epsi®
l'école d'ingénierie informatique

COMPÉTENCES & DÉVELOPPEMENT
Réseau d'écoles et de centres de formation

DEEP LEARNING

Conference #5 July 2025
Summer School

Rakib SHEIKH [1,2]
rakib.sheikh@cyu.fr

[1] EPSI - Compétences & Développement
[2] CY Cergy Paris Université - CY-Tech

Lecturers of this conference

Rakib SHEIKH

Lecturer at CY Cergy Paris Université

Lecturer at EPSI Paris and EPSI Arras

This material will soon be available at https://cyu.fr/rakib-sheikh

01
Context

Context

Deep Learning is today the most popular paradigm in Data Science.

Popularized since 2006, first by some academic actors, then by big players (GAFAM)

- Initiated a “paradigm shift” in the field of AI
  - *(Tensorflow and Jax by Google, PyTorch by Meta, CudaDNN by Nvidia...)*
  - **It allows to speed up development time of complex processing chains**
  - Making complex DL methods available for a large community

Today DL is developing at a much larger scale including

- Software development platform and environments
- Services in multiple domain : *Biotech, health, finance, client management, etc, ...*

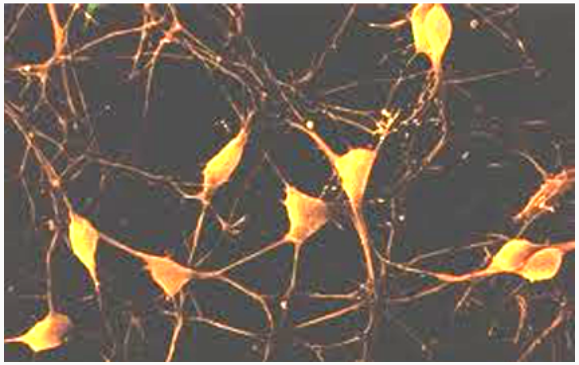
02
Fundamentals of Neural Networks

Brain methaphor

Artificial Neural Networks are an important paradigm in statistical machine learning and Artificial Intelligence

Human brain is used as a source of inspiration and a **metaphor** for developing Artificial NN.

- Human brain is a dense network of 10^{11} of simple computing unit, the **neurons**. Each neuron is connected -in mean- to 10^4 neurons.
- Brain as a computation model:
  - Distributed computations by simple processing unit
  - Information and control are distributed
  - Learning is performed by observing / analyzing hugh quantities of data and also by trials and errors



The foundation calculus of artificial network is : $f(X) = \sum_{i=0}^{n} x_i w_i + b_0$

Where :

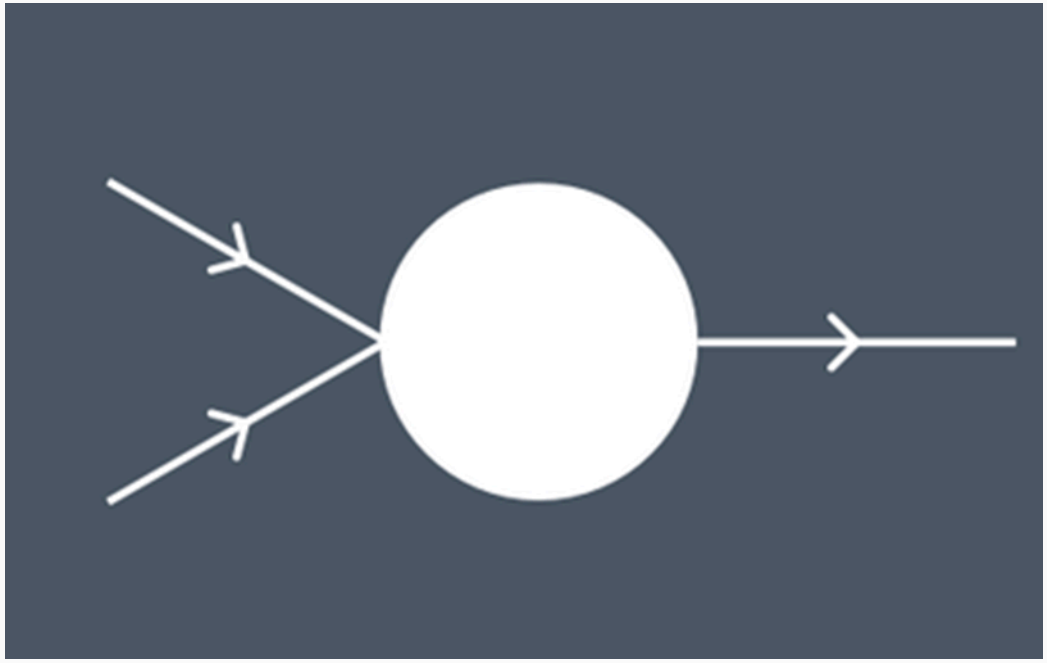
- X are the features inputs (X_i is 1 feature of the set X)
- w_i are the weight of the node
- b_0 is the bias on the node

Single Neural Network visually explained

i | Definition : **Neuron**

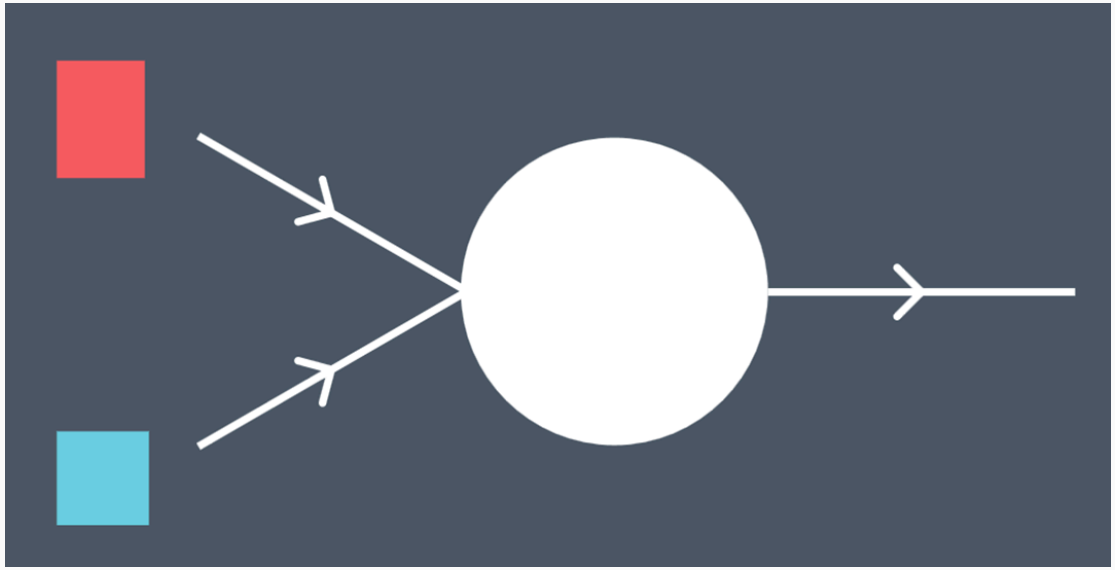
A neuron is the smallest unit of a neural network. A neural network is a set of connected neuron.

A neuron takes a **sets of inputs**, represented by a dataset, **applies some formulas** to produce an **output**



Single Neural Network visually explained

- **Inputs** / **output**: Number, either positive or negatives
- In the following figure, I have **two inputs** and **one output**.



🗒️ | Note

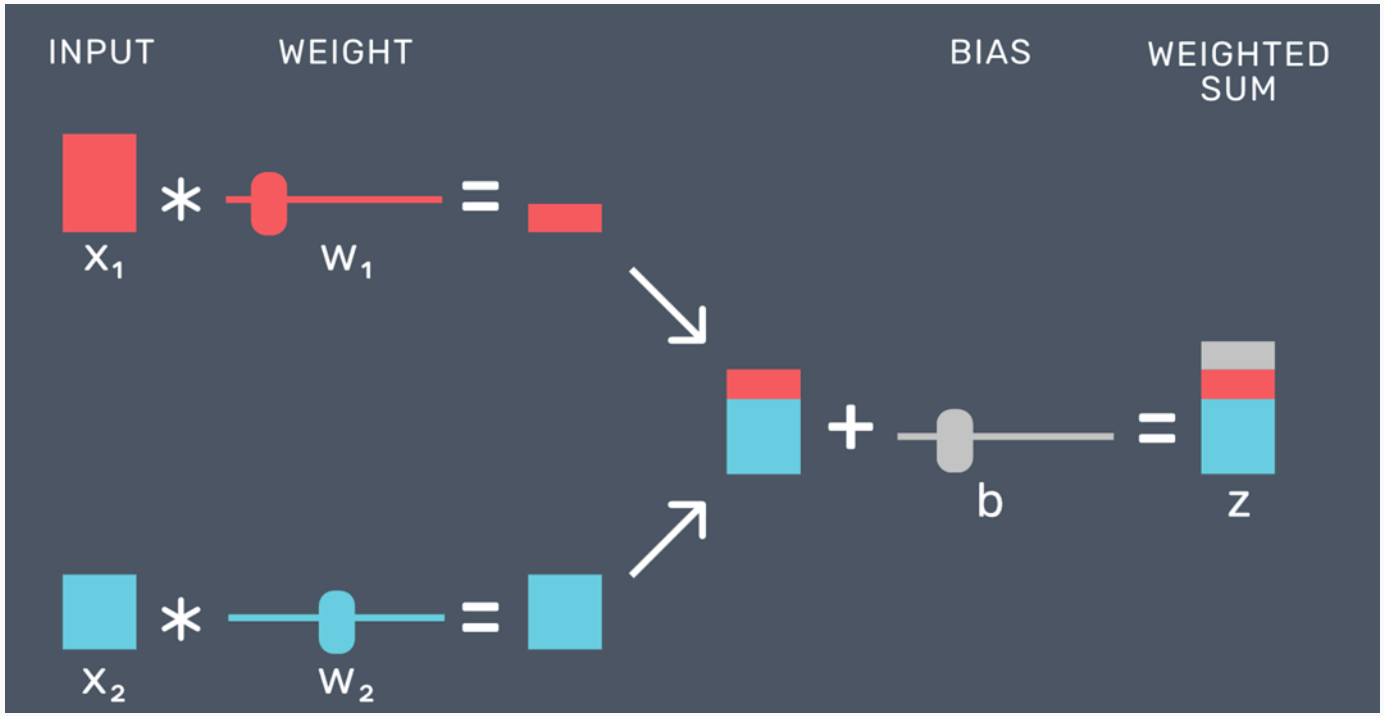
There are no limits about the number of inputs and outputs

Single Neural Network visually explained

Weighted sum and parameters

This is the first calculus that a neuron applies

- It takes each input and multiply it by a **weigth** w_i
- All results are then summed by a bias b_0

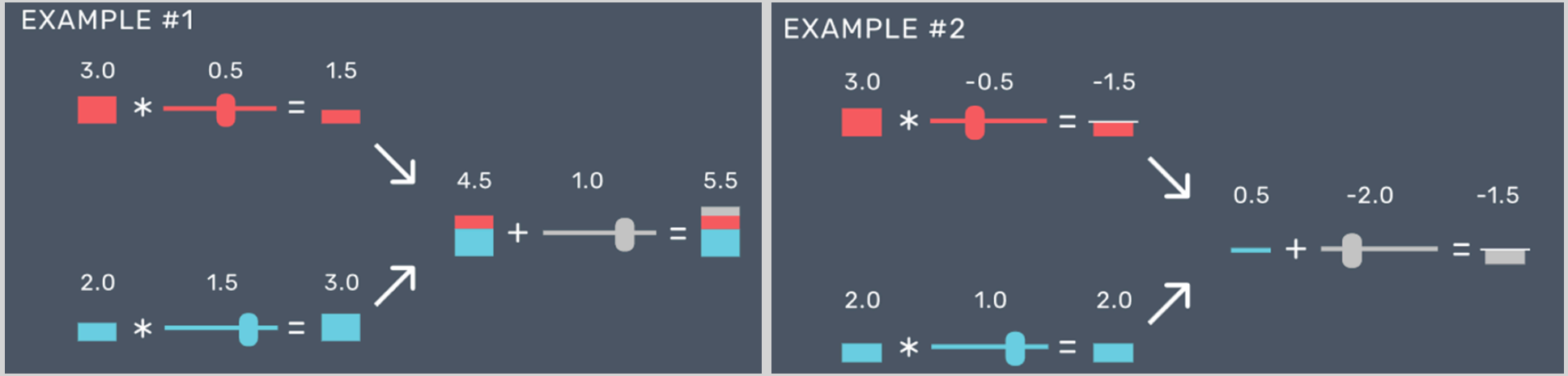


Weight and bias are called **parameters.**

These parameters can be changed, it is a way to learn a neural network

Single Neural Network visually explained

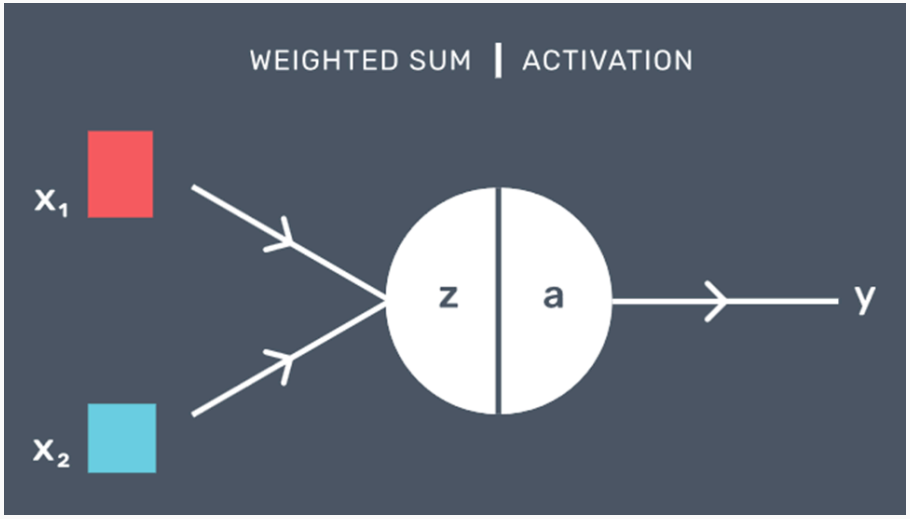
i | Example :



Single Neural Network visually explained

Activation function

- It is the second calculus made by a neuron.
- It introduce some non-linearity in the model (each features are independent)
- It takes the result of the previous calculus to apply into a new activation function.
- The results are called the **output** than can be passed on a another neuron.



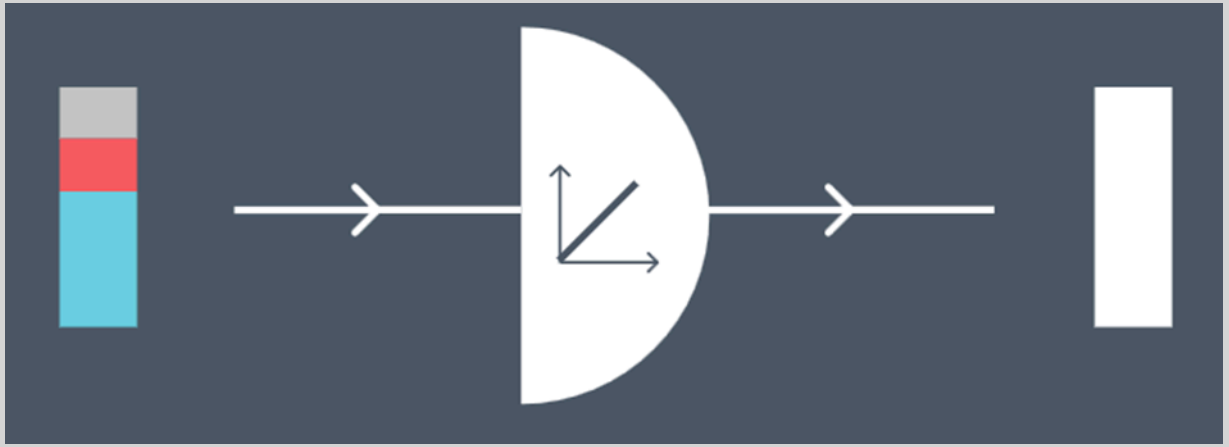
Single Neural Network visually explained

Activation function

i | Example : Example of Activation Function : Linear Activation

A Linear activation is the most basic activation function. It takes the input and throw it as an output (no calculus applied)

$$f(x) = x$$



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron.

 | **Goal** : Predict the price of one night hotel room based from the city center distance.

