Universidad Intercontinental de la Empresa

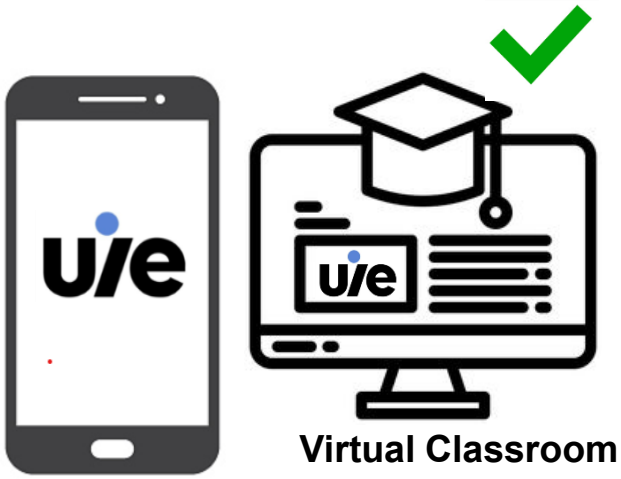
Asignatura

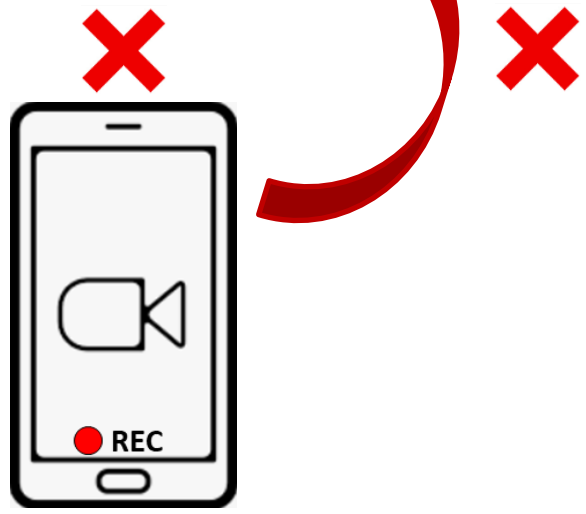
Advanced Machine Learning

Profesor

Alejandro Esteban Martínez


Recordar


Virtual Classroom UIE


REC


Recordar

Participate

Universidad Intercontinental de la Empresa

Sesión 03

Unidad I

Deep Learning

Tema 2

Deep Feed-Forward Neural Networks

Deep Learning

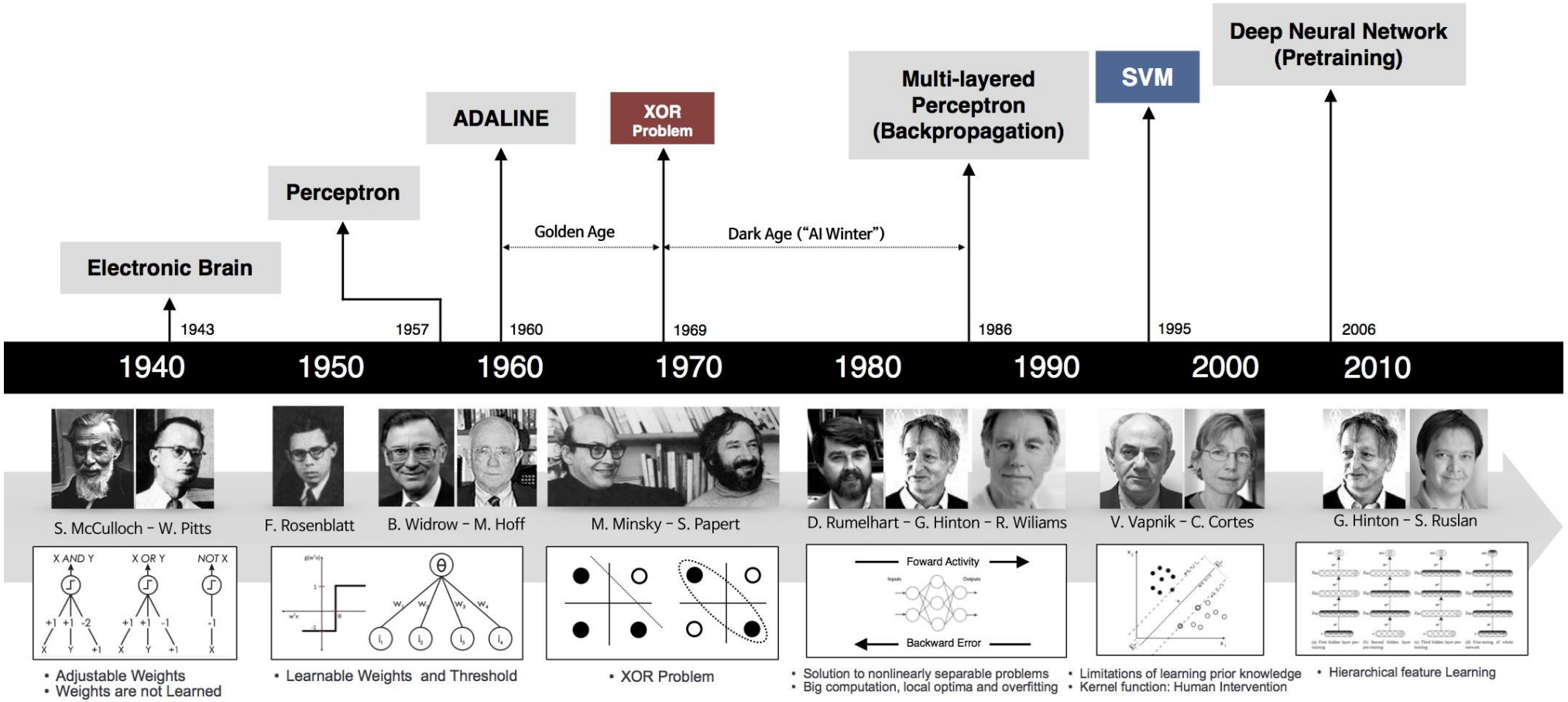
Session 03. Context

S03 **06/02/2025**	**Unit I Deep Learning** 1.2 Deep Feed-Forward Neural Networks	LO01 LO06 LO07	Review of the document: Sesion3.pdf	Active Exposition of Session3.pdf Practical activity	Review the after_session_3.pdf

At last class we saw…

Deep Learning

Fundamentals of Deep Learning

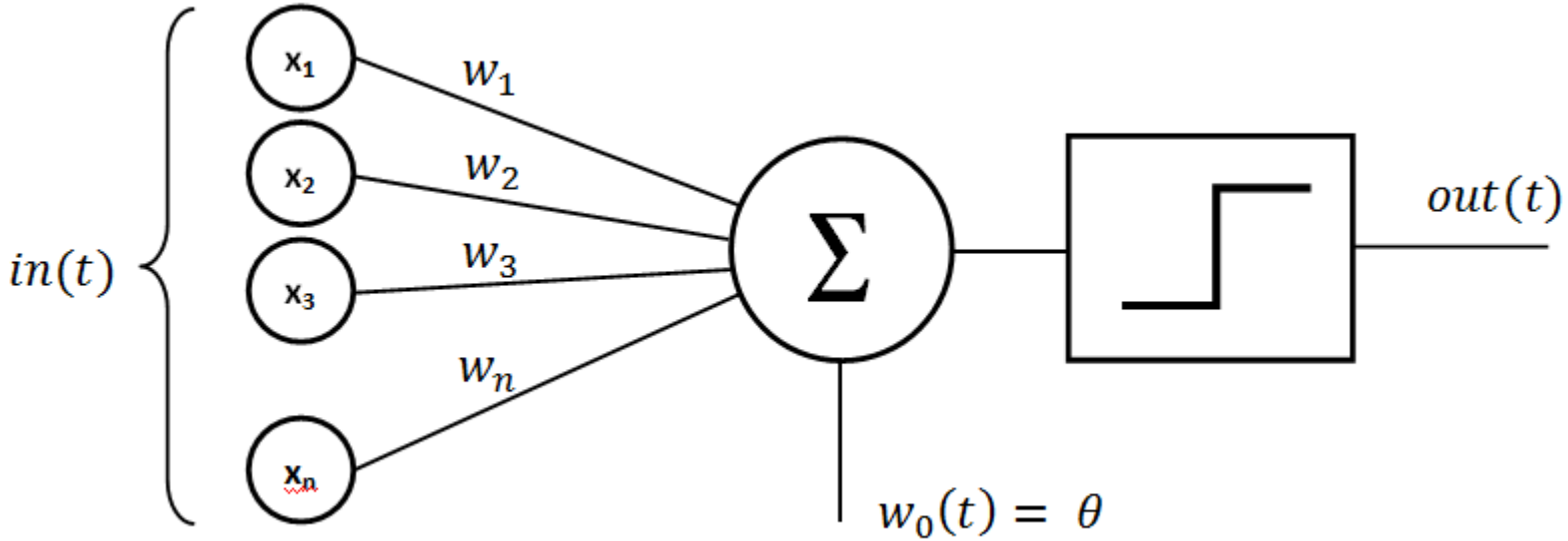


Lets check the assignments!

