Deep Learning

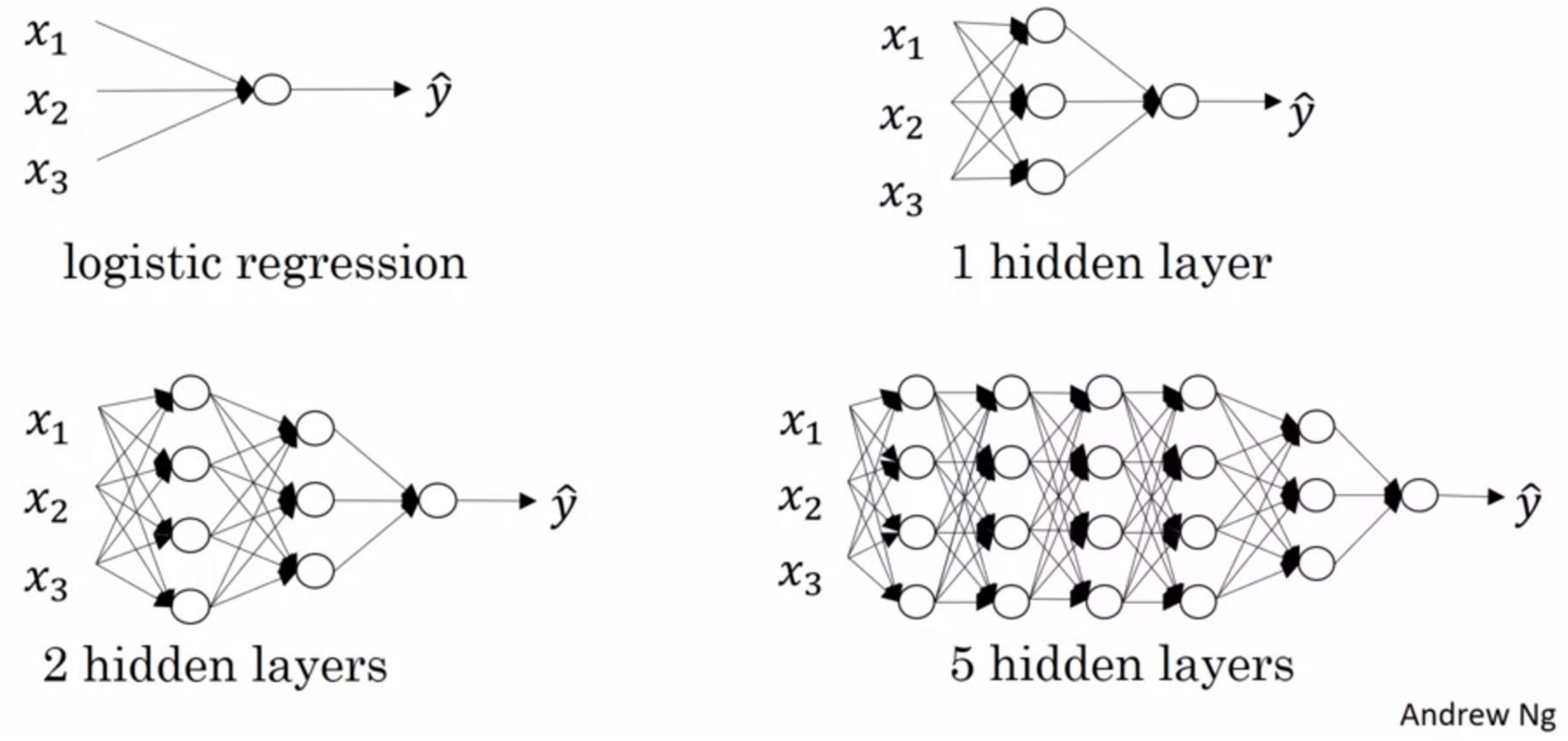
Big Data & Machine Learning Bootcamp - Keep Coding

