

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



deeplearning.ai

Hyperparameter tuning

Tuning process

Hyperparameters

→ α

β 0.9

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

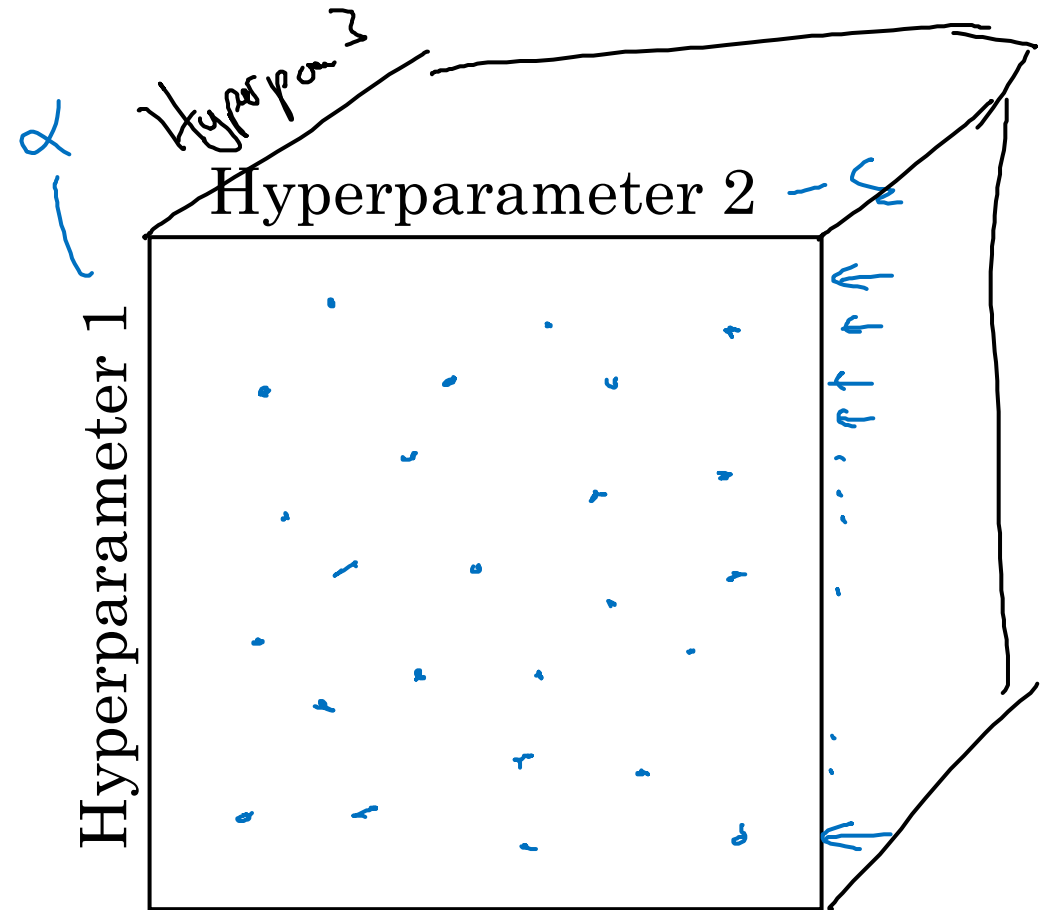
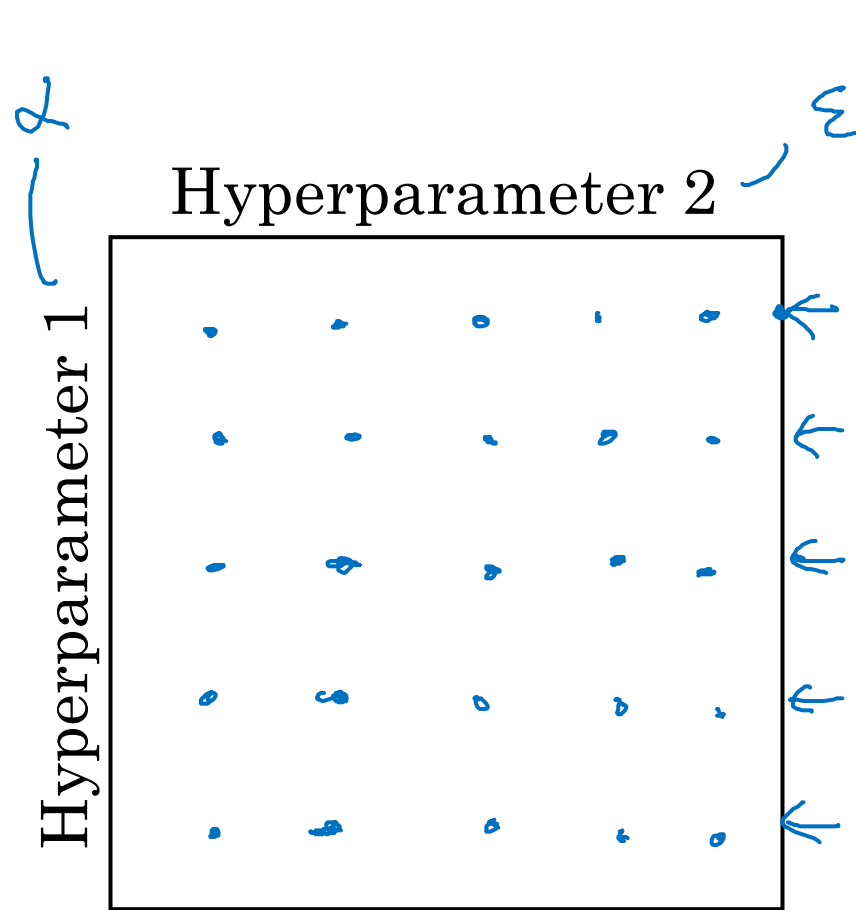
layers