Let's have a dataset with 1 feature and 1 target to predict

- Distance from city center will be our feature (m)
- Price ($) will be our target, so the value we want to predict.

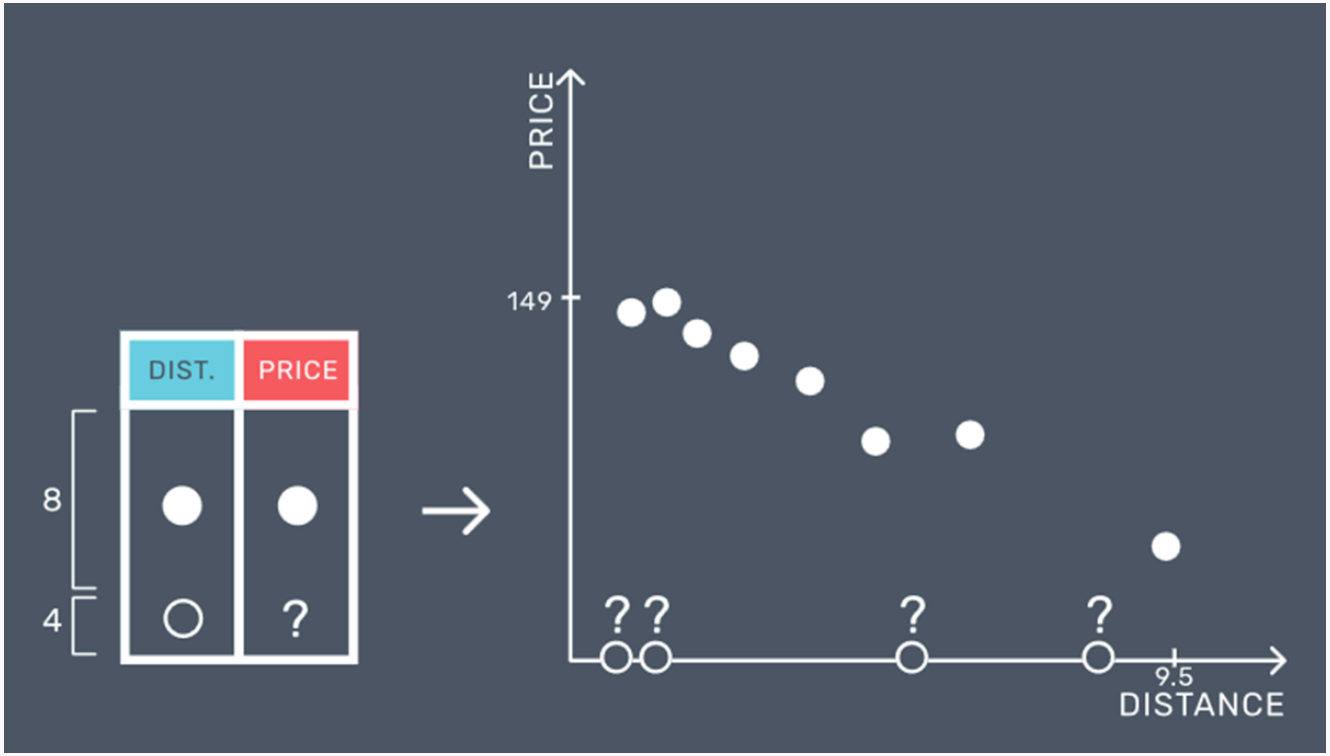
We will use a Linear Regressor with a neuron.

DISTANCE (MI)	PRICE ($)
0.5	146.00
1.1	149.00
1.6	140.00
2.4	134.00
3.5	127.00
4.6	110.00
6.2	112.00
9.5	81.00
0.3	156.00
0.7	168.00
4.9	116.00
8.5	99.00

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron.

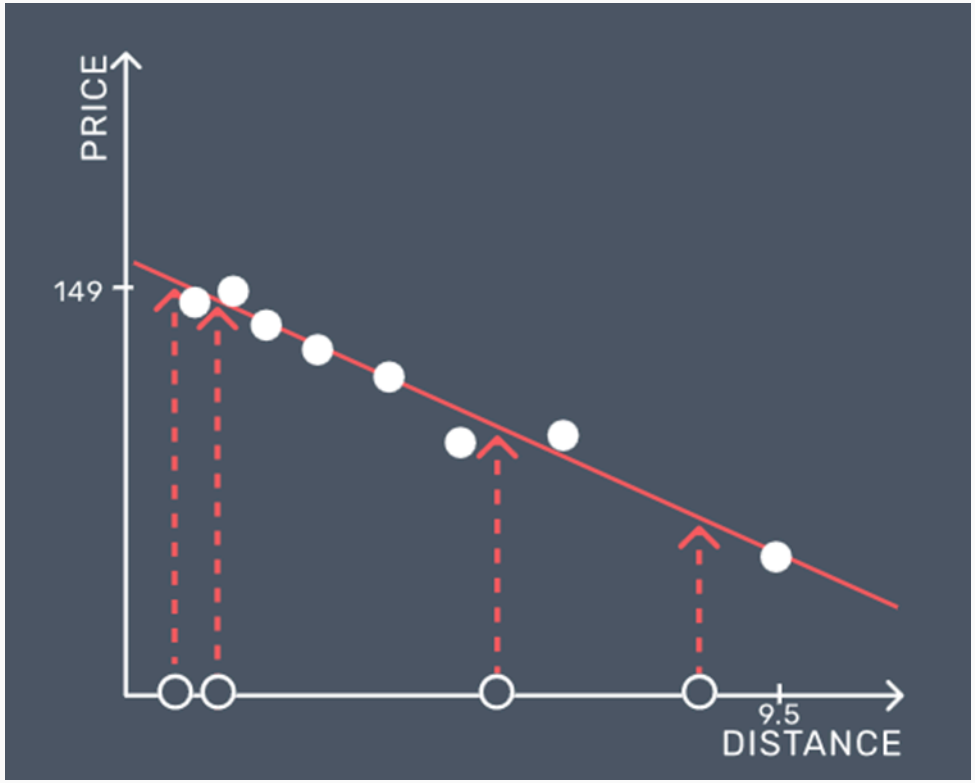
The task is a regression ! Because we are predicting a number in $\mathbb{R}$.



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron.

In the following example, is it obvious that we can draw a line that tries to pass all points in the dataset. **This is what we should teach to our neuron!**

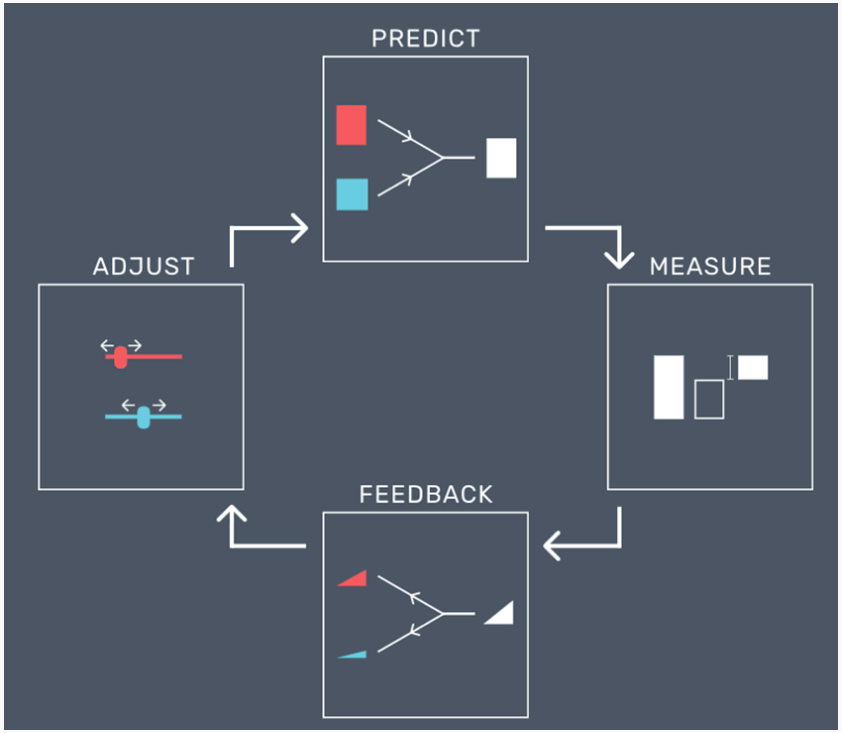


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron.

Let's take only 1 neuron for our neural network model to predict price of our example

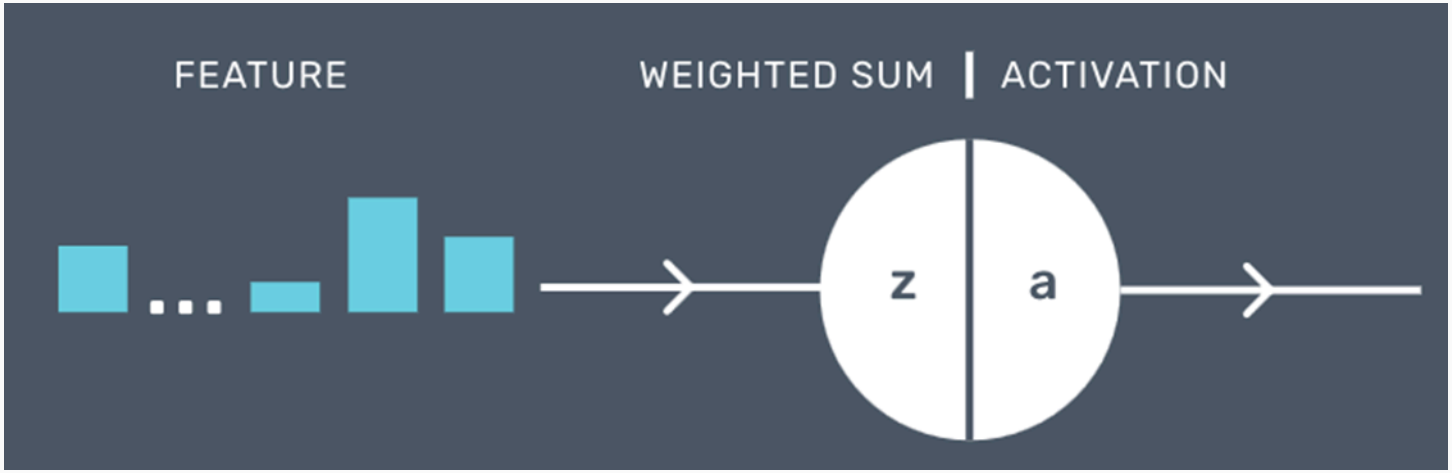
- *Before moving on, we should formalize how we will make our neuron to learn that line. The following figure will visually explains in 4 steps as a loop.*



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Adjusting Parameters*

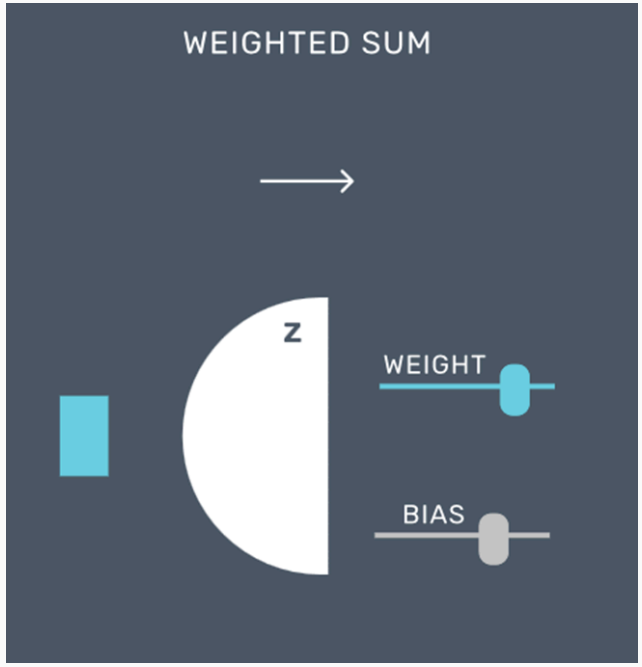
We are going to apply 2 calculus in 1 neuron (**weigth w_1 and activation function $a(z)$**)



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Adjusting Parameters*

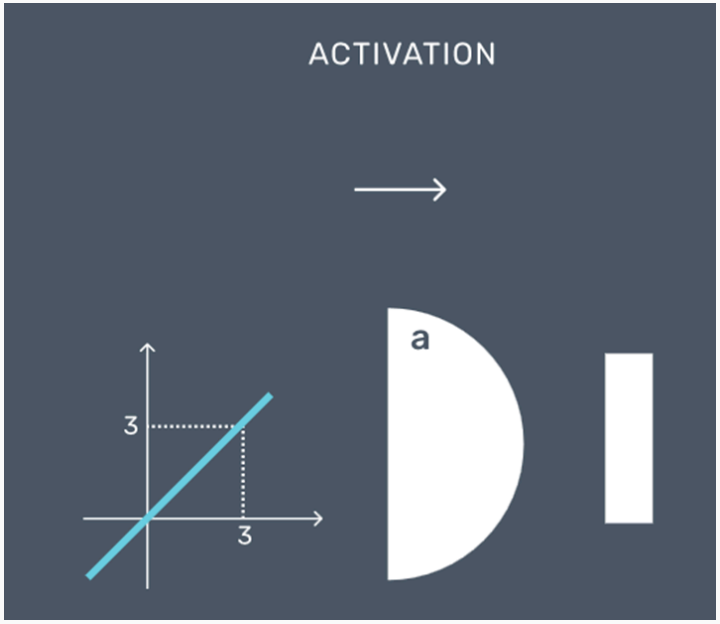
- Since we have 1 feature, we will have 1 input
- We are going to initialize the w_1 and b_0 with randomized values for the first iteration.
- The calculated value will be stored as z value.



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Adjusting Parameters*

- We will now apply the activation function
- For this example, we will stay at Linear Activation, so we will have $a(z) = z$

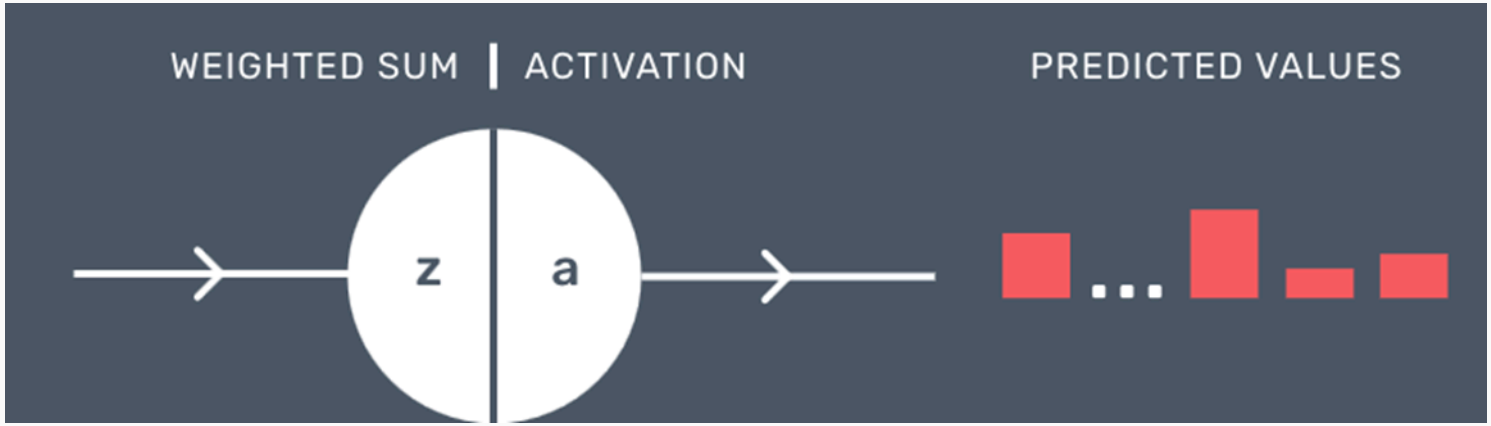


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Launching Predictions*

- We just finished adjusting parameters, we can now launch a prediction !
- **We still have not learned at this point**

While predicting a value, we can miss our target from afar !



