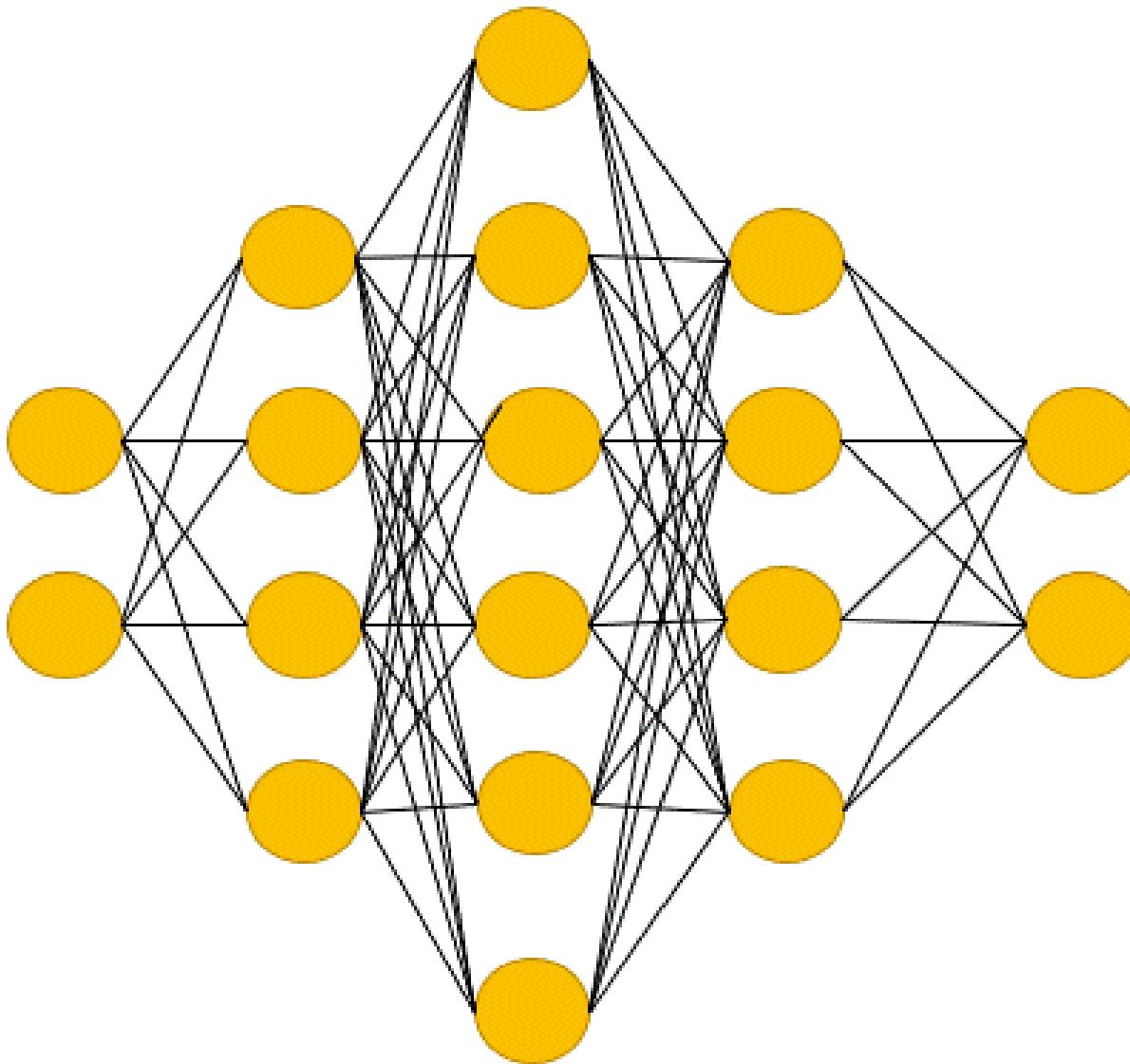






Let me ask,
Gru

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



NN-SVG



Publication-ready NN-architecture schematics.

[Download SVG](#)

[FCNN style](#) [LeNet style](#) [AlexNet style](#)

Style:

Edge width proportional to edge weights

Edge Width

Edge opacity proportional to edge weights

Edge Opacity

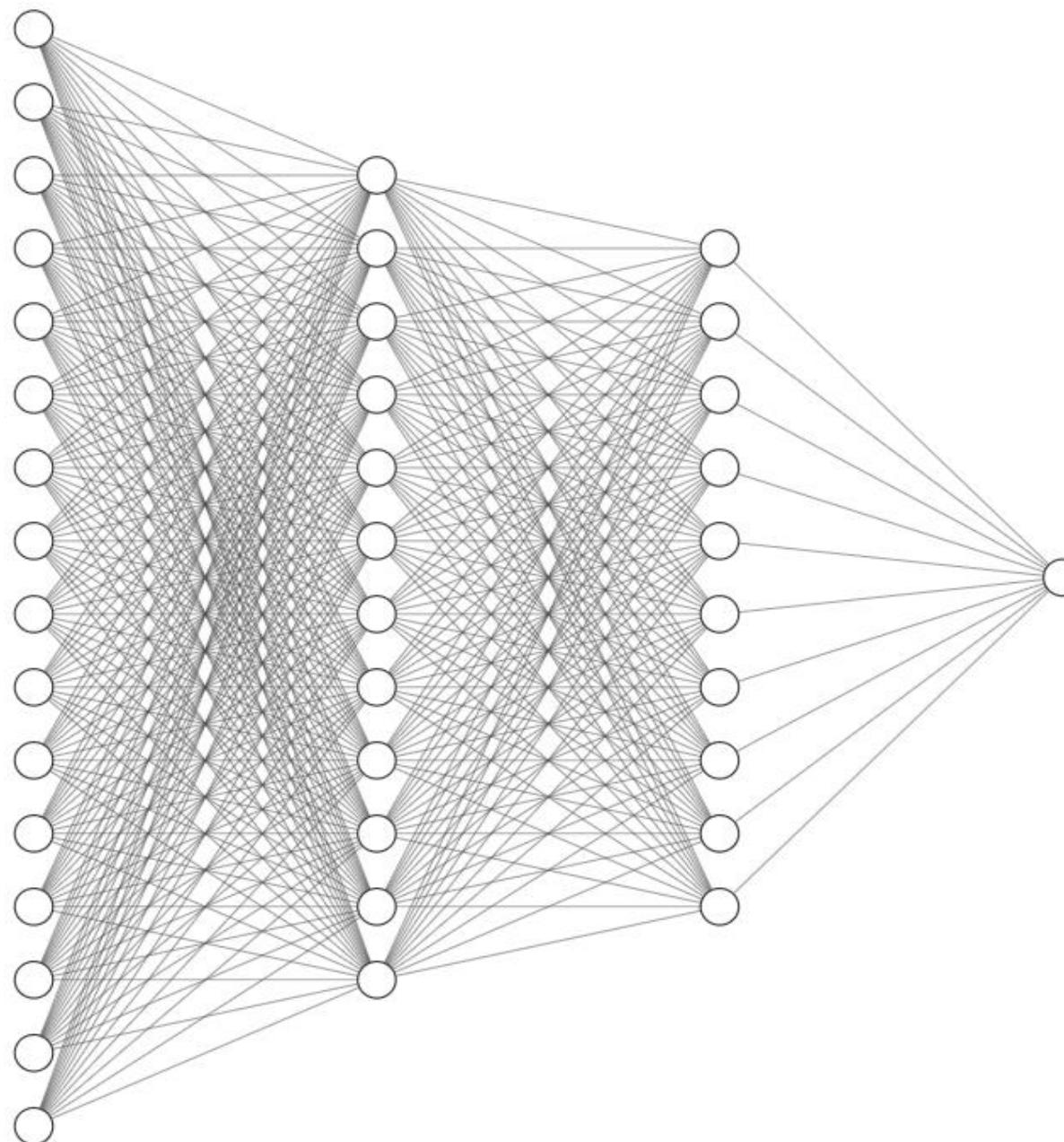
Edge color proportional to edge weights

Negative Edge Color

Positive Edge Color

Default Edge Color

Use Bezier Curves



Input Layer $\in \mathbb{R}^{16}$

Hidden Layer $\in \mathbb{R}^{12}$

Hidden Layer $\in \mathbb{R}^{12}$

Output Layer $\in \mathbb{R}^1$

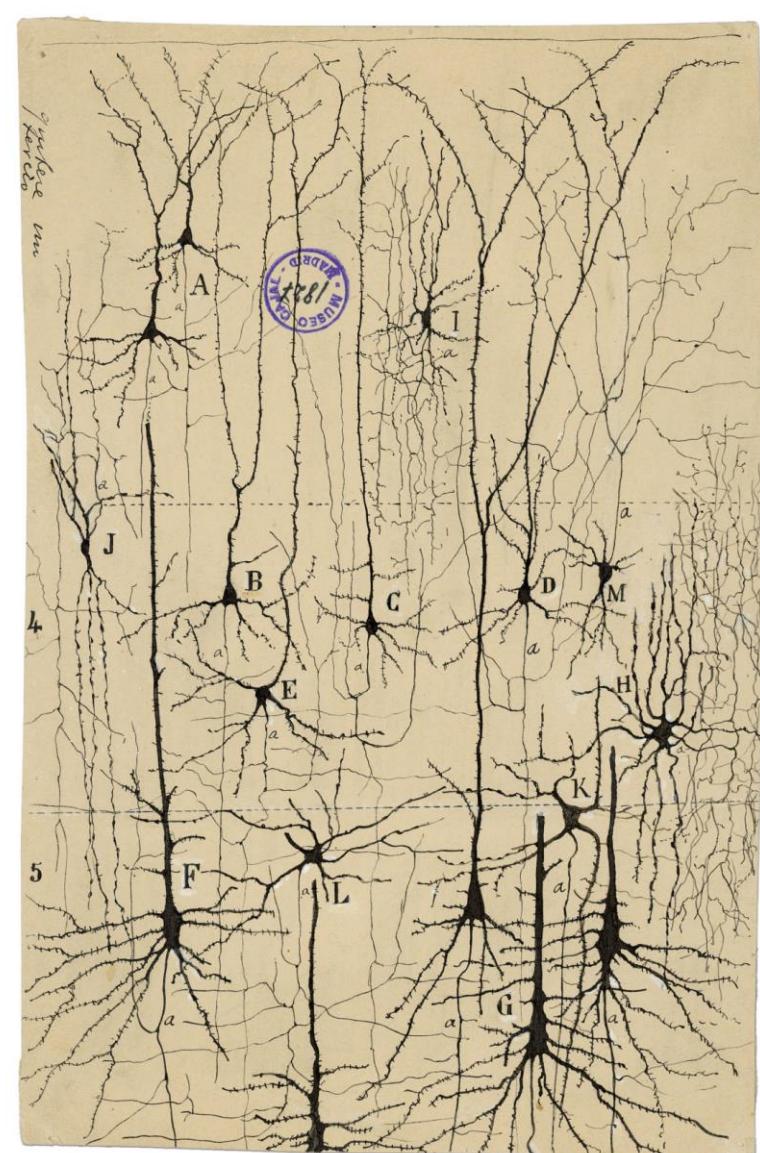
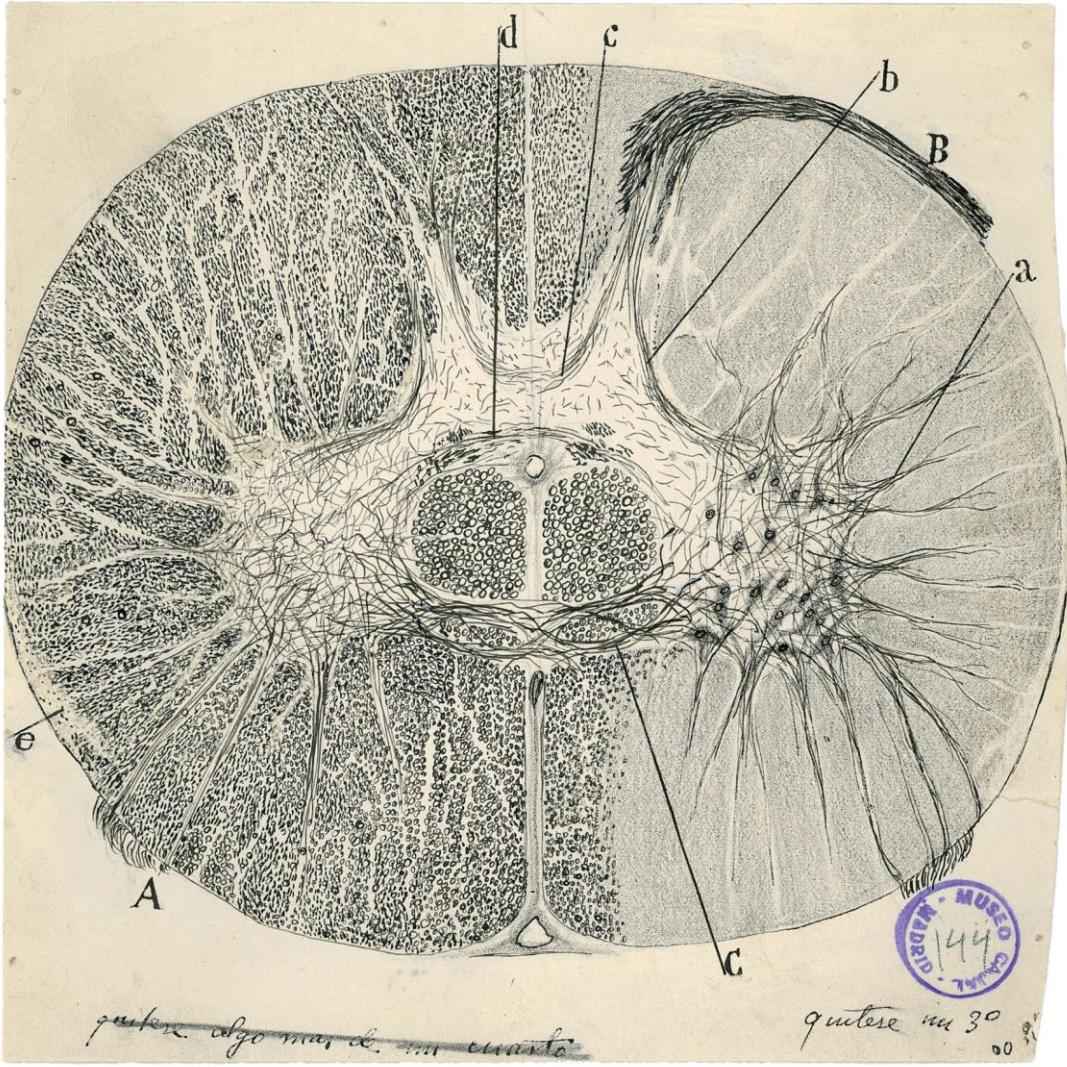


ANN

ARTIFICIAL NEURAL NETWORK

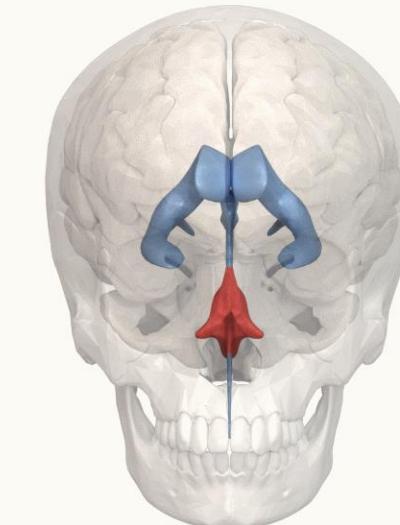
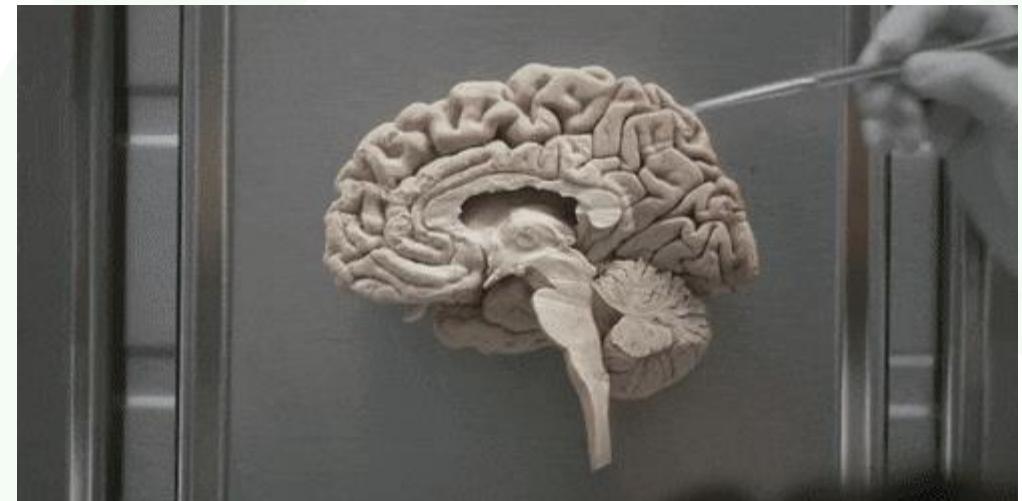
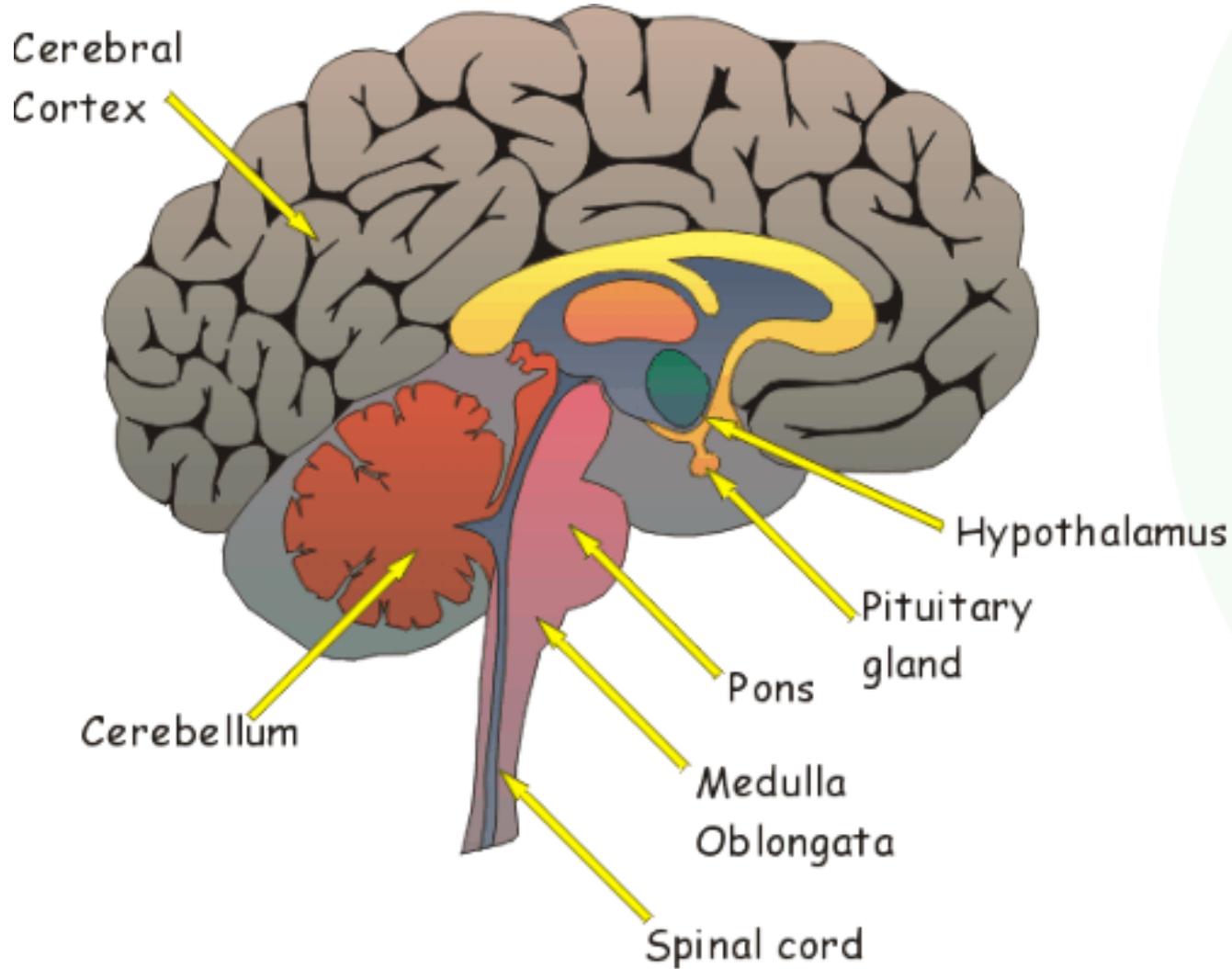


