Building Neural Networks with PyTorch

Machine Learning

It's not Magic

There's No Free Lunch

Aritra Ghosh

Dec. 6, 2022

Yale Software Journal Club

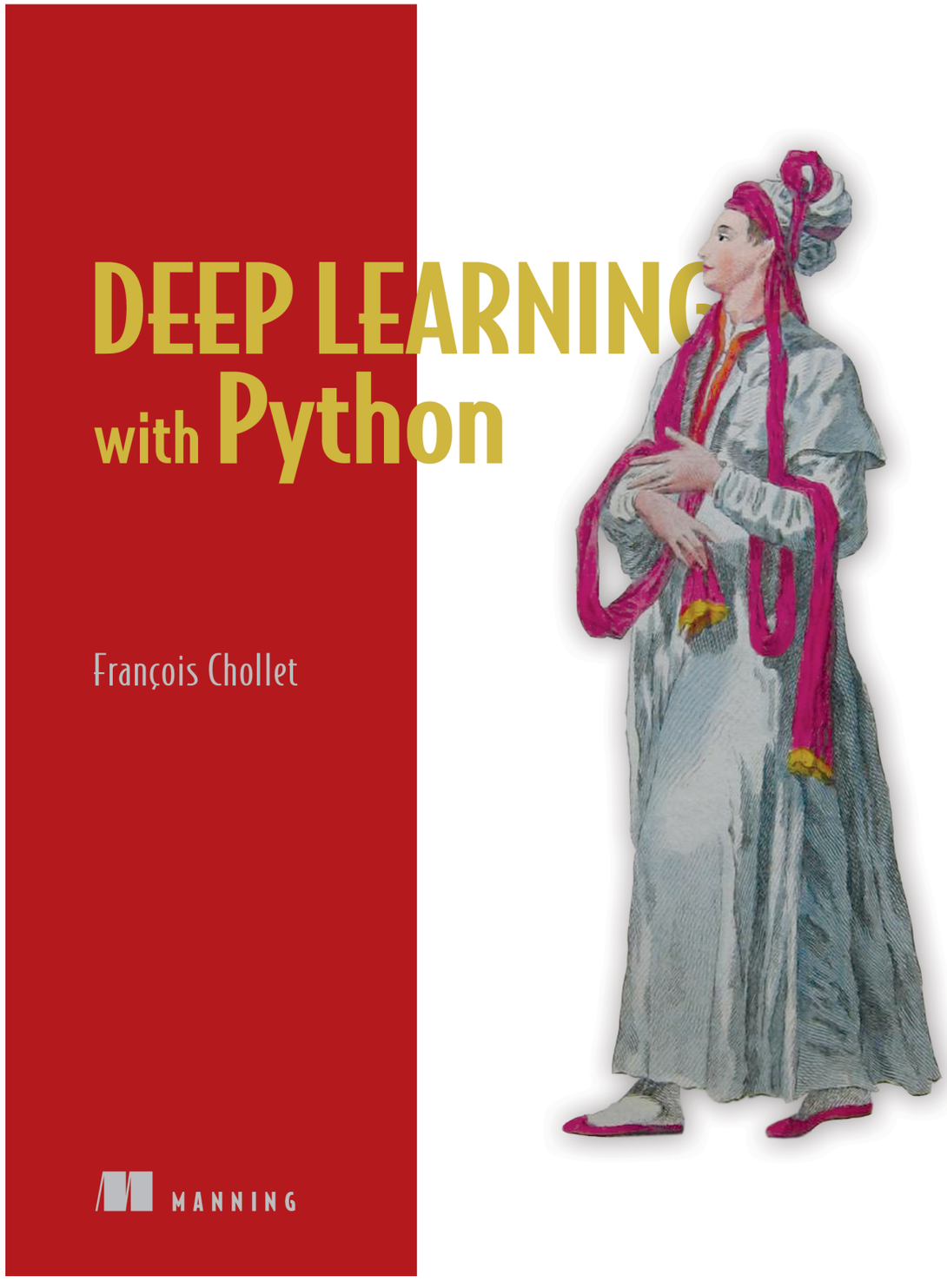
Two Very Good References

Neural Networks and Deep Learning

Neural Networks and Deep Learning is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.



Live Demo: NNs in action!

Some “State-of-the-Art” Neural Networks

Can take any text prompt and turn it into an image

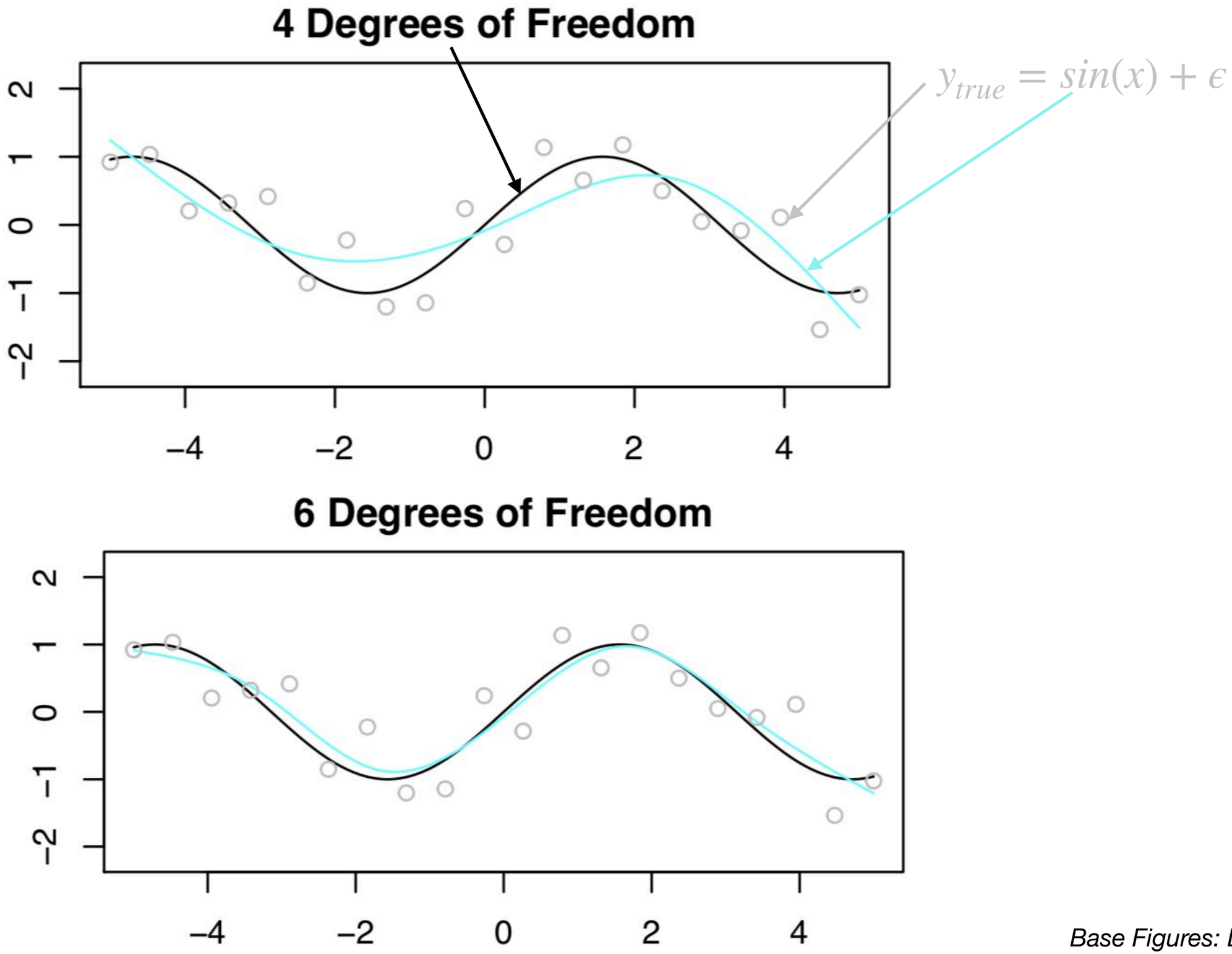
State of the Art in NLP

Can accurately predict 3D models of protein structures

Modern state-of-the art ML frameworks have a wide array of applications!
Many of them have a neural network component at their core!

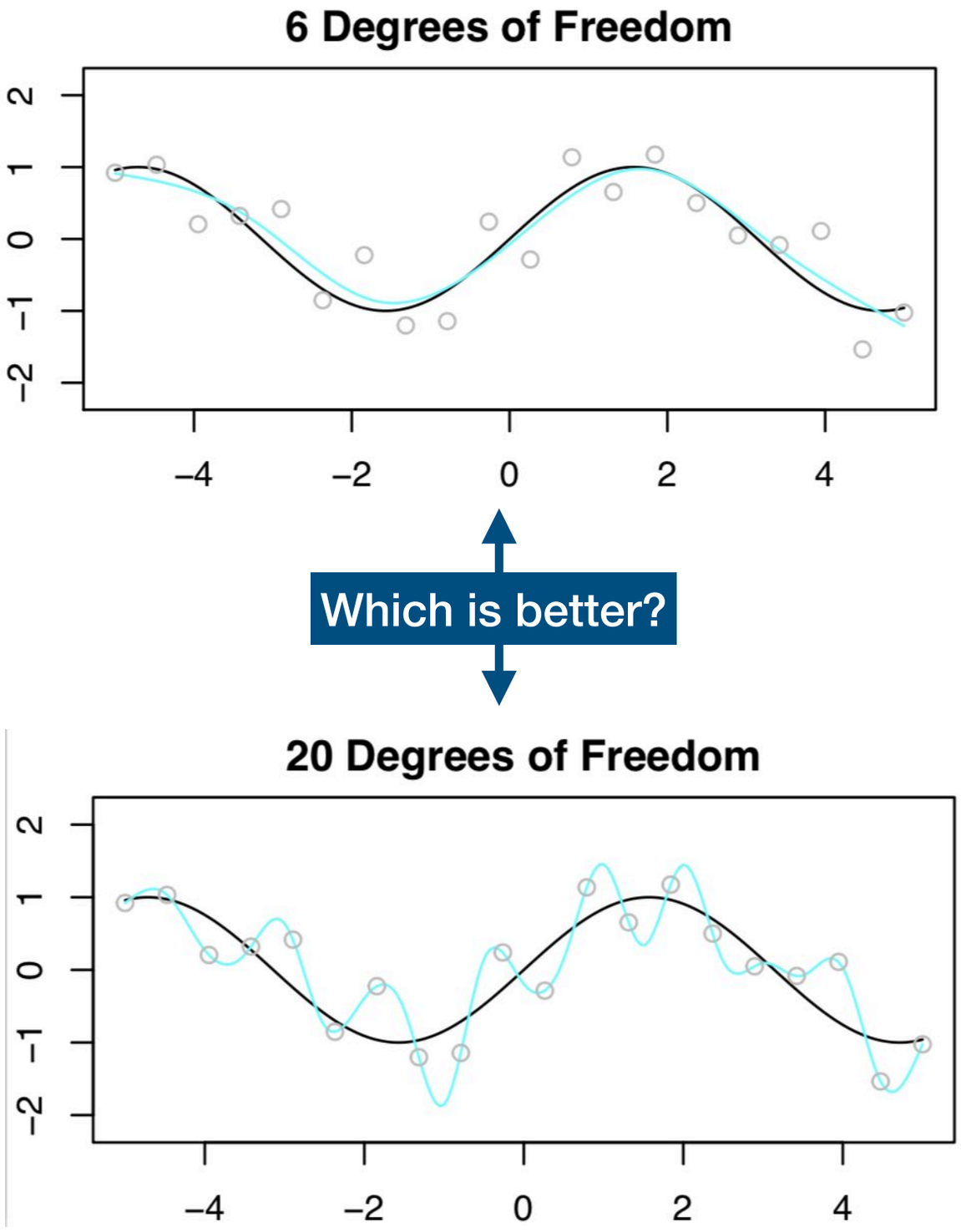
Some Very Fundamental Concepts About Why NNs work!

Fitting with Increasingly Complex Splines

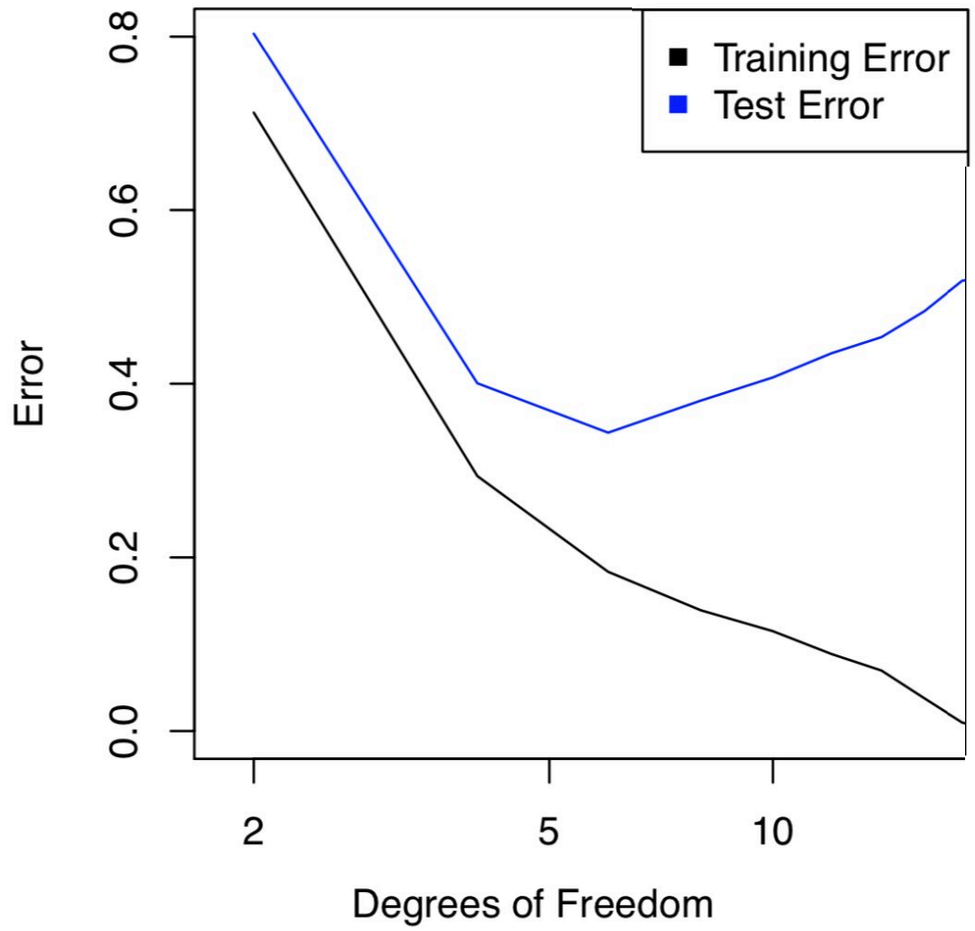


Base Figures: Daniella Witten, UW

Fitting with Increasingly Complex Splines



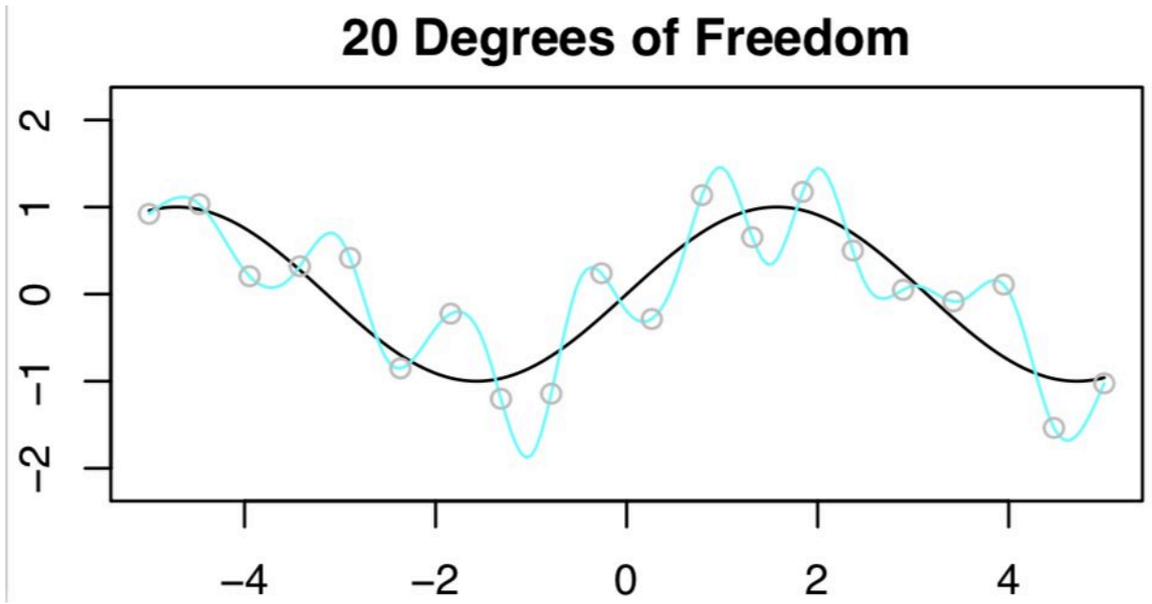
The bias variance tradeoff curve confirms our intuition!