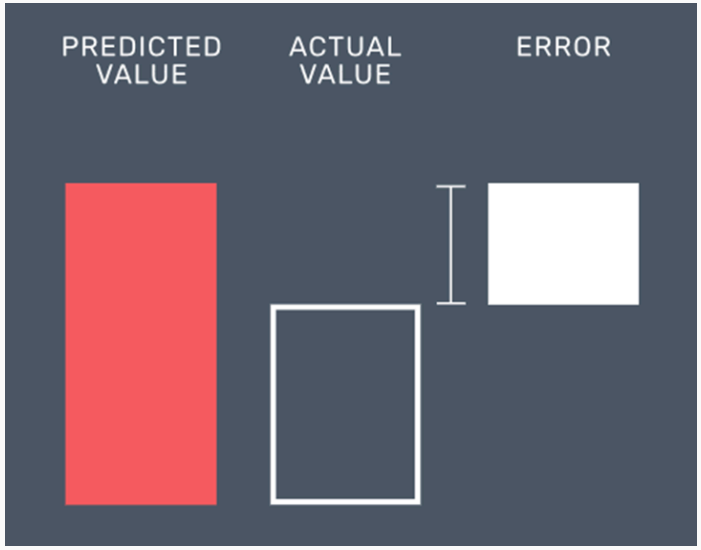
⚠ | Warning

This process is call a **Forward Pass**

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Measuring*

- Since we know the target value, we can quantify the model performance by taking the difference between the predicted value and the actual target.
- This is called the **error value**

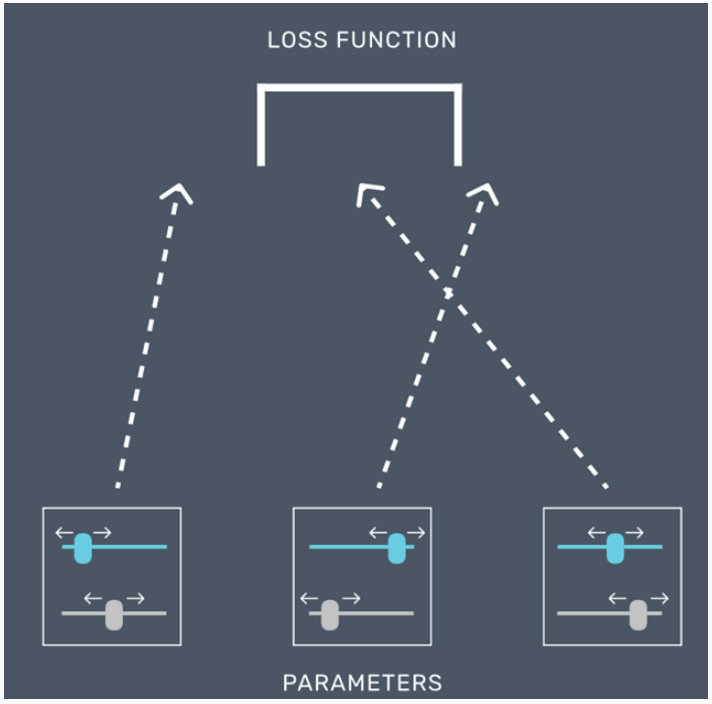


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Loss function*

The loss function evaluate how far we missed our target from the prediction, based on the current parameters.

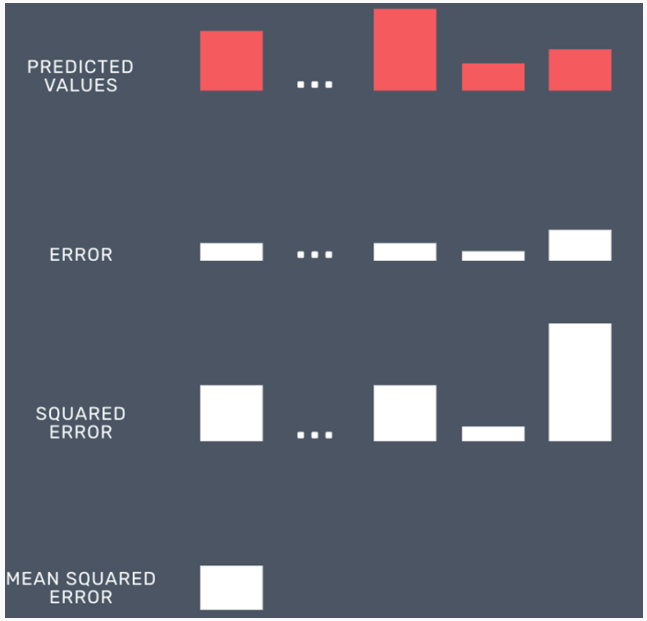
- *Imagine you are shooting a penalty kick in football. You have to get the right power, the right aim to score de goal.*



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Loss function : MSE***

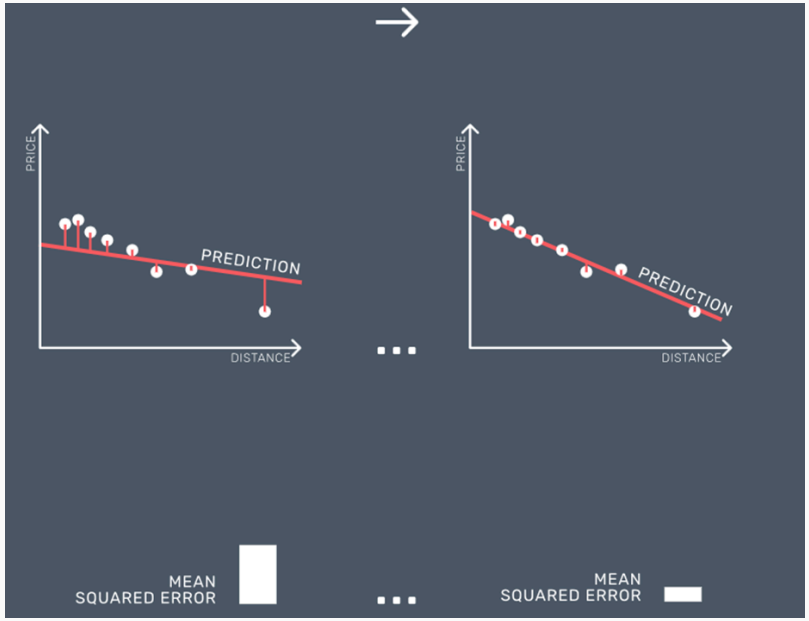
- We will be using Mean Squared Error
- Each value will be raised to square, so we can have a squared error. Then we apply the mean, hence the name mean squared error.



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Loss function : Defining a goal*

- Our goal through the Loss function is to minimize it during our training loop.
- This is one of the key concept for optimizing hyper-parameters.



Single Neural Network visually explained

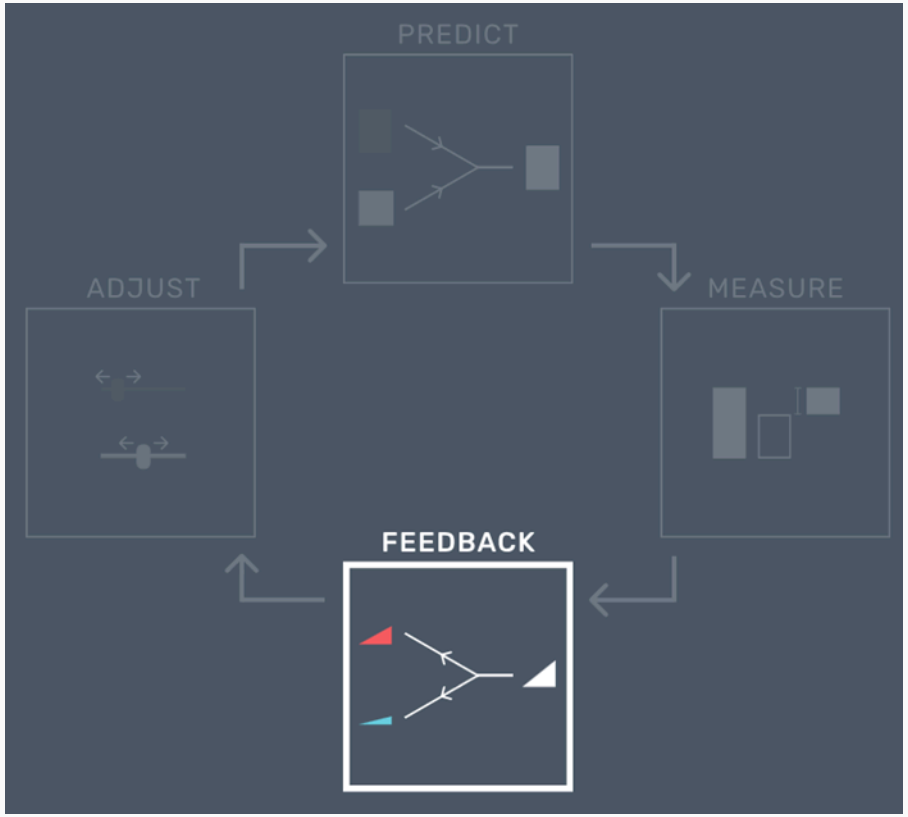
Applied example : Linear Regressor with a neuron. ***Recalling w_1 and b_0***

- Recall a neural network learn by adjusting it's parameters w_1 and b_0
- We are going to see how w_1 can affect the value of the loss function

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback*

At this step, our neuron have not learnt yet!. It is at the feedback step that our neuron will learn.



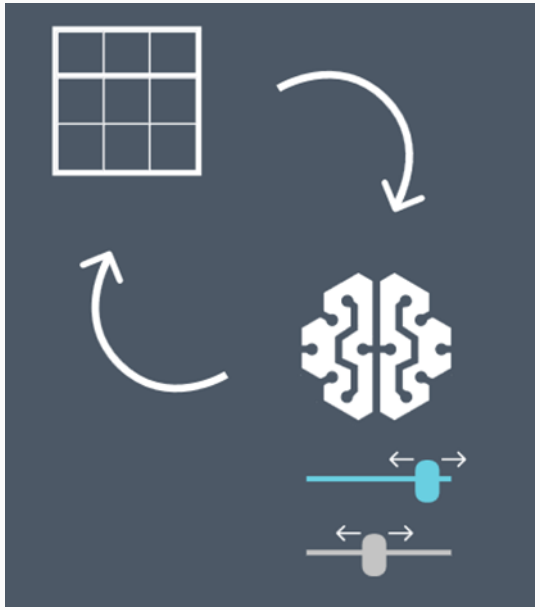
Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback : Learning*

Recall the established goal of the loss function was to minimize the error value.

- *This can be possible by adjusting w_1 and b_0 parameters.*

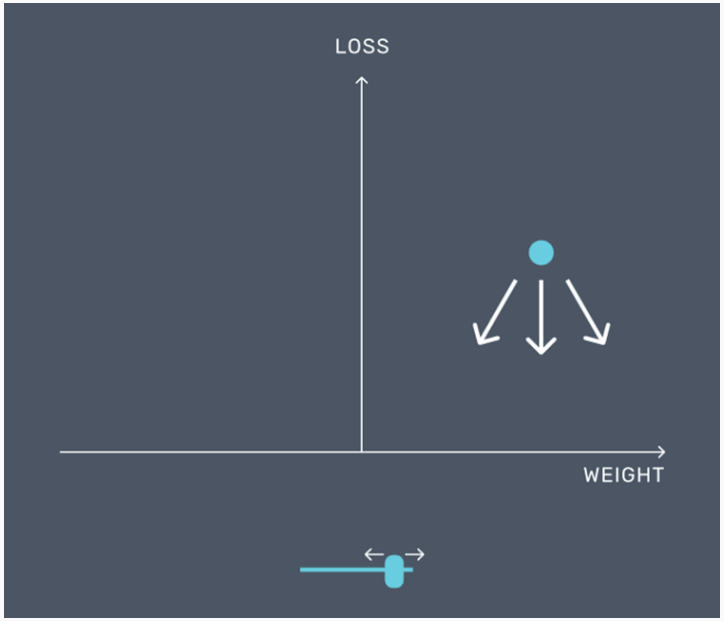
How far do we have to adjust those parameters?



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback : Minimizing loss*

Let's start 1 training loop step and plot the error value.



- *We have to minimize this value close to 0, by adjusting either to the right or to the left, the* w_1 value

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback : Loss curve*

The Mean Squared Error gives us the following mathematical property:

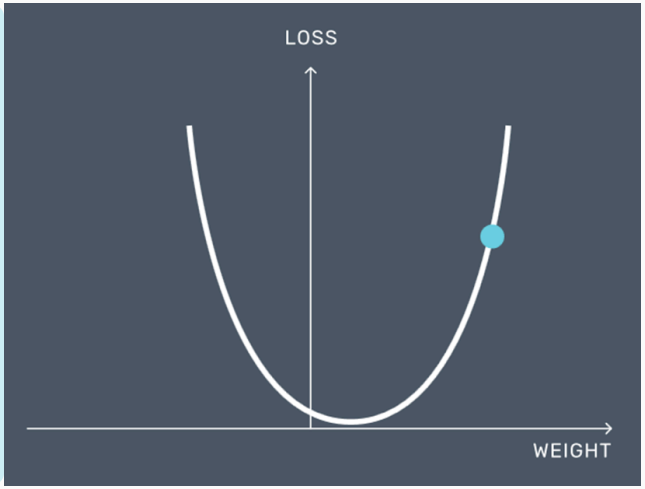
- *By plotting all the possible value of the loss function from w_i we have a convex function which we will recall it definition.*

i | Definition : Convex Function

Let f an application of I in $\mathbb{R}$, f is a **convex** function in I if

$$\forall (x, y) \in I^2, \forall t \in [0, 1], f(tx + (1 - t)) \leq tf(x) + (1 - t)f(y)$$

By geometric interpretation, A function f is convex in I if and only if all graph arc of f is below of the chord.

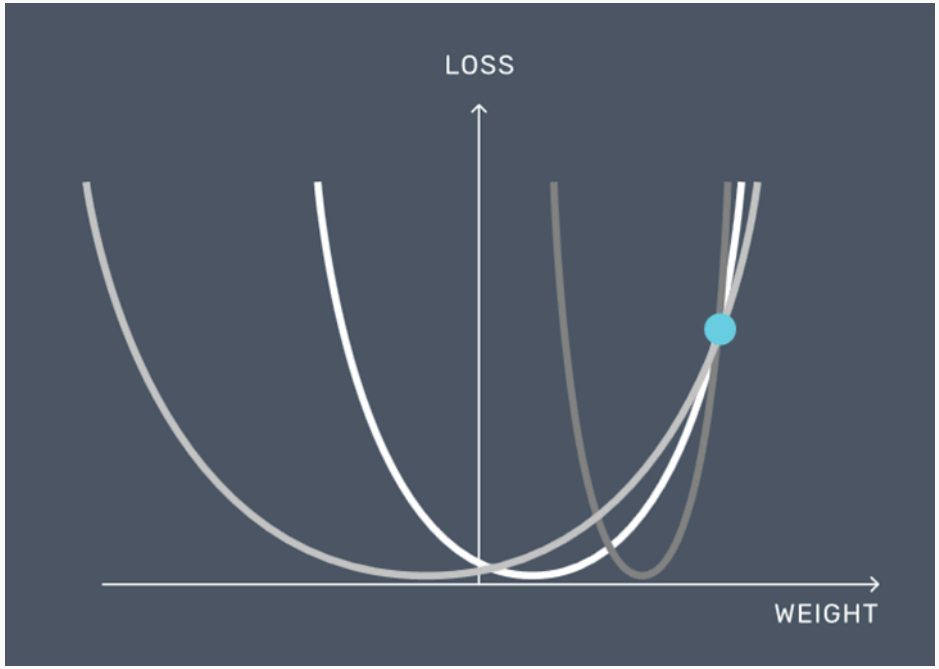


Therefore, the following plot look like x^2 function. Since we already know that x^2 is convex, we are sure that our loss function contains 1 value that is the absolute minimum. Hence, the optimal value for w_1.