ARTIFICIAL NEURAL NETWORK





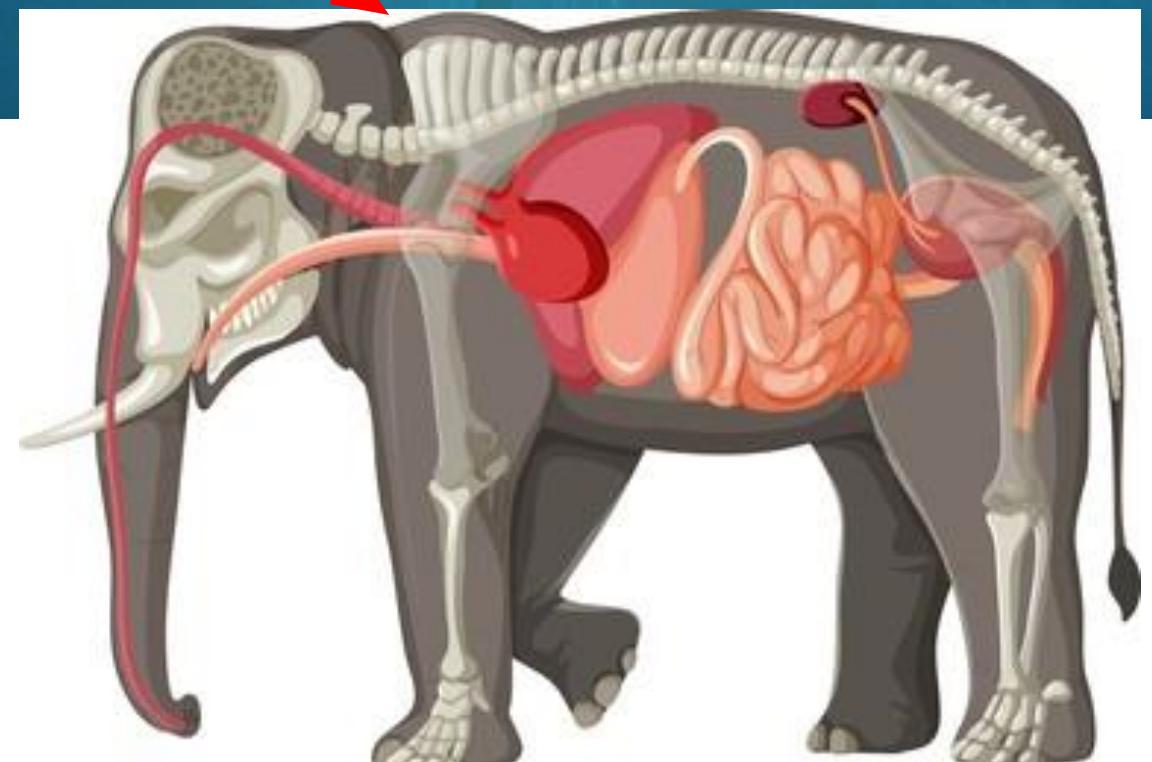
ARTIFICIAL NEURAL NETWORK



HUMAN



1.35 KG

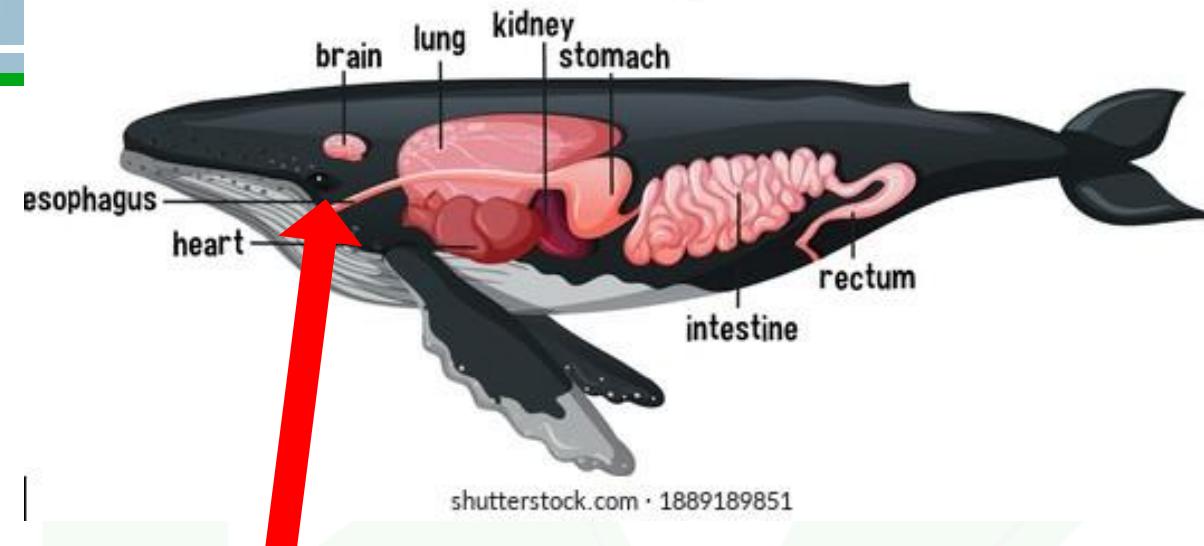


ELEPHANT



5.99 KG

Internal Anatomy of a Whale



shutterstock.com · 1889189851



HUMAN
BRAIN



WHALE





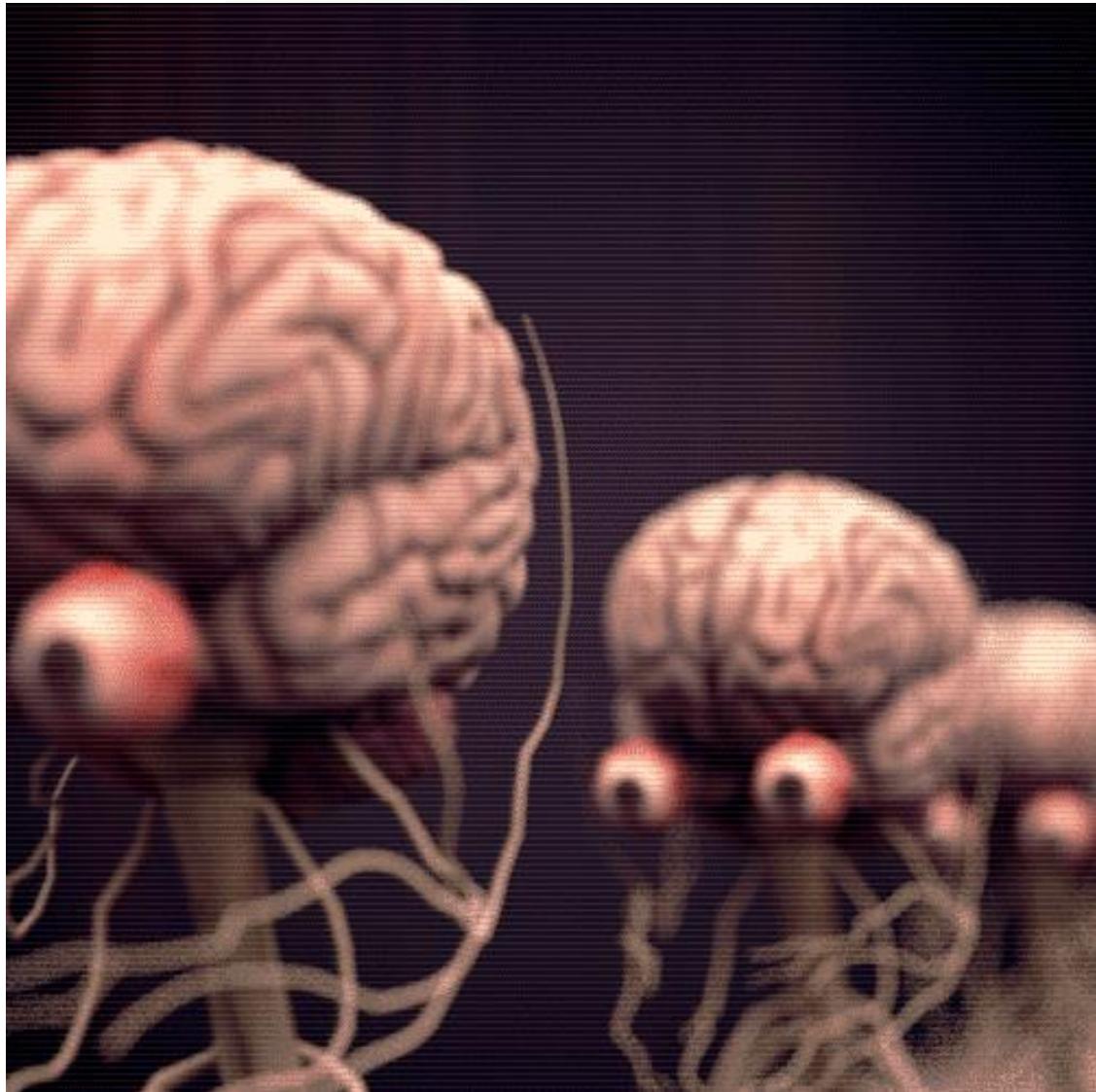
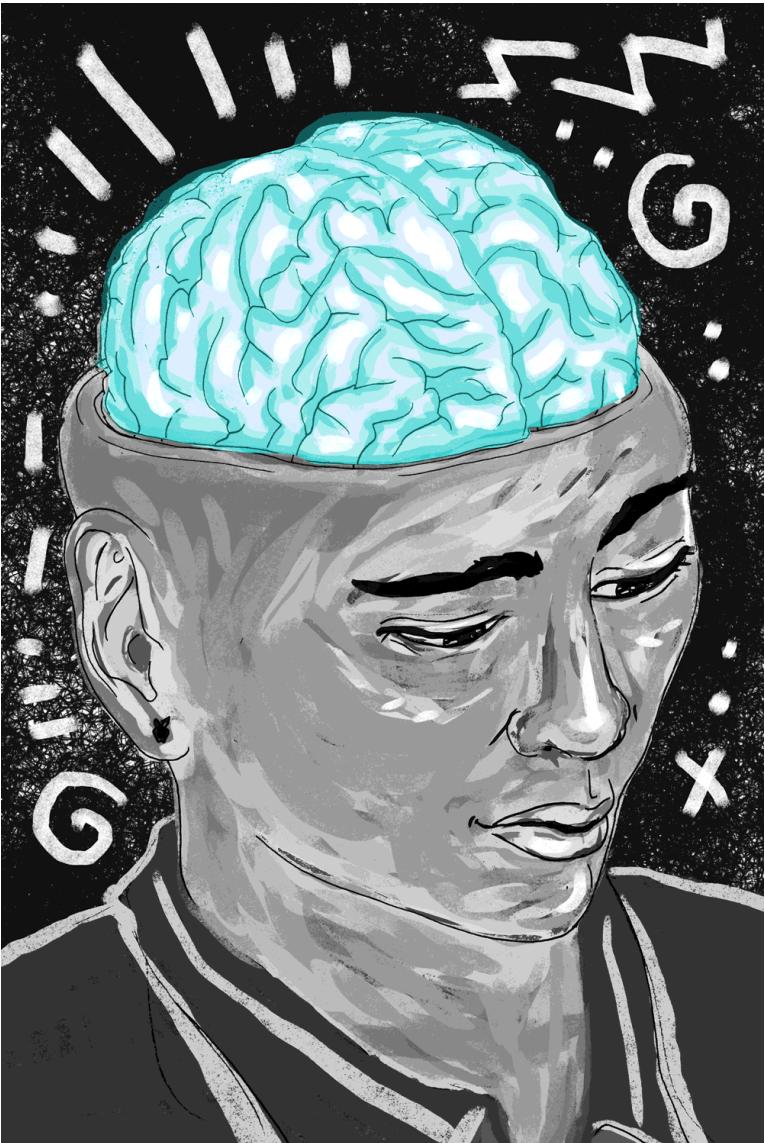
ARTIFICIAL NEURAL NETWORK

86 BILLION NEURONS



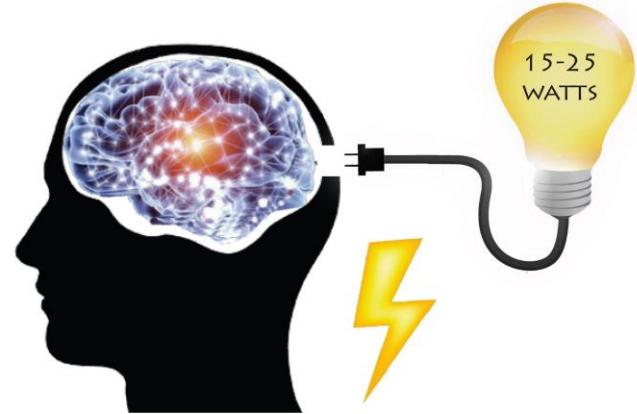


ARTIFICIAL NEURAL NETWORK





ARTIFICIAL NEURAL NETWORK



PER MINUTE

1.000.000.000.000 byte





10.000.000.000.000.000

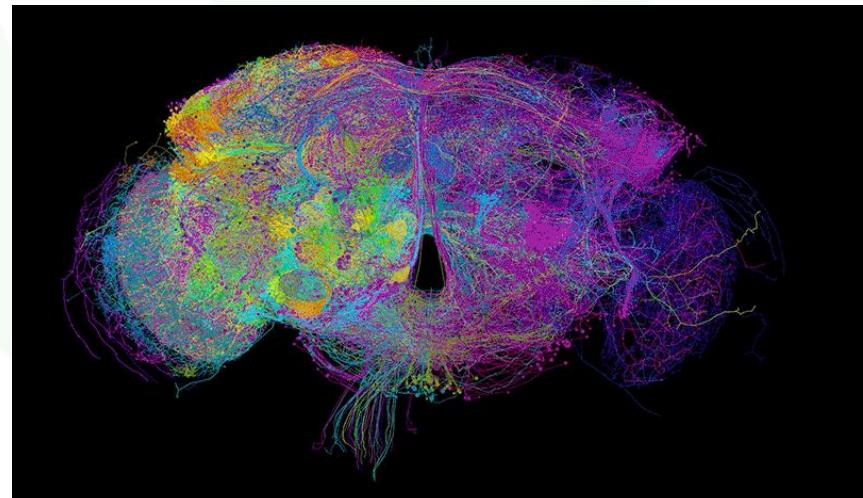


There are 100 billion neurons in the adult human brain, and each neuron makes something like 1,000 to 10,000 contacts with other neurons in the brain. Based on this, people have calculated that the number of permutations and combinations of brain activity exceeds the number of elementary particles in the universe.

— *Vilayanur S. Ramachandran* —



ARTIFICIAL NEURAL NETWORK

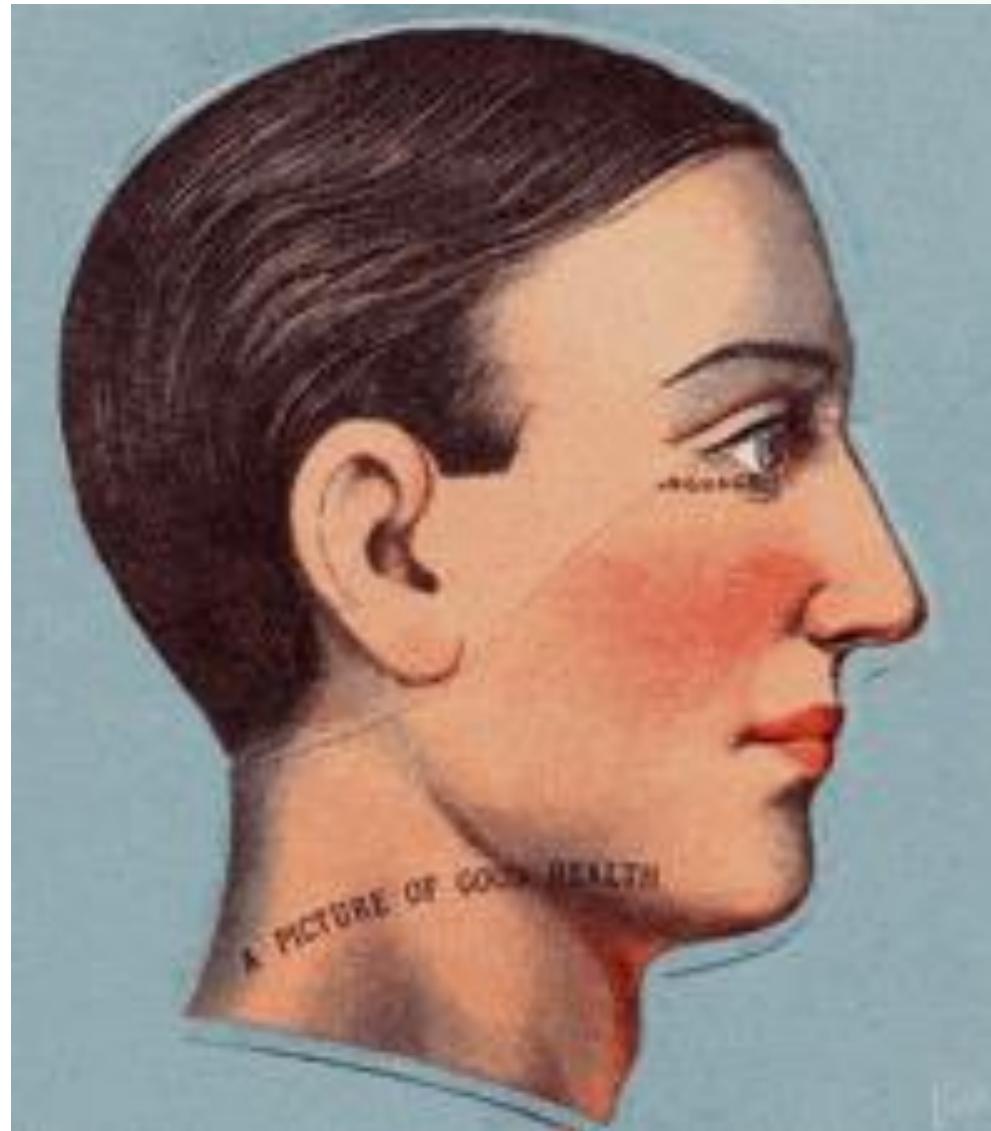


Reset





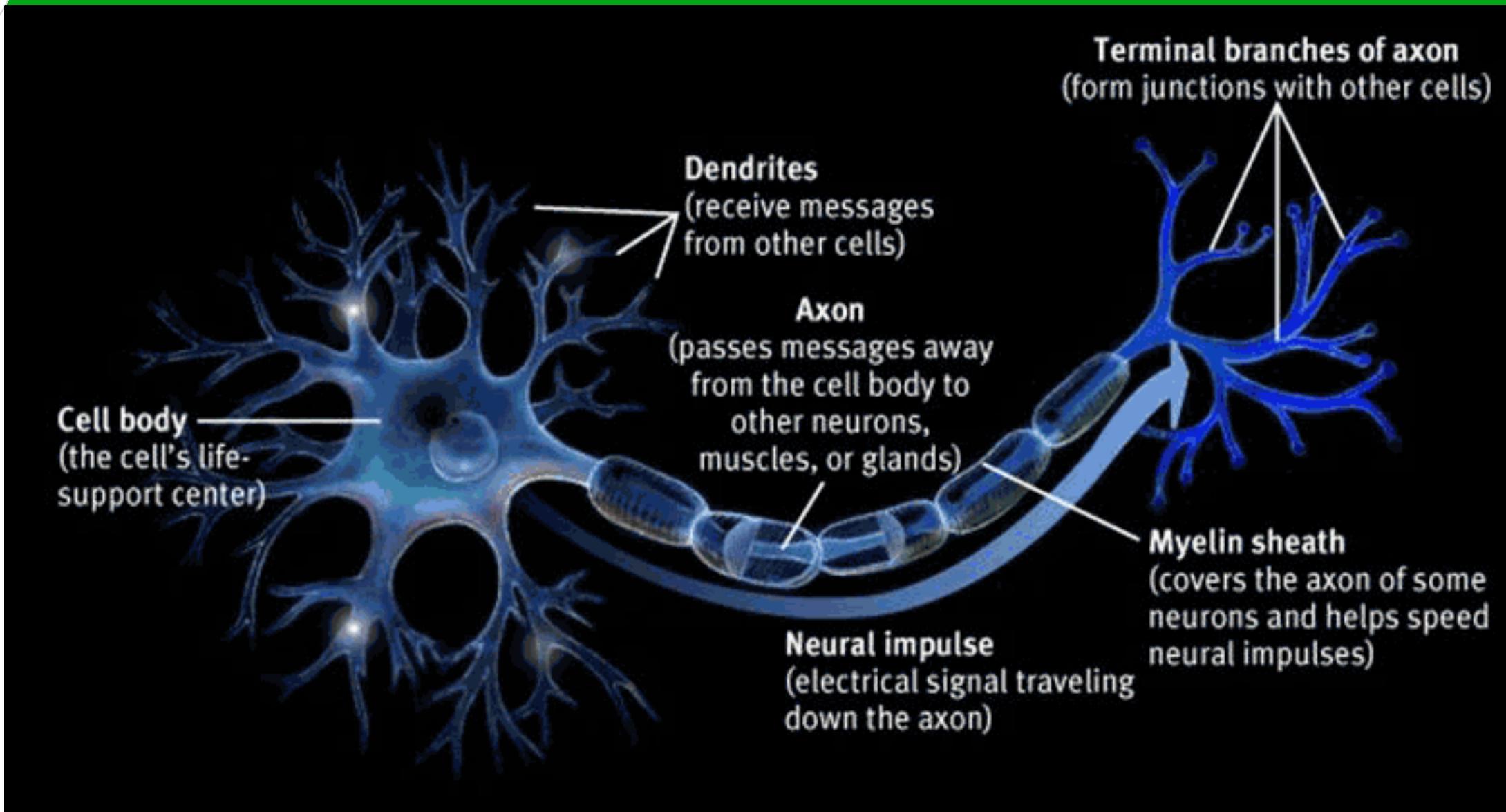
ARTIFICIAL NEURAL NETWORK





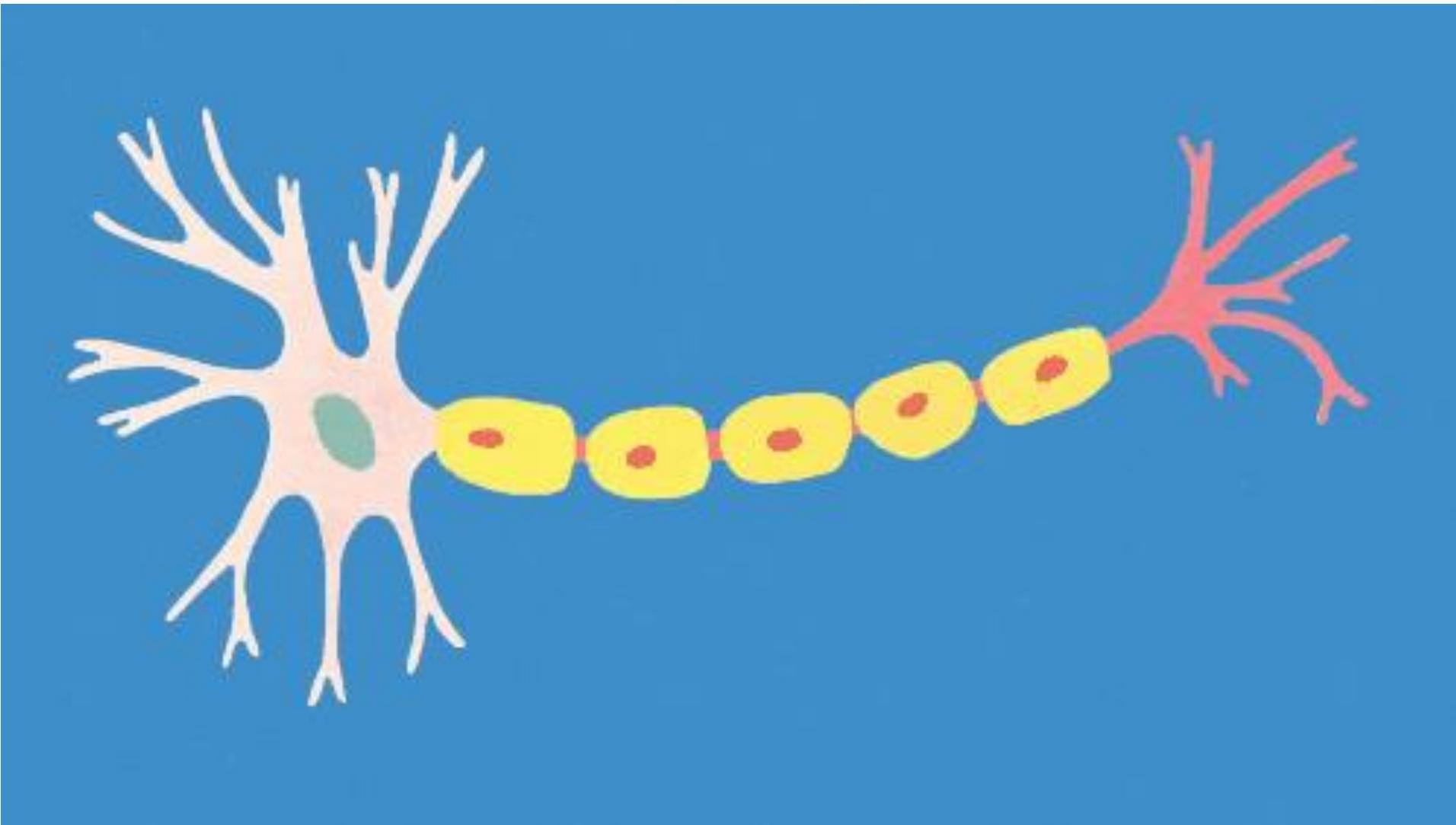
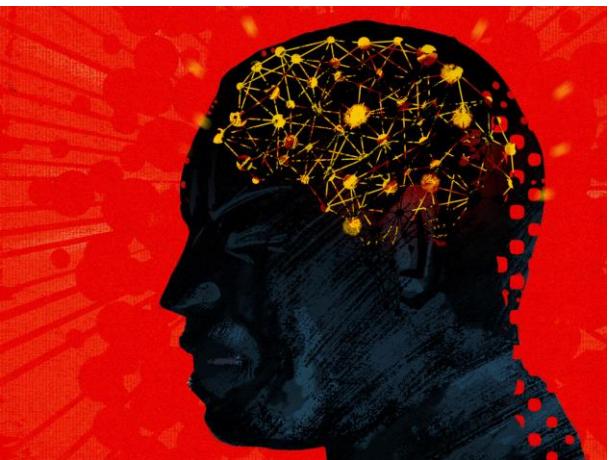


ARTIFICIAL NEURAL NETWORK





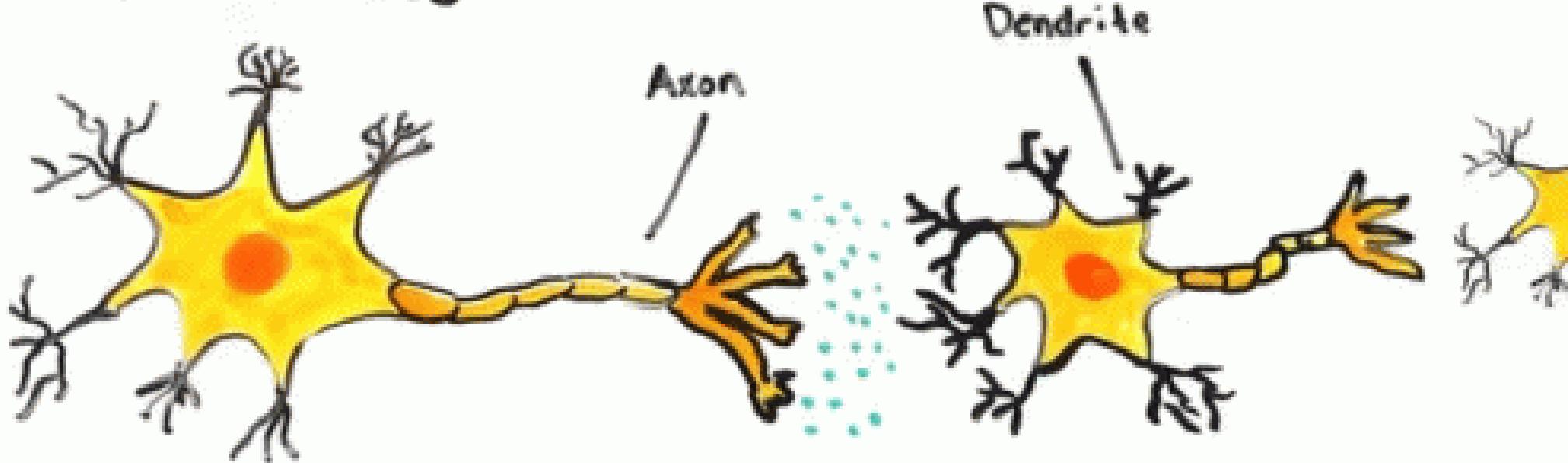
ARTIFICIAL NEURAL NETWORK





ARTIFICIAL NEURAL NETWORK

NEURONS



NEUROTRANSMITTERS

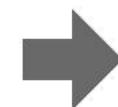
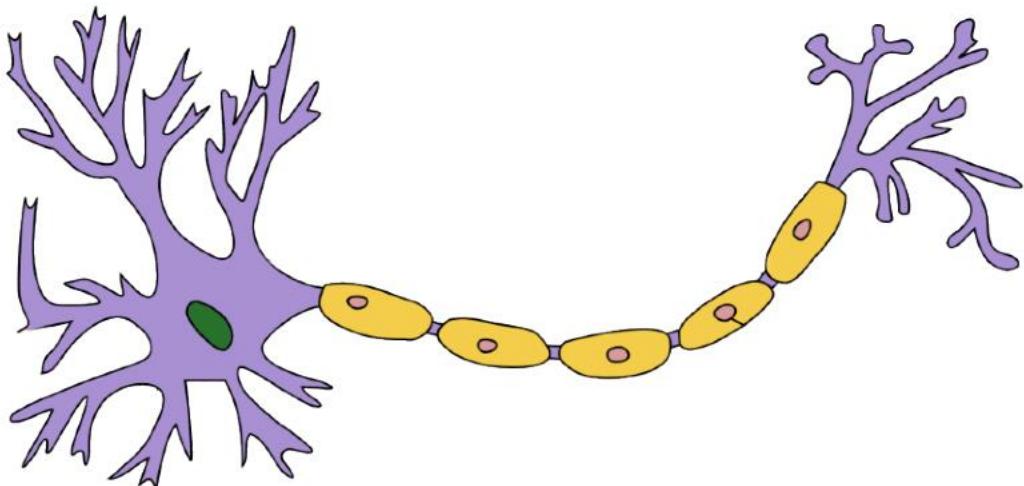


