DATE : 25.03.2024

DT/NT : NT

LESSON : DEEP LEARNING

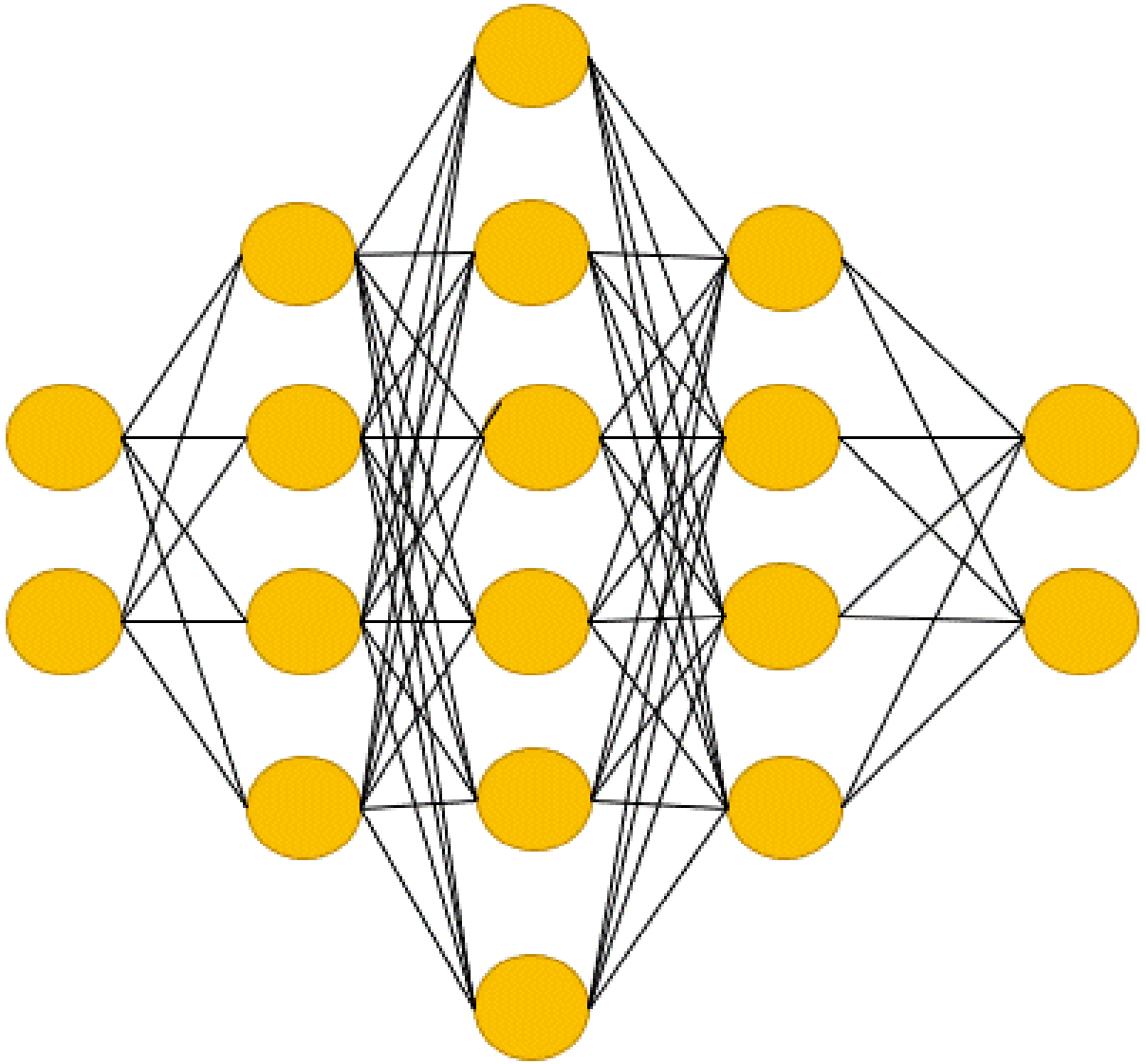
SUBJECT: Deep Learning - ANN

BATCH : B223

TECHPRO EDUCATION


Let me ask,
Gru

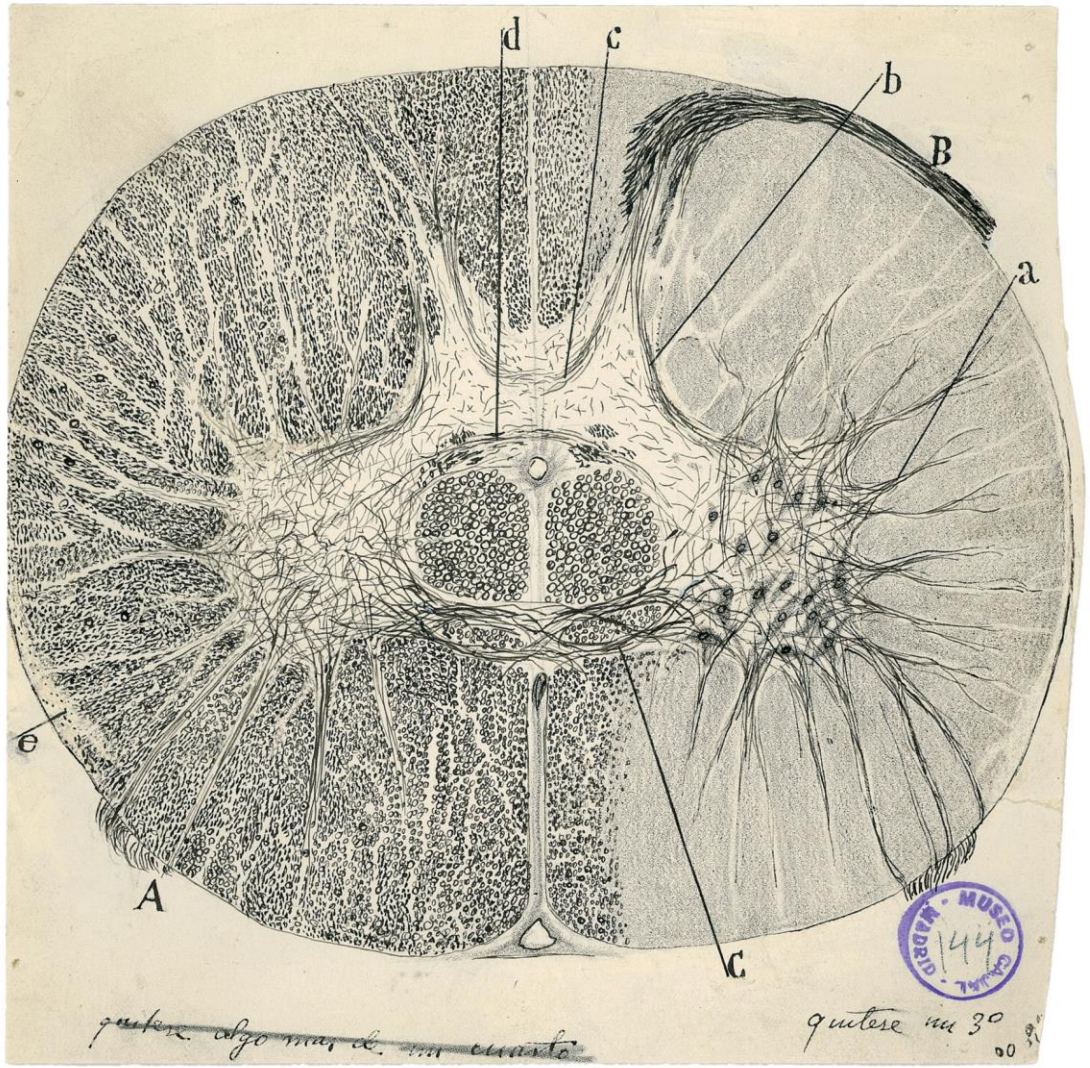

What is the Price
of the Jet, Bob?
It must be 50K

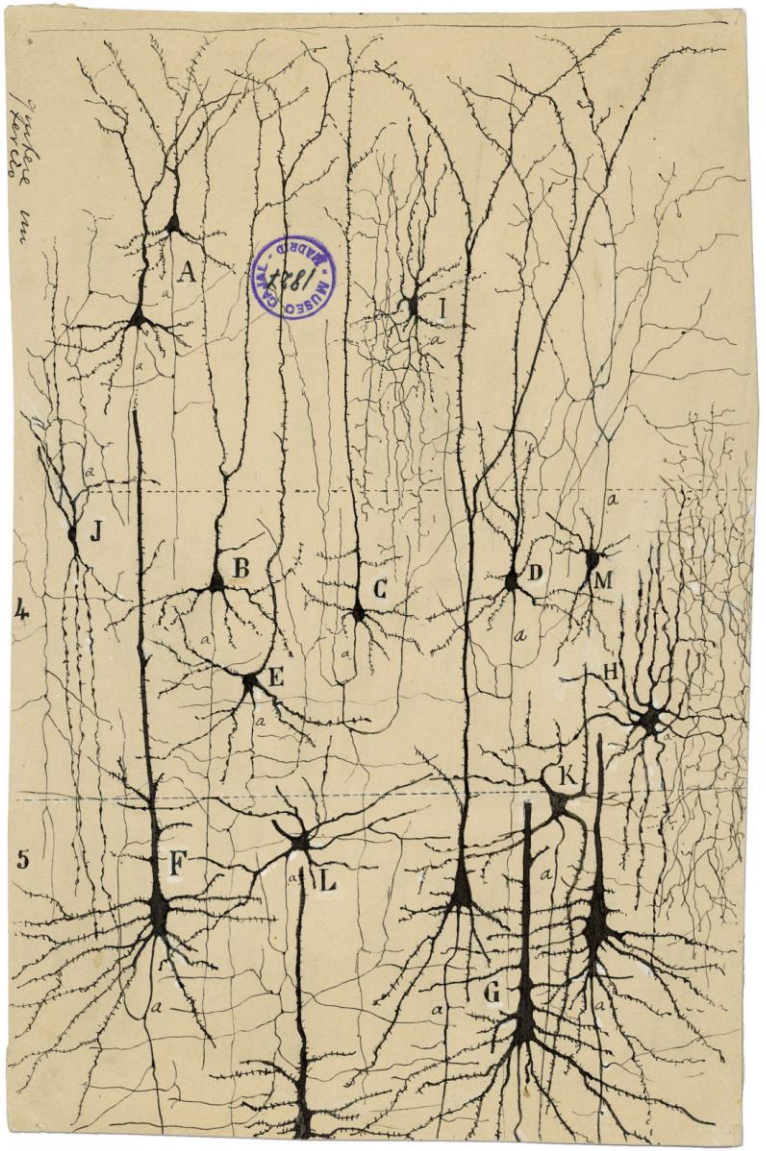


ANN

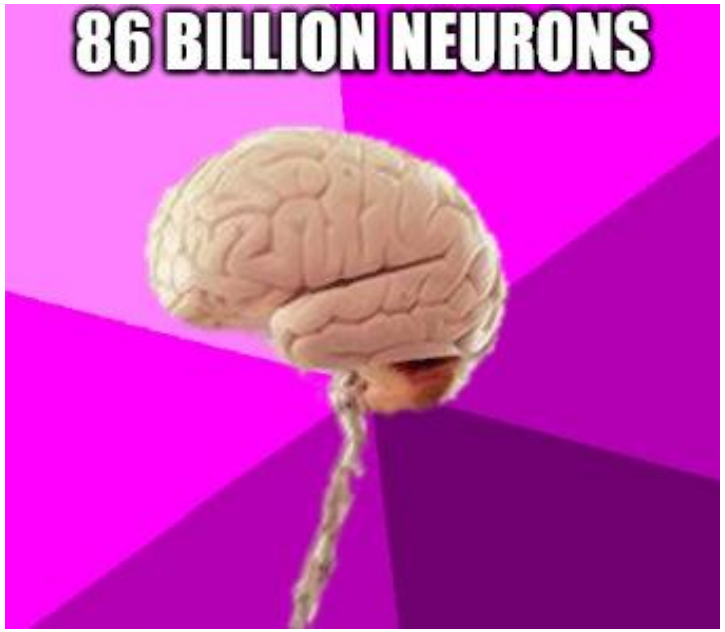
ARTIFICIAL NEURAL NETWORK

ARTIFICIAL NEURAL NETWORK

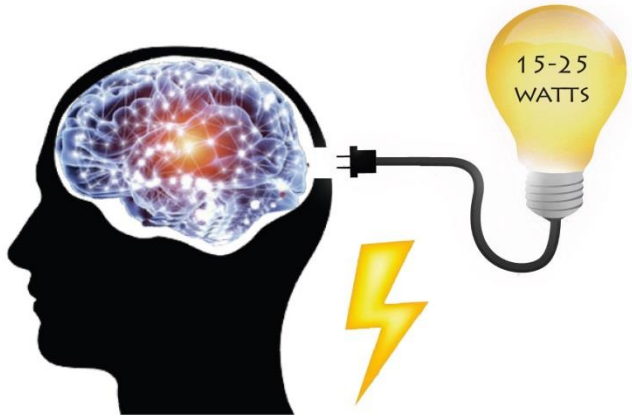




ARTIFICIAL NEURAL NETWORK

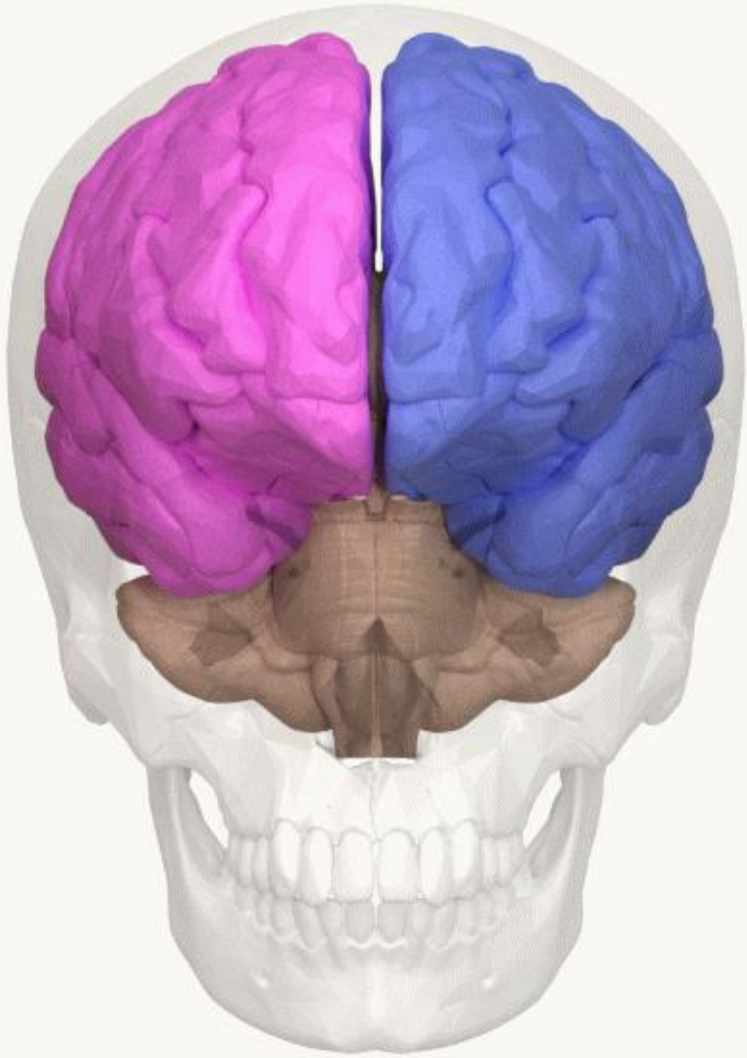