hidden units

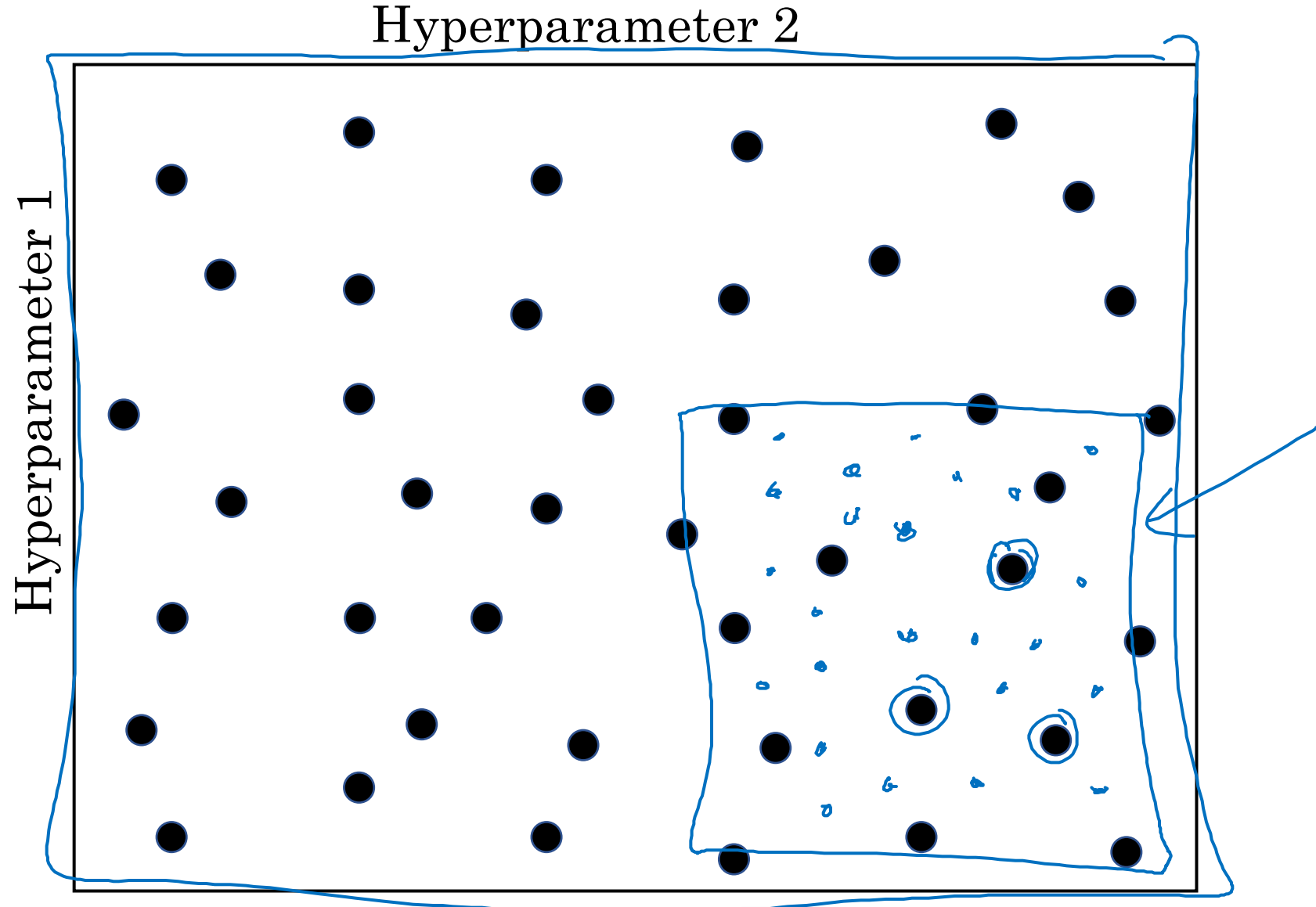
learning rate decay

mini-batch size

Try random values: Don't use a grid



Coarse to fine





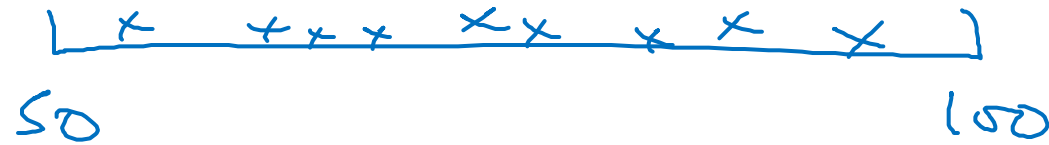
deeplearning.ai

Hyperparameter tuning

Using an appropriate
scale to pick
hyperparameters

Picking hyperparameters at random

→ $n^{\text{test}} = 50, \dots, 100$

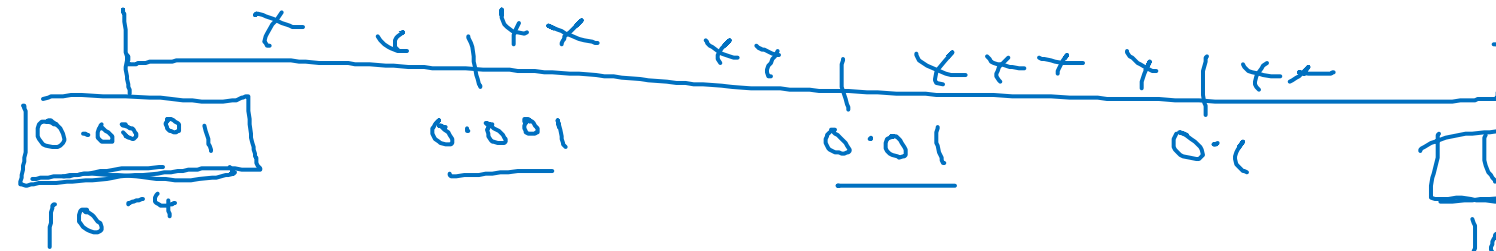
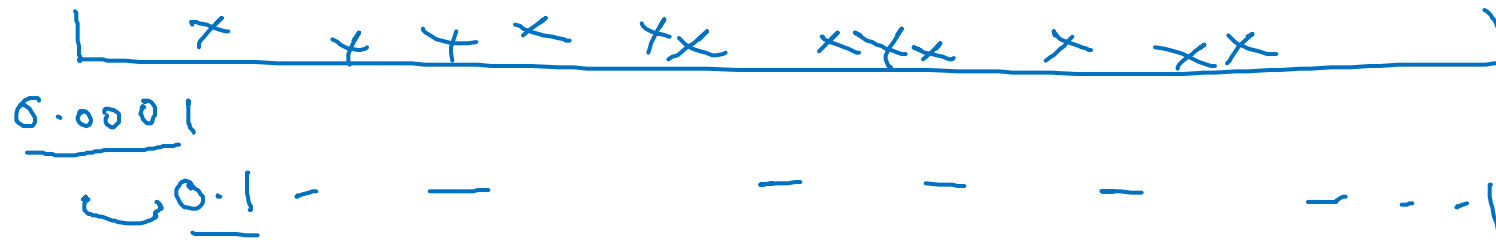


