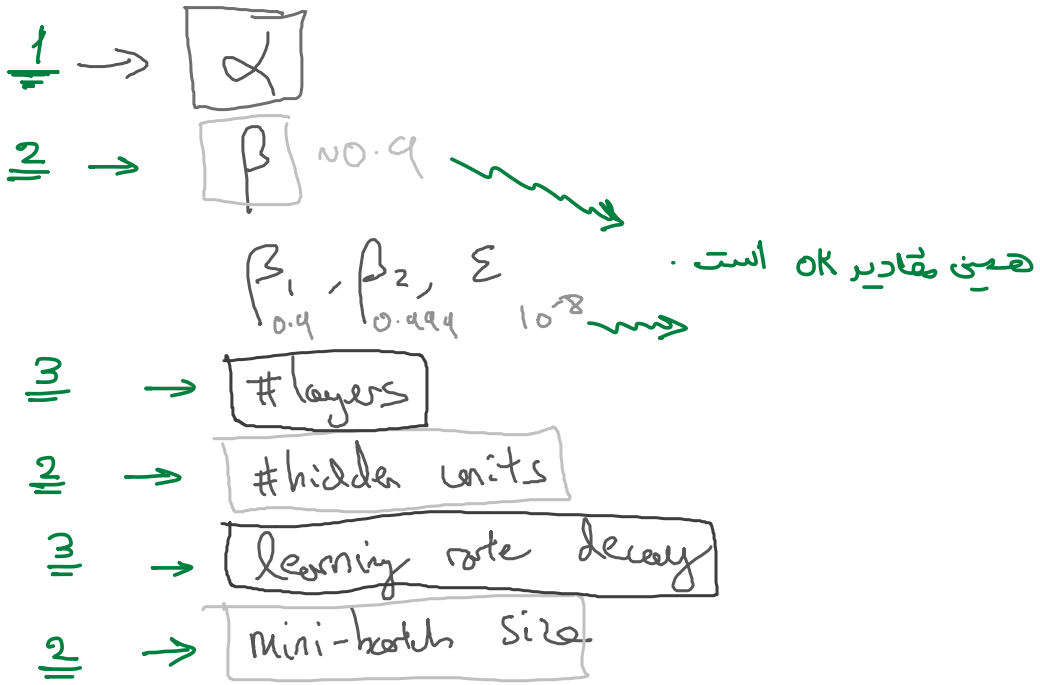
deeplearning.ai

Hyperparameter tuning

Tuning process

Hyperparameters

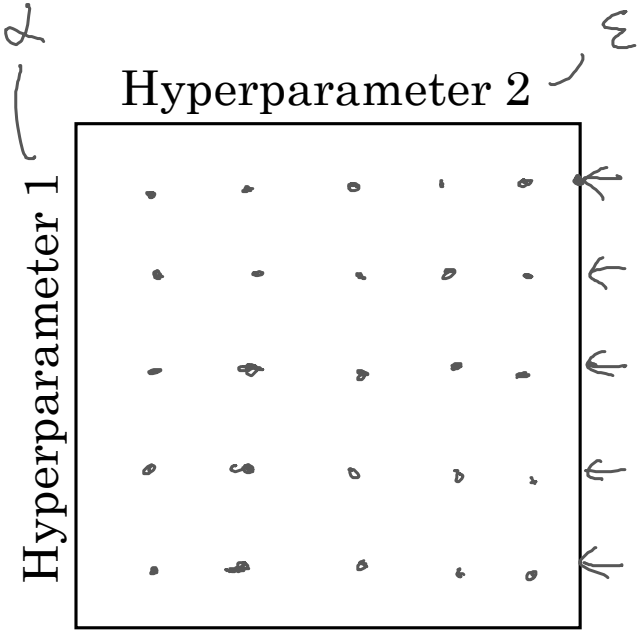


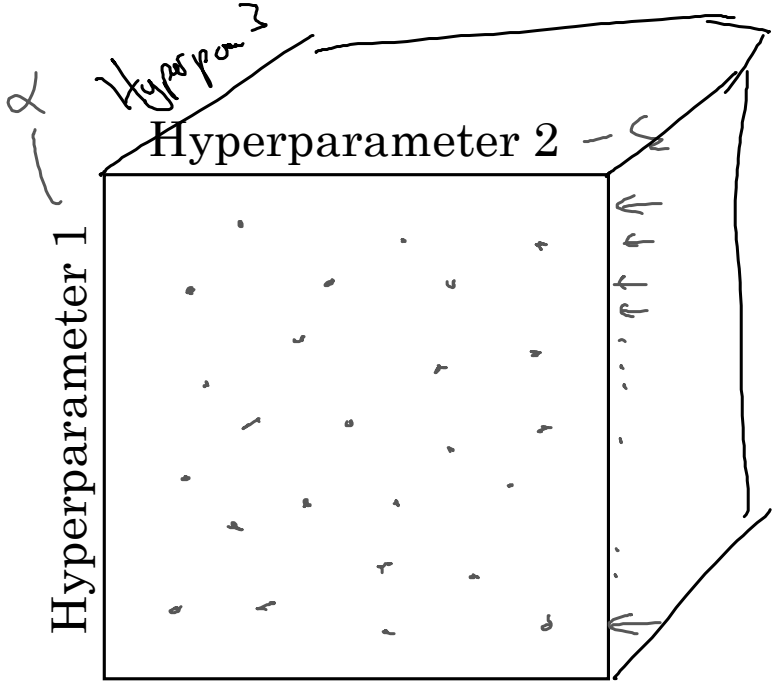
importance

When working with Big Data to tune the hyperparameters

⇒ (not neccessarily uniform) random search.