Outline

1. Deep Neural Network
2. Why deep representations?
3. Parameters and hyperparameters
4. Train - Development and Test sets
5. Regularization
6. Normalizing Inputs
7. Vanishing/Exploding gradients

1. Deep Neural Network

"Shallow network"

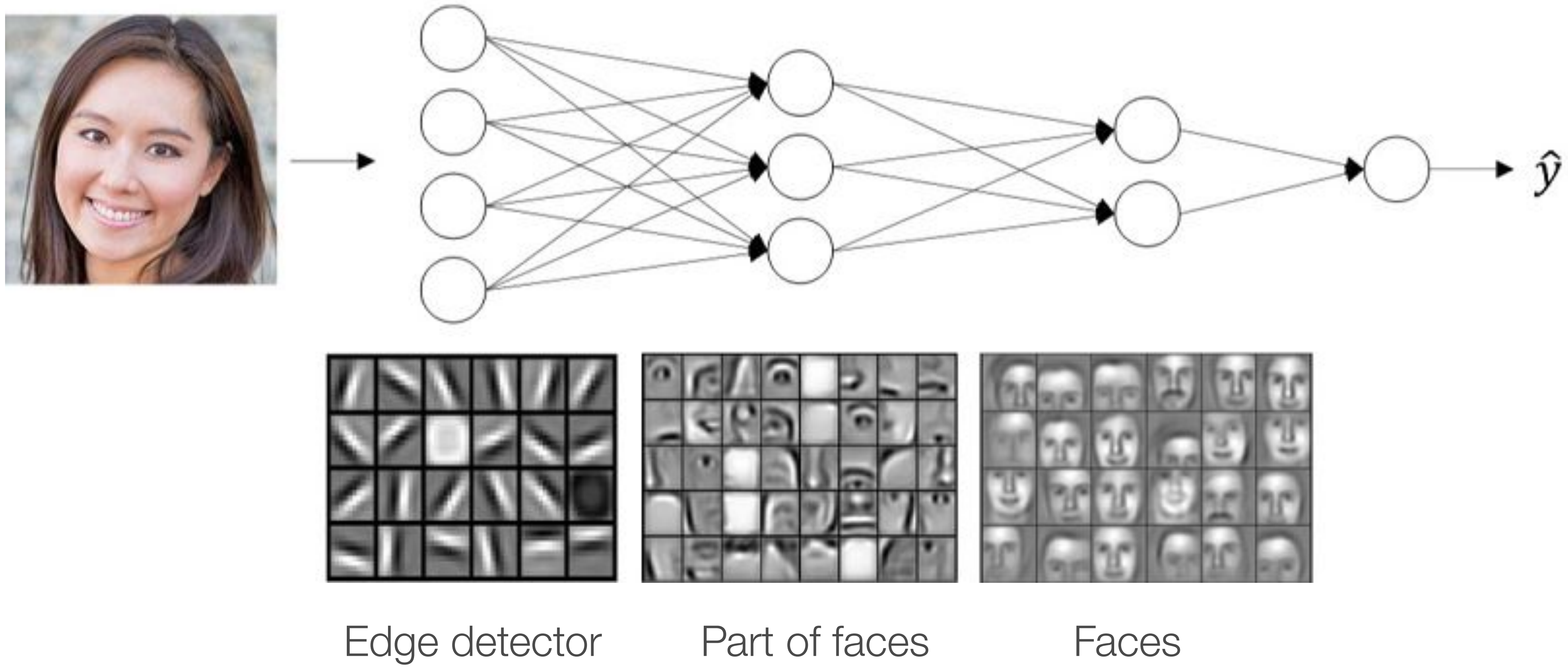


We don't count the input and the output layers. We only count the hidden layers!

Sources:
- Coursera

2. Why deep representations?

The first layers detect low level features while deep layers detect more complex features



Sources:
- Coursera

2. Why deep representations?

Informally: There are functions you can compute with a “small” L-Layer deep neural network that shallower networks require exponentially more hidden units to compare.