Base Figures: Daniella Witten, UW

Fitting with Increasingly Complex Splines

Now, let's push this even further!



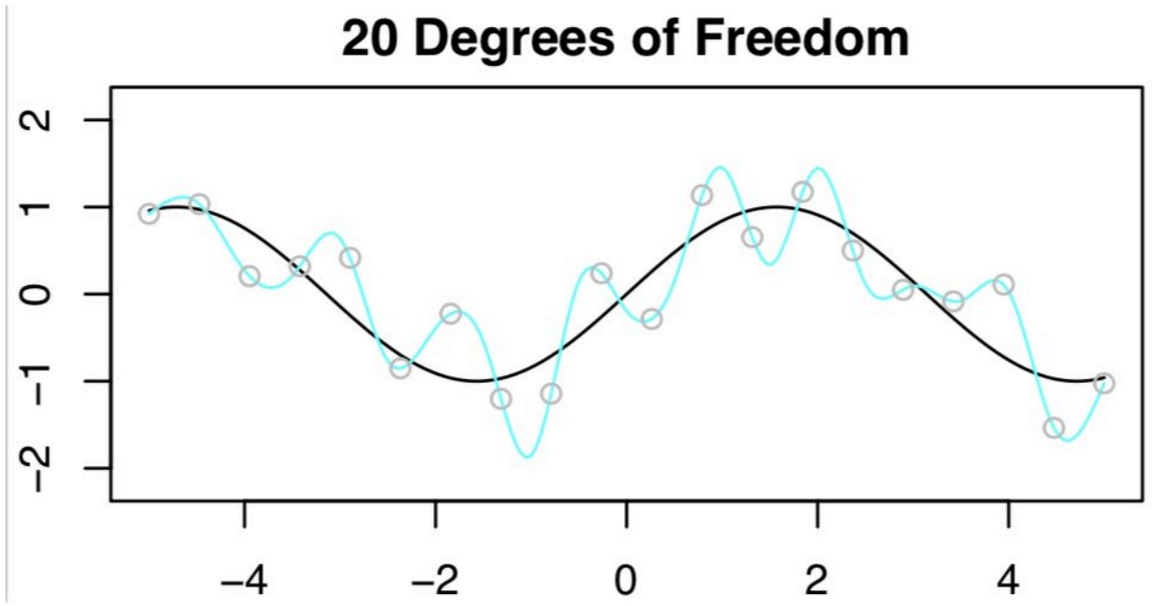
Now, for since there are only 20 points, if we want to fit this with more than 20 DOF, the answer via least squares is no longer unique!

Among the infinite number of possible solutions, let's pick the one with the "minimum" norm fit: the one with the smallest sum of squared co-efficients!

Base Figures: Daniella Witten, UW

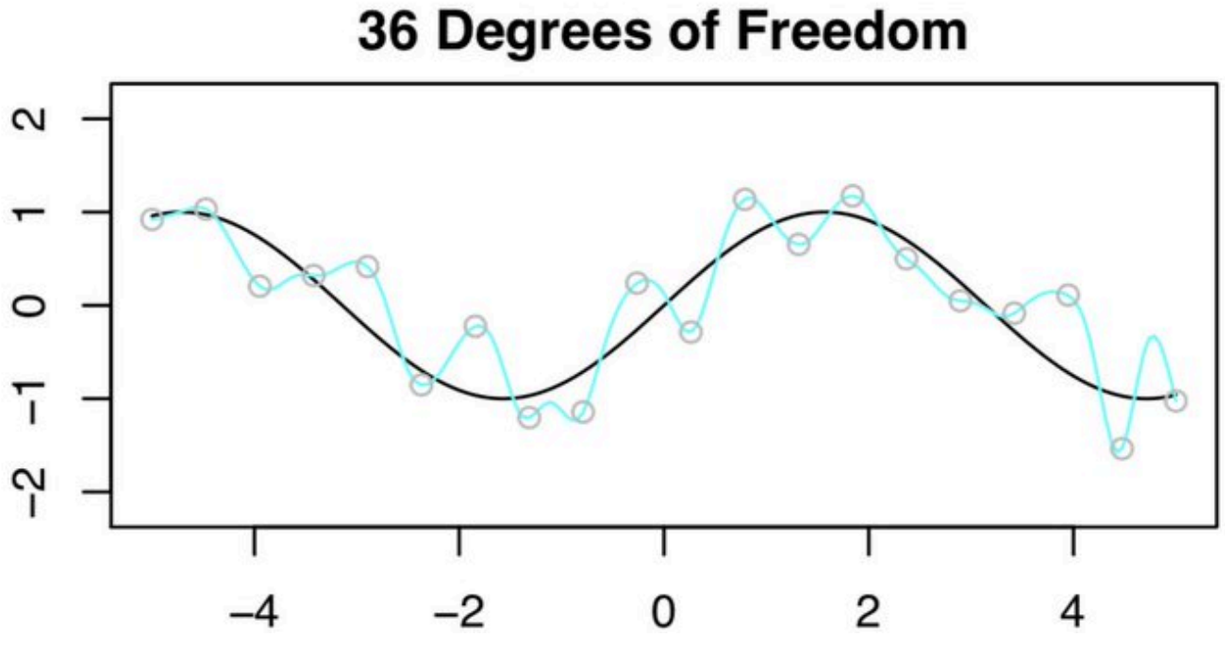
Fitting with Increasingly Complex Splines

Now, let's push this even further!



Let's take a look at the training and test errors to decide!

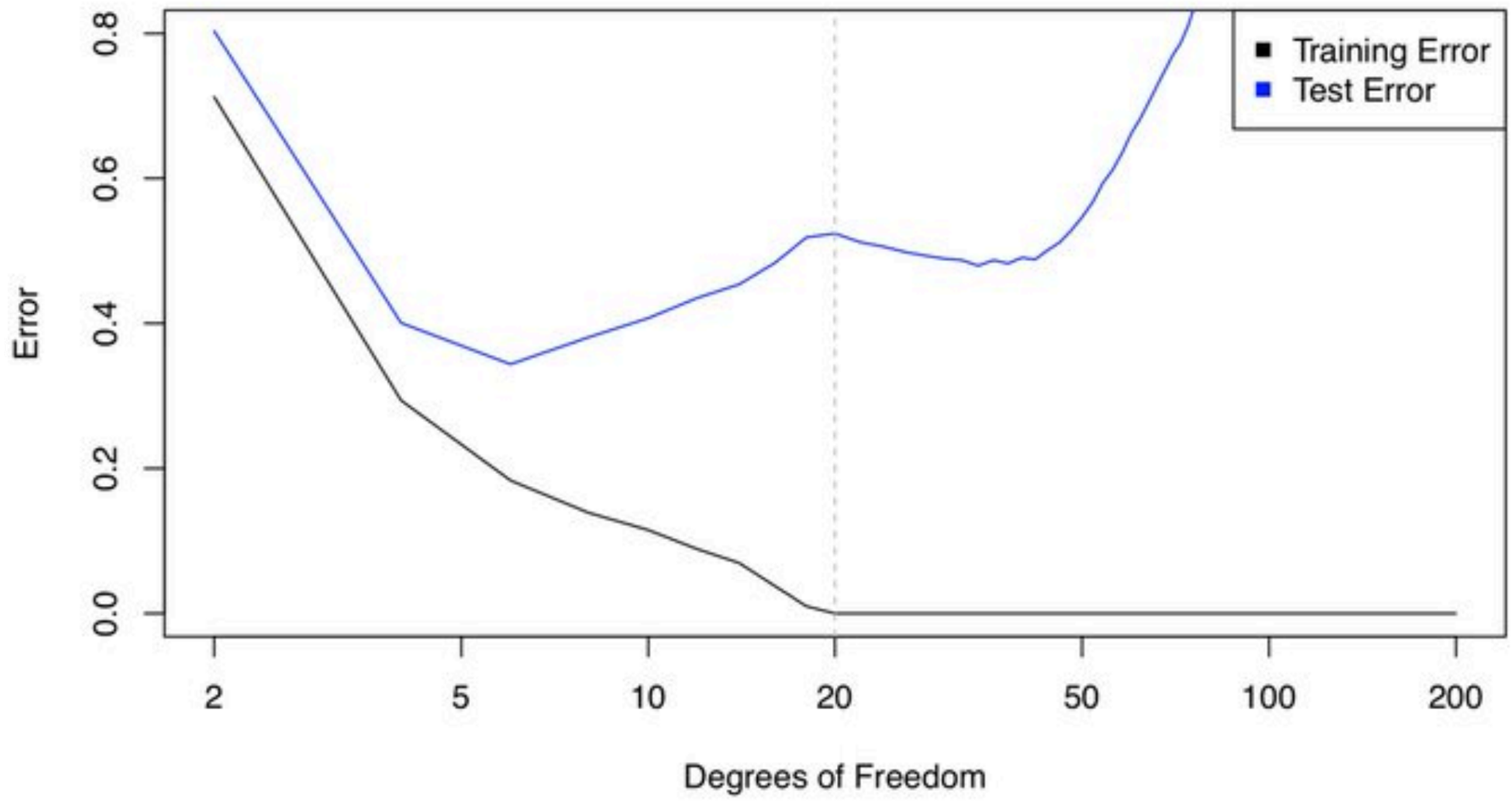
Which cyan line is a worse fit? (As in higher error?)



Base Figures: Daniella Witten, UW

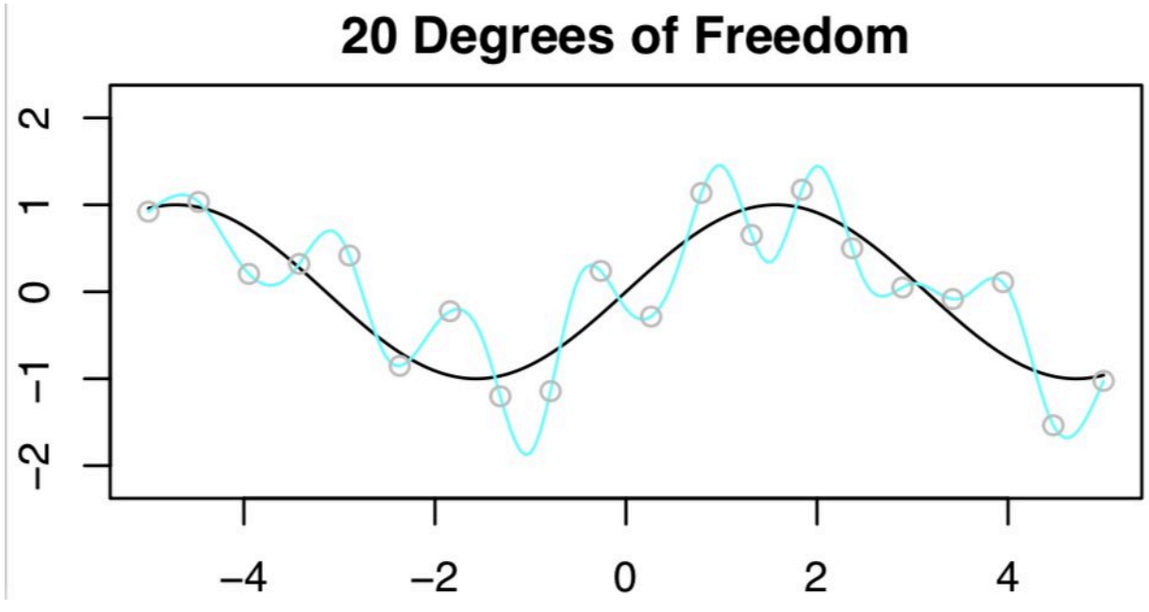
Fitting with Increasingly Complex Splines

From the bias-variance tradeoff, shouldn't the 36 DOF fit be much worse than the 20 DOF fit?

