Branch: **master** ▾

Find file Copy path

deep-learning-specialization-coursera / 02-Improving-Deep-Neural-Networks / week3 /
hyperparameter-tuning-and-programming-frameworks.ipynb

andersy005 add datasets

64a8031 on Oct 15, 2018

1 contributor

Raw Blame History

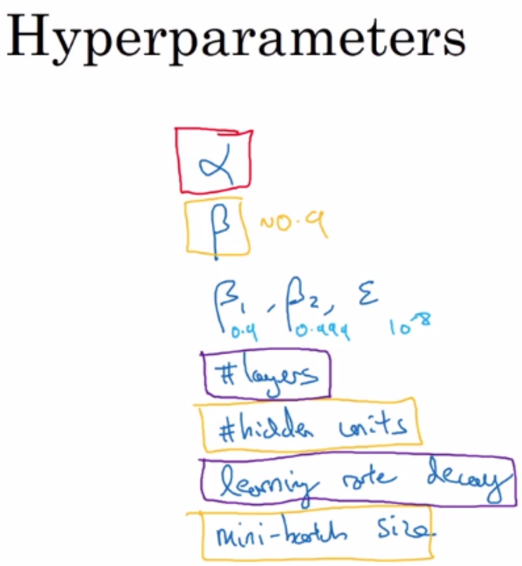
455 lines (455 sloc) 12.7 KB

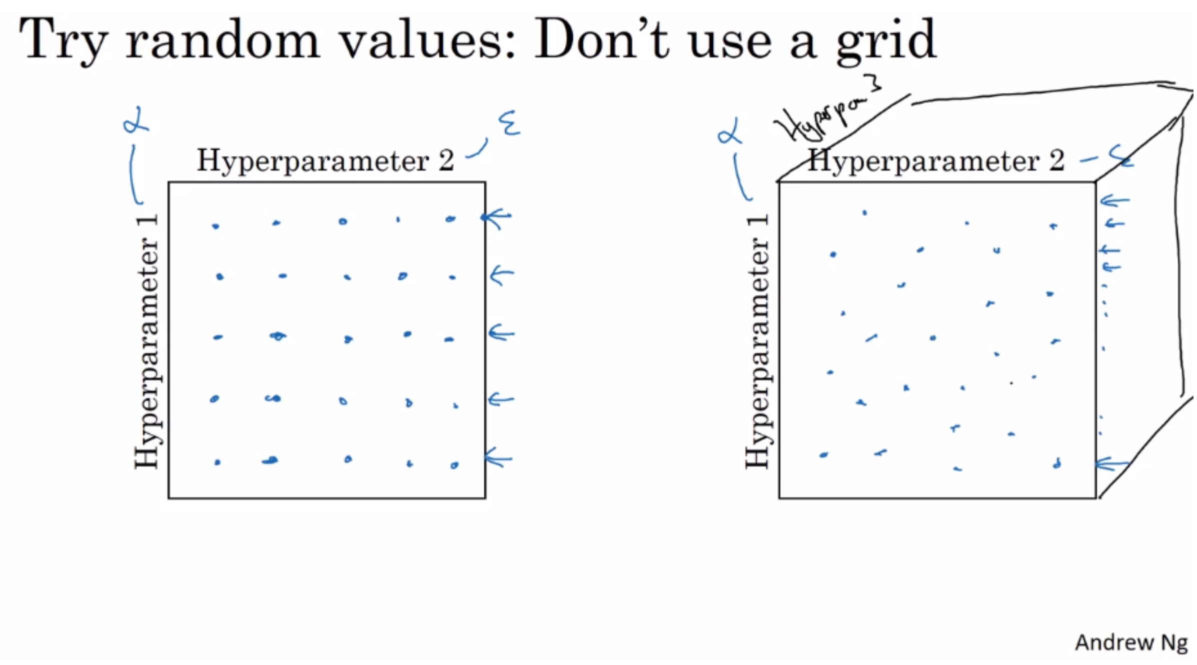
Table of Contents

Hyperparameter tuning, Batch Normalization and Programming Frameworks