→ #layers $L : 2 - 4$

2, 3, 4

Appropriate scale for hyperparameters

$$\alpha = 0.0001, \dots, 1$$



$$10^a$$

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

$$= -4$$

$$r = -4 * \text{np.random.rand}()$$

$$\alpha = 10^r$$

$$r \in [-4, 0]$$

$$10^{-4} \dots 10^0$$

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

$$= 0$$

$$\frac{10^a \dots 10^b}{}$$

$$\frac{r \in [a, b]}{[-4, 0]}$$

$$\underline{\alpha = 10^r}$$

Hyperparameters for exponentially weighted averages

$$\beta = 0.9 \quad \dots \quad 0.999$$

\downarrow
 10

\downarrow
 1000

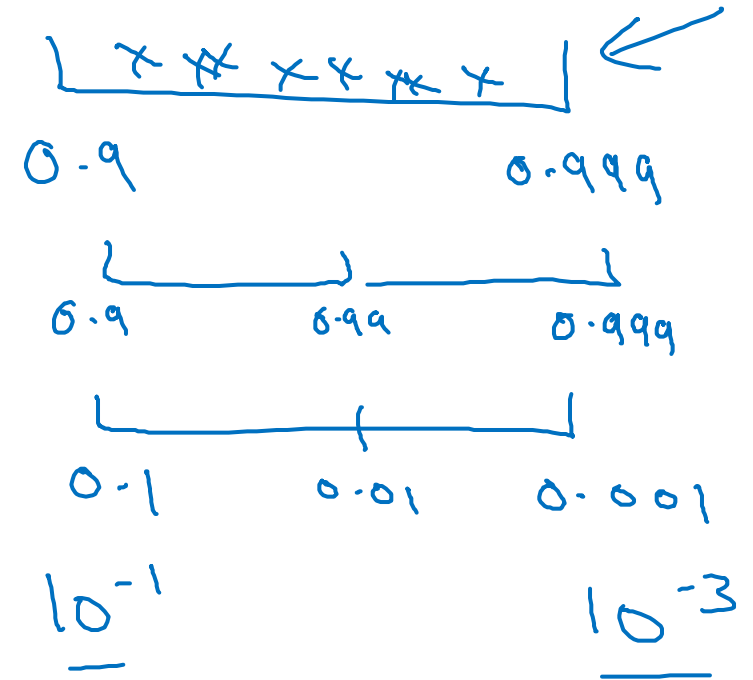
$$1 - \beta = 0.1 \quad \dots \quad 0.001$$