ARTIFICIAL NEURAL NETWORK

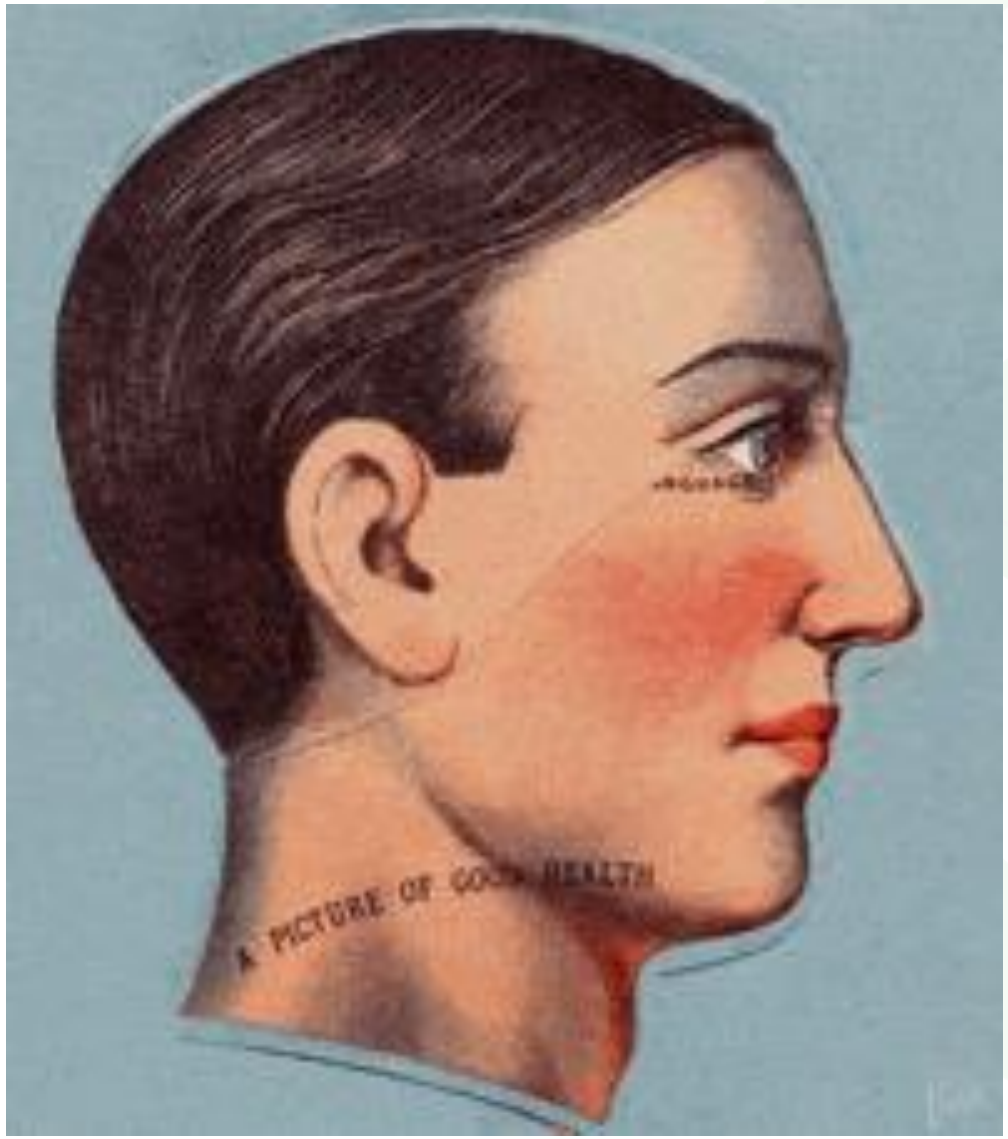


PER MINUTE

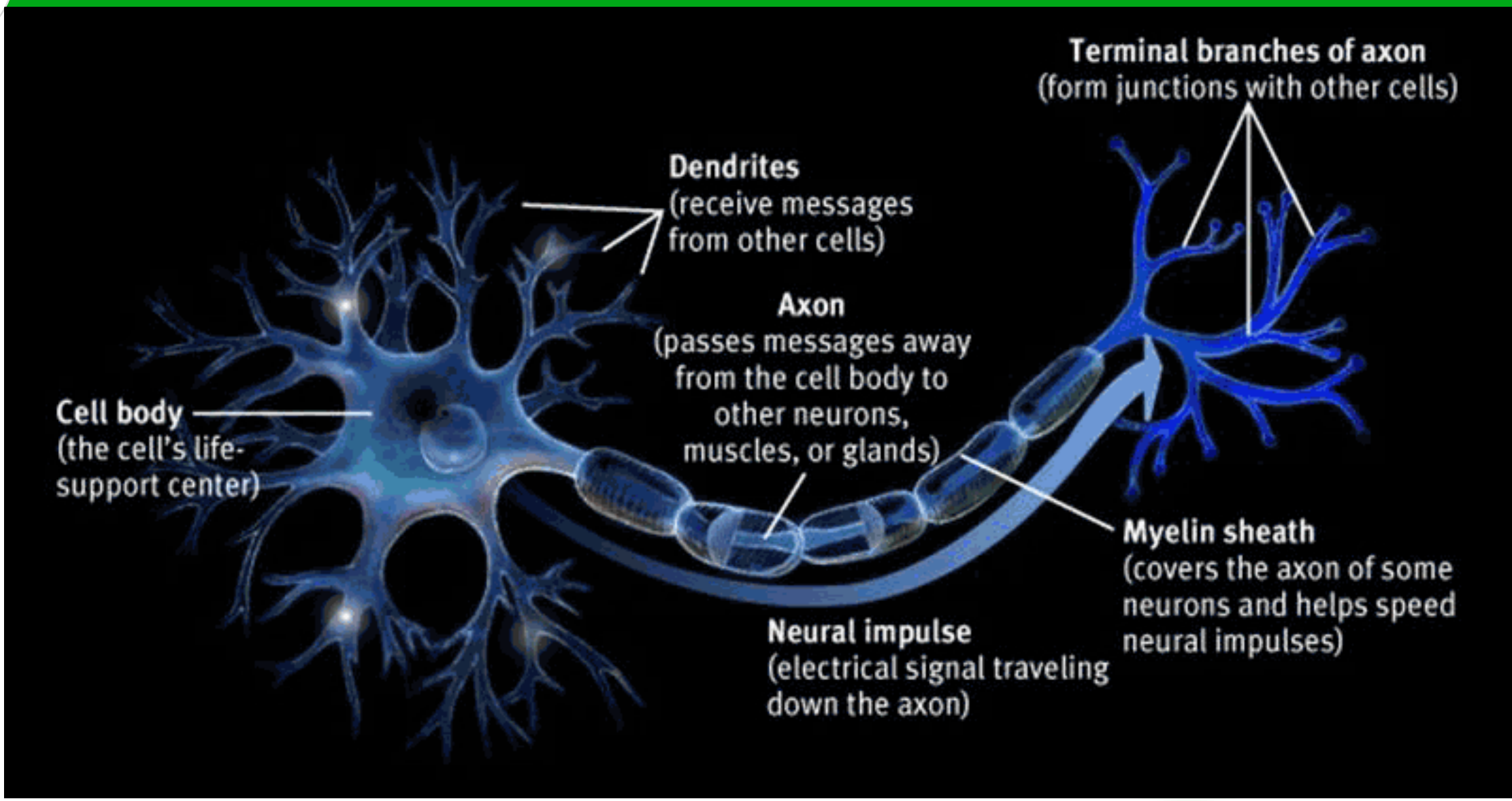
1.000.000.000.000.000 byte



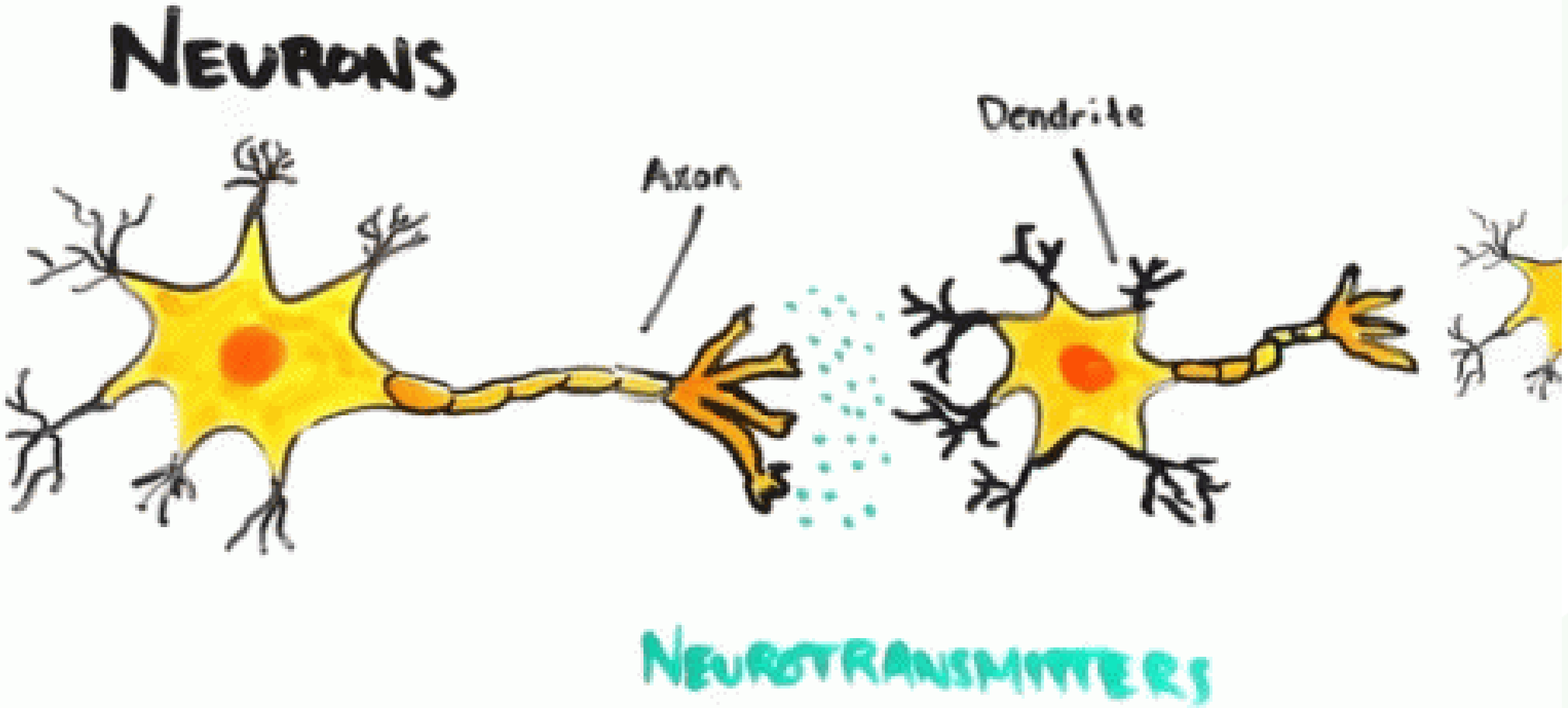
ARTIFICIAL NEURAL NETWORK



ARTIFICIAL NEURAL NETWORK



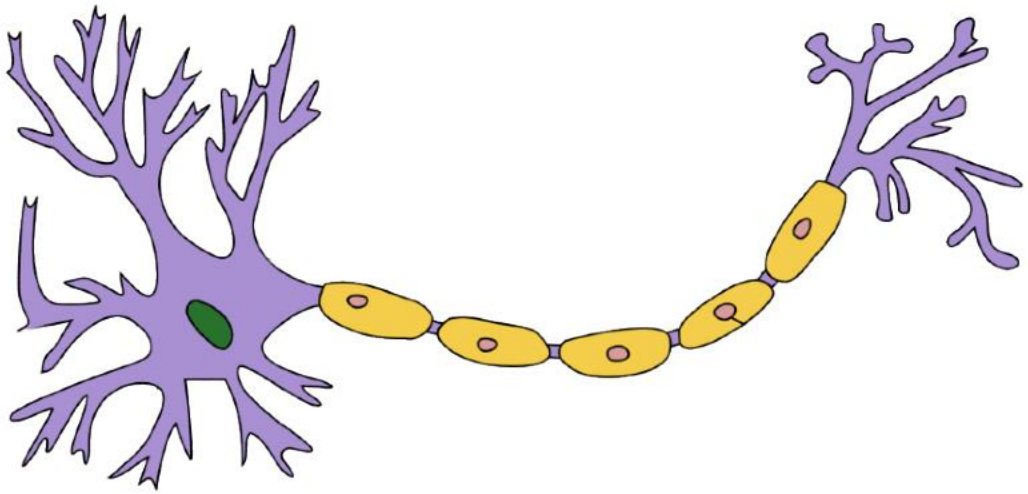
ARTIFICIAL NEURAL NETWORK



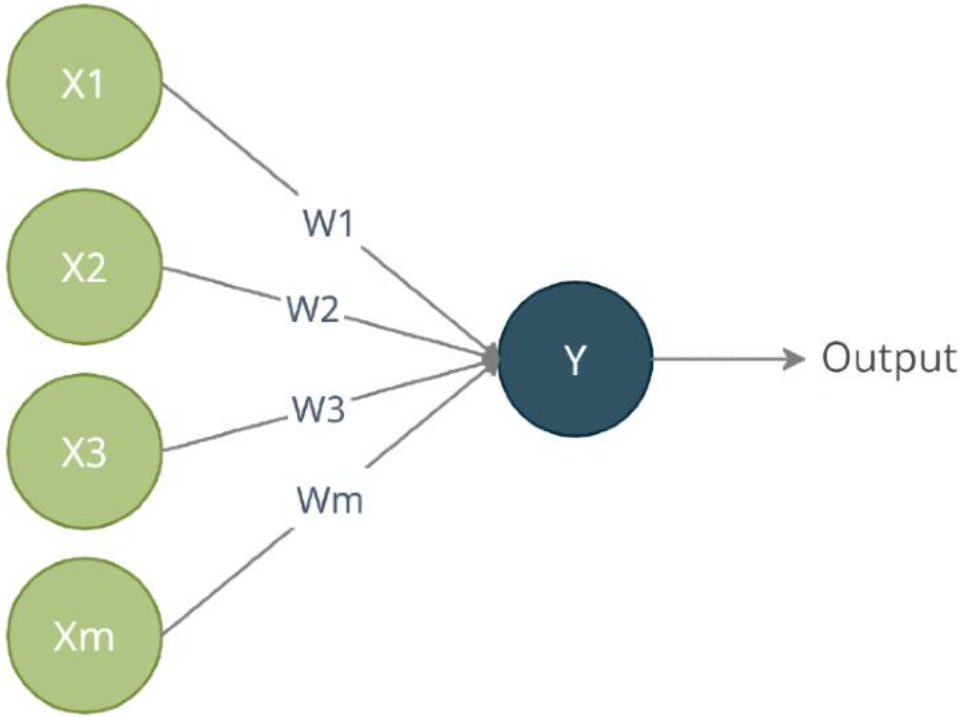
PERCEPTRON MODELS

PERCEPTRON MODELS

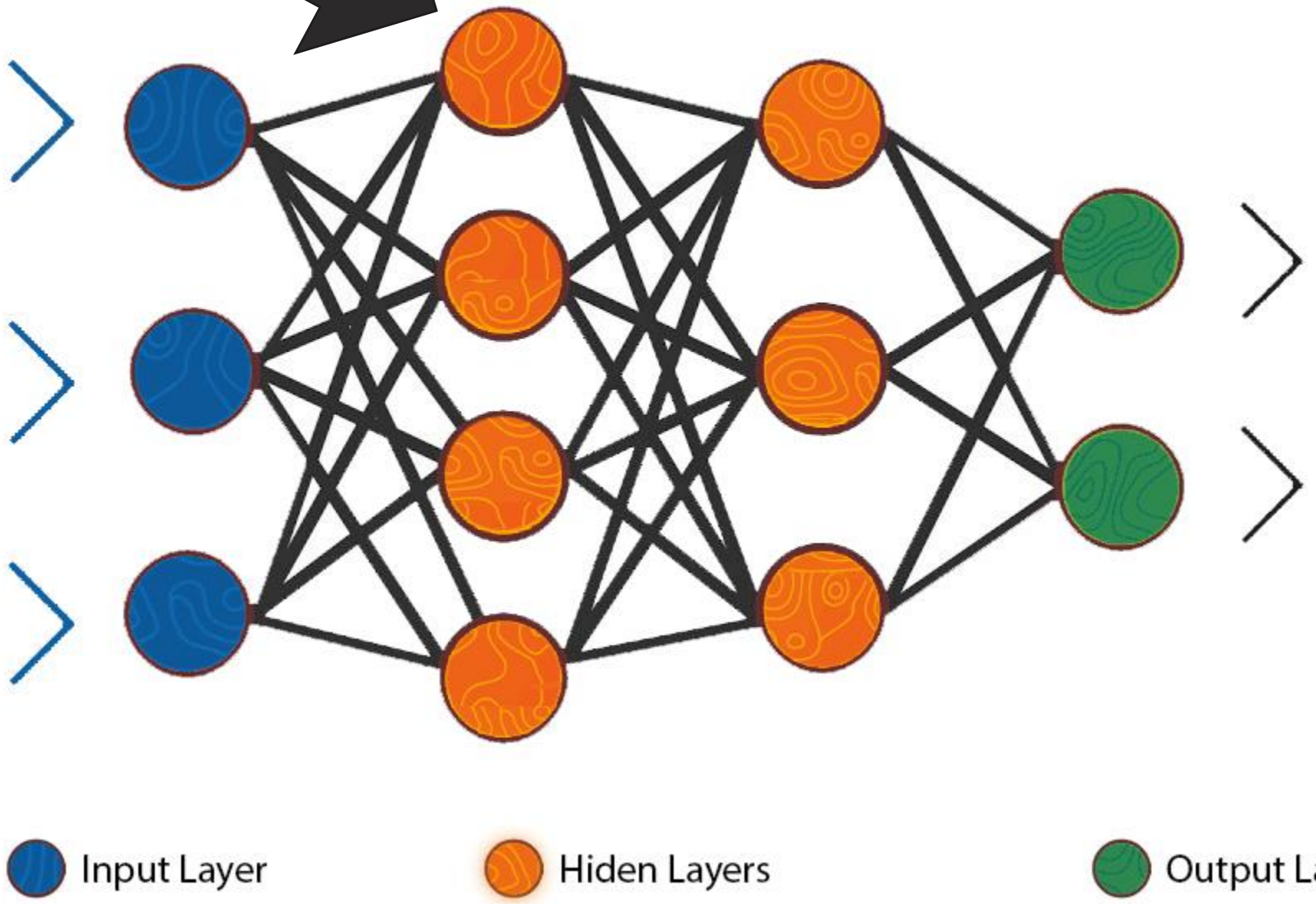
Neuron



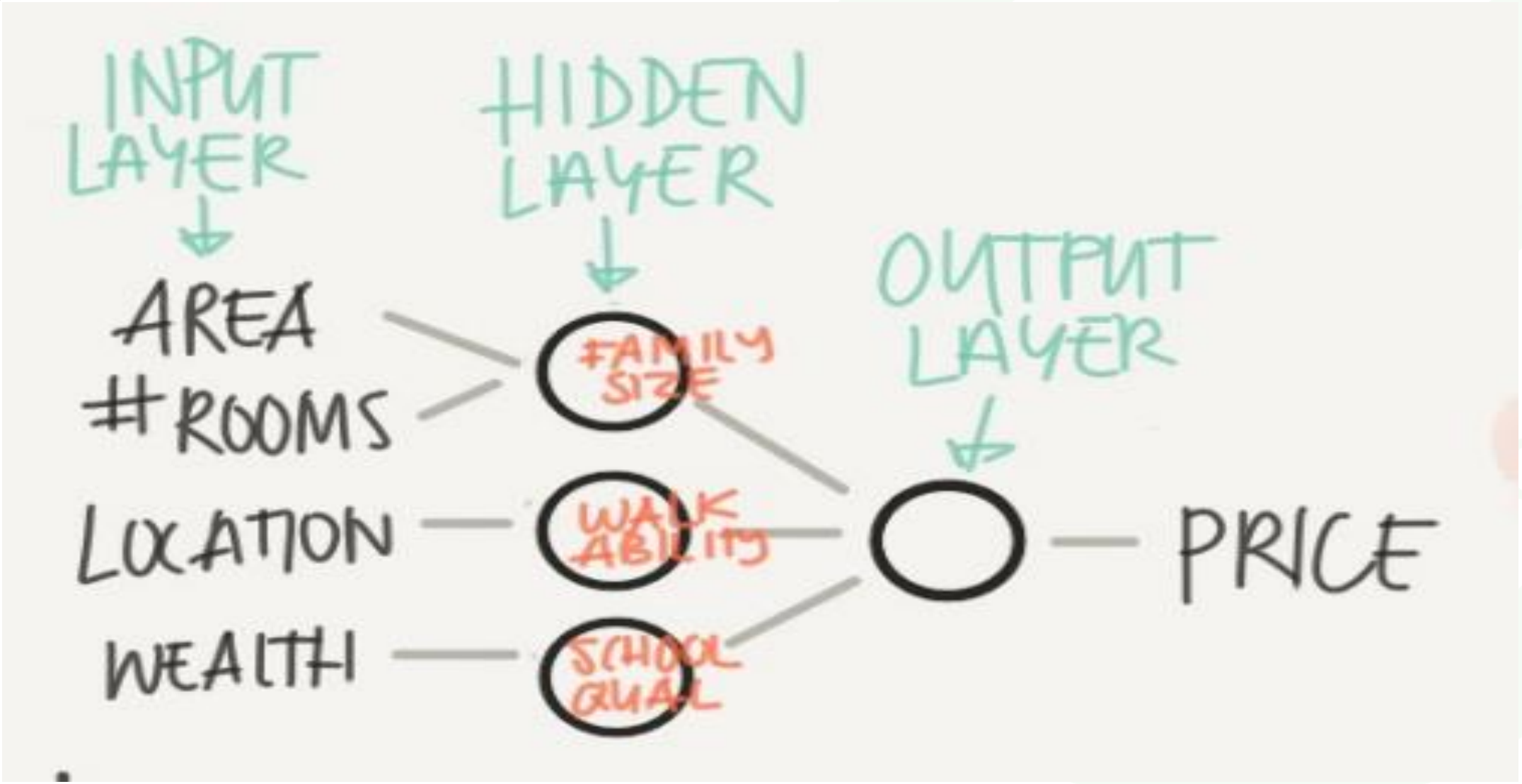
Perceptron



NEURON (NODE)