Deep Learning

1950s-1970s: Birth of AI and first developments

Alan Turing creates the Turing test to evaluate for intelligent reasoning in machines and the Perceptron is developed in 1958 by F. Rosenblatt which can adjust the weights and **learn** from experience, but it is quickly followed by an AI Winter because it is not possible to solve nonlinear problems (Such as the **XOR problem**)



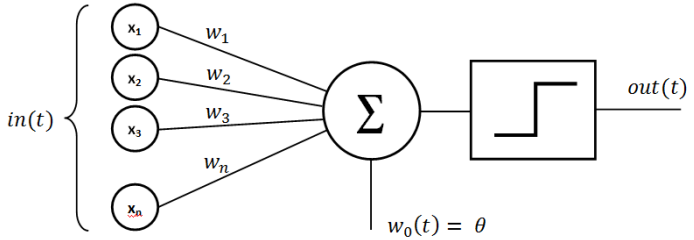
Symbol	A	B	Q
	0	0	0
	1	0	1
	0	1	1
	1	1	0
Q = A ⊕ B			

Today we will talk about…

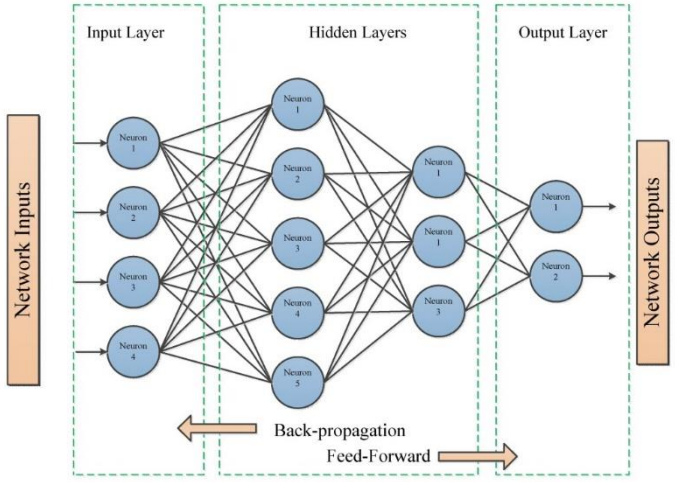
Deep Learning

Deep Feed-Forward Neural Networks

- The perceptron
  - What is it
  - How to train it (Forward Propagation)
  - Problems
  - Implementation



- The Multilayer Perceptron
  - What is it
  - How to train it (Back propagation and gradient descent)
  - Implementation

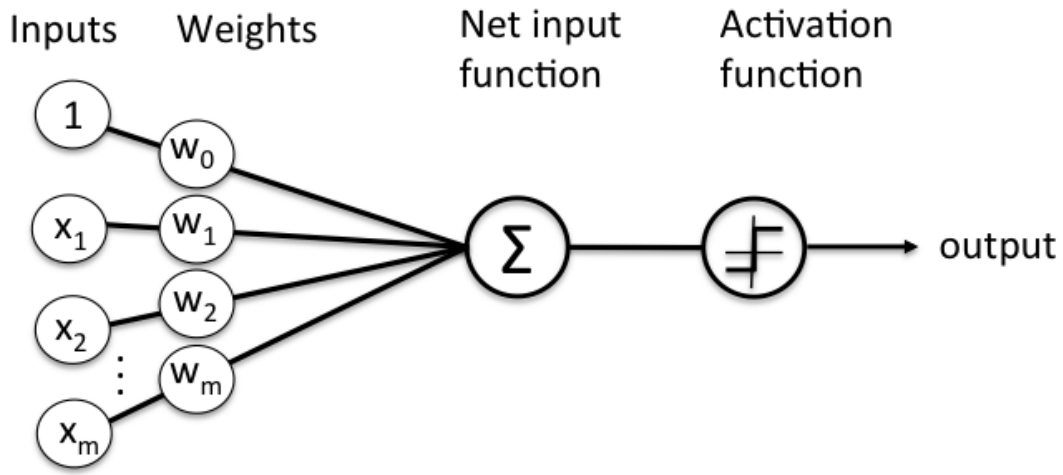


What even is a Perceptron?

Deep Learning

What is a perceptron?

It is the building block of every deep learning model today (Basically what we call a neuron).
Every model has thousands of them and even models like GPT-3.5 are estimated to have around 60B neurons.



$$y = f(i = \sum_{i=1}^{n} w_i \cdot x_i + b)$$

Perceptron Function

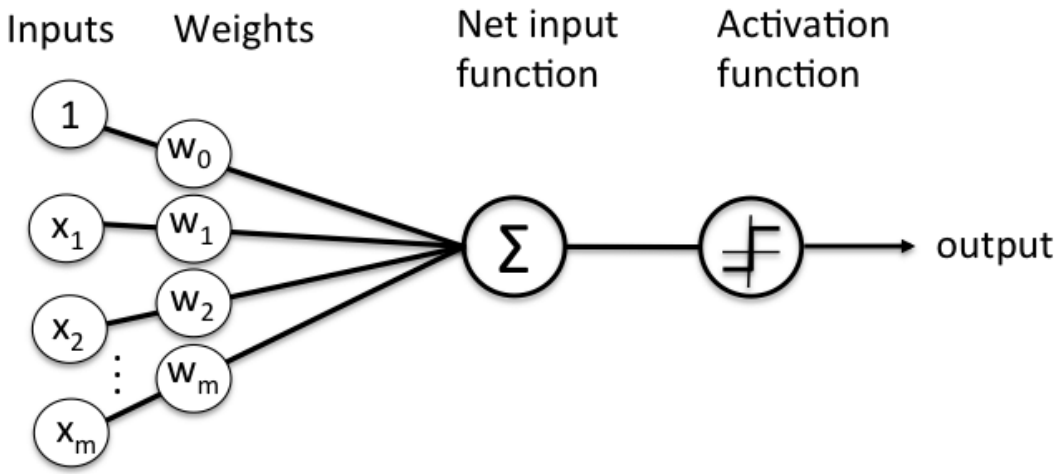
Deep Learning

Let's explain the formula

The sum $\sum_{i=1}^{n} w_i x_i$ is actually a matrix multiplication (dot product) $W \cdot X$ Where $W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$ and $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

f represents the activation function

b represents the bias that will allow our activation function to shift the decision boundary b

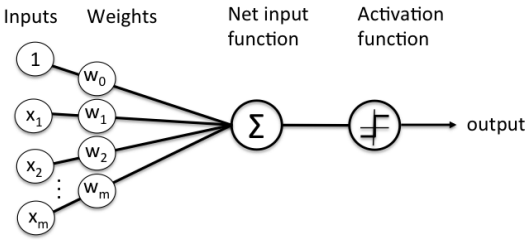


$$y = f(i = \sum_{i=1}^{n} w_i x_i + b)$$

Perceptron Function

Deep Learning

How does it learn?