PERCEPTRON MODELS

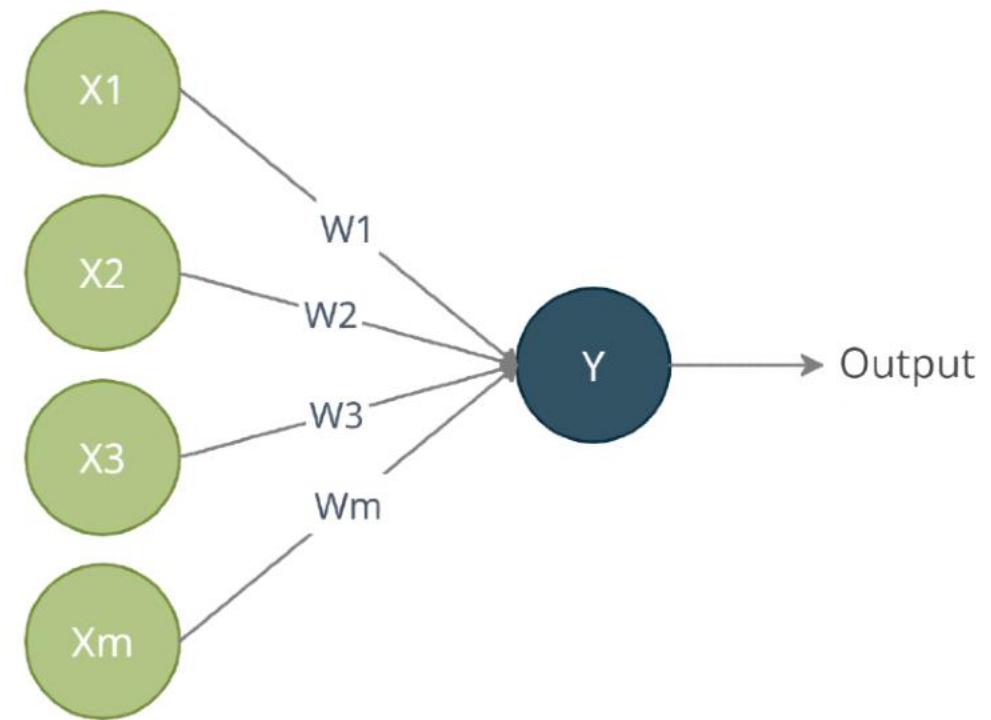


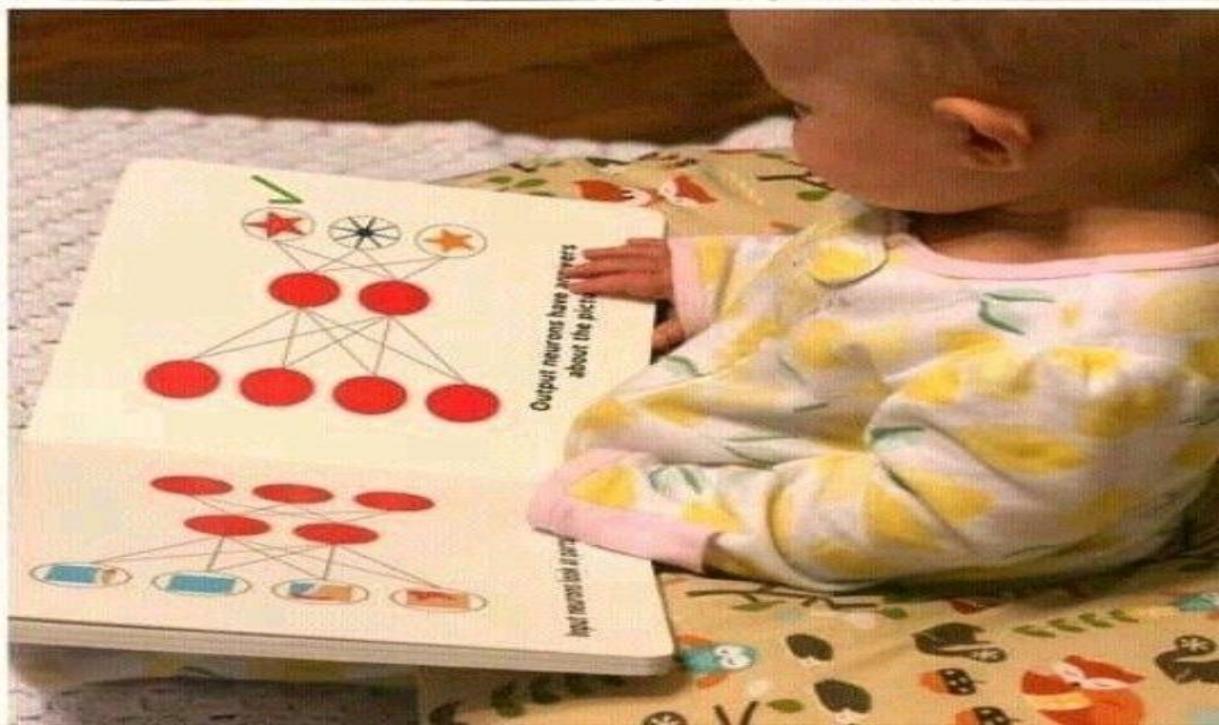
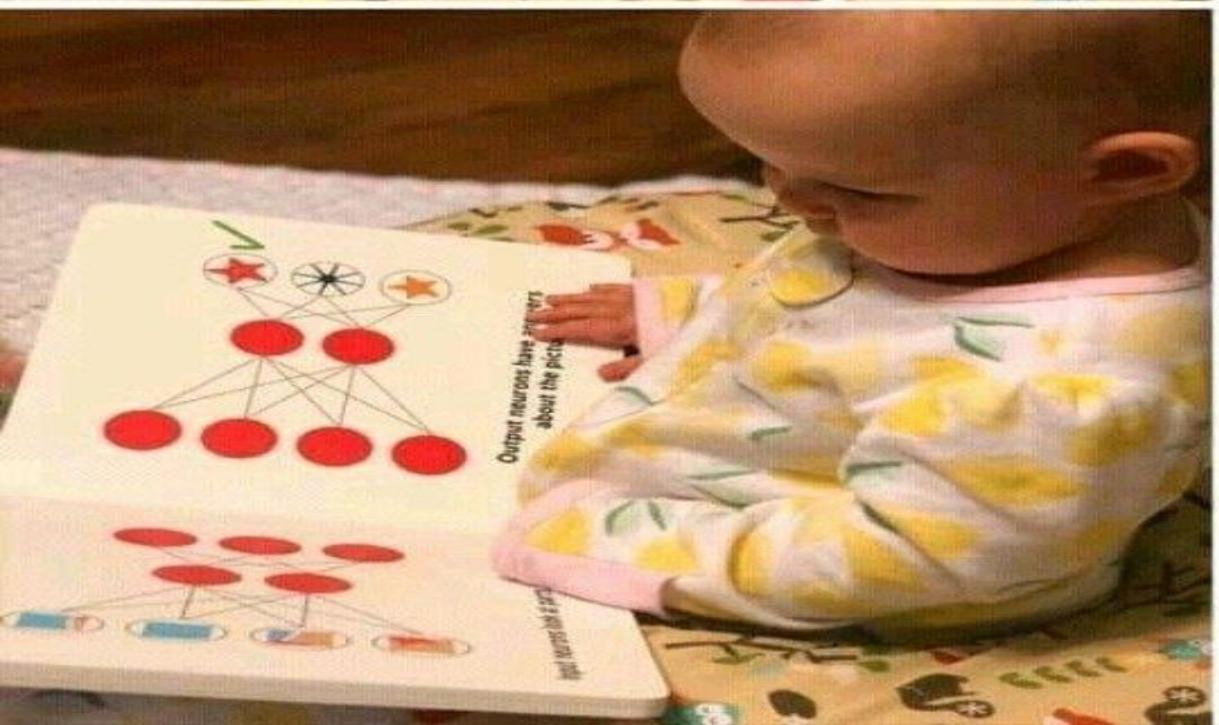
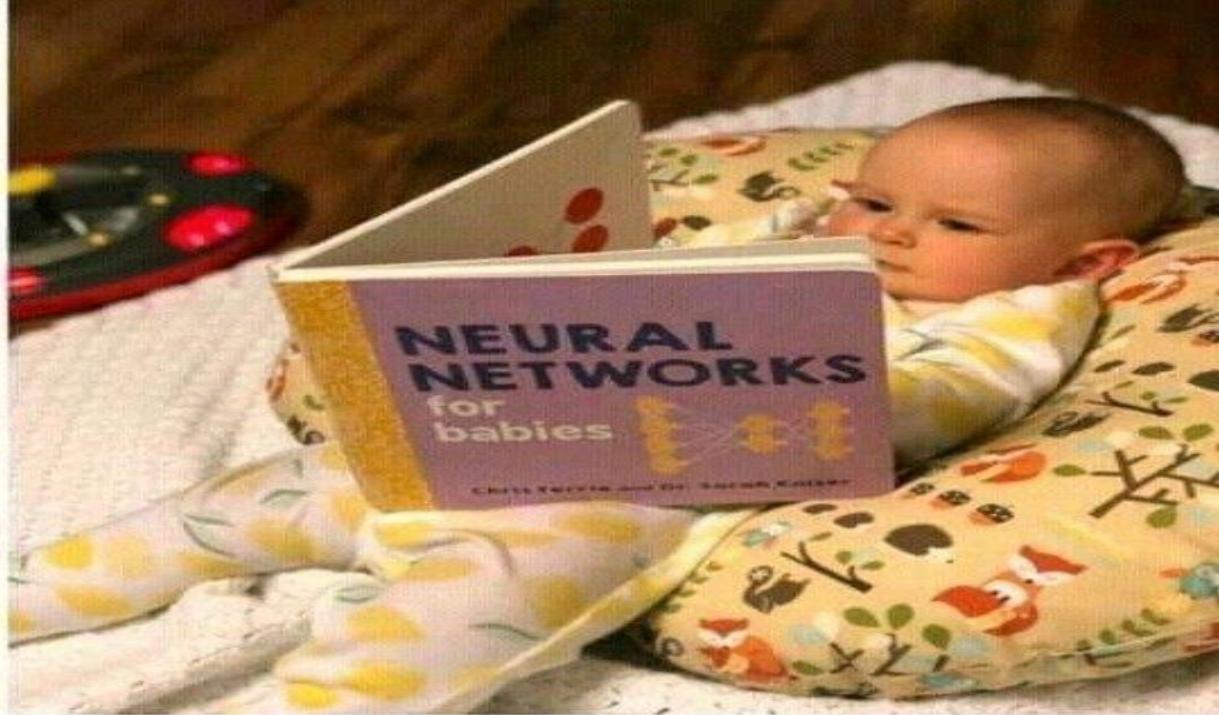
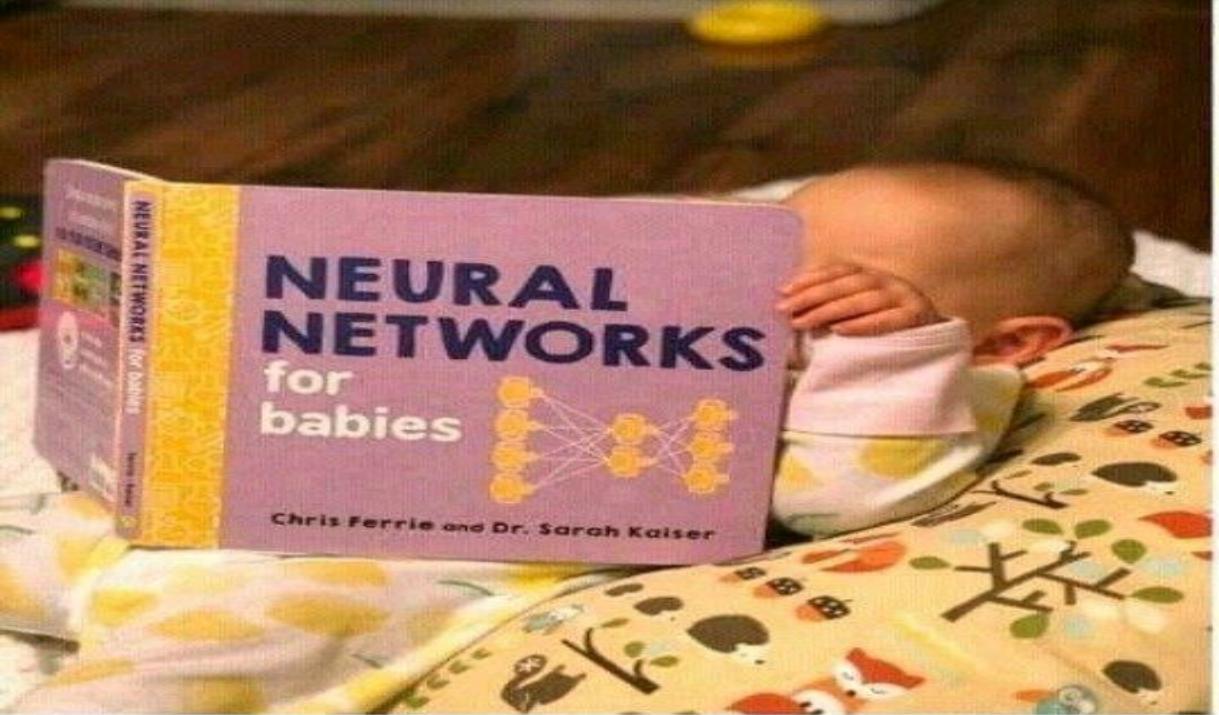
PERCEPTRON MODELS

Neuron



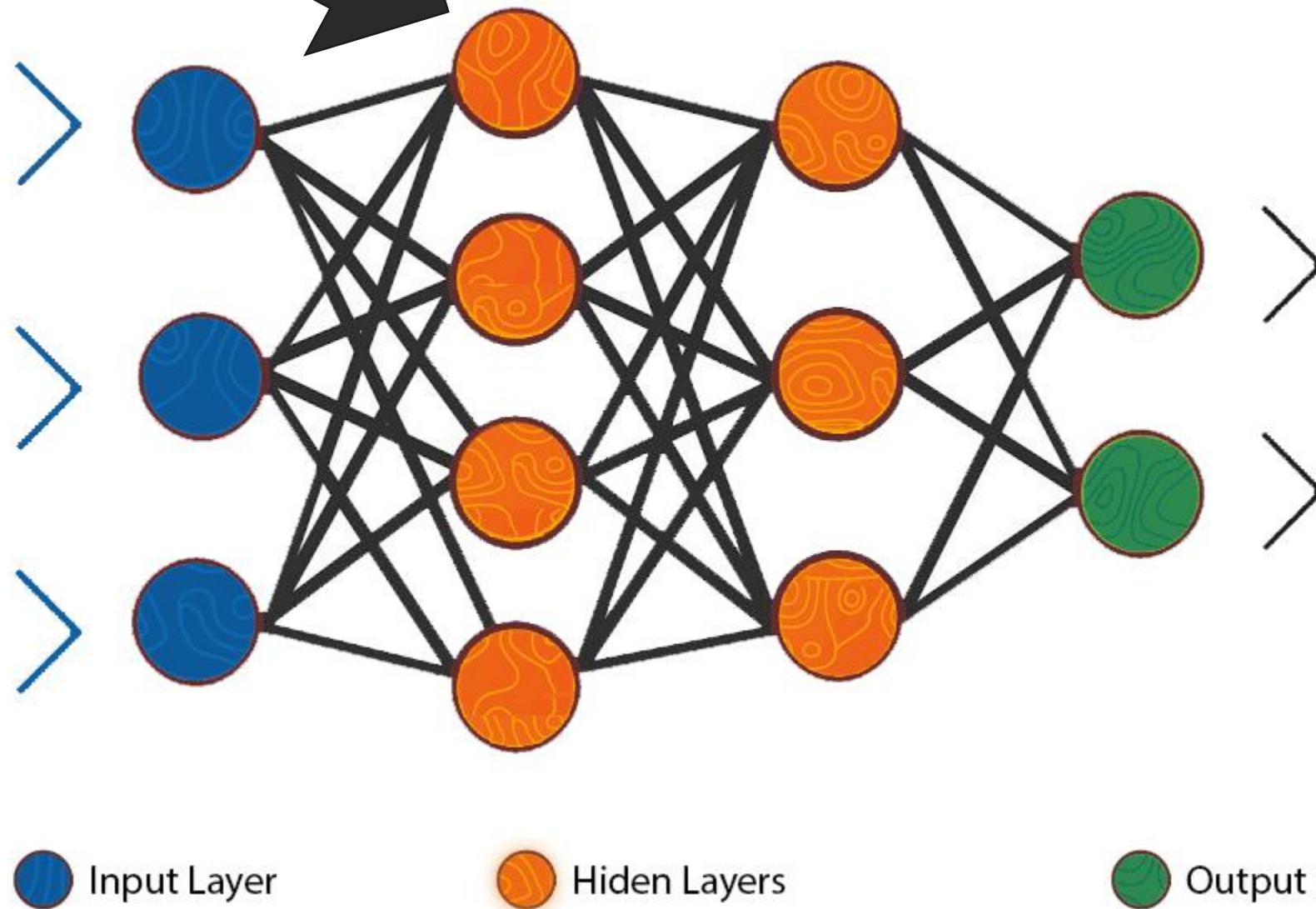
Perceptron





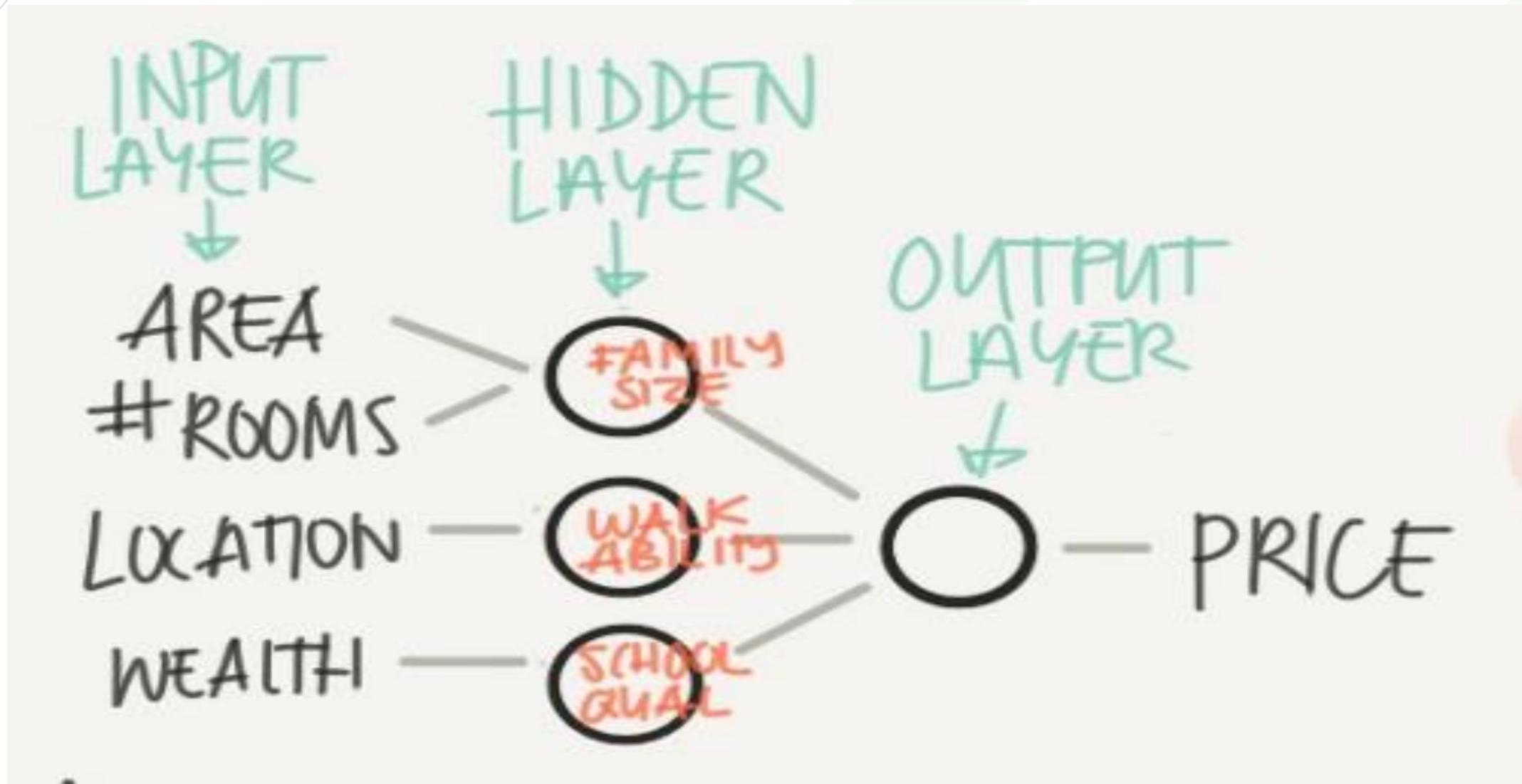


NEURON (NODE)



