2 = 1}$

$y_1 = y_3 = y_4 = 0$

$$a^{[L](i)} = \hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \quad (4,1)$$

$C = 4$

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

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

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

Make $\hat{y}_2$ big.

$$Y = [y^{(1)}\ y^{(2)} \dots, y^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$$

$(4, m)$

$$\hat{Y} = [\hat{y}^{(1)}, \dots, \hat{y}^{(m)}]$$

$$= \begin{bmatrix} 0.3 & \dots \\ 0.2 & \\ 0.1 & \\ 0.4 & \end{bmatrix}$$

$(4, m)$

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

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

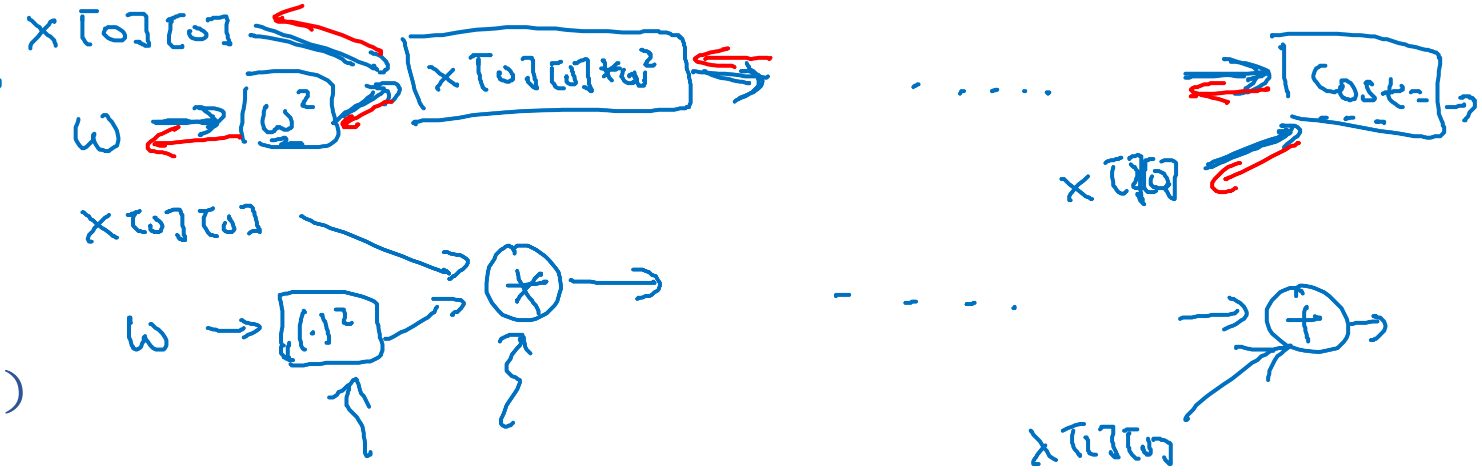
(cost)

$(w-5)^2$

$w = 5$

$J(w, b)$

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))

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

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

Andrew Ng