Initialize weights to zeros
Initialize bias to zero

FOR each epoch:
    FOR each sample (x, y) in training data:
        Compute net input: output = dot_product(x, weights) + bias
        Apply activation function: prediction = 1 if output > 0 else 0
        Compute error: error = y - prediction
        Update weights: weights = weights + (error * x)
        Update bias: bias = bias + error

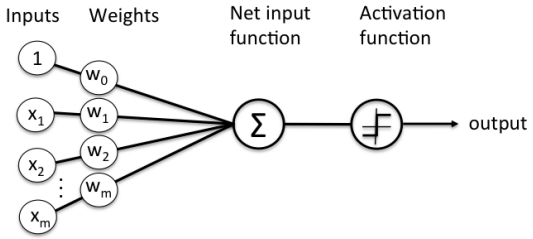
RETURN trained weights and bias

$$y = f(i = \sum_{i=1}^{n} w_i x_i + b)$$

Perceptron Function

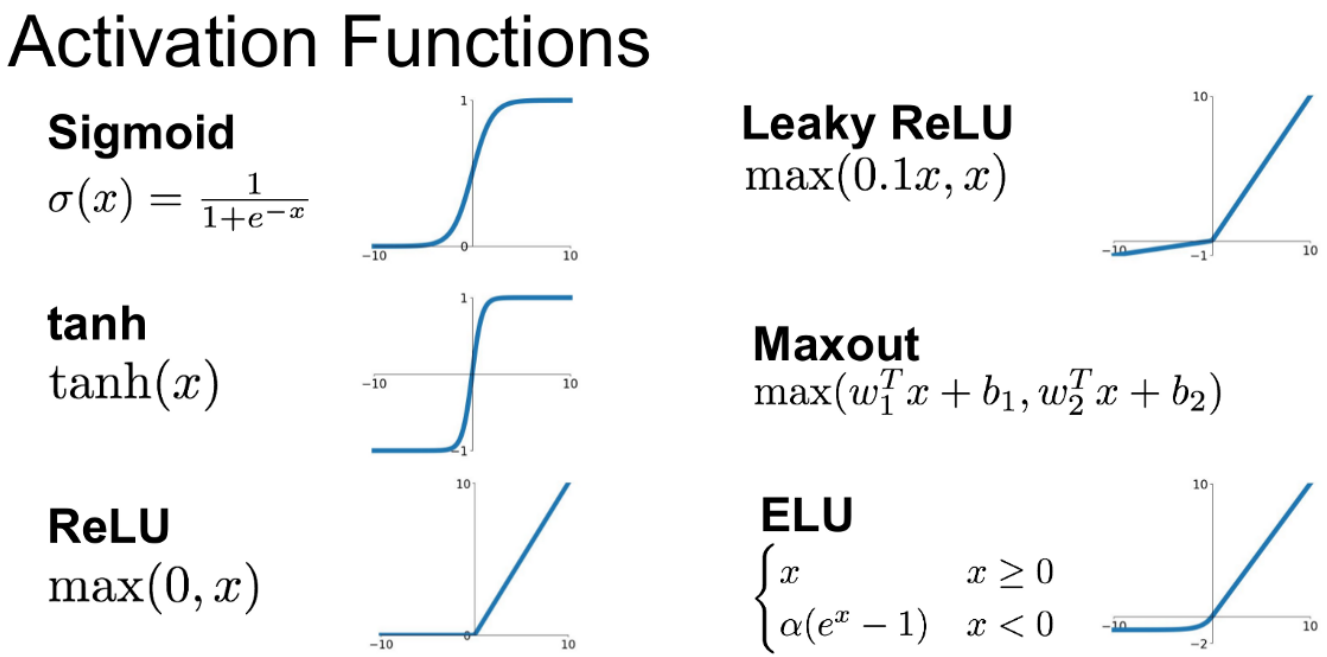
Deep Learning

What is an activation function?



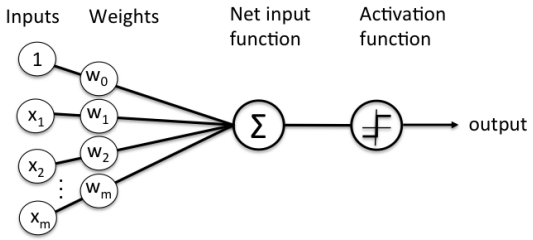
It's a transformation applied to the output of our neuron that allows nonlinearity to be expressed with the neural network. In the case of single-layer Perceptrons, as they weren't capable to capture nonlinear relations, the activation function was normally the step function $f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$

The explanation of this will be given with the introduction of the Multilayer Perceptron



Deep Learning

How does it learn?



Initialize weights to zeros
Initialize bias to zero

FOR each epoch:
    FOR each sample (x, y) in training data:
        Compute net input: output = dot_product(x, weights) + bias
        Apply activation function: prediction = 1 if output > 0 else 0
        Compute error: error = y - prediction
        Update weights: weights = weights + (error * x)
        Update bias: bias = bias + error

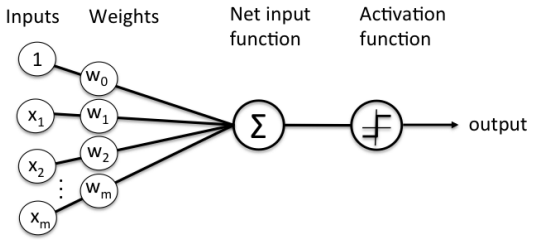
RETURN trained weights and bias

Solve the OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Deep Learning

How does it learn?



Initialize weights to zeros
Initialize bias to zero

FOR each epoch:
    FOR each sample (x, y) in training data:
        Compute net input: output = dot_product(x, weights) + bias
        Apply activation function: prediction = 1 if output > 0 else 0
        Compute error: error = y - prediction
        Update weights: weights = weights + (error * x)
        Update bias: bias = bias + error

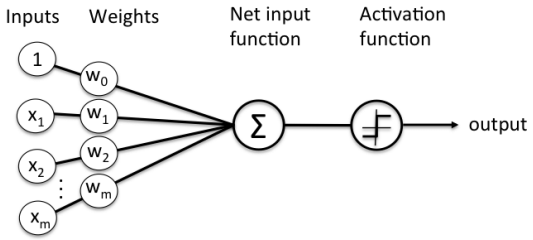
RETURN trained weights and bias

Solve the AND gate

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Deep Learning

How does it learn?



Initialize weights to zeros
Initialize bias to zero

FOR each epoch:
    FOR each sample (x, y) in training data:
        Compute net input: output = dot_product(x, weights) + bias
        Apply activation function: prediction = 1 if output > 0 else 0
        Compute error: error = y - prediction
        Update weights: weights = weights + (error * x)
        Update bias: bias = bias + error

RETURN trained weights and bias

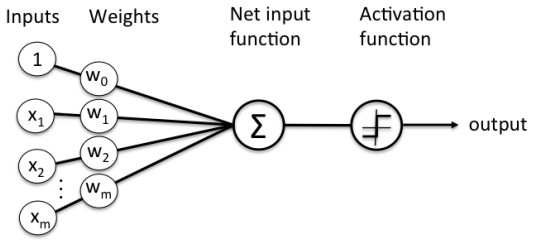
What happens with the XOR gate?

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0


single neuron

Deep Learning

How do we implement the Perceptron?

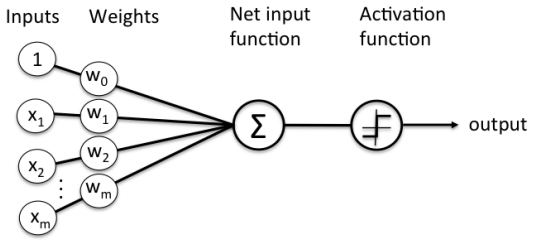


QUICK! Think of a simple implementation of a perceptron in Python, what would we need?

- A prediction (Forward) function
- A training function
- An error (loss) function

Deep Learning

How do we implement the Perceptron?



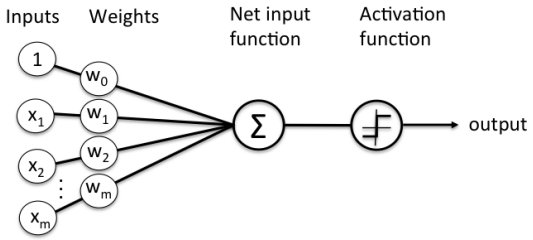
QUICK! Think of a simple implementation of a perceptron in Python, what would we need?

- A prediction (Forward) function
- A training function
- An error (loss) function

```
# The Training Loop (Perceptron)
for x, y in training_data:

    # 1. Forward Pass (Prediction)
    # Hint: w * x + b
    prediction = _____________________________

    # 2. Calculate Error
    # Hint: target - prediction
    error = _____________________________________

    # 3. Backward Pass (Update Weights)
    # Hint: weights + (error * input)
    weights += _____________________________
    bias    += _____________________________
```

Deep Learning

How do we implement the Perceptron?



QUICK! Think of a simple implementation of a perceptron in Python, what would we need?