DEEP LEARNING



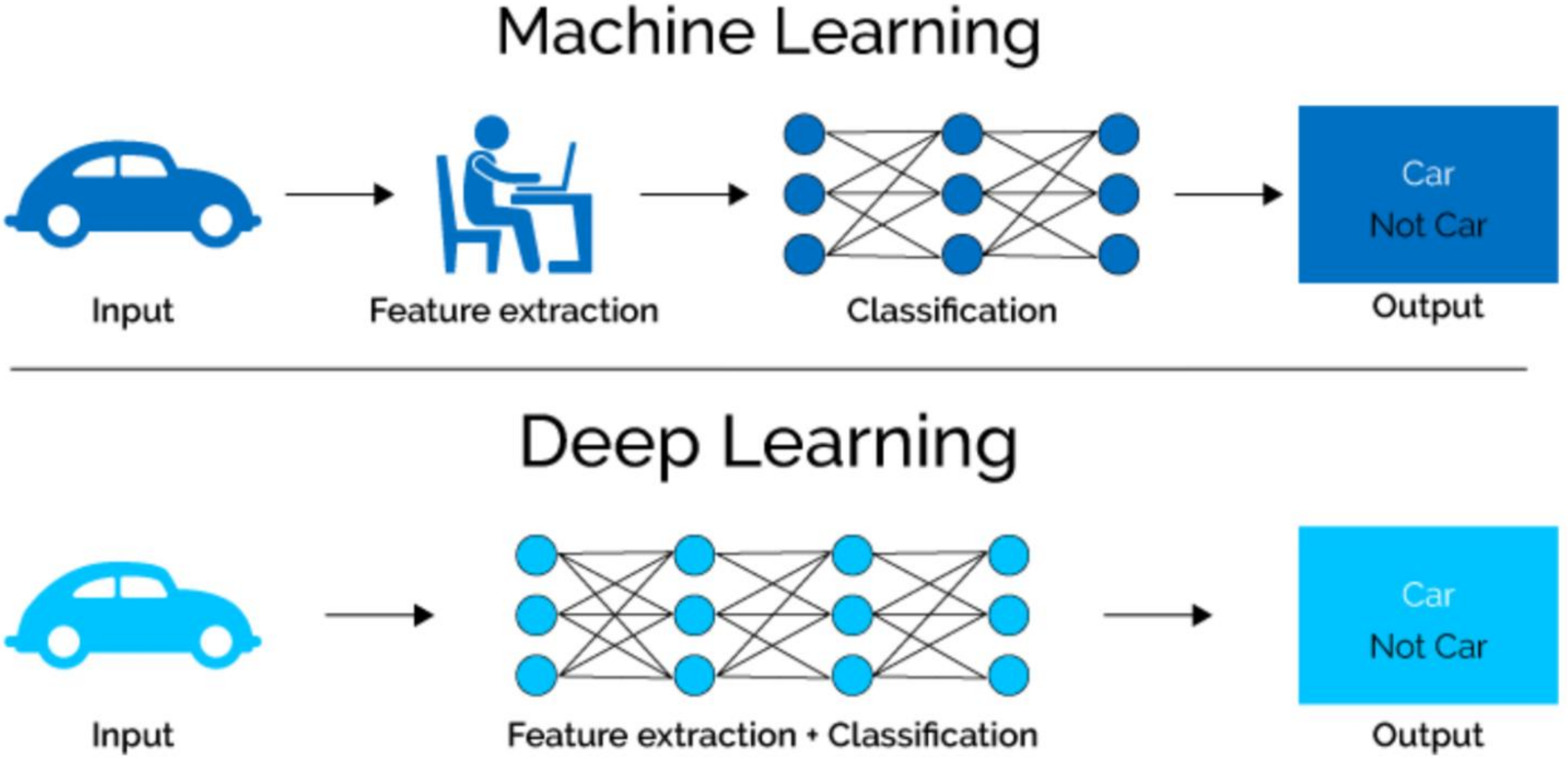
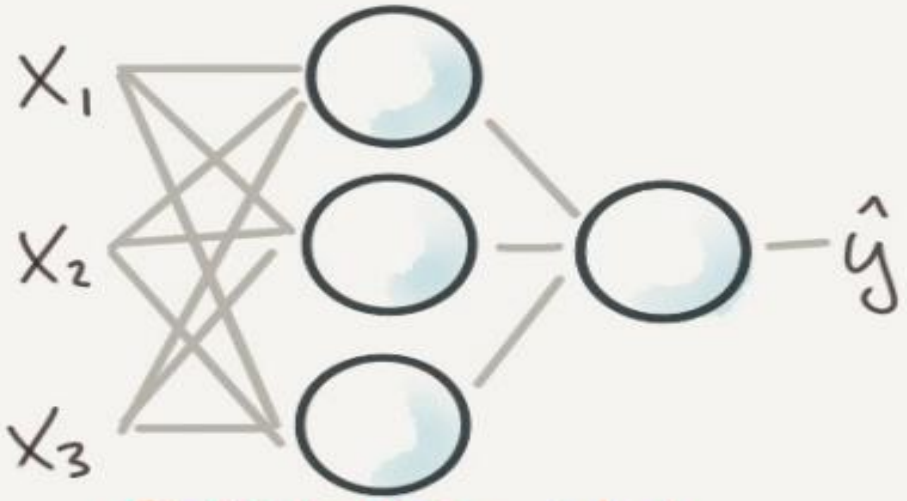
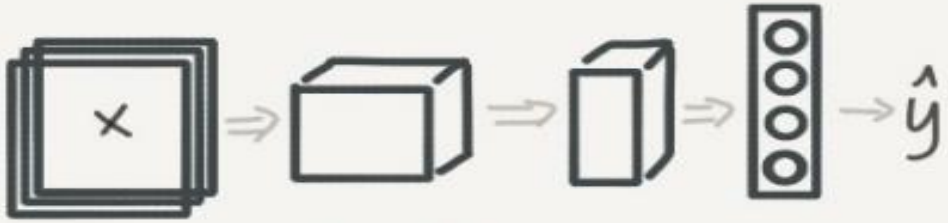