Hyperparameter Tuning

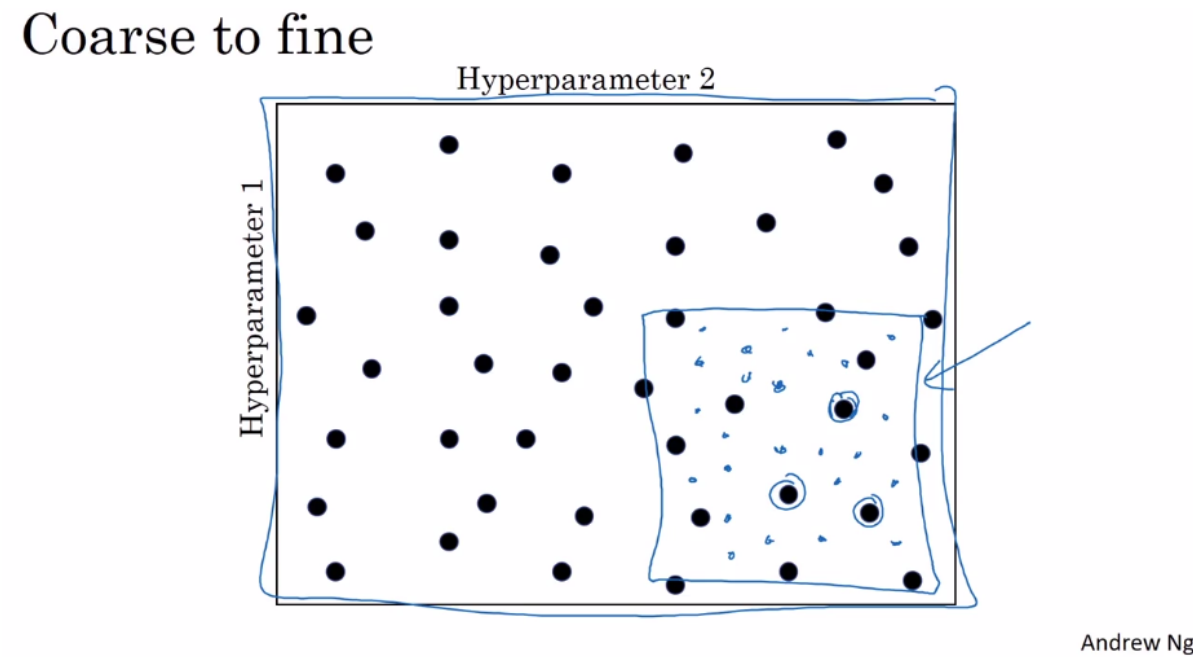
Tuning Process


Hyperparameters



Try random values: Don’t use a grid

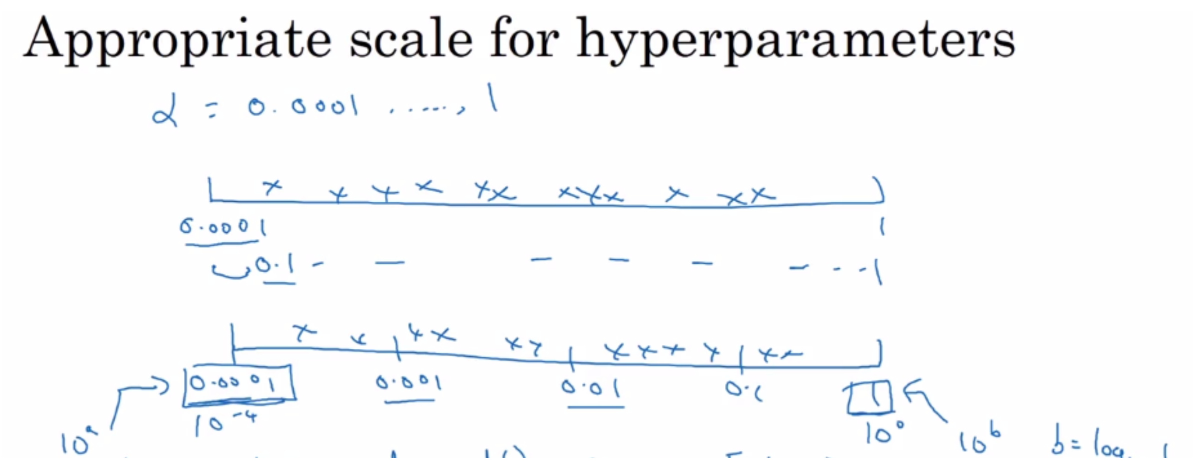
Andrew Ng



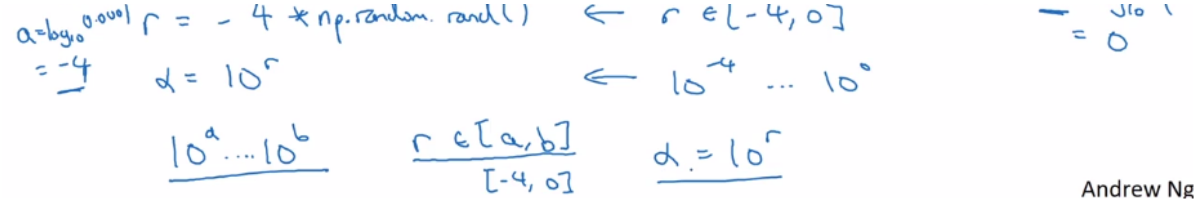
Coarse to fine

Andrew Ng

Using an appropriate scale to pick hyperparameters



Appropriate scale for hyperparameters

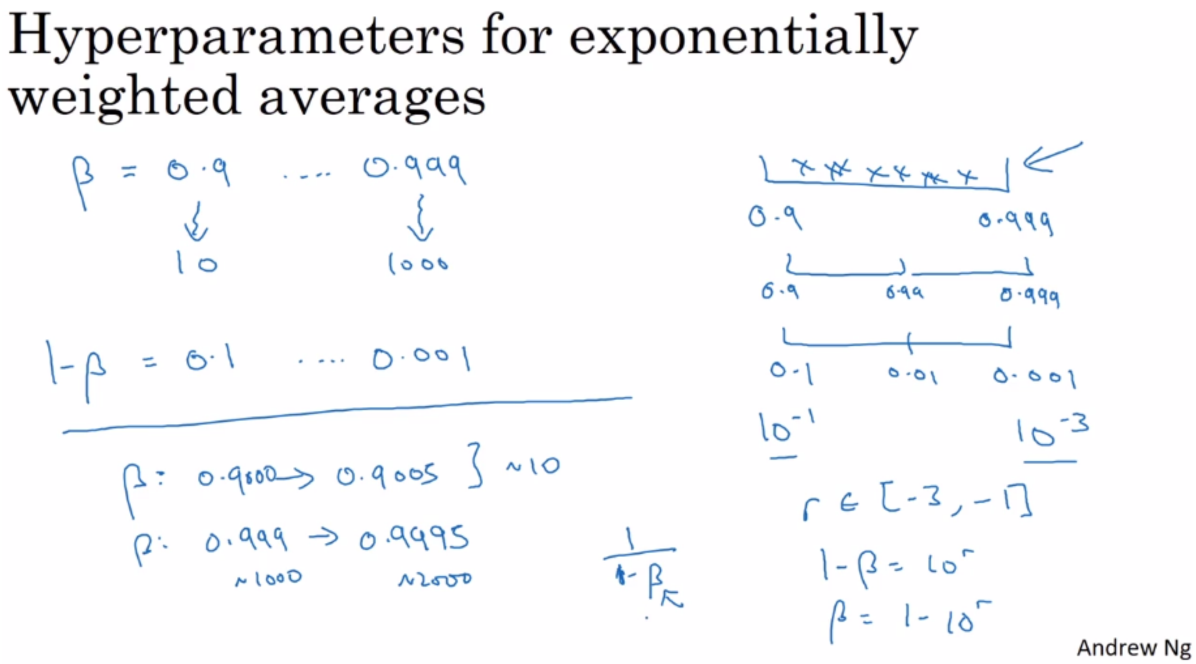


$a = \log_{10} 0.0001 = -4$ $r = -4 * \text{np.random.rand()}$ ← $r \in [-4, 0]$ $= 0$