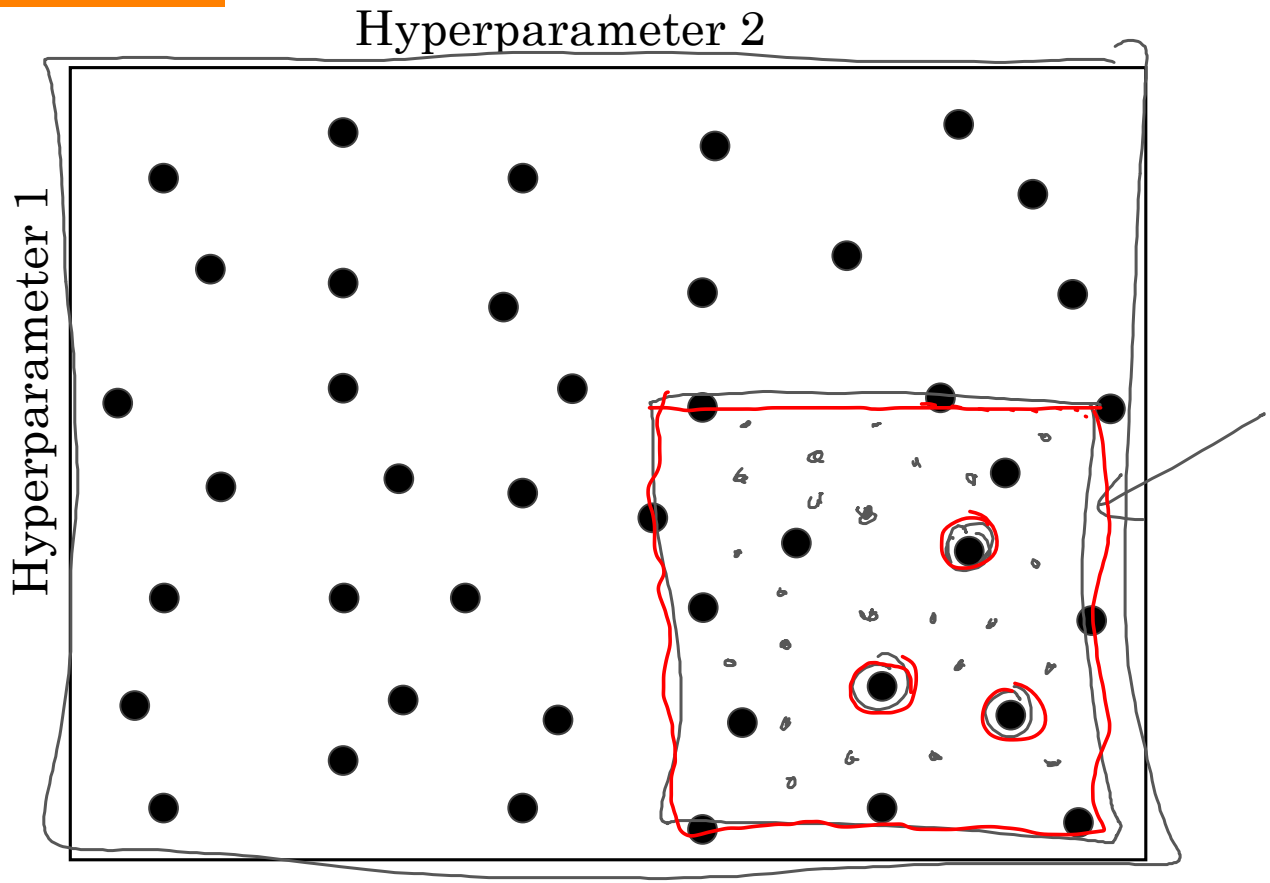
Try random values: Don't use a grid





ایده: اگر سه نقطه خوب پیدا کردی، ادامه search را در حوالی همان انجام بده!