Figure 1: Machine Learning VS Deep Learning

DEEP LEARNING

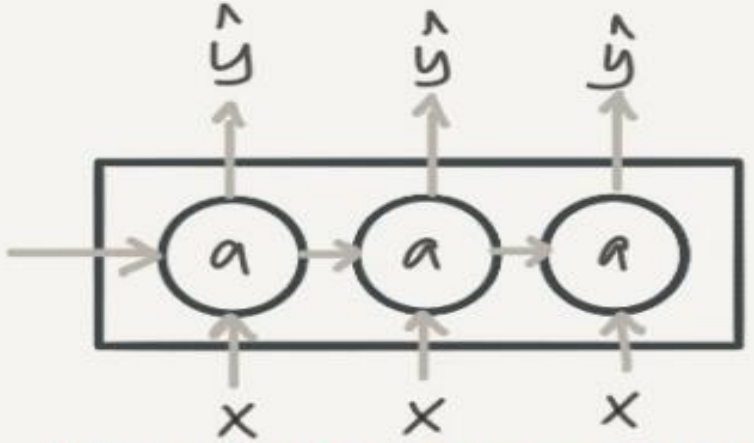


STANDARD NN



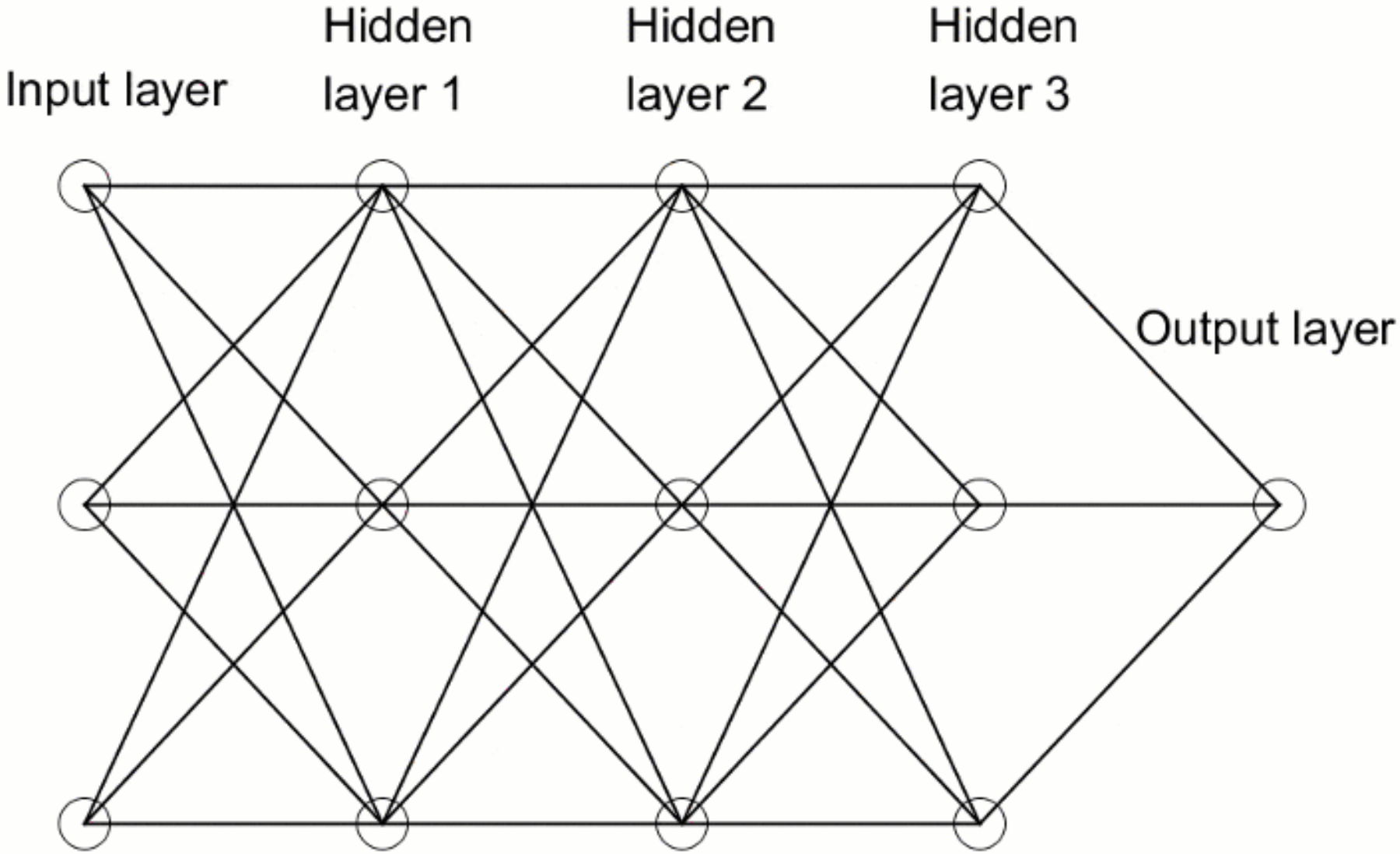
CONVOLUTIONAL NN

NETWORK ARCHITECTURES

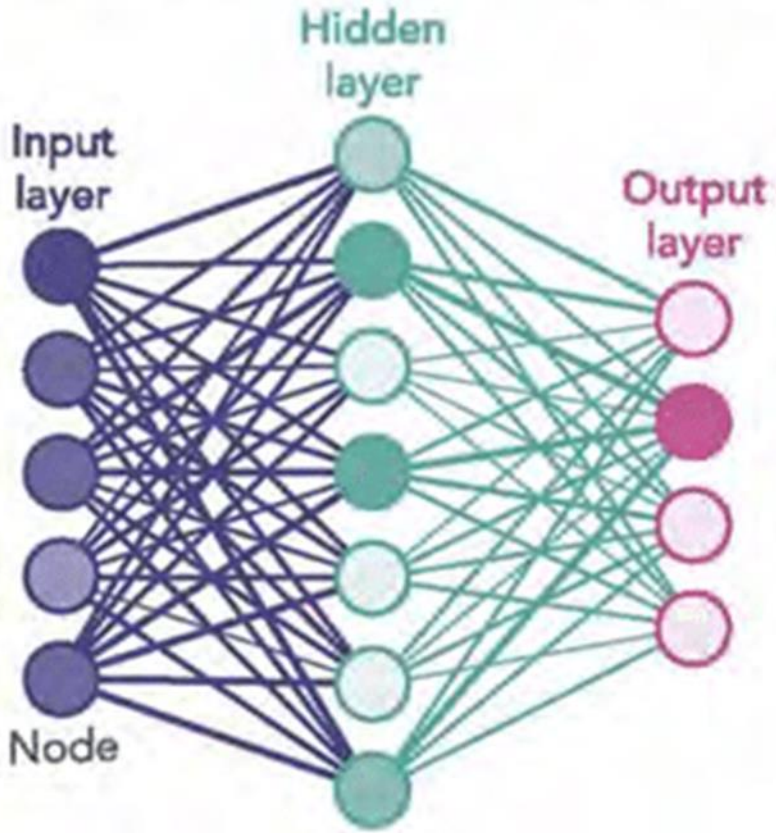


RECURRENT NN

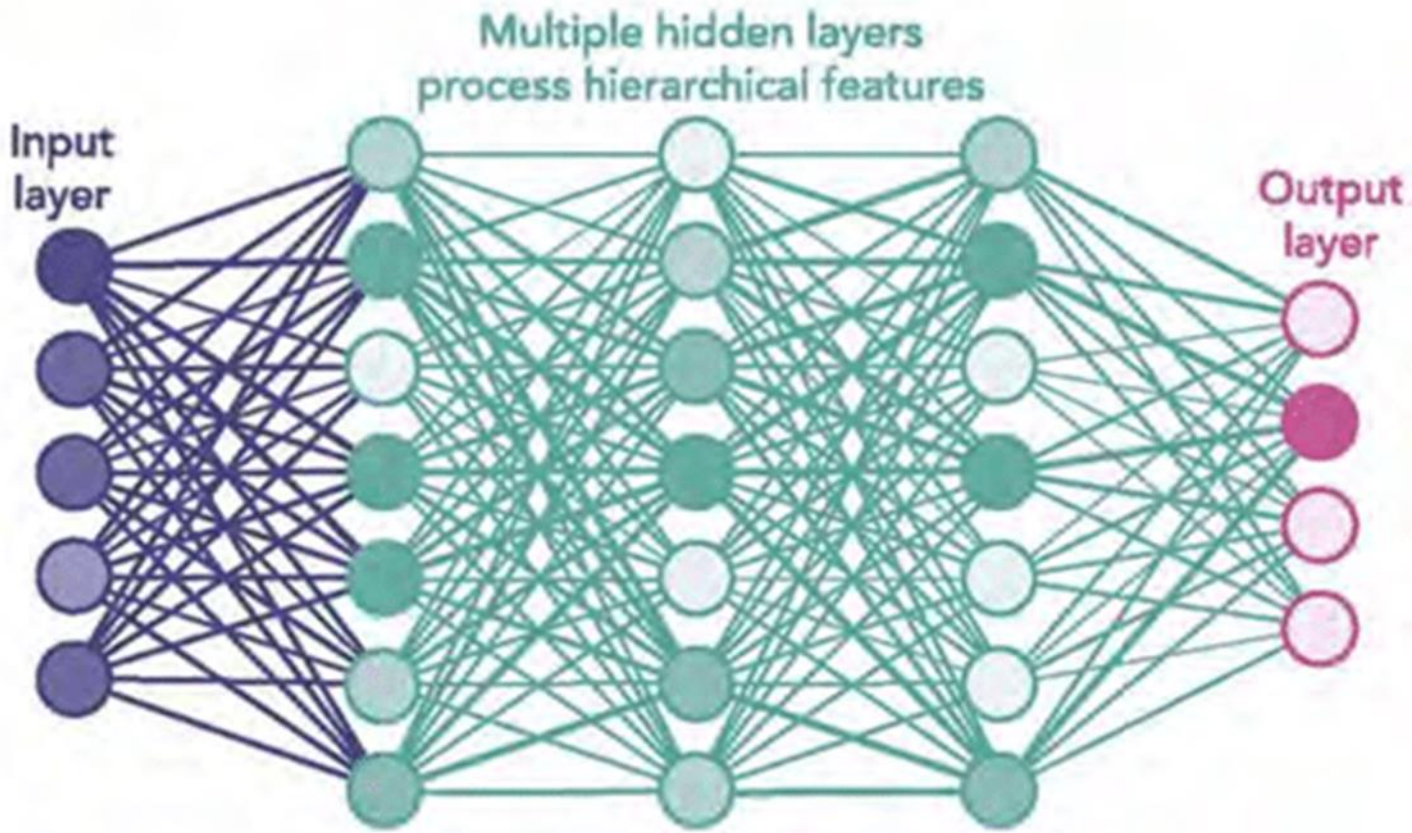
DEEP LEARNING



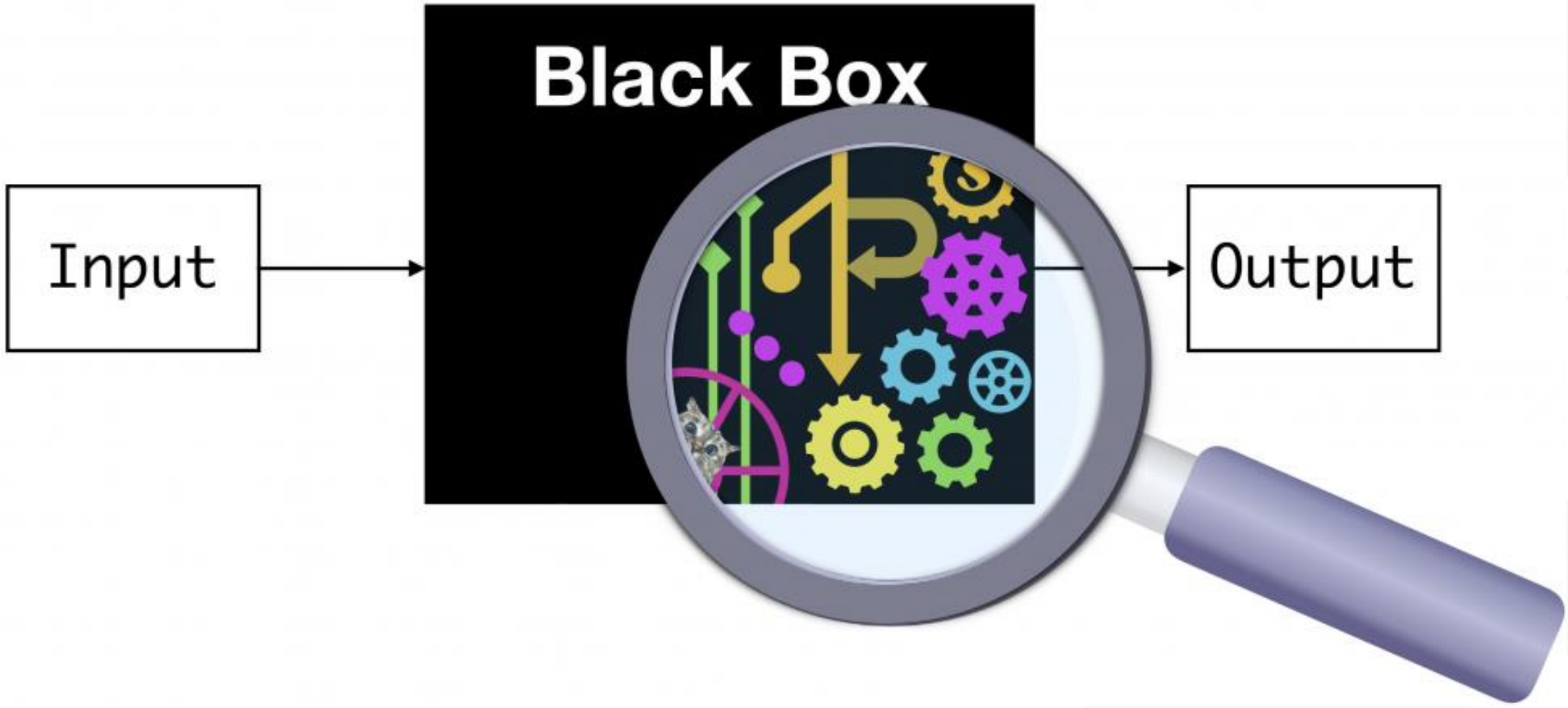
SHALLOW NEURAL NETWORK



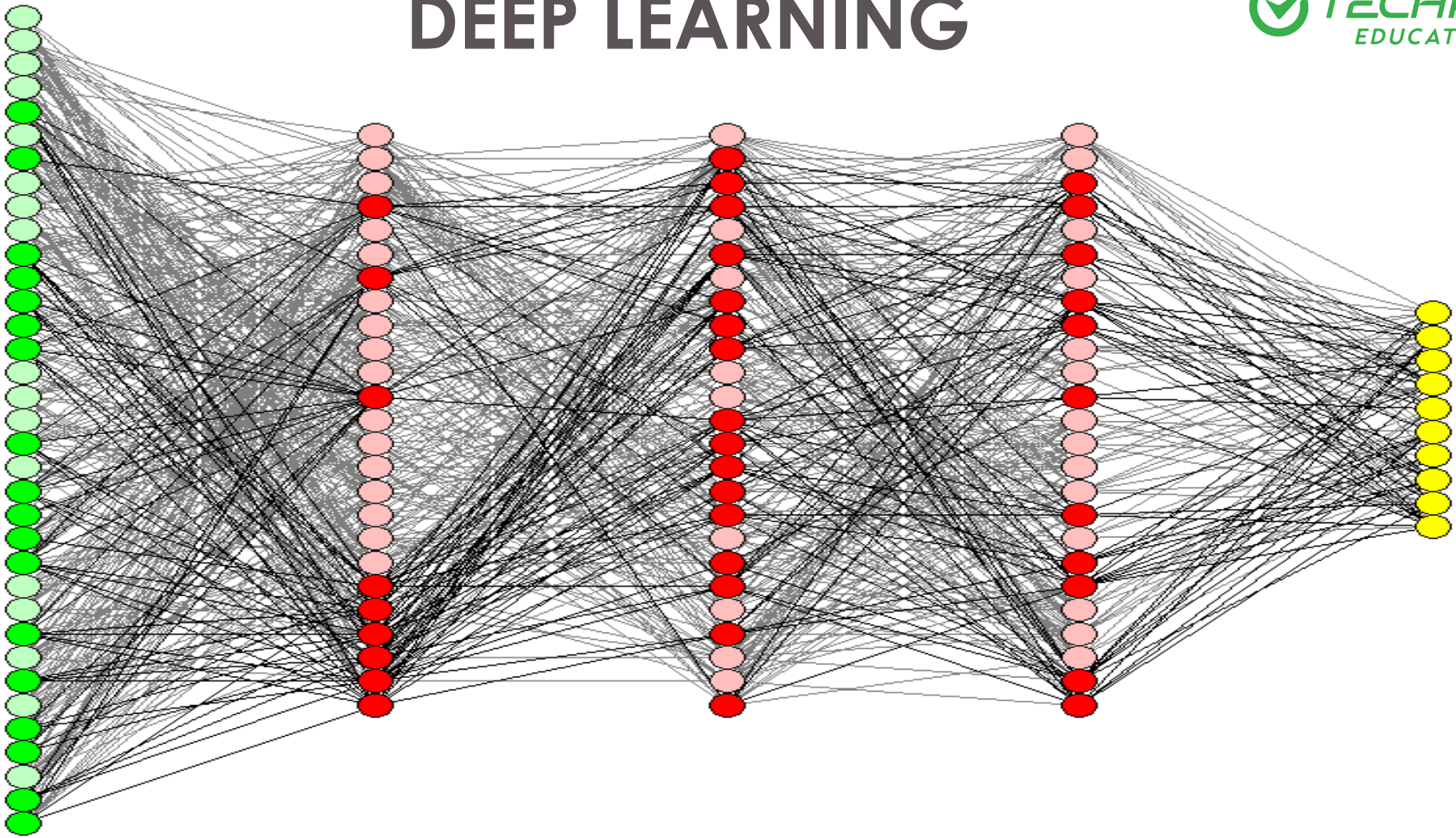
DEEP NEURAL NETWORK



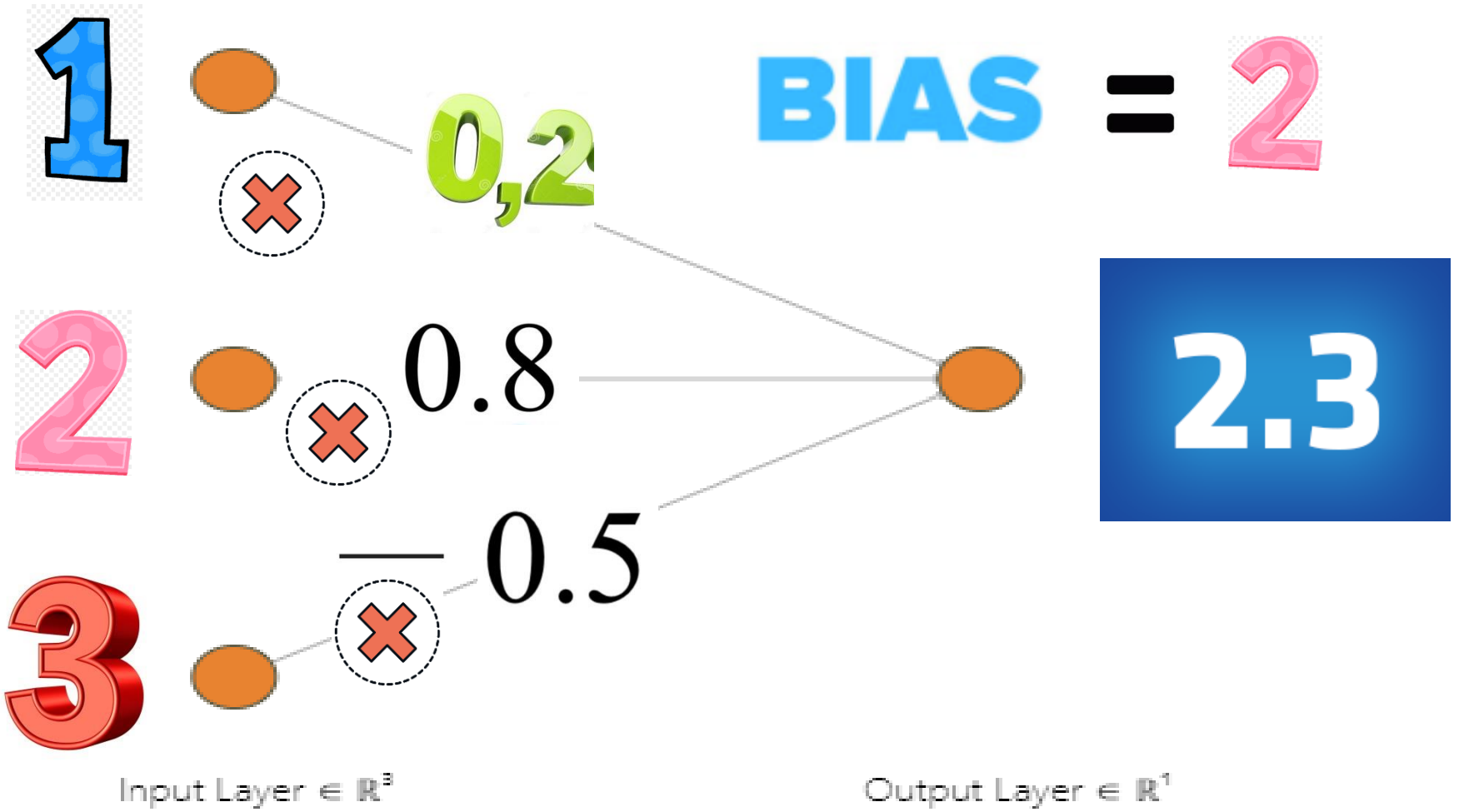
BLACK BOX



DEEP LEARNING




0,2
1.6
1.5
0.3


1
2
3
0,2
0.8
0.5
BIAS = 2
2.3
Input Layer ∈ ℝ³
Output Layer ∈ ℝ¹

PERCEPTRON MODELS

```
inputs = [1, 2, 3]
weights= [0.2, 0.8, -0.5]
bias = 2
```

```
output= inputs[0]*weights[0]+inputs[1]*weights[1]+inputs[2]*weights[2]+bias
```

```
print(output)
```

```
2.3
```

PERCEPTRON MODELS

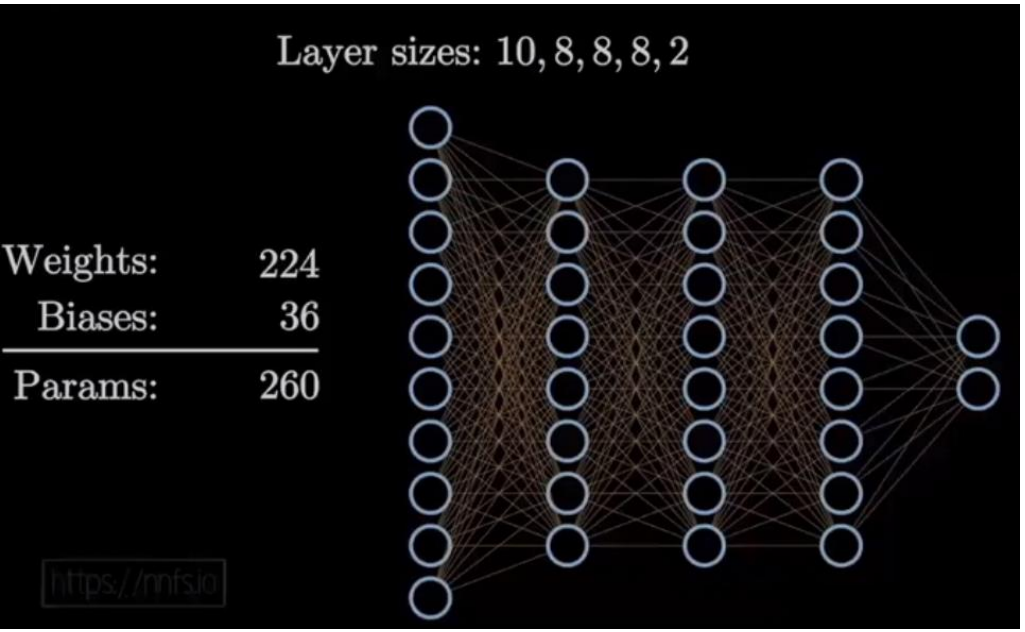
```
inputs = [1.2, 5.1, 2.1]
weights= [3.1, 2.1, 8.7]
bias = 3
```

```
output= inputs[0]*weights[0]+inputs[1]*weights[1]+inputs[2]*weights[2]+bias
```

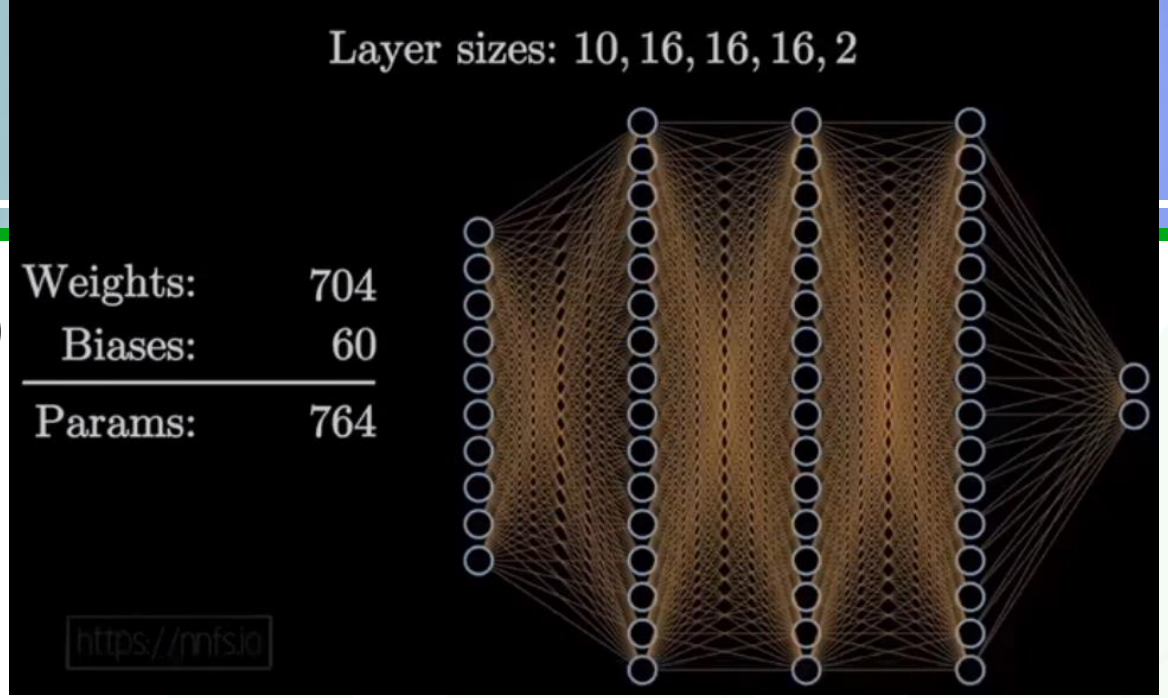
```
print(output)
```

```
35.7
```

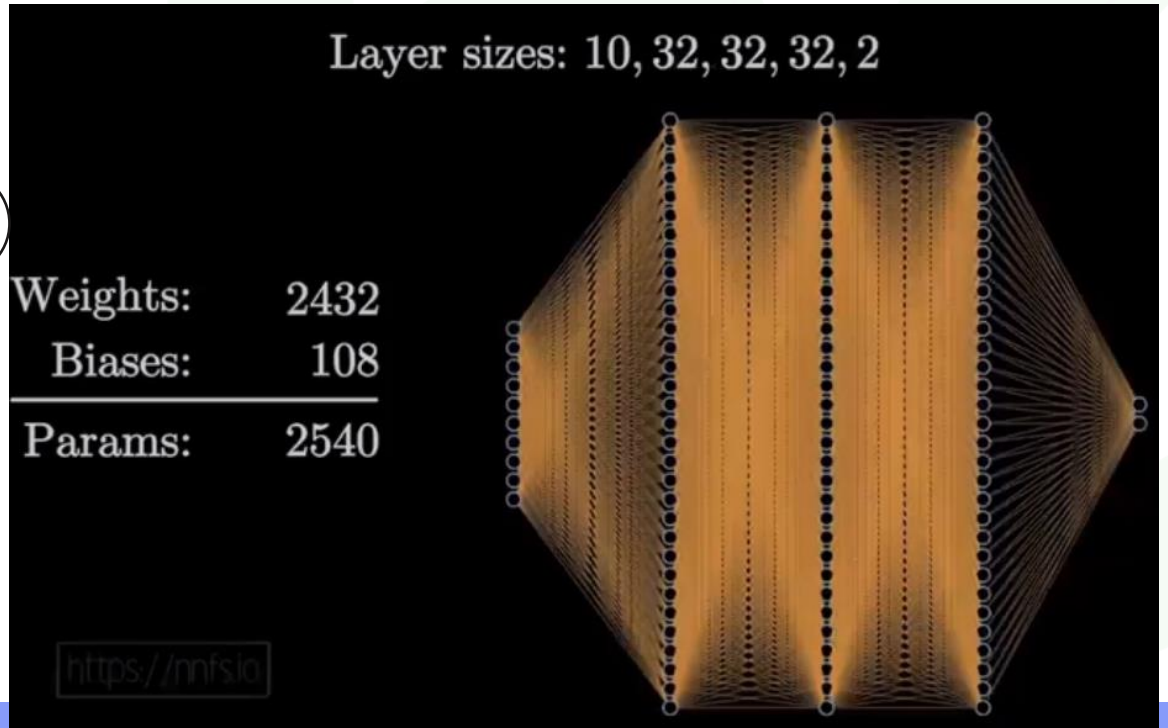
1

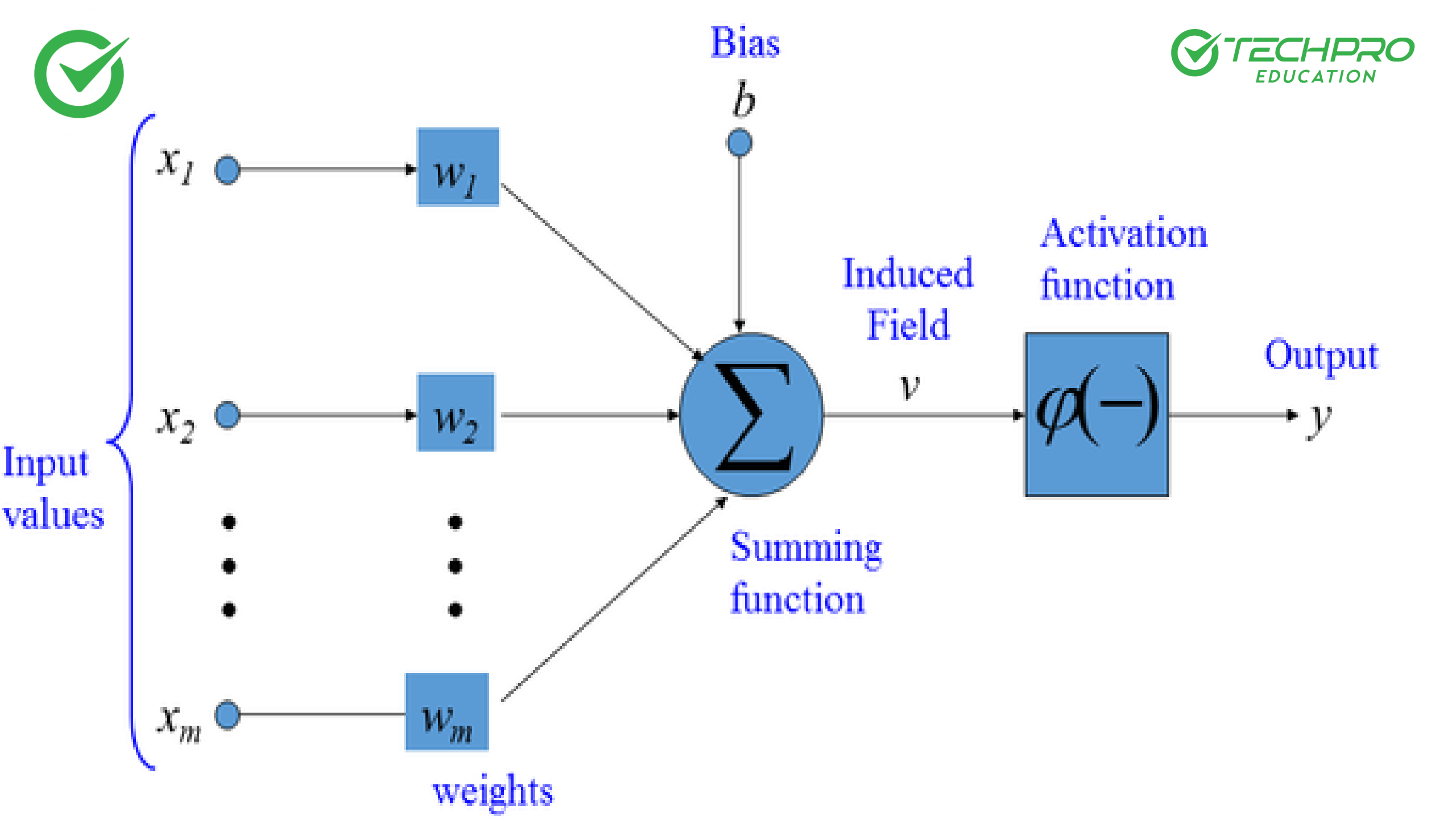

Layer sizes: 10, 8, 8, 8, 2
Weights: 224
Biases: 36
Params: 260
https://nnfs.io

2


Layer sizes: 10, 16, 16, 16, 2
Weights: 704
Biases: 60
Params: 764
https://nnfs.io