A shallow neural network requires **exponentially more neurons** than a simple deep neural network to approximate a function!

AND also because using **deep learning capture the popular imagination**! But seriously, they work really well! :)

Sources:
- Coursera

3. Parameters and hyperparameters

The parameters are the weights and the biases, right?

So what are the hyperparameters? The values that determine the real parameters

- Learning rate α

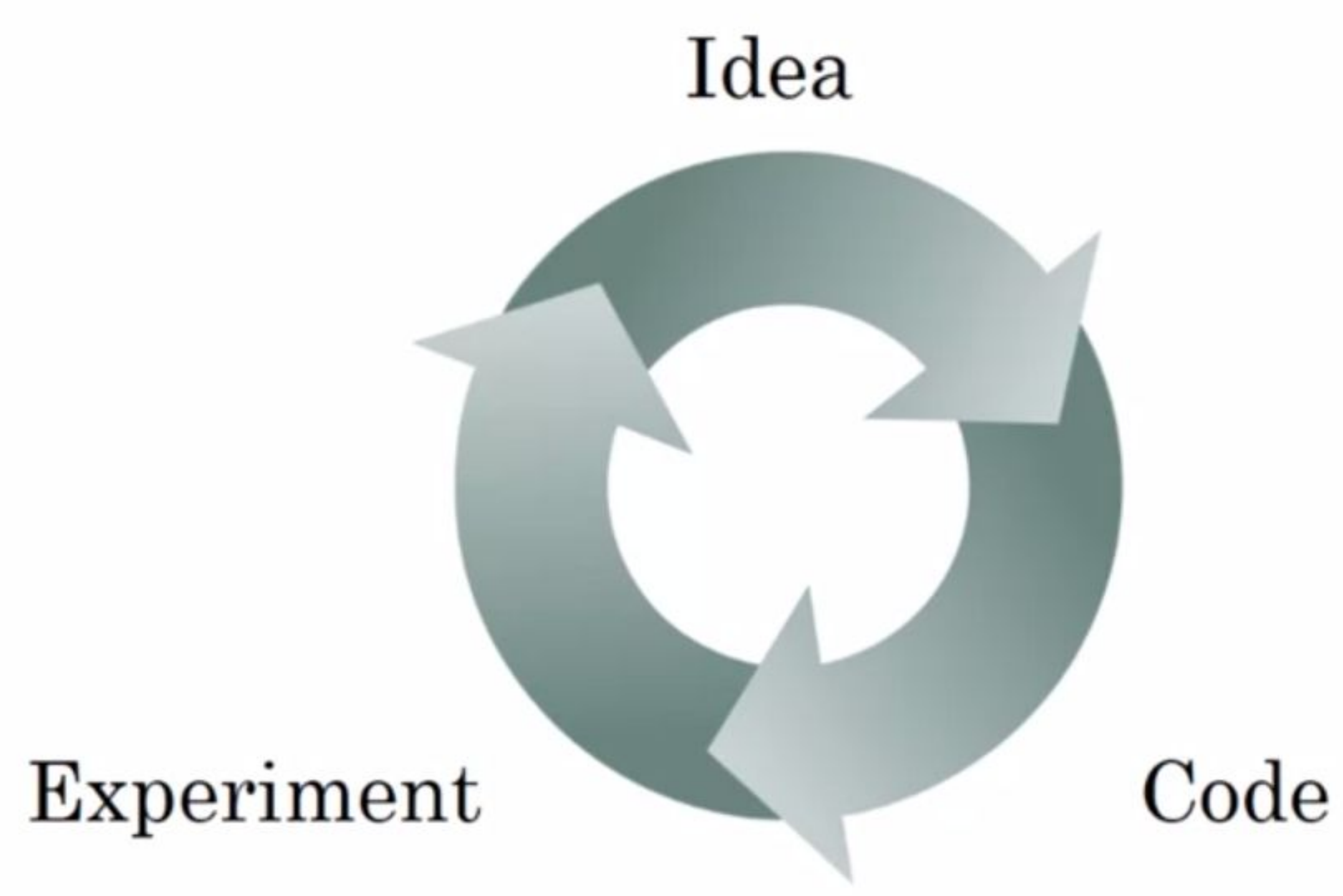
$$\boxed{w = w - \alpha\,\delta w}$$

- Number of iterations for the gradient descent
- Number of hidden layers
- Choice of activation function
- Later we'll also see momentum, minibatch size and regularization

Sources:
- Coursera

3. Parameters and hyperparameters

Applied Deep learning is a very empirical process



Empirical process is a fancy way of saying trial and error.

