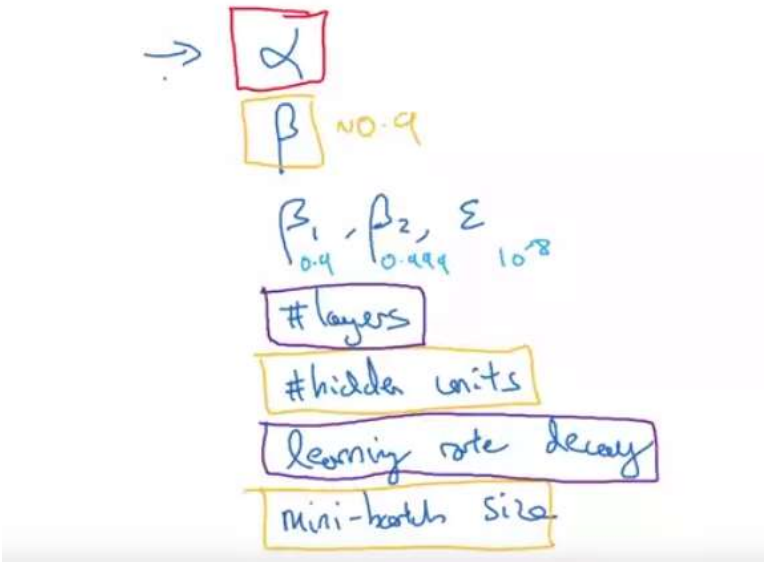
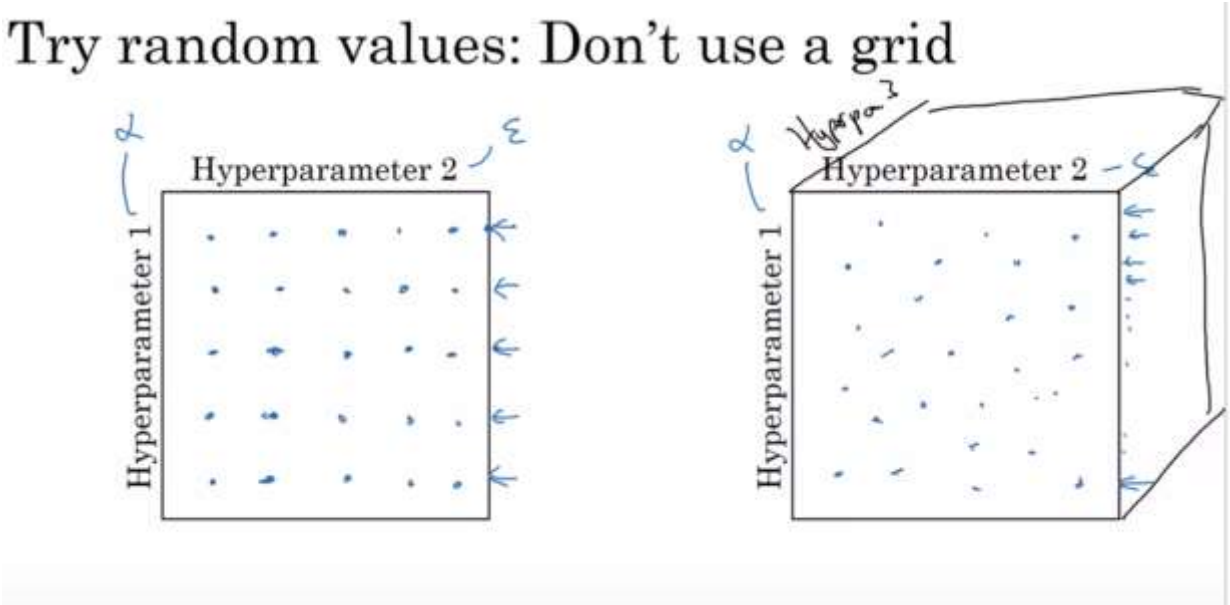
Hyperparameters Tuning