$$\beta: 0.999 \rightarrow 0.9995 \quad \} \sim 10$$

$$\beta: 0.999 \rightarrow 0.9995$$

~ 1000
 ~ 2000

$$\frac{1}{1 - \beta_K}$$



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

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

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

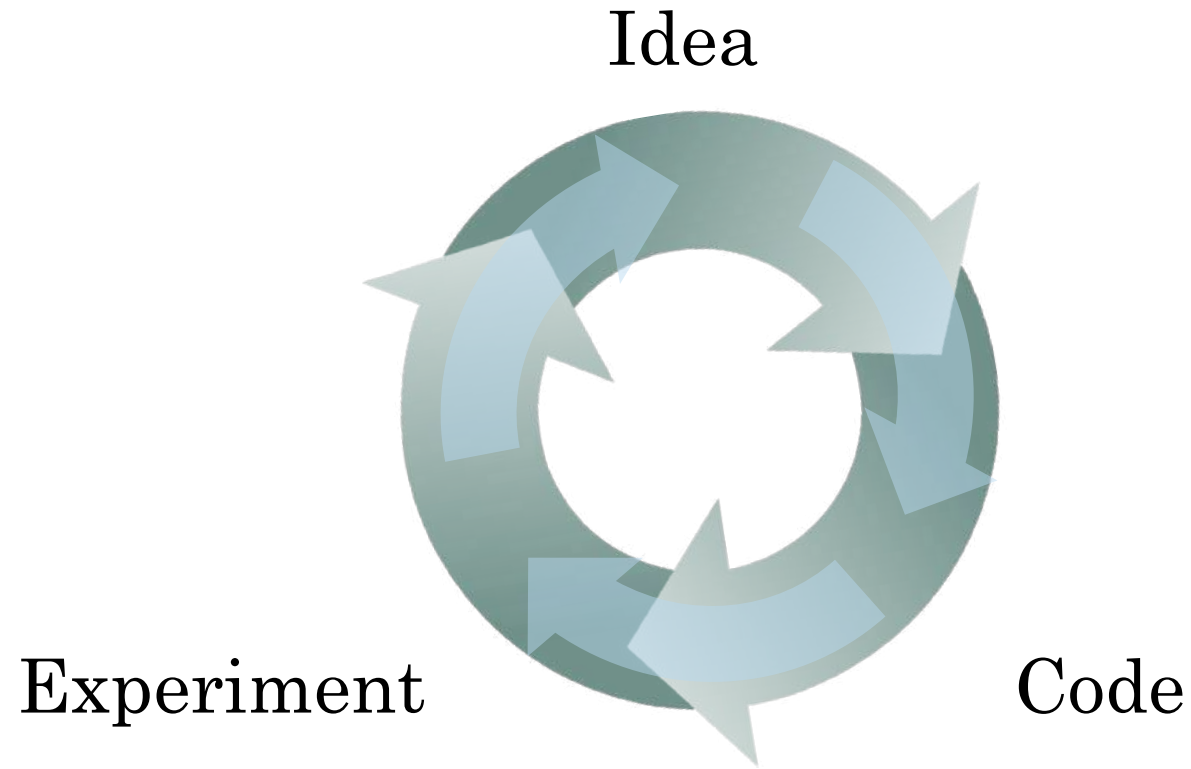


deeplearning.ai

Hyperparameters tuning

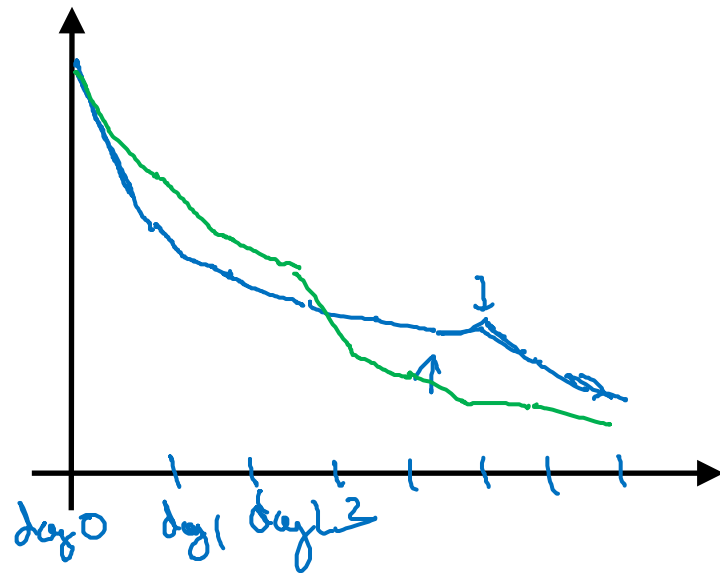
Hyperparameters
tuning in practice:
Pandas vs. Caviar

Re-test hyperparameters occasionally



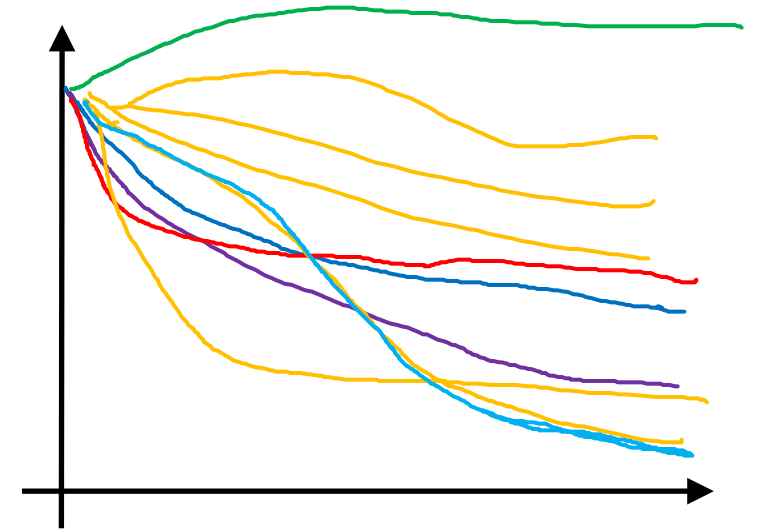
- NLP, Vision, Speech,
Ads, logistics,
- Intuitions do get stale.
Re-evaluate occasionally.

Babysitting one model



Panda ←

Training many models in parallel



Caviar ←

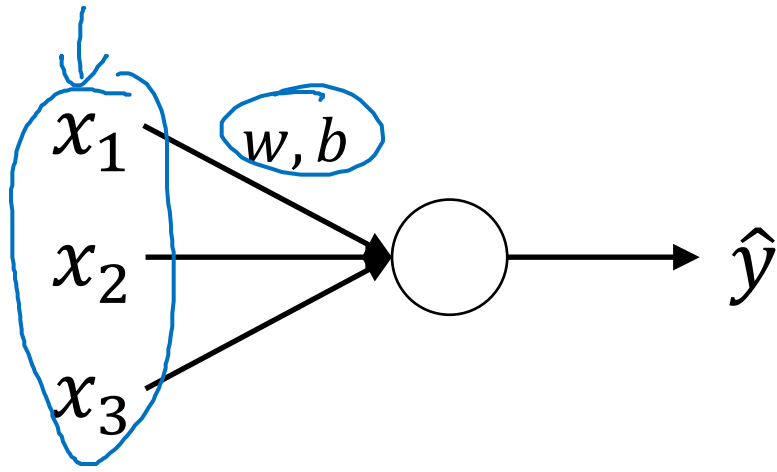


deeplearning.ai

Batch Normalization

Normalizing activations
in a network

Normalizing inputs to speed up learning

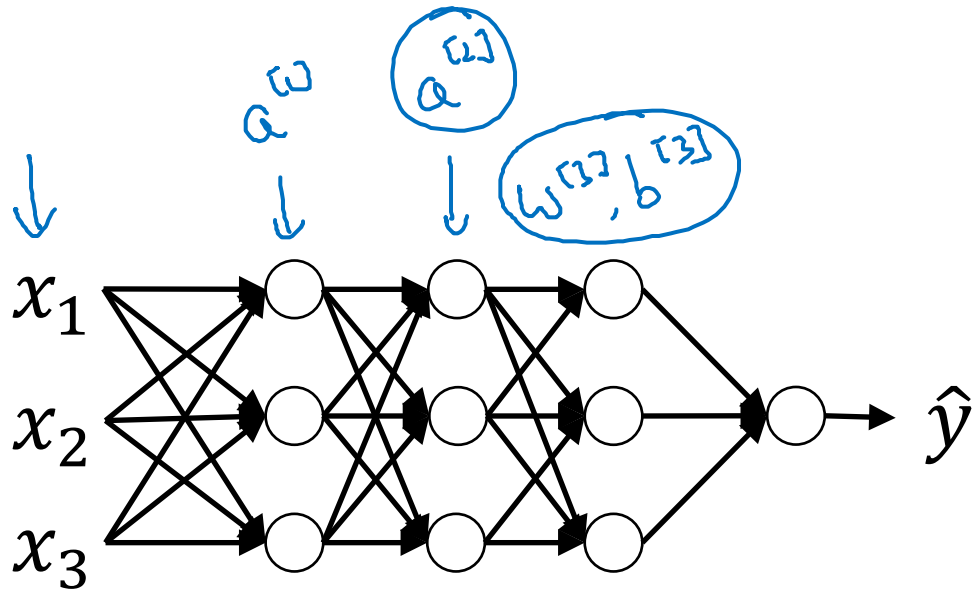
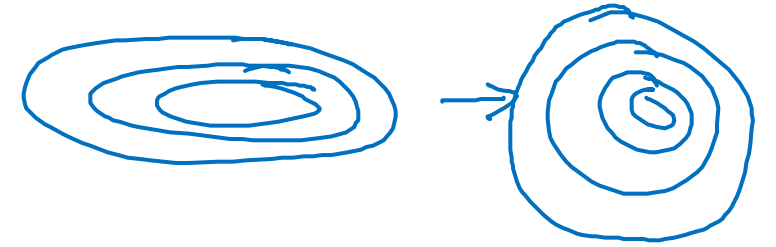