3


Layer sizes: 10, 32, 32, 32, 2
Weights: 2432
Biases: 108
Params: 2540
https://nnfs.io


Bias
b
x_1
w_1
x_2
w_2
x_m
w_m
Input values
weights
Induced Field
v
Activation function
$\varphi(-)$
Output
y
Summing function

Weights & Biases

```
model.get_weights()
```

```
[array([[1.1685071, 2.410842 , 1.4807949, 1.5210158],
        [3.331014 , 3.7875195, 3.6944022, 2.8389838]], dtype=float32),
 array([0.36392605, 1.0143974 , 0.5387995 , 0.83042043], dtype=float32),
 array([[ 1.0237961 ,  1.9563375 , -0.32918864,  2.0940597 ],
        [ 0.57272196,  2.8120105 , -0.35754463,  2.5533383 ],
        [ 1.2114699 ,  1.7479924 ,  0.42147276,  2.3708396 ],
        [ 0.9405575 ,  2.825859  ,  0.4124322 ,  1.6864424 ]],
       dtype=float32),
 array([-0.48698762,  1.0052861 , -0.39161626,  0.99313134], dtype=float32),
 array([[ 0.90866494, -0.40719548, -0.54099476, -0.24452515],
        [ 2.275784  ,  0.163791  , -0.4405466 , -0.22369042],
        [-0.38126084, -0.6168557 , -0.13984388,  0.7046616 ],
        [ 2.335869  , -0.75989604, -0.11639786, -0.2
       dtype=float32),
 array([ 0.9449791 ,  0.        ,  0.        , -0.06
 array([[ 3.194789  ],
        [-0.5085477 ],
        [ 0.05616736],
        [-0.56941813]], dtype=float32),
 array([0.96774864], dtype=float32)]
```

```
model.weights #eğitim sonrası ağırlık değerleri
```

```
[<tf.Variable 'dense_28/kernel:0' shape=(2, 4) dtype=float32, numpy=
 array([[ 1.1207535 ,  2.6153436 ,  2.2236052 , -0.41278544],
        [ 4.2392306 ,  3.935392  ,  3.923037  ,  0.14543056]],
       dtype=float32)>,
 <tf.Variable 'dense_28/bias:0' shape=(4,) dtype=float32, numpy=array([ 0.84559083,  0.8556627 ,  0.8424017 , -0.07687276], dty
pe=float32)>,
 <tf.Variable 'dense_29/kernel:0' shape=(4, 4) dtype=float32, numpy=
 array([[ 3.2622933 ,  2.0089328 ,  0.05100155,  0.16982514],
        [ 2.545713  ,  2.846746  , -0.8356694 , -0.8377631 ],
        [ 3.034534  ,  2.1935809 , -0.28192574, -0.4393447 ],
        [ 0.0160602 , -0.29168776, -0.1409725 , -0.8440891 ]],
       dtype=float32)>,
 <tf.Variable 'dense_29/bias:0' shape=(4,) dtype=float32, numpy=array([0.83379275, 0.83337206, 0.        , 0.        ], dtype=f
loat32)>,
 <tf.Variable 'dense_30/kernel:0' shape=(4, 4) dtype=float32, numpy=
 array([[ 0.13299656, -0.64306873,  3.2317638 , -0.20053375],
        [-0.26852682, -0.71713513,  2.7907538 , -0.12816662],
        [ 0.7763695 ,  0.16881973,  0.5821449 ,  0.07822013],
        [-0.2722649 , -0.03520322, -0.07845533, -0.7386498 ]],
       dtype=float32)>,
```

ACTIVATION FUNCTIONS

DANCE MOVES OF ACTIVATION FUNCTIONS

Sigmoid

$$y = \frac{1}{1+e^{-x}}$$

Tanh

$$y = \tanh(x)$$

Step Function

$$y = \begin{cases} 0, & x<n \\ 1, & x \geq n \end{cases}$$

Softplus

$$y = \ln(1+e^{x})$$

source: sefiks

ReLU

$$y = \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases}$$

Softsign

$$y = \frac{x}{(1+|x|)}$$

ELU

$$y = \begin{cases} \alpha(e^{x}-1), & x<0 \\ x, & x \geq 0 \end{cases}$$

Log of Sigmoid

$$y = \ln\left(\frac{1}{1+e^{-x}}\right)$$

Swish

$$y = \frac{x}{1+e^{-x}}$$

Sinc

$$y = \frac{\sin(x)}{x}$$

Leaky ReLU

$$y = \max(0.1x, x)$$

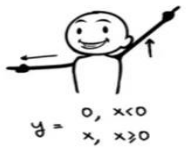
Mish

$$y = x(\tanh(\text{softplus}(x)))$$

Sigmoid

$y = \frac{1}{1+e^{-x}}$

ReLU



$y = \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases}$

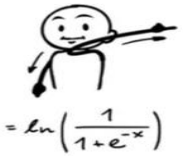
Swish

$y = \frac{x}{1+e^{-x}}$

Softplus

$y = \ln(1+e^{x})$

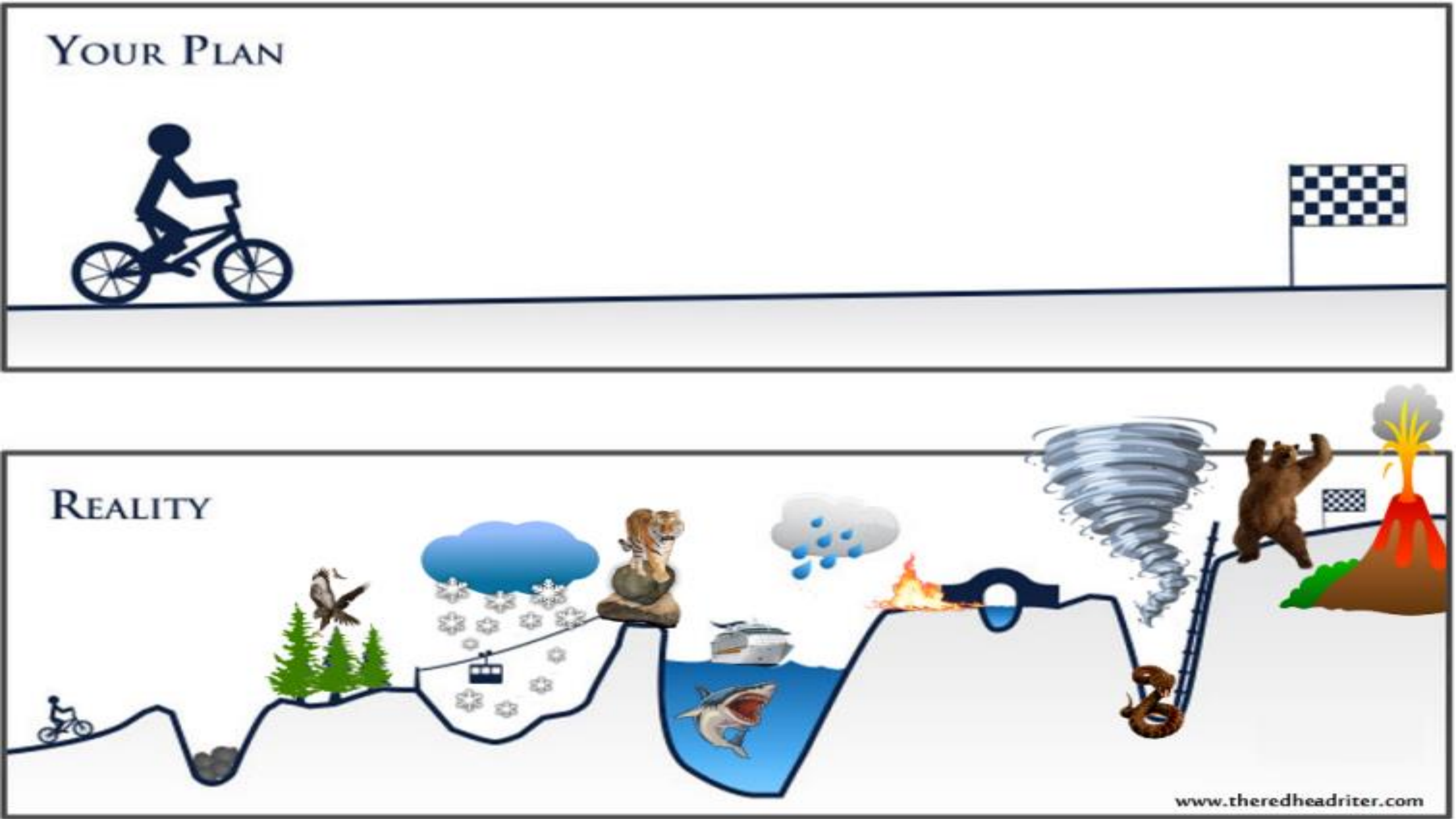
Log of Sigmoid



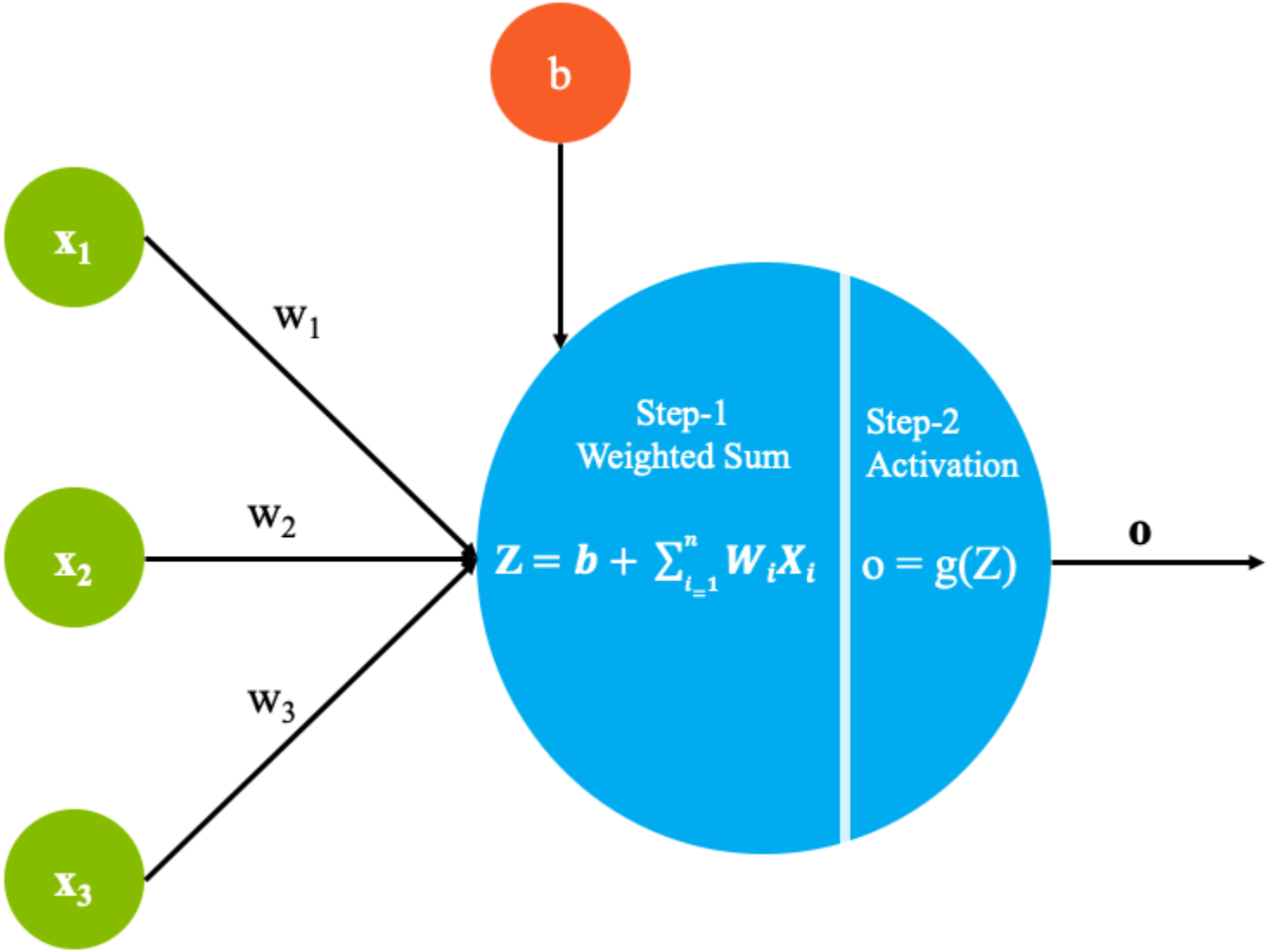
$y = \ln\left(\frac{1}{1+e^{-x}}\right)$