You choose certain values for learning rate, minibatch etc and try them

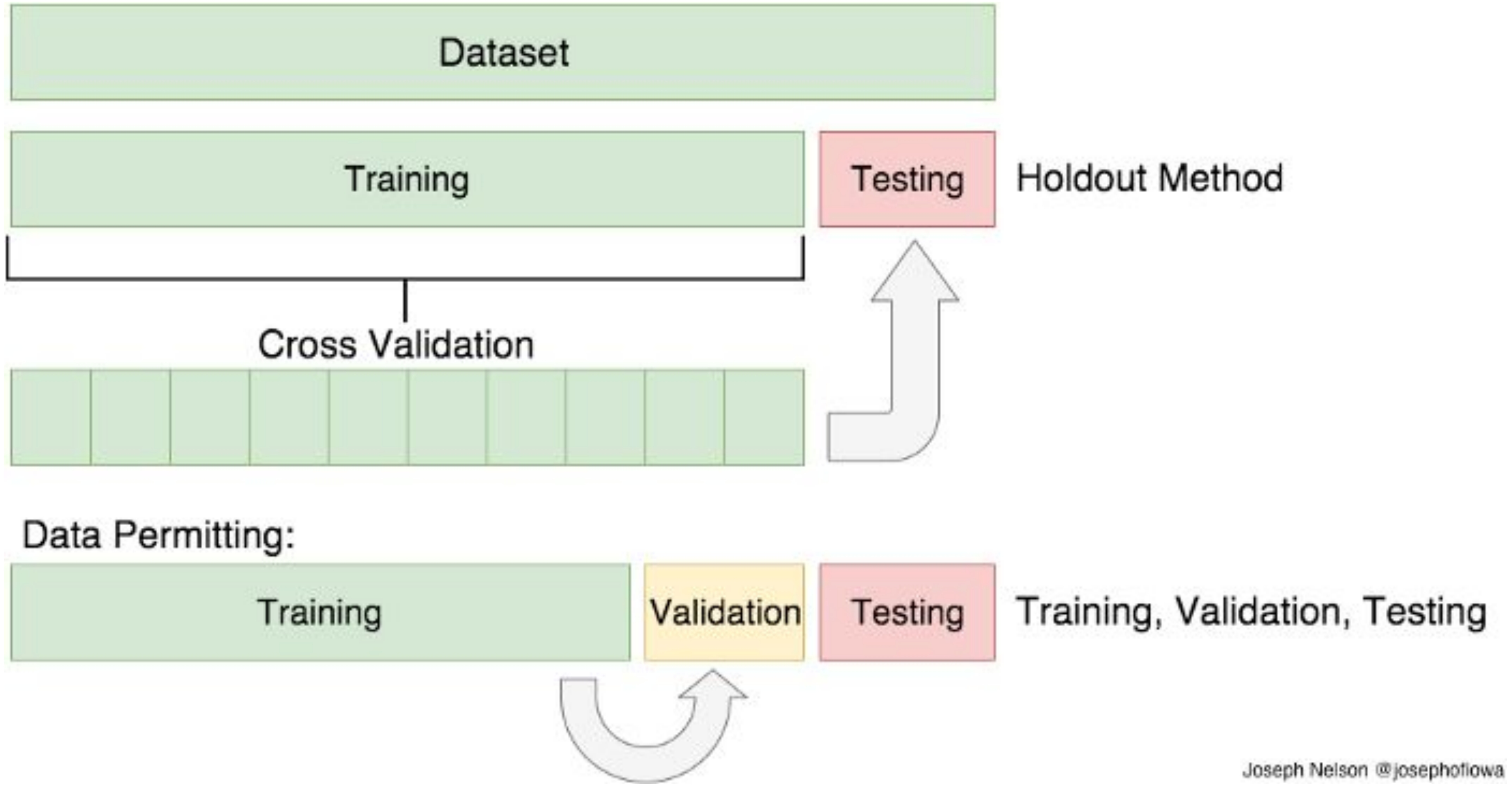
4. Train - Development and Test sets

Making good choices in how you set up your training, development, and test sets can make a huge difference in helping you quickly find a good high performance neural network.

In addition to choosing learning rate, number of hidden layer, activations functions, etc

4. Train - Development and Test sets

BUT this always depends on the application! Some application don't need many testing images while for others testing is key



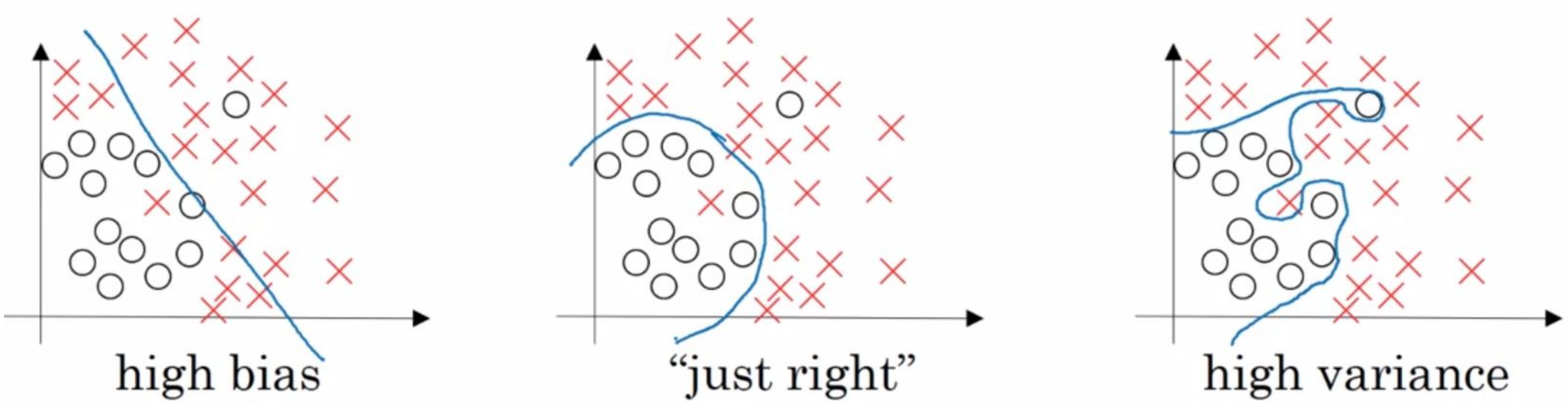
Make sure the validation and test set come from the distribution or sensor

Sources:
- Coursera
- https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

4. Train - Development and Test sets

Sometimes it is OK to not have test set (Only development set) as long as you controlled the bias and the variance or the **bias/variance trade-off**

But what's bias and variance? Let's see this in a 2D example



Underfitting ***Overfitting***

Sources:
- Coursera

4. Train - Development and Test sets

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	High variance (Overfitting)	High bias (Underfitting)	High bias & high variance	Low bias and low variance

This is assuming that the human error or the optimal error is 0%

We'll discuss this later. But this doesn't occur in real life. ***The optimal error is around 15%***

Sources:
- Coursera

4. Train - Development and Test sets

To deal with with bias or high variance there is a **“recipe” for deep learning**

- **Does the network have high bias?** (training data performance)
  Solution: Bigger network, train longer or neural architecture search

- **Does the network have high variance?** (dev set performance)
  Solution: More data, regularization, or neural architecture search (more appropriate network)

Sources:
- Coursera

5. Regularization

If you suspected that the system is **overfitting the training data** or it has high variance. One of the first thing you should try is regularization!