$$\mu = \frac{1}{n} \sum_i x^{(i)}$$

$$X = X - \mu$$

$$\sigma^2 = \frac{1}{n} \sum_i x^{(i)2} \quad \leftarrow \text{element-wise}$$

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



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

Normalize $\frac{z^{[2]}}{\uparrow}$

Implementing Batch Norm

Given some intermediate values in NN

$z^{(1)}, \dots, z^{(m)}$
 $z^{[l]}(i)$

$$\begin{aligned} \mu &= \frac{1}{m} \sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum_i (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \hat{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta \end{aligned}$$

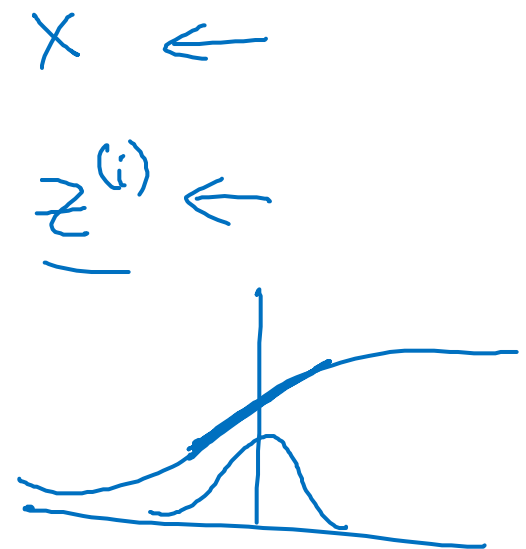
If

$$\gamma = \sqrt{\sigma^2 + \epsilon}$$

$$\beta = \mu$$

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

learnable parameters of model.



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

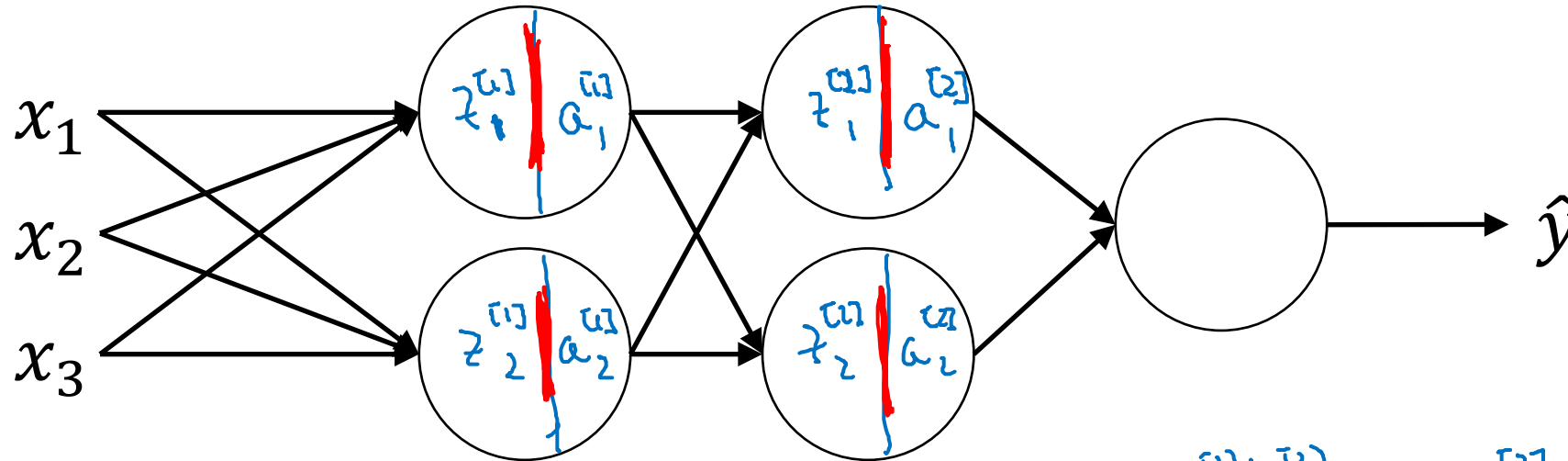


deeplearning.ai

Batch Normalization

Fitting Batch Norm
into a neural network

Adding Batch Norm to a network



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

$a = g(z)$

Parameters: $\left\{ W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots, W^{(L)}, b^{(L)} \right\}$

$\rightarrow \underline{\beta^{(1)}, \gamma^{(1)}, \beta^{(2)}, \gamma^{(2)}, \dots, \beta^{(L)}, \gamma^{(L)}}$

$\rightarrow \underline{\beta}$

$$d\beta^{(L)} \quad \beta = \beta - \alpha d\beta^{(L)}$$

tf.nn.batch-normalization ←

Working with mini-batches

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

$$\boxed{X^{[2]}} \rightarrow \underline{z^{[2]}} \xrightarrow[\text{BN}]{\beta^{[2]}, \gamma^{[2]}} \underline{\tilde{z}^{[2]}} \rightarrow \dots$$

$$X^{[3]} \rightarrow \dots$$

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

\uparrow $(n^{[L]}, 1)$ \uparrow $(n^{[L]}, 1)$ \uparrow $(n^{[L]}, 1)$

\uparrow $z^{[L]}_{(n^{[L]}, 1)}$

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

\uparrow

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

$$z^{[L]}_{\text{norm}}$$

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

Implementing gradient descent

for $t = 1 \dots \text{num Mini Batches}$

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

In each hidden layer, use BN to replace $\underline{z}^{\{t\}}$ with $\underline{\hat{z}}^{\{t\}}$.

Use backprop to compute $\underline{dw}^{\{t\}}$, ~~$\underline{db}^{\{t\}}$~~ , $\underline{dp}^{\{t\}}$, $\underline{df}^{\{t\}}$

Update params
$$\left. \begin{aligned} w^{\{t\}} &:= w^{\{t-1\}} - \alpha dw^{\{t\}} \\ \beta^{\{t\}} &:= \beta^{\{t-1\}} - \alpha dp^{\{t\}} \\ f^{\{t\}} &:= \dots \end{aligned} \right\} \leftarrow$$

Works w/ momentum, RMSprop, Adam.