Coarse to fine



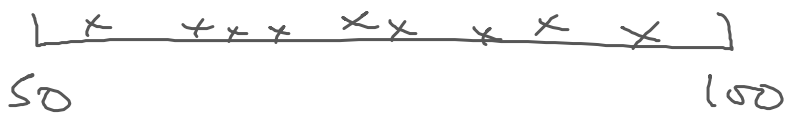
deeplearning.ai

Hyperparameter tuning

Using an appropriate scale to pick hyperparameters

Picking hyperparameters at random

→ $n^{[l]} = 50, \ldots, 100$



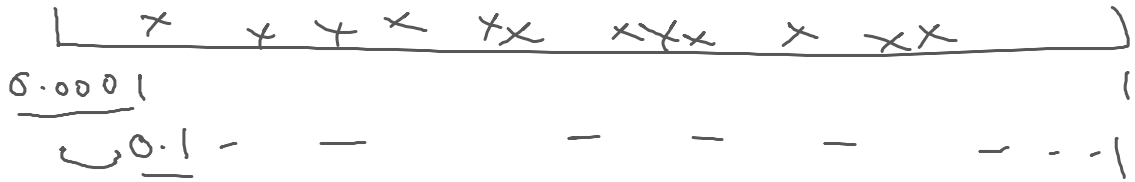
→ #layers L: 2 − 4

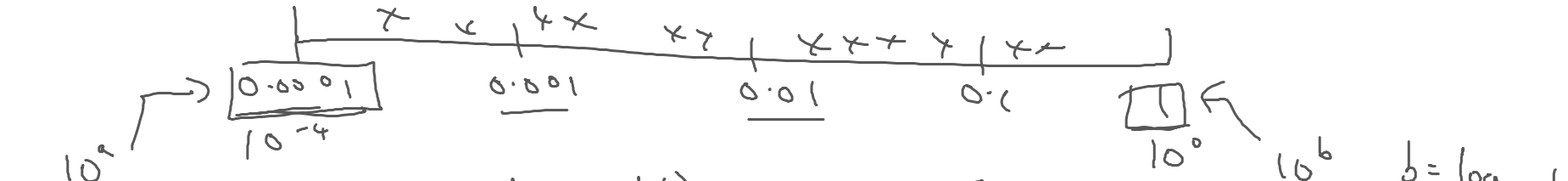
2, 3, 4

استفاده از scale مناسب برای random search ضروری است. آن حدوده مجاز $\alpha \in [10^{-3}, 1]$ باشد نباید uniformlysearch کرد بلکه از تبدیل آن که $r = \log \alpha \in [-4, 0]$ است به صورت uniform search کنیم.

Appropriate scale for hyperparameters

$\alpha = 0.0001 \ldots, 1$





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

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

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

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

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

تبدیل مناسب برای جا زدن β به صورت uniform search روی بازه $\log(1-\beta)$ است.

Hyperparameters for exponentially weighted averages

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

$10 \qquad\qquad 1000$

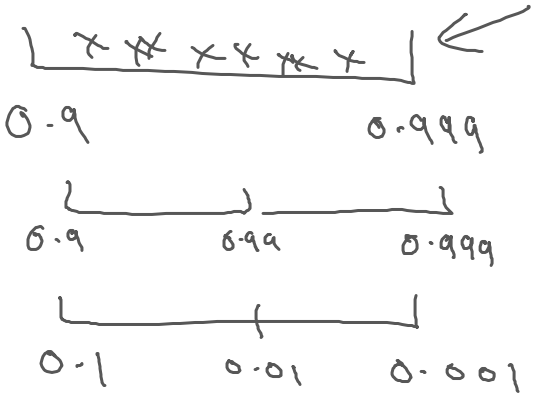
$1-\beta = 0.1 \quad \dots \quad 0.001$

$\beta: 0.900 \rightarrow 0.9005$ } ~ 10

$\beta: 0.999 \rightarrow 0.9995$

$\sim 1000 \qquad \sim 2000$

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



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

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

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

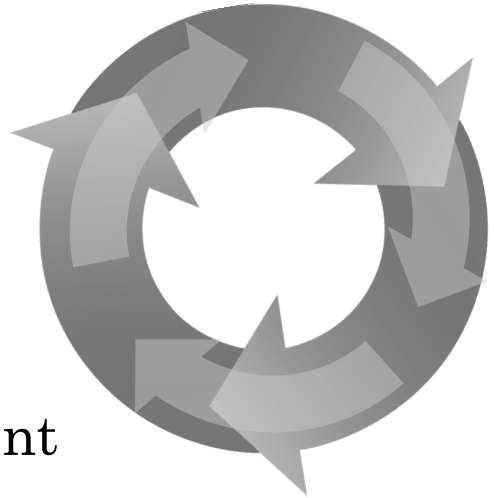
$\beta = 1 - 10^{r}$

deeplearning.ai

Hyperparameters tuning in practice: Pandas vs. Caviar

Re-test hyperparameters occasionally

Idea



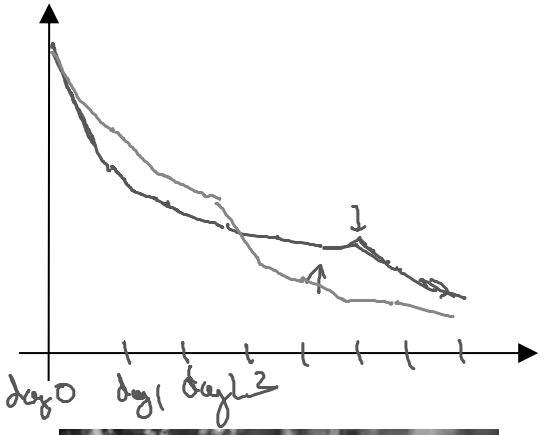
Experiment

Code

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

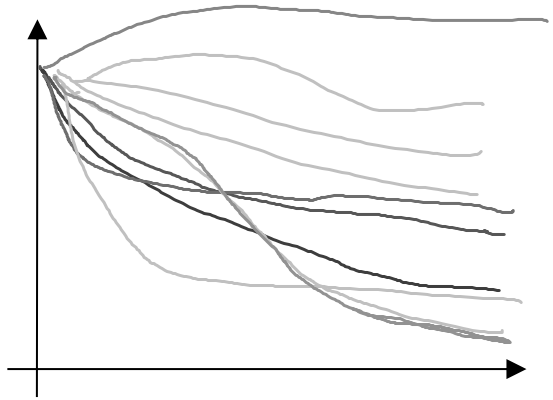
- Intuitions do get stale. Re-evaluate occasionally.