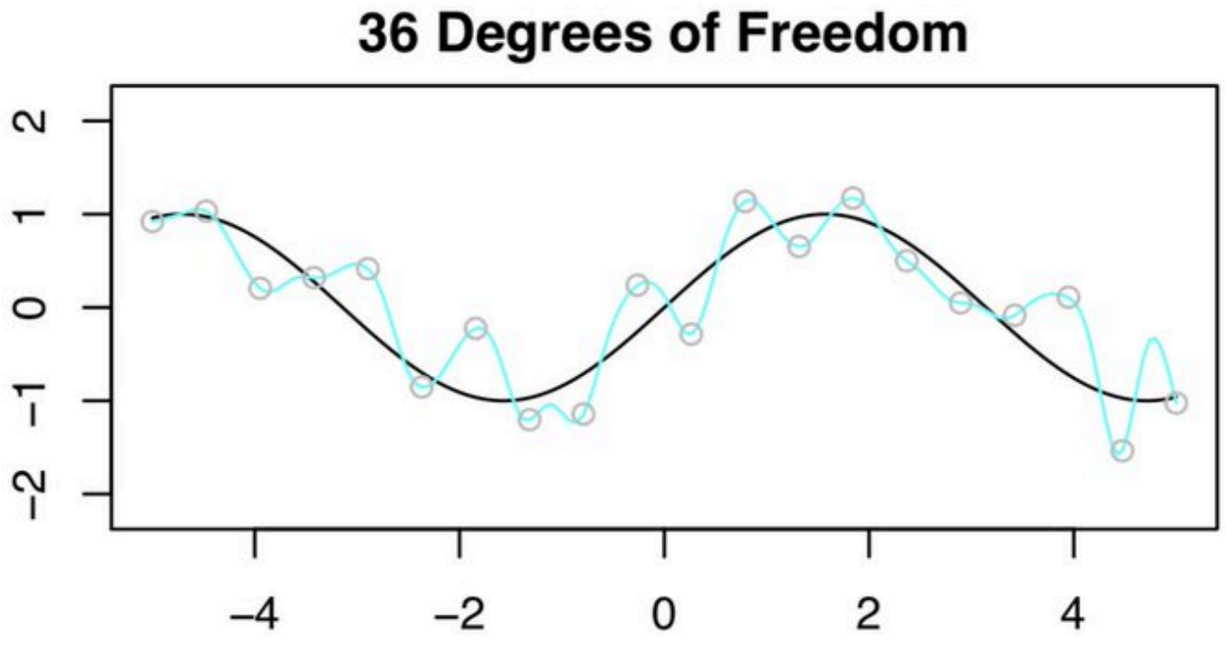


Base Figures: Daniella Witten, UW

Fitting with Increasingly Complex Splines



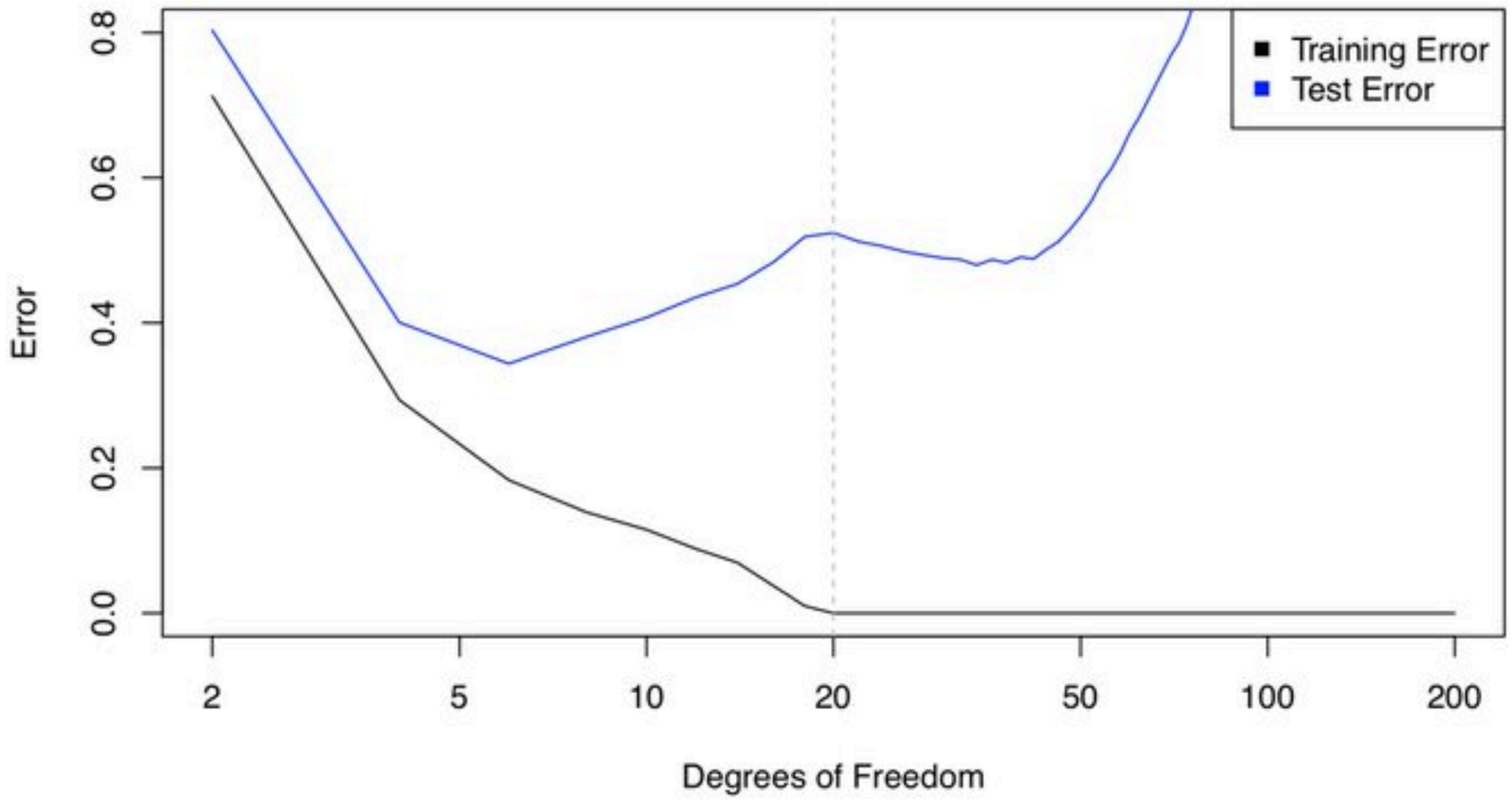
The key point is with 20 DF, n=p, and there's exactly ONE least squares fit that has zero training error. And that fit happens to have oodles of wiggles



The minimum norm least squares fit is the “least wiggly” of the many many fits out there! In fact, the “least wiggly” is even less wiggly than the p=n case

Base Figures: Daniella Witten, UW

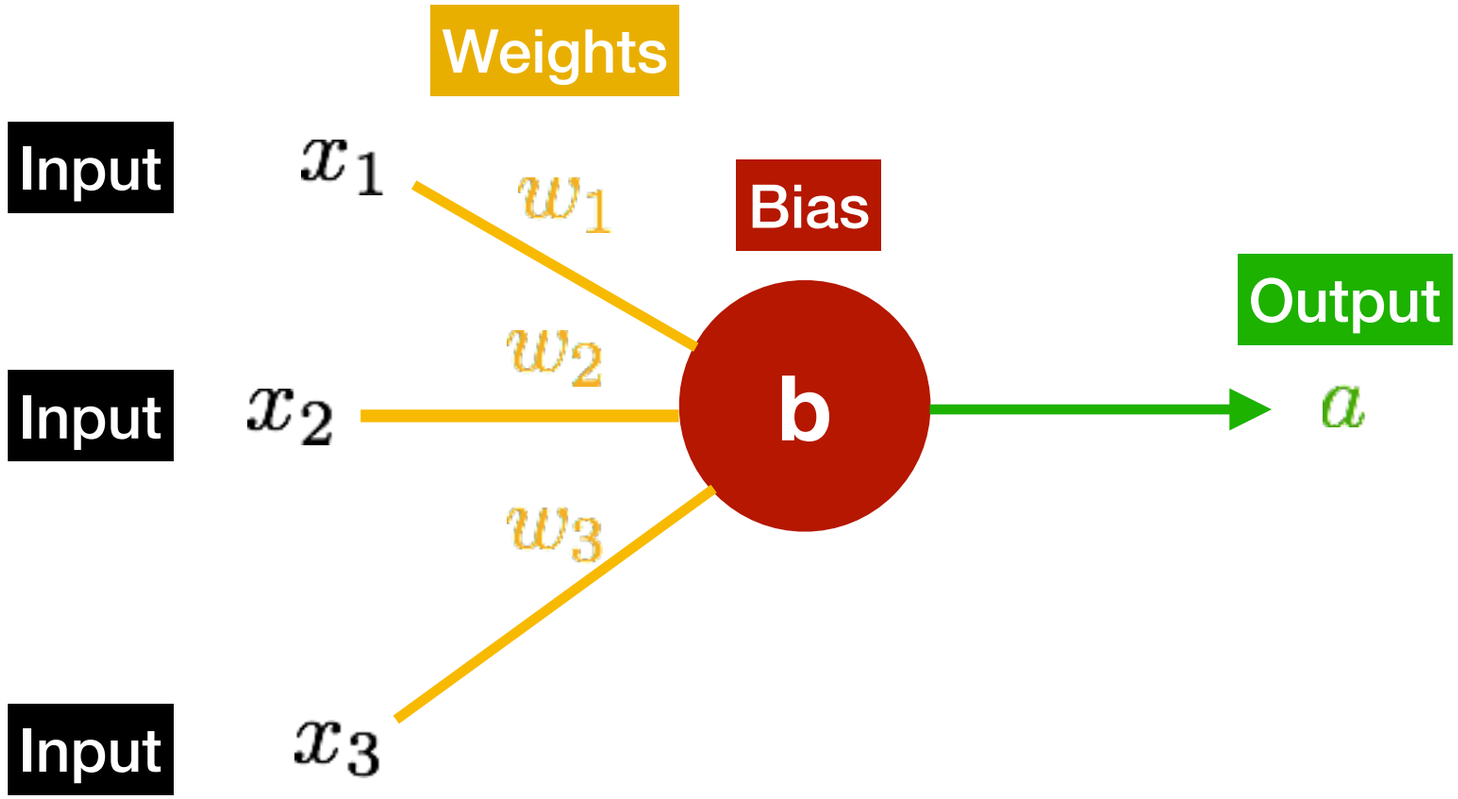
Fitting with Increasingly Complex Splines



Therefore, all is well! “Double Descent” is happening because choosing the minimum norm least squares fit makes the p=36 model less flexible than the p=20 spline model. Thus, this is NOT magic!

Deep Learning works primarily because usually we use a technique called stochastic gradient descent (SGD) to train deep learning models, and doing SGD is equivalent to picking the minimum norm solution!

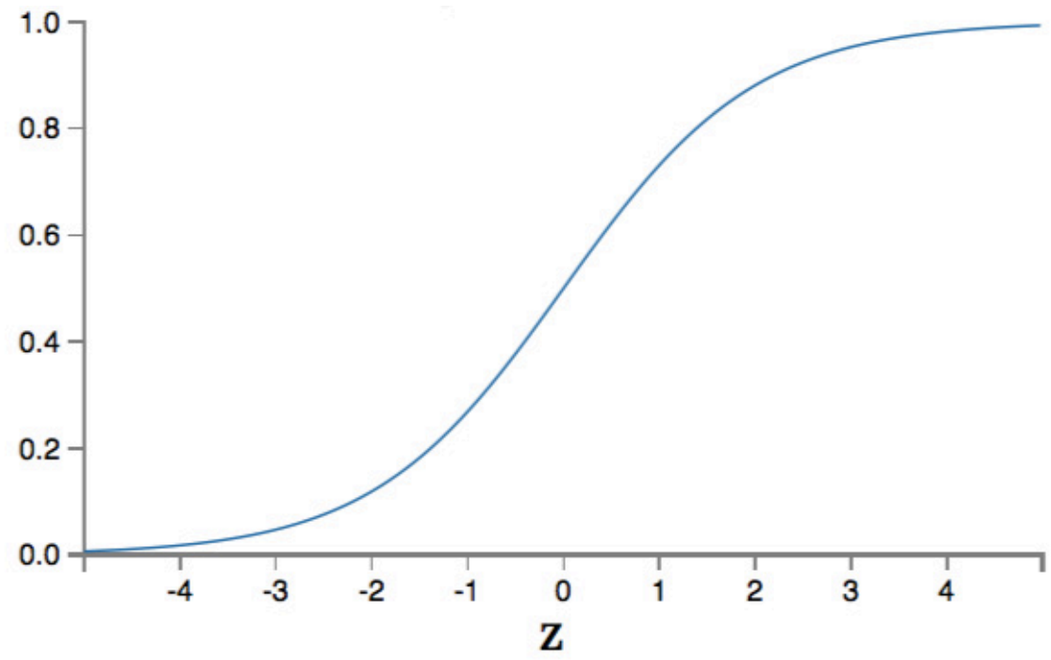
The Sigmoid Neuron: The Mathematics



$$a = \sigma(w \cdot x + b)$$

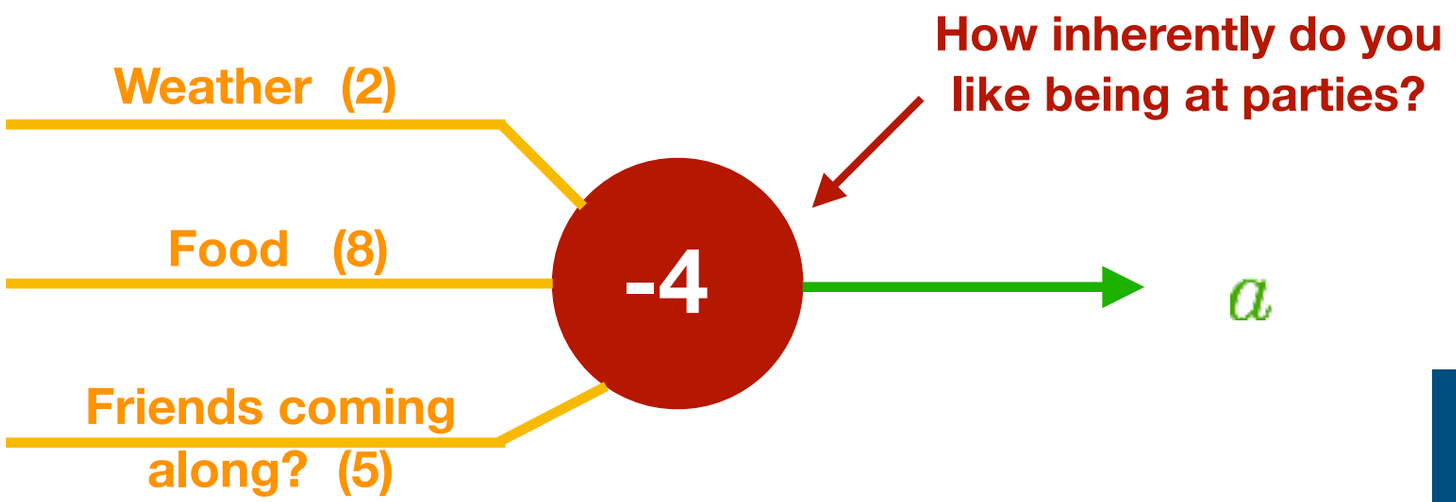
Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



The Sigmoid Neuron: The Logic

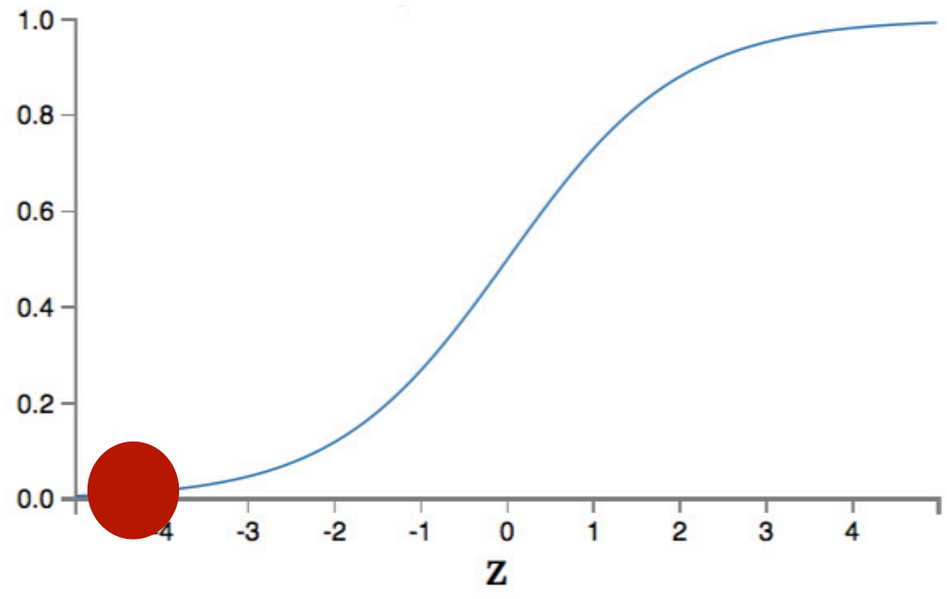
Should I go to XYZ party?



Neuron has been "triggered"

$$a = \sigma(w \cdot x + b)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

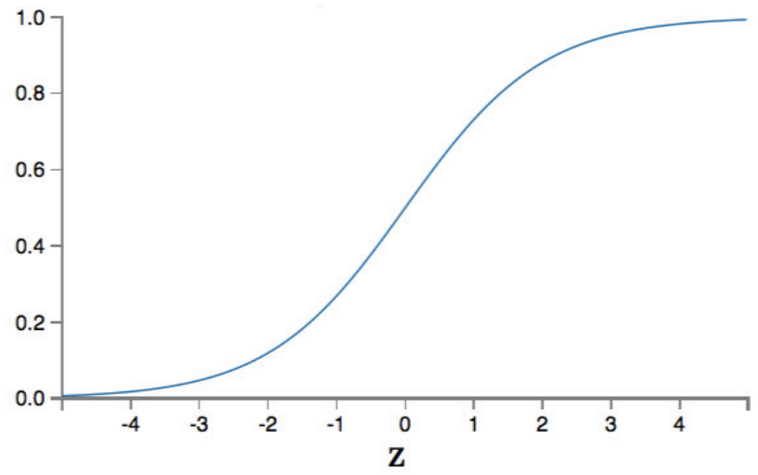


Therefore neurons in a neural network represent individual units that help us take "informed decisions" by taking into account a combination of different external factors!