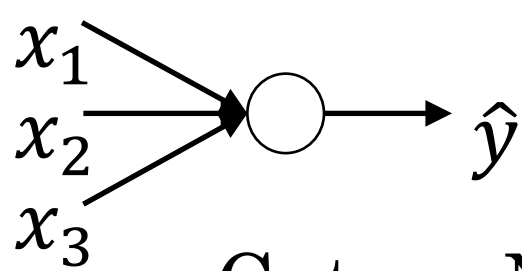


deeplearning.ai

Batch Normalization

Why does
Batch Norm work?

Learning on shifting input distribution

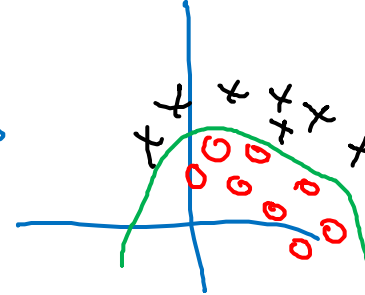
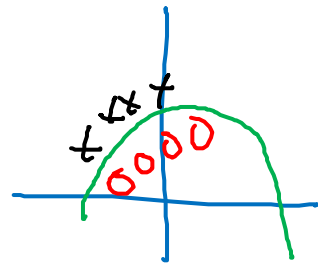


Cat

Non-Cat

$y = 1$ ✓

$y = 0$



$y = 1$ ✓

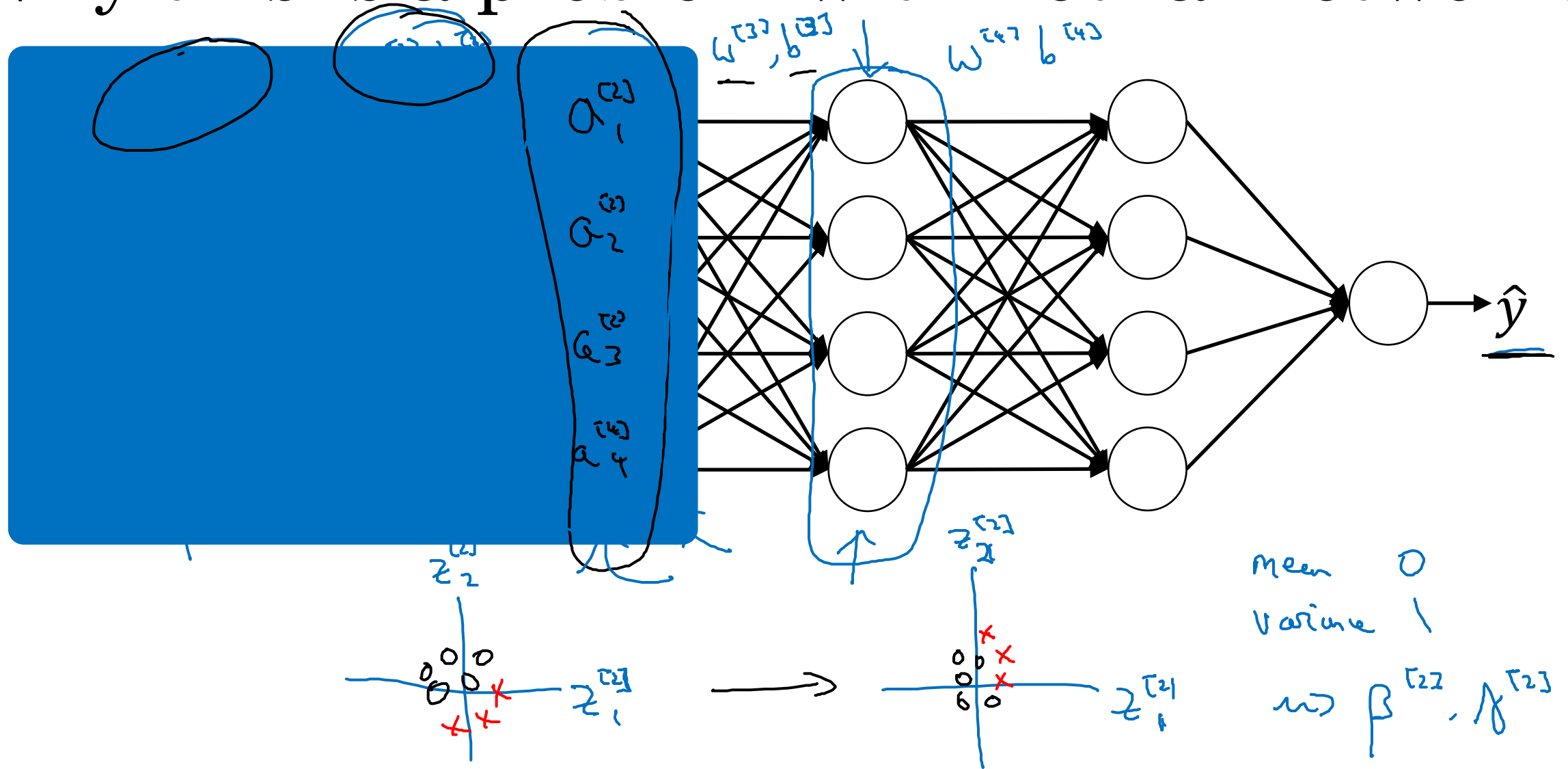
$y = 0$



"Covariate shift"

$\underline{x} \rightarrow y$

Why this is a problem with neural networks?