$\alpha = 10^r$ ← $10^{-4} \dots 10^0$

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

Andrew Ng



Hyperparameters for exponentially weighted averages

$\beta = 0.9 \dots 0.999$

10 1000

$1 - \beta = 0.1 \dots 0.001$

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

$\beta: 0.999 \rightarrow 0.9995$ ~ 1000 ~ 2000

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

0.9 0.999

0.9 0.99 0.999

0.1 0.01 0.001

10^{-1} 10^{-3}

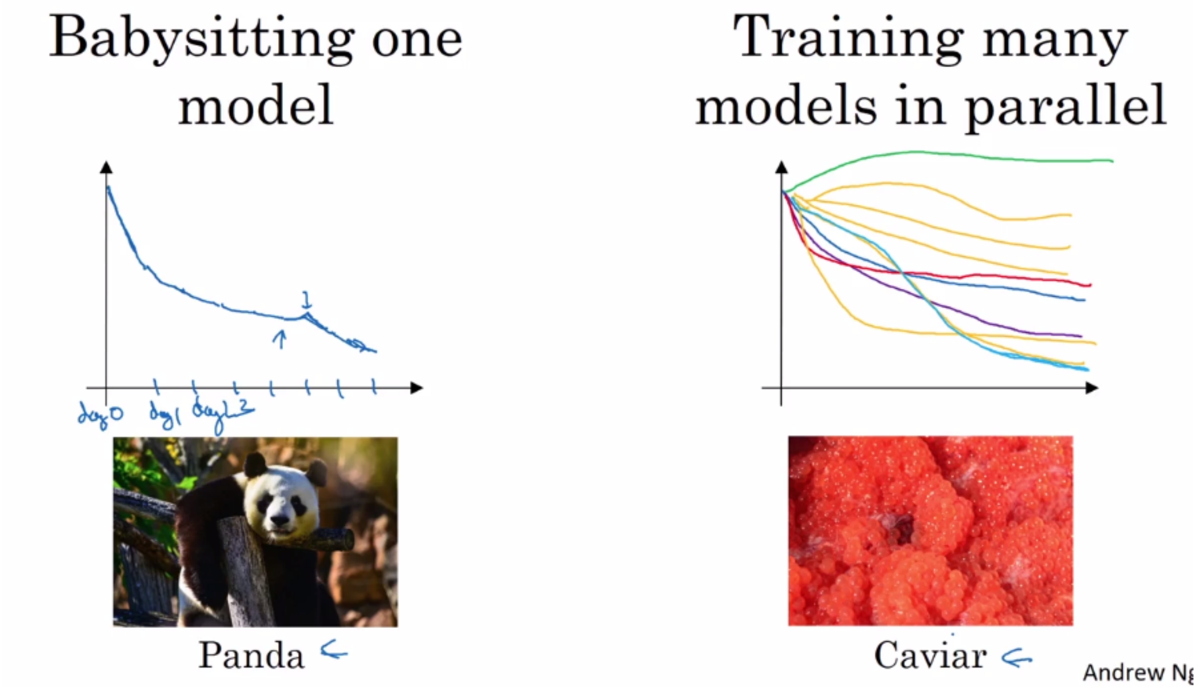
$r \in [-3, -1]$

$1 - \beta = 10^r$

$\beta = 1 - 10^r$

Andrew Ng

Hyperparameters tuning in practice: Pandas vs. Caviar



Babysitting one model

Training many models in parallel

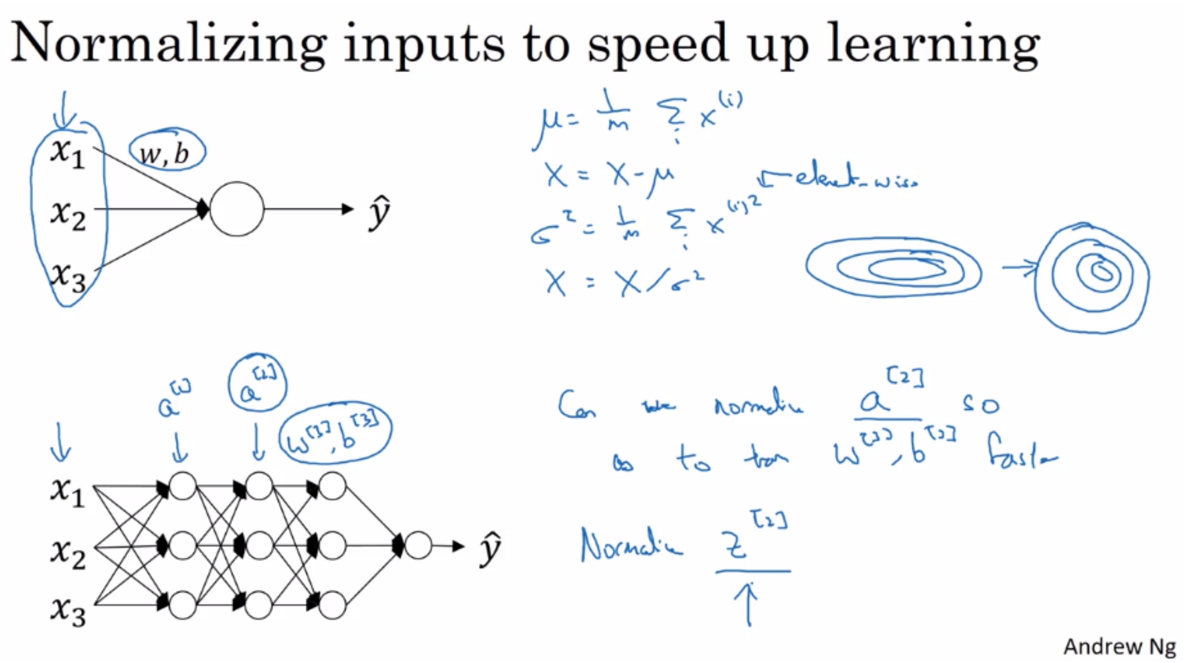
Panda ←

Caviar ←

Andrew Ng

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

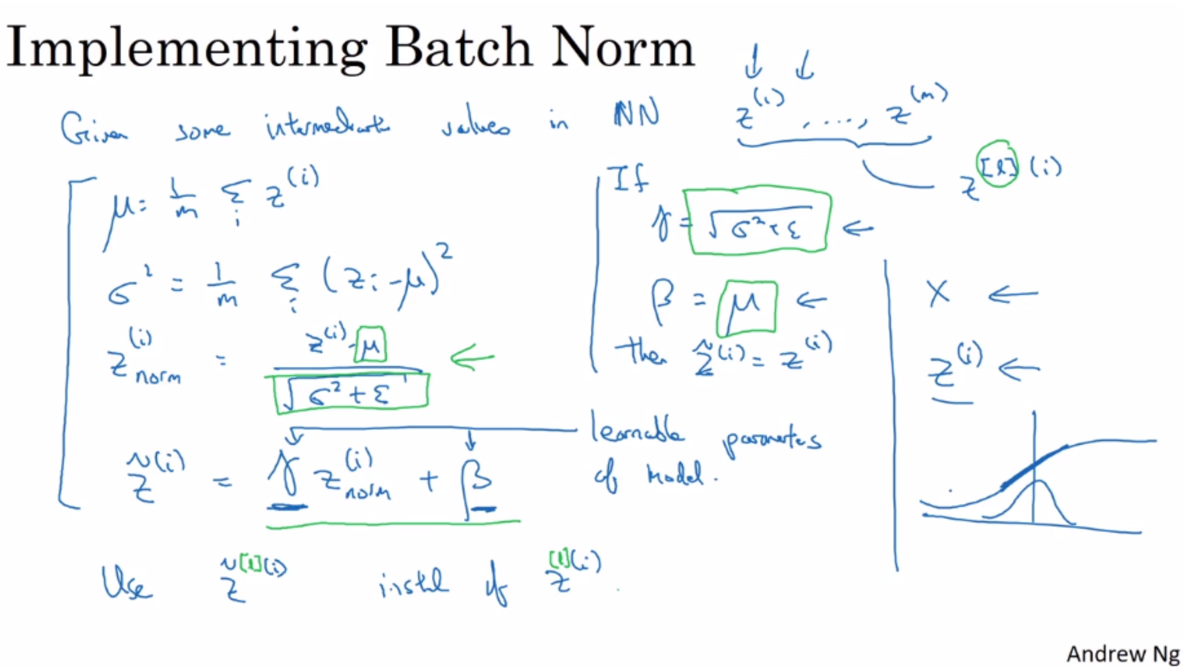
$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$

$X = X / \sigma^2$

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

Normalize $z^{[2]}$

Andrew Ng



Implementing Batch Norm

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

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

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

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

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

learnable parameters of model.

If

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

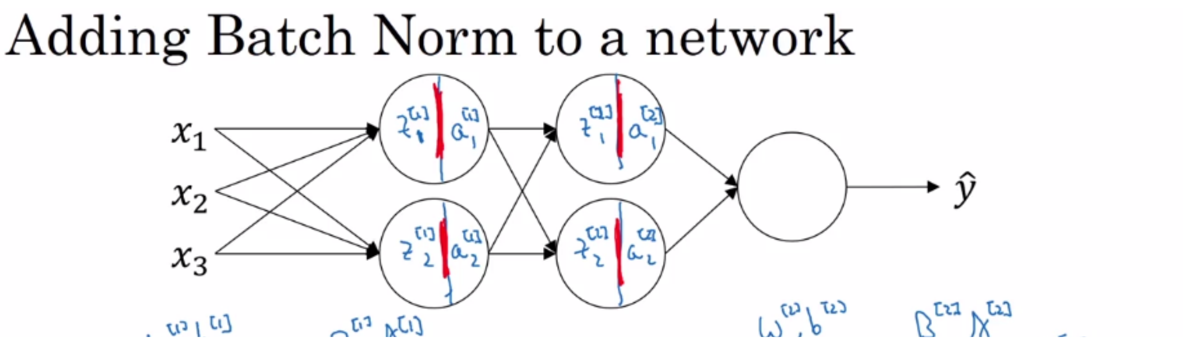
$\beta = \boxed{\mu}$ ←

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

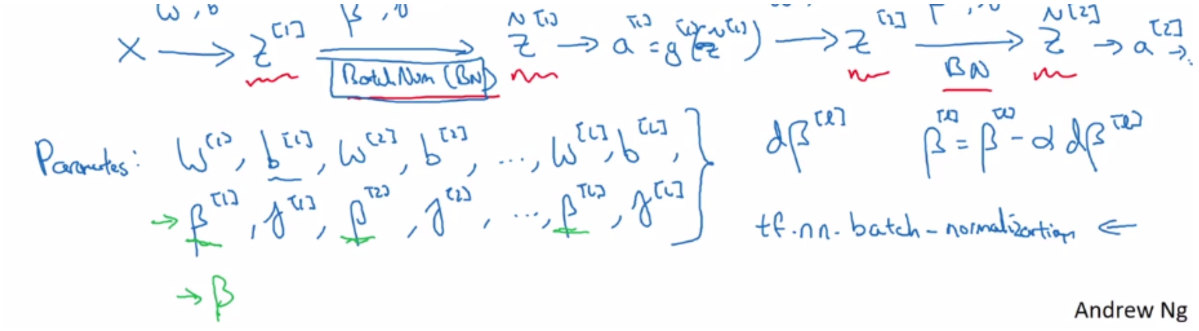
Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