L2 regularization: Add a term to the loss function that is function of the parameters in he neural network

L2 regularization is also called **weight decay.** Let's see why!

Sources:
- Coursera

5. Regularization

L2 norm regularization or **weight decay:**

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right)+(1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

m: number of samples
λ: Regularization parameter (This is also a hyperparameter to choose when designing a neural network)

L2 regularization method is also called **weight decay** as λ will also multiplied the weight update during backpropagation

Sources:
- Coursera
- https://towardsdatascience.com/understanding-the-scaling-of-l%C2%B2-regularization-in-the-context-of-neural-networks-e3d25f8b50db

5. Regularization

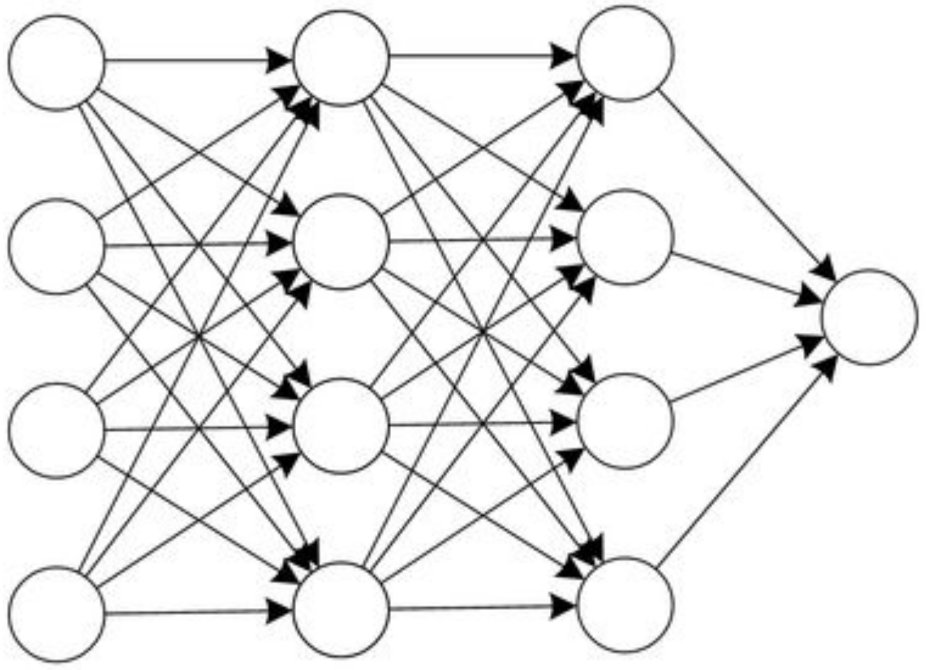
The reason L2 regularization method helps to reduce overfitting is that it shrinks the weight/parameter values for some units.