The Activation Function

Is this the only functional form that the neutron can take?

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



NO!

There are many more functions that can be used as "activation functions". Here are few more frequently used examples

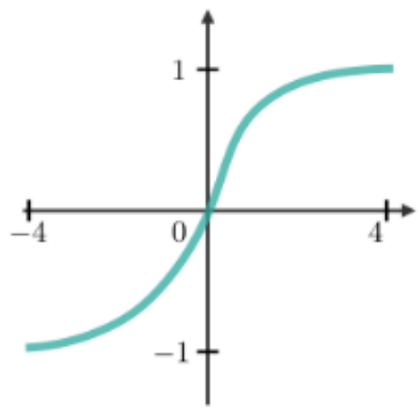
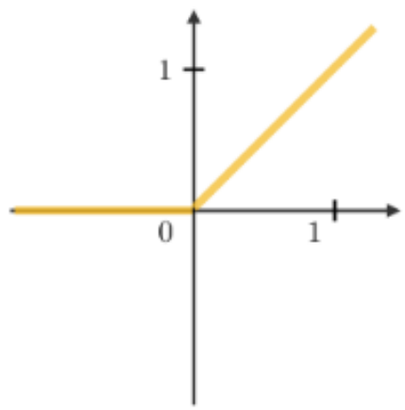
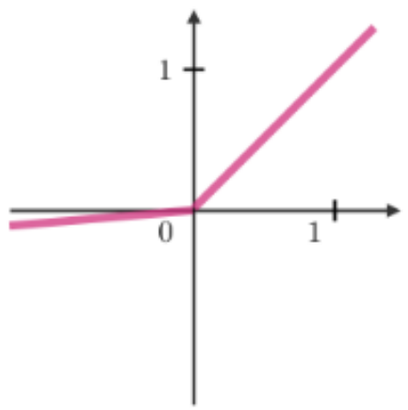
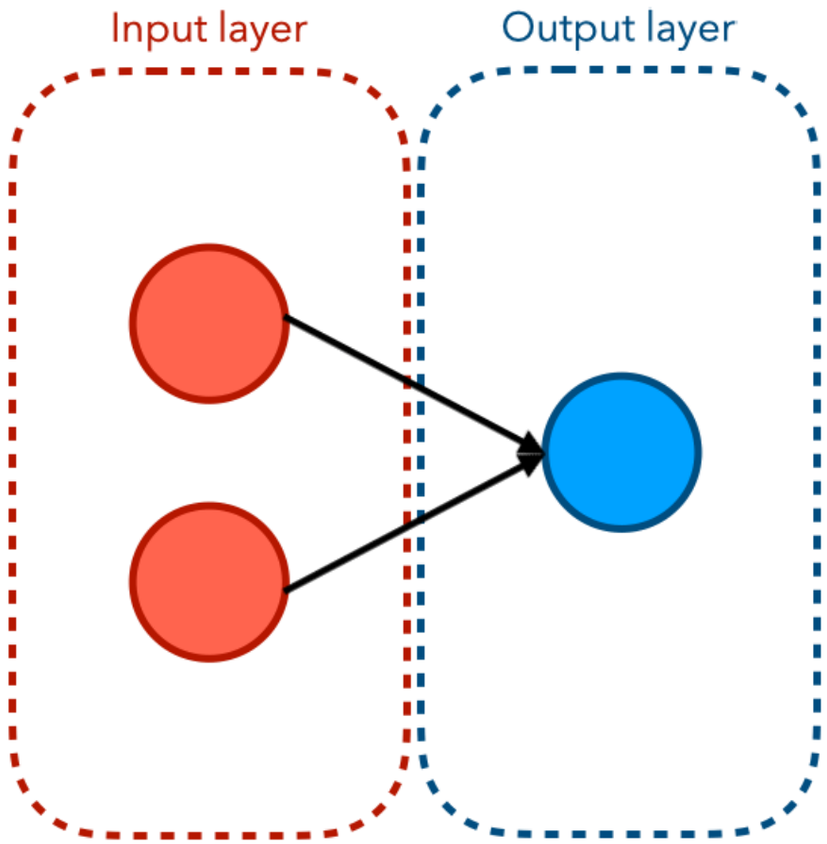
Tanh	ReLU	Leaky ReLU
$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
		

Image: Stanford CS229

Question: Can any mathematical function be used as an activation function?

From a Neuron to a Neural Network

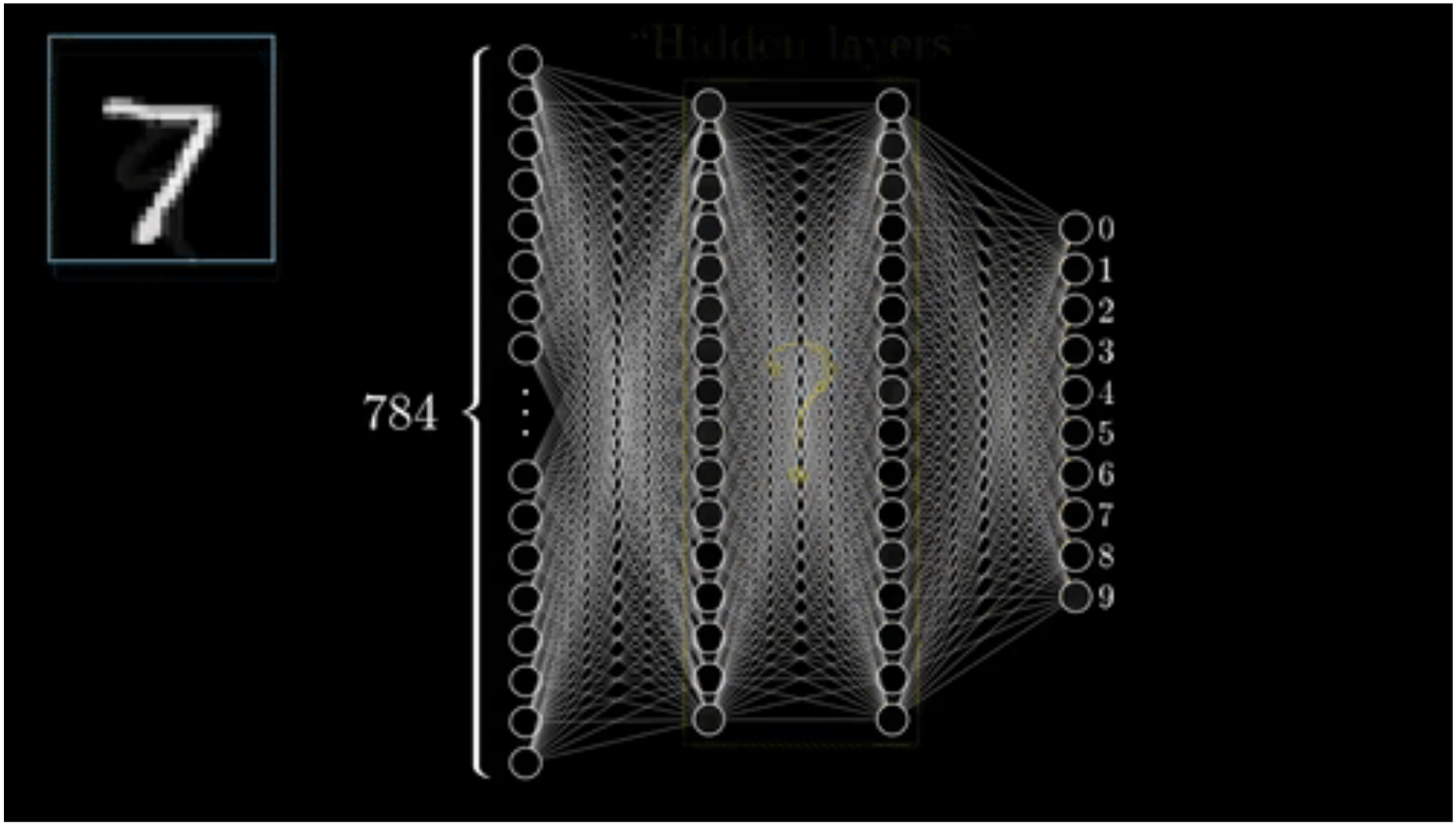
The decision making power of neurons *can be combined* to create a of *feedforward network* of many neurons — this is called a neural network!



Credit: Towards Data Science

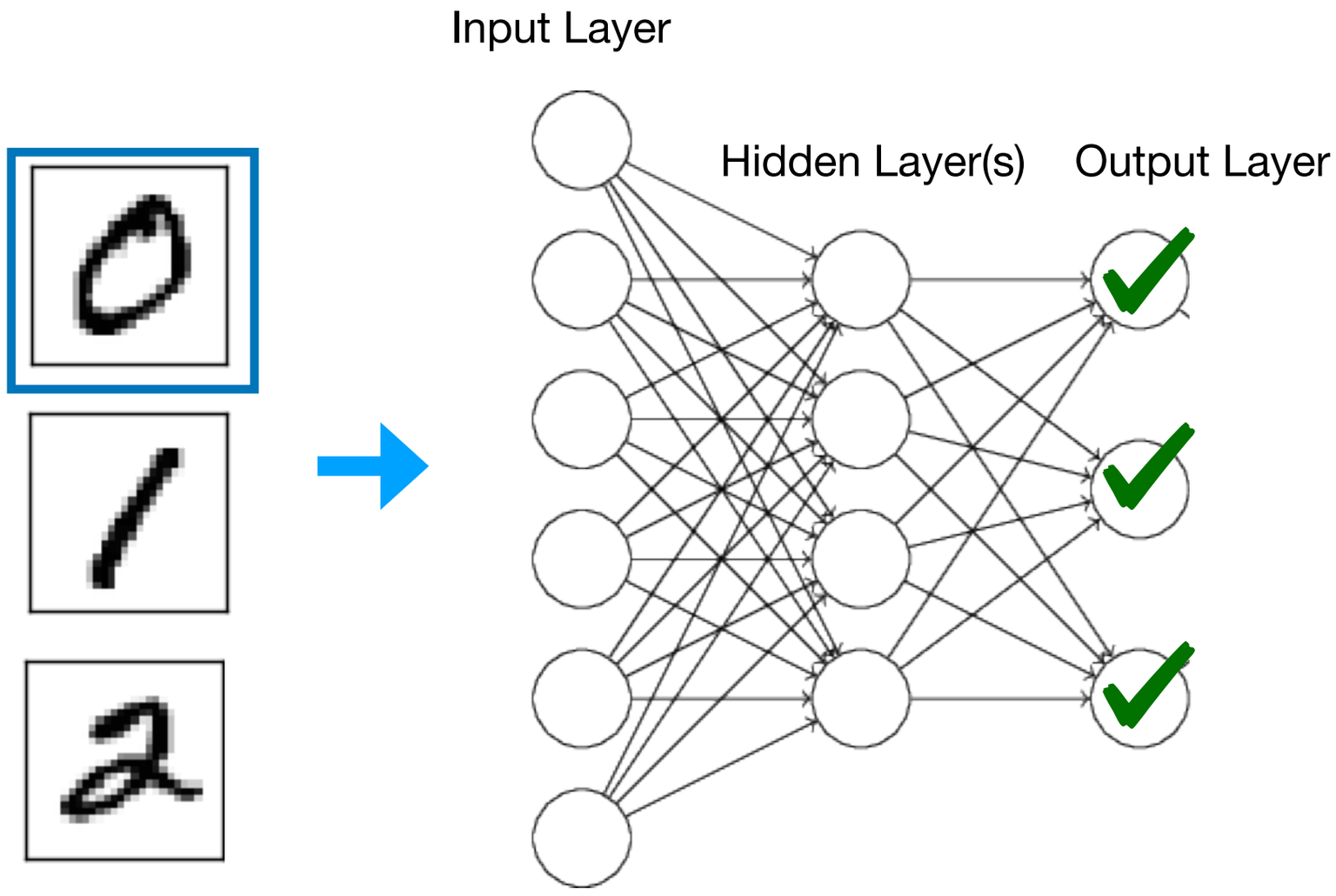
Basic Design Principle: The output of every neuron in a layer is fed as an input to *all* neurons in the next layer — *hierarchical decision making*

What happens during a forward pass?



In order for a neural network to make the correct prediction given an input, *the correct series of neurons need to be "activated"*!
This of course depends on the values of the *weights & biases of the entire network*!

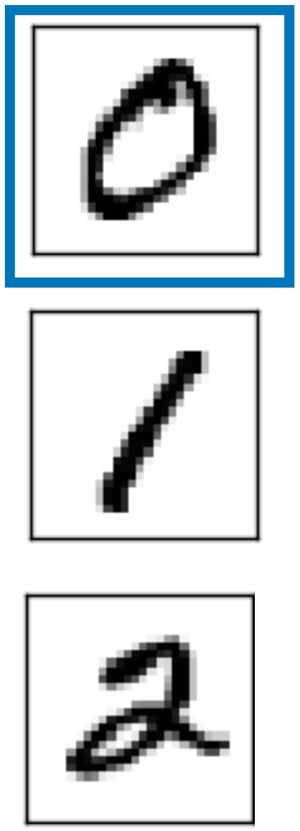
Training a tiny neural network



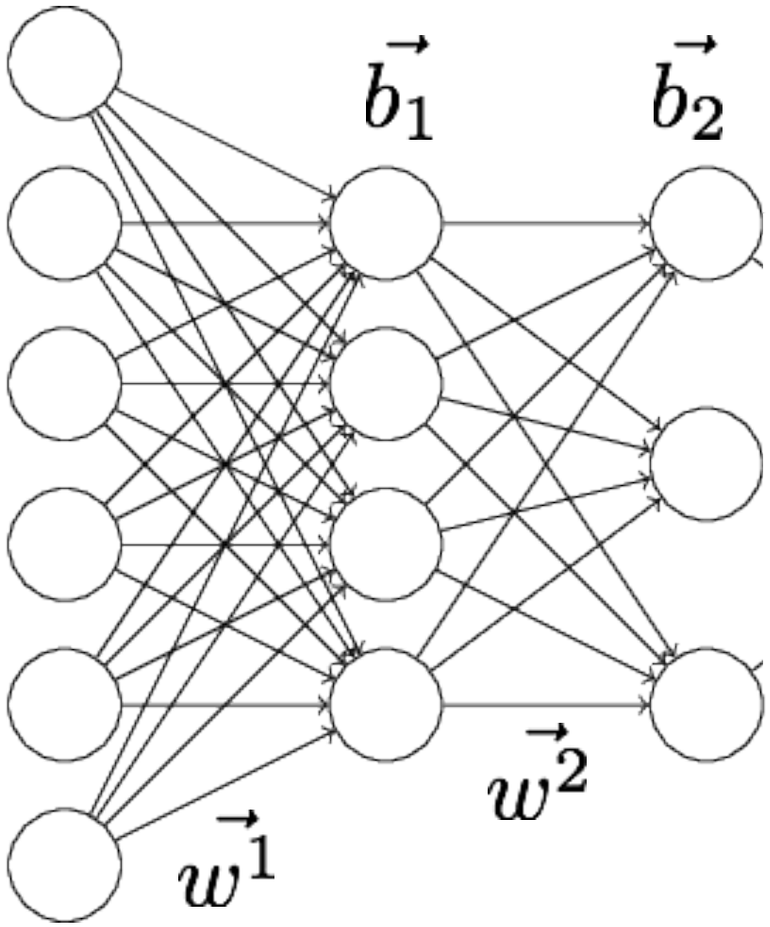
At the heart of the learning algorithm is the loss function!