Andrew Ng

Fitting Batch Norm into a neural network



Adding Batch Norm to a network



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

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

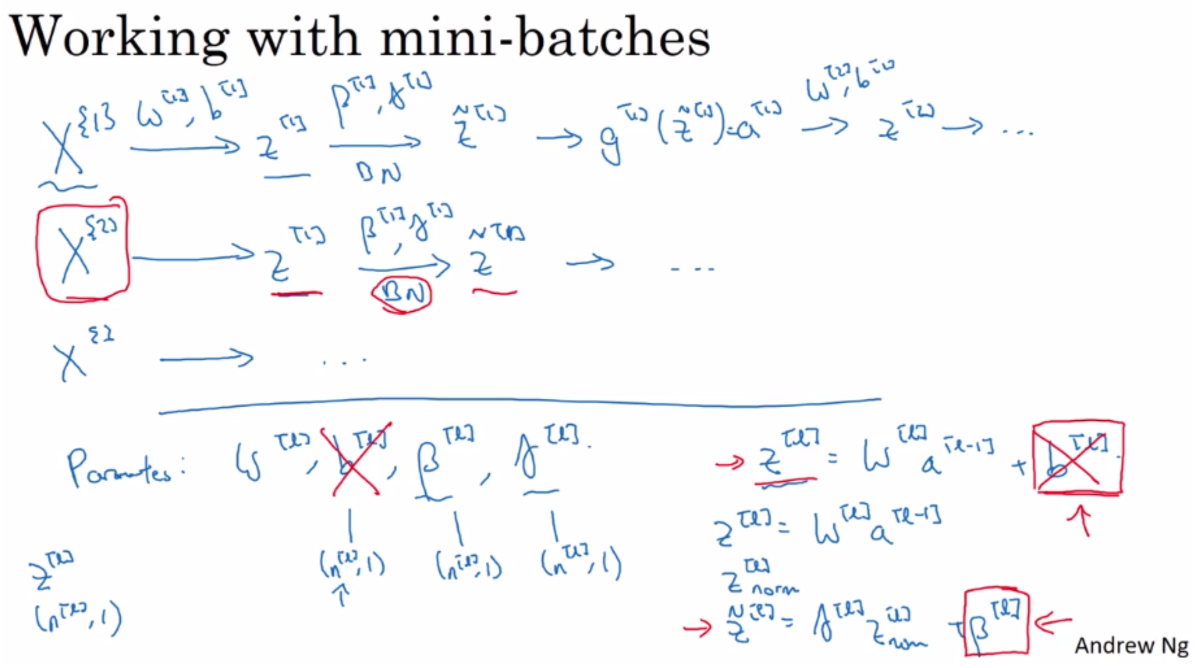
$\rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots, \beta^{[L]}, \gamma^{[L]}$

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

tf.nn.batch_normalization $\leftarrow$

$\rightarrow \beta$

Andrew Ng



Working with mini-batches

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

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

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

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

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

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

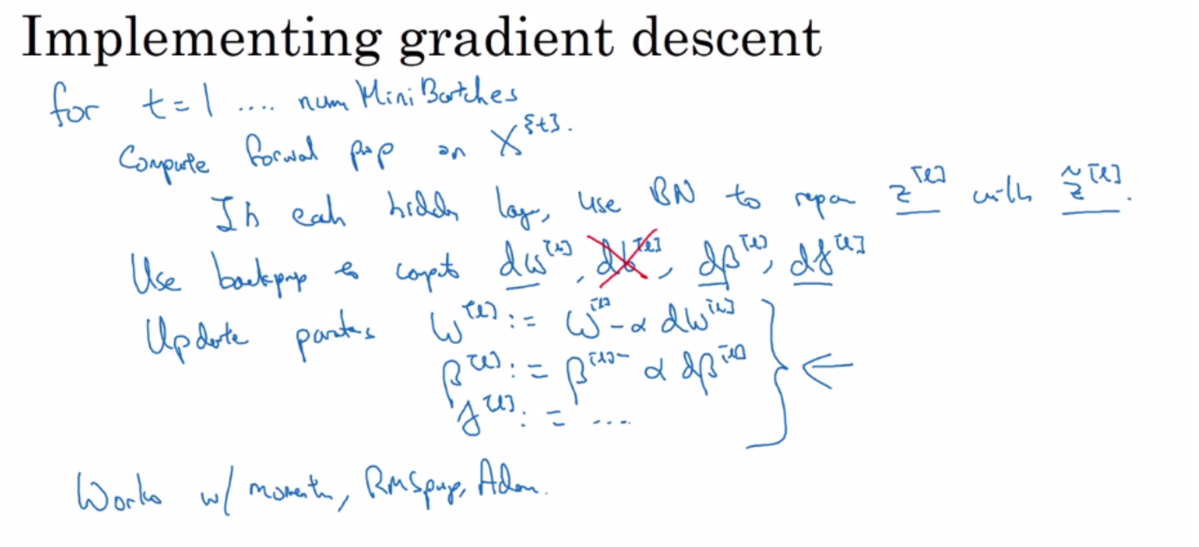
$\rightarrow z^{[l]} = W^{[l]} a^{[l-1]} + \boxed{\cancel{b^{[l]}}}$

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

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

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

Andrew Ng

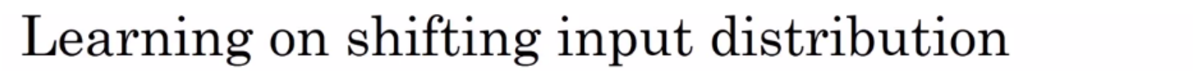


Implementing gradient descent

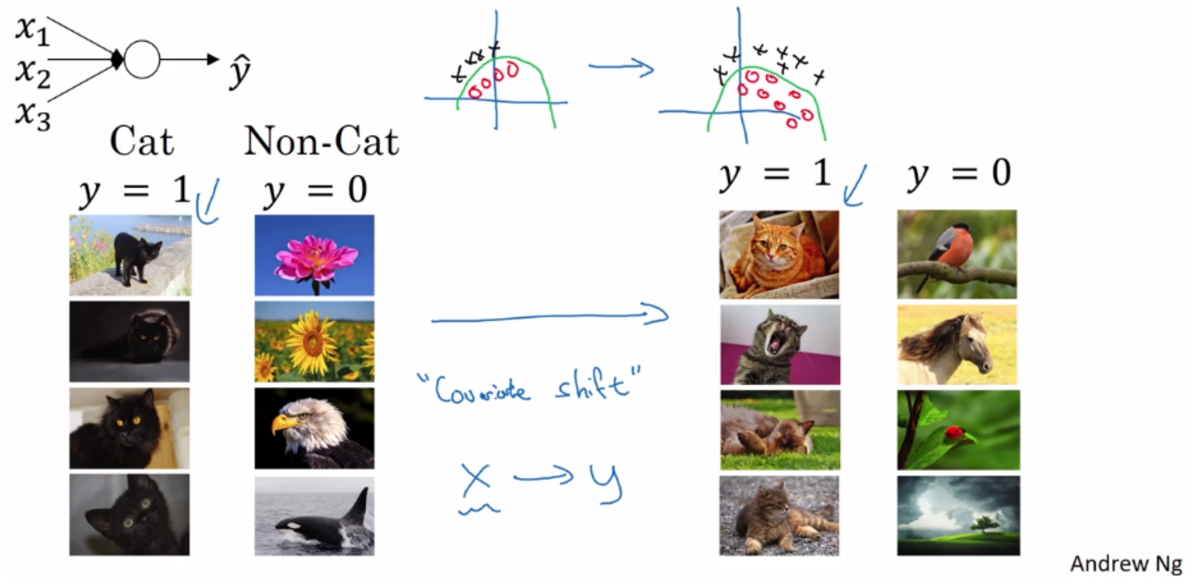
```
for t = 1 .... numMiniBatches
    Compute forward prop on $X^{\{t\}}$.
        In each hidden layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$.
    Use backprop to compute $dW^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$
    Update params  $W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$
                   $\beta^{[l]} := \beta^{[l]} - \alpha\, d\beta^{[l]}$
                   $\gamma^{[l]} := \ldots$
```

Works w/ momentum, RMSprop, Adam.