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback : Loss curve*

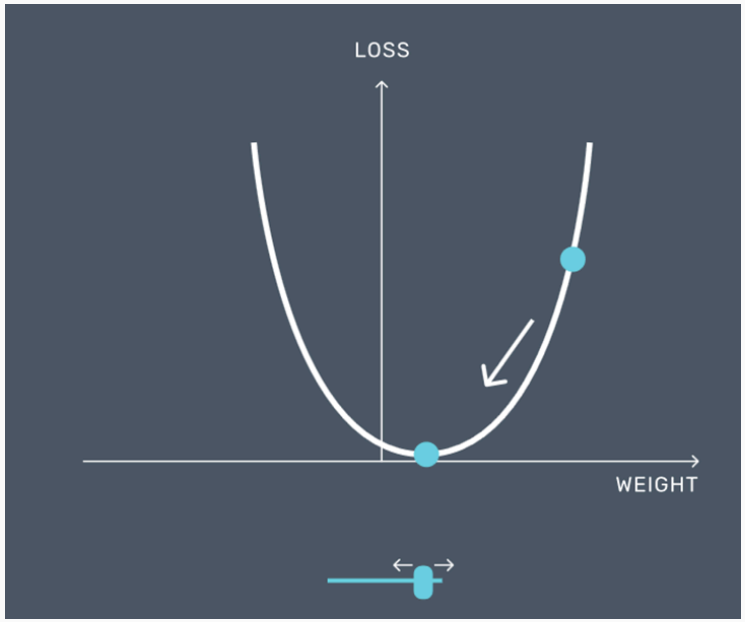
- Its width and position may vary, but its shape will always remain the same.
- Therefore, we can use an alternative definition : $\forall x \in I, \exists a \in R, f(a) < f(x)$
- *The f function have a minimum at a if for all values of x in the interval I represented by w_i we have $f(x) > f(a)$*



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Feedback : New goal : find*** a

- Remember : Artificial Intelligence is mathematics (probability and statistics). We cannot find the true a, but we can get as close as possible to a.

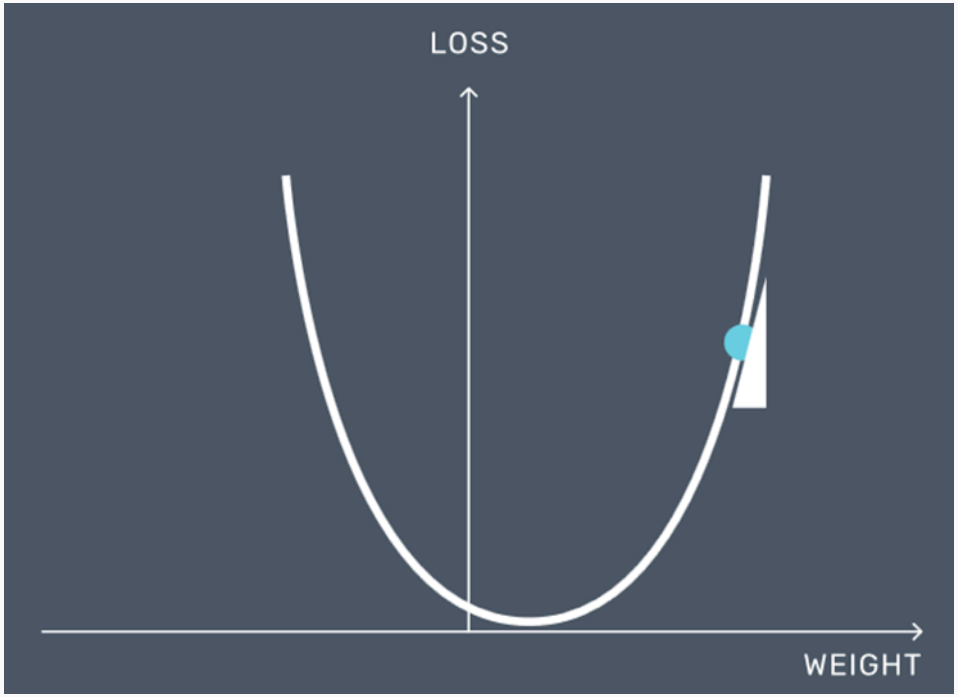


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Feedback : Derivative or Gradient Descent***

How can we bring tell our neuron to adjust its parameters so we can minimize the loss ?

- Good news, we can derivate our function, known as gradient descent.
- The gradient value will tell us which direction we should adjust our w_1 value.

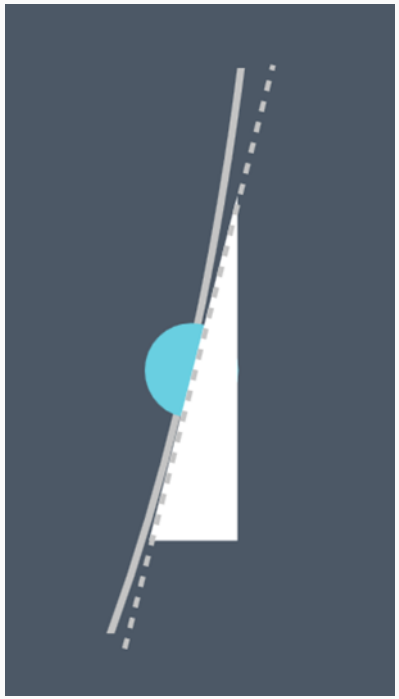


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Feedback : Derivative or Gradient Descent*

Gradient descent inform us the steepness of the slope.

- *The steeper the slope, the larger the gradient*
- *A higher gradient indicates that we still far from the optimal minimized value.*

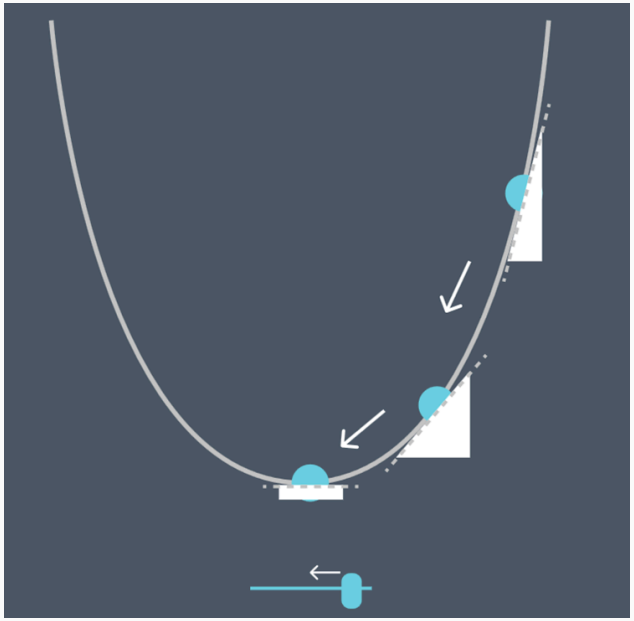


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Feedback : Derivative or Gradient Descent***

If we reduce the w value, the gradient will also be reduced until we reach 0.

- The first property of a gradient is his **magnitude**.
- The gradient magnitude informs the neuron how far we are from the optimal value

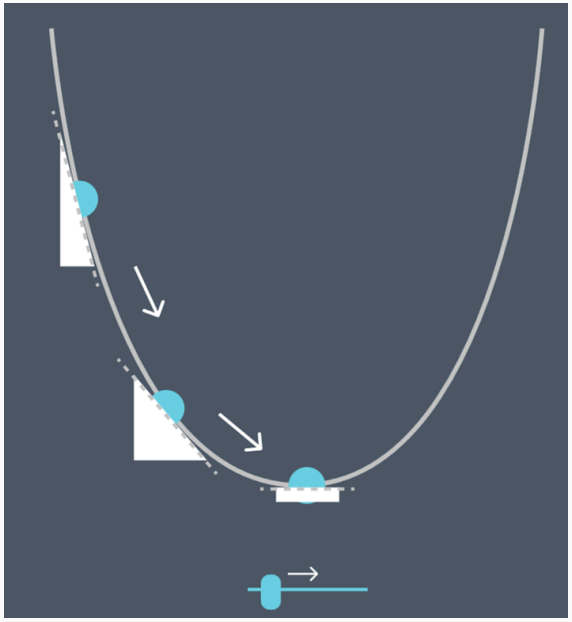


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Feedback : Derivative or Gradient Descent***

If we reduce the w value, the gradient will also be reduced until we reach 0.

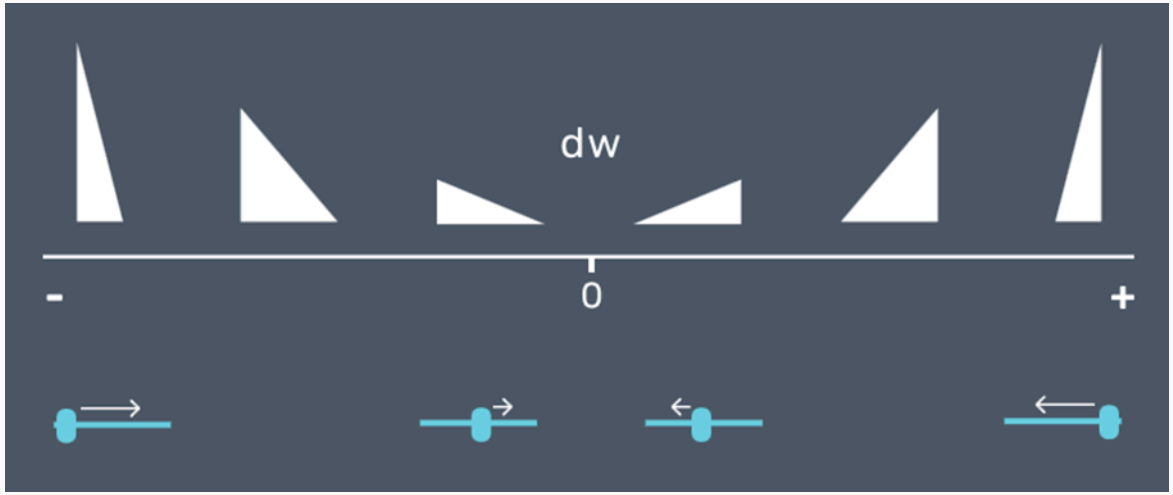
- The **second property** of a gradient is his **direction**.
- The gradient sign informs the neuron to which direction we have to adjust the value of the w_i
- *If the sign of the gradient is negative, then we have to raise the w_1 value. The opposite is true.*



Single Neural Network visually explained

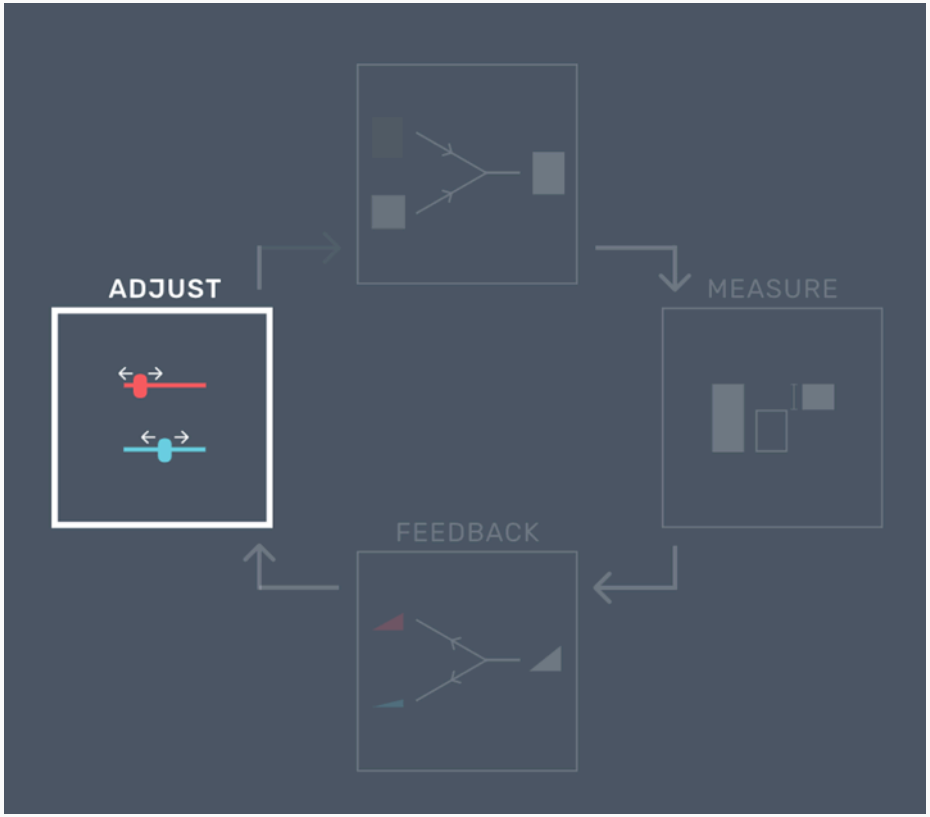
Applied example : Linear Regressor with a neuron. ***Feedback : Derivative or Gradient Descent***

- The magnitude and the direction of the gradient are the two information that we give to the neuron.
- Since the neuron will have many training steps, we will have plenty of chances to find the minimum.
- For that reason, we call it Gradient Descent.



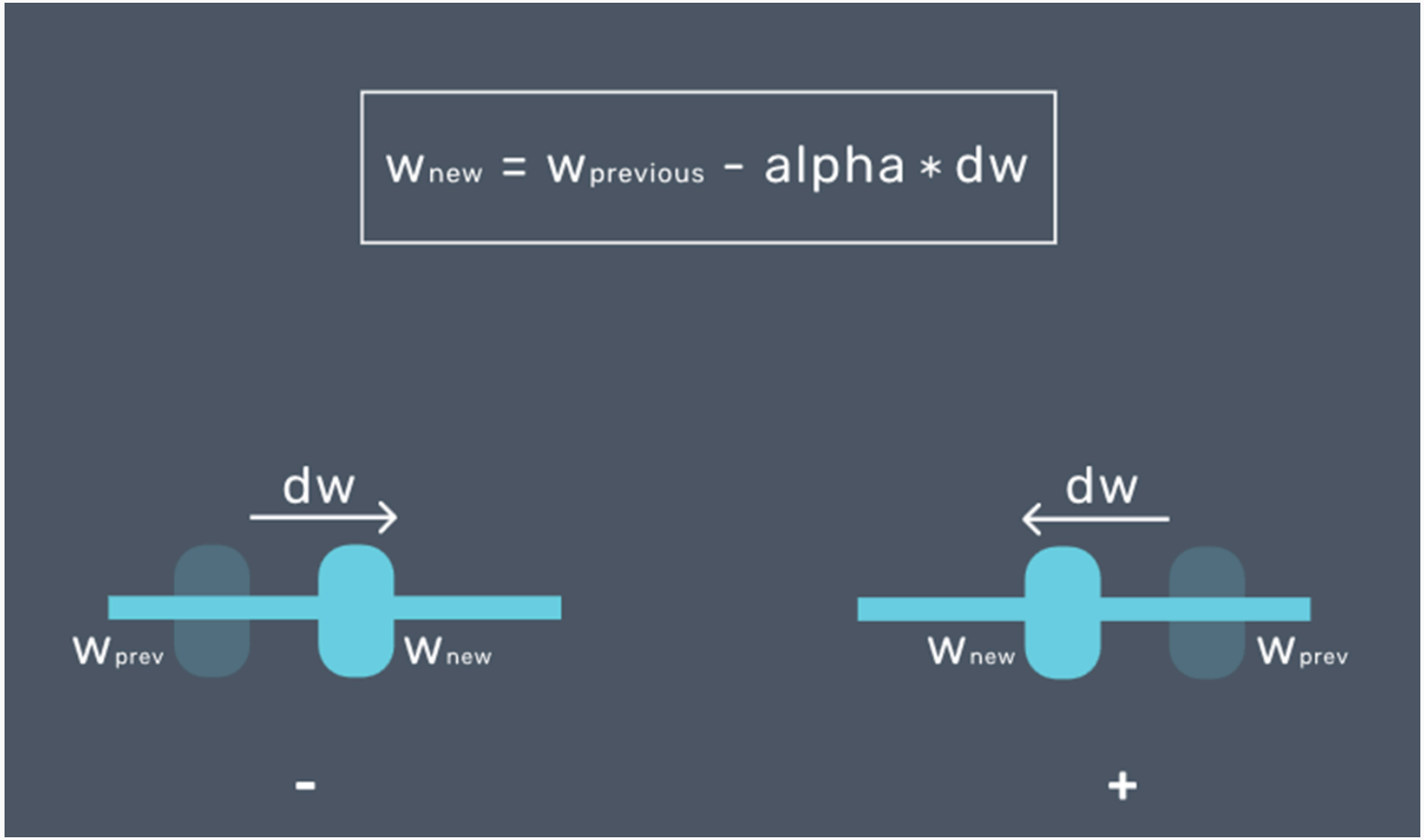
Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Last Step : Adjusting value based on the feedback***



Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Last Step : Learning rate***

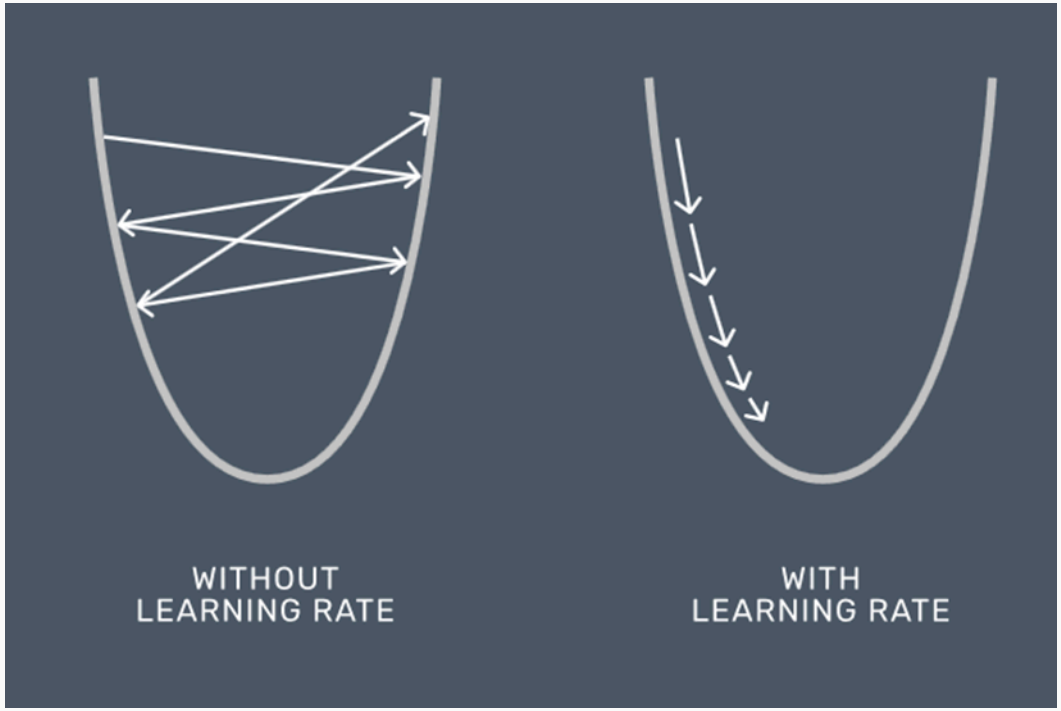


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Last Step : Learning rate***

We want the Descent Gradient to be the smoothest possible. So we are going to introduce a learning rate, which will lower the value of the calculated gradient.

The closer we are to 0, the lower will the learning rate.

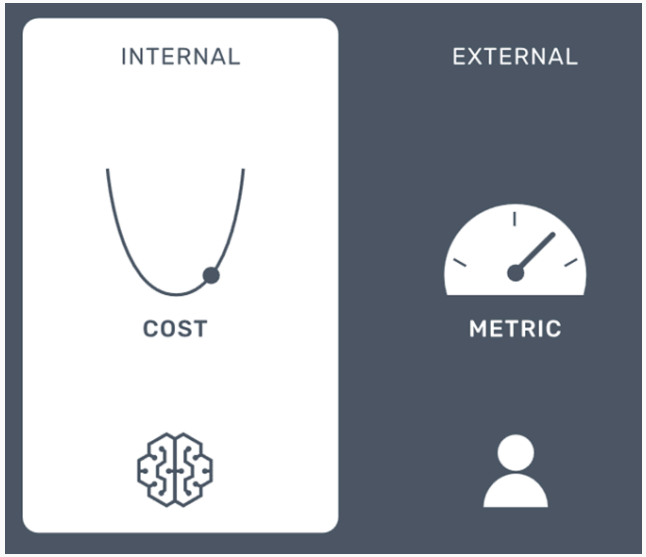


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. *Last Step : Cost & Metrics*

- At measure step, we have two measures that is calculated
  1. ***Cost*** *: It is the mean of the loss values while training the neural network. It is a more precise term to define the loss value at training.*
     - **It allows us to monitor internal performance of the network. The lower, the better**
  2. ***Metric*** *: Equivalent of MSE in our example.*
     - *Used to evaluate the external performance of the network*

     Depend of the business context

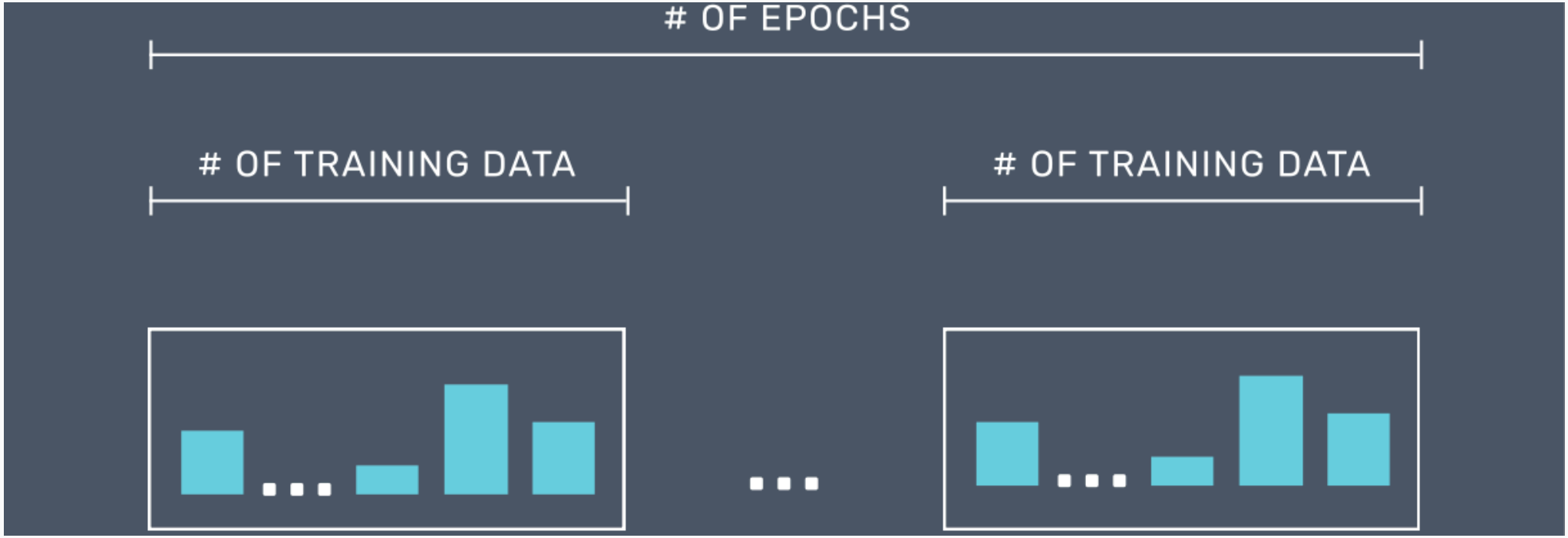


⚠ | Warning

- The whole process of updating both w and b parameters is called a **Backward Pass**
- We have completed one iteration of 4 steps of training, named an **epoch**

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Last Step : Cost & Metrics***

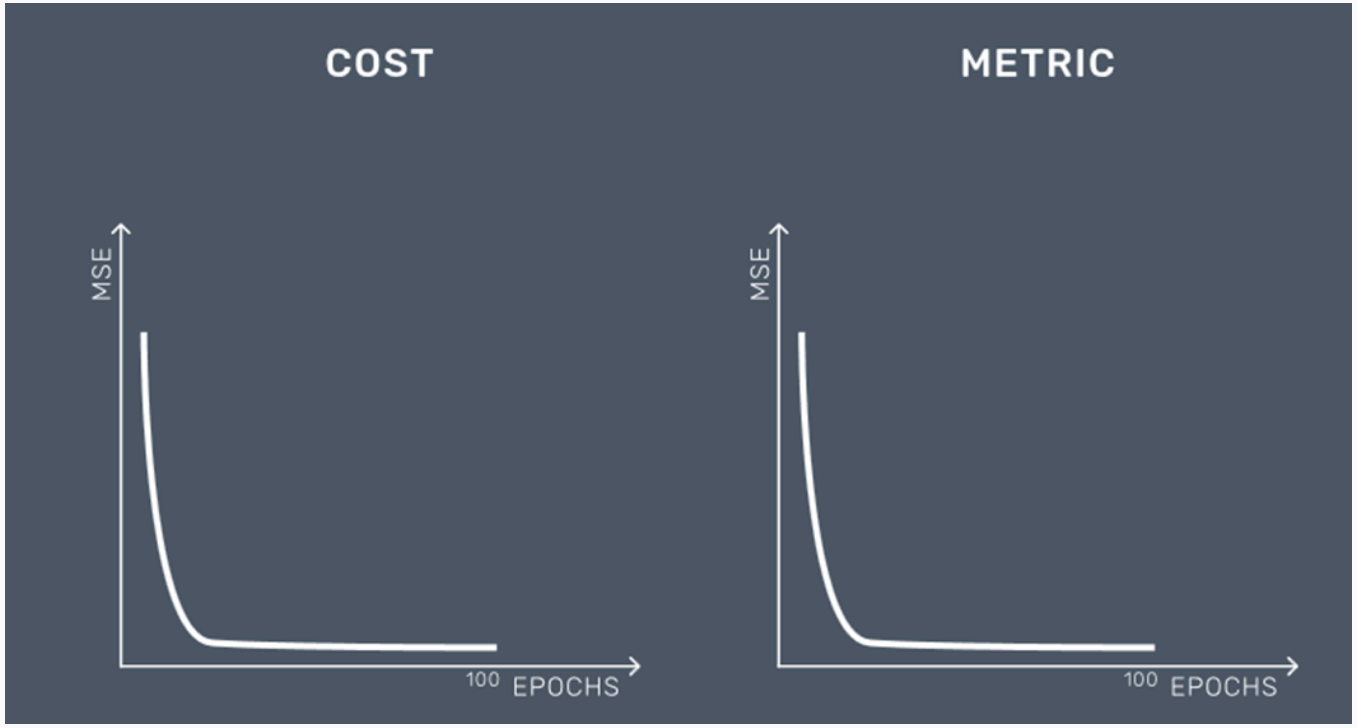


We'll repeat the four steps for 100 epoch. And once we've gone through all the epochs, training will be complete!

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. ***Last Step : Cost & Metrics***

After multiple iteration (epochs), we notice that the MSE is improving and converge closer to 0.



⚠ | Warning

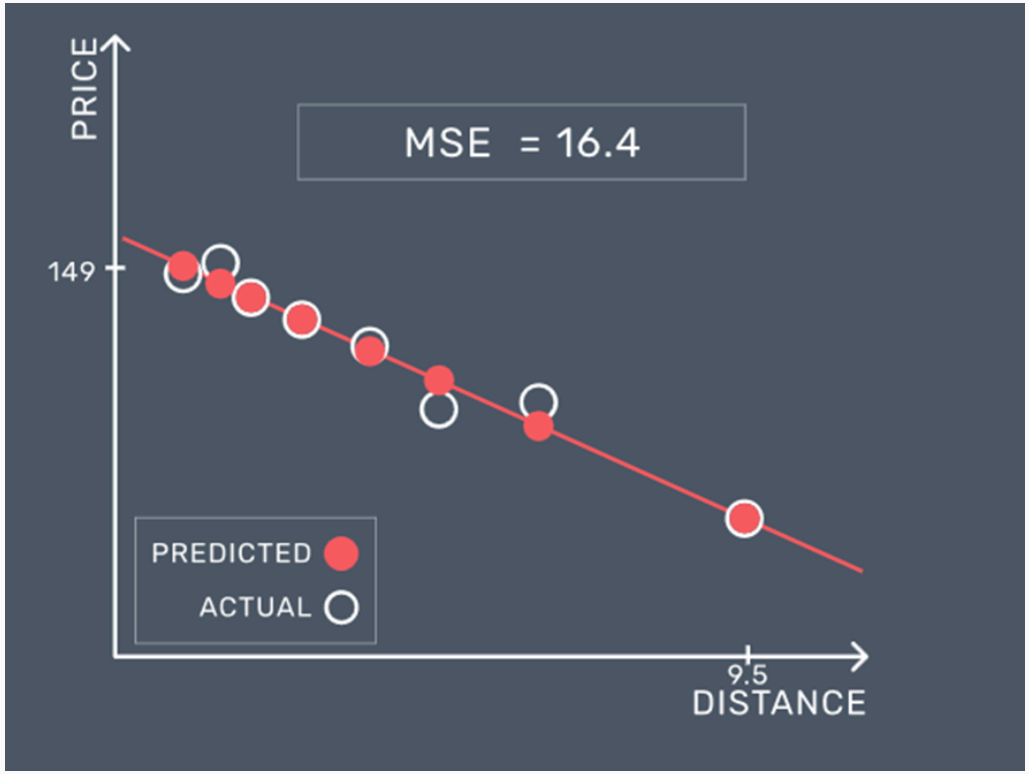
The whole process (iteration of Forward Pass + Backward Pass) is called **Back propagation**

Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. Performance during training

At the end of 100 epochs, we trained a neural network with a MSE of 16.4.

- **It's means that for each prediction we made, the corrected value is within $+-16.4$**

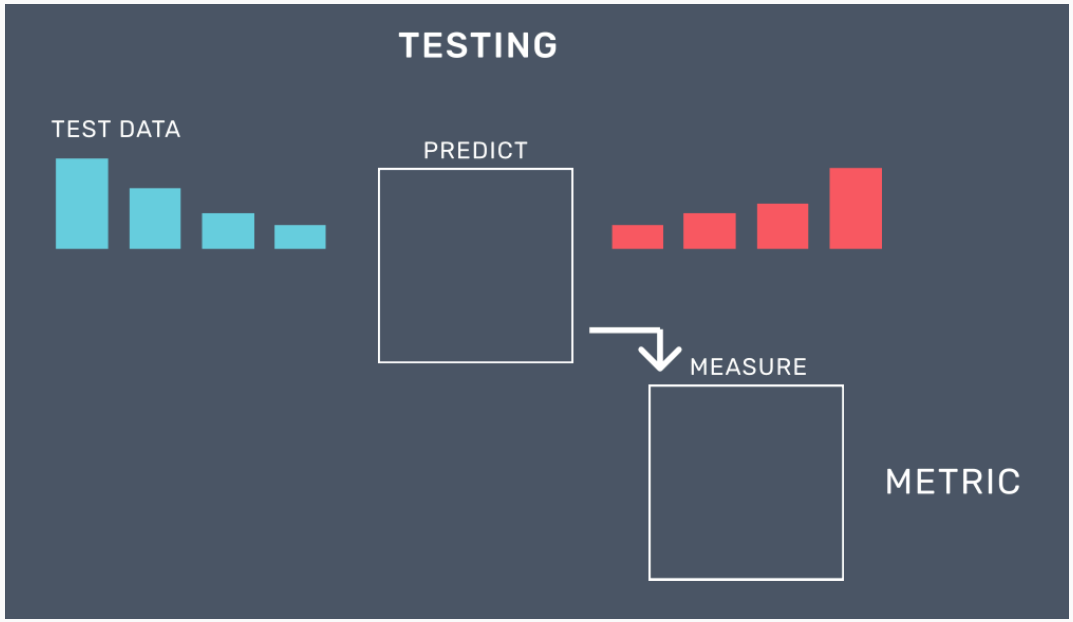


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. Performance during training

We need to test the model, with data that we never seen before! ***We don't need to go through all four steps.***

- *In prediction we pass through the features (distance) through the neural network and get the prediction (price) at the other hand*
- *In measure, we compute the metric (the MSE) of the prediction. The cost is internal to the model and it's used only during training, so we don't need it*

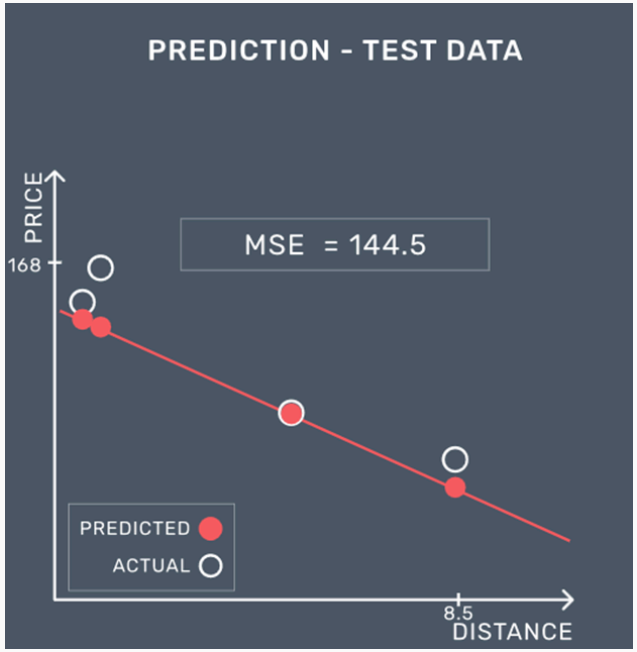


Single Neural Network visually explained

Applied example : Linear Regressor with a neuron. Performance during test

Now, if we are to evaluate a dataset never shown to our network we might have some surprise. We have a MSE of 144.5. Which is worse than the training phase.