Training Input

$\vec{x}$



Input Layer Hidden Layer(s) Output Layer



Desired Output

$\vec{y(x)}$ [1, 0, 0]

$\vec{a}$

Network Output

Loss function (Categorical Cross Entropy) $C(w,b) = \sum_{j=1}^{3} y_j \log(a_j)$

Examples of Different Loss Functions

Selecting an appropriate loss function is one of *the most important steps* in designing a neural network!

Desired Output $y(\vec{x})$

Network Output $\vec{a}$

Classification Problems

Categorical Cross Entropy (M-Classes)

$$C(w,b) = \sum_{j=1}^{M} y_j \log(a_j)$$

Binary Cross Entropy

$$C(w,b) = \sum_{j=1}^{2} y_j \log(a_j)$$

Regression Problems

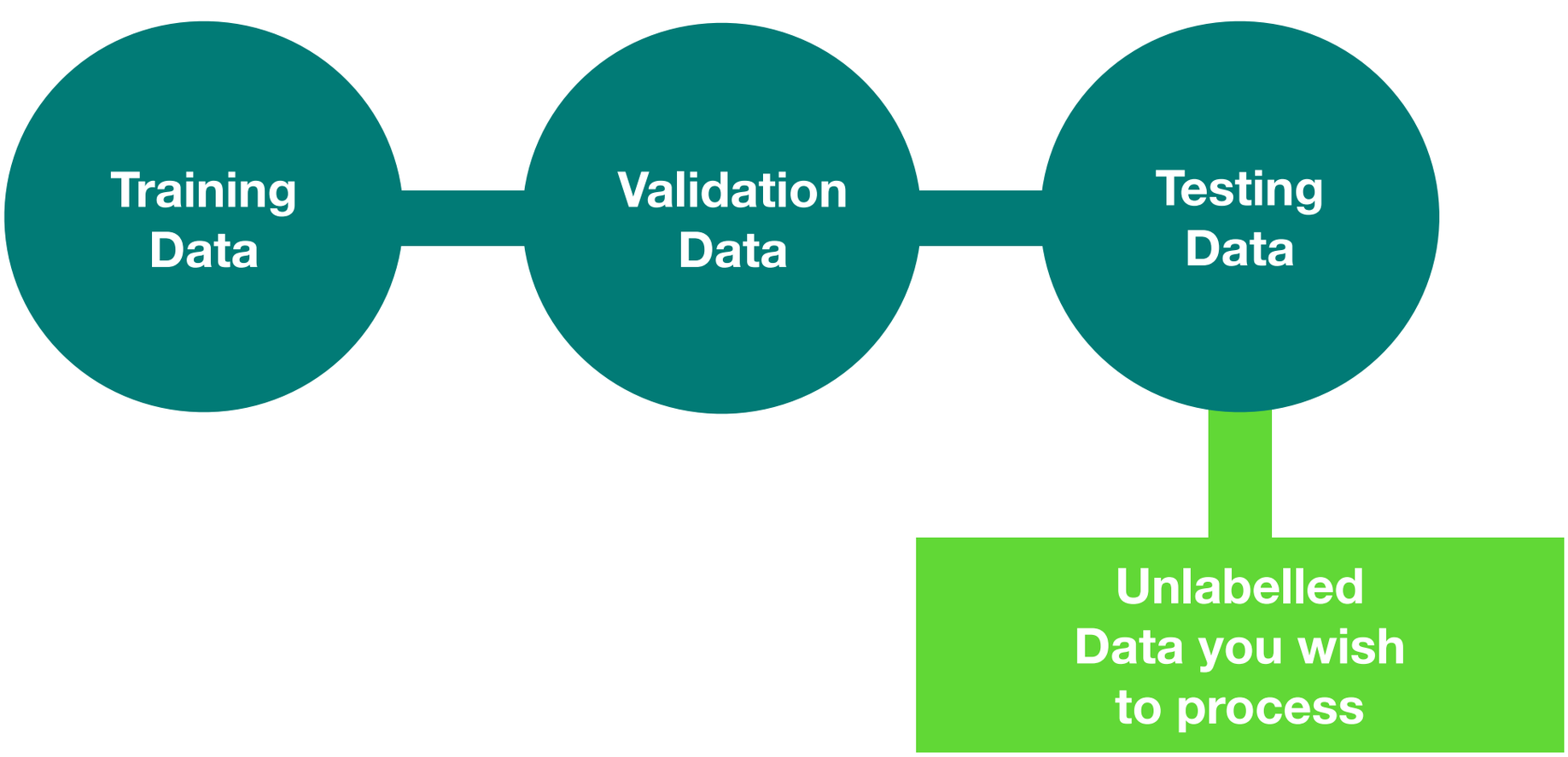
Mean Squared Error

$$C(w,b) = (y-a)^2$$

What does training an ML algorithm mean?

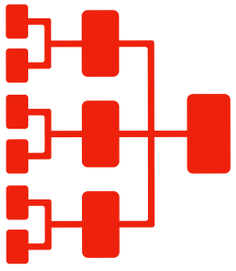
Step 1: Divide your dataset into three mutually exclusive portions.

You must know the right answers / have labels for all three sets



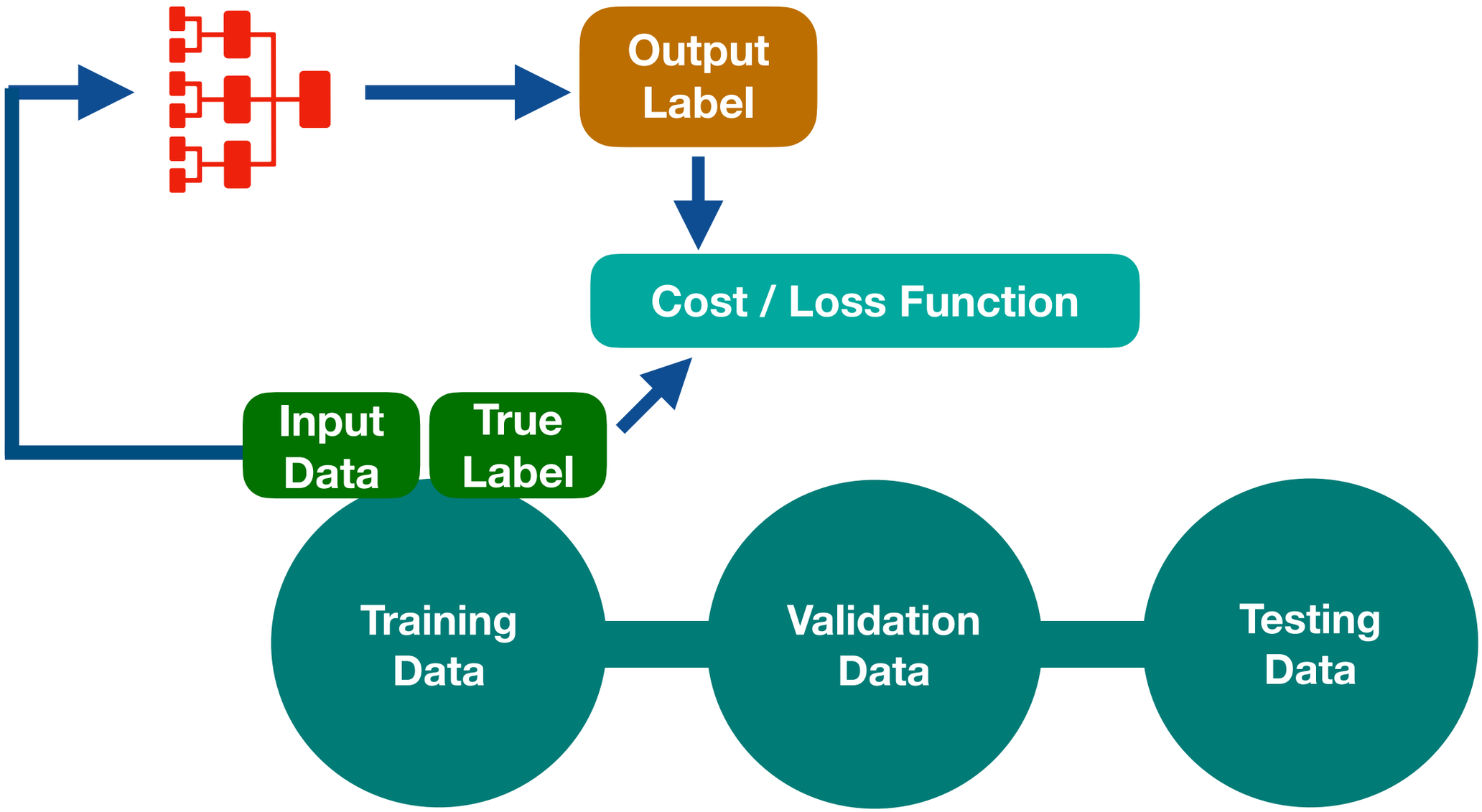
What does training an ML algorithm mean?

Step 2: Choose a Model



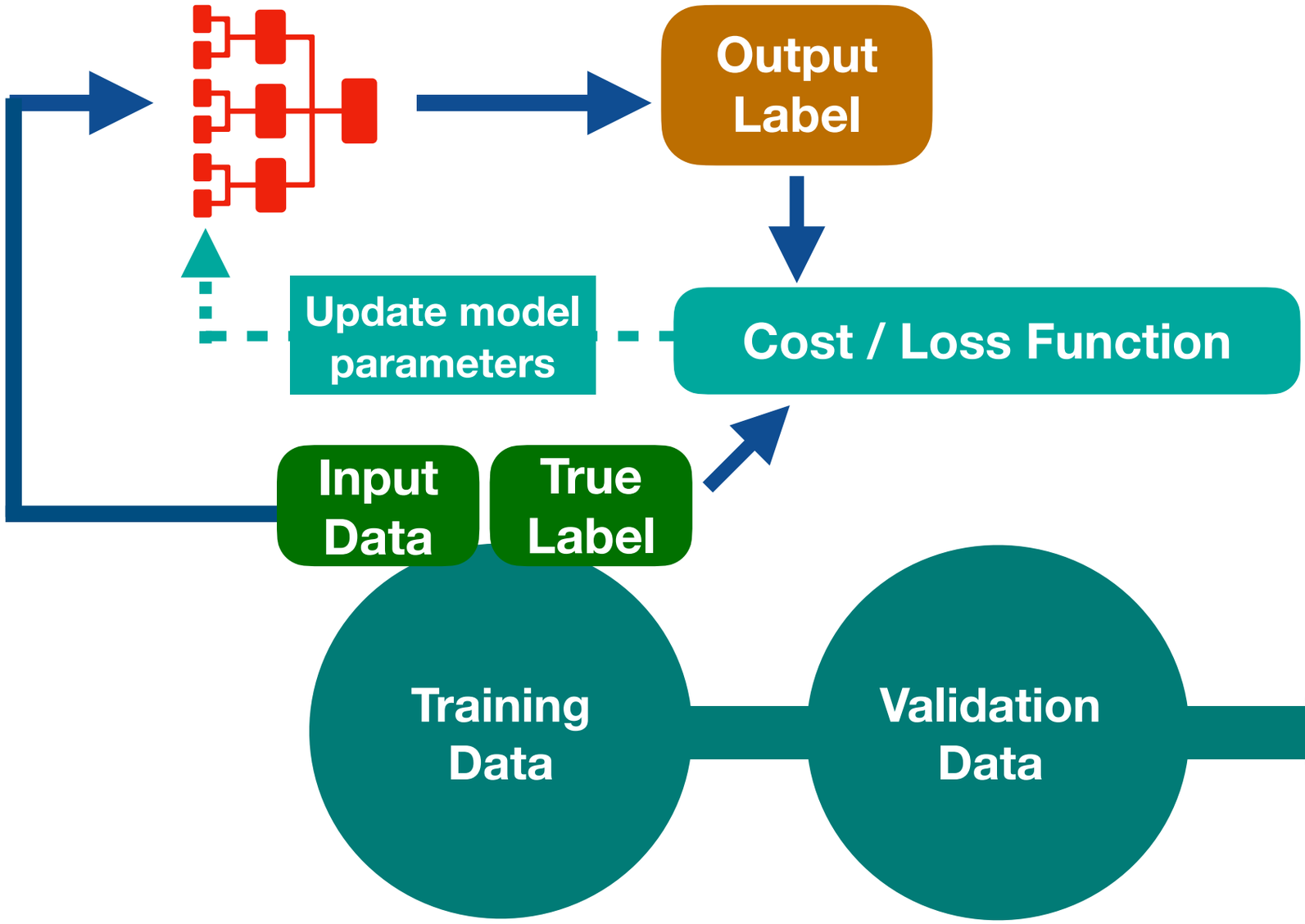
What does training an ML algorithm mean?

Step 3: Define a Cost Function
(A function that evaluates the level of agreement between the true label/value and predicted value)

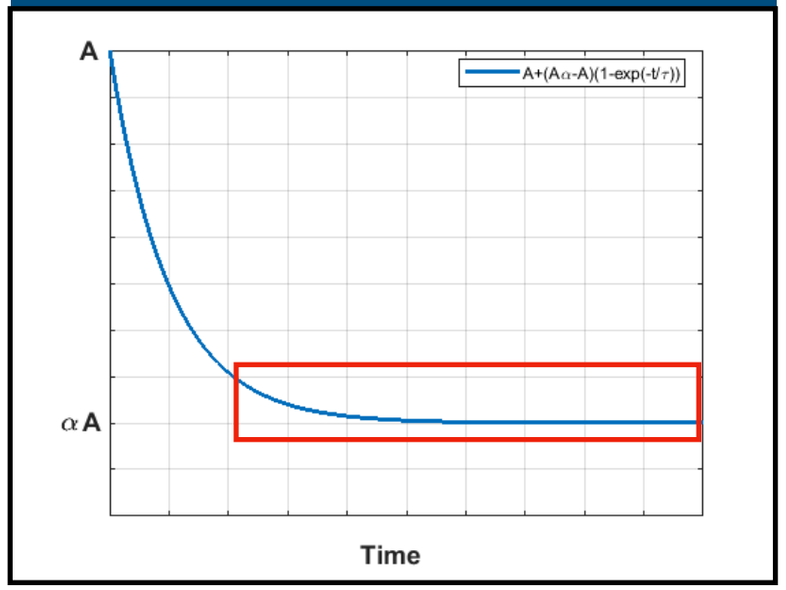


What does training an ML algorithm mean?