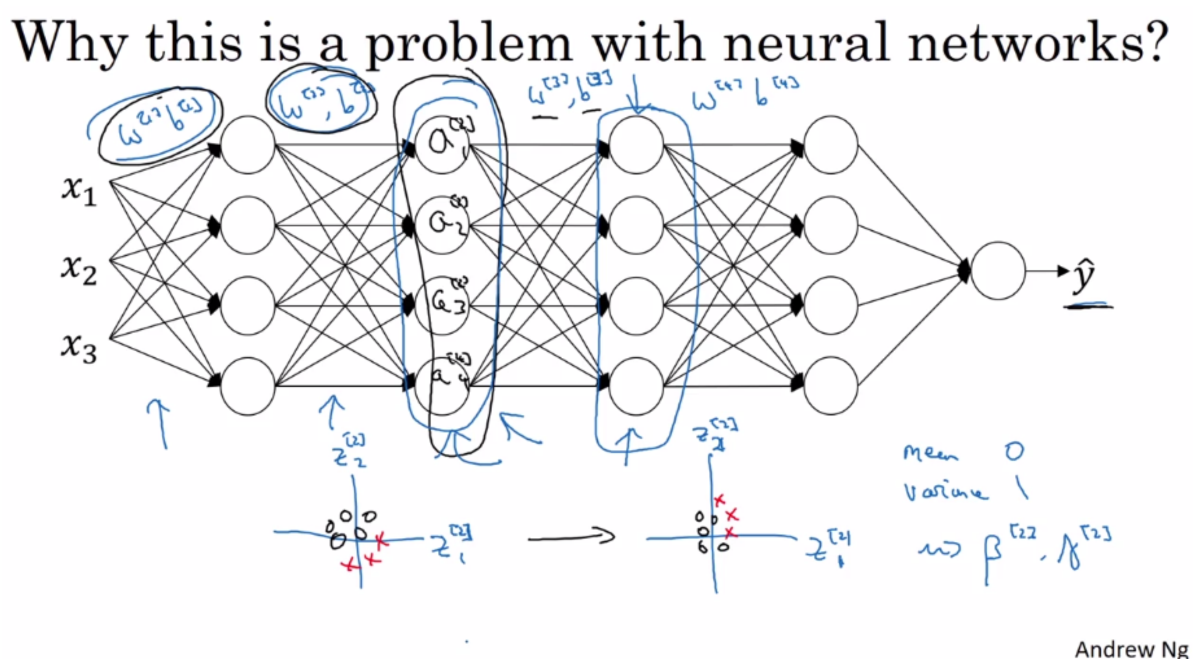
Why does Batch Norm work?



Learning on shifting input distribution

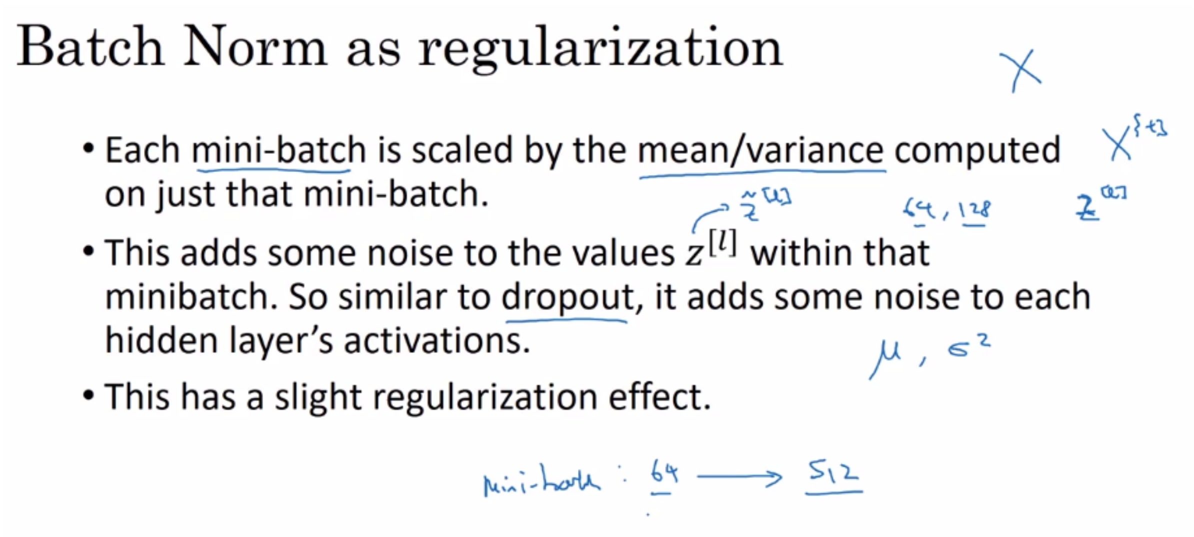


Andrew Ng



Why this is a problem with neural networks?

Andrew Ng

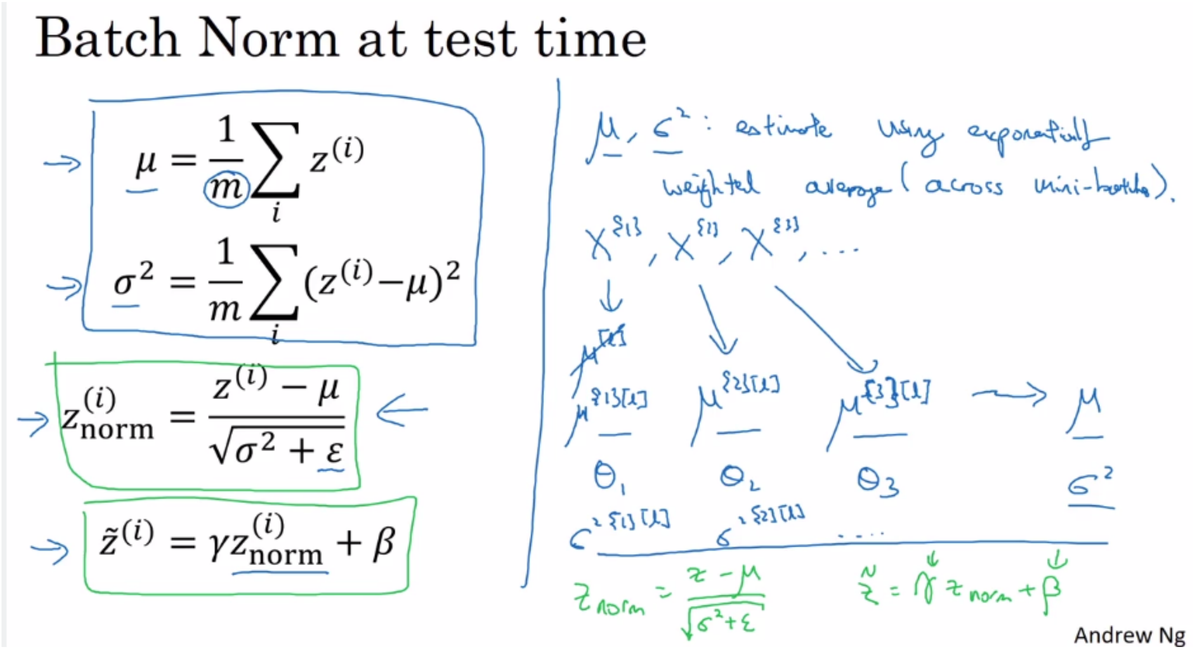


Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- 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.
- This has a slight regularization effect.