- Be wary of that value, by plotting predicted value, we can see that our predicted values matches the line trending. Therefore, our model is successfully trained and ready to be deployed.

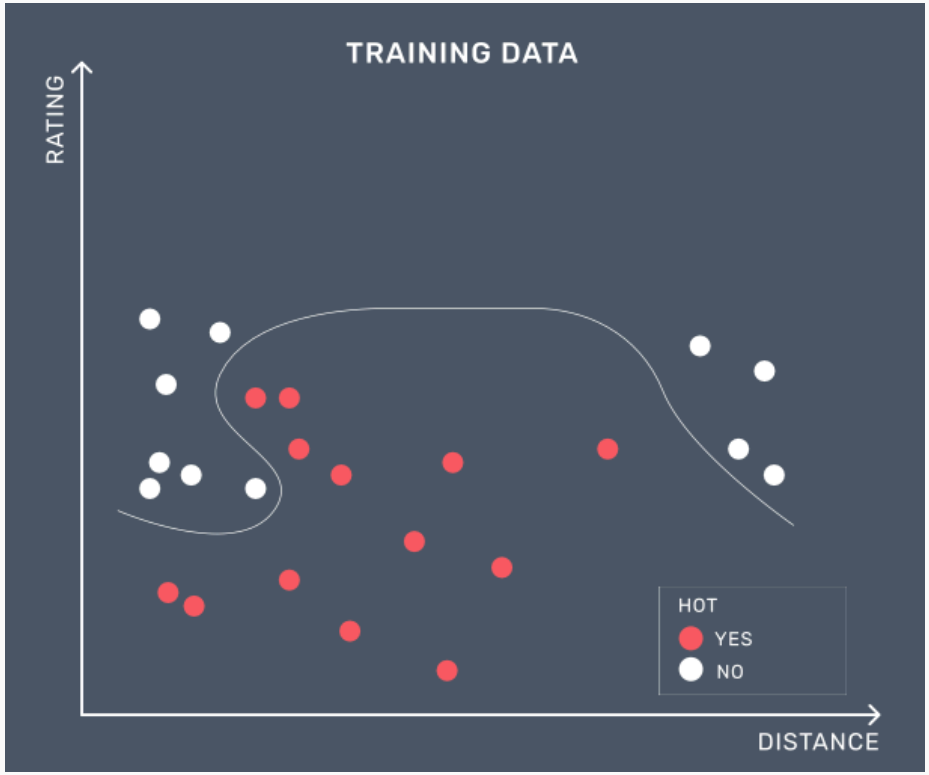


Single Neural Network visually explained

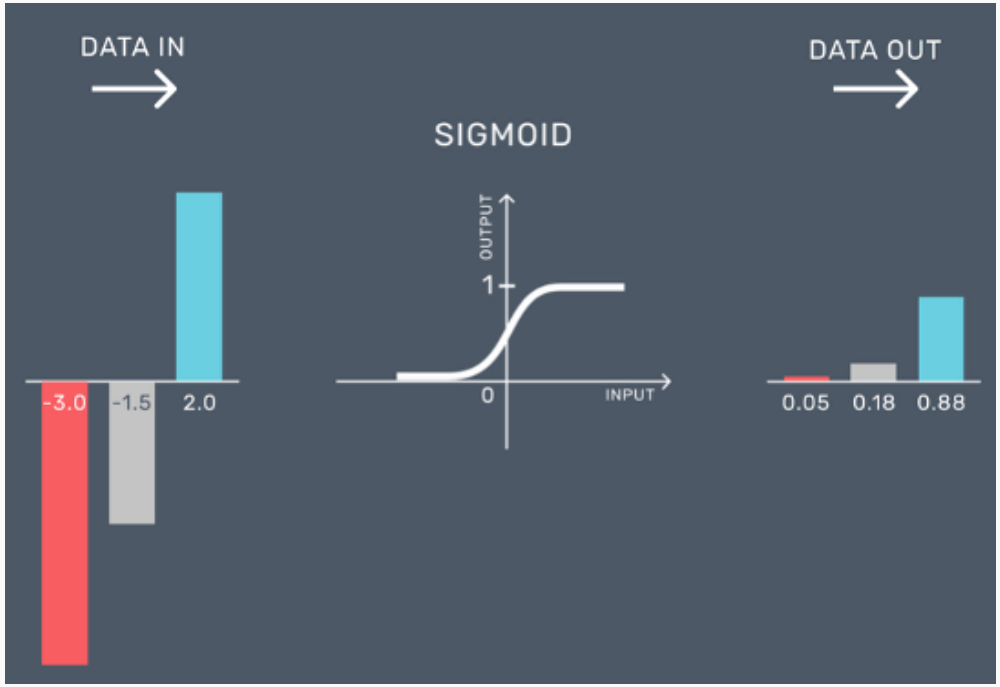
What happen when we do not have a convex loss function ?

Let's suppose we have the following dataset, and the task is classification

- *Our goal is to classify the following data into twos classes (Binary)*



Single Neural Network visually explained



When we are going to predict, we will have a value in $\mathbb{R}$. But since we are in a classification task, we should have either 0 or 1.

So we are going use Sigmoid, that will bind any output value between 0 and 1.

Our activation function will be changed from linear to sigmoid

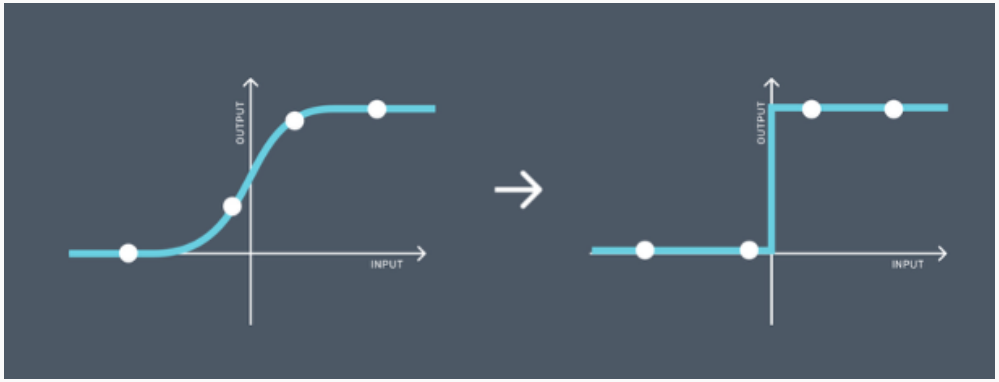
Single Neural Network visually explained

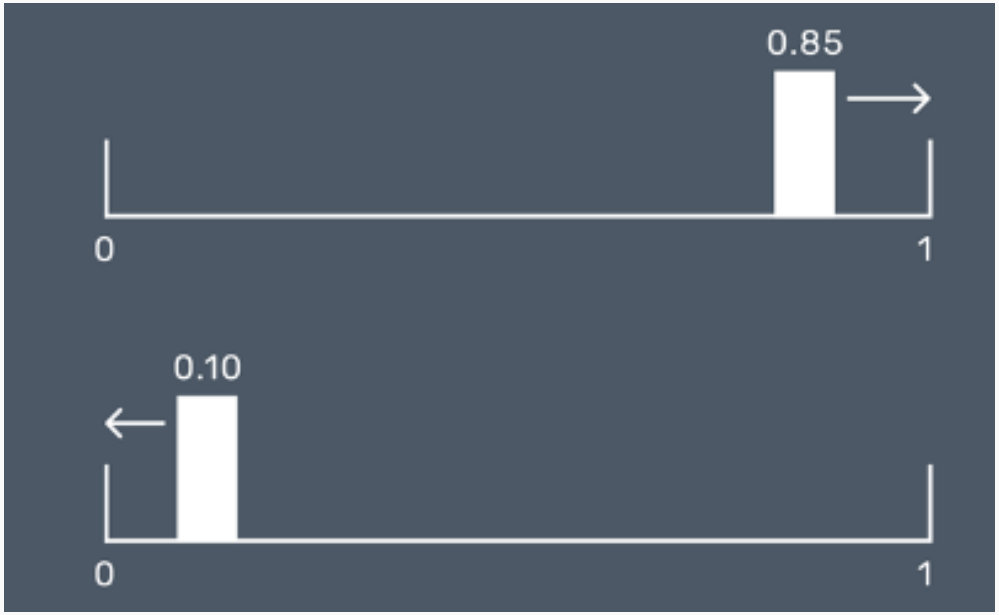
Remember, deep learning is mathematics based, having a value between 0 and 1 can be compared into a probabilistic problem, such as :

- The probability of the final class 1 is 80%
- The probability of the final class 0 is 20 %.

We can fix a threshold that will round the percentage either at 0 or 1 (by default 0,5)

How can we modelise this pattern ?

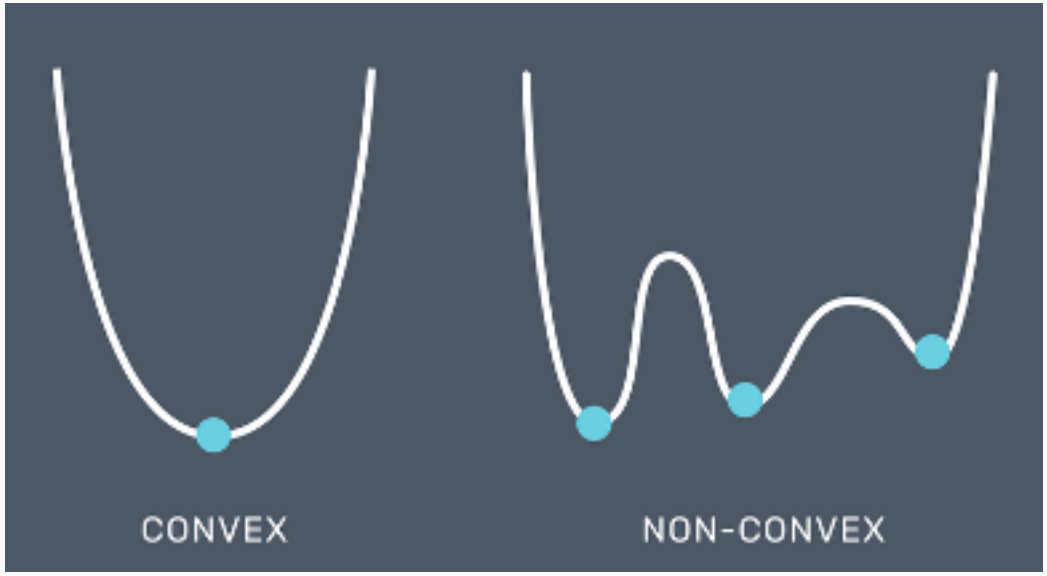




Single Neural Network visually explained

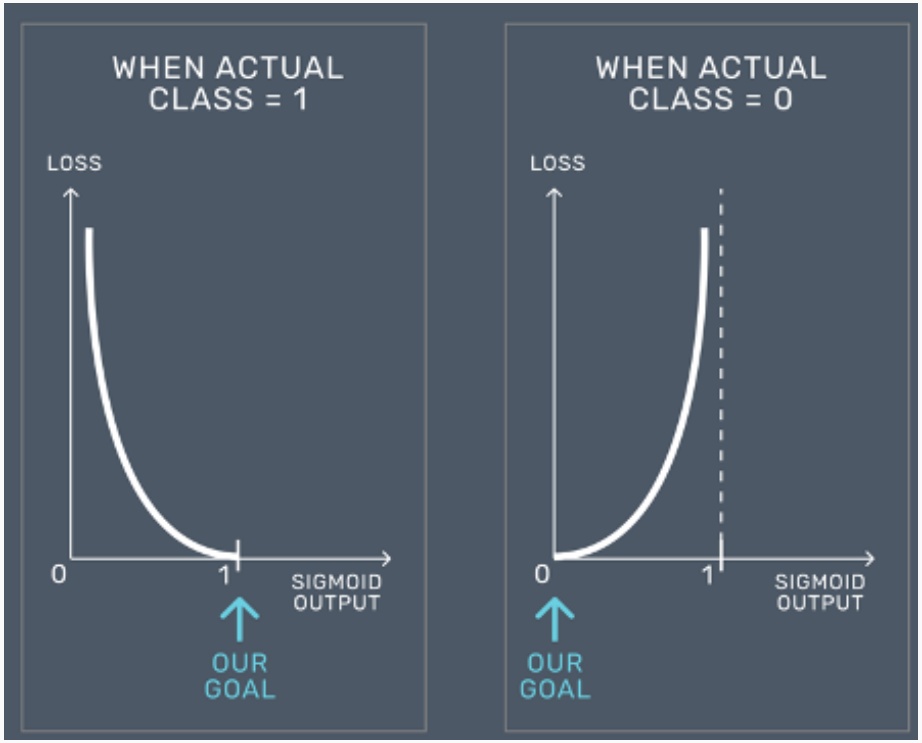
Let's recall how a loss is conceived

- For a regression task, we are using MSE, since it does have a property of having a function that posses a minimum value (a convex function)
- If we use MSE for a classification tasks, our function will have multiple minimum value, hence not convex.