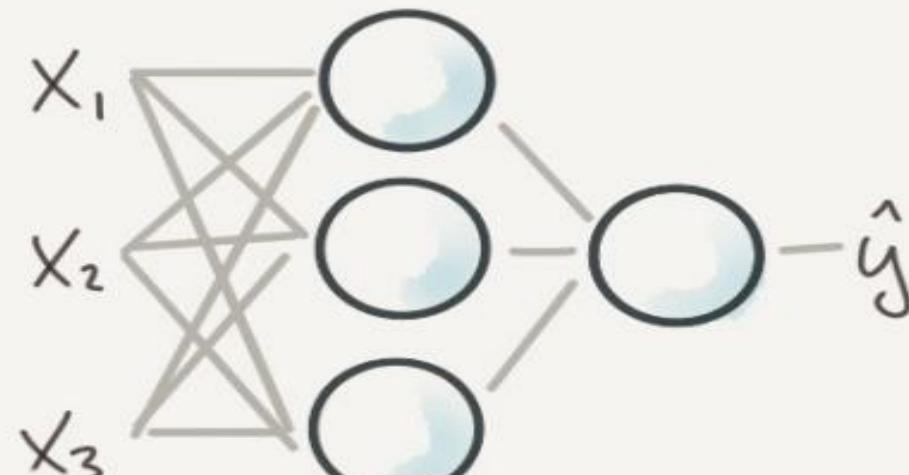


DEEP LEARNING

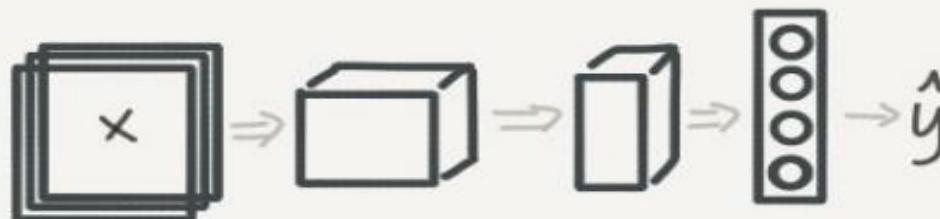




DEEP LEARNING

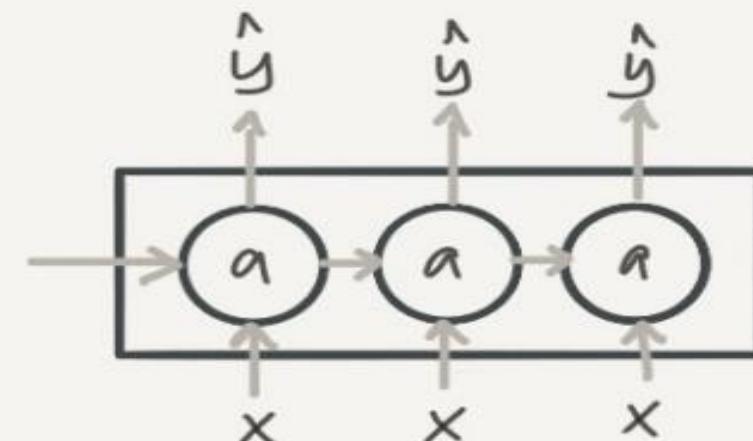


STANDARD NN



CONVOLUTIONAL NN

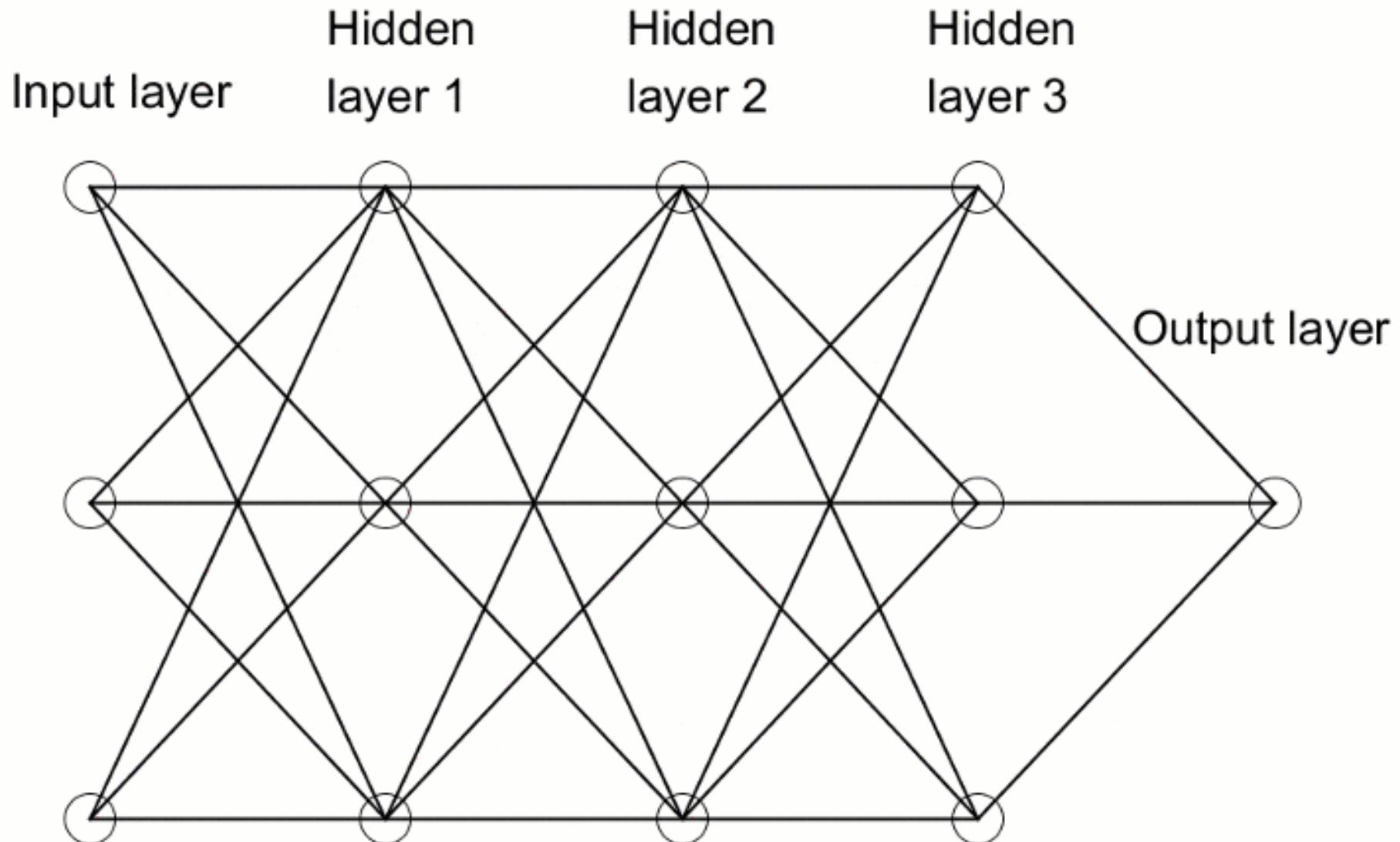
NETWORK
ARCHITECTURES



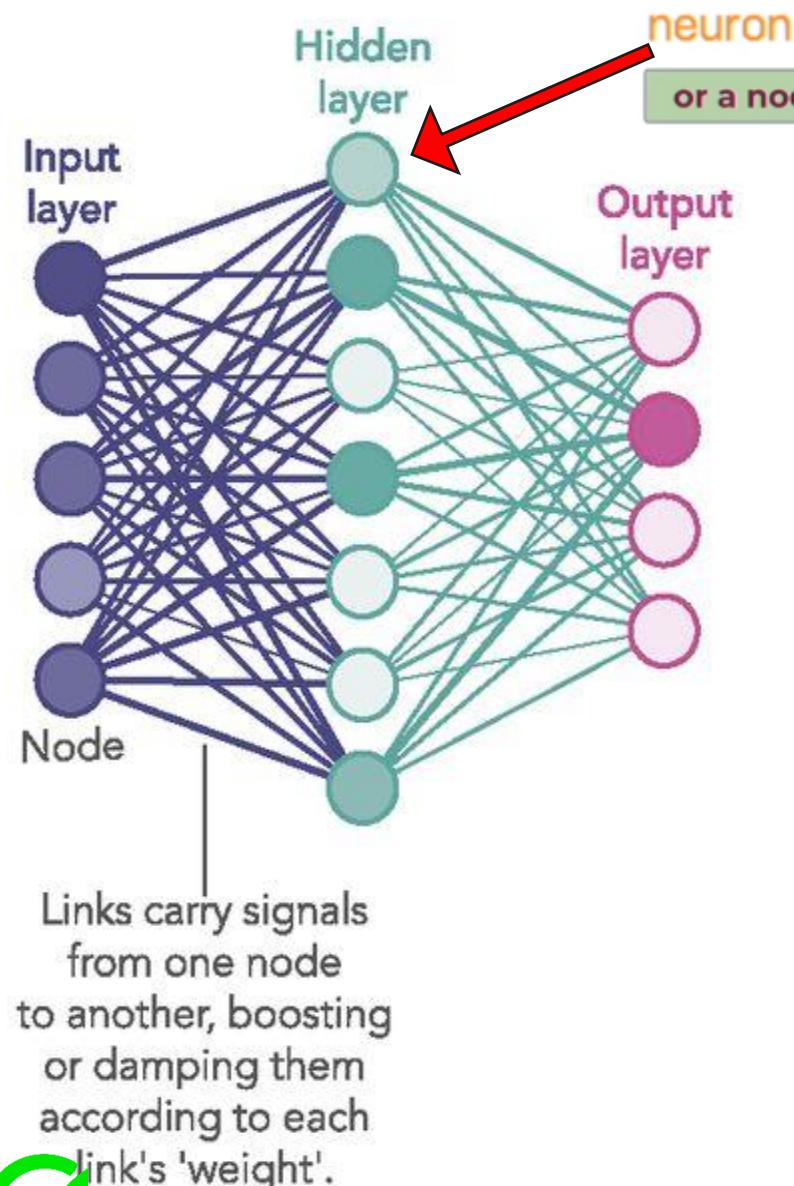
RECURRENT NN



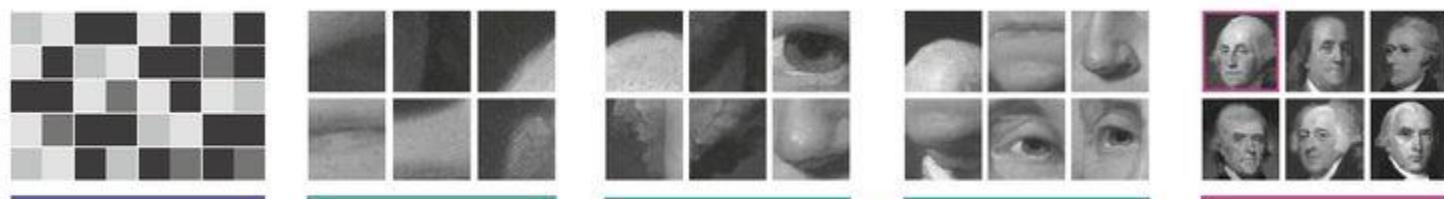
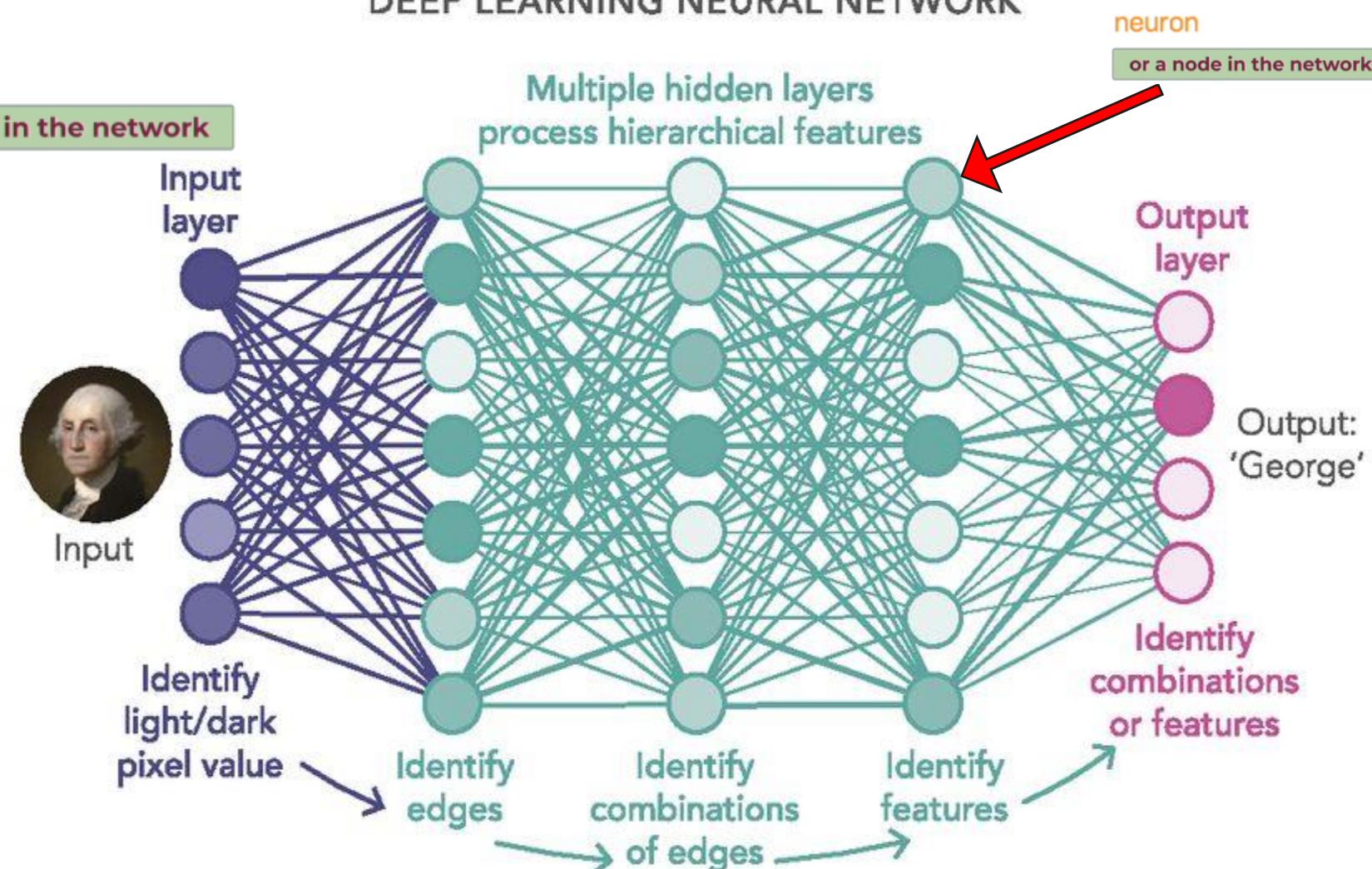
DEEP LEARNING