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.

mini-batch : 64 \longrightarrow 512



deeplearning.ai

Multi-class classification

Softmax regression

Recognizing cats, dogs, and baby chicks



3



1



2



0



3



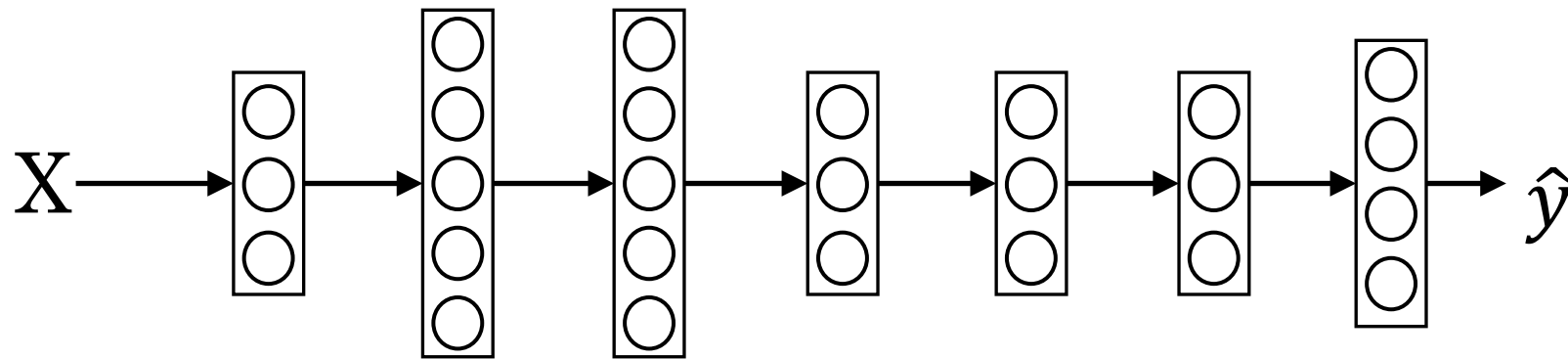
2



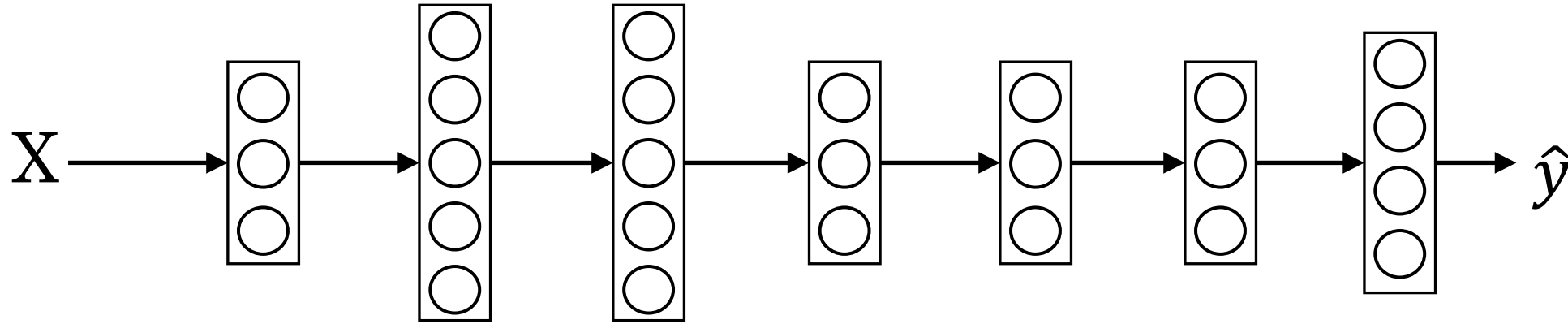
0



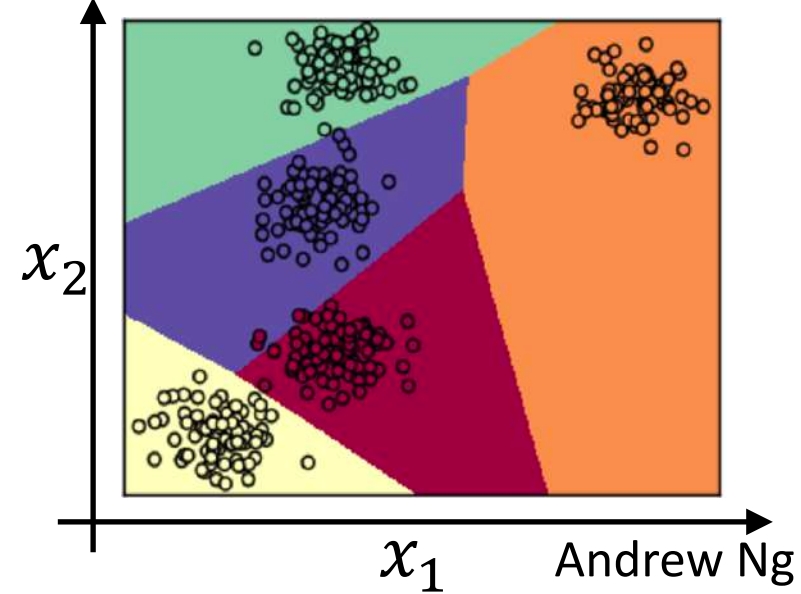
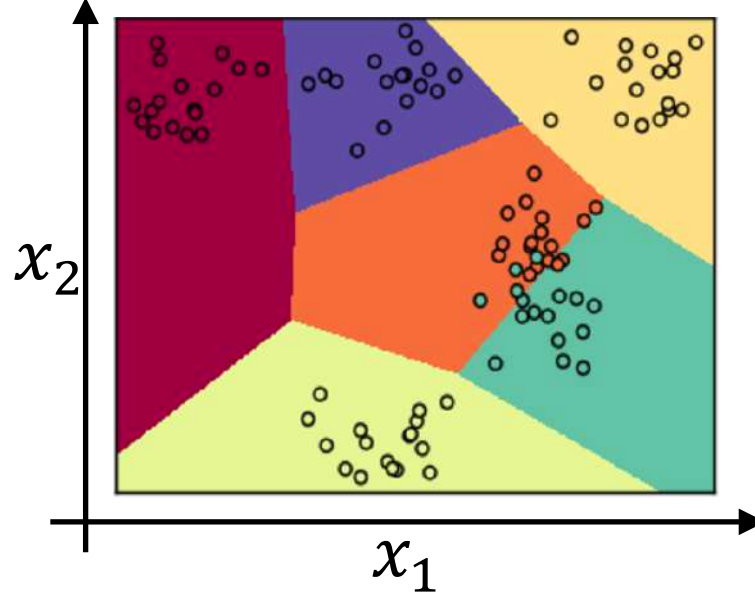
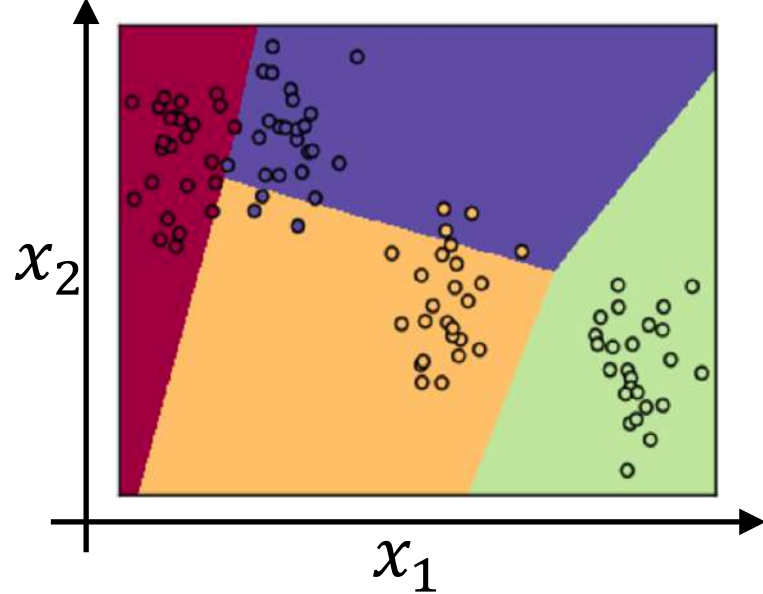