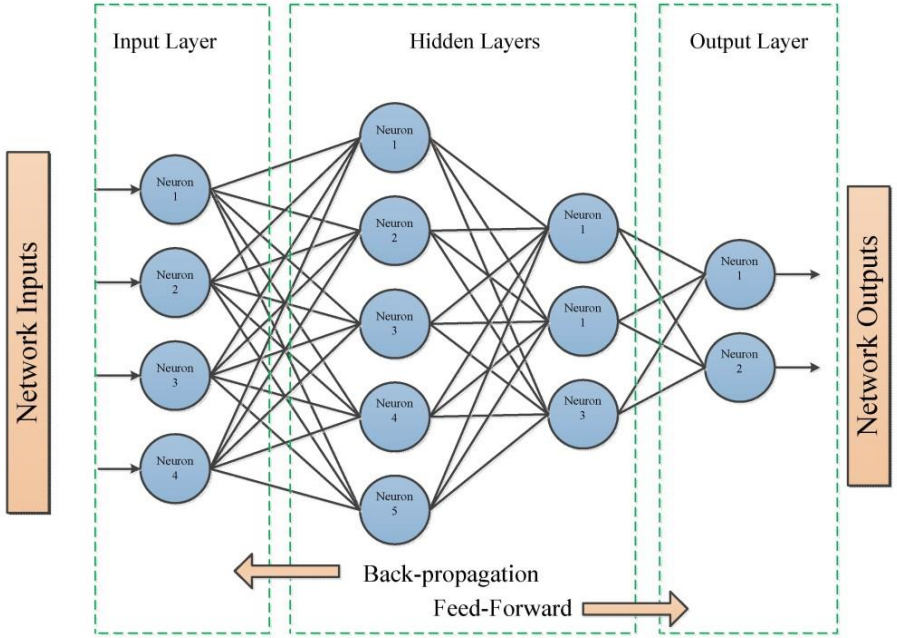
- A prediction (Forward) function
- A training function
- An error (loss) function

Rest of 20 Minutes

Deep Learning

1980s-1990s: Development of the Multilayer Perceptron

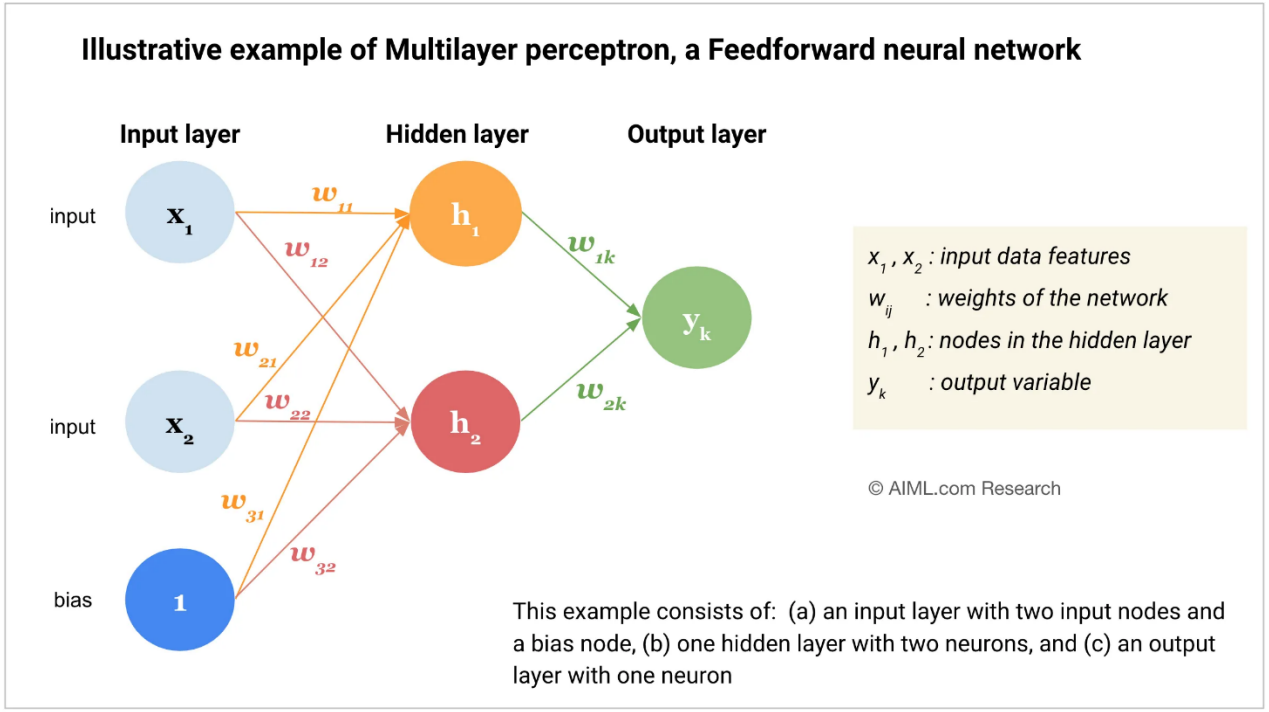
The XOR problem is solved with the development of **Backpropagation** in 1986, that allows for the creation of the Multilayer Perceptrons that can solve nonlinear problems.



Deep Learning

What is the Multilayer Perceptron?

As the name says, it is just a Perceptron that has multiple layers, but each layer actually works as the output of the previous and the input of the next one.
It is formed as multiple Perceptrons all densely connected with each other into multiple layers.

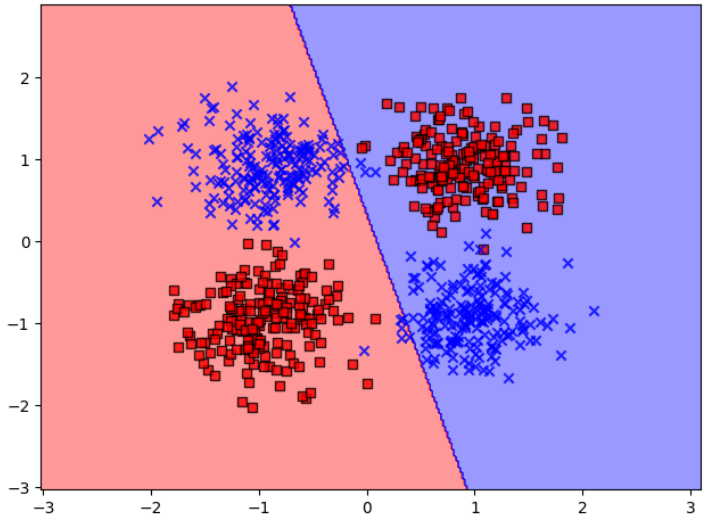


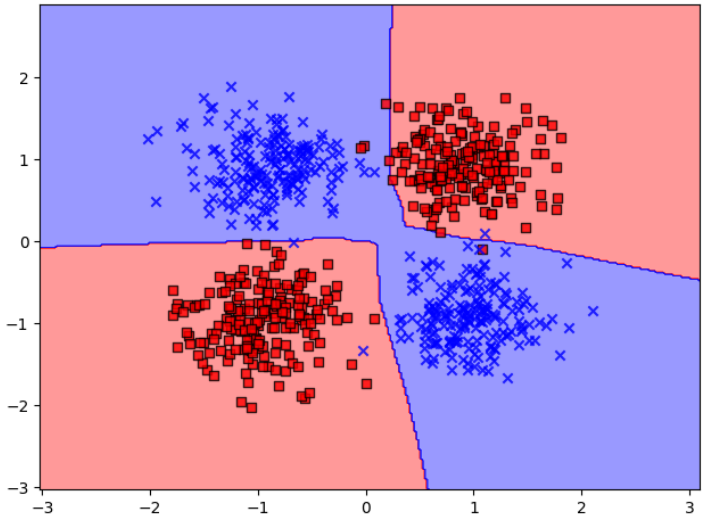
Deep Learning

Why do we really need more layers?

We have seen that there are some really interesting problems that could be solved by a single-layer perceptron (Like to detect tanks in images) but the problem is that the world is normally not linearly separable. A Perceptron could never understand for example that tuna and chocolate shouldn't go together (example of the XOR problem)

But now with more layers (and the use of non-linear activation functions) we can solve this problems!