Step 4: Fine-tune parameters of model iteratively

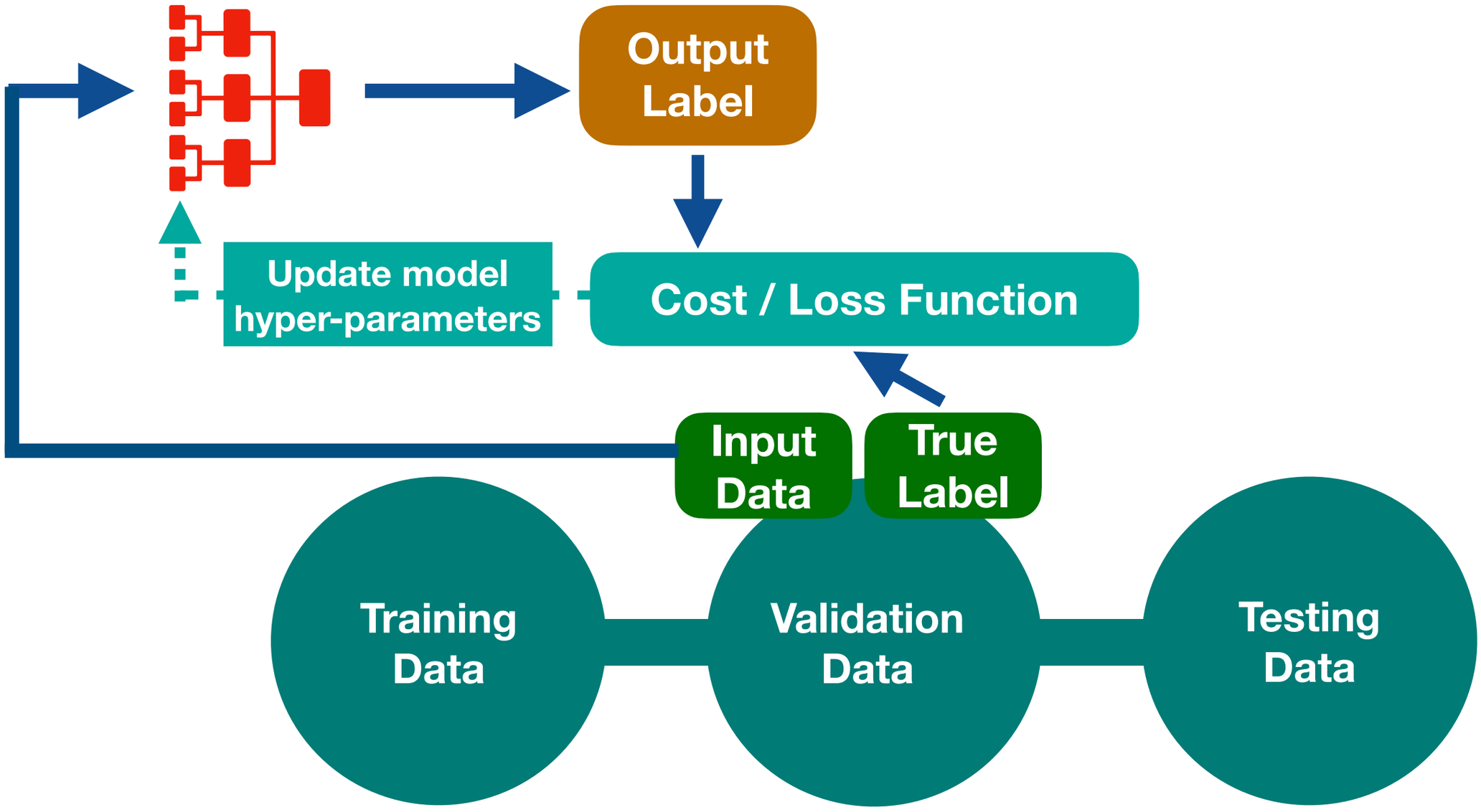


When do you stop?



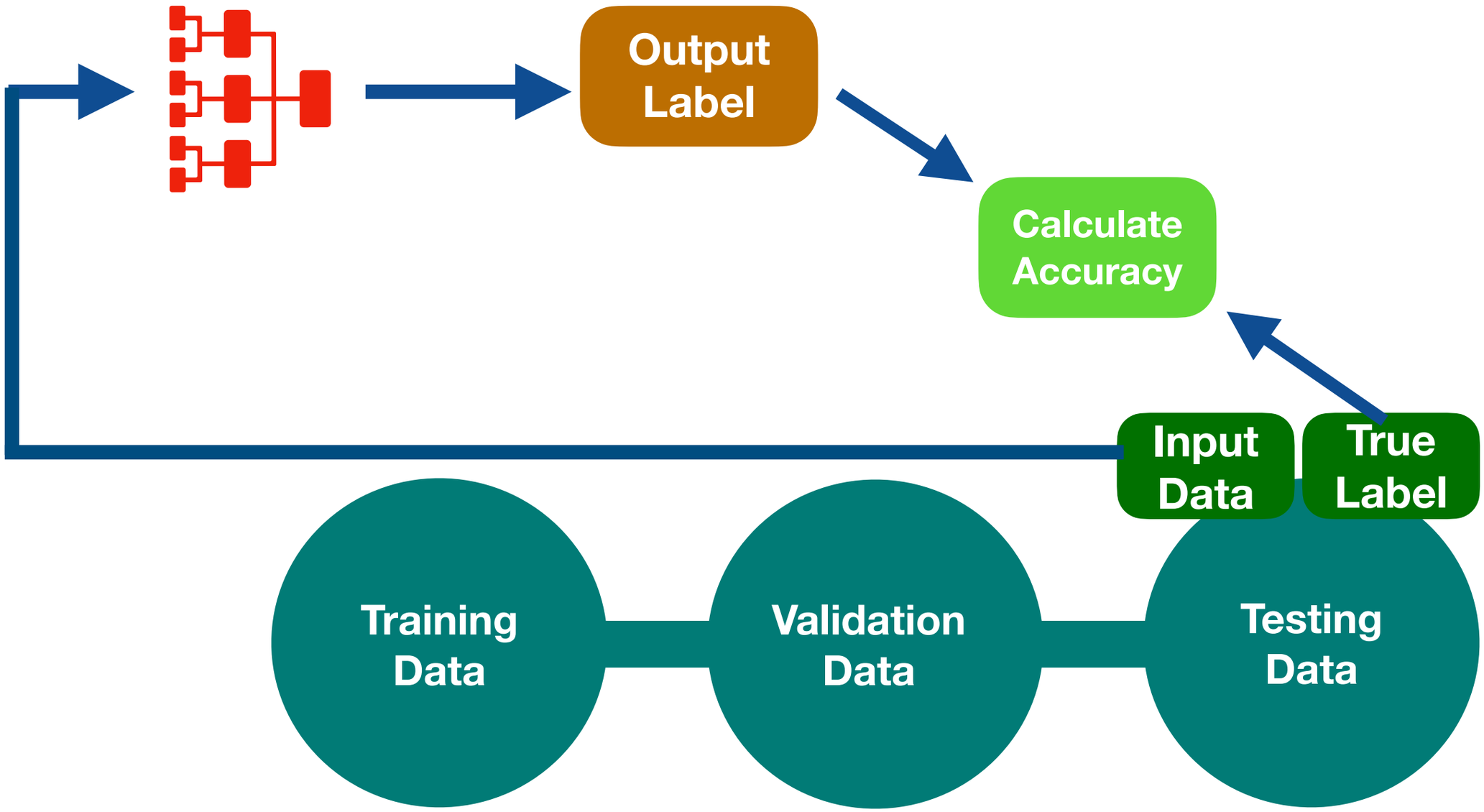
What does training an ML algorithm mean?

Step 5: Fine Tune Hyper-Parameters of the model



What does training an ML algorithm mean?

Step 6: Evaluate performance on testing set
(IMP!!! : model should have never seen this data before)

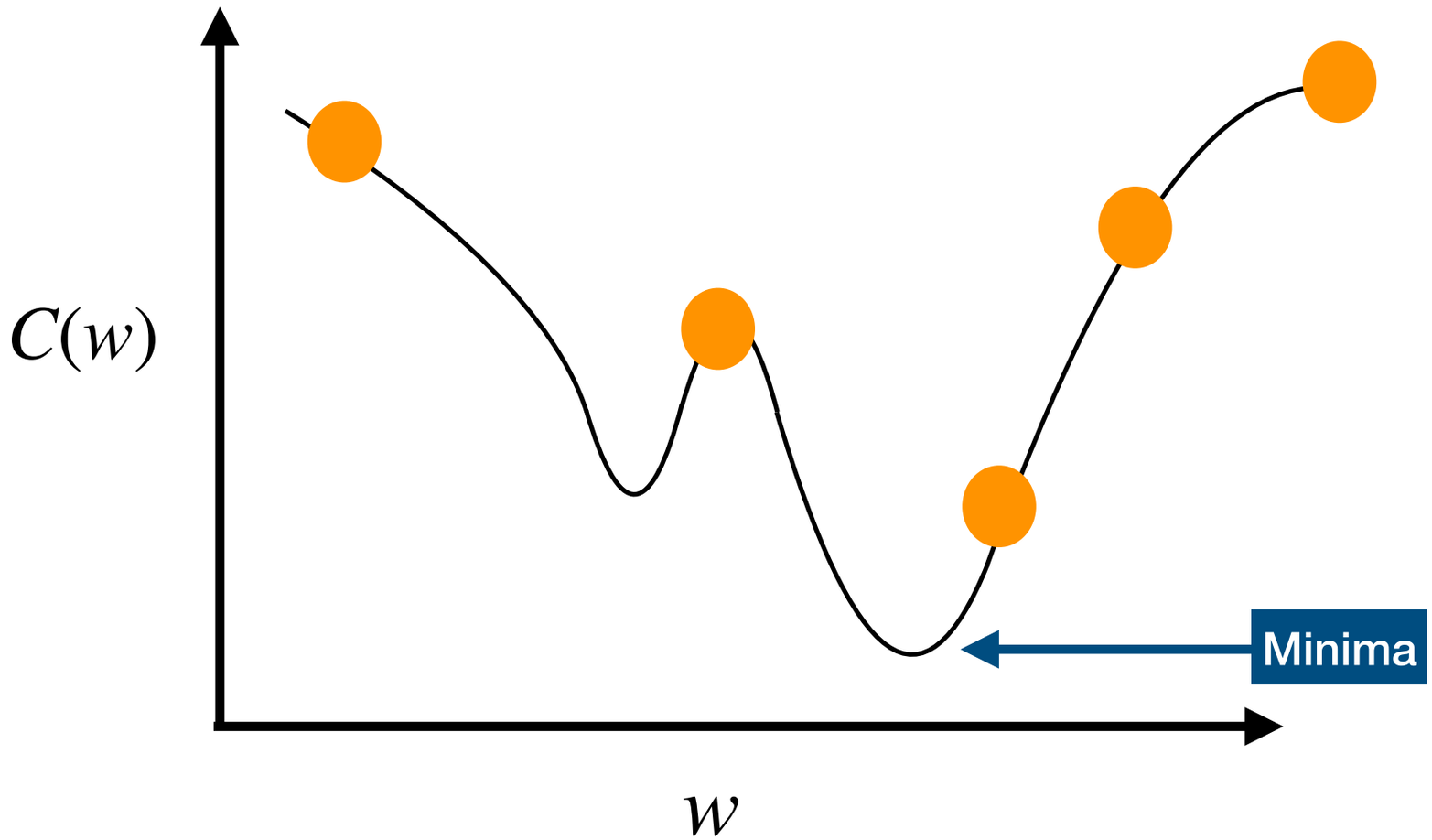


Visualising the cost/loss/objective function

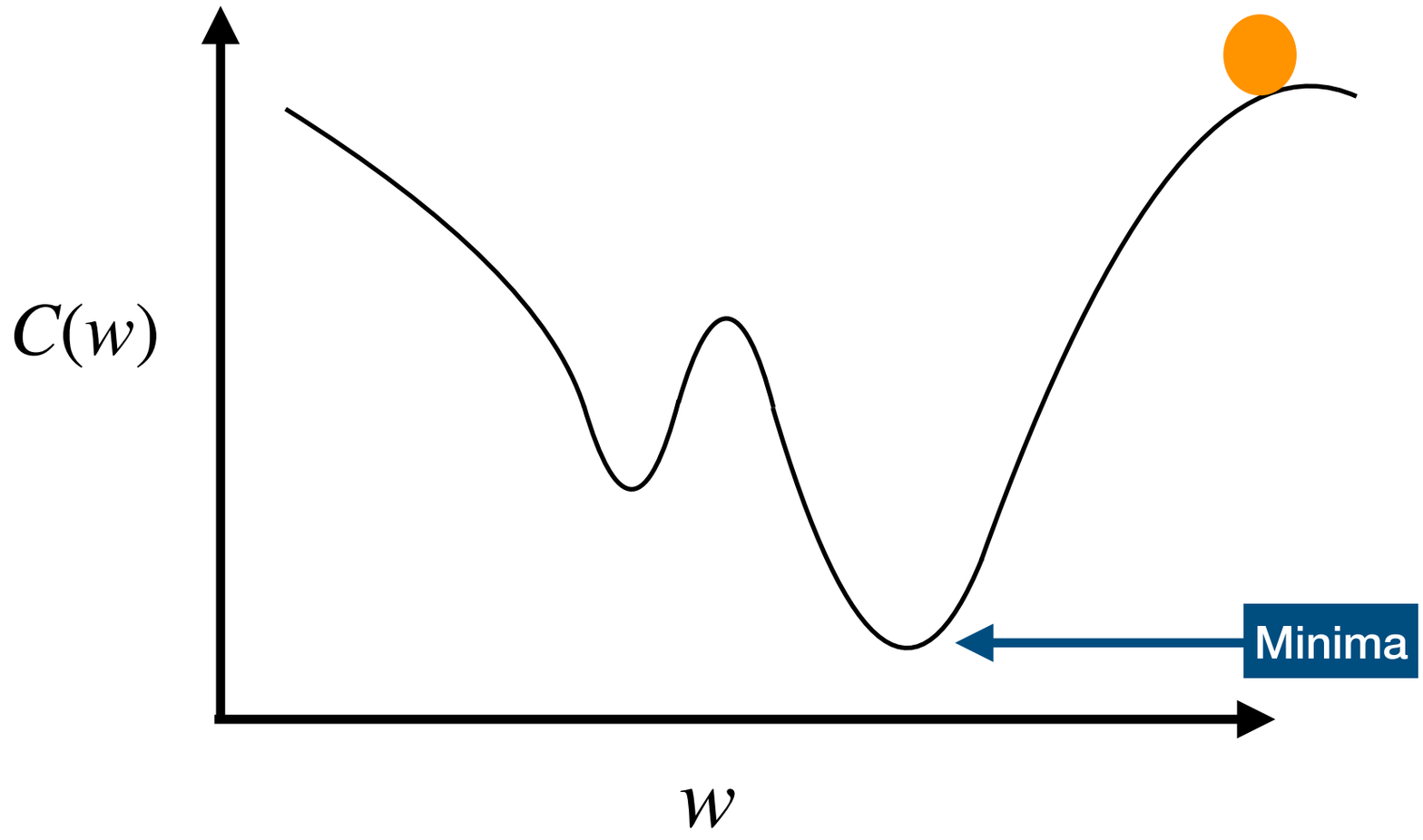
$$C(w,b) = f(w,y_j)\sum_{j=1}^{3} y_j \log(a_j)$$

Now, in reality the cost-function is an extremely high dimensional function usually. But, to simplify matters, let's think of it as a one dimensional function, dependant only on w

What is the initial value of the cost function?



How do we minimize the cost/loss function?