Week 3

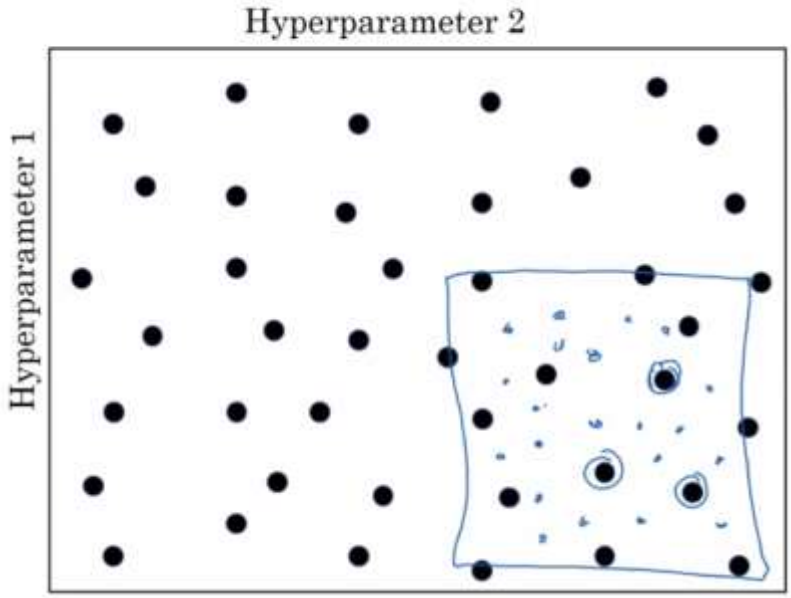
Hyperparameters



Try random values: Don't use a grid



Coarse to fine



Picking hyperparameters at random

→ $n^{[l]}$ = 50, ..., 100

50 ——— 100

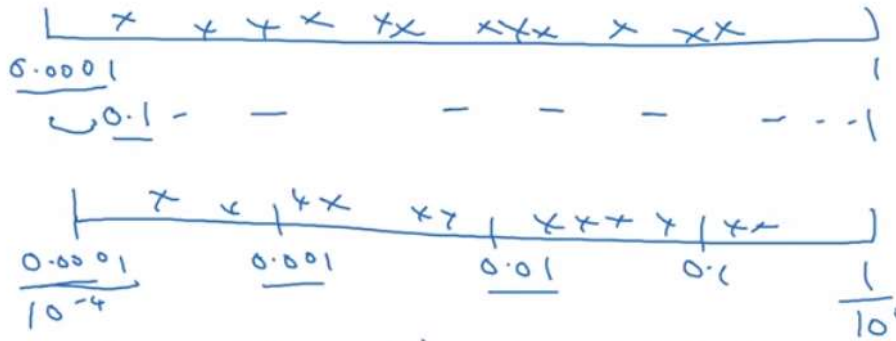
→ #layers L: 2 – 4

2, 3, 4

We should not randomly search for hyperparameters, but randomly search in a specific range of values.

Appropriate scale for hyperparameters

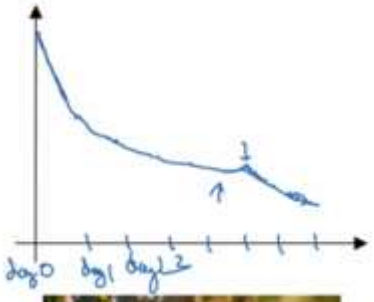
$\alpha = 0.0001 \ldots, 1$



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

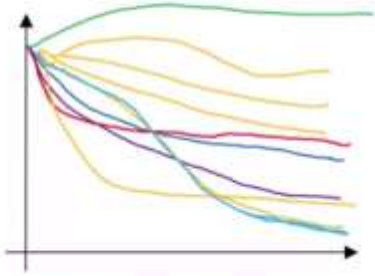
$\alpha = 10^{r} \quad \leftarrow \quad 10^{-4} \ldots 10^{0}$