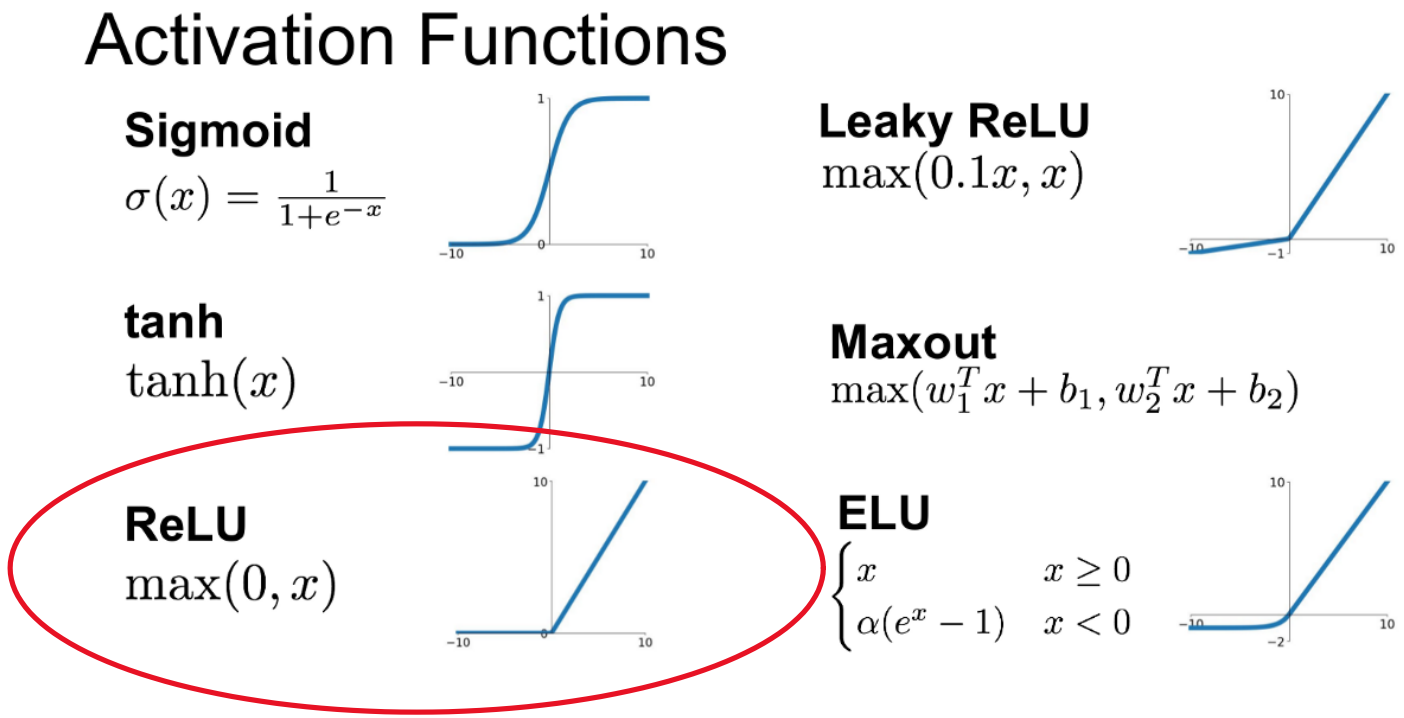




Deep Learning

What is an activation function?

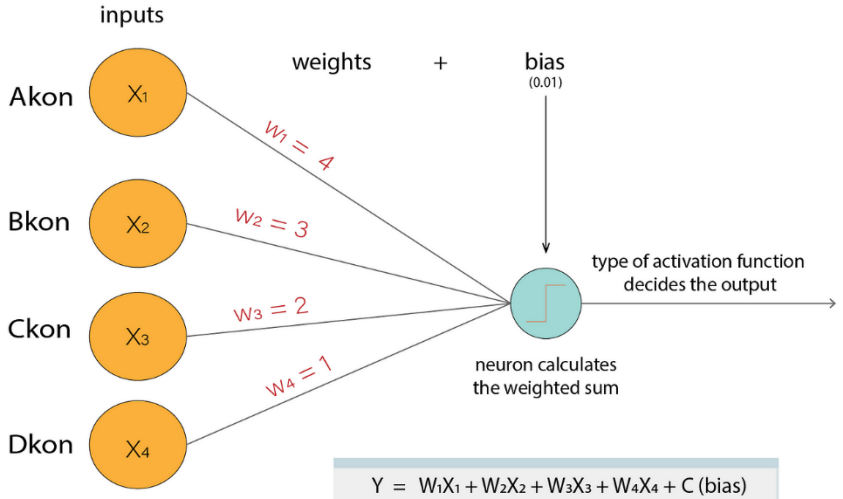
So why do we use them now?
This allows us the “bend” the decision boundary, so now instead of having a straight line we can have complex relationships.
Take into account the the linear transformation of a linear function will just give us another linear function, so we need nonlinear transformations (activation functions) to approach nonlinear problems.



Deep Learning

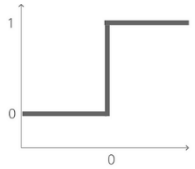
What is an activation function?

So why do we use them now?
This allows us the “bend” the decision boundary, so now instead of having a straight line we can have complex relationships.



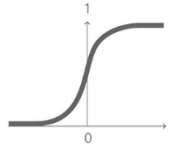
$Y = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + C \text{ (bias)}$

Thailand	(4x-9) + (3x10) + (2x3) + (1x-2)	=	- 2
Jamaica	(4x-4) + (3x5) + (2x-1) + (1x8)	=	5
Dubai	(4x-3) + (3x6) + (2x-2) + (1x6)	=	8



Step

Thailand: 0
Jamaica: 1
Dubai: 1



Sigmoid
(Probabilities)

Thailand: 0.002
Jamaica: 0.85
Dubai: 0.94

*probabilities do not add to 1

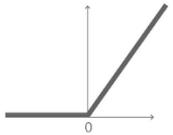
Softmax
(Probabilities)

Thailand: 0.02
Jamaica: 0.22
Dubai: 0.76

*probabilities add to 1

tanh

Thailand: -0.2
Jamaica: 0.5
Dubai: 0.8



ReLU

Thailand: 0
Jamaica: 5
Dubai: 8

Deep Learning

Universal Approximation Theorem

The Universal Approximation Theorem states that a feedforward neural network with a single hidden layer and a finite number of neurons can approximate any continuous function on a compact subset of the real numbers, given an appropriate activation function.

However, the theorem doesn't prescribe how to find the right weights or how long it will take to train a neural network for a given problem.

Deep Learning

And how do we train it?

We have multiple problems now, with more layers also comes more responsibility and that means that training becomes harder. How can we now actualize the weight values of the hidden layers?
We need to Propagate the Error **Backwards,** and we do this with Calculus!

Back-Propagation Equations:

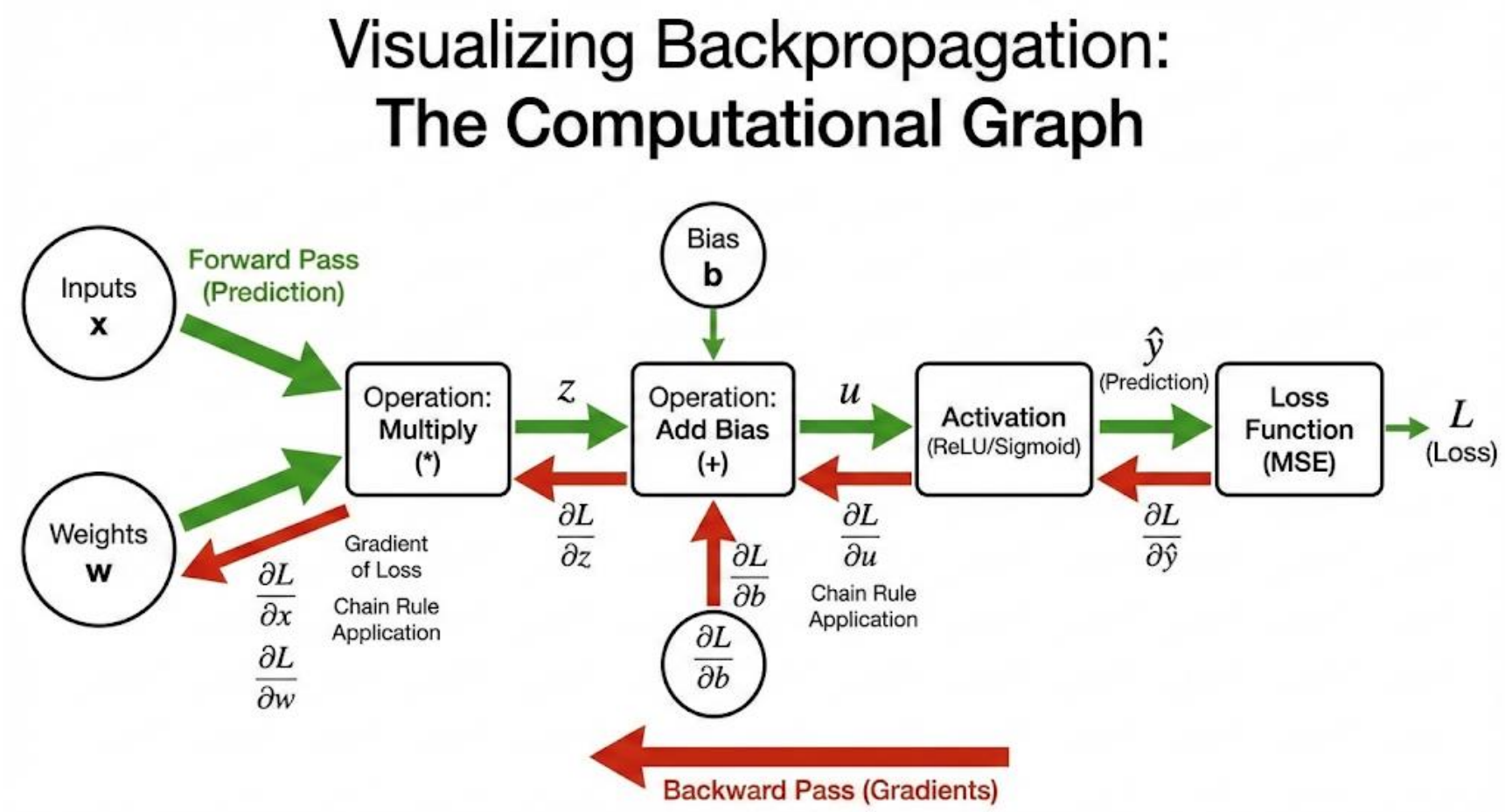
$$\frac{\partial L}{\partial \hat{y}} = \frac{1}{2} * 2(y - \hat{y}) * -1 = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z} = \sigma(z) * \big(1 - \sigma(z)\big) = \hat{y} * (1 - \hat{y})$$