Mish

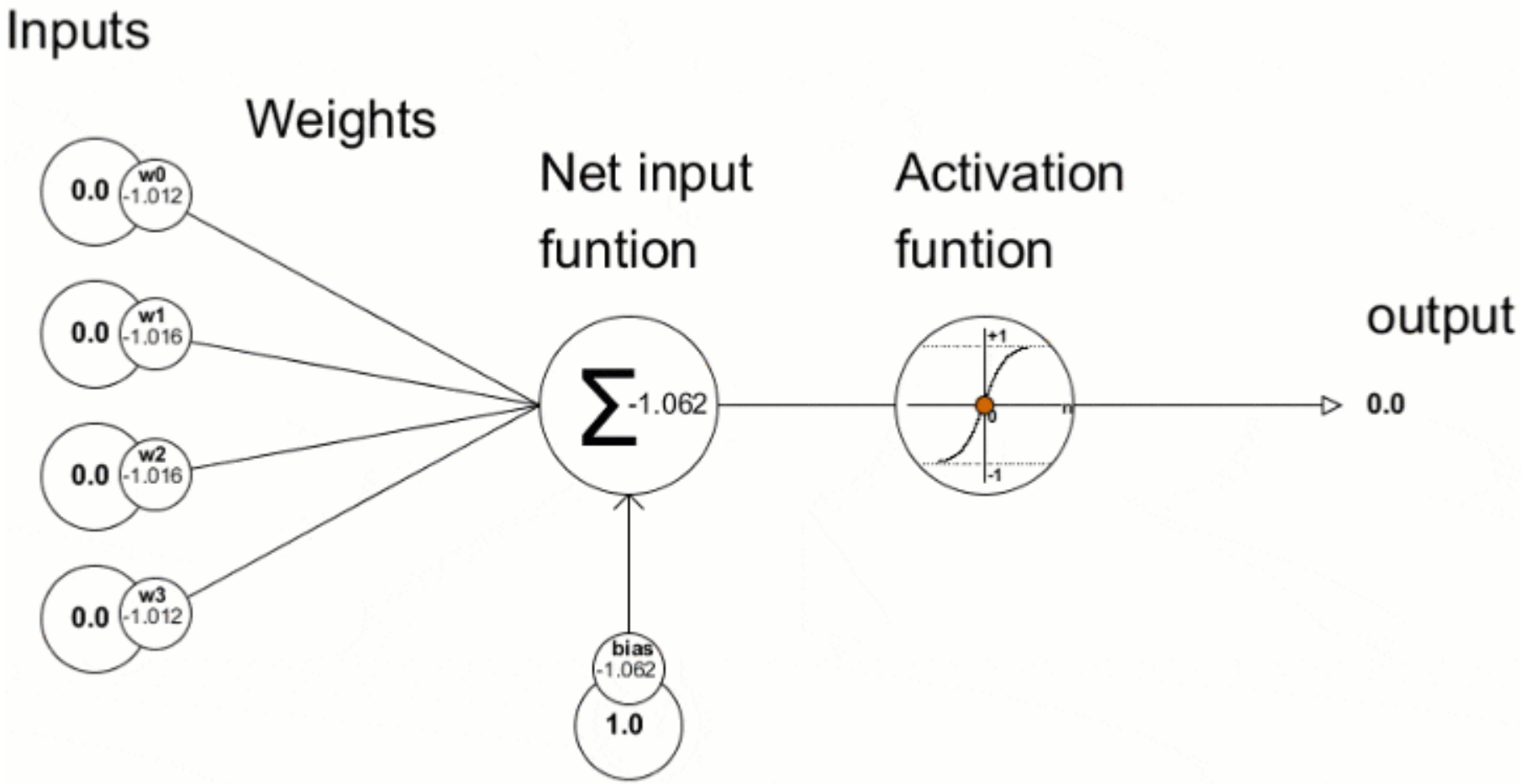
$y = x(\tanh(\text{softplus}(x)))$


Your Plan
Reality
www.theredheadriter.com

ACTIVATION FUNCTIONS



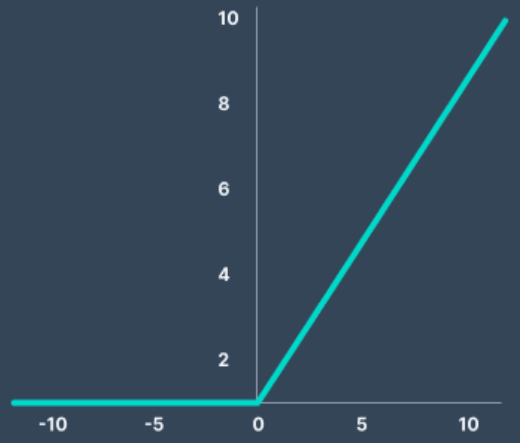
ACTIVATION FUNCTIONS



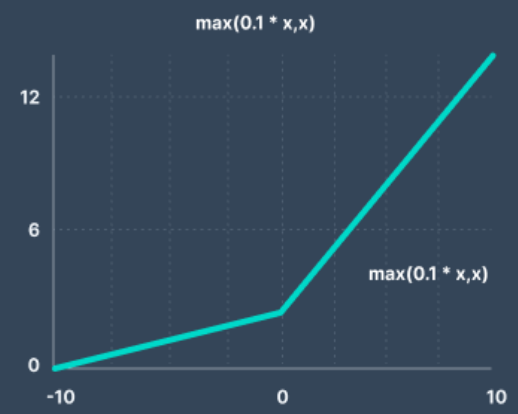
ACTIVATION FUNCTIONS

TECHPRO EDUCATION

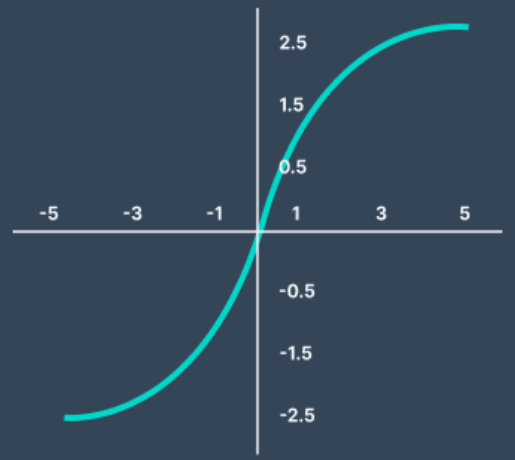
ReLU



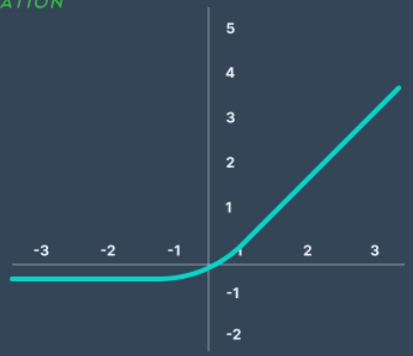
Leaky ReLU



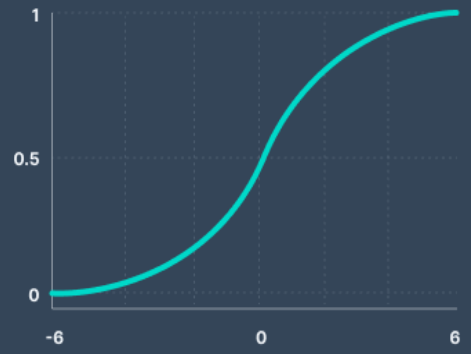
Tanh



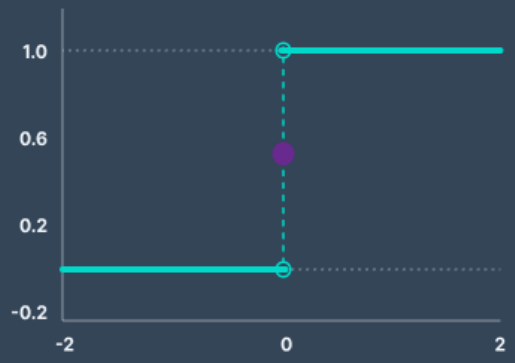
ELU



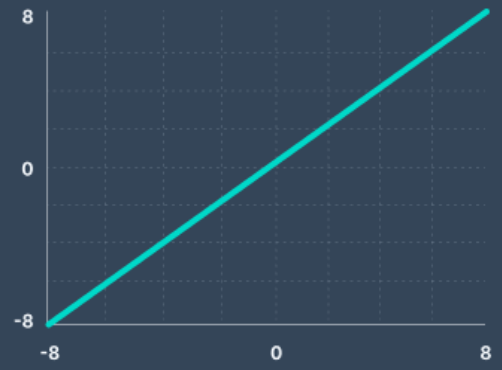
Sigmoid / Logistic



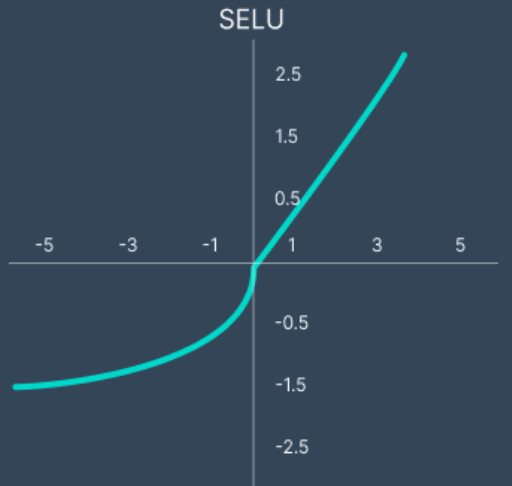
Binary Step Function



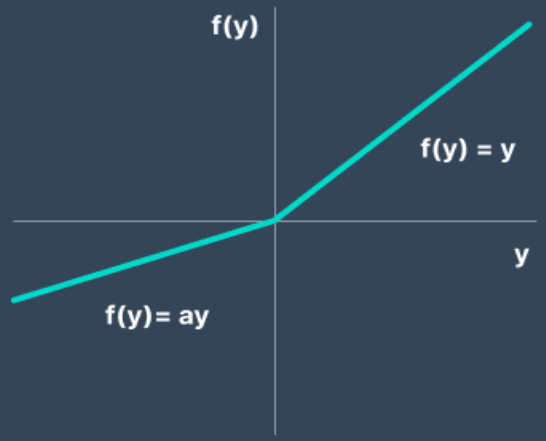
Linear



SELU



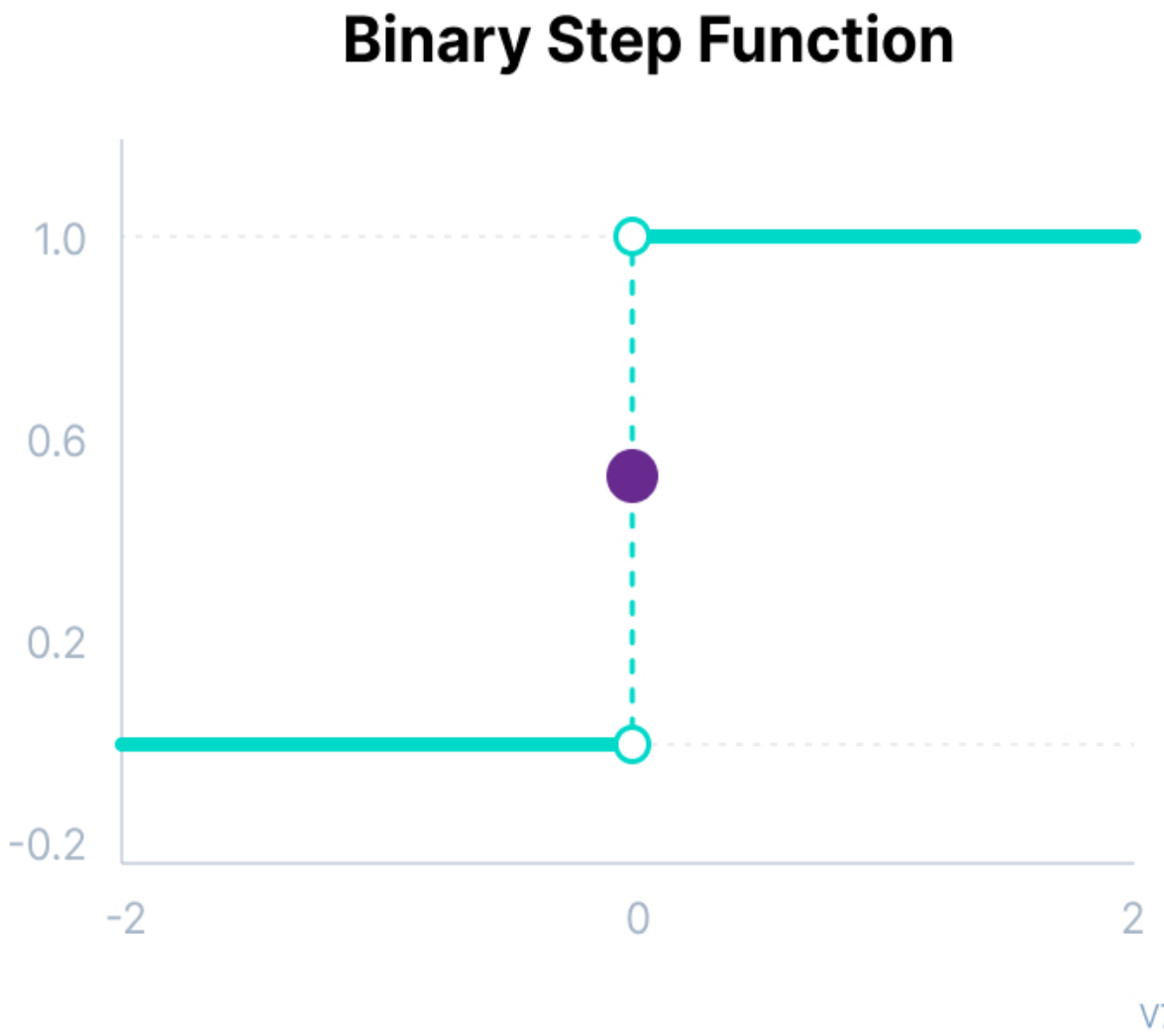
Parametric ReLU



BINARY STEP ACTIVATION FUNCTION

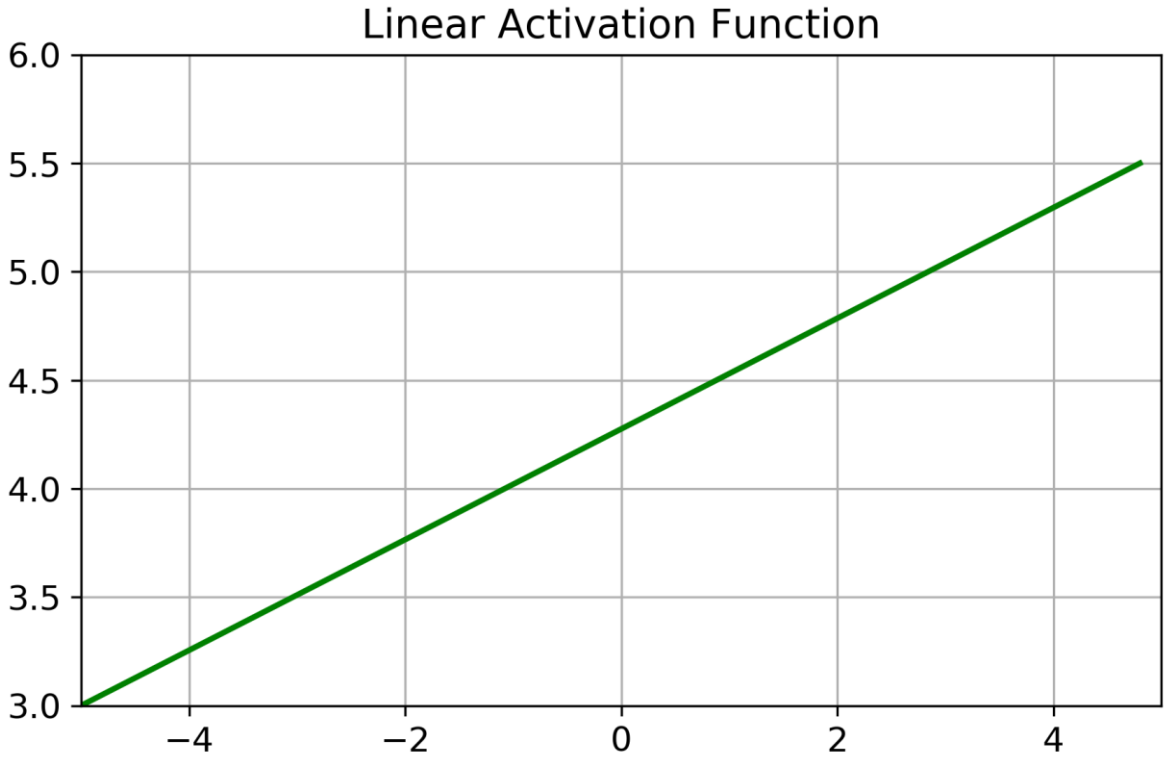
Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.



LINEAR ACTIVATION FUNCTION

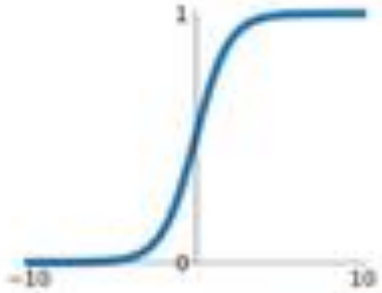
It is a simple straight line activation function where our function is directly proportional to the weighted sum of neurons or input.



NON-LINEAR ACTIVATION FUNCTIONS

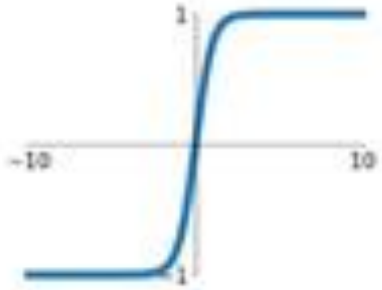
Sigmoid

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



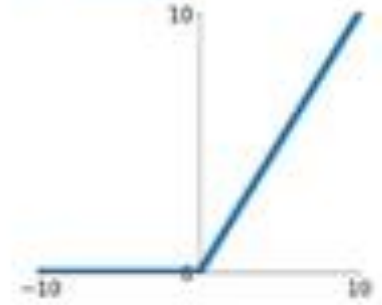
tanh

$$\tanh(x)$$



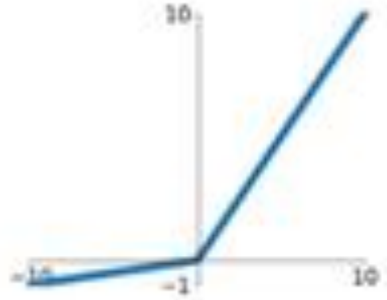
ReLU

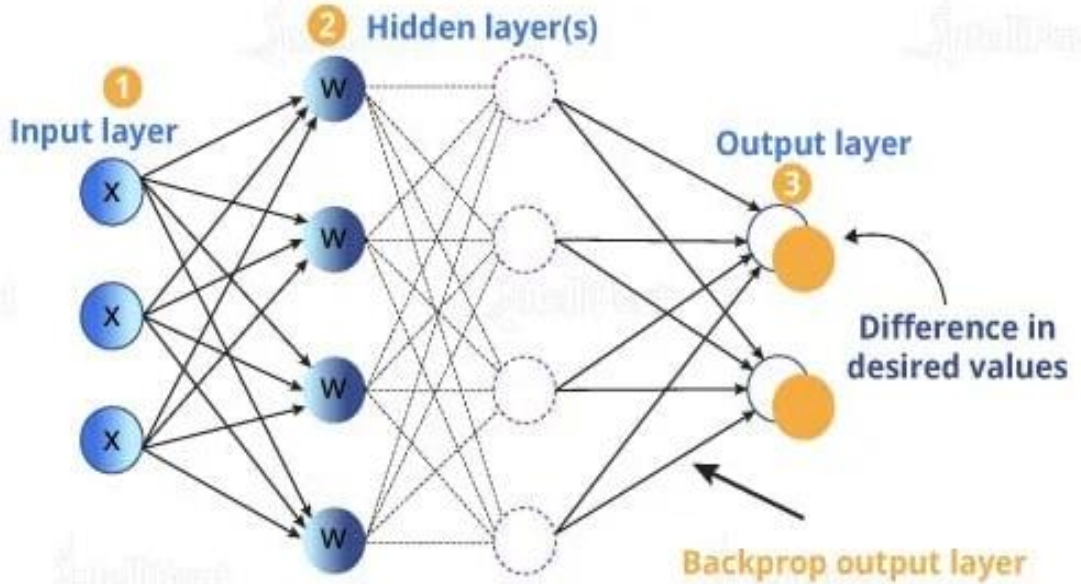
$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



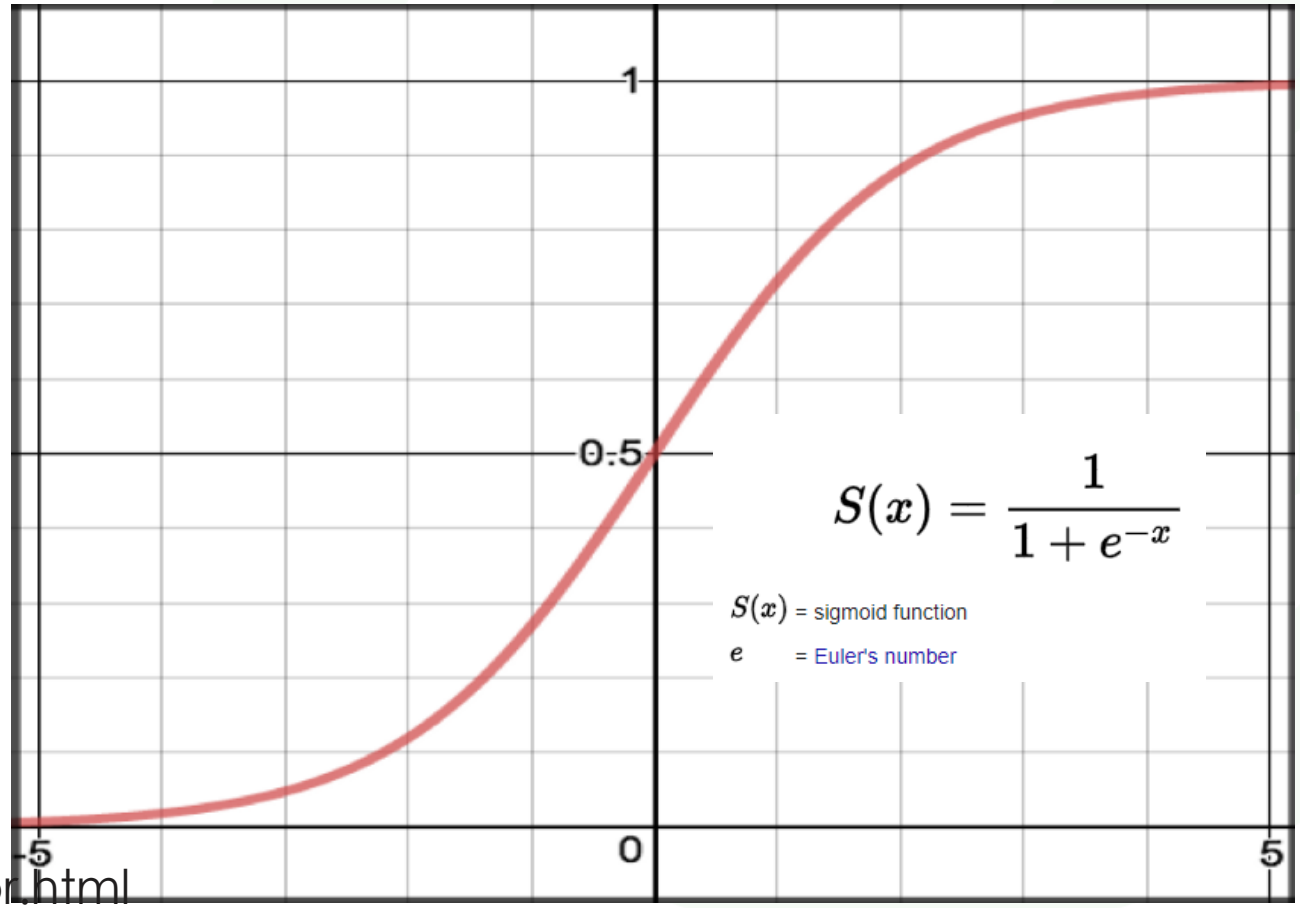


SIGMOID ACTIVATION FUNCTION

Sigmoid Calculator
sigmoid = 1 / 1+e^-x
x: 1.51
Sigmoid
S(X) 0.8190612068479367
S'(X) 0.14819994628473815 *sigmoid prime(X)*