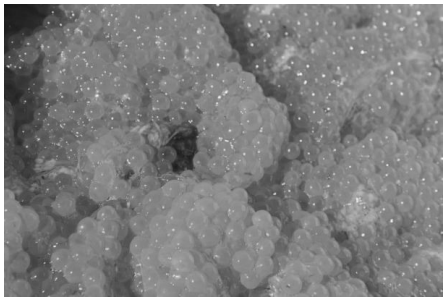
Babysitting one model



Panda

Training many models in parallel





Caviar

deeplearning.ai

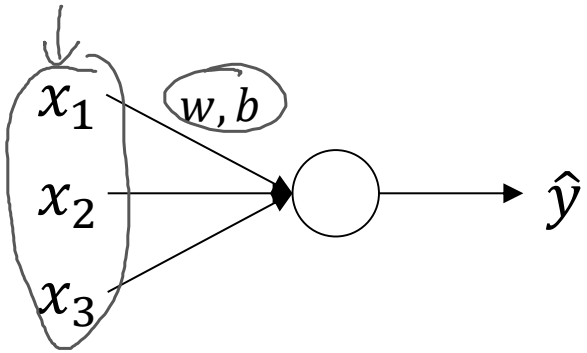
Batch Normalization

Normalizing activations in a network

Batch normalization makes your hyperparameter search easier and makes your NN more robust. The choice of hyperparameters is a bigger range of hyperparameters that work well, and enable you to easily train deep networks.

همان طور که normalize کردن x_i های ورودی باعث بهبود سرعت می‌شود، نرمالیزه کردن $z^{[l]}$ (یا $a^{[l]}$) در Hidden layers بسیار مفید است.

Normalizing inputs to speed up learning

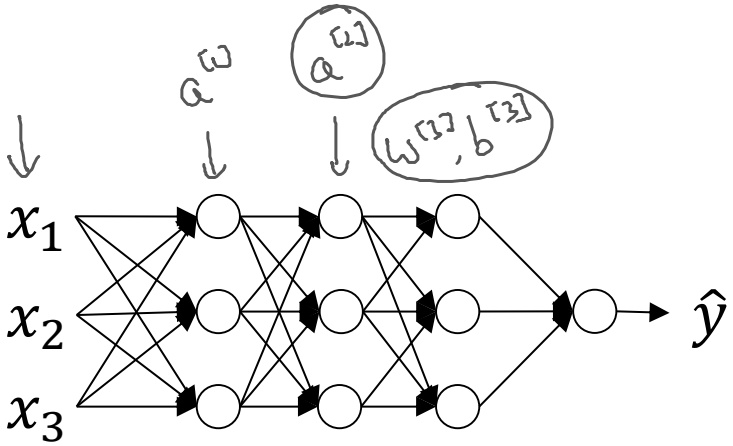


$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$X = X - \mu \quad \text{element-wise}$$

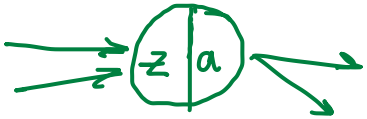
$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)2}$$

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



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

Normalize $z^{[2]}$



Implementing Batch Norm

Given some intermediate values in NN

$z^{(1)}, \dots, z^{(m)}$ ⇒ همگی باید مربوط به یک لایه باشند

$z^{[l](i)}$

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

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

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

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

learnable parameters of model.

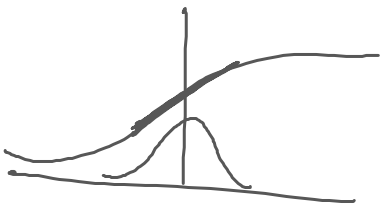
If $\gamma = \sqrt{\sigma^2 + \varepsilon}$ ←

$\beta = \mu$ ←

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

$X \leftarrow$

$z^{(i)} \leftarrow$



با آدم حسابی میتوان مقادیر γ و β را انتخاب کرد ⇐ Hyperparameter

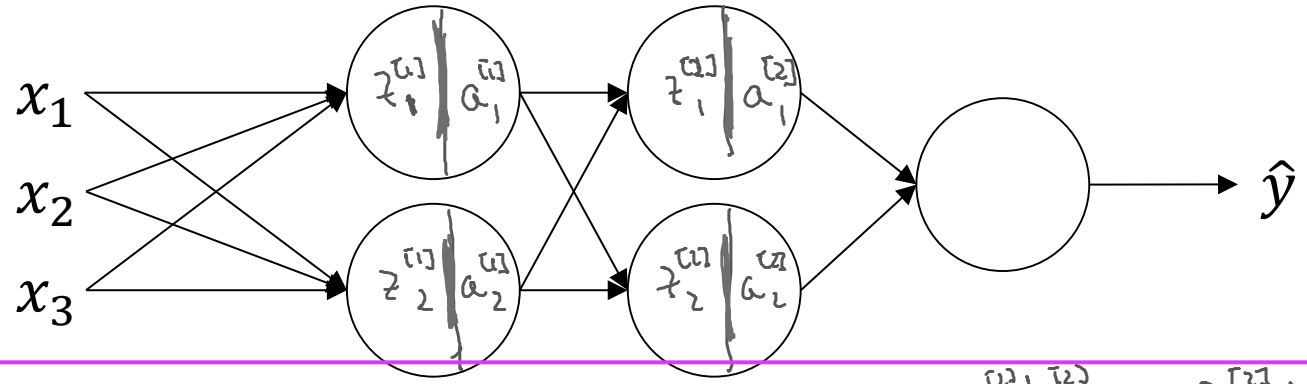
Use $\tilde{z}^{[l](i)}$ instd of $z^{[l](i)}$.