X
$X^{\{t\}}$
$\tilde{z}^{[l]}$
64, 128
$z^{[l]}$
μ, σ^2
mini-batch : 64 → 512

Batch Norm at test time



Batch Norm at test time

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

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

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

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

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

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

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

θ_1 θ_2 θ_3 σ^2

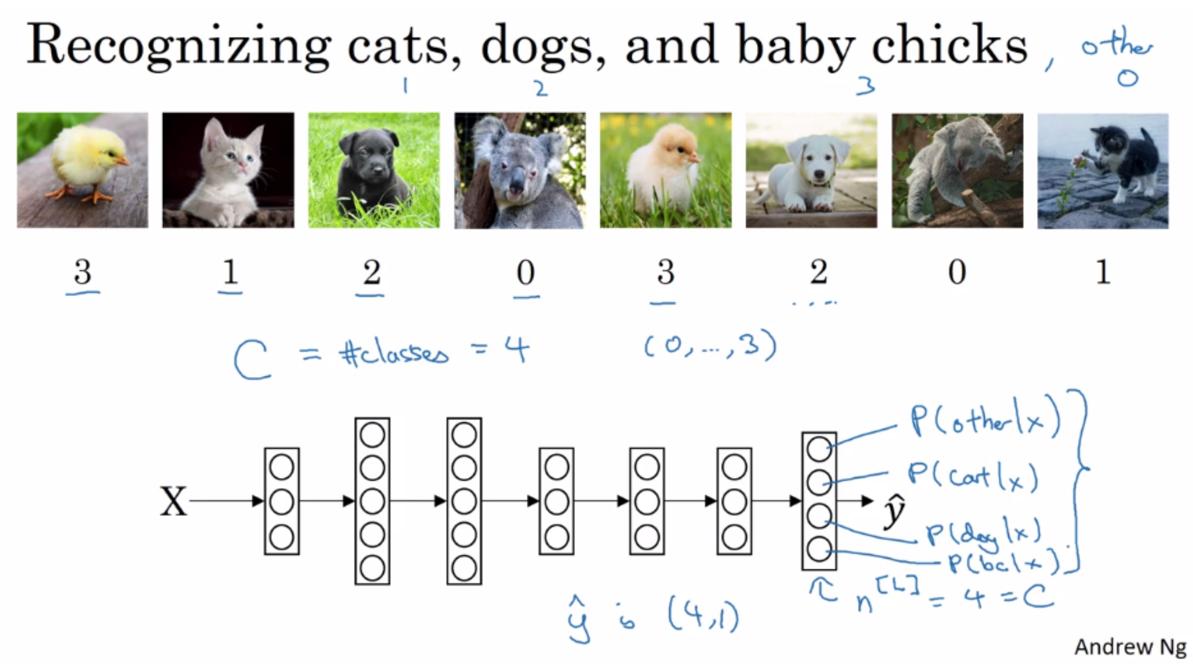
$\sigma^{2\{1\}[l]}$ $\sigma^{2\{2\}[l]}$ ....

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

Andrew Ng

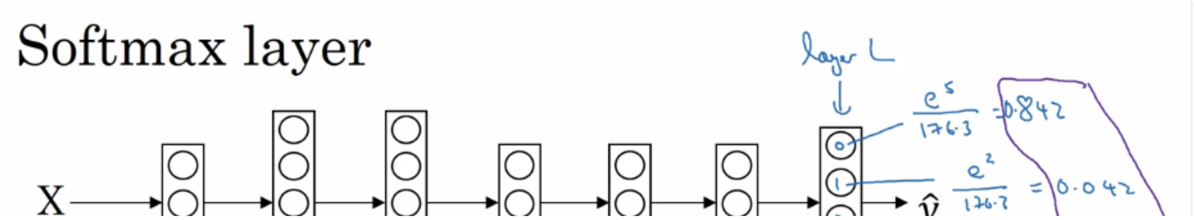
Multi-class classificiation

Softmax Regression

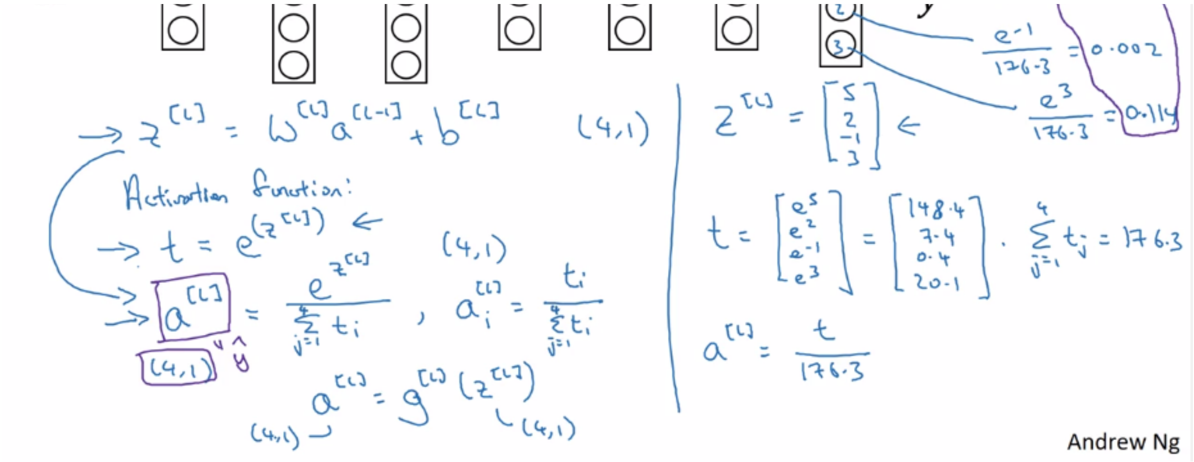


Recognizing cats, dogs, and baby chicks, other

Andrew Ng



Softmax layer



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

Activation function:

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

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

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

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

(4,1) (4,1)