ReLU Calculator
ReLU = MAX(X,0)
x: 0,458
ReLU
S(X)
S'(X) *ReLU prime(X)*

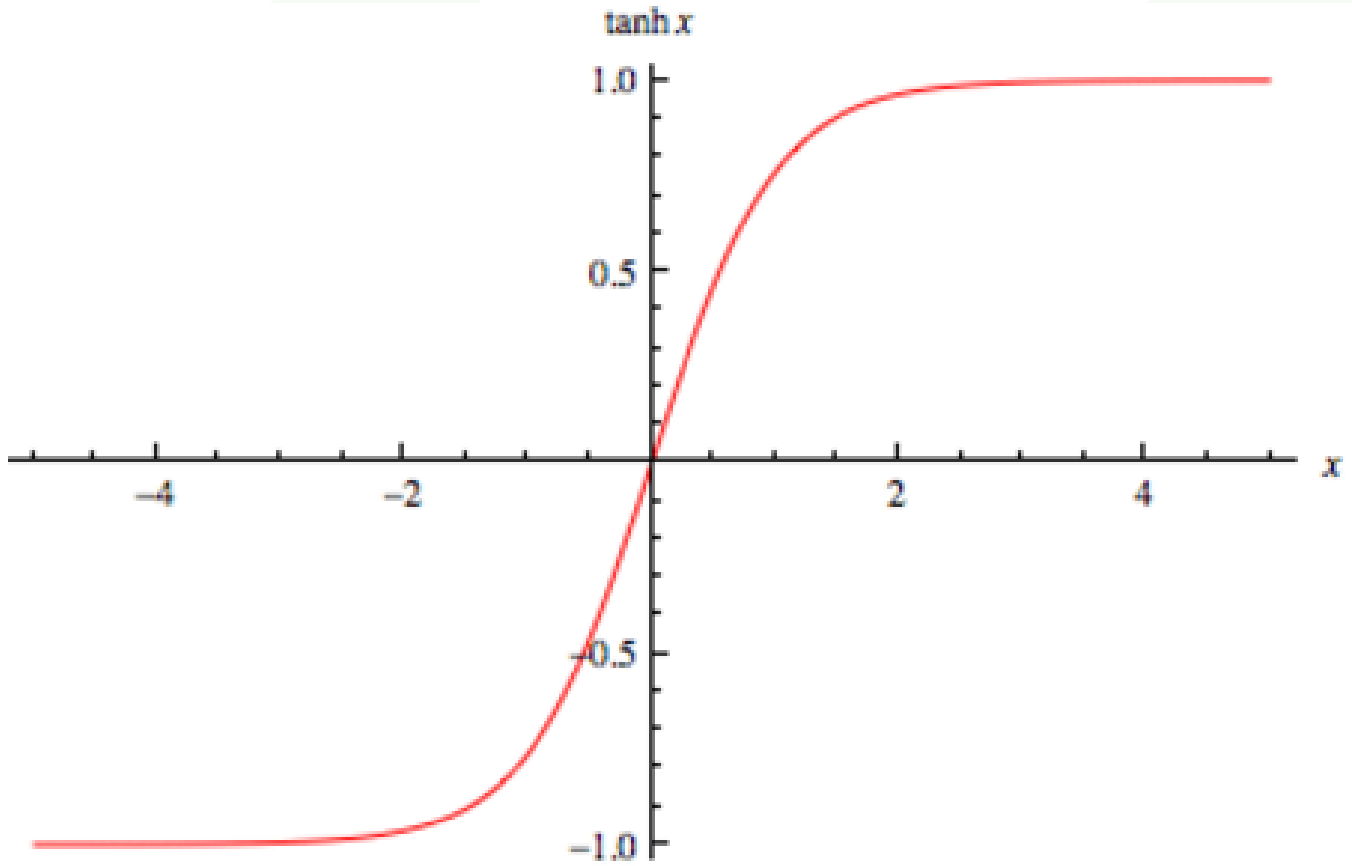
Sigmoid takes a real value as the input and outputs another value between 0 and 1. The sigmoid activation function translates the input ranged in (-∞,∞) to the range in **(0,1).**



http://www.tinkershop.net/ml/sigmoid_calculator.html

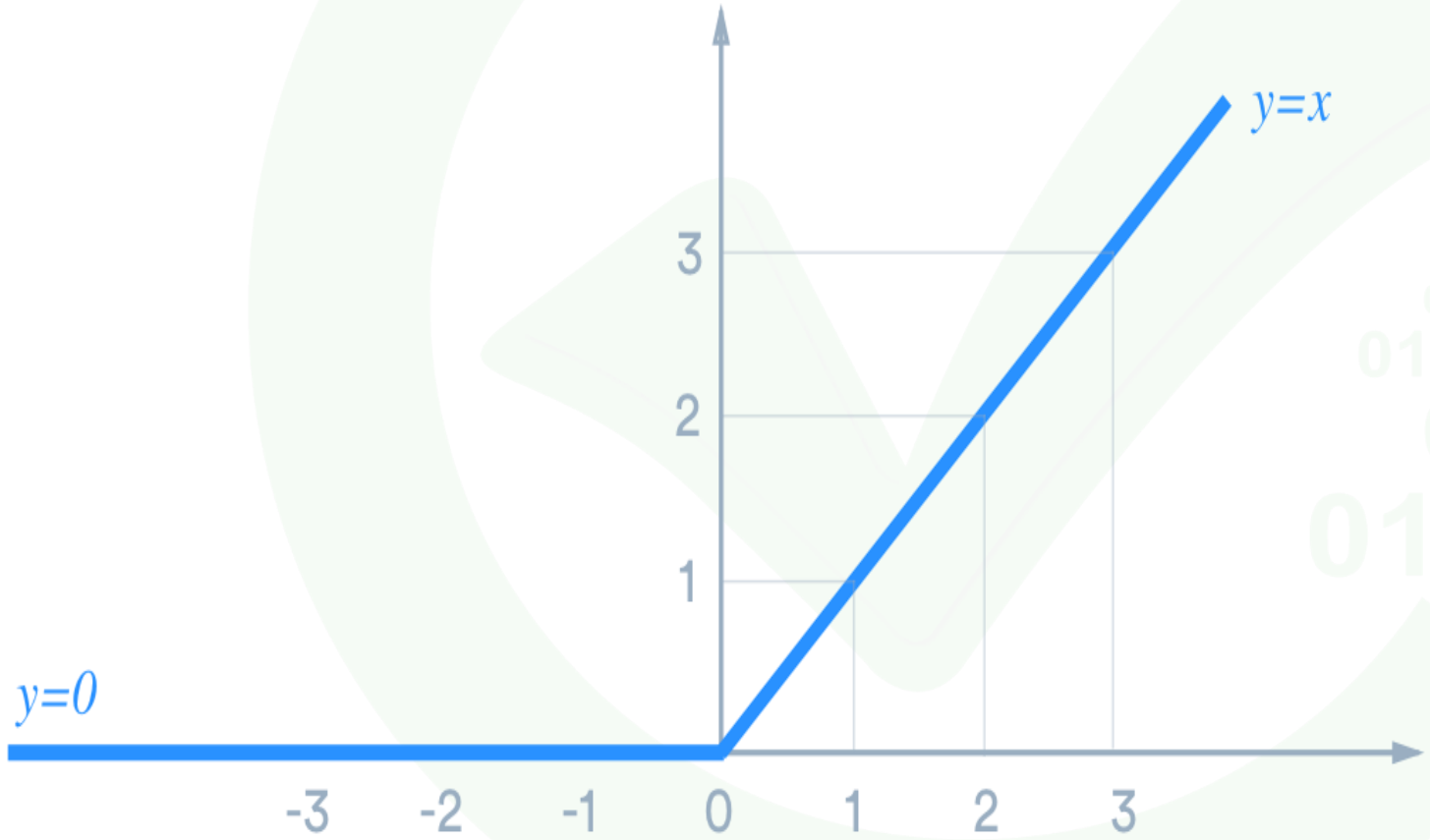
TanH ACTIVATION FUNCTION

It is very similar to the sigmoid activation function and even has the same S-shape. The function takes any real value as input and outputs values in the range **-1 to 1.**



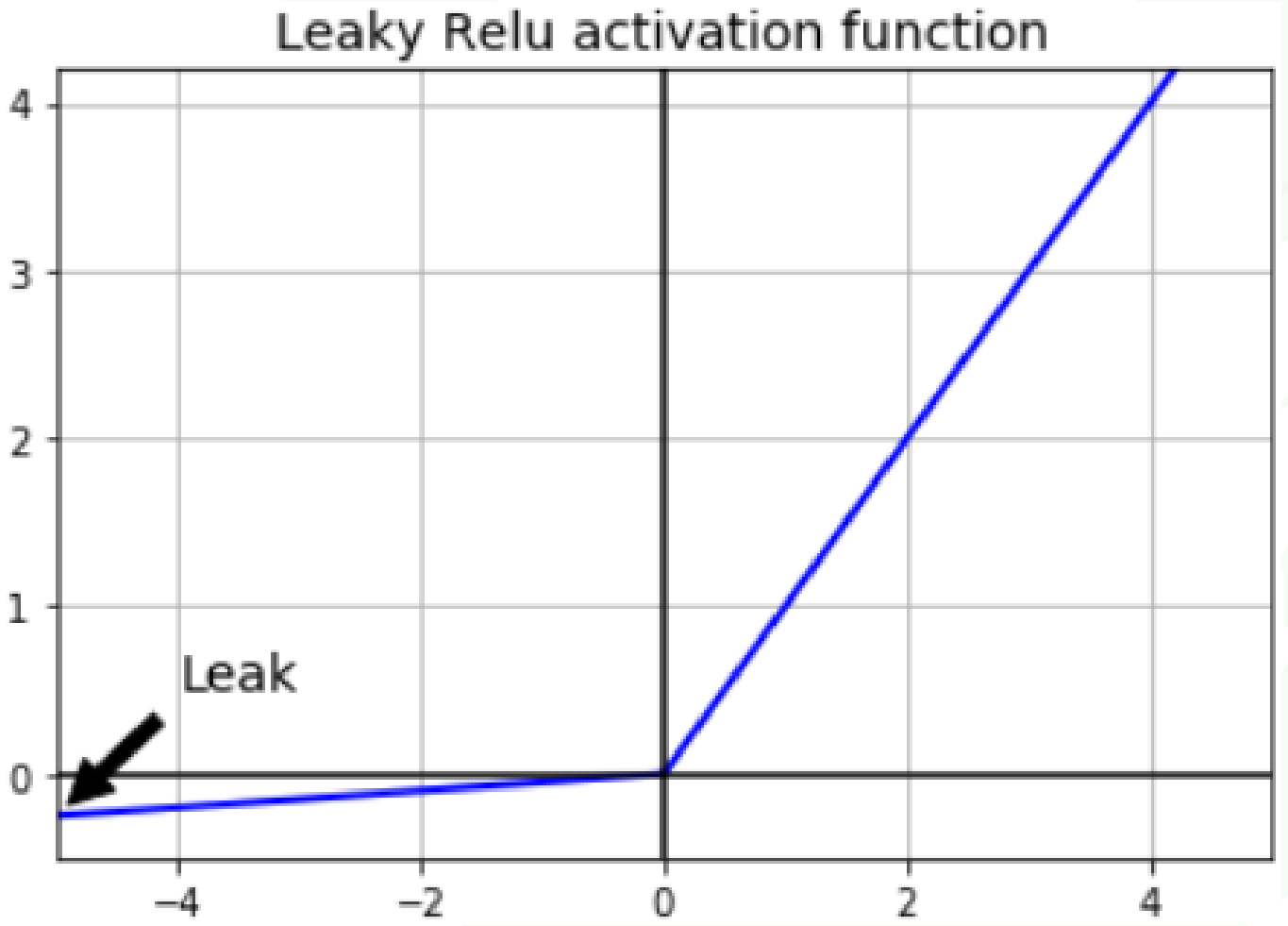
ReLU ACTIVATION FUNCTION

The ReLU (Rectified Linear Unit) is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.

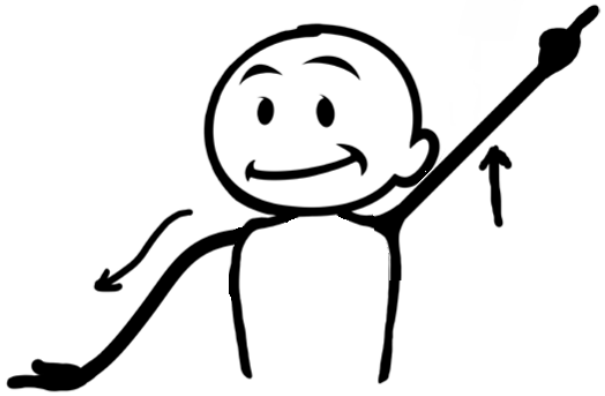


Leaky ReLU ACTIVATION FUNCTION

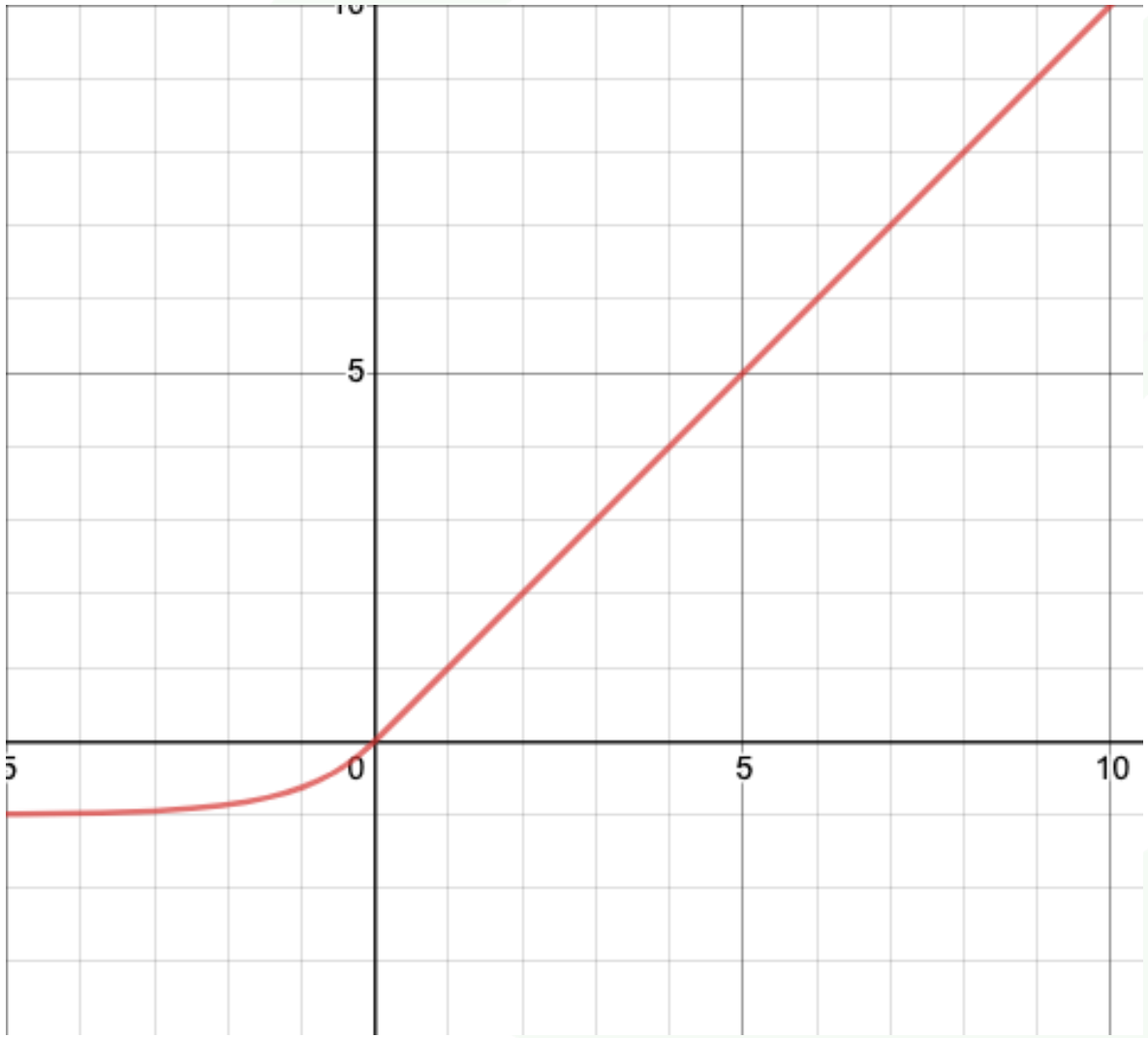
The Leaky ReLU (LReLU or LReL) modifies the function to allow small negative values when the input is less than zero.



ELU ACTIVATION FUNCTION

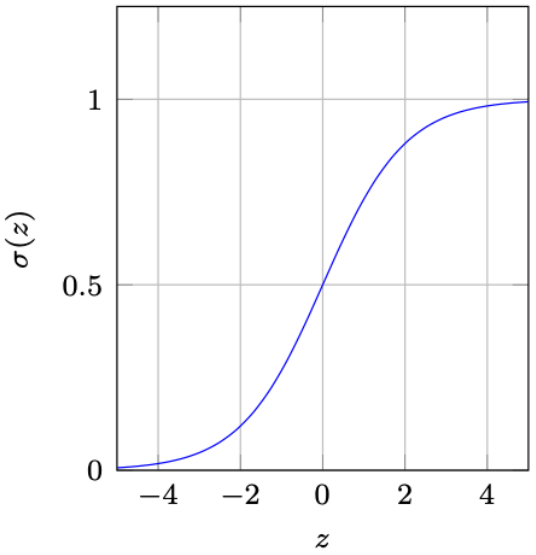


$$y = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

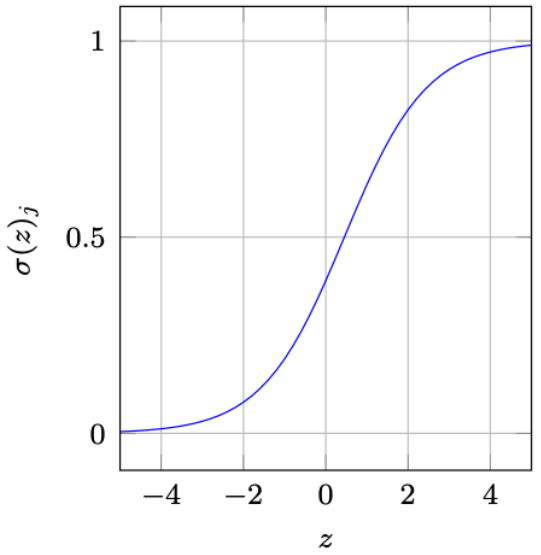


SOFTMAX ACTIVATION FUNCTION

Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.

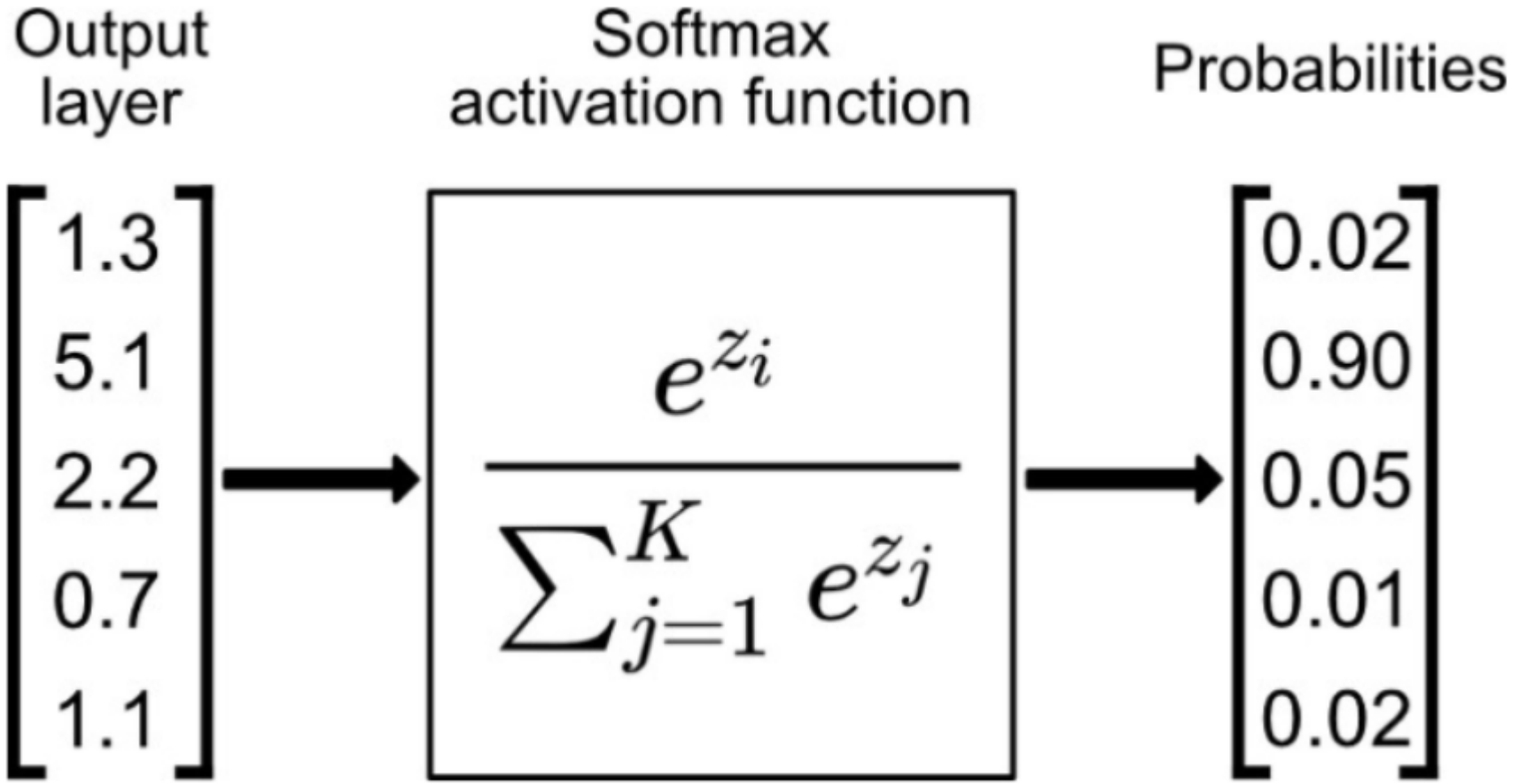


(a) Sigmoid activation function.