deeplearning.ai

Batch Normalization

Fitting Batch Norm into a neural network

Adding Batch Norm to a network



Batch Norm in DL

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

$w^{[1]}, b^{[1]}$ — $w^{[2]}, b^{[2]}$

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}$

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

$\rightarrow \beta$

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

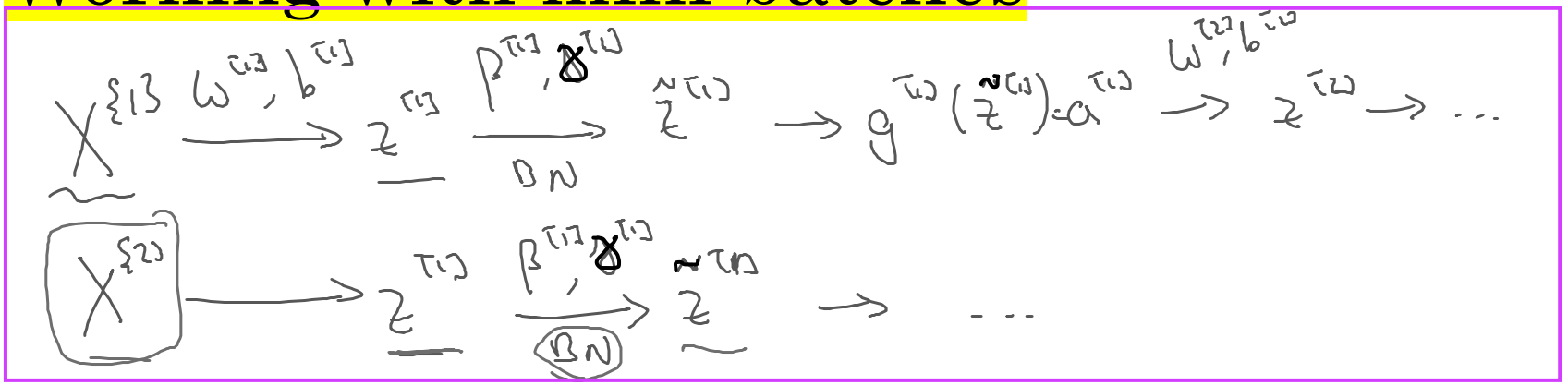
tf.nn.batch_normalization ⇐

↖ tf در عمل نیازی به کد جالا نیست

Remark: Here, β_{BN} is not β_{Adam} ⇒ just bad notation

In practice, BN should be combined with mini batches

Working with mini-batches



Parameters: $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$.

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

dimension

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

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

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

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

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

وقتی BN داریم می‌توانیم پارامتر $b^{[l]} = 0$ در نظر بگیریم چون BN میانگین و واریانس را تنظیم می‌کند و اثری را از بین می‌برد.

Final Implementation

Implementing gradient descent

for $t = 1 \dots$ 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 $w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$

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

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

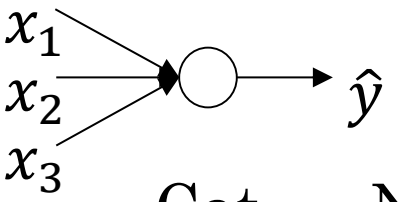
Also works for momentum, RMSprop, Adam. $\Longleftrightarrow$ کافی است این بخشی را جایگزین کنیم updating

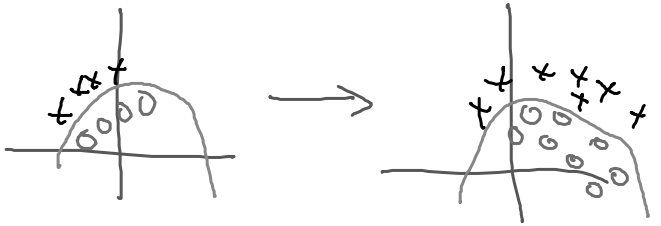
deeplearning.ai

Batch Normalization

Why does Batch Norm work?