$$\frac{\partial z}{\partial w_7} = \frac{\partial (h_1 w_7 + h_2 w_8 + h_3 w_9)}{\partial w_7} = h_1$$

Deep Learning

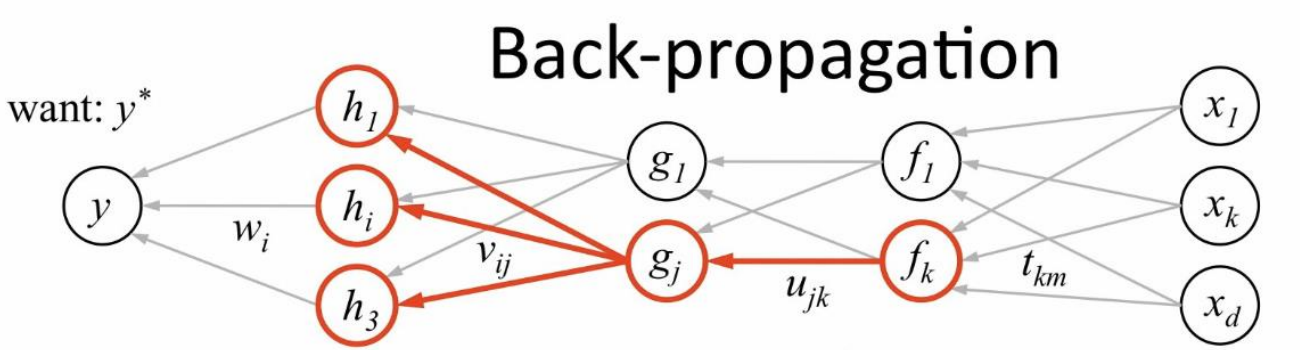
And how do we train it?



Deep Learning

And how do we train it?

Ok, not like that because this is an applied degree, but I do want you to understand why we need it and how we use it



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y\text{-}y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\frac{\partial E}{\partial g_j} = \sum_i \sigma'(h_i) v_{ij} \frac{\partial E}{\partial h_i}$$

should g_j be higher or lower? | how h_i will change as g_j changes | was h_i too high or too low?

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \sigma'(g_j) f_k$$

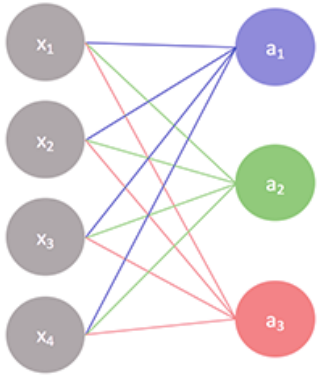
do we want g_j to be higher/lower | how g_j will change if u_{jk} is higher/lower

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

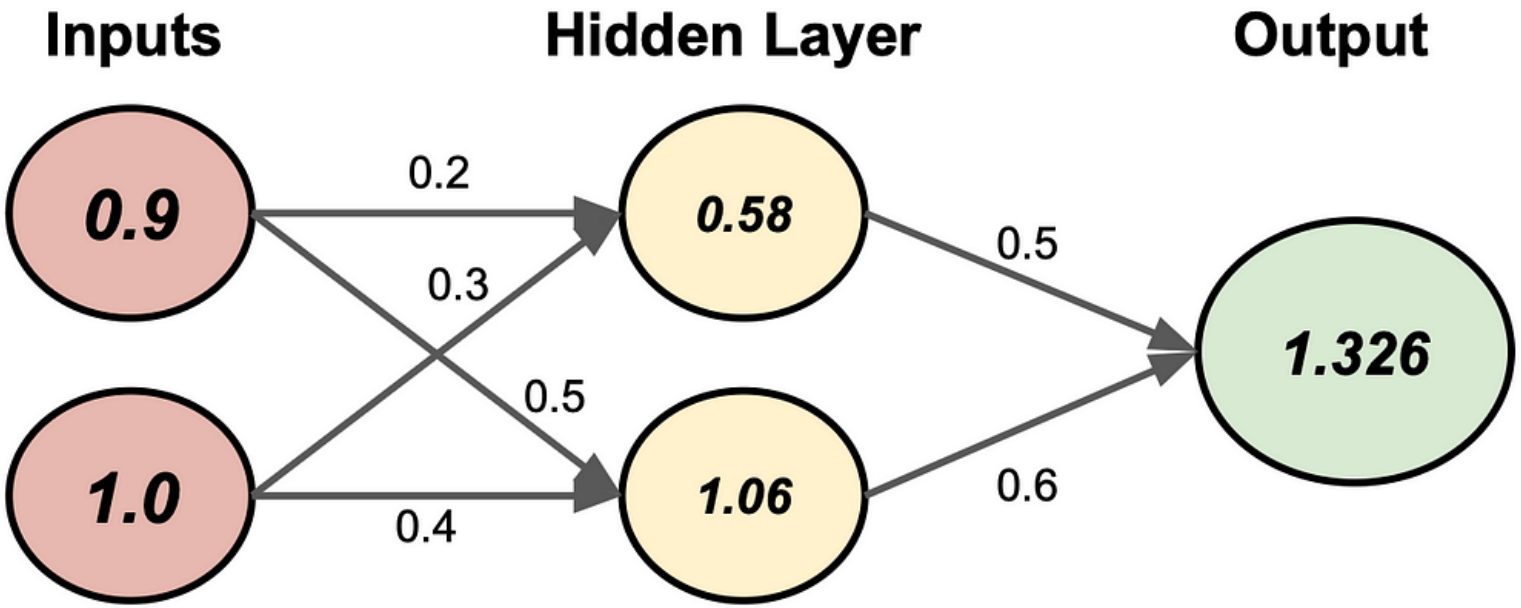
Deep Learning

1) Forward Pass



$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

We already know this, it is just more complicated but, in the end, it is still just matrix multiplications.
Lots of them, that's why for deeper neural networks GPU computing is a must.



Deep Learning

2) Calculate the error

In the field of ML we are used to call it Loss, and depending of the type of problems we can have different functions to calculate it (Because now we are going to train in Batches to be efficient).