Babysitting one model



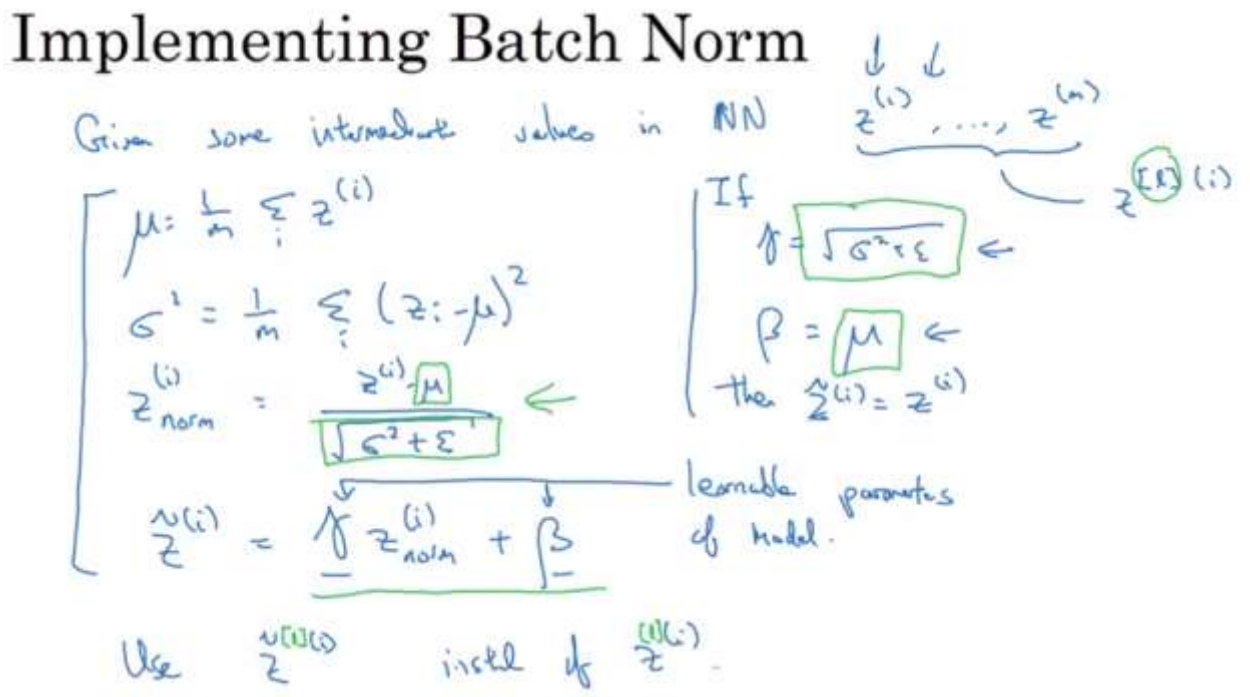
Panda ←

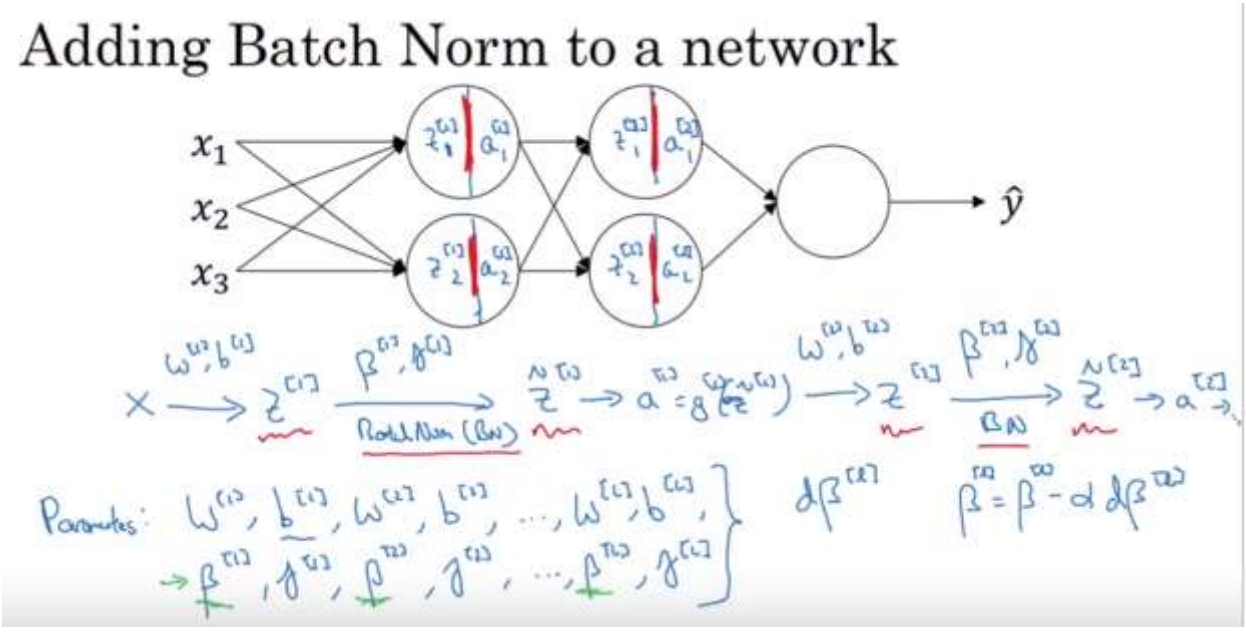
Training many models in parallel



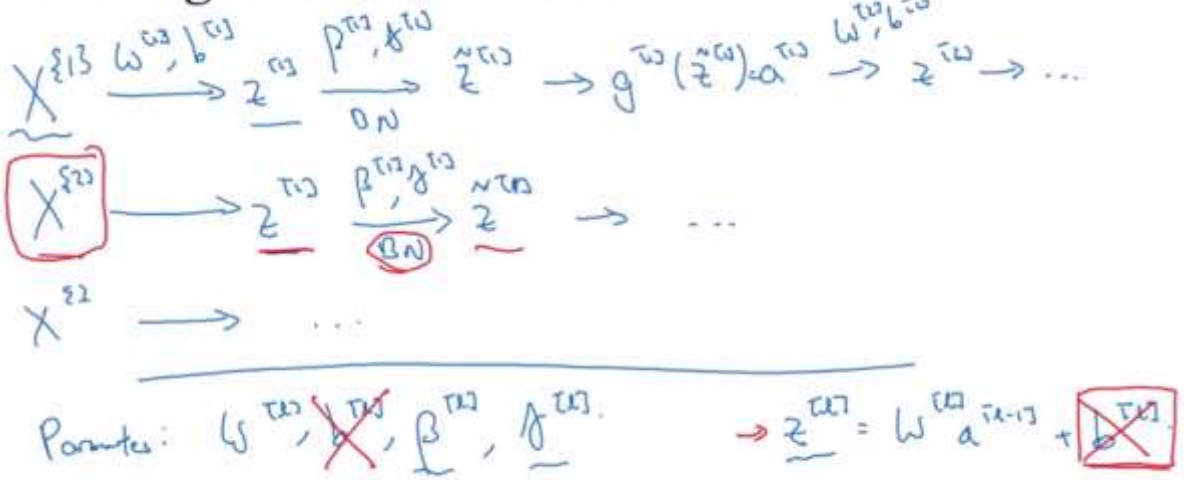
Caviar ←

Batch Normalization





Working with mini-batches



Implementing gradient descent

for $t = 1 \ldots$ 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 $dW^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$