This effect helps the neural network to learn functions that are less complex or more "linear"

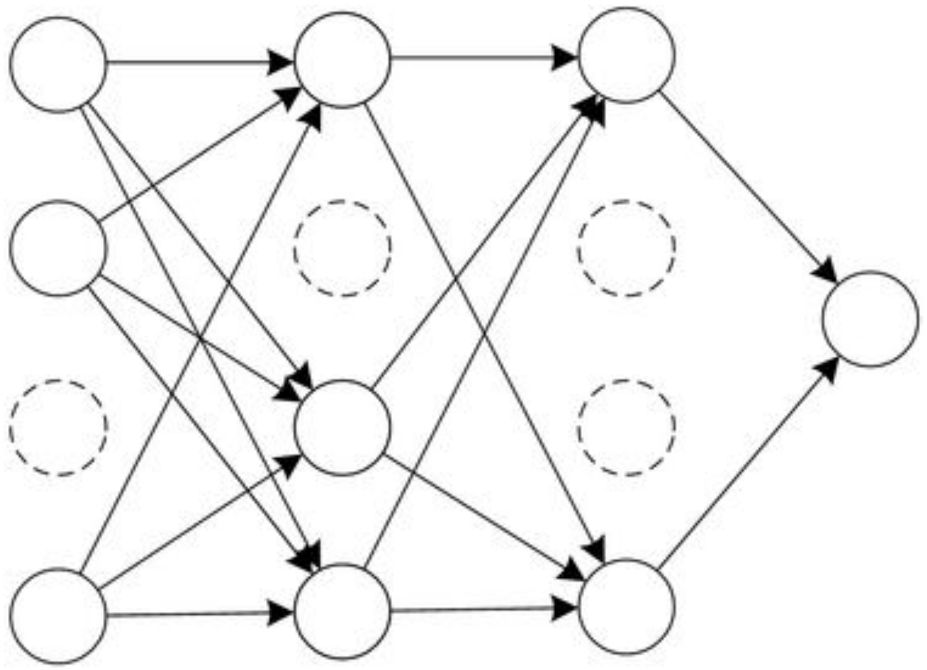
Sources:
- Coursera

5. Regularization

Dropout is also another regularization technique! How does it work?



(a) Standard Neural Network



(b) Network after Dropout

Essentially you define a ***probability of keeping a neuron.***

A common way of implementing dropout is using the "inverted dropout"

Sources:
- Coursera
- Amine ben khalifa et al. Multiple Instance Fuzzy Inference Neural Networks

5. Regularization

Why Dropout works as a regularizer?

Intuition: Can’t rely on any on feature, so have to spread out weights.

In the end, Dropout works similar to L2 norm regularization. It doesn’t allow the system to rely on certain units.

Sources:
- Coursera

5. Regularization

There are other regularization techniques such as:

- **Data augmentation** (mirroring, horizontal and vertical rotation, zooming, etc)
- **Early stopping:** You stop the training process when the dev set error gets bigger when compared to the training set error.

  Early stopping is not recommended as it breaks orthogonalization. This means, orthogonalization doesn't allow you to work on optimizing the cost function and avoiding overfitting independently.

Sources:
- Coursera

6. Normalizing Inputs

When training a neural network, one of the techniques that will speed up your training is input normalization. This consists of:

- Subtract out the mean
- Divide by the standard deviation.

This should be performed for the training and test set! For the test set you should use the mean and standard deviation extracted from the training set.