(b) Softmax activation function.

SOFTMAX ACTIVATION FUNCTION



ACTIVATION FUNCTIONS

TECHPRO EDUCATION

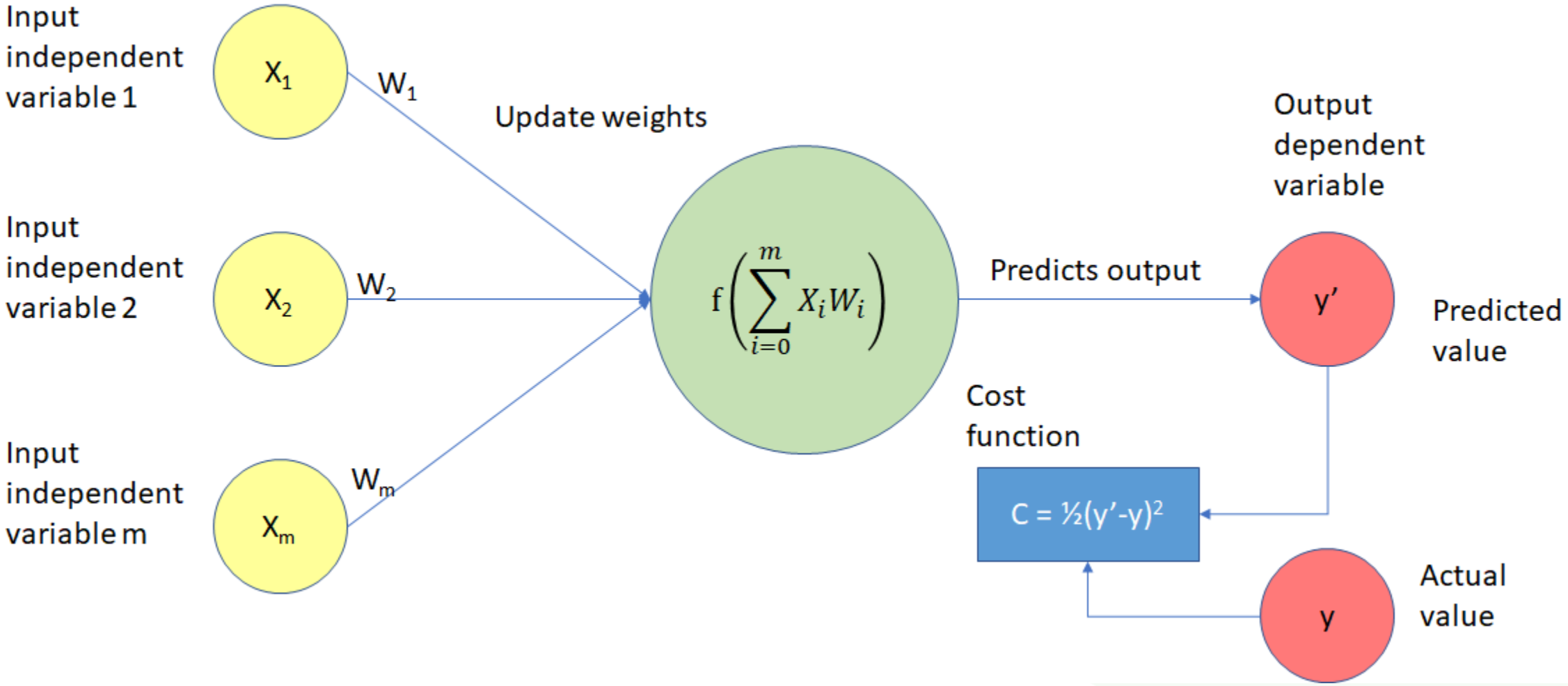
AKTİVASYON FONKSİYON	DENKLEM	ARALIK
Doğrusal Fonksiyon	$f(x) = x$	$(-\infty, \infty)$
Basamak Fonksiyonu	$f(x) = \begin{cases} 0 \text{ için } x < 0 \\ 1 \text{ için } x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Fonksiyon	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hiperbolik Tanjant Fonksiyonu	$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \text{ için } x < 0 \\ x \text{ için } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky (Sızıntı) ReLU	$f(x) = \begin{cases} 0.01 \text{ için } x < 0 \\ x \text{ için } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Fonksiyonu	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \text{ için } f(x) = x \\ \beta \to \infty \text{ için } f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

Aktivasyon Fonksiyonu	En Çok Kullanıldığı Alan	Avantajları	Dezavantajları	Aralığı
Sigmoid	Lojistik Regresyon, Binary Classification, GANs	0-1 aralığında olması nedeniyle olasılık hesaplamalarında faydalıdır.	Gradient vanishing problemi yaşanabilir.	(0,1)
Tanh	RNNs, Autoencoders, Feedforward Neural Networks	Sıfıra göre simetrik olması nedeniyle sigmoid fonksiyonundan daha iyi sonuçlar verebilir.	Gradient vanishing problemi yaşanabilir.	(-1,1)
ReLU	CNNs, Feedforward Neural Networks	Hızlı hesaplamalar ve daha iyi sonuçlar vermesi nedeniyle diğer fonksiyonlara göre daha yaygın kullanılır.	Gradient exploding problemi yaşanabilir.	[0,inf)

Aktivasyon Fonksiyonu	En Çok Kullanıldığı Alan	Avantajları	Dezavantajları	Aralığı
Leaky ReLU	CNNs, Feedforward Neural Networks	Dying ReLU problemini çözebilir.	Eğimi belirleyen parametrenin optimizasyonu zor olabilir.	(-inf,inf)
Softmax	Multi-Class Classification	Çoklu sınıflandırma problemleri için idealdir.	Sınıf sayısı arttıkça hesaplama maliyeti artabilir.	[0,1] ve toplamı 1

COST FUNCTION

COST FUNCTION

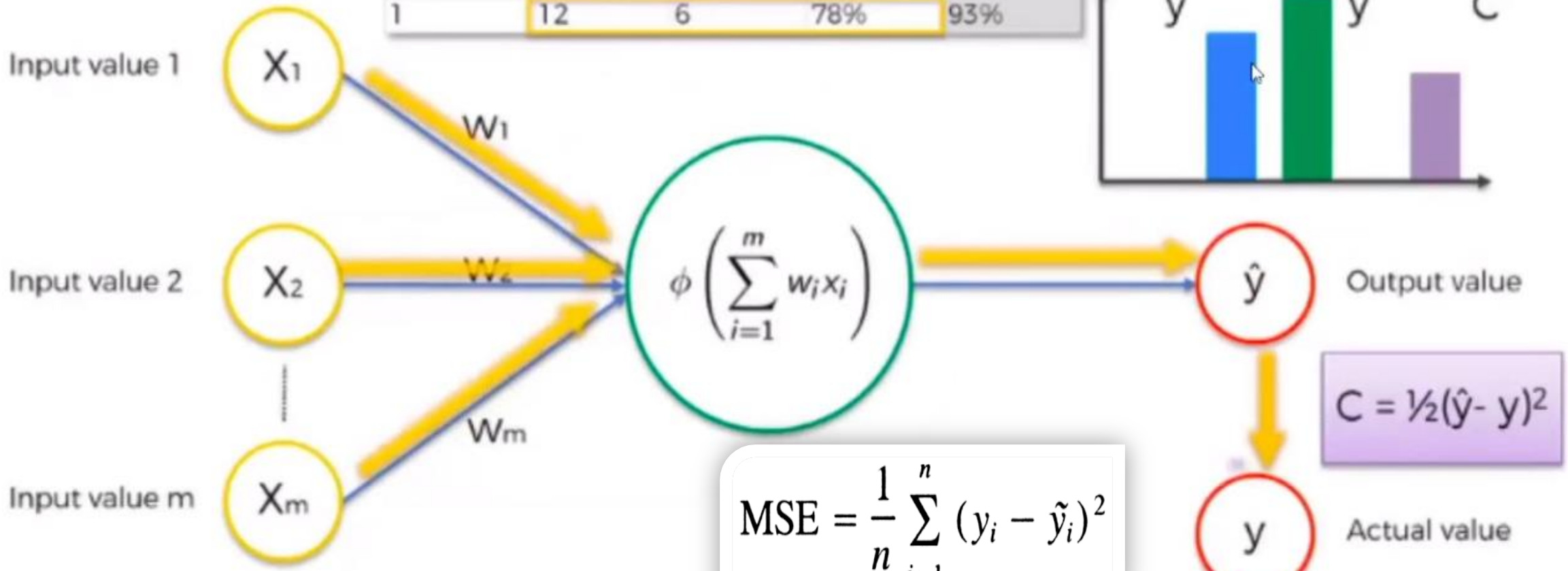


COST FUNCTION

TECHPRO EDUCATION

EXAMPLE

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%



$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

COST FUNCTION

TECHPRO EDUCATION

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Burada:

- N = Toplam veri noktası sayısı
- y_i = Gerçek çıktı değeri (örneğin, regresyon problemindeki hedef değer)
- $\hat{y}_i$ = Tahmin edilen çıktı değeri

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right)$$

Burada:

- N = Toplam veri noktası sayısı
- y_i = Gerçek sınıf etiketi (0 veya 1)
- $\hat{y}_i$ = Tahmini sınıf olasılığı

$$CE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

Burada:

- N = Toplam veri noktası sayısı
- C = Toplam sınıf sayısı
- y_{ij} = Gerçek sınıf etiketi (1, eğer örnek i sınıf j ise; aksi halde 0)
- $\hat{y}_{ij}$ = Tahmini sınıf olasılığı

SUMMARY

Epoch
000,321

Learning rate
0.03

Activation
Tanh

Regularization
None

Regularization rate
0

Problem type
Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$

2 HIDDEN LAYERS

4 neurons

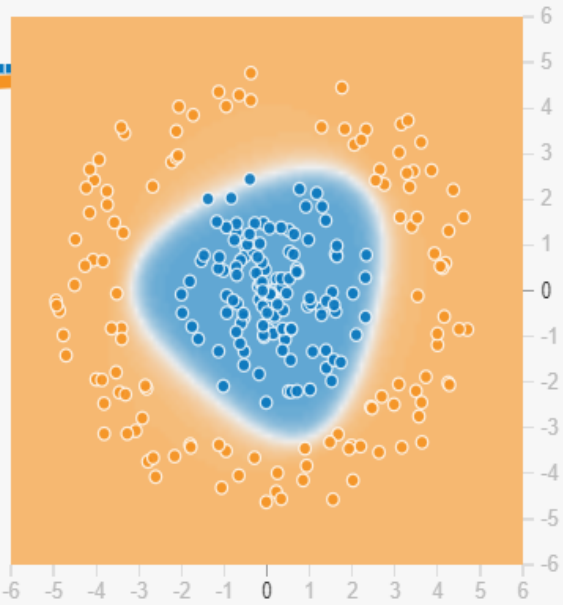
2 neurons

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

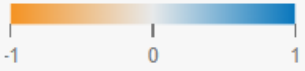
*This is the output from one **neuron**. Hover to see it larger.*

OUTPUT

Test loss 0.005
Training loss 0.001



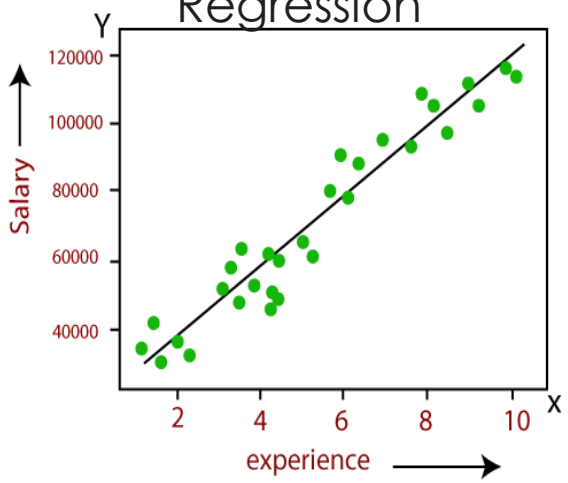
Colors shows data, neuron and weight values.



Show test data

Discretize output

Regression




```
model = Sequential()
```

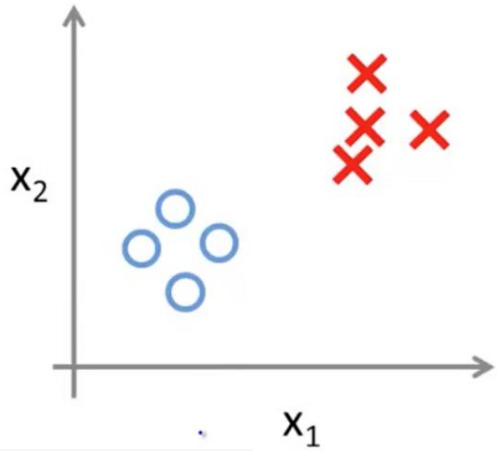
```
model.add(Dense(29, activation = 'relu'))
model.add(Dense(29, activation = 'relu'))
model.add(Dense(15, activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1))

model.compile(optimizer = 'adam', loss = 'mse')
```

Aktivasyon eklememize gerek yok
Otomatik olarak **Linear** olacaktır!

Optimizer=Adam

Loss= **Mean Squared Error (mse)**

Binary classification:




```
model = Sequential()
```

```
model.add(Dense(16, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

opt = Adam(lr=0.002)
model.compile(optimizer=opt,
              loss="binary_crossentropy",
              metrics=["Recall"])
```

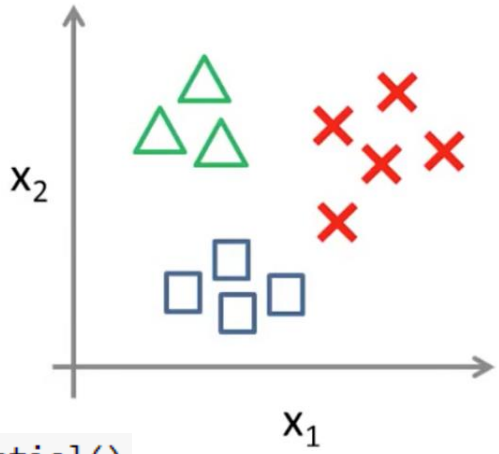
Aktivasyon **SIGMOID** olmaldır!

Loss= **binary_crossentropy**

Optimizer=Adam

Metrics= **Recall/Accuracy**

Multi-class classification:




```
model = Sequential()
```

```
model.add(Dense(units=4,activation='relu'))
model.add(Dense(units=8,activation='tanh'))
model.add(Dense(units=6,activation='tanh'))
model.add(Dense(units=3,activation='softmax'))
```

```
opt = Adam(lr = 0.003)
model.compile(optimizer = opt,
              loss = "categorical_crossentropy",
              metrics=['accuracy'])
```

Aktivasyon **SOFTMAX** olmaldır!

Loss= **categorical_crossentropy**

Optimizer=Adam

Metrics= **Recall/Accuracy**

NOTEBOOKS

Regression Loss Functions

```
model.compile(optimizer='adam',loss='mse')
```

1. Mean Squared Error Loss
2. Mean Squared Logarithmic Error Loss
3. Mean Absolute Error Loss

Multi-Class Classification Loss Functions

1. Multi-Class Cross-Entropy Loss
2. Sparse Multiclass Cross-Entropy Loss
3. Kullback Leibler Divergence Loss

```
1 model.compile(loss='categorical_crossentropy',
```

Binary Classification Loss Functions

1. Binary Cross-Entropy
2. Hinge Loss
3. Squared Hinge Loss

```
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
Epoch 1/600
426/426 [==============================] - 0s 1ms/sampl - loss: 0.6807 - val_loss: 0.6623
Epoch 2/600
426/426 [==============================] - 0s 108us/sa e - loss: 0.6498 - val_loss: 0.6330
Epoch 3/600
426/426 [==============================] - 0s 80us/samp - loss: 0.6188 - val_loss: 0.6006
.
.
.
Epoch 99/600
426/426 [==============================] - 0s 68us/sa e - loss: 0.0485 - val_loss: 0.1003
Epoch 100/600
426/426 [==============================] - 0s 73us/sa e - loss: 0.0483 - val_loss: 0.1069
Epoch 101/600
426/426 [==============================] - 0s 68us/sample - loss: 0.0500 - val_loss: 0.1036
Epoch 00101: early stopping
```

decreses as model learns

NOTEBOOKS

TECHPRO EDUCATION

In [221]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.neural_network import MLPClassifier
```

In [222]:
```
model = Sequential()
model.add(Dense(units = 24,activation='relu'))
model.add(Dense(units = 24,activation='relu'))
model.add(Dense(units = 1,activation='sigmoid'))
```

In [223]:
```
model.compile(loss='binary_crossentropy', metrics = ["accuracy"])
```

In [224]:
```
model.fit(x=X_train, y=y_train, batch_size = 19, epochs= 50)
```

```
Epoch 1/50
159/159 [==============================] - 1s 2ms/step - loss: 4.8735 - accuracy: 0.7850
Epoch 2/50
159/159 [==============================] - 0s 2ms/step - loss: 3.1734 - accuracy: 0.8036
Epoch 3/50
159/159 [==============================] - 0s 2ms/step - loss: 3.0606 - accuracy: 0.7923
Epoch 4/50
159/159 [==============================] - 0s 2ms/step - loss: 2.8676 - accuracy: 0.8019
Epoch 5/50
159/159 [==============================] - 0s 2ms/step - loss: 2.5919 - accuracy: 0.7986
Epoch 6/50
159/159 [==============================] - 0s 2ms/step - loss: 2.2373 - accuracy: 0.8062
```

NOTEBOOKS

```
model = Sequential()

model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam' loss='mse')
```

```
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dense(units=15,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
model.compile loss='binary_crossentropy' , optimizer='adam')
```

```
model.add(Dense(78,  activation='relu'))
model.add(Dropout(0.2))

# hidden Layer
model.add(Dense(39, activation='relu'))
model.add(Dropout(0.2))

# hidden Layer
model.add(Dense(19, activation='relu'))
model.add(Dropout(0.2))

# output Layer
model.add(Dense(units=1,activation='sigmoid'))

# Compile model
model.compile loss='binary_crossentropy' , optim
```