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

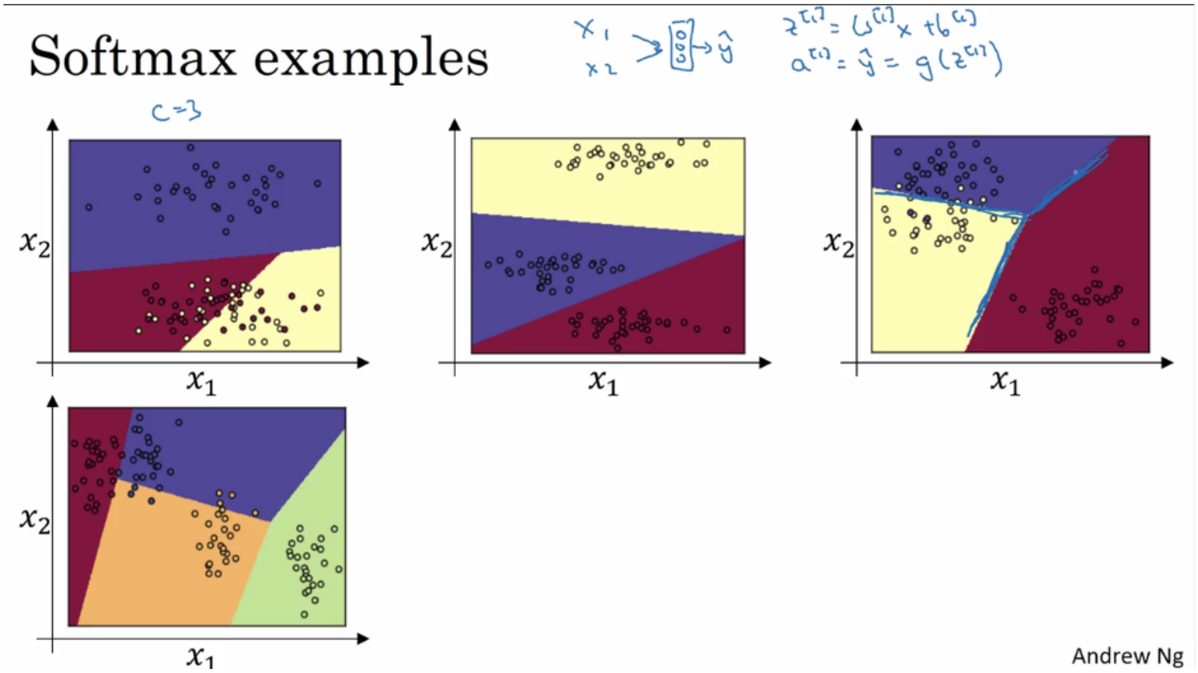
$\frac{e^{-1}}{176.3} = 0.002$

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

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

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

Andrew Ng



Softmax examples

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

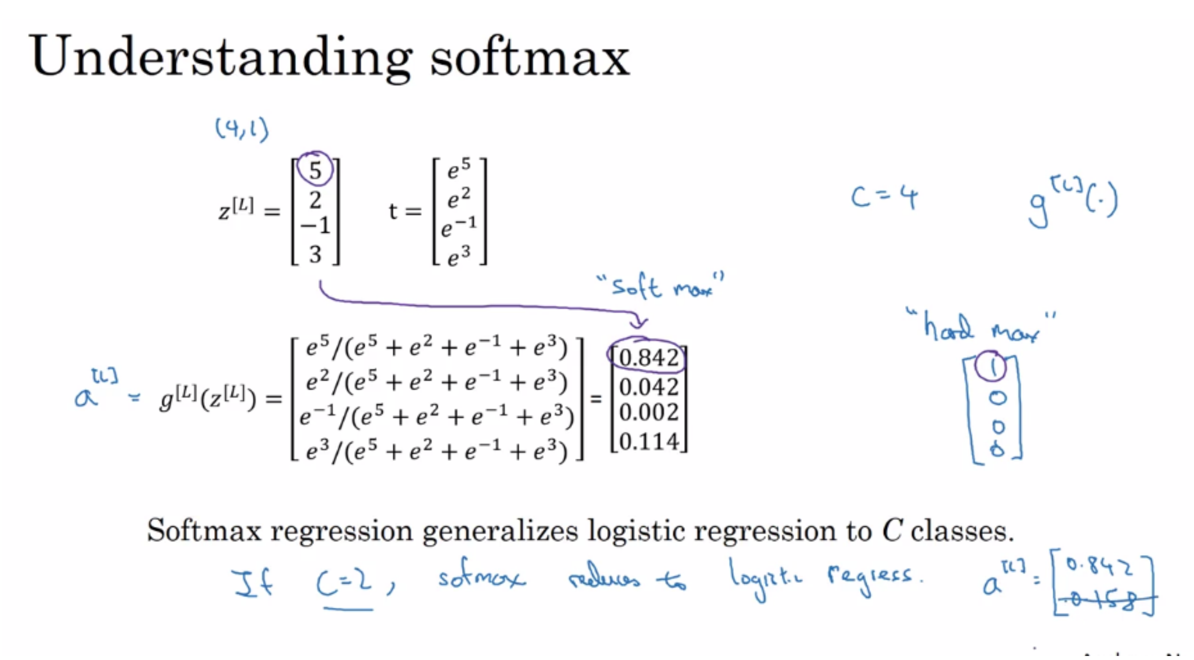
$z^{[L]} = W^{[L]} x + b^{[L]}$

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

C=3

Andrew Ng

Training a softmax classifier



Understanding softmax

(4,1)

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

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

"soft max"

"hard max"

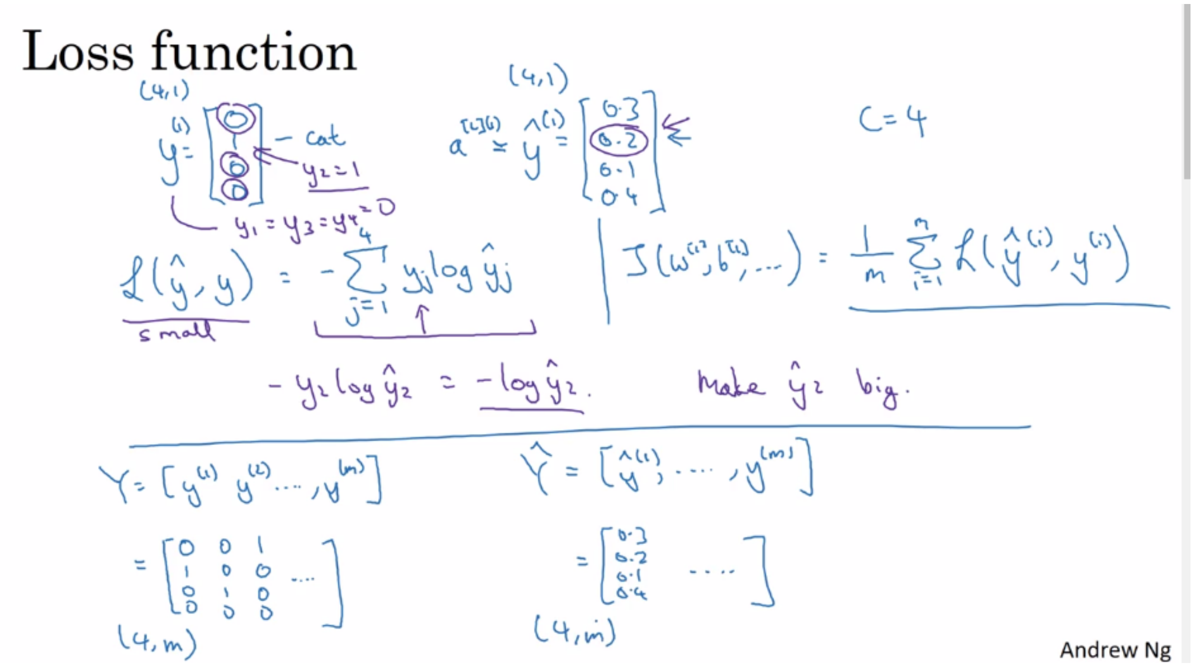
$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$

$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Softmax regression generalizes logistic regression to C classes.