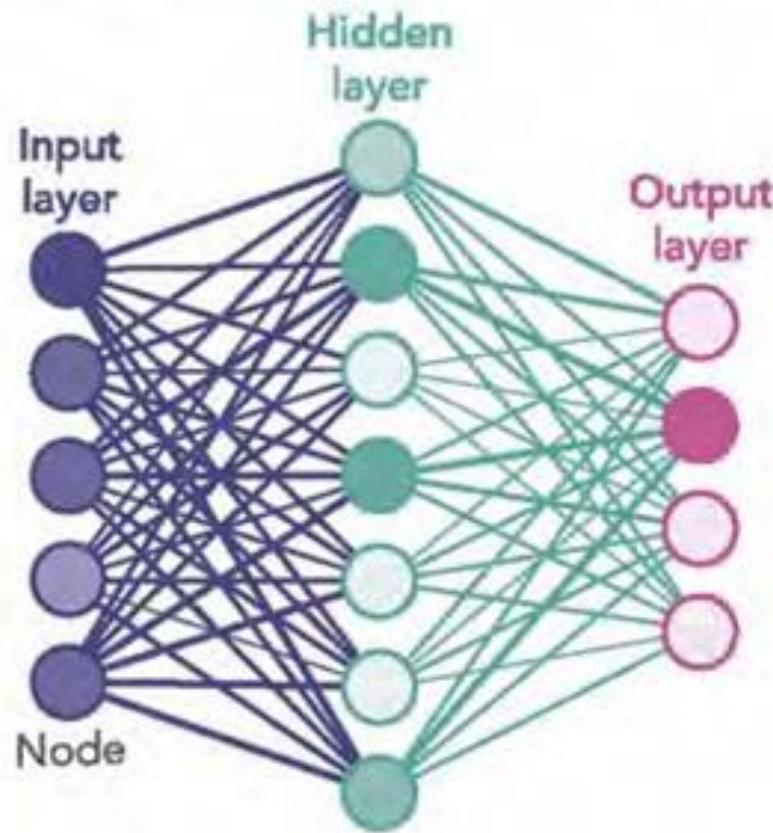
1980S-ERA NEURAL NETWORK



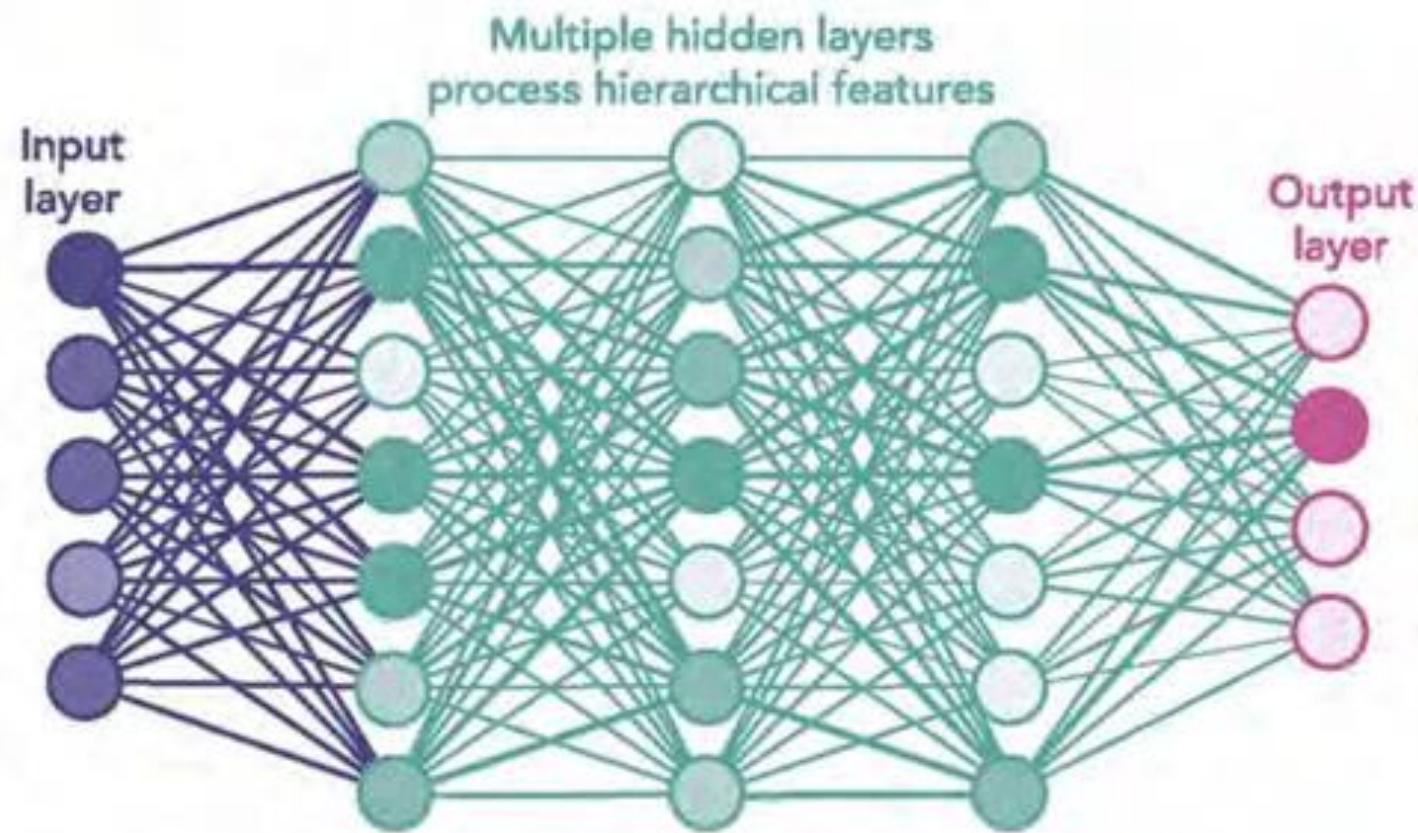
DEEP LEARNING NEURAL NETWORK



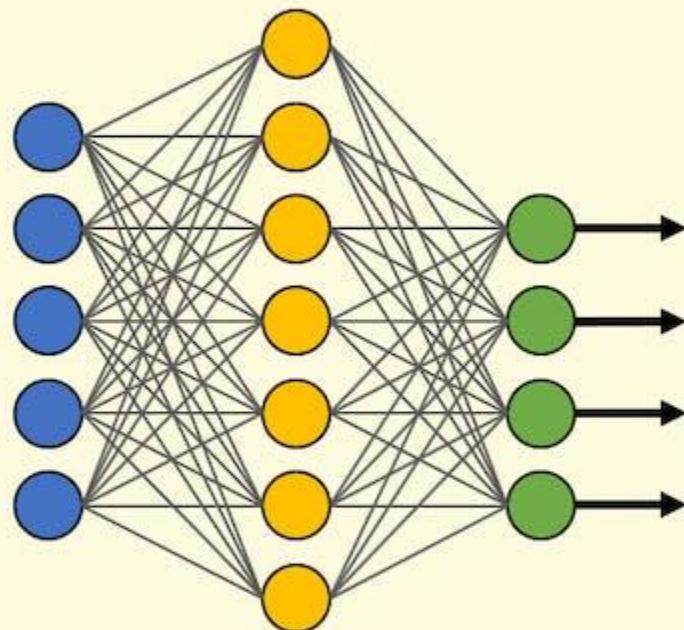
SHALLOW NEURAL NETWORK



DEEP NEURAL NETWORK



Simple Neural Network

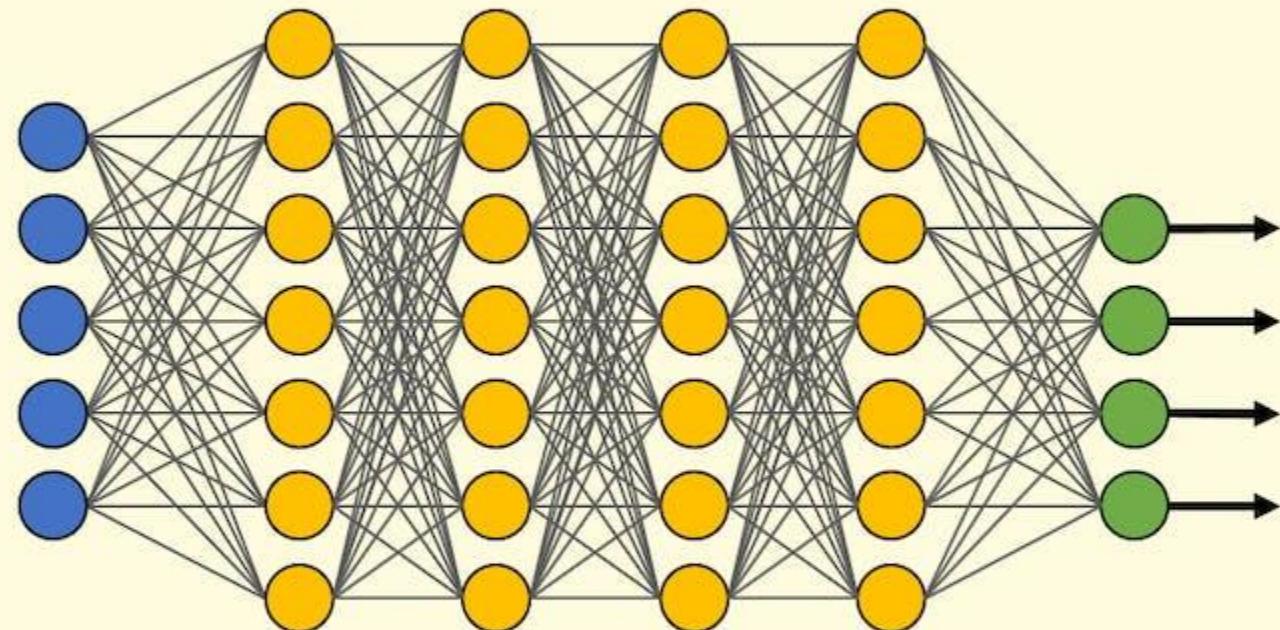


● Input Layer

● Hidden Layer

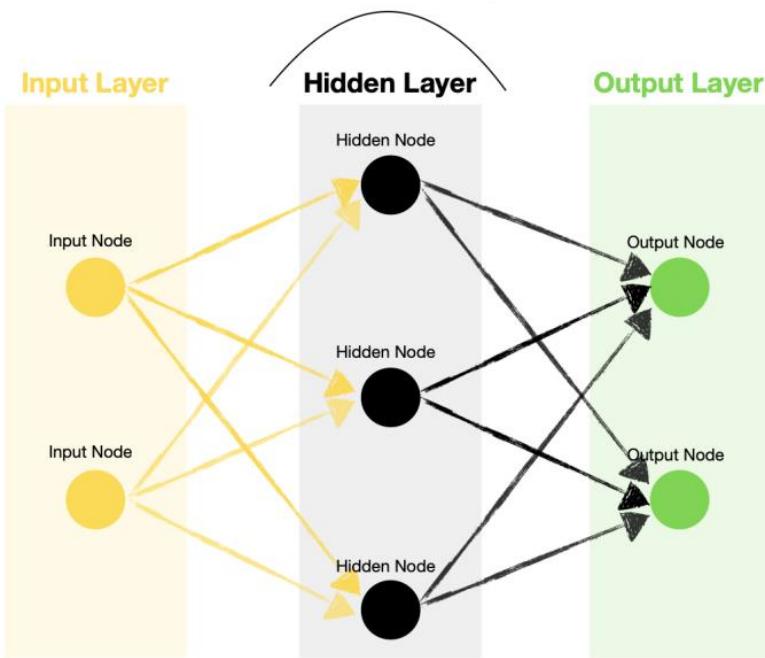
● Output Layer

Deep Learning Neural Network



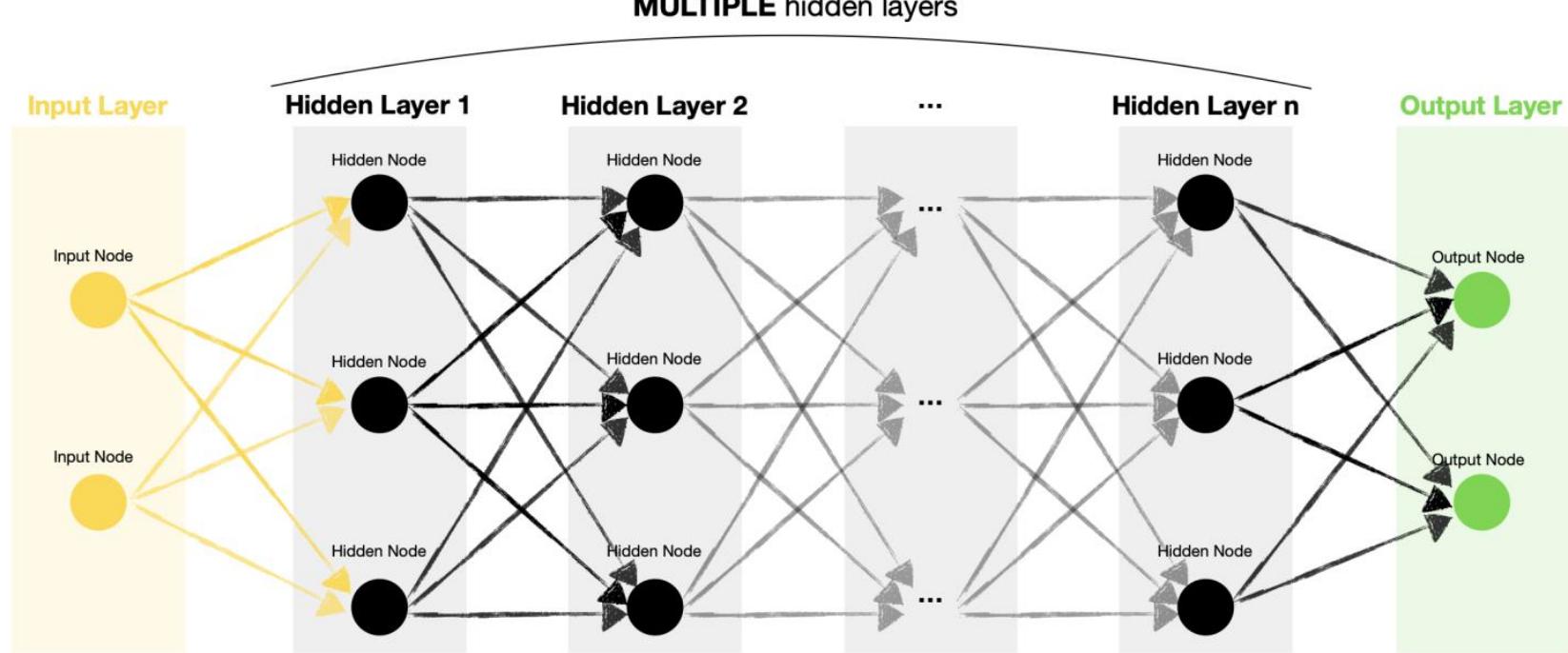
Feed Forward Neural Network

SINGLE hidden layer



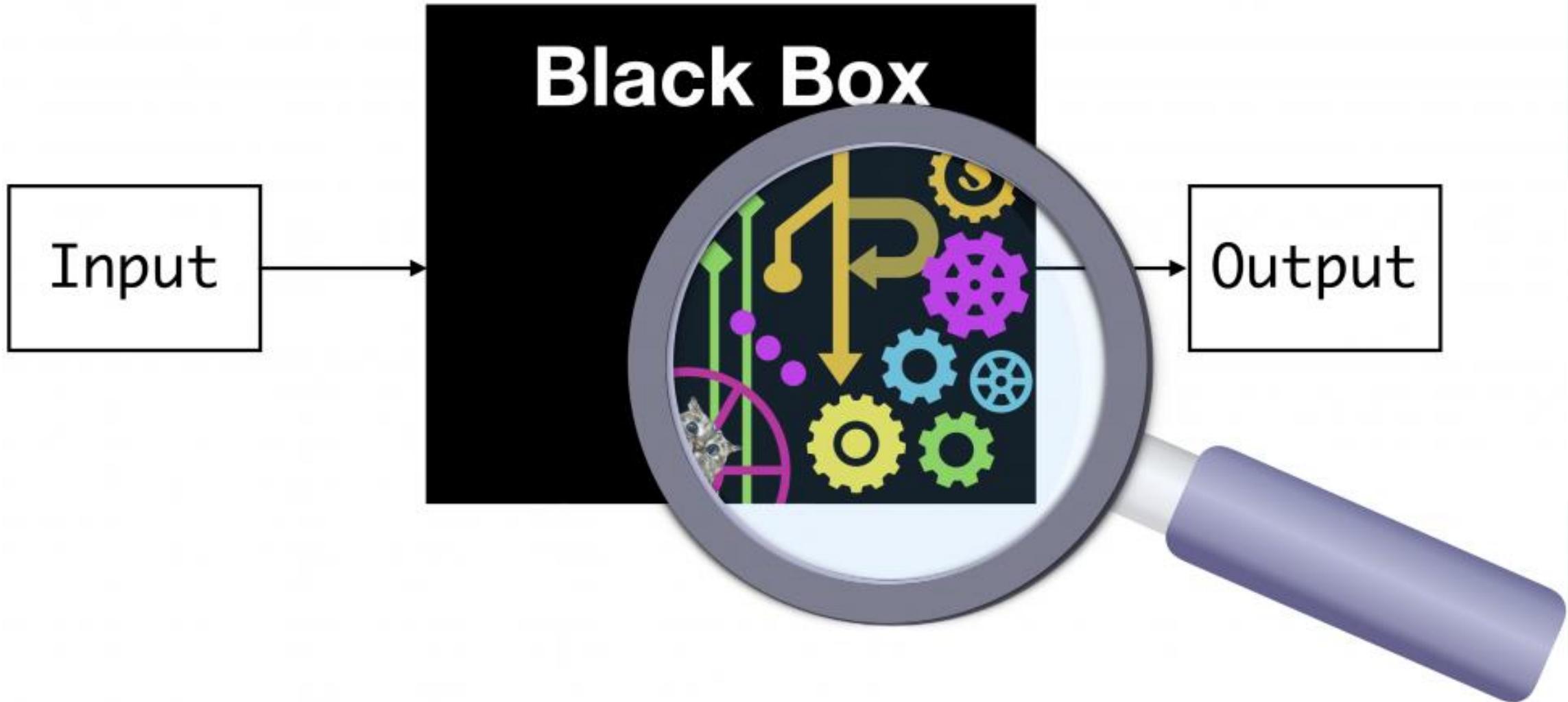
Deep Feed Forward Neural Network

MULTIPLE hidden layers



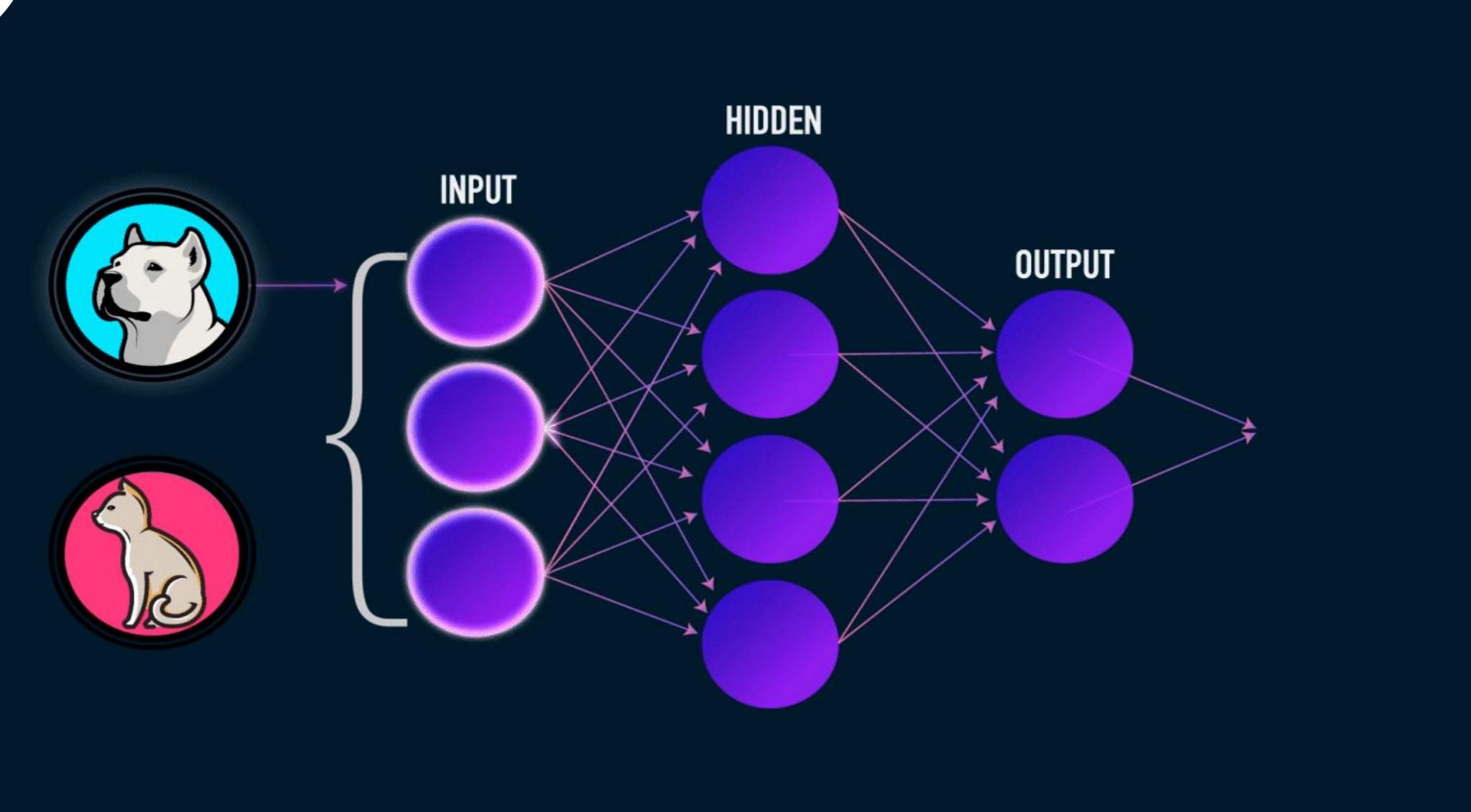


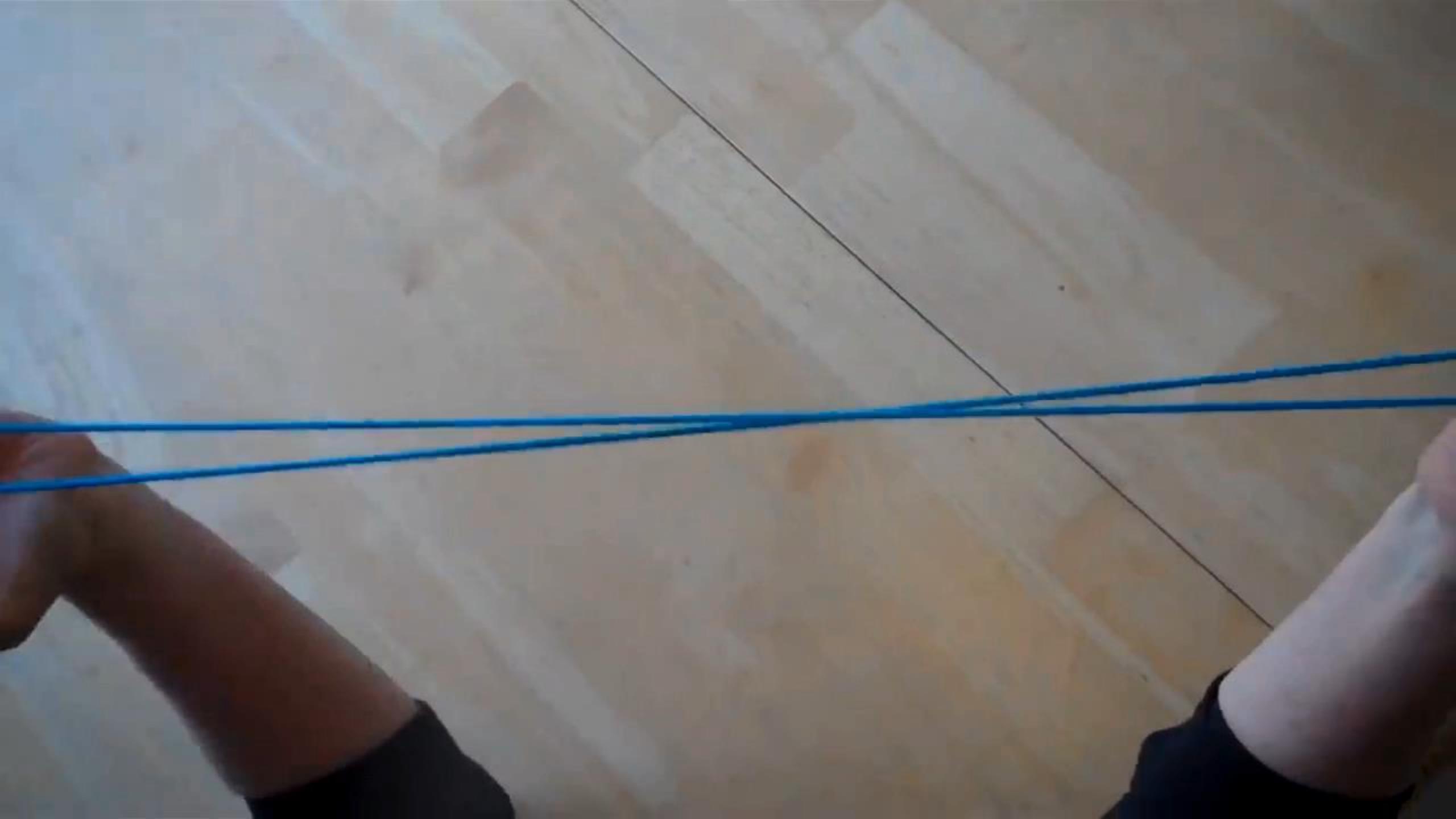
BLACK BOX





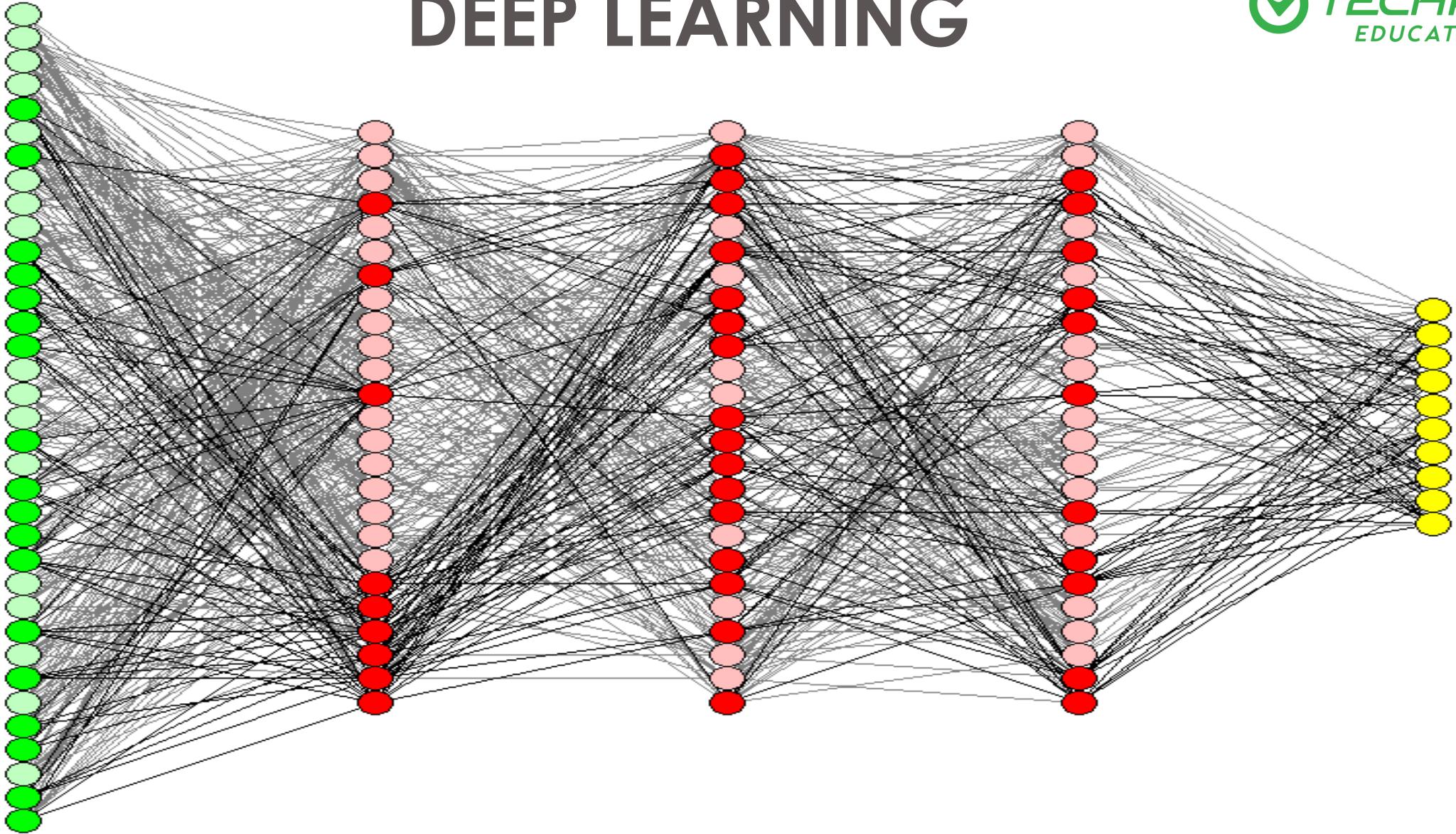
ANN







DEEP LEARNING





DEEP LEARNING

Deep Neural Network

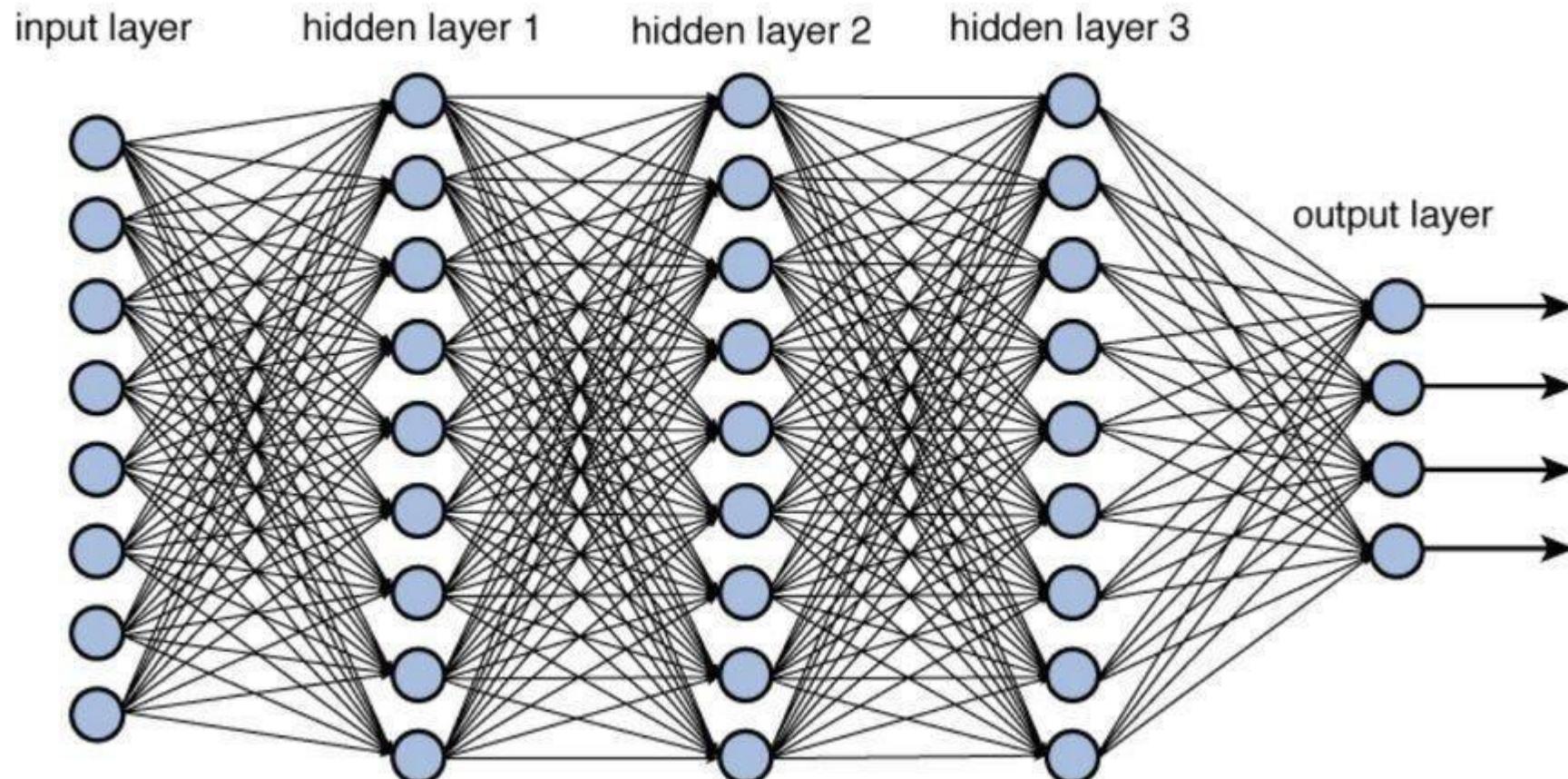
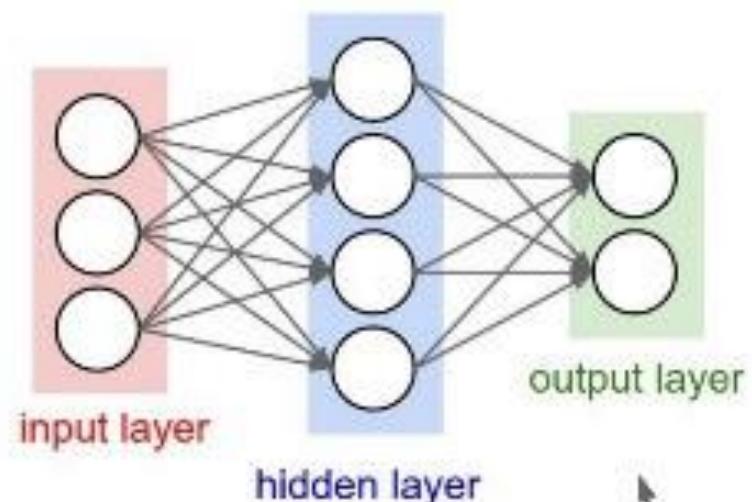


Figure 12.2 Deep network architecture with multiple layers.