Learning on shifting input distribution





Cat $y = 1$ | Non-Cat $y = 0$

Black Cats

"Covariate shift"

$X \rightarrow y$

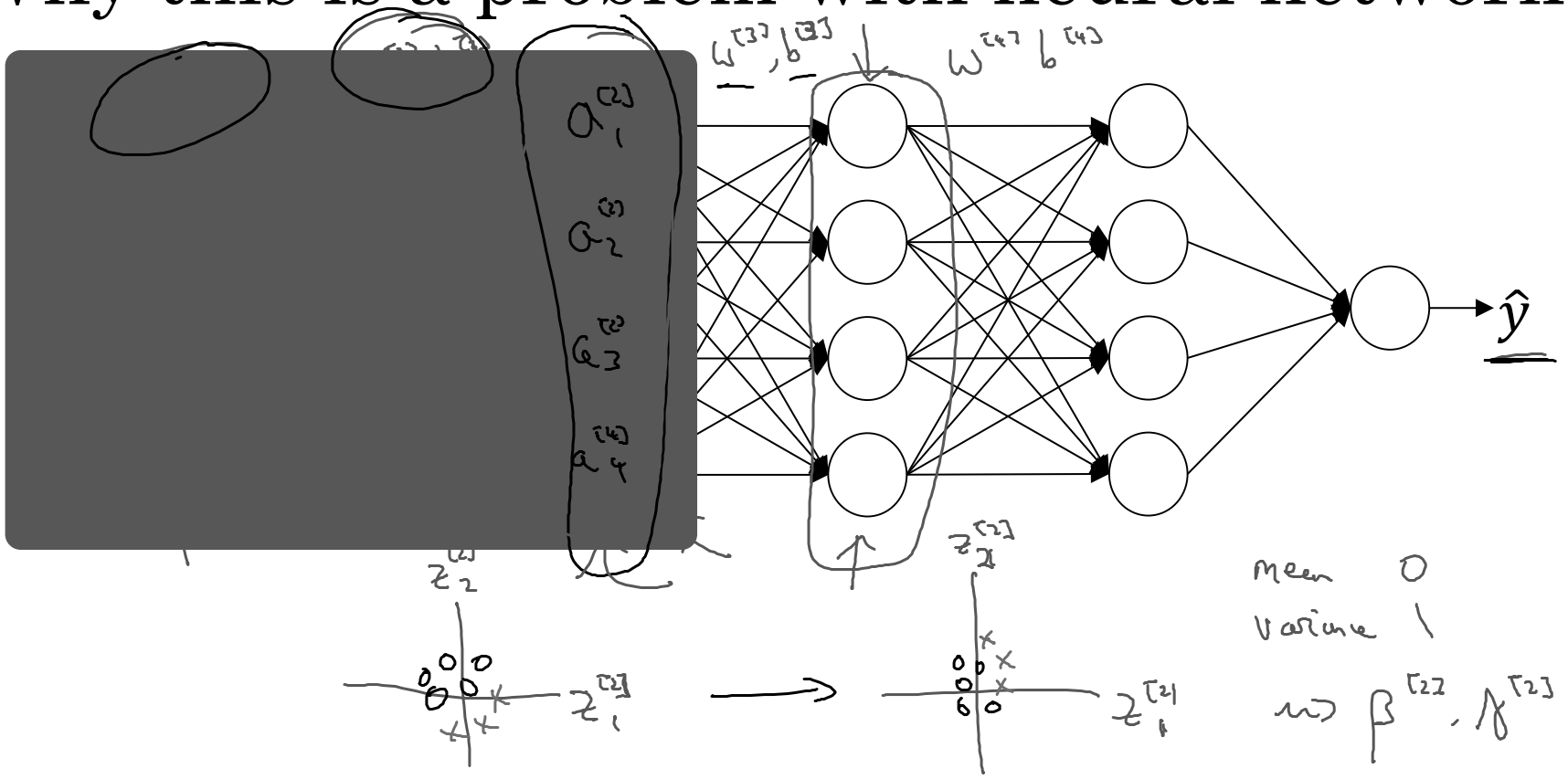
$y = 1$ | $y = 0$

Coloured Cats

آلگوریتم classifier برای گربه‌های سیاه بسازیم، الزاماً برای گربه‌های رنگی خوب عمل نمی‌کند چون covariate shift داشته ایم.

BN limits the amount to which updating the parameters in the earlier layers can affect the distribution of values that the third layer sees. Hence, it reduces the problem of the input values changing: it makes these values more stable so that the later layers have more firm ground to stand on.

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 ⟶ 512

deeplearning.ai

Batch Normalization

Batch Norm at test time

BN handles data one mini-batch at a time: it computes mean and variances on each mini-batch.

At test time, you might not have a mini-batch of examples: you might be processing one single example at the time. So, you need to do something different at test time.

Standard BN (for training)

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).

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

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

θ_1 θ_2 θ_3 σ^2

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

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

training minibatches های $\sigma^{2\{i\}}$, $\mu^{\{i\}}$ exponentially weighted ave روی σ^2 و μ جدید از روی BN (for test) با کمک

بدست آورد و سپس حین test از این μ و σ^2 استفاده کرد.