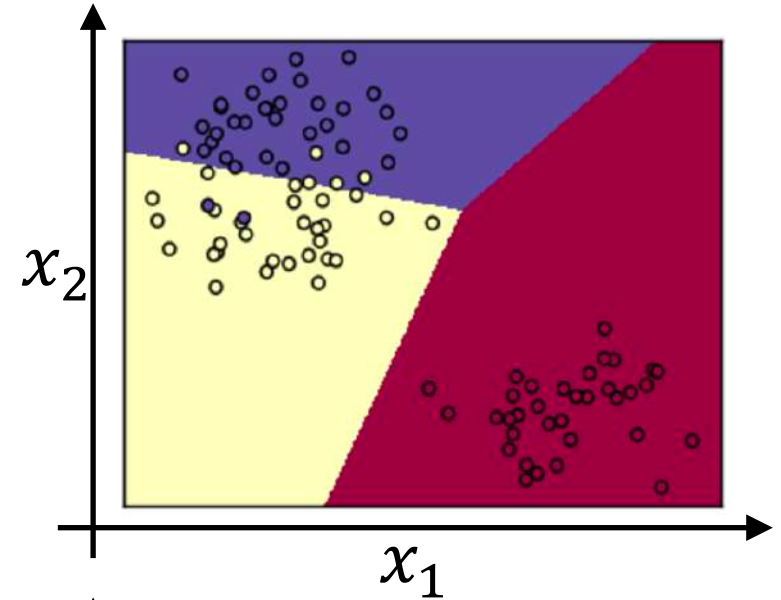
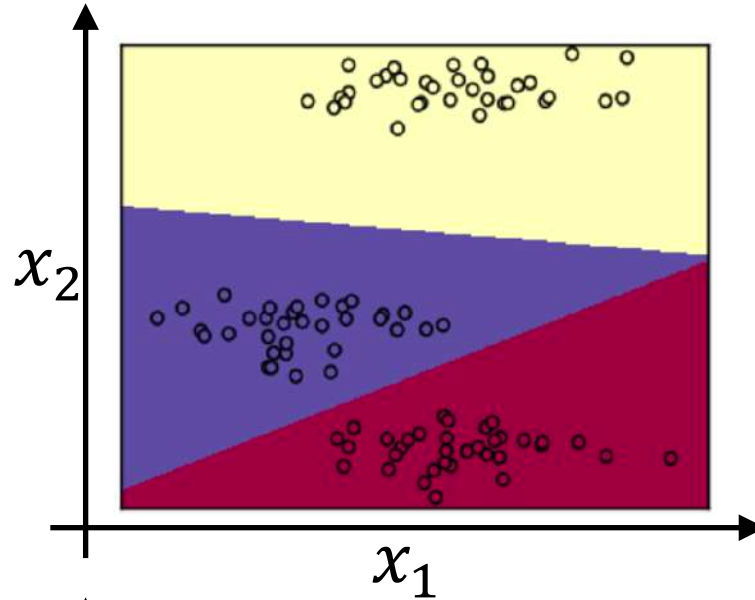
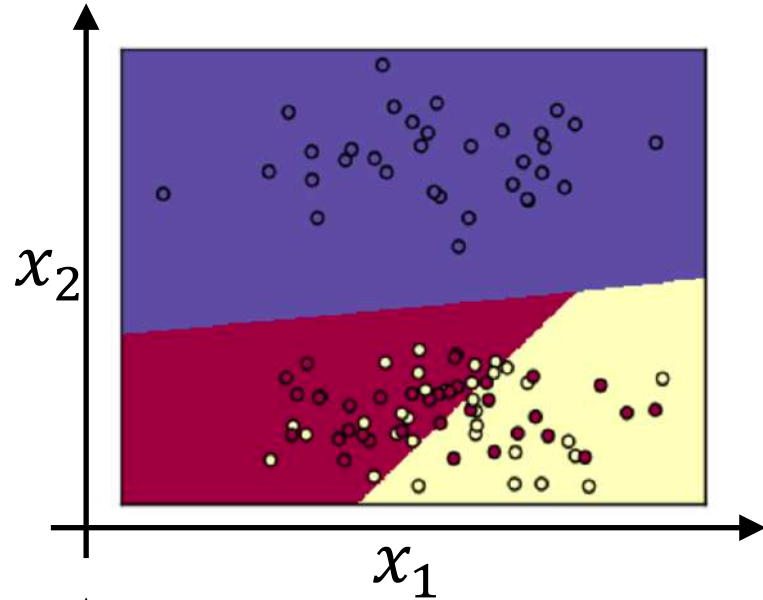
1



Softmax layer



Softmax examples





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

$$\begin{aligned} J(w) &= \boxed{w^2 - 10w + 25} \\ &\quad \swarrow \\ &\quad (w-5)^2 \\ &\quad w=5 \end{aligned}$$

$$\begin{aligned} J(w, b) \\ \uparrow \quad \uparrow \end{aligned}$$

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))
```