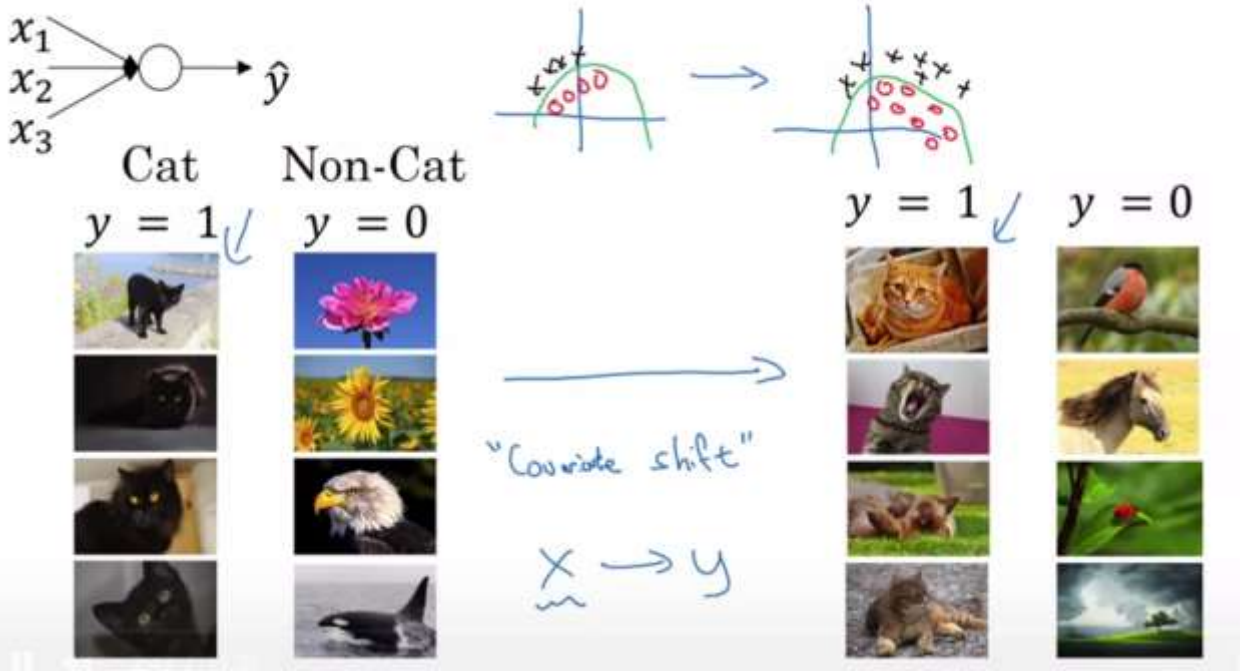
Update parameters $W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$