deeplearning.ai

Multi-class classification

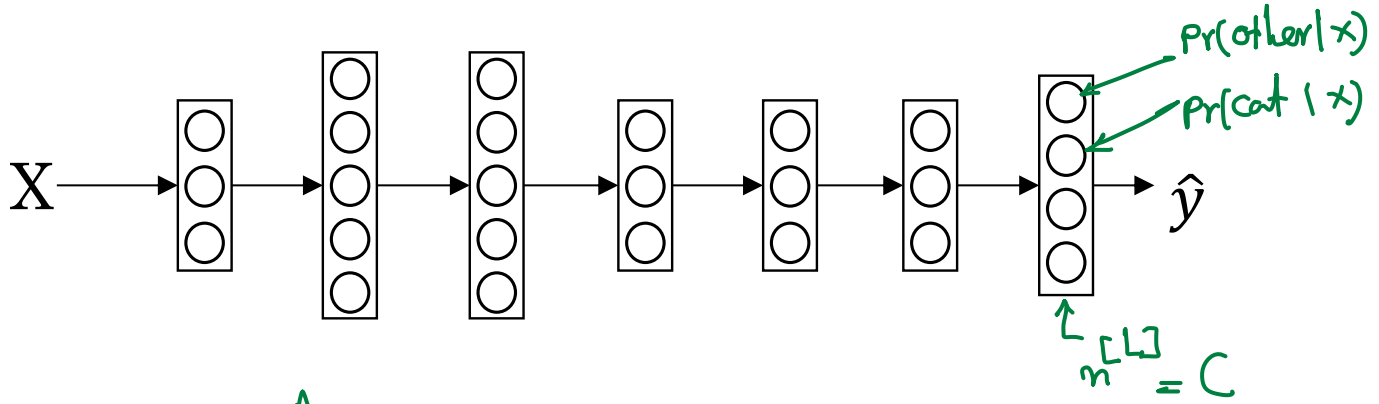
Softmax regression

Four classes $\{0, 1, 2, 3\}$ (0 = other)

Recognizing cats, dogs, and baby chicks

3 1 2 0 3 2 0 1

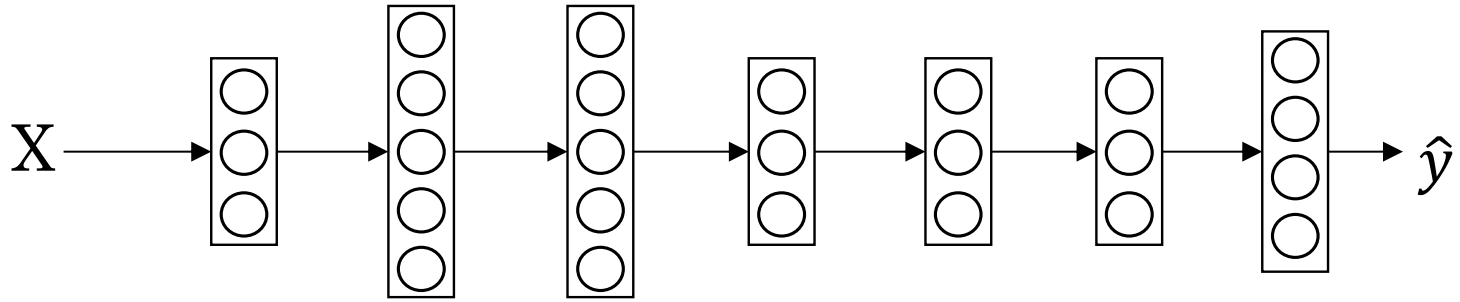
C = # classes = 4 $(0, \ldots, 3)$



$\hat{y}$: $(C, 1)$

$$a^{[L]} = \frac{e^{z^{L}}}{\sum_{i=1}^{4} t_i}$$

Softmax layer



$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$$

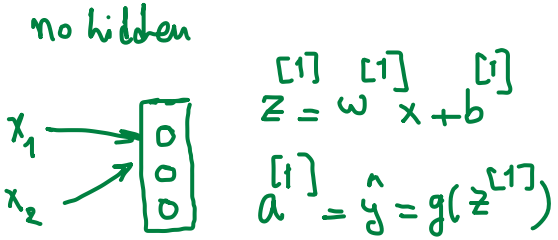
Activation Function

$$a^{[L]}_i = \frac{\exp(z^{[L]}_i)}{\sum_{i=1}^{4} \exp(z^{[L]}_i)} = \frac{t_i}{\sum_{i=1}^{4} t_i} \quad \text{where} \quad t_i = \exp(z^{[L]}_i)$$