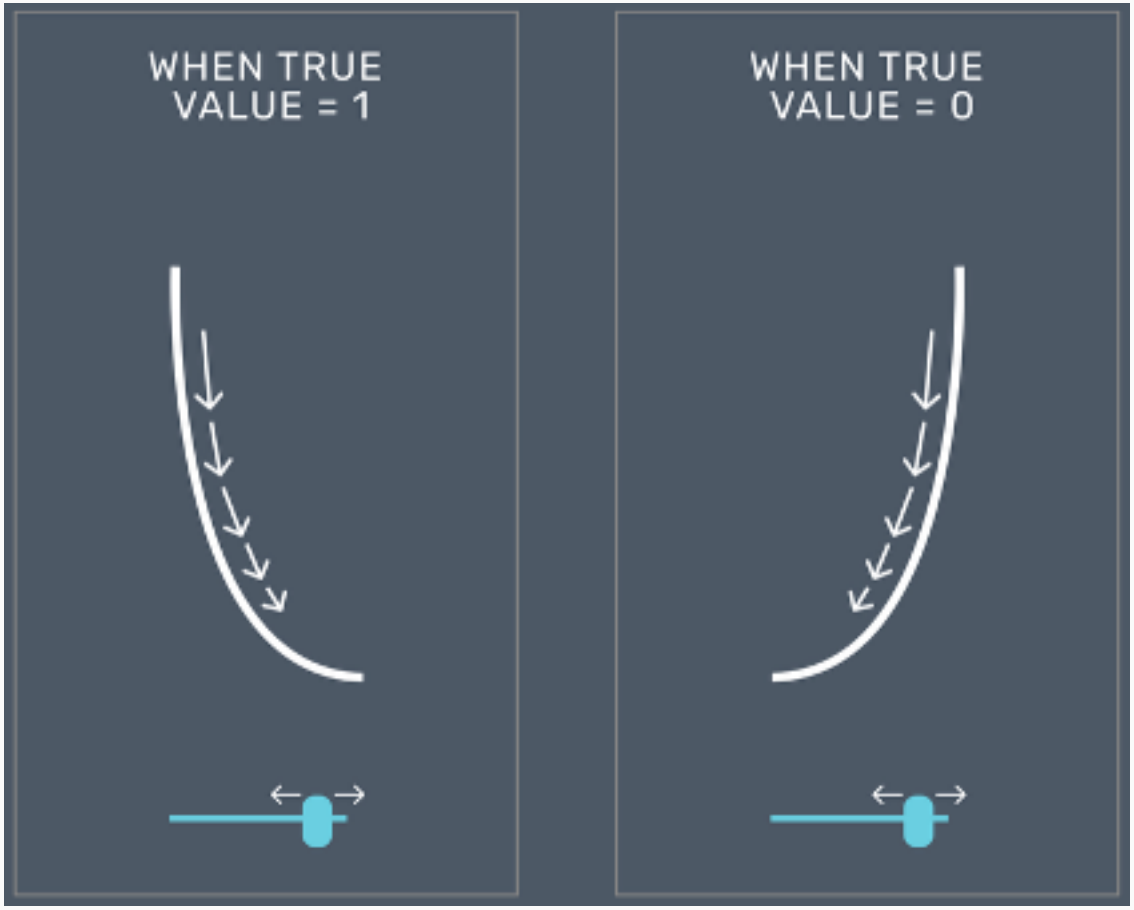
Single Neural Network visually explained

We are going to switch our loss function into Binary Cross Entropy ! So we can breakdown our classification values into a binary. So we'll have two curves to optimize since we now have a convex function :



Single Neural Network visually explained

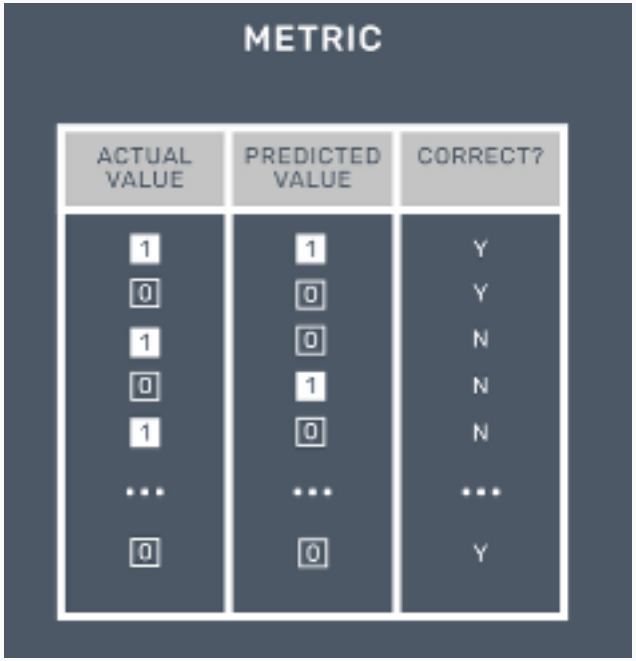
All we have to do is to apply our gradient descent (SGD) we've seen previously



Single Neural Network visually explained

Wait ! There's still some work to be done !

- *We also used the MSE as a Metric, so we can ensures that our model learnt something*
- *We should change our metric, it can be Accuracy*



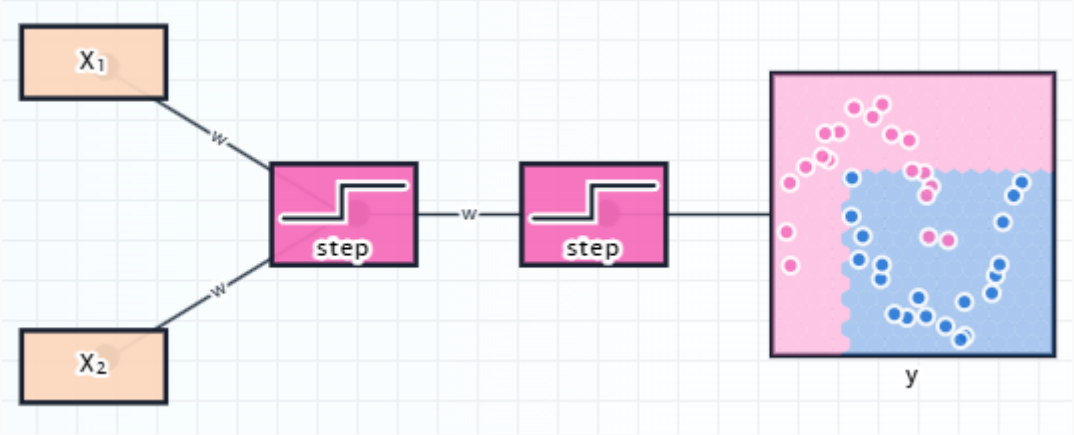
$$\text{ACCURACY} = \frac{\text{TOTAL CORRECT}}{\text{TOTAL PREDICTIONS}}$$

03
Multi Layer Perceptron

Multi Layer Perceptron

What if I want a more complexe neural network ? By adding a neuron or changing activation function into a step, you have now a Multi-Layer Perceptron.

By chaining multiple neuron together, we truly have a neural network !



Multi Layer Perceptron

ℹ | Definition : **Neural Network Architecture**

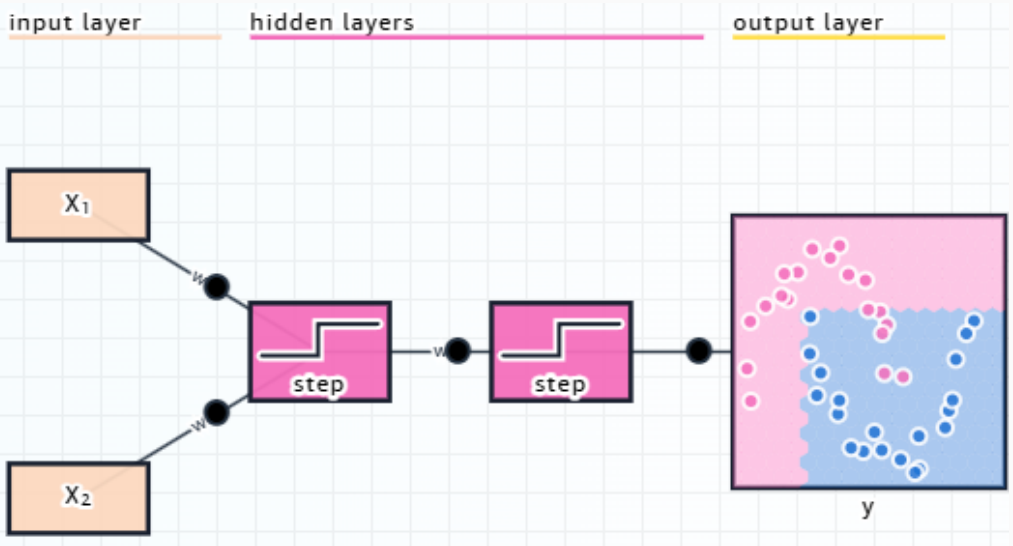
In general, a neural network architecture (a configuration) consist of three layers types:

- **Input layer**: A layer with a node for each network input
- **hidden layer(s)**: A layer full of artificial neurons.
- **output layer**: A layer representing the network's output

Each layer has it's own number of neurons, or units.

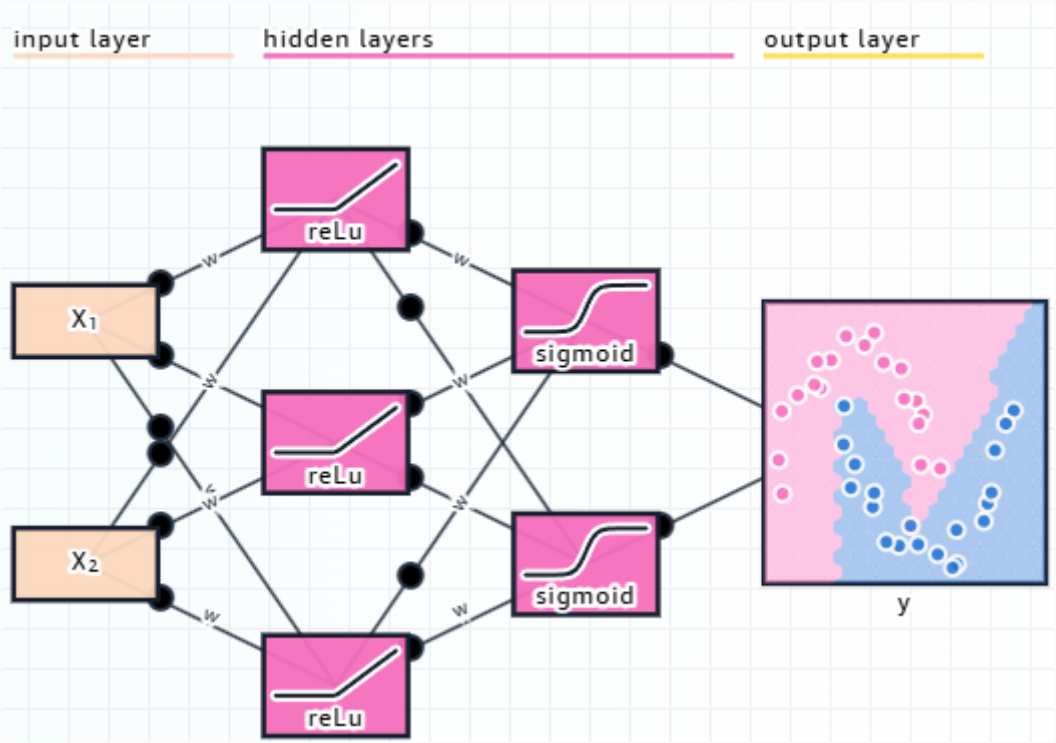
⚠ | Warning

There should be only one input and one output layer, but there may be an arbitrary number of hidden layers.



Multi Layer Perceptron

Neural networks can be **wide**: having many neuron in a given hidden layer, or **deep** having many hidden layer in the network. It's up to you to choose the right amount of neuron without over-fitting, and have a acceptable computational cost.



Multi Layer Perceptron

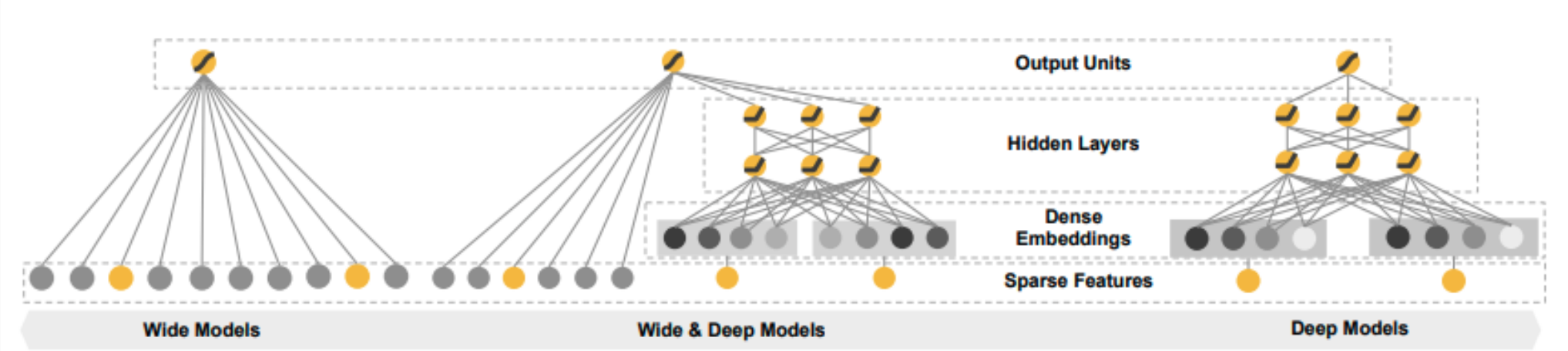


Figure 1: The spectrum of Wide & Deep models.

https://arxiv.org/abs/1606.07792v1

04
Example of code implementation

Example of code implementation

Applied example : Linear Regressor with a neuron. PyTorch

If we were to implement this through a code, we will be using PyTorch implementation. Hence, you can notice this implementation matches this conference visually explained neural network

```py
import torch
# Create a linear model with 1 inpute and 1 target
class LinearModel(nn.Module):
  def __init__(self):
    super(LinearModel, self).__init__()
    self.linear = nn.Linear(1, 1)

  def forward(self, x):
    return self.linear(x) # This is our Linear Activation

model = LinearModel() # Initialize our model
criterion = nn.MSELoss() # Setting our loss metrics to MSE
# Setting our Gradient Descent, as shown visually earlier
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```py
# training loop
for epoch in range(epochs)
  inputs = torch.from_numpy(x_train)
  target = torch.from_numpy(y_train)

  optimizer.zero_grad() # Clear any gradient calculation, we dont want them to carry over the next iteration loop.
  ouputs = model(inputs) # Launch prediction, store them in outputs
  loss = criterion(outputs, target) # Calculate loss value for the predicted output
  loss.backward() # Calculate feedback gradient
  optimizer.step() # Apply parameters from feedback gradient
  print("Epoch {}, Loss {}".format(epoch, loss.item()))
```

Example of code implementation

Applied example : Linear Regressor with a neuron. PyTorch

Now to launch prediction with a trained model, we need to implements the inference code like this

```py
import torch

with torch.no_grad(): # We don't need gradient in the inference
  predicted = model(torch.from_numpy(x_test)) # Launch prediction
  print(predicted) # Print prediction
```

Note : PyTorch might feel overwhelming at first, but it is the most used framework for researching, including open source LLM !

Example of code implementation

Applied example : MLP Tensorflow + Keras for image classification (10 classes)

```py
model = tf.keras.models.Sequential([
  tf.keras.layer.Flatten(input_shape(28, 28)),
  tf.keras.layer.Dense(128, activation='relu'),
  tf.keras.layer.Dropout(0.20), # Deactivate randomly 20% of the network
  tf.keras.layer.Dense(128, activation='relu'),
  tf.keras.layer.Dense(10)
])
model.compile(
  optimizer=tf.keras.optimizer.Adam(0.001), # Self adjusting LR
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=[tf.keras.metrics.SparseCategoricalAccuracy()]
)
model.fit(ds_train, epochs=6, validation_data=ds_test) # Do everything we said in the training loop
```

Less popular than PyTorch, but beginner friendly.

Example of code implementation

Sidenode : Backend architecture. (Specific for PyTorch)

As you may know, we have multiple hardware available for training.

- `mps` for Mac Silicon chip (M1, M2, M3, M4, M5)
- `cuda` for Nvidia CUDA and AMD ROCm support
- `MTIA` for META (Facebook) TPU Support (Meta Training and Inference Accelerator)
- `XPU` For Intel GPU

Question is : **How to check and enable automatically a backend accelerator ?**

```py
if torch.accelerator.is_available():
    device = torch.accelerator.current_accelerator()
else
    device = torch.device("cpu")
```

05
Multiclass-classification

Multiclass-classification

Let's make it more complicated by introducing a new type of problem : A multiclass-classification.

For that, we need to predict the following classes :

- **Silver**
- **Gold**
- **Bronze**

The label is a category, which means that order is not implied, we just want to classify them without worrying about which class is better than which.