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

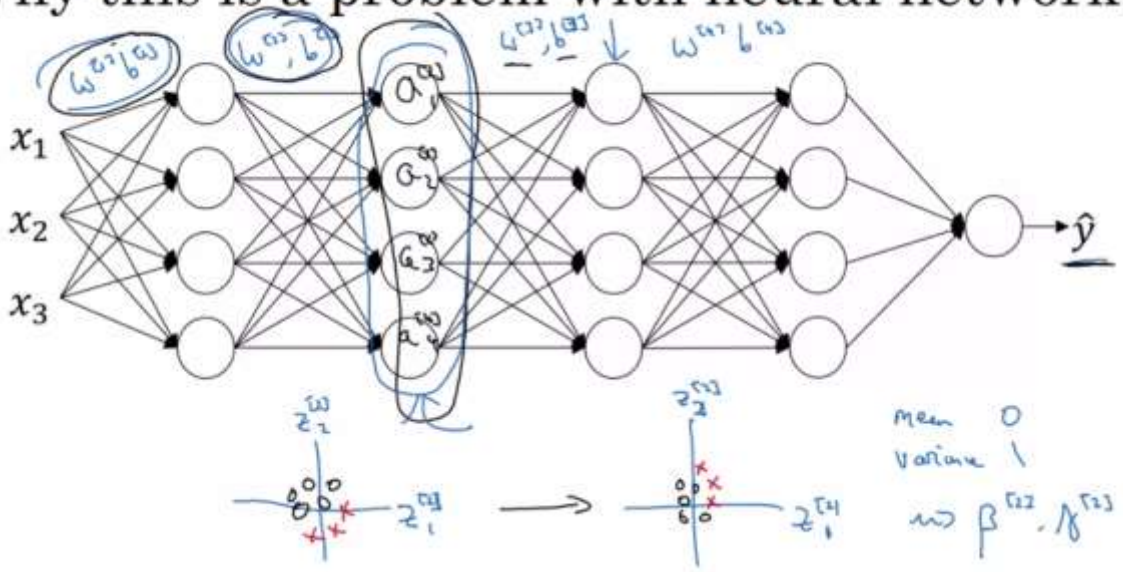
$\gamma^{[l]} := \ldots$

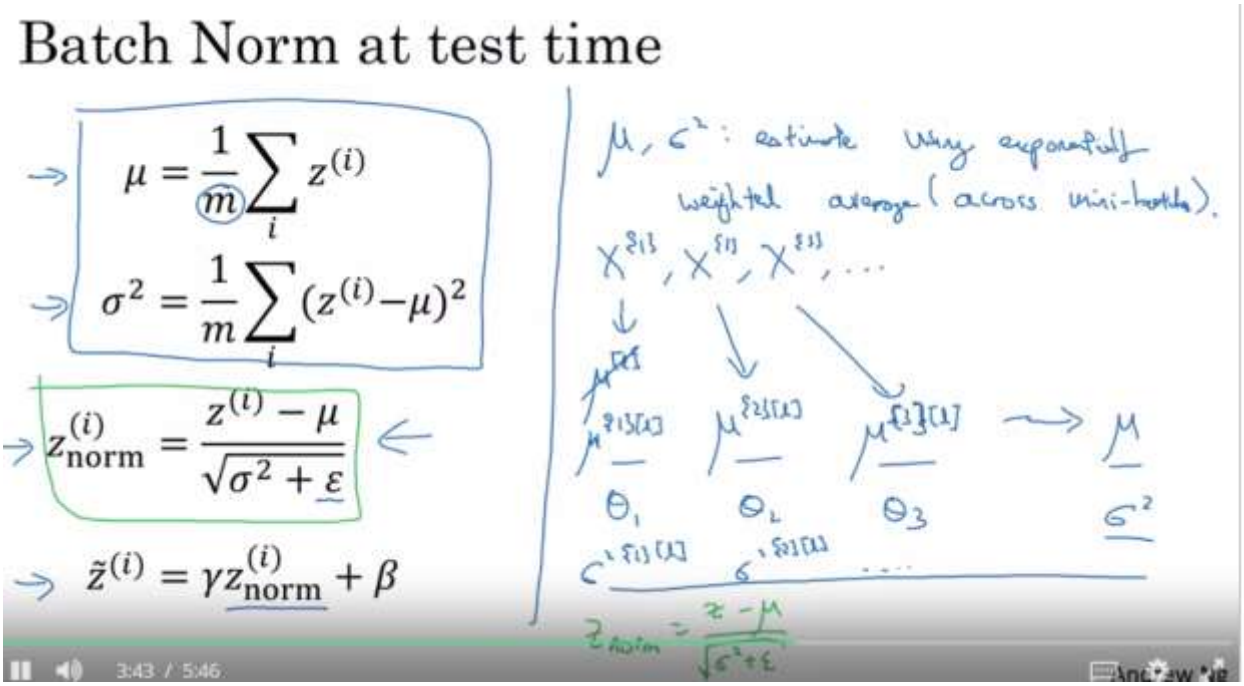
Works w/ momentum, RMSprop, Adam.