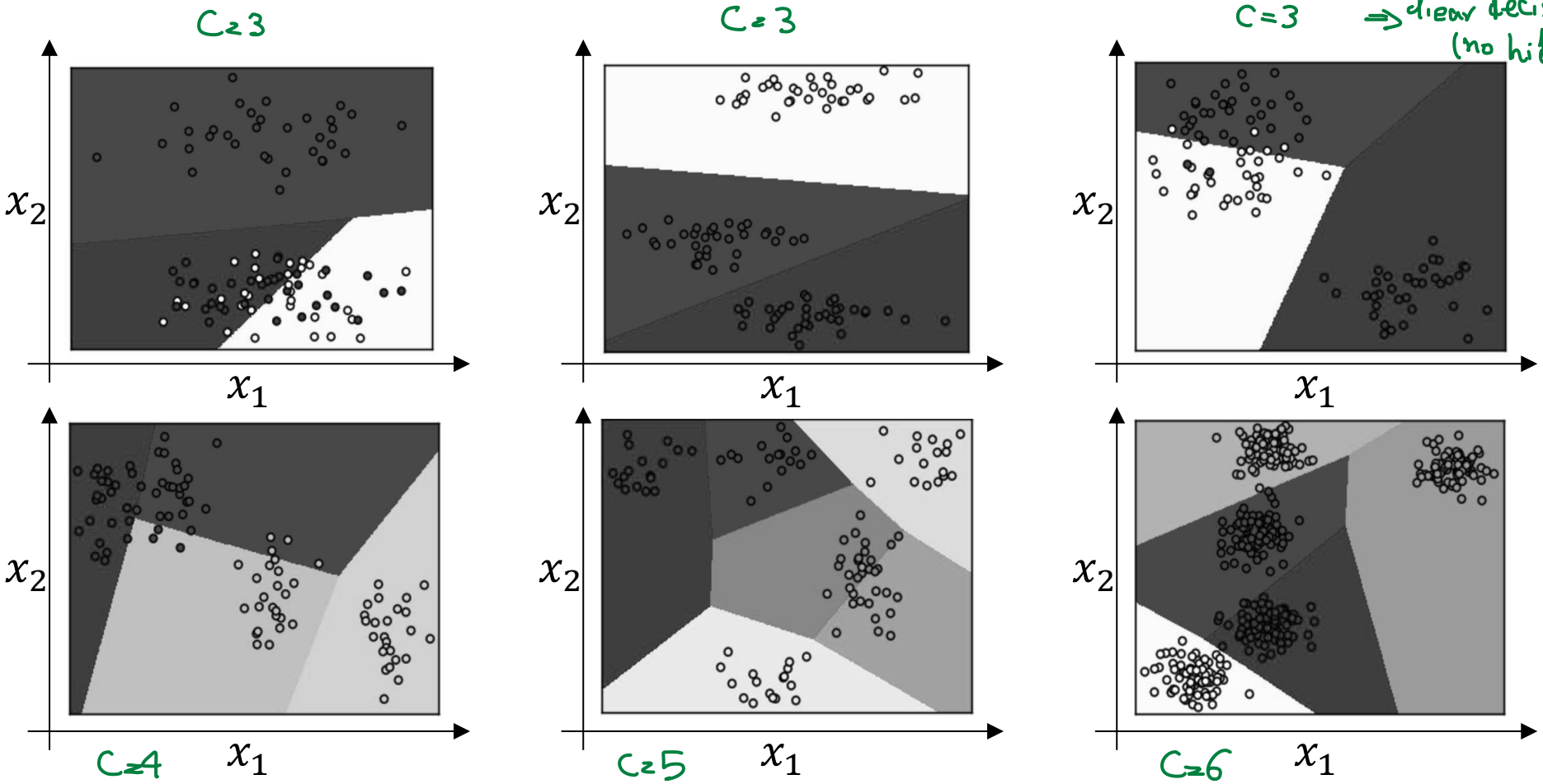
$$a^{[L]} = g^{[L]}(z^{[L]})$$

$(4,1)$ $(4,1)$

softmax activation function



Softmax examples

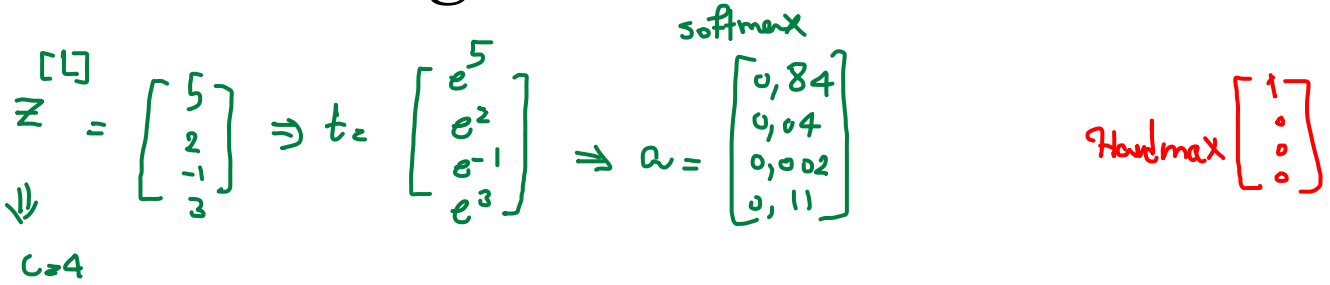


deeplearning.ai

Multi-class classification

Trying a softmax classifier

Understanding softmax



Softmax regression generalizes logistic regression to C classes. ⇒ If C=2, softmax=Logistic

Loss function

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j \log \hat{y}_j \quad \text{and} \quad J(w^{[l]}, b^{[l]}, \dots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

With DL frameworks, you just need to focus on the forward prop: the primary will figure out how to do back prop.

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)

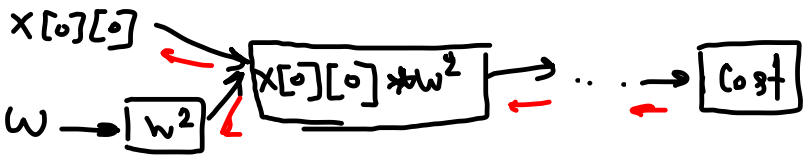
deeplearning.ai

Programming Frameworks

TensorFlow

Code example

$J(w) = w^2 - 10w + 25 \Rightarrow w_{min} = 5$




```
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))
```

equiv

```
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))
```