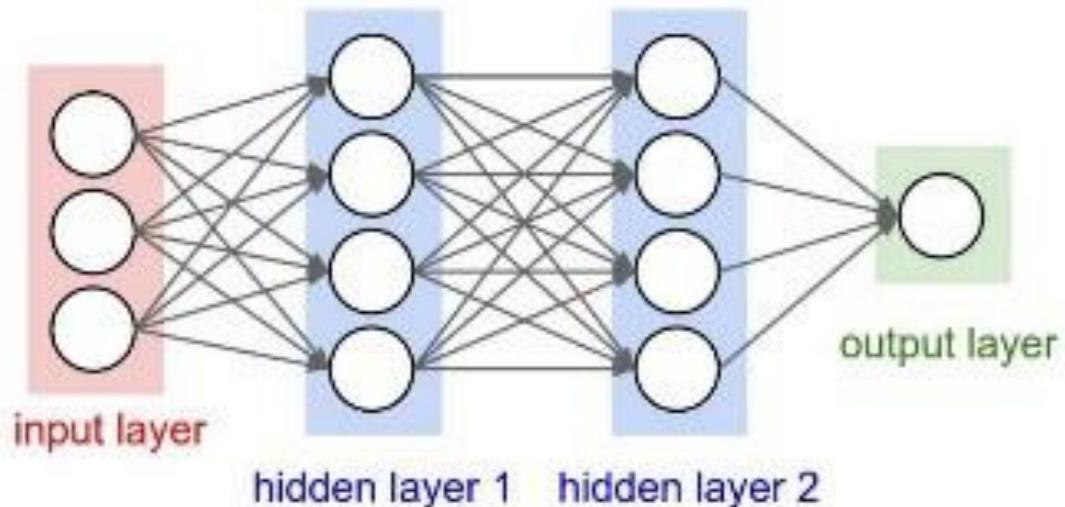
FULLY CONNECTED NEURAL NETWORK

Fully connected neural network



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

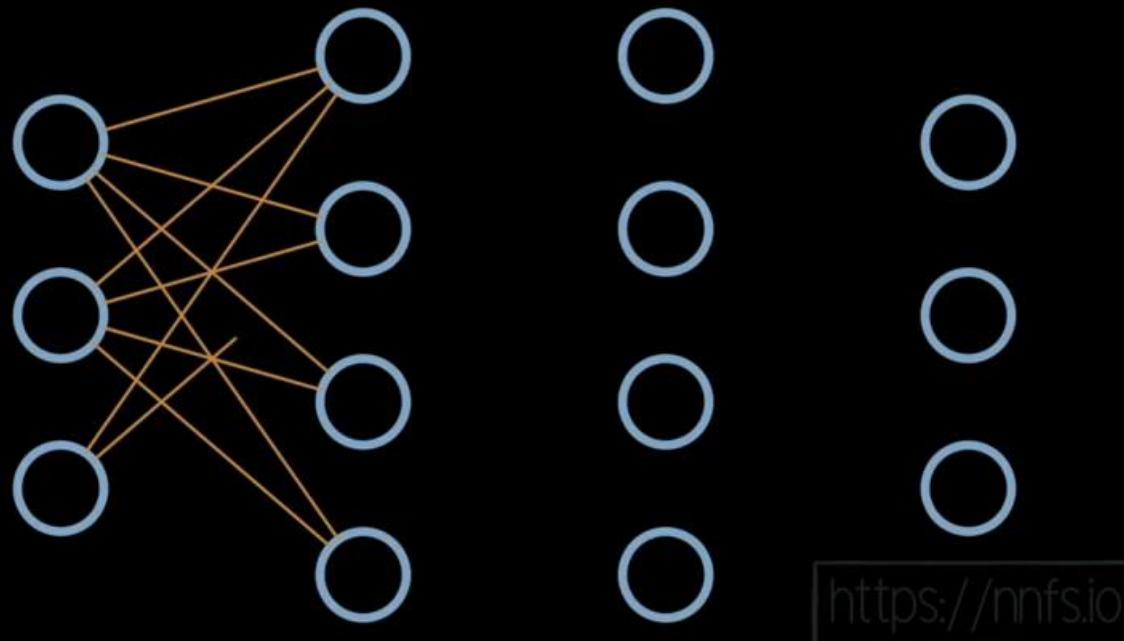
“Fully-connected” layers



“3-layer Neural Net”, or
“2-hidden-layer Neural Net”



PERCEPTRON MODELS





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



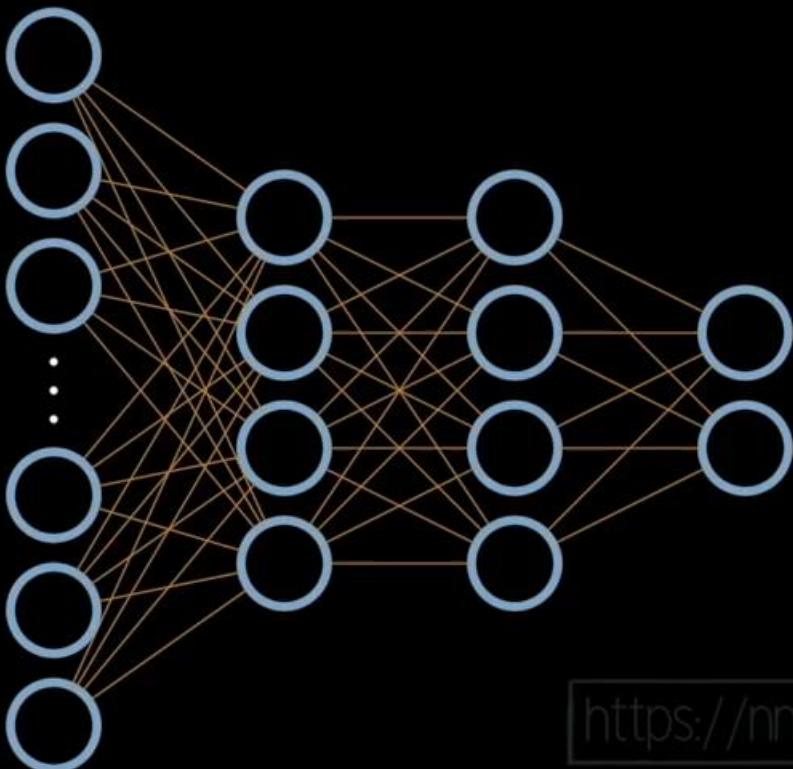
PERCEPTRON MODELS

```
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)
```



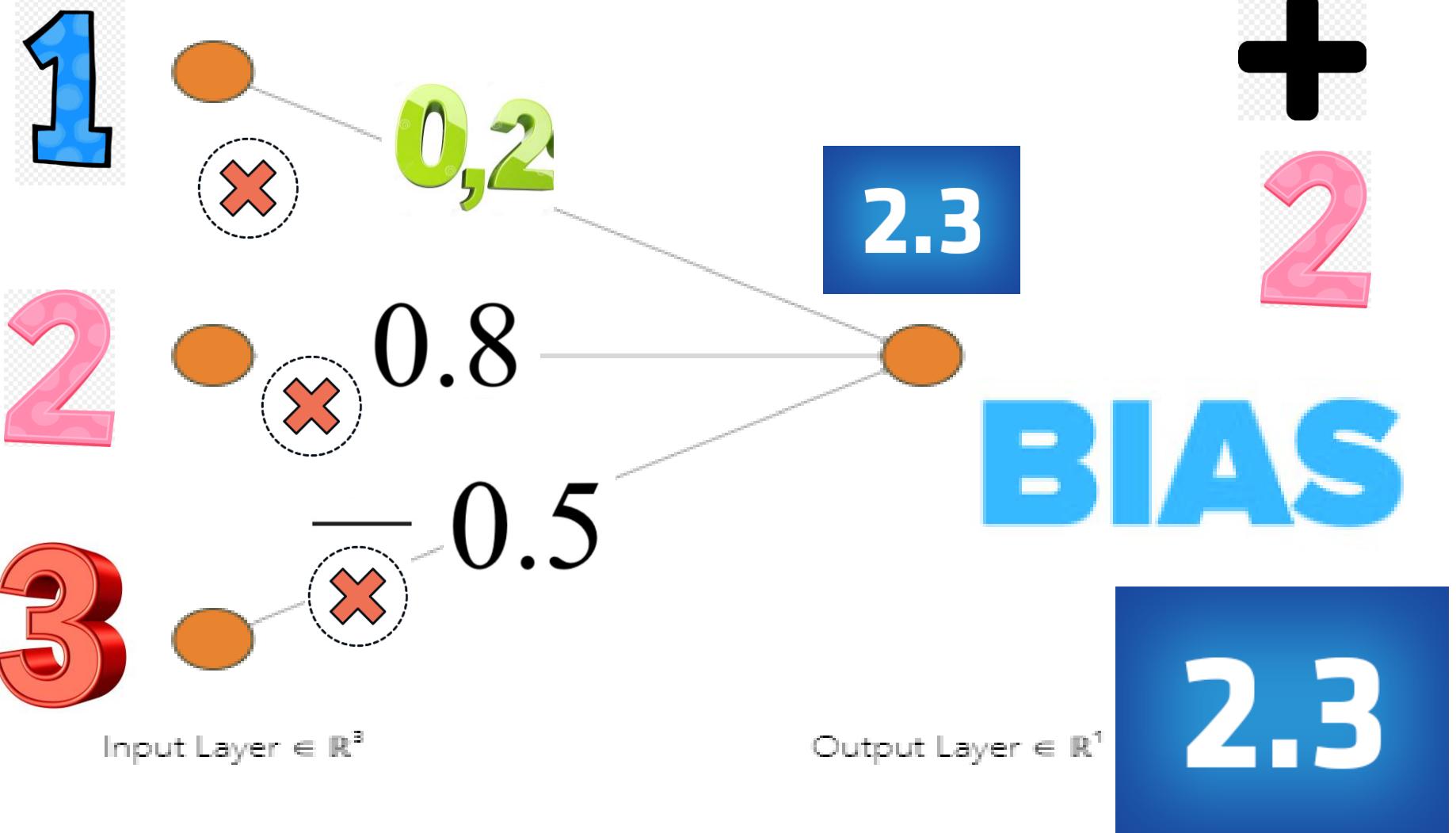
ANN

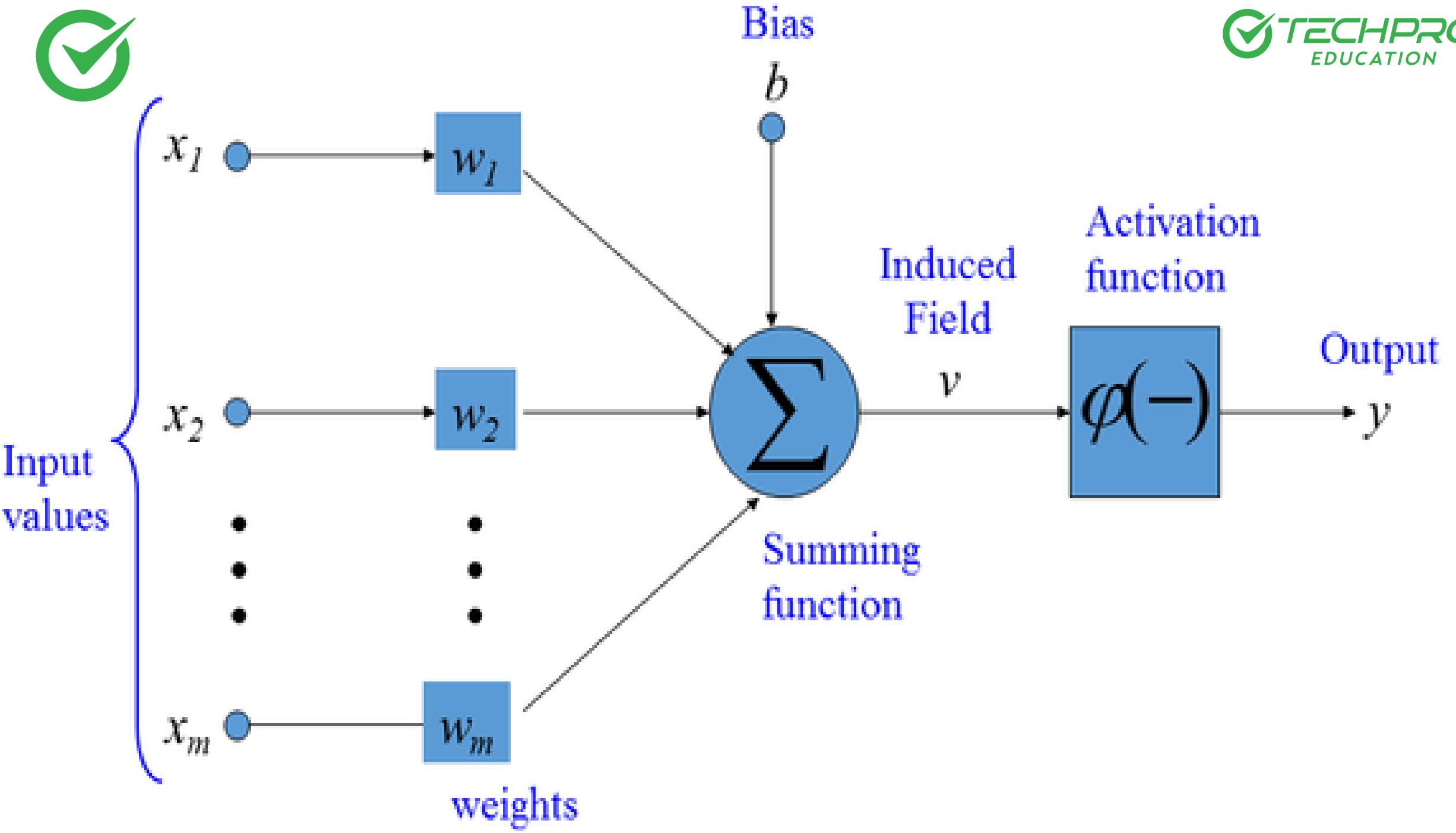


<https://nnfs.io>



0,2
1.6 = 1.5 = 0.3







Girdi
değerleri

x_1

w_1

x_2

w_2

⋮

⋮

x_m

w_m

ağırlıklar

Bias

b

Σ

Toplama
fonksiyonu

Yerel
Alan

v

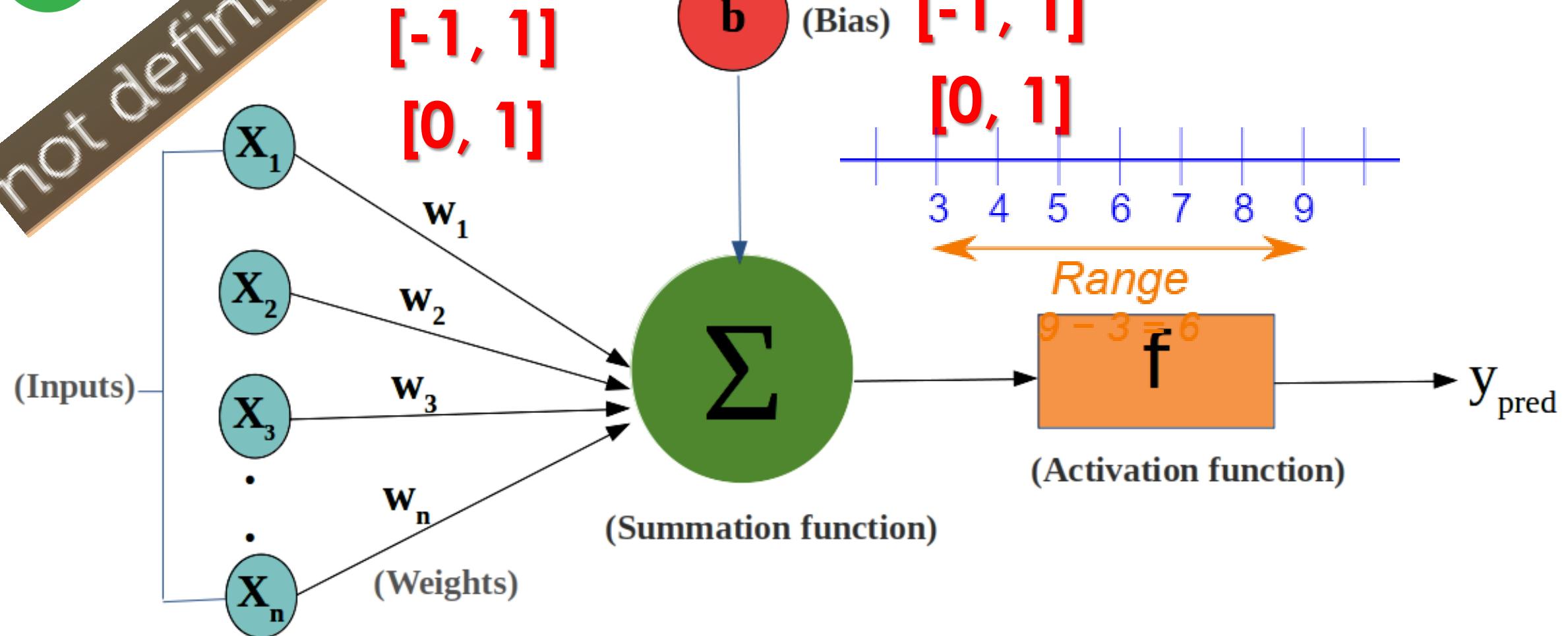
Aktivasyon
Fonksiyonu

$\varphi(-)$

Çıktı
 y



not definite





```
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], dtype=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=float32)>,
 <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)>]
```

Weights & Biases

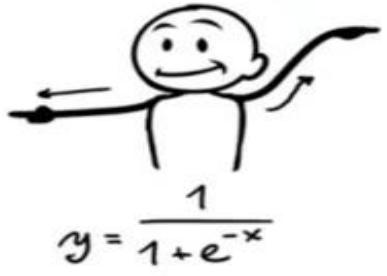


ACTIVATION FUNCTIONS

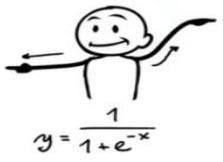


DANCE MOVES OF ACTIVATION FUNCTIONS

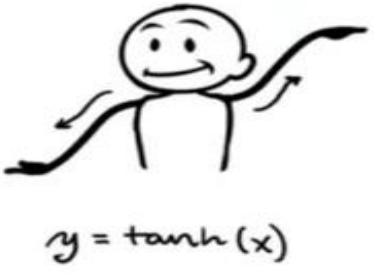
Sigmoid



Sigmoid

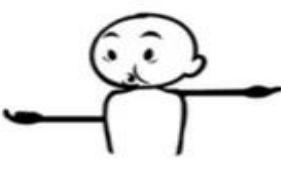


Tanh



$y = \tanh(x)$

Step Function



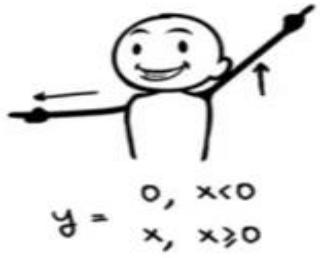
$$y = \begin{cases} 0, & x < n \\ 1, & x \geq n \end{cases}$$

Softplus



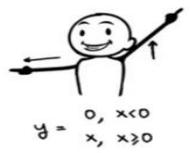
$$y = \ln(1+e^x)$$

ReLU



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

ReLU



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

Swish



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

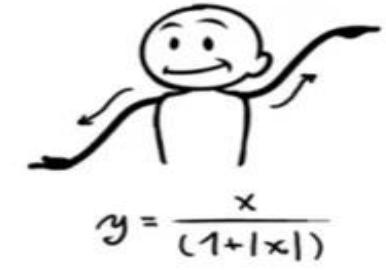
Swish



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

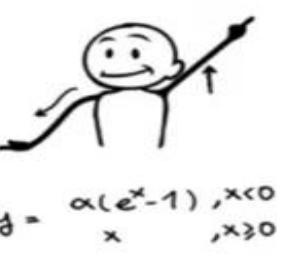
source: sefiks

Softsign



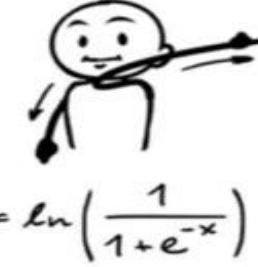
$$y = \frac{x}{(1+|x|)}$$

ELU



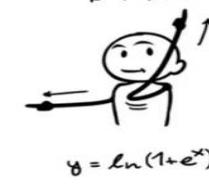
$$y = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

Log of Sigmoid



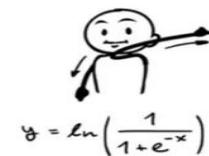
$$y = \ln\left(\frac{1}{1+e^{-x}}\right)$$

Softplus



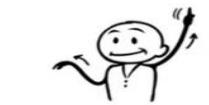
$$y = \ln(1+e^x)$$

Log of Sigmoid



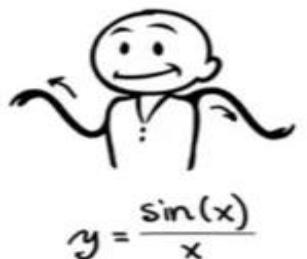
$$y = \ln\left(\frac{1}{1+e^{-x}}\right)$$