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

Andrew Ng



Loss function

$C=4$

$$\underset{(4,1)}{y^{(i)}} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \text{— cat}, \quad y_2 = 1, \quad y_1 = y_3 = y_4 = 0$$

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

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

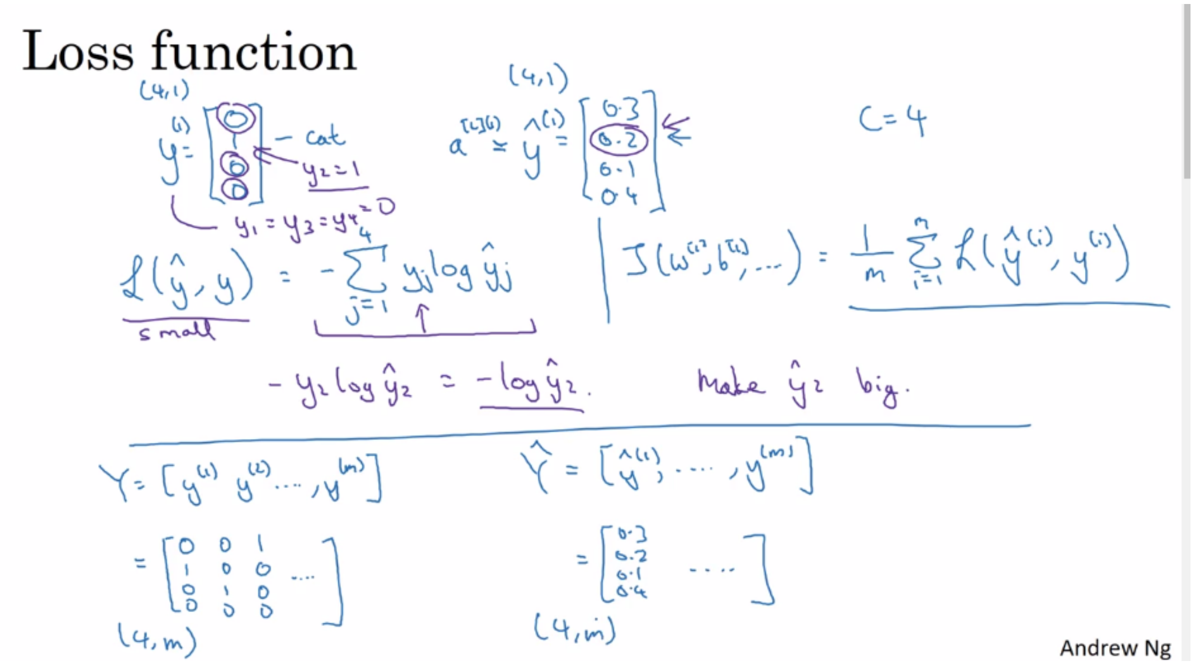
$$J(w^{[1]}, b^{[1]}, \dots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$-y_2 \log \hat{y}_2 = \underline{-\log \hat{y}_2}. \quad \text{Make } \hat{y}_2 \text{ big.}$$

$$Y = [y^{(1)}\ y^{(2)} \dots, y^{(m)}] = \underset{(4,m)}{\begin{bmatrix}0 & 0 & 1 & \dots\\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}}$$

$$\hat{Y} = [\hat{y}^{(1)}, \dots, y^{(m)}] = \underset{(4,m)}{\begin{bmatrix}0.3 & \dots\\ 0.2 & \\ 0.1 & \\ 0.4 & \end{bmatrix}}$$

Andrew Ng



Loss function

$C=4$

$$\underset{(4,1)}{y^{(i)}} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \text{— cat}, \quad y_2 = 1, \quad y_1 = y_3 = y_4 = 0$$

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

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

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

$$-y_2 \log \hat{y}_2 = \underline{-\log \hat{y}_2}. \quad \text{Make } \hat{y}_2 \text{ big.}$$

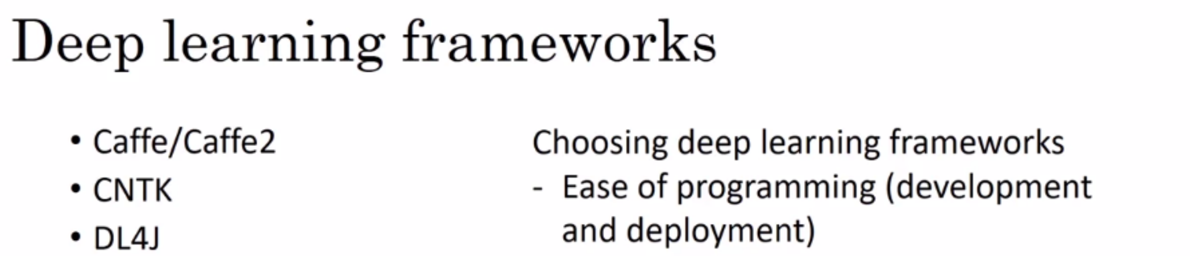
$$Y = [y^{(1)}\ y^{(2)} \dots, y^{(m)}] = \underset{(4,m)}{\begin{bmatrix}0 & 0 & 1 & \dots\\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}}$$

$$\hat{Y} = [\hat{y}^{(1)}, \dots, y^{(m)}] = \underset{(4,m)}{\begin{bmatrix}0.3 & \dots\\ 0.2 & \\ 0.1 & \\ 0.4 & \end{bmatrix}}$$

Andrew Ng

Introduction to programming frameworks

Deep Learning Frameworks

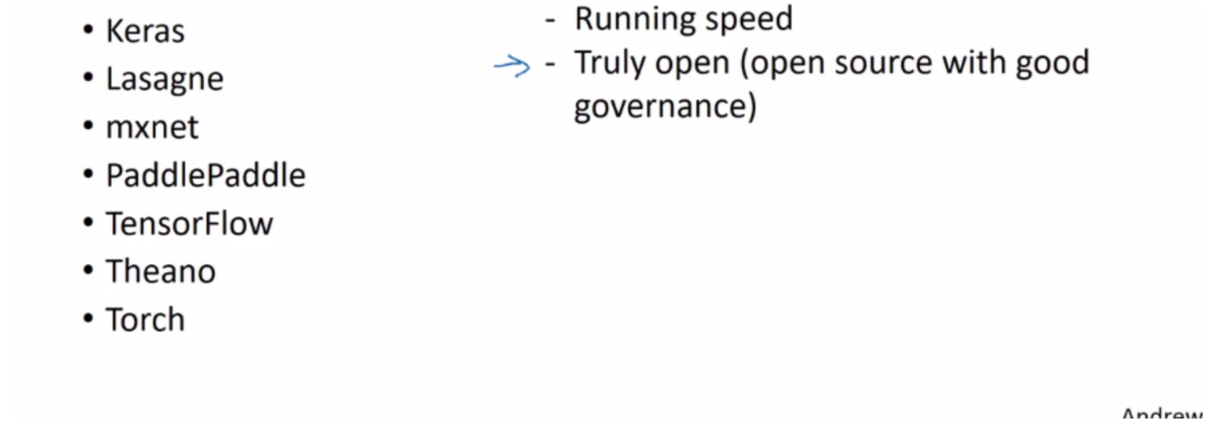


Deep learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J

Choosing deep learning frameworks

- Ease of programming (development and deployment)



- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

- Running speed
- Truly open (open source with good governance)

Andrew

TensorFlow

In [1]:
```
import tensorflow as tf
import numpy as np
```

In [2]:
```
w = tf.Variable(0, dtype=tf.float32)
cost = tf.add(tf.add(w**2, tf.multiply(-10., w)), 25)
```

In [3]:
```
train = tf.train.GradientDescentOptimizer(0.01).minimize(cos
t)
```

In [4]:
```
init = tf.global_variables_initializer()
```

In [5]:
```
session = tf.Session()
```

In [6]:
```
%time session.run(init)
```

```
CPU times: user 6.07 ms, sys: 2.41 ms, total: 8.48 ms
Wall time: 5.82 ms
```

In [7]:
```
print(session.run(w))
```

```
0.0
```

In [8]:
```
%time session.run(train)
```

```
CPU times: user 16.1 ms, sys: 2.34 ms, total: 18.4 ms
Wall time: 15.9 ms
```

In [9]:
```
%time print(session.run(w))
```

```
0.099999994
CPU times: user 815 µs, sys: 243 µs, total: 1.06 ms
Wall time: 807 µs
```

In [10]:
```
%load_ext version_information
%version_information tensorflow, numpy
```

Out[10]:

Software	Version
Python	3.6.6 64bit [GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-

Python	602.0.53)]
IPython	7.0.1
OS	Darwin 17.7.0 x86_64 i386 64bit
tensorflow	1.10.0
numpy	1.15.1
Sun Oct 14 21:48:53 2018 MDT	