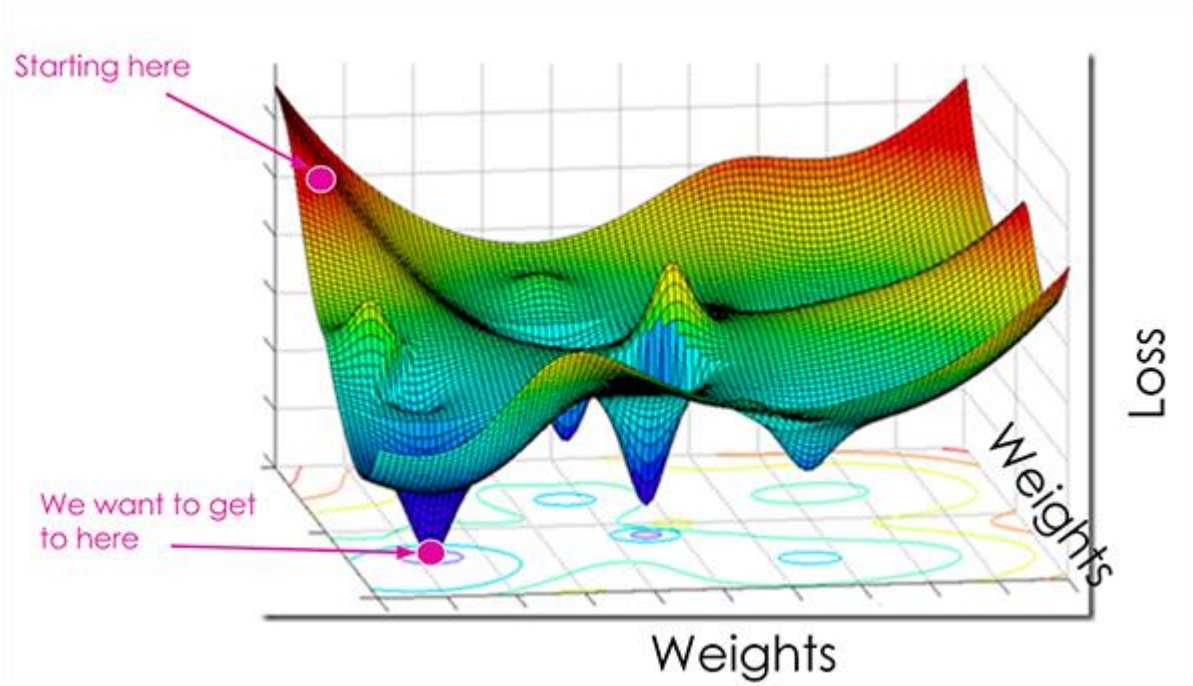
- Mean Squared Error $$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$ This will be used for Regression problems

- Cross-Entropy Loss $$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$ This will be used for Binary Classification

Deep Learning

3) Back-Propagate the error (With gradient descent)

Now we want to be able to adjust the total weights to minimize the Loss, how? We calculate the gradient of our loss function and *Descend* in the opposite direction (This is just with 2 Weights)



So in this case we will need the use of partial derivatives to compute the slope of the Loss function in relation to one weight at a time and finally get how we need to change our weights to decrease a tiny bit our loss (depending on the learning rate)

With this we can get the literal changes that should be applied to each weight to fastest decrease the Loss in a given layer.

Deep Learning

3) Back-Propagate the error (With gradient descent)

Now we just have to propagate the changes backwards to the following layer with the Chain Rule

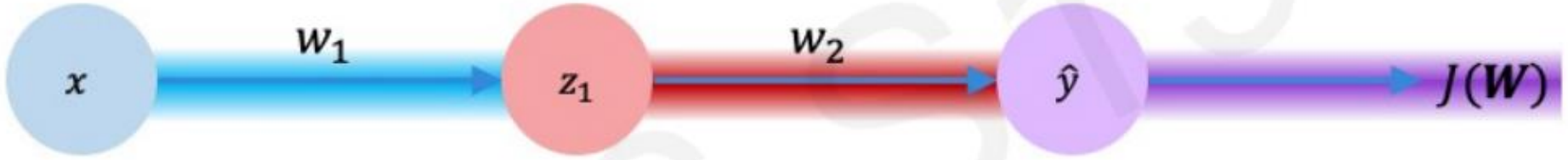
$$\frac{d}{dx}\left[\left(f(x)\right)^{n}\right] = n\left(f(x)\right)^{n-1} \cdot f'(x)$$

$$\frac{d}{dx}\left[f\left(g(x)\right)\right] = f'\left(g(x)\right)g'(x)$$

Deep Learning

3) Back-Propagate the error (With gradient descent)

Now we just have to propagate the changes backwards to the following layer with the Chain Rule



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

*Repeat this for **every weight in the network** using gradients from later layers*

Deep Learning

Now we know how to train an MLP

Build one and train it!

Deep Learning

Now we know how to train an MLP

But how do we optimize it now? Because a simple MLP can have multiple problems:

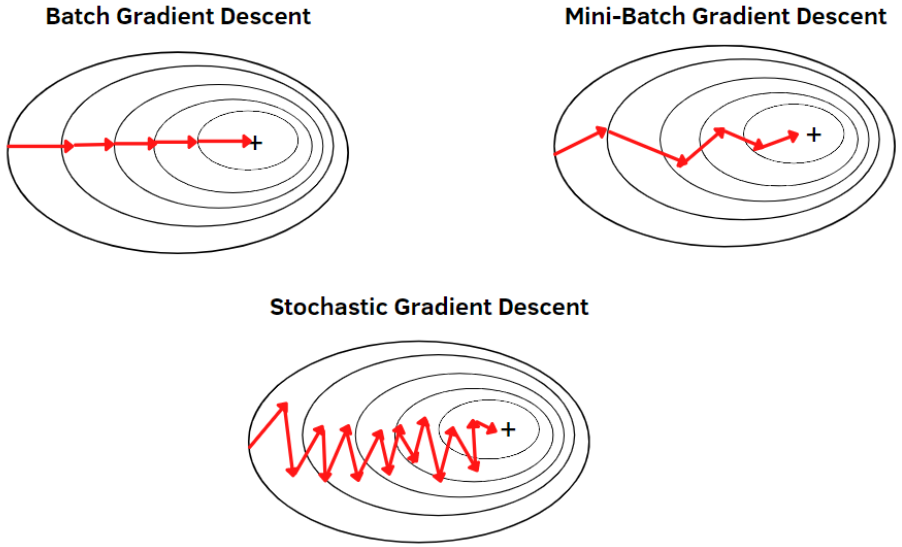
- Gradient descent is very computationally intensive
- Learning rate can really affect our learning abilities (Too slow of training or get into local minimas)
- Over and underfitting

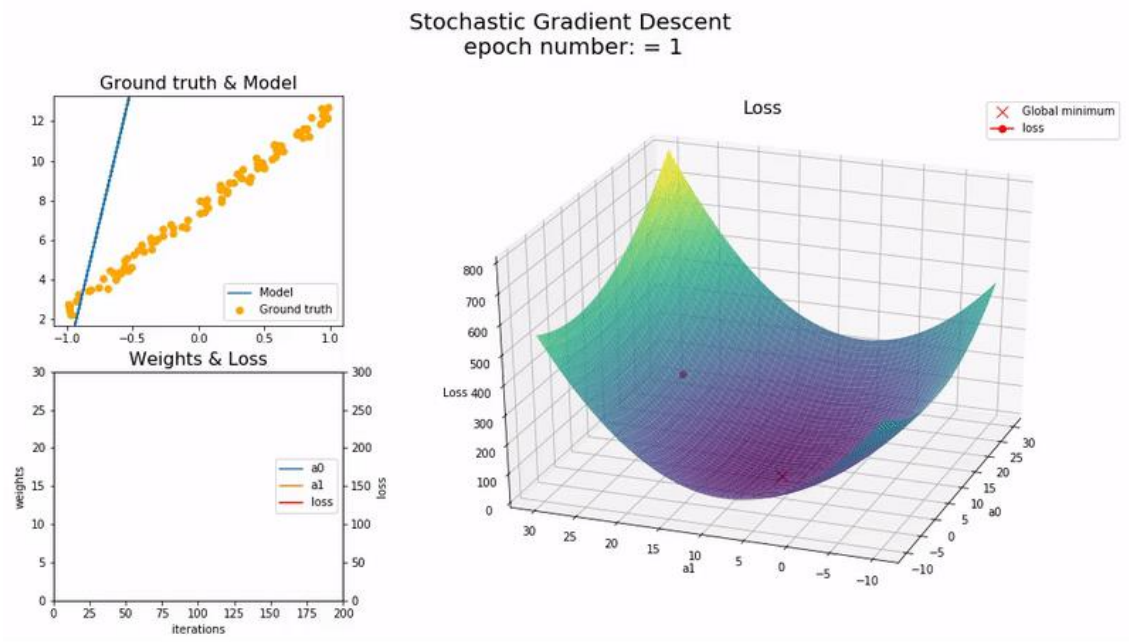
Deep Learning

Fixing Gradient Descent

There are multiple GD algorithms that we can use that will make it way faster:

- Stochastic Gradient Descent
- Adam
- Adagrad
- …

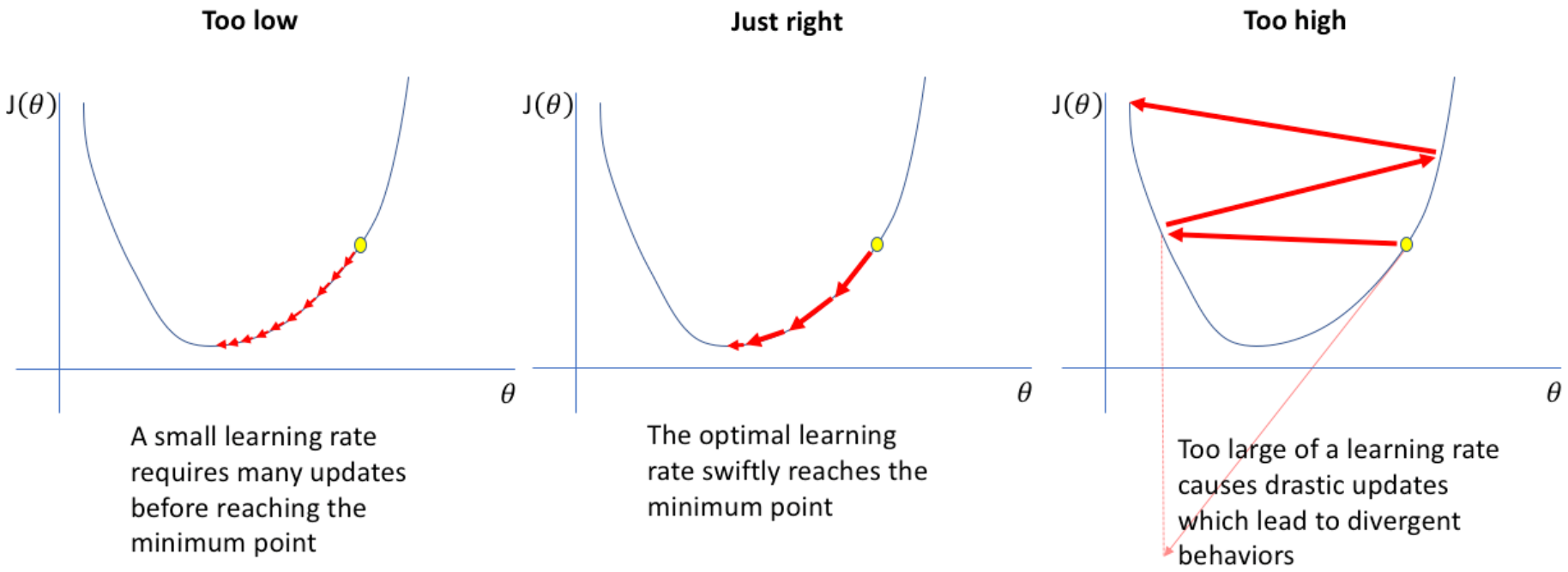




Deep Learning

Fixing Learning Rate

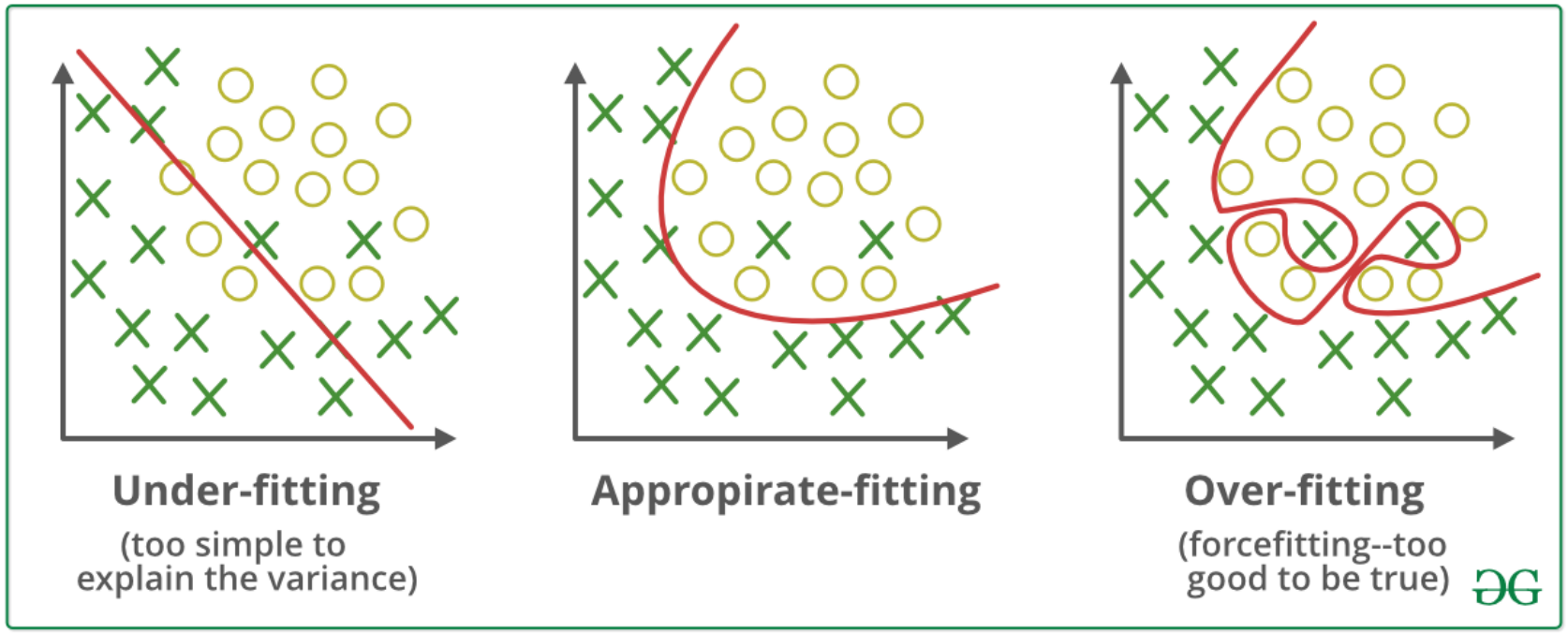
We can use an adaptative learning rate that decreases or increases depending on how large the gradient is and how fast the learning is happening. The example in the center is an Adaptative LR



Deep Learning

Fixing Fitting problems

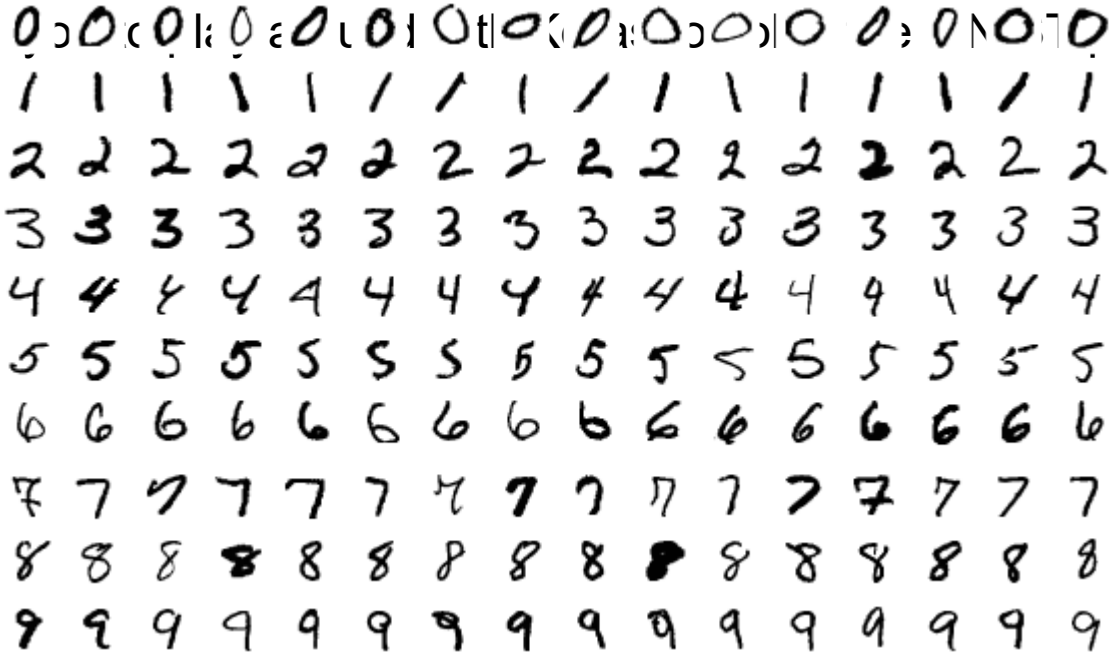
How do we fix this problems? Homework



Deep Learning

Deep Feed-Forward Neural Networks

For the rest of the class I want to solve it using Pytorch



Deep Learning

Fundamentals of Deep Learning

In the next class we will talk about Convolutional Neural Networks