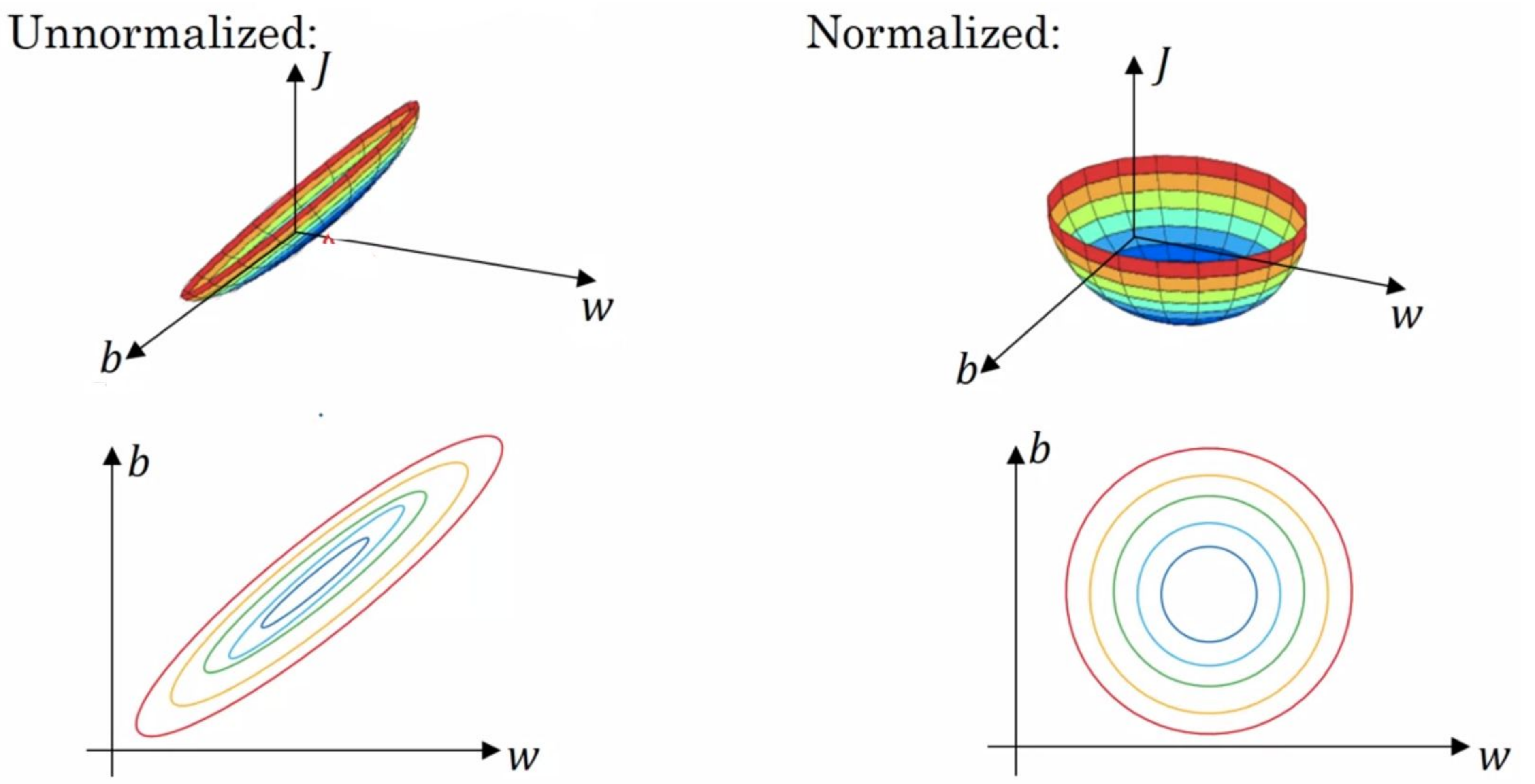
Dividing by the higher value on each input variable is another method.

The main idea is to keep the similar ranges among the input variables

Sources:
- Coursera

6. Normalizing Inputs

The main reason for normalizing the inputs is that the cost function is more rounded and easy to optimize!



Weights and biases will be higher or smaller depending on the input range.

That is why the cost function could be elongated when not normalizing the inputs.

Sources:
- Coursera

7. Vanishing/Exploding gradients

When training very deep neural networks, **slopes or the derivatives can get very big (exploding) or very very small (vanishing)**

To remember:

- Weight values that are smaller than 1 make the updates to decrease when doing multiplication ***(vanishing gradients)***
- The contrary happens when having derivative values bigger than 1, updates increase ***(exploding gradients)***

A proper weight initialization method may help to reduce this problem!

Sources:
- Coursera

Let’s move to Google Colab!

Notebooks:

- *3_First_Deep_Neural_Network_Part1.ipynb*
- *4_First_Deep_Neural_Network_Part2_Application.ipynb*

Sources:
- Coursera