Mish



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

Sinc



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

Leaky ReLU



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

Mish

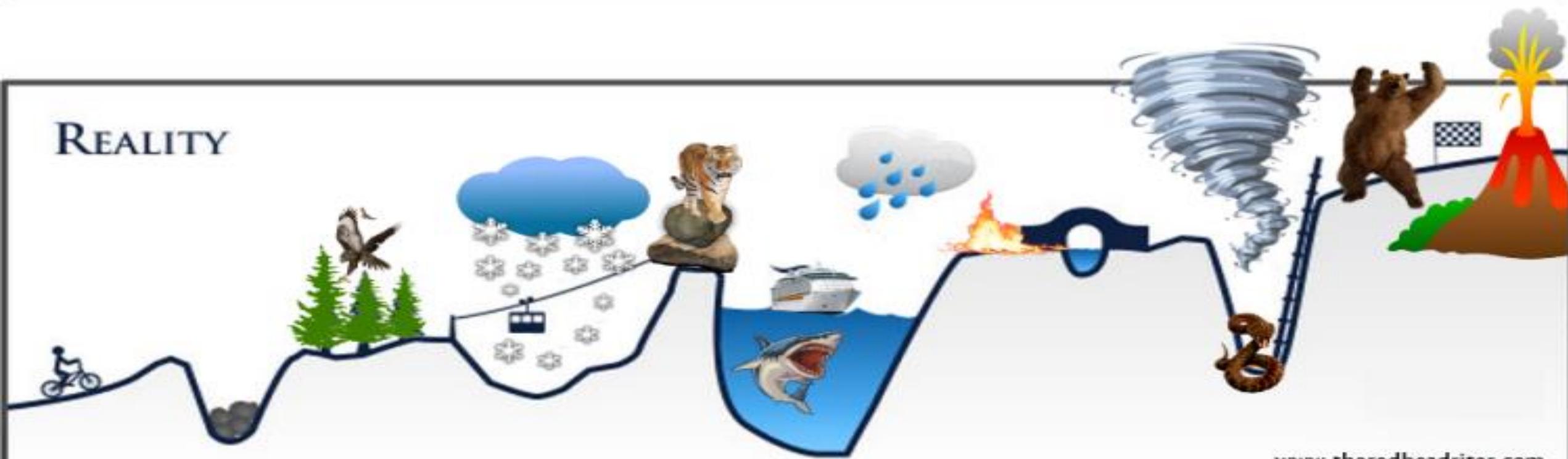


$$y = x(\tanh(\text{softplus}(x)))$$

YOUR PLAN

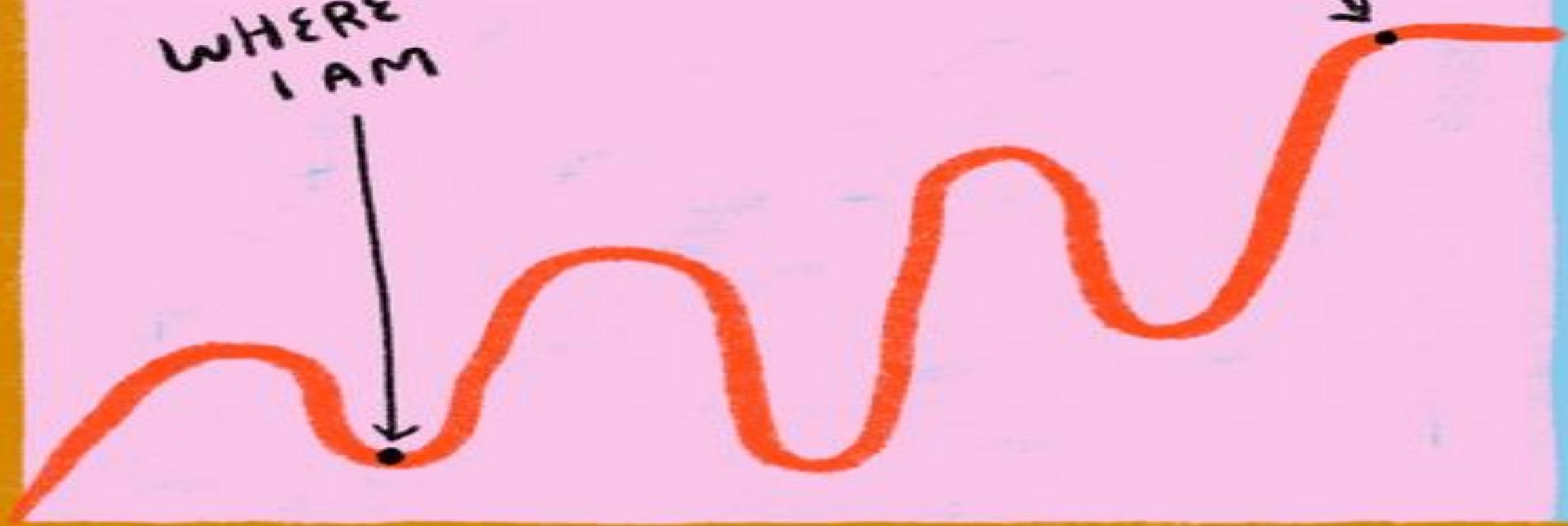


REALITY



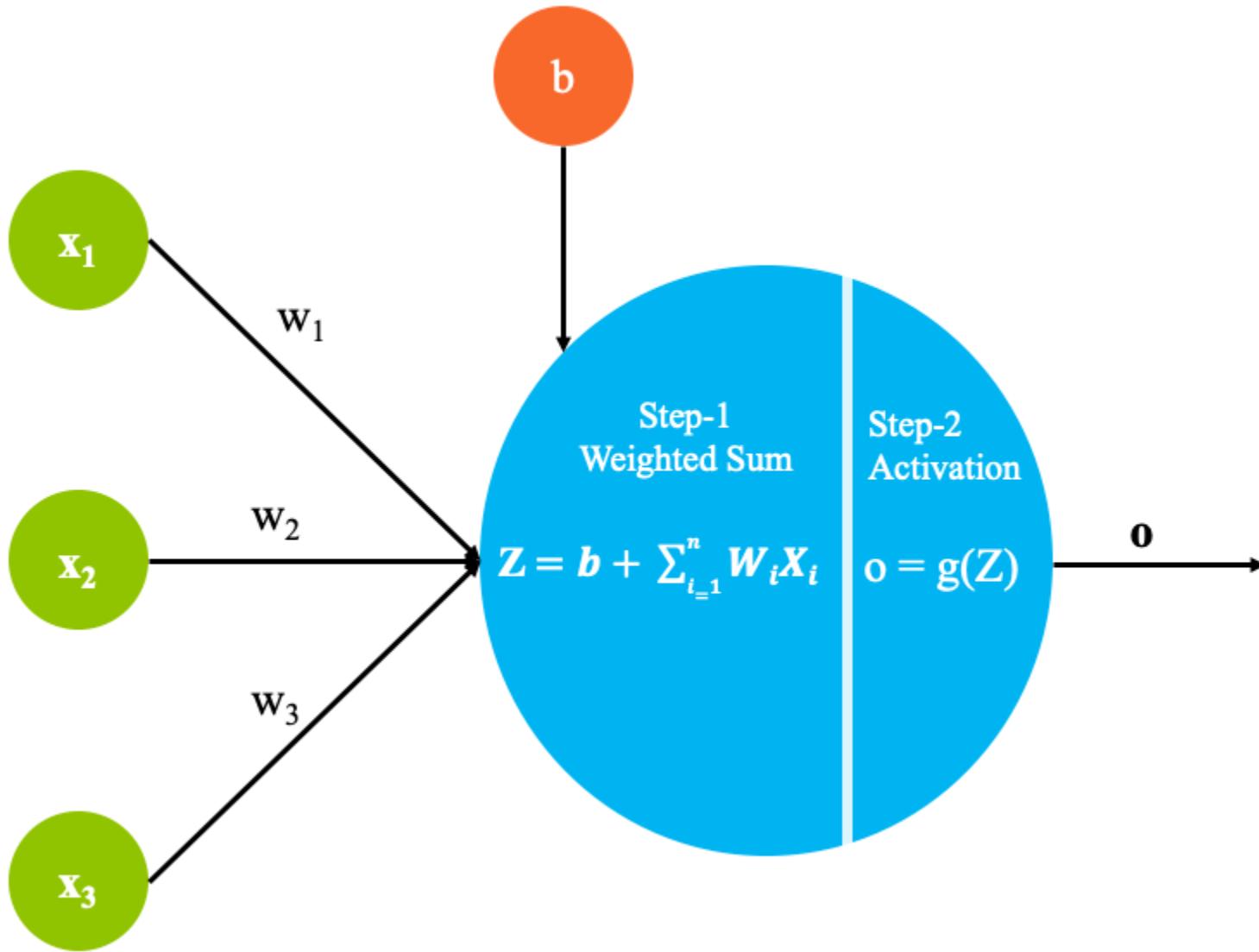
WHERE
I AM

WHERE
I WANT TO BE



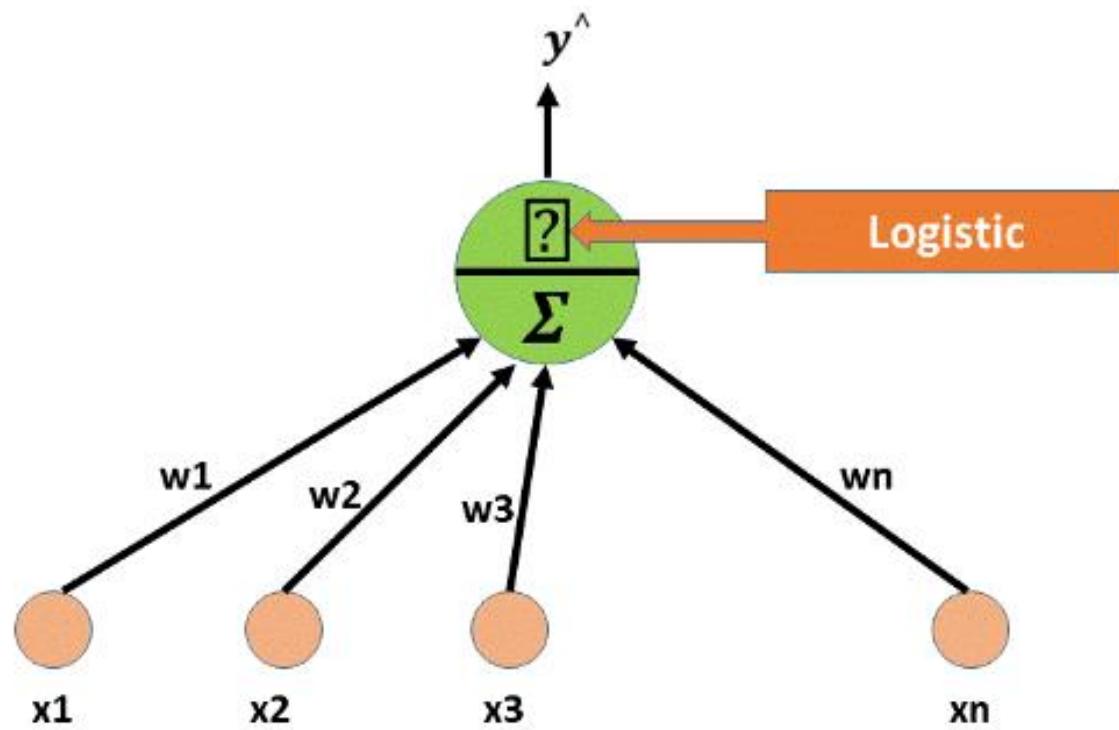
PERSONAL RECOVERY IS NOT
A LINEAR PROCESS

ACTIVATION FUNCTIONS



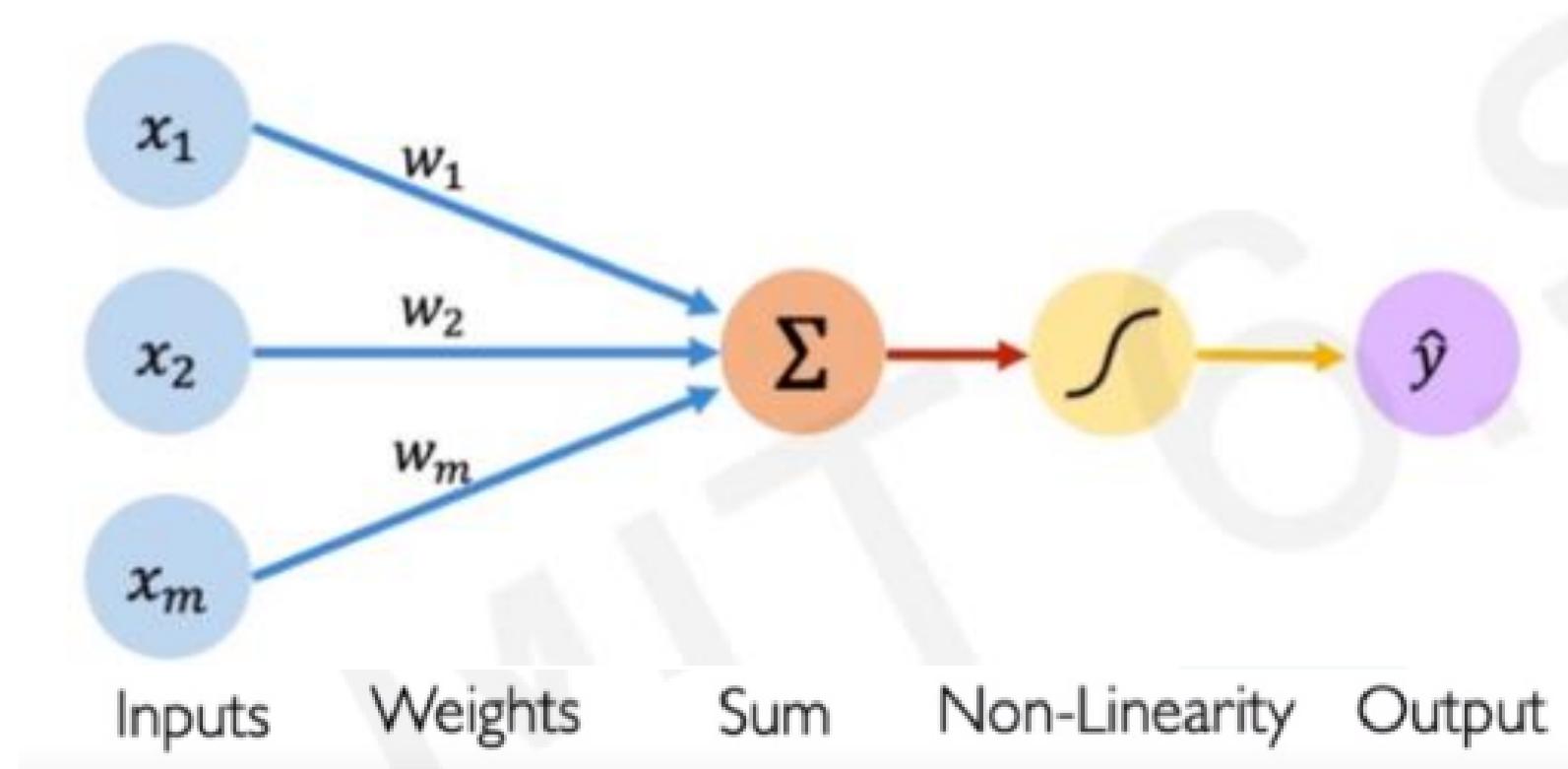


ACTIVATION FUNCTIONS

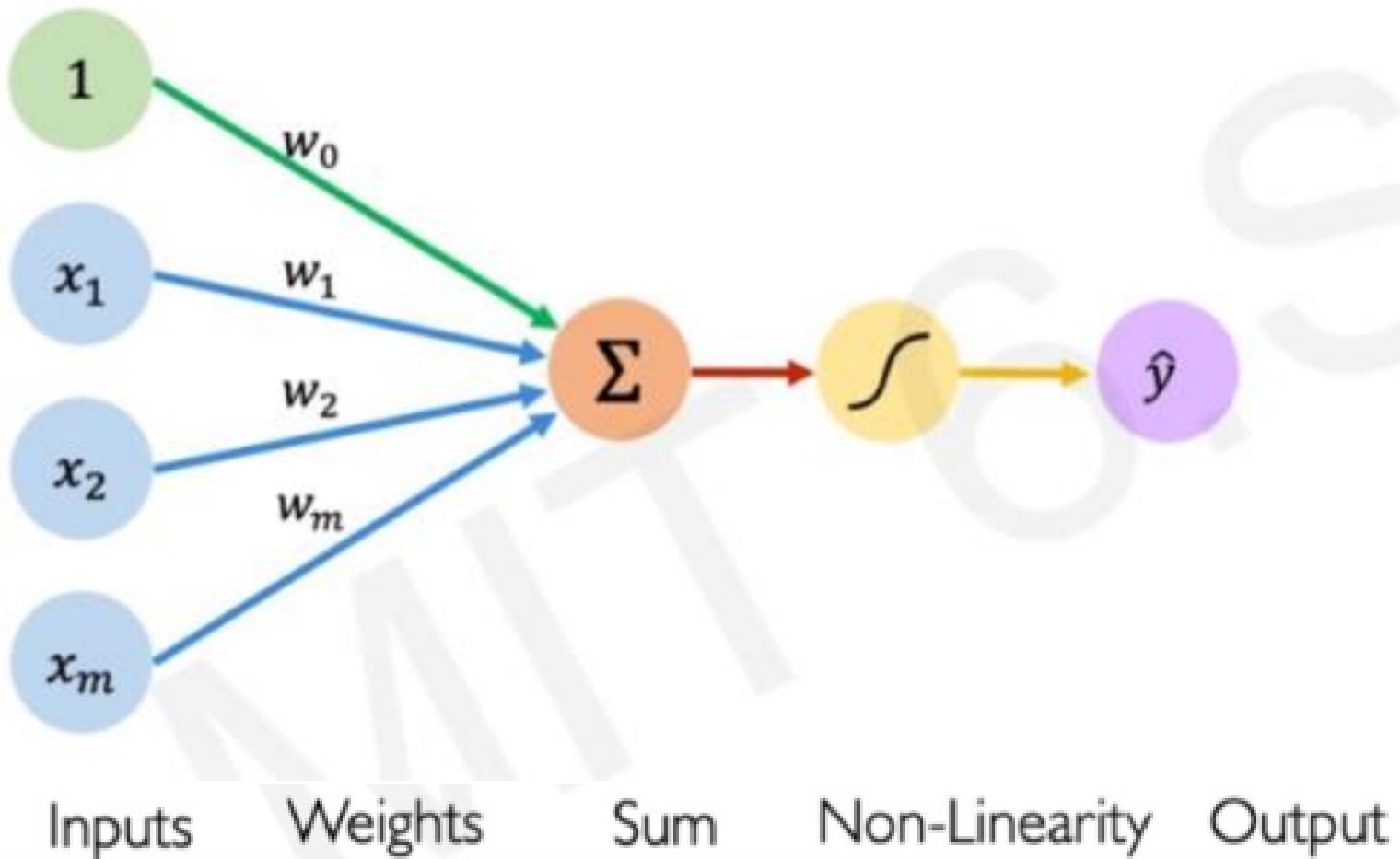




ACTIVATION FUNCTIONS



ACTIVATION FUNCTIONS



Linear combination of inputs

Output

$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$

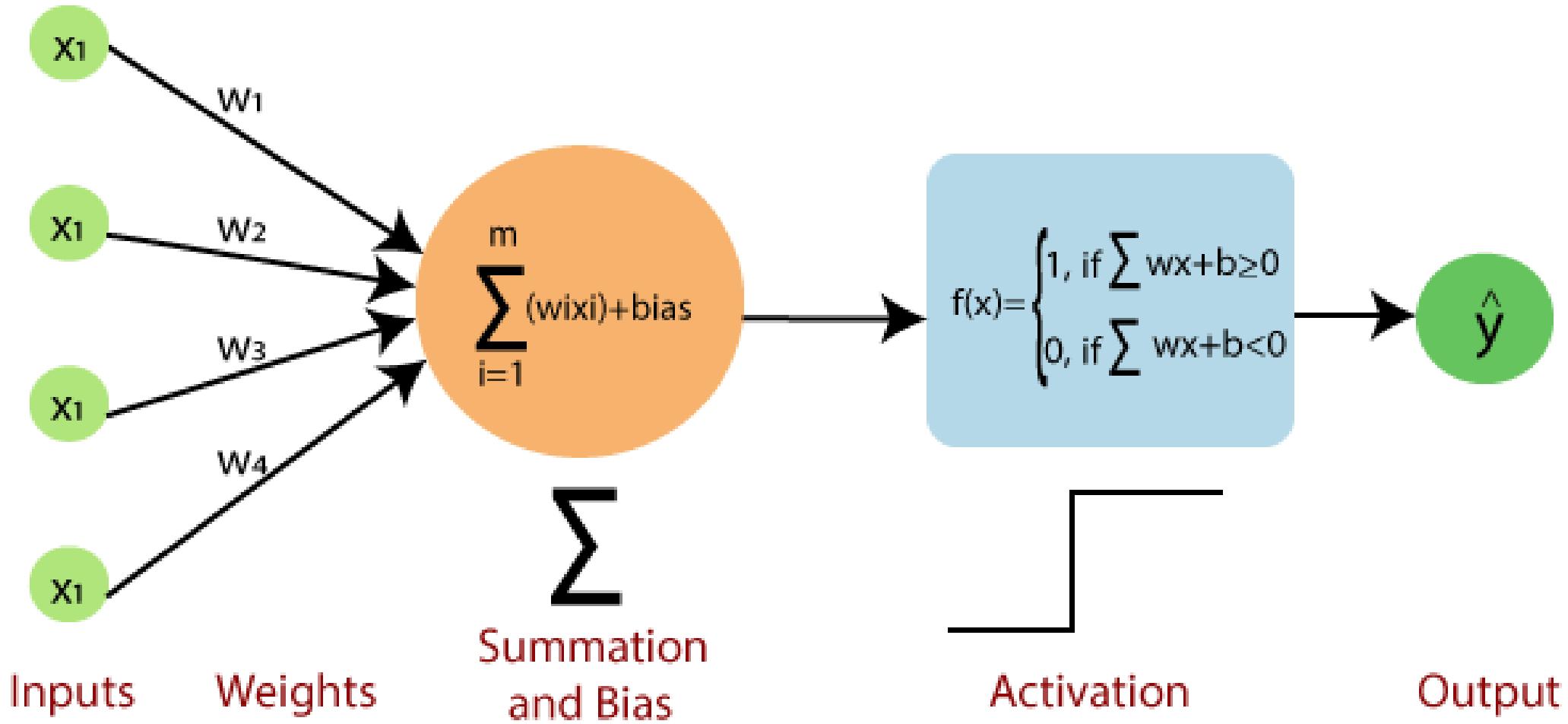
Non-linear activation function

Bias

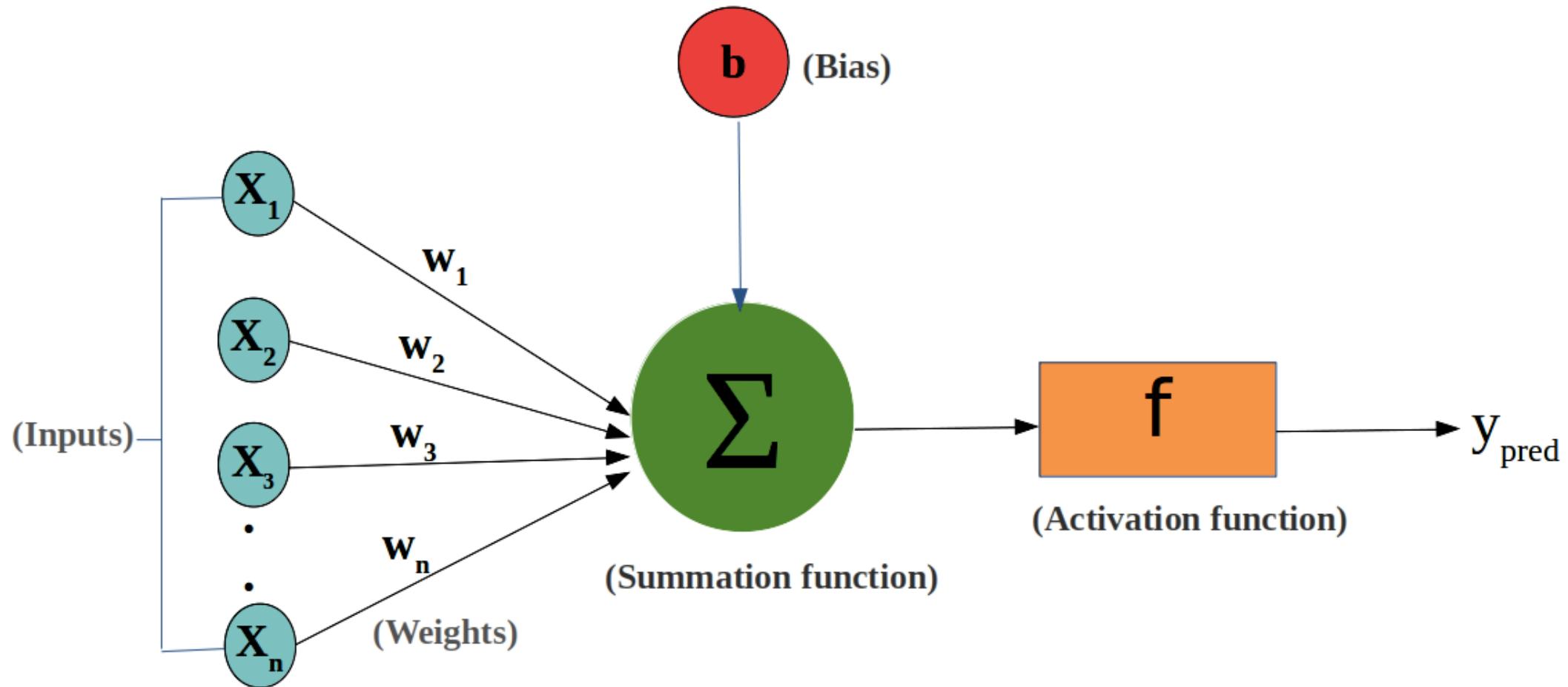
$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

where: $\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

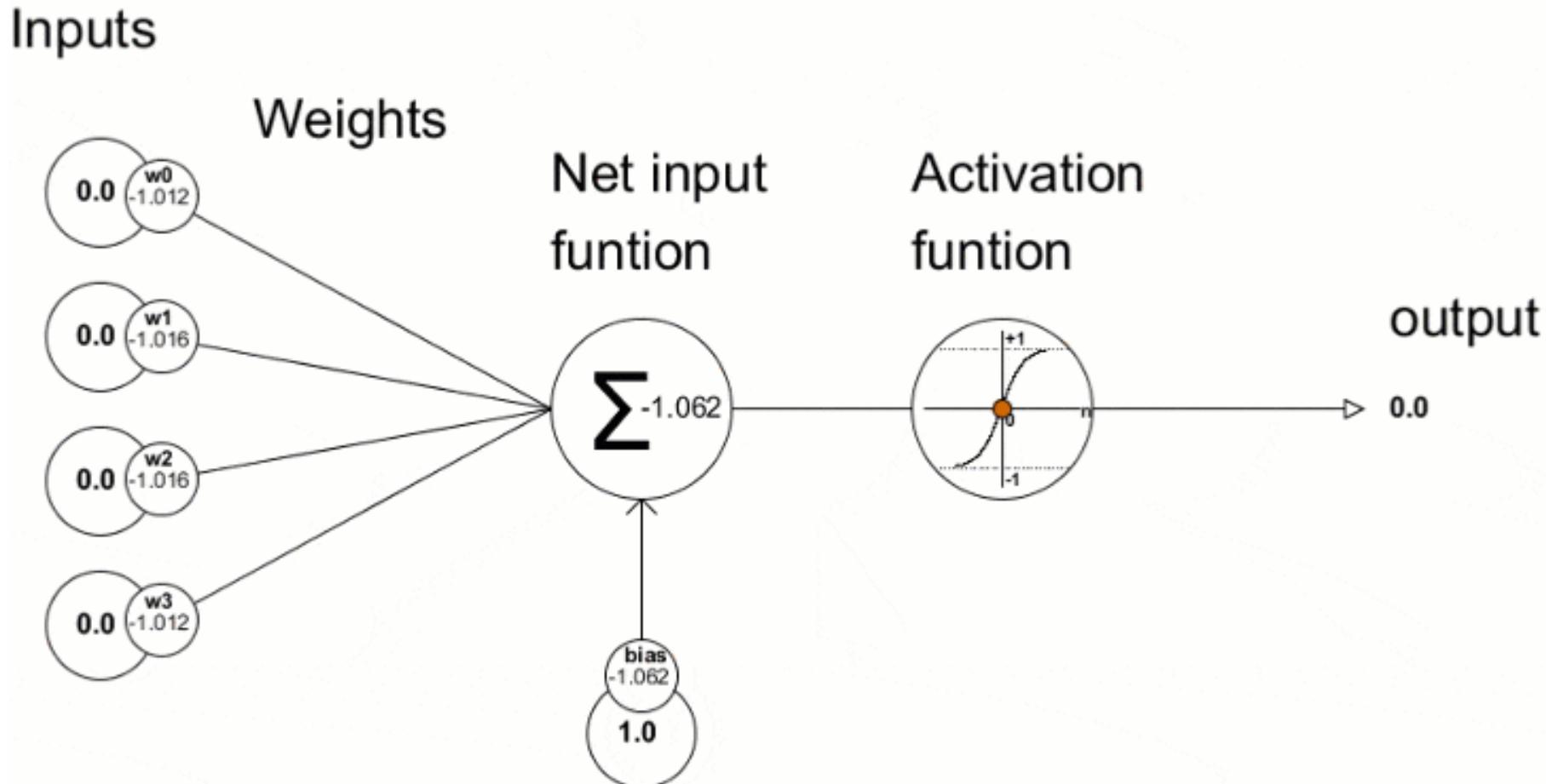
ACTIVATION FUNCTIONS



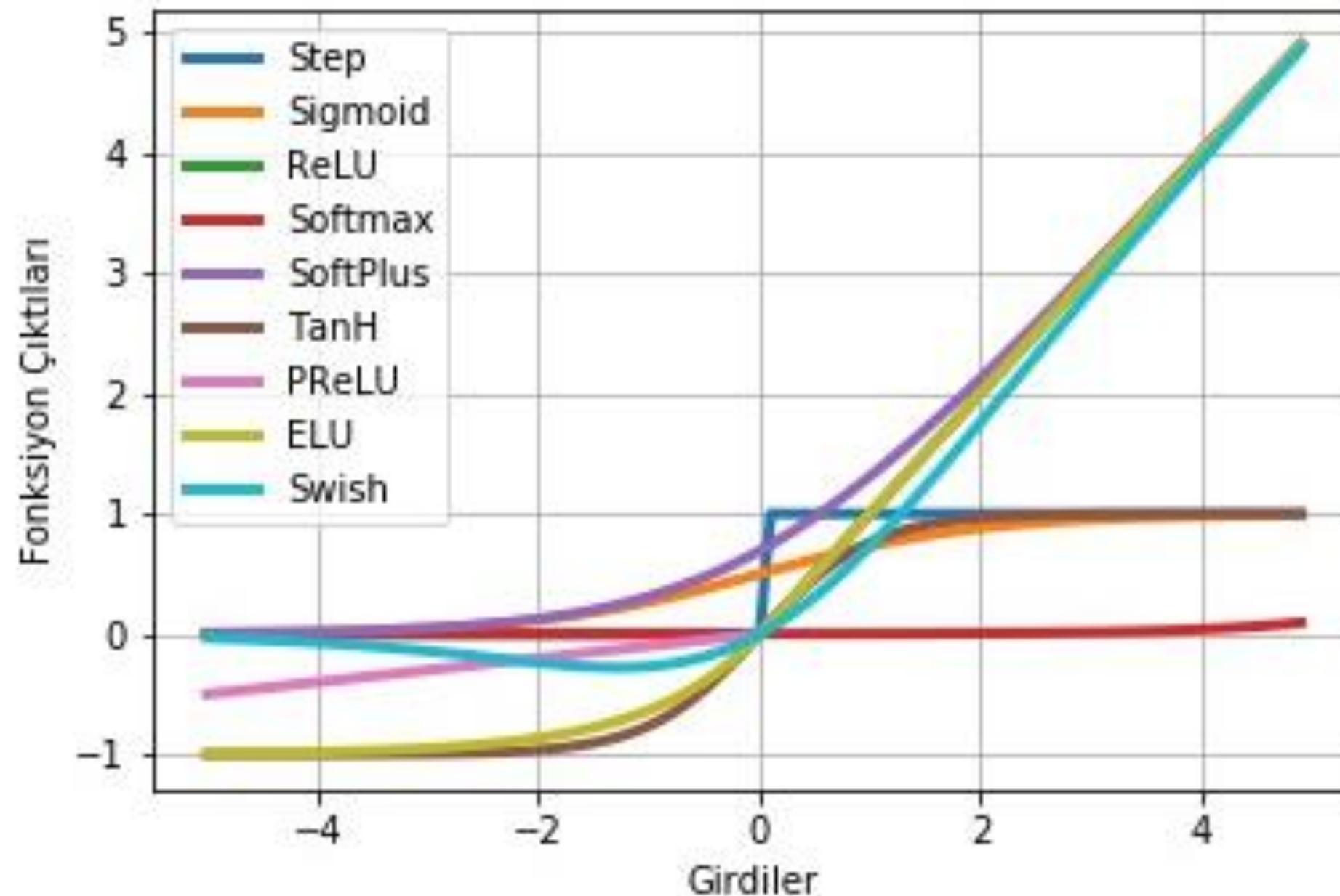
ACTIVATION FUNCTIONS



ACTIVATION FUNCTIONS



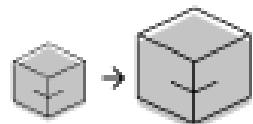
ACTIVATION FUNCTIONS



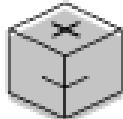


ACTIVATION FUNCTIONS

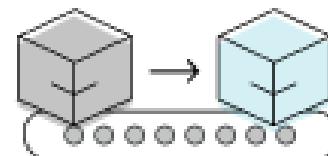
HOW IT WORKS?