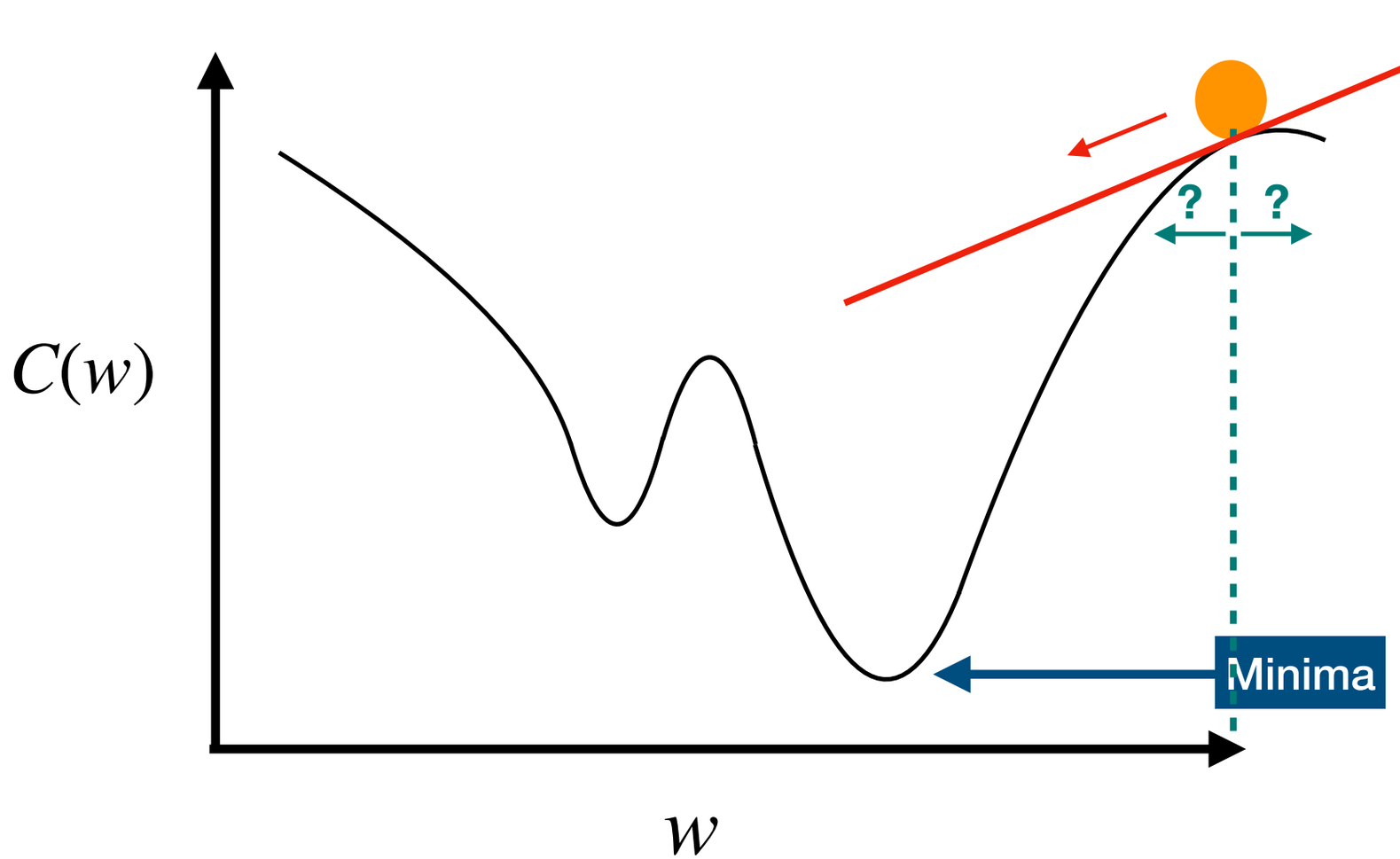


Essentially, what we have to replicate is a ball rolling down a hill!

How do we minimize the cost/loss function?

Essentially, what we have to replicate is a ball rolling down a hill!

How do we do this mathematically?

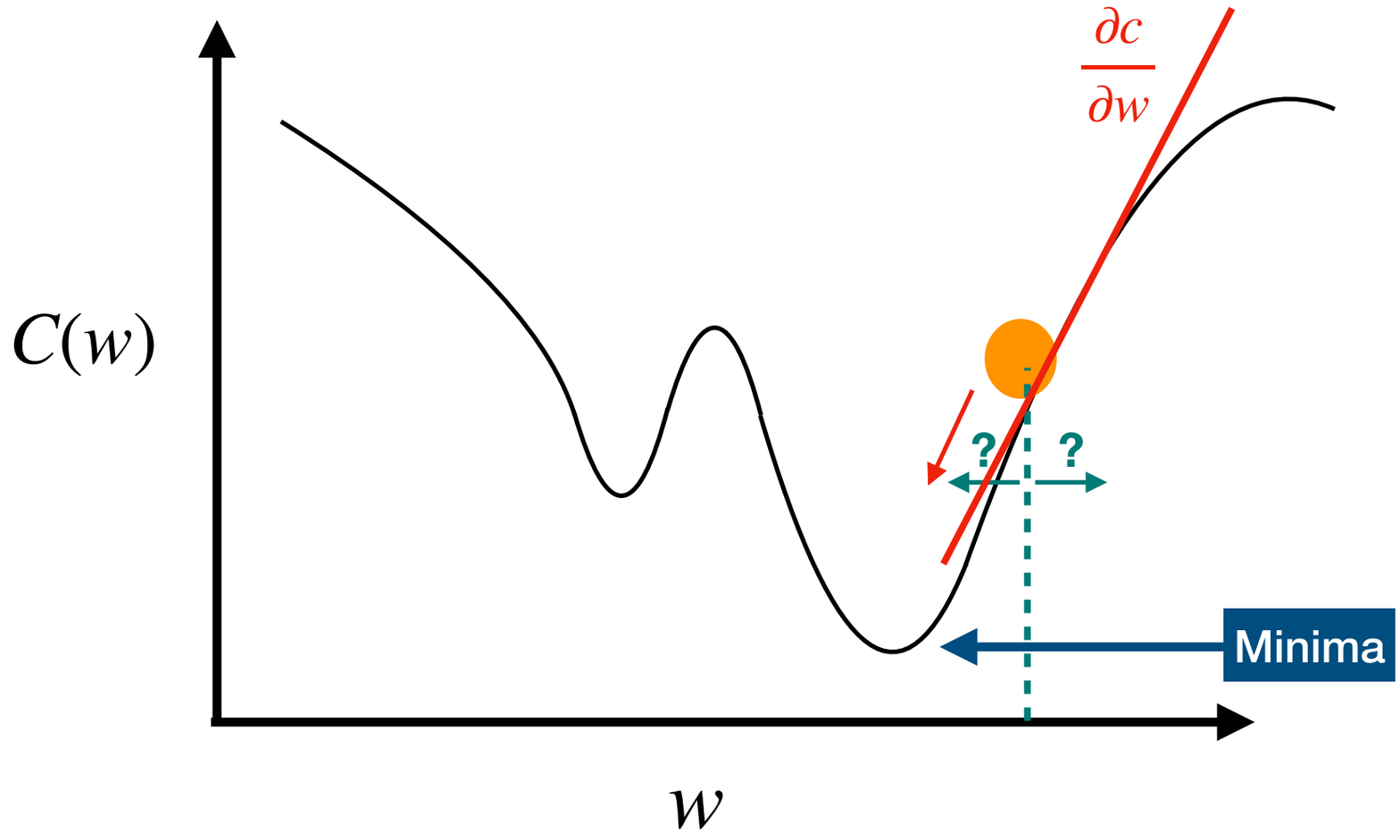


Question: What mathematical function can help us to decide which direction to move in?

How do we minimize the cost/loss function?

Essentially, what we have to replicate is a ball rolling down a hill!

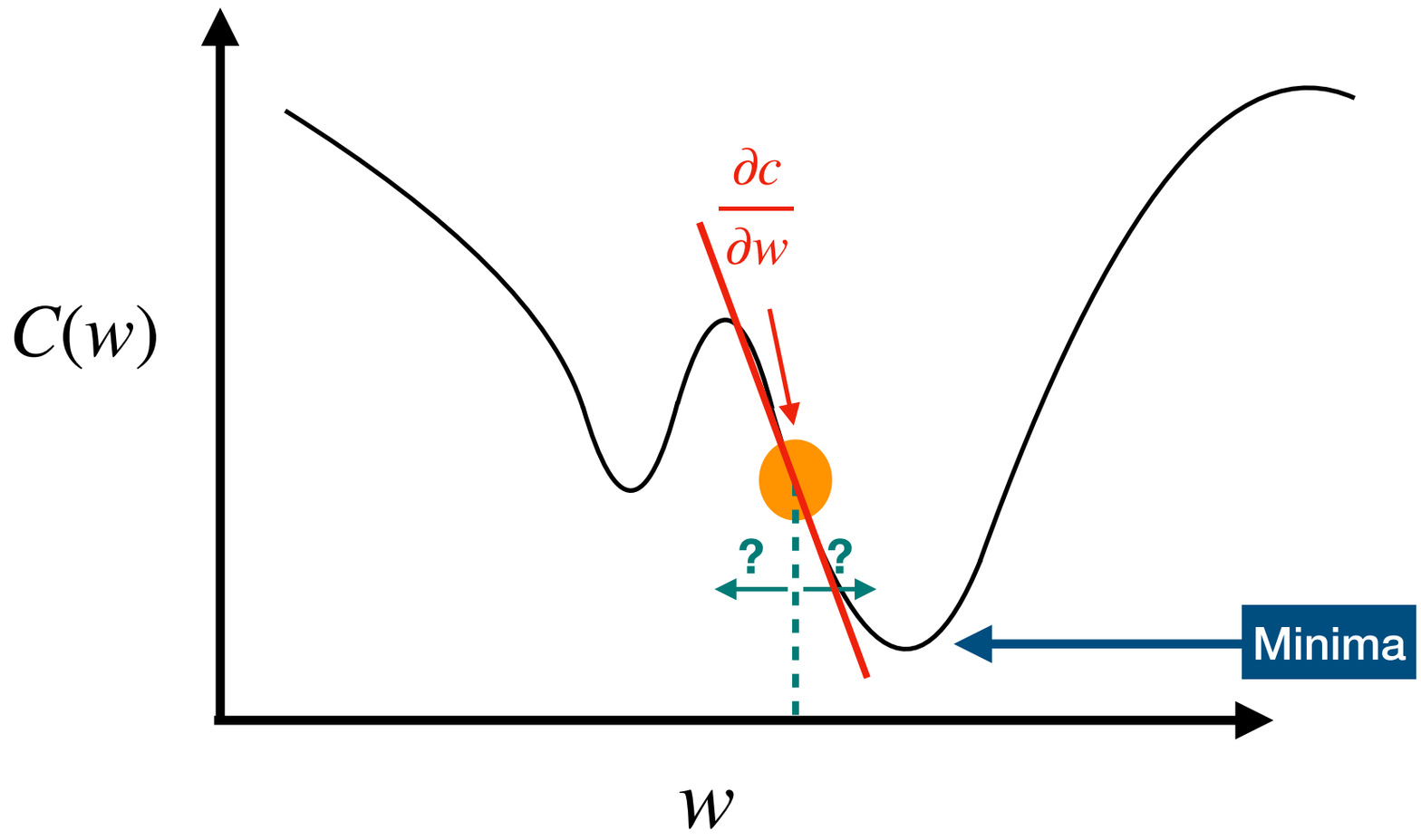
How do we do this mathematically?



How do we minimize the cost/loss function?

Essentially, what we have to replicate is a ball rolling down a hill!

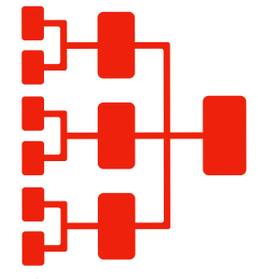
How do we do this mathematically?



Question: Along with calculating the derivative, what other factor was crucial in us reaching the local minima?

How much we move in each step!
This is also called the *learning rate*
This is one of the most important hyper-parameters in training a neural network!

Updating Weights & Biases



Output Label

Update model parameters

Cost / Loss Function

$$C(w_1, b_1, w_2, b_2, \ldots, w_m, b_m)$$

$$\Delta\nu = (\Delta w_1, \Delta b_1, \Delta w_2, \Delta b_2, \ldots, \Delta w_m, \Delta b_m)$$

Define small steps

$$\nabla C = (\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial b_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b_2}, \ldots, \frac{\partial C}{\partial w_m}, \frac{\partial C}{\partial b_m})$$

Calculate derivatives

η Learning Rate

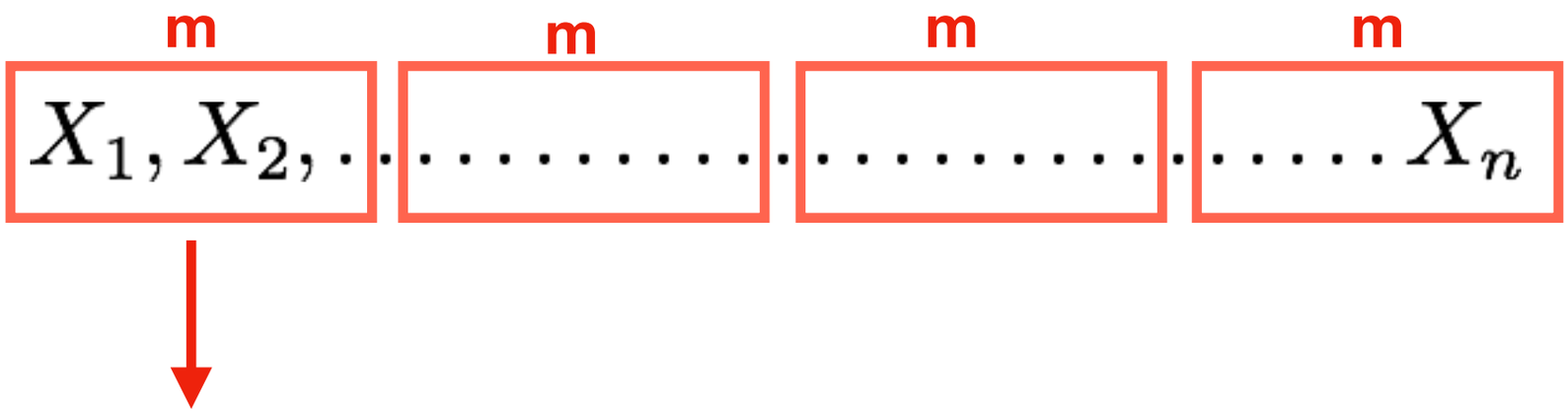
$$w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \qquad b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

Update weights & biases

Doing Gradident Descent in Batches

$$w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \qquad b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

Break Up your training set into batches of size m



One Epoch of Training

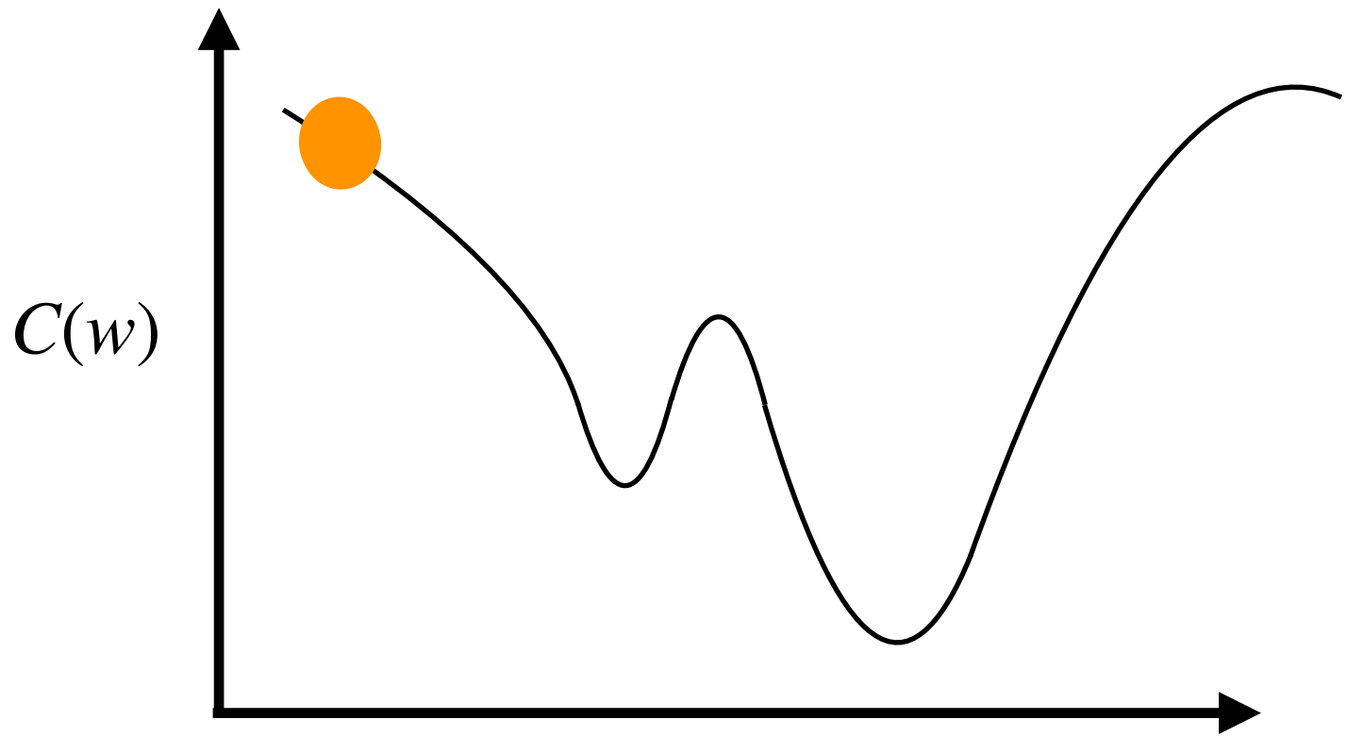
$$w'_k = w_k - \frac{\eta}{m} \Sigma_j \frac{\partial C_{X_j}}{\partial w_k} \qquad b'_l = b_l - \frac{\eta}{m} \Sigma_j \frac{\partial C_{X_j}}{\partial b_l}$$

Apply Gradient Descent after each batch & average over all samples in the batch

Question: What is the logic behind doing gradient descent in batches?