Learning on shifting input distribution

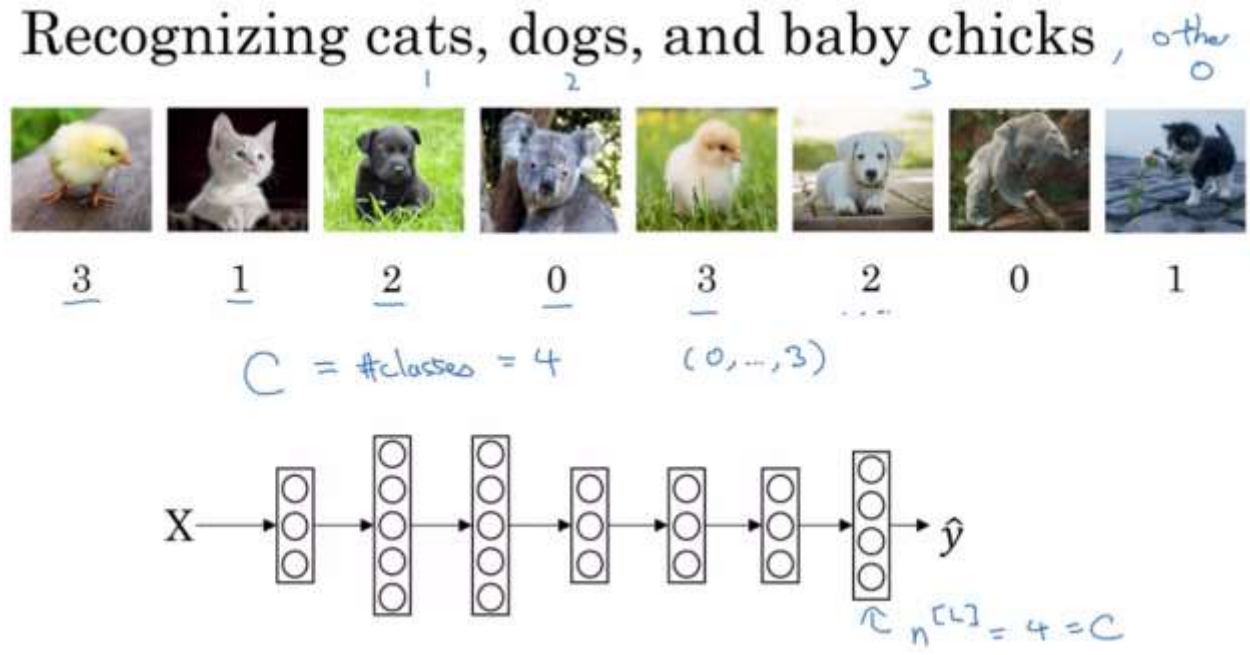


Why this is a problem with neural networks?

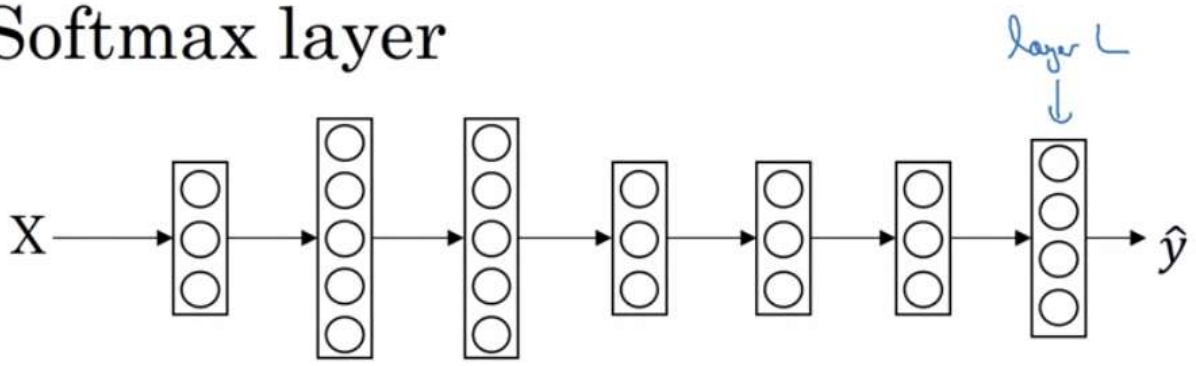




At test time, we will not have a mini-batch to compute the mean and std; during training our network, we will keep an exponentially weighted average for the mean and std., for each layer.



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)

$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$ | $J(w^{[1]}, b^{[1]}, \ldots) = \frac{1}{m} \sum_{i=1} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

small

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

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)

Tensorflow

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

In [5]:
```
w = tf.Variable(0,dtype=tf.float32)
#cost = tf.add(tf.add(w**2,tf.multiply(-10.,w)),25)
cost = w**2 - 10*w + 25
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))
```

```
0.0
```

In [6]:
```
session.run(train)
print(session.run(w))
```

```
0.1
```

In [7]:
```
for i in range(1000):
    session.run(train)
print(session.run(w))
```

```
4.99999
```

6:14 / 16:07

```
coefficients = np.array([[1.], [-10.], [25.]])

w = tf.Variable(0,dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
#cost = tf.add(tf.add(w**2,tf.multiply(-10.,w)),25)
#cost = w**2 - 10*w + 25
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))
```

```
0.0
```

```
session.run(train, feed_dict={x:coefficients})
print(session.run(w))
```

As of 2019, Google launched TensorFlow 2. TensorFlow 2 borrowed its syntax from Keras. Keras is a high level library that can operate on top of TensorFlow 1 as well as other deep learning libraries. In later courses of this specialization, you will learn Keras.

- When you code in tensorflow you have to take the following steps:
  - Create a graph containing Tensors (Variables, Placeholders ...) and Operations (tf.matmul, tf.add, ...)
  - Create a session
  - Initialize the session
  - Run the session to execute the graph
- You can execute the graph multiple times as you've seen in model()
- The backpropagation and optimization is automatically done when running the session on the "optimizer" object.