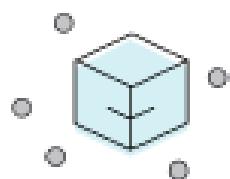
Take input and multiply by the neuron's weight.



Add bias*



Feed the result, x , to the activation function: $f(x)$



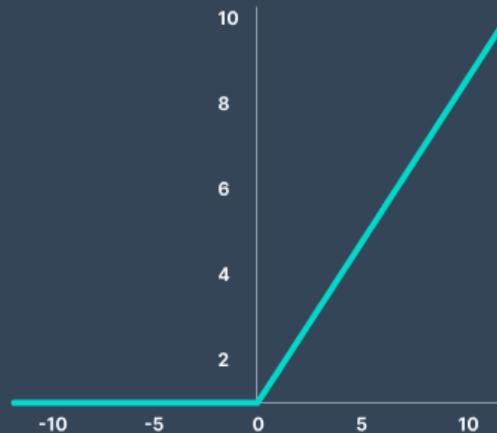
Take the output and transmit to the next layer of neurons.

* Modern neural network models use non-linear activation functions

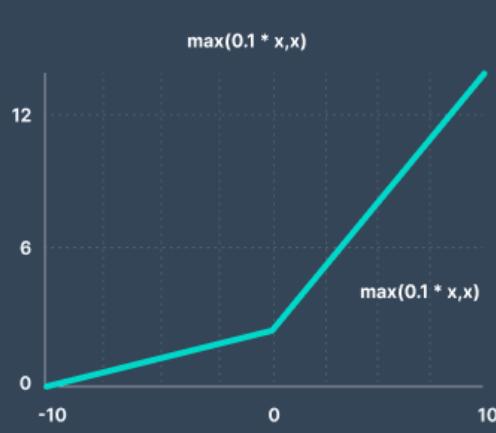
ACTIVATION FUNCTIONS



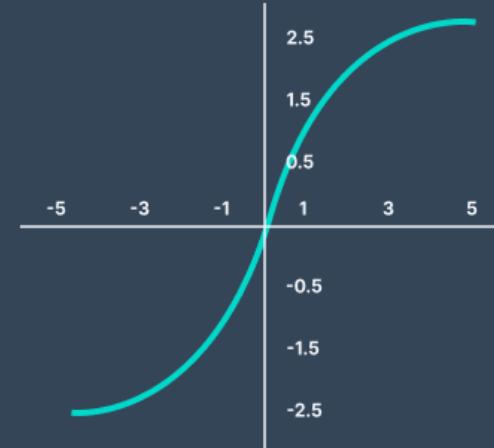
ReLU



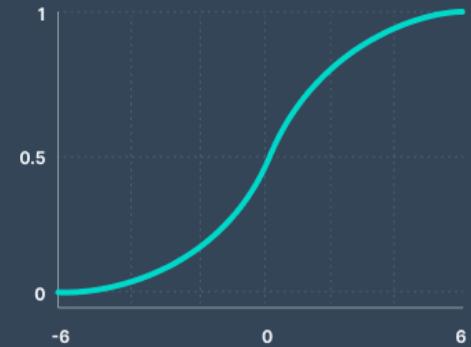
Leaky ReLU



Tanh



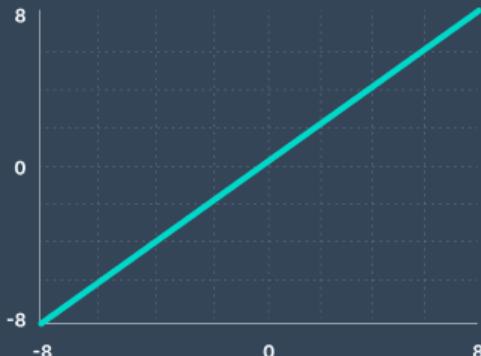
Sigmoid / Logistic



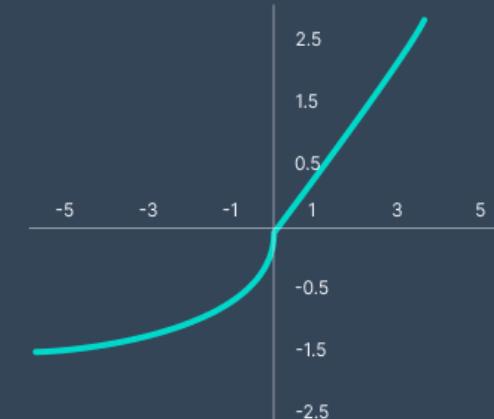
Binary Step Function



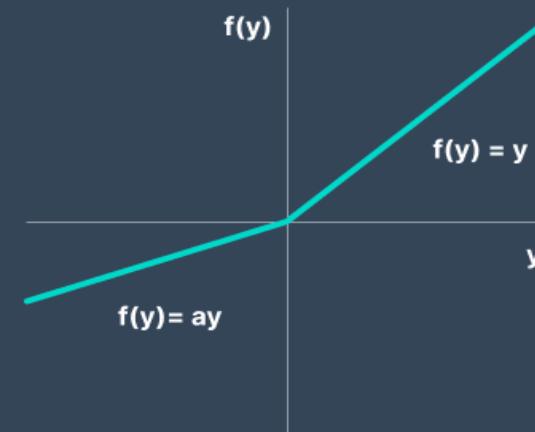
Linear



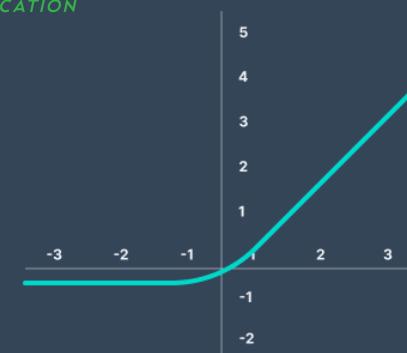
SELU



Parametric ReLU



ELU





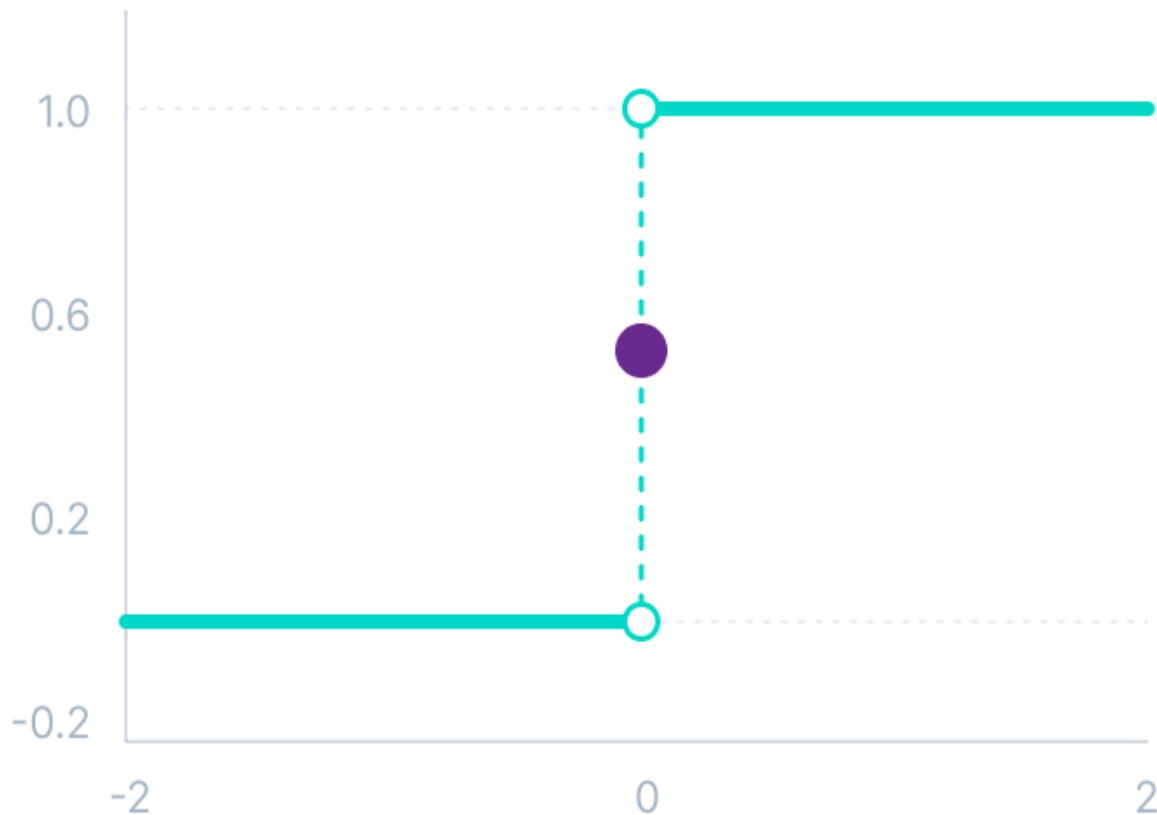
BINARY STEP ACTIVATION FUNCTION

TECHPRO
EDUCATION

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.

Binary Step Function





BINARY STEP ACTIVATION FUNCTION

BINARY CLASSIFICATION



X



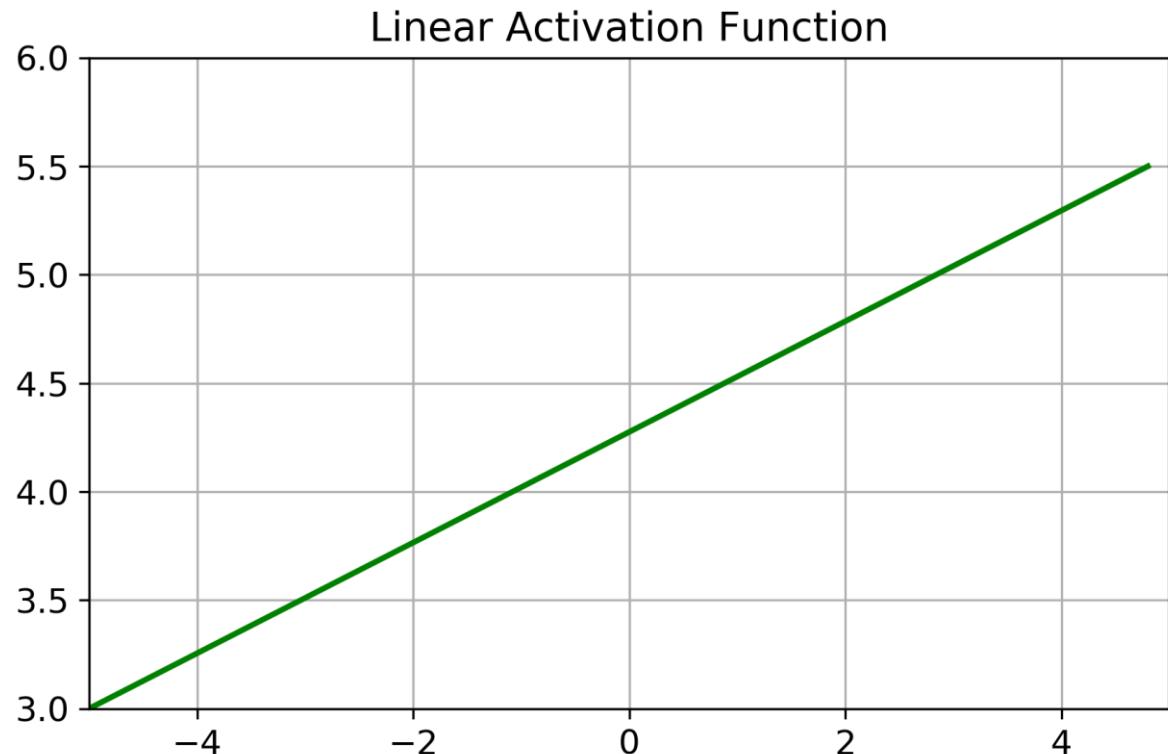
Y

1 : CAT
0 : NOT CAT



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.

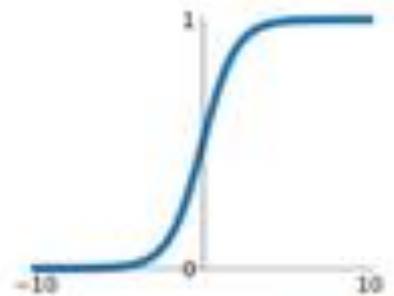




NON-LINEAR ACTIVATION FUNCTIONS

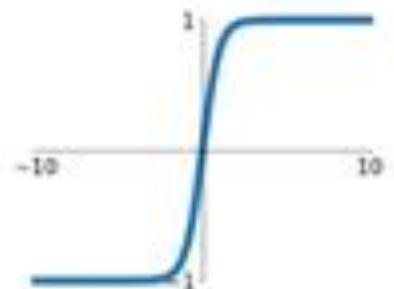
Sigmoid

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



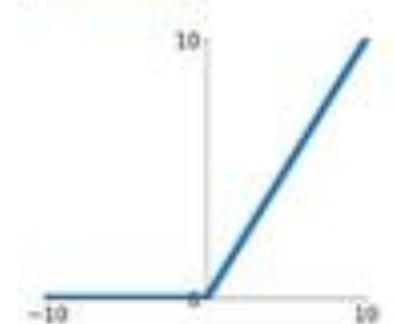
tanh

$$\tanh(x)$$

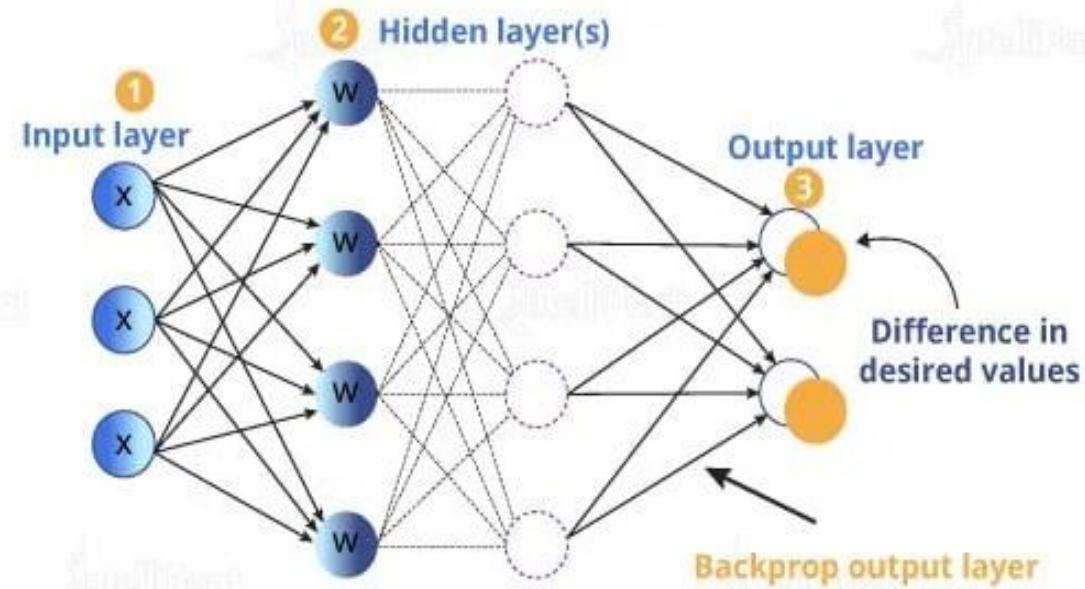
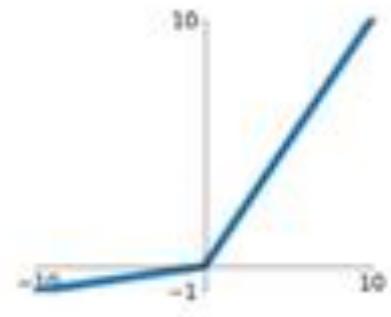


ReLU

$$\max(0, x)$$



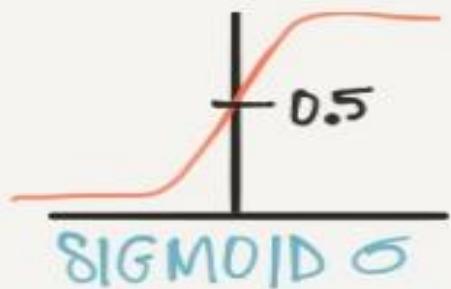
Leaky ReLU
 $\max(0.1x, x)$





NON-LINEAR ACTIVATION FUNCTIONS

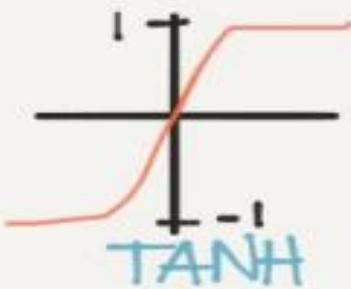
ACTIVATION FUNCTIONS



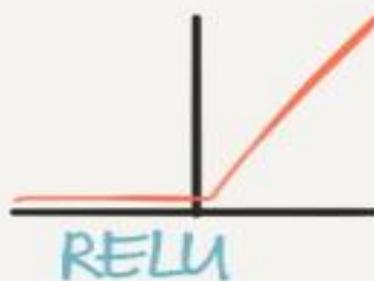
BINARY CLASSIFIER

- ONLY USED FOR
OUTPUT LAYER

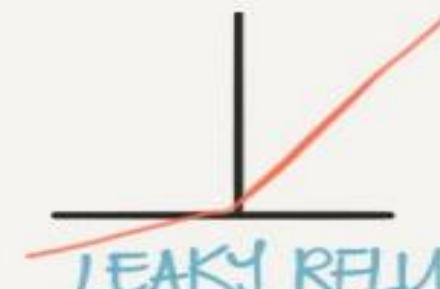
SLOW GRAD
DESCENT SINCE
SLOPE IS SMALL
FOR LARGE/SMALL VAL



NORMALIZED
 \Rightarrow GRADIENT
DESCENT IS
FASTER



DEFAULT
CHOICE FOR
ACTIVATION
SLOPE = 1/0

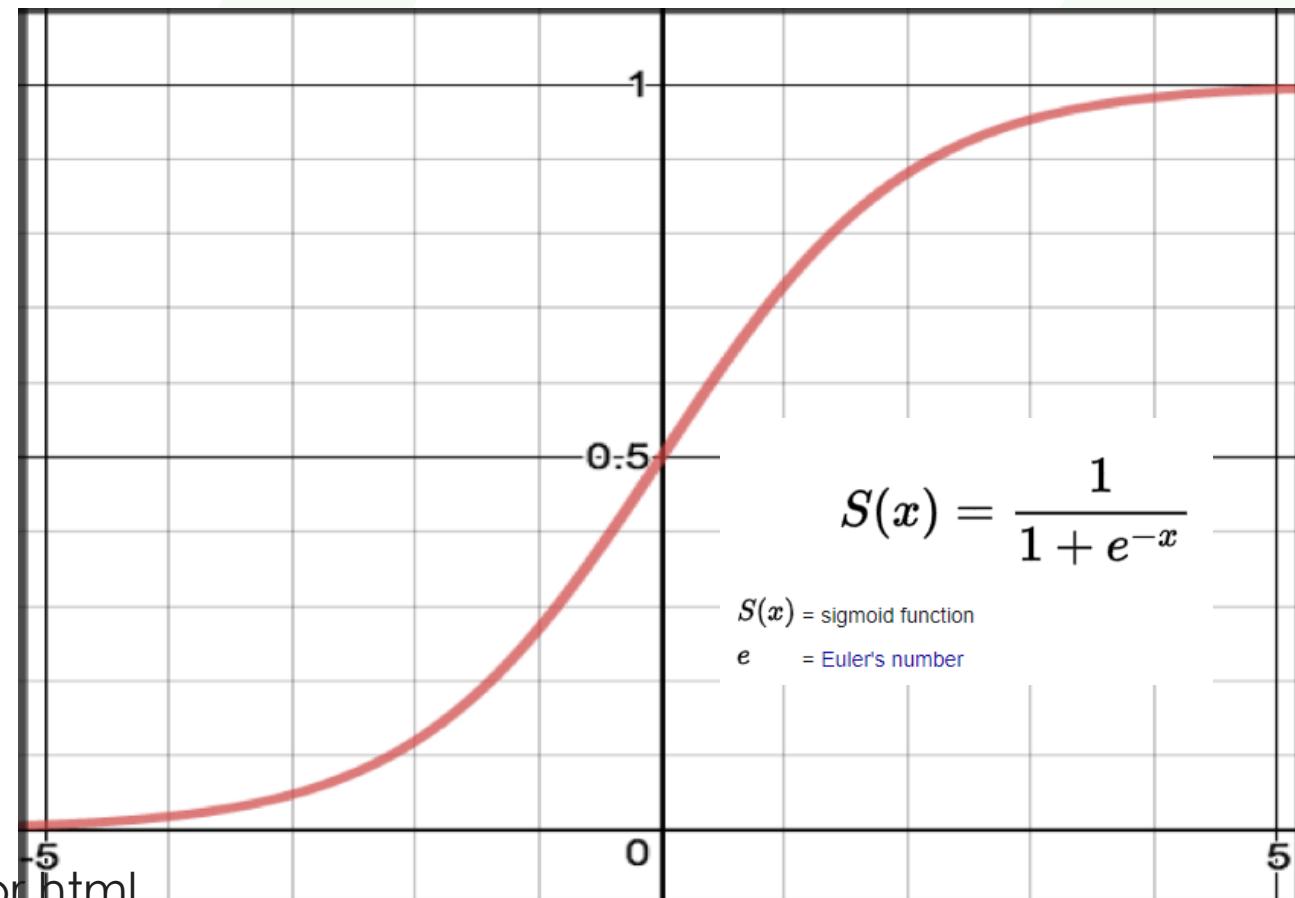


AVOIDS UNDEF