!Note! : Batch Size and Number of Epochs are both hyper-parameters!

Adding Some Momentum!



Adding momentum can help in avoiding getting stuck in notches!

$$w_k' = w_k - \eta \frac{\partial C}{\partial w_k} + \gamma v_w \qquad b_l' = b_l - \eta \frac{\partial C}{\partial b_l} + \gamma v_b$$

γ —> Momentum (another hyperparameter)
v —> Last Update to parameter

But adding momentum can have interesting effects on your learning algorithm
Head to https://distill.pub/2017/momentum/

The Backbone of Modern NN: Backpropagation

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

To calculate gradients, most modern ML frameworks use a technique called back propagation.

Although introduced in the 1970s, it's safe to say that most ML algorithms of today wouldn't work without this 4-page short paper by Rumelhart et. al. in 1986

CHAPTER 2

How the backpropagation algorithm works

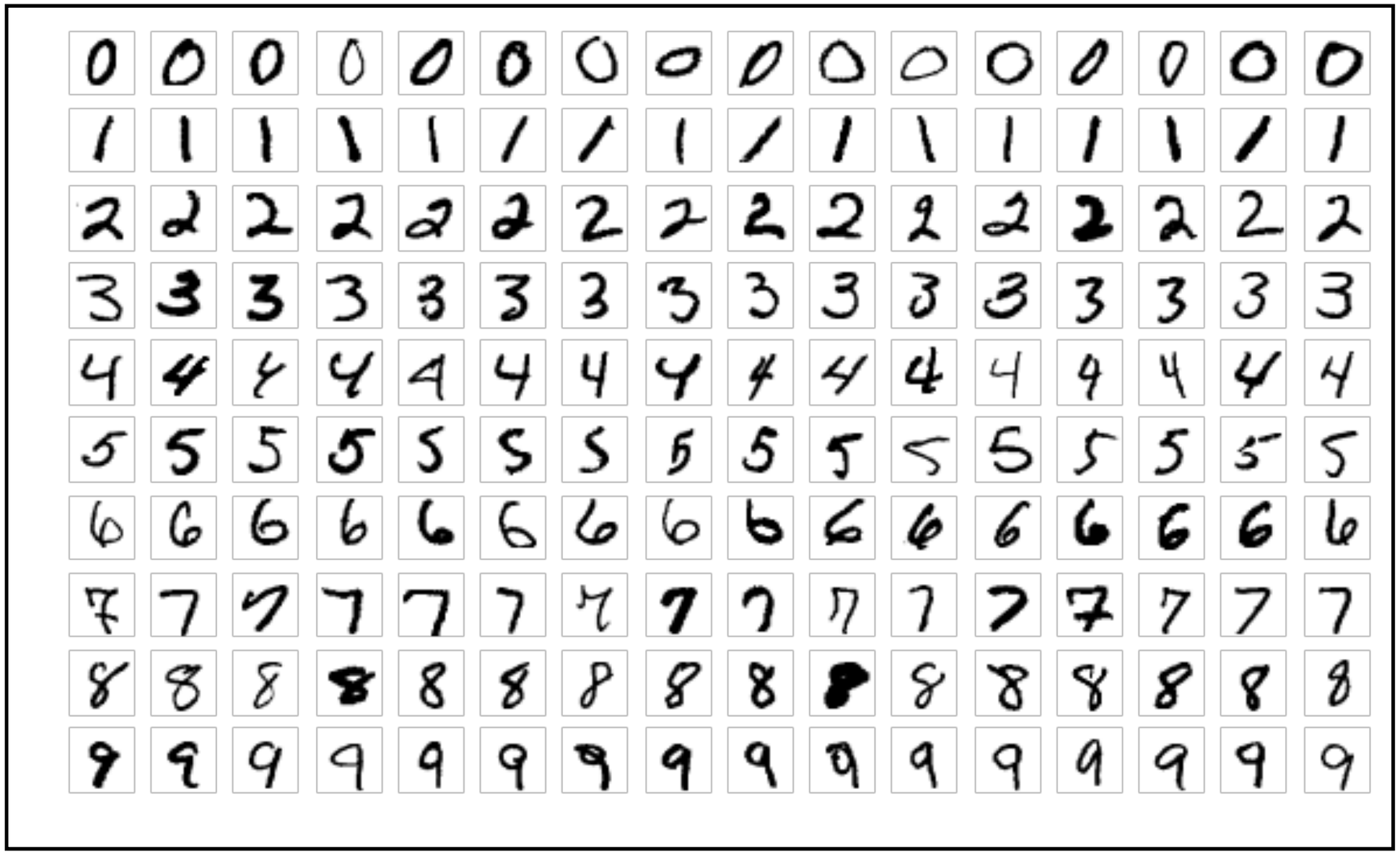
In the last chapter we saw how neural networks can learn their weights and biases using the gradient descent algorithm. There was, however, a gap in our explanation: we didn't discuss how to compute the gradient of the cost function. That's quite a gap! In this chapter I'll explain a fast algorithm for computing such gradients, an algorithm known as *backpropagation*.

The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams. That paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural nets to solve problems

Read more at neuralnetworksanddeeplearning.com

Lab: Building Our First Neural Network

Classifying MNIST Handwritten Digits!



These digits were written by census workers and was used to train the first generation of neural networks to detect zip codes!

Convolutional Neural Networks (CNNs) 101

A little demo: I will ask you to draw something. You will have 20 seconds!

0:[redacted]

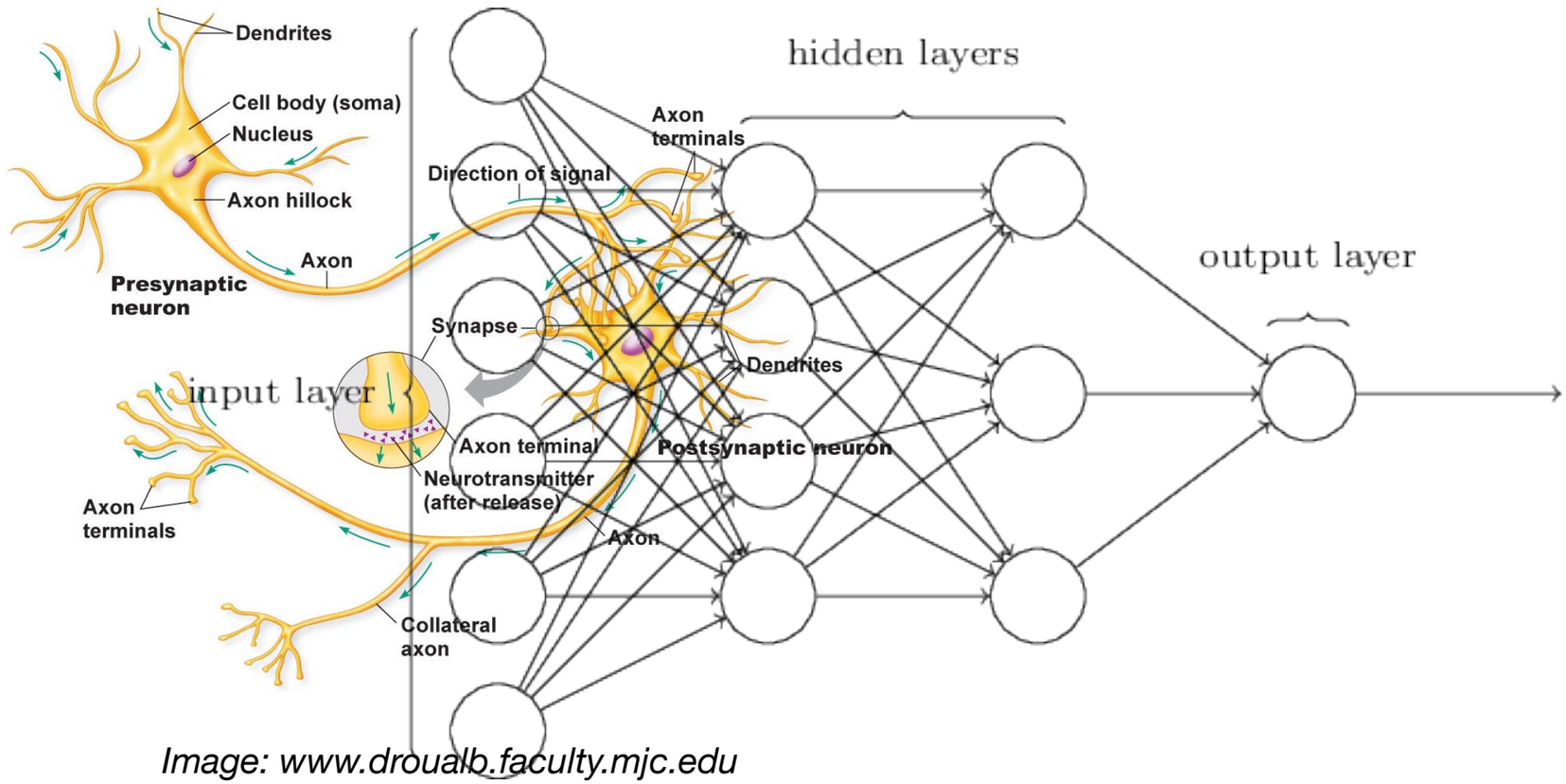
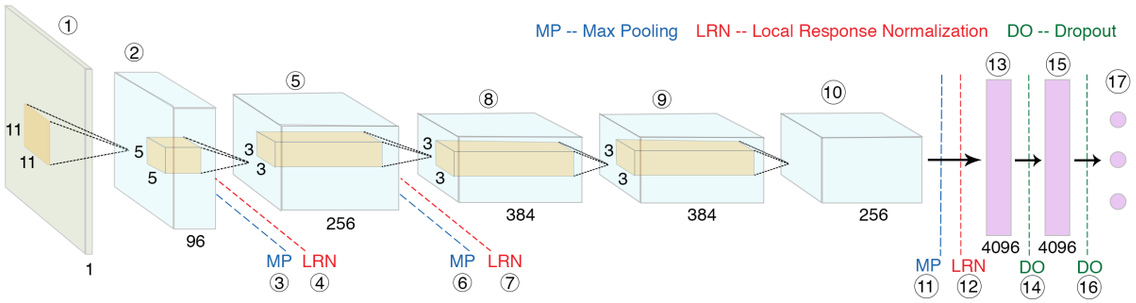
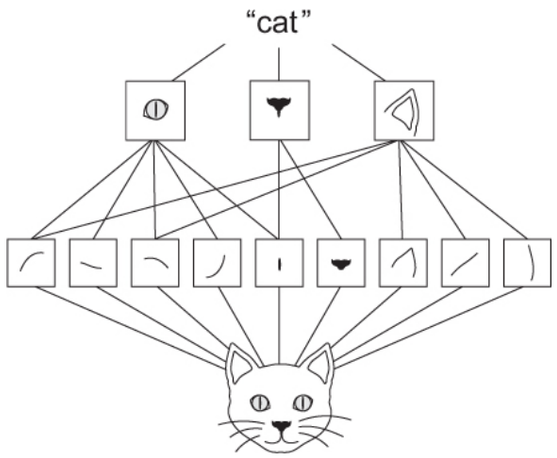


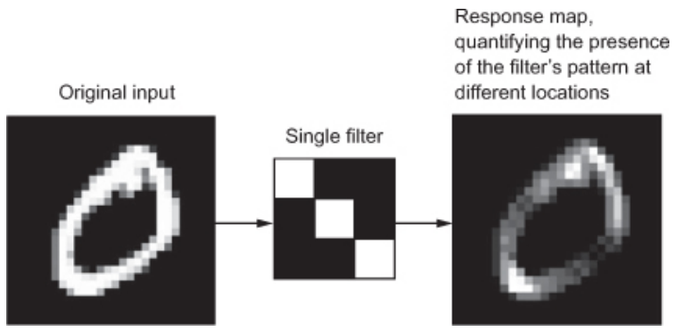
Image: www.droualb.faculty.mjc.edu

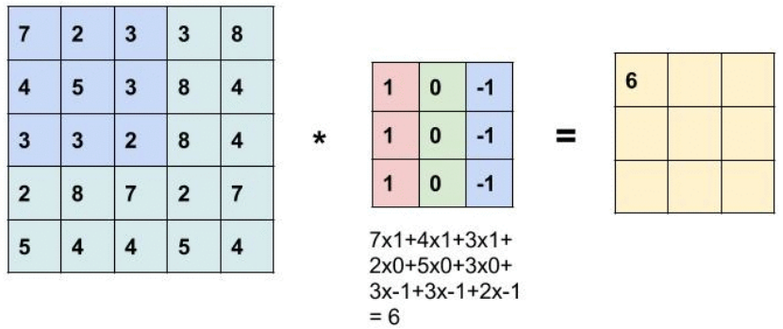
What is a convolutional layer?



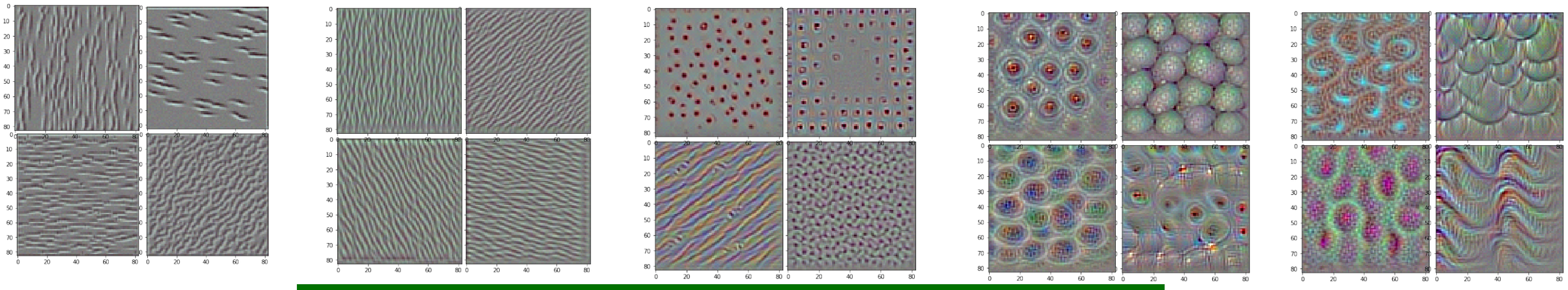
Learns to infer the presence of specific patterns in the image







Shallower Layers **Different Filters in GaMorNet** **Deeper Layers**



CNNs represent a hierarchical decision making process

Ghosh et. al. (2020)

Introducing Regularization!

Reminder: In every situation, we should favour the simplest possible model!

In terms of neural networks, this boils down to either using a smaller network (which might not be preferred) or using smaller weights in an existing network.

“Suppose our network mostly has small weights — the smallness of the weights means that the behaviour of the network won't change too much if we change a few random inputs here and there. That makes it difficult for a regularized network to learn the effects of local noise in the data. “ — *Nielsen*

L1 Regularization

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|$$

L2 Regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

Essentially, we penalize the network for using higher weights.

λ —> Regularisation Parameter (Hyperparameter)

n —> Number of Samples in the Training Set