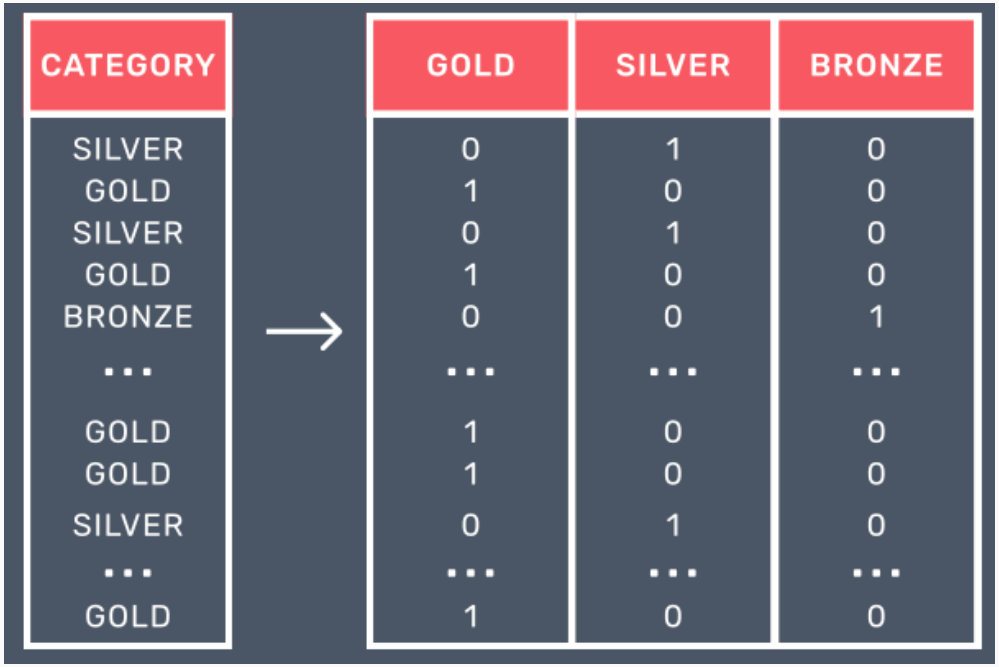
	DIST (MI)	RATING	CATEGORY
TRAINING DATA (24)	0.2	3.5	SILVER
	0.2	4.8	GOLD
	0.5	3.7	SILVER
	0.7	4.3	GOLD
	0.8	2.7	BRONZE
	1.5	3.6	SILVER
	1.6	2.6	BRONZE
	2.4	4.7	GOLD
	3.5	4.2	SILVER
	3.5	3.5	SILVER
	4.6	2.8	BRONZE
	4.6	4.2	SILVER
	4.9	3.8	SILVER
	6.2	3.6	SILVER
	6.5	2.4	BRONZE
	8.5	3.1	BRONZE
	9.5	2.1	BRONZE
	9.7	3.7	BRONZE
	11.3	2.9	BRONZE
	14.6	3.8	SILVER
	17.5	4.6	GOLD
	18.7	3.8	SILVER
	19.5	4.4	GOLD
	19.8	3.6	SILVER
TEST DATA (8)	0.3	4.6	GOLD
	0.5	4.2	GOLD
	1.1	3.5	SILVER
	1.2	4.7	GOLD
	2.7	2.7	BRONZE
	3.8	4.1	SILVER
	7.3	4.6	BRONZE
	19.4	4.8	GOLD

Multiclass-classification

We know that computers cannot handle string data, since we are using math.

- We are going to use the **One-Hot Encoding**.

It will create a new column for each class. Then treat each column as a binary classification output assigning 1 for yes, 0 for no.



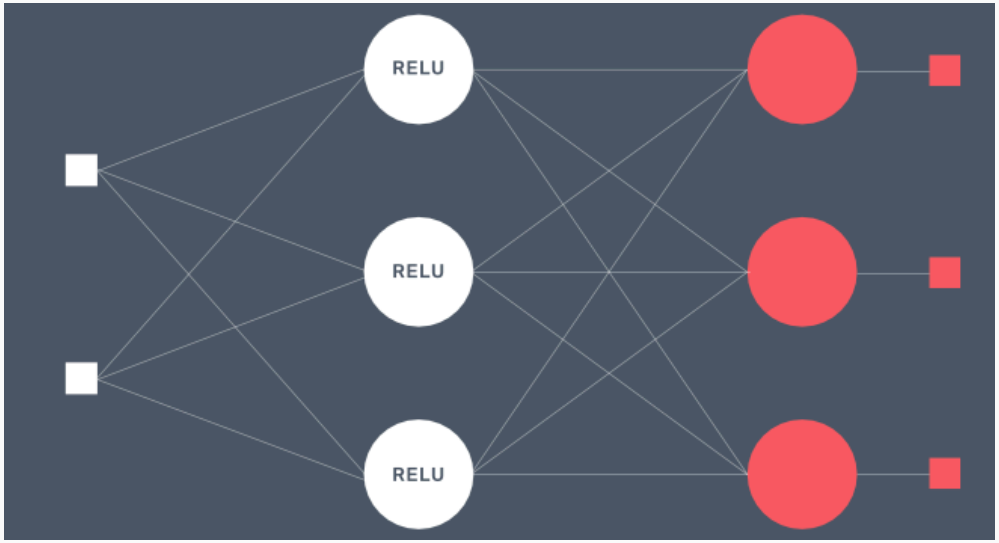
CATEGORY
SILVER
GOLD
SILVER
GOLD
BRONZE
...
GOLD
GOLD
SILVER
...
GOLD

GOLD	SILVER	BRONZE
0	1	0
1	0	0
0	1	0
1	0	0
0	0	1
...	...	...
1	0	0
1	0	0
0	1	0
...	...	...
1	0	0

Multiclass-classification

We're going to start building our network:

- this time using a another **activation function : ReLU**
- We have 2 columns as input (square)
- And having three neuron. (white circle)
- We need to match the number of class as output, so we'll have three neurons for final prediction of either 1 or 0 (in red)



Multiclass-classification

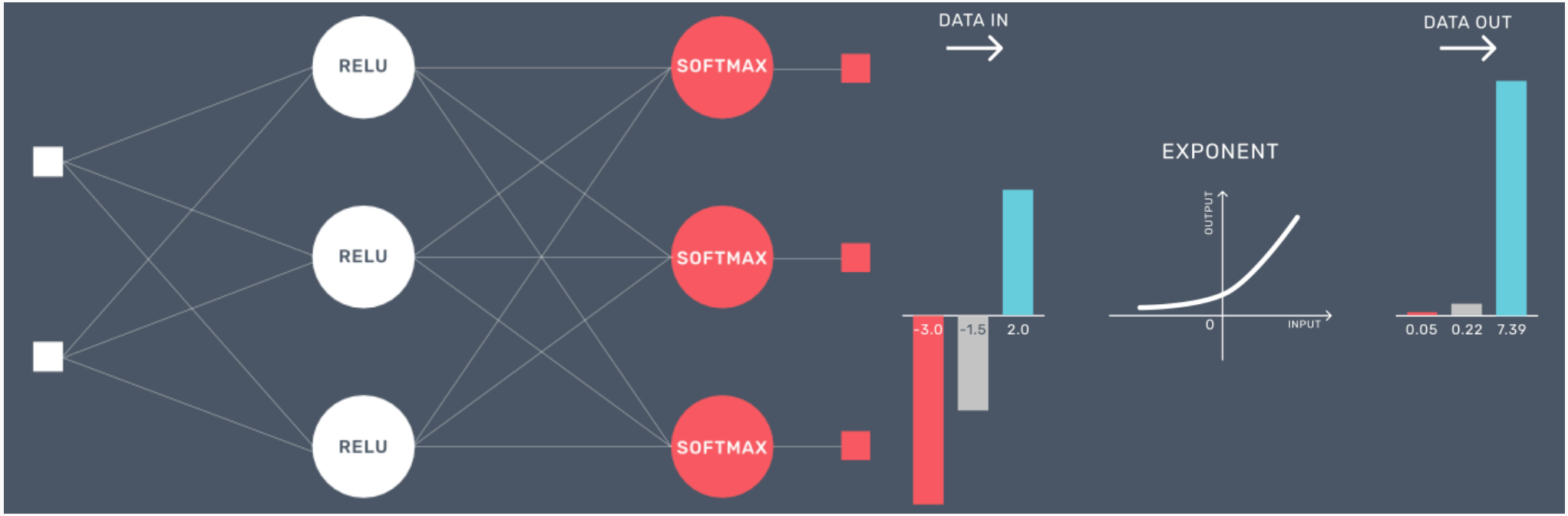
Here, we have the 5 first classification.

Take the first data point as an example. for the silver class should ideally predict 0, 1, and 0 for the first, second, and third neurons the neural network

In short neurons for each dada point, the neuron of the actual class should output 1 while other should output 0.

Multiclass-classification

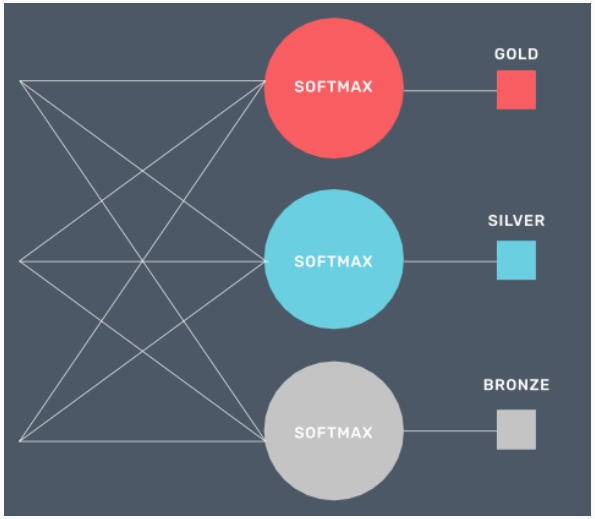
We're also changing the activation function : **Softmax** The softmax activation performs a two-step computation on its input : Exponential + Normalisation



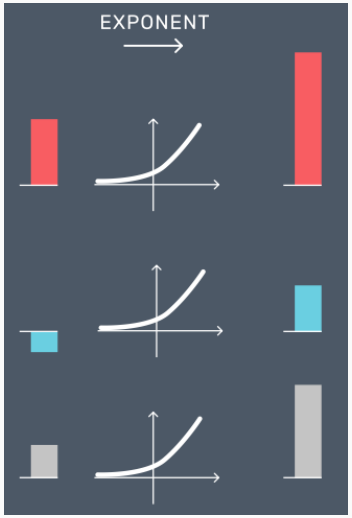
Why do we have normalization at the end ?

Multiclass-classification

1. We'll need to check for the full layer. So we have the three unit of neurons, one for each class.

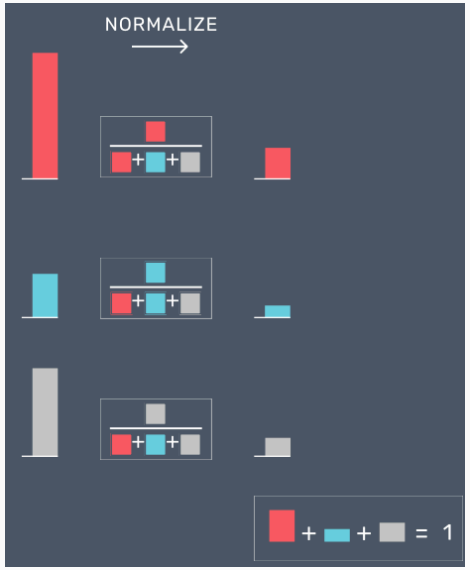


2. Each neuron performs the exponentiation on its input, which then becomes the input for the normalization step



3. Each input is divided by the sum of all inputs. This become the output of the neural network.

As a result the sum of all output will always be 1. This is a useful outcome because we can now treat the outputs as probability values

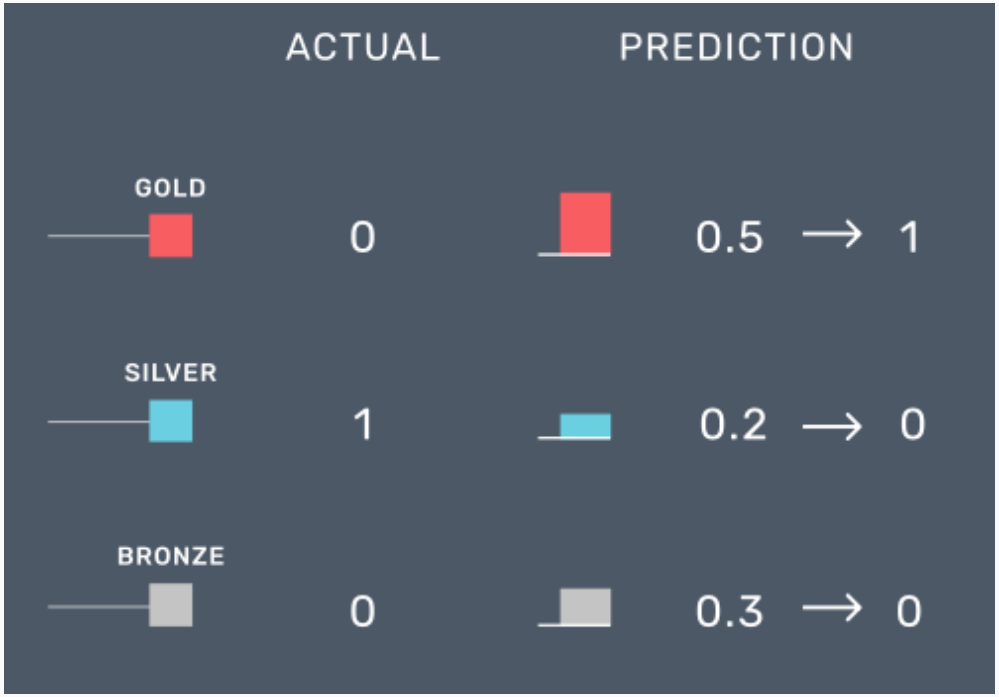


Multiclass-classification

Let’s take an example where the actual class is silver, And we suppose that each neuron’s softmax activation produce 0.5, 0.2, and 0.3

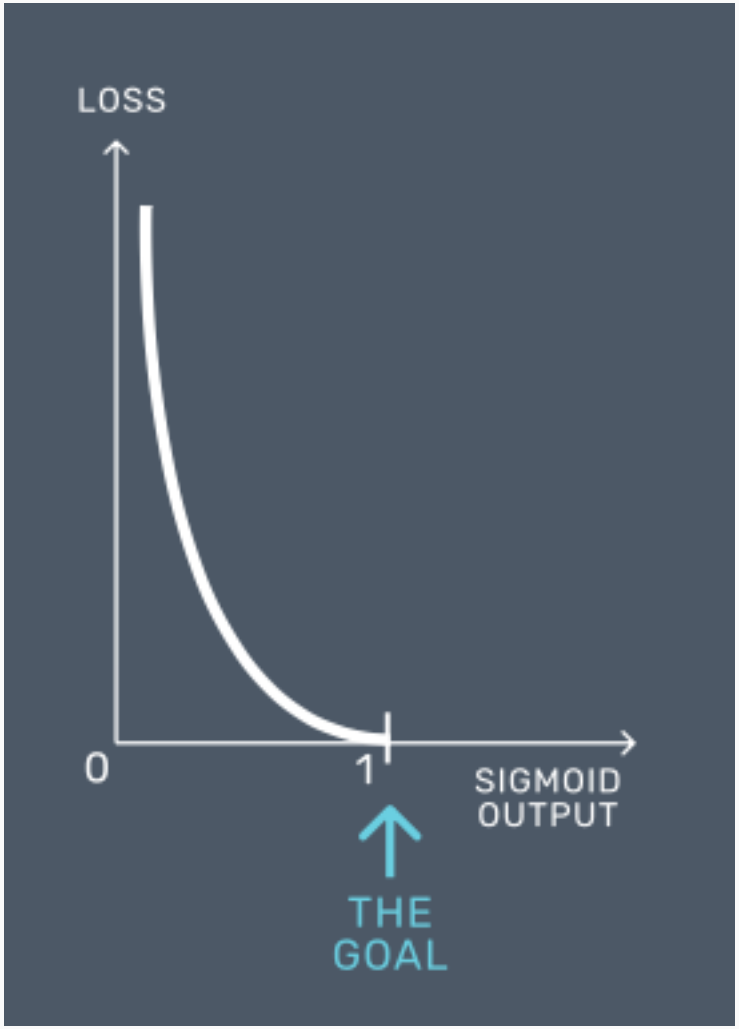
Treating them as probabilities, we assign 1 to the neuron with the largest output and 0 to the other neurons.

In this example, the predicted class does not match the actual class. This brings us to the next discussion, the loss function



Multiclass-classification

We are going to use :
CategoricalCrossEntropy basically the same as BinaryCrossEntropy but for more than 2 classes.



Multiclass-classification

Here is an example of the loss function. The actual class is Silver, so we want the NN to output the highest probability at the second neuron.

- In one of the earlier epoch, we can see that the output at the second neuron is 0.3 (bad)
- In one the later epoch, this neuron will produce 0.6 which is better (loss decreased)

06
Some popular model right know

Some popular model right know

- General use case : LSTM, RNN, GAN, CNN, ...
- LLM via Transformers : **NEW SINCE 15 JULY 2025 : KIMI-K2** Mistral, DeepSeek, Qwen, Stable-Diffusion, Grok, GPT, Claude..
- Computer Vision: Yolo, EfficientNet, ResNet..

Common points : they are all feed-forward based archicture.

- Deep Reinforcement Learning Algorithms : Q-Learning, Deep Q-Learning, Double Deep Q-Learning..

Thank you for your attention !

This is the end of the morning session !

Bon appétit !

References

- Patrick Gallinari - Introduction to Deep Learning (Sorbonne Université)
- https://www.tensorflow.org/datasets/keras_example
- https://mlu-explain.github.io/neural-networks/
- https://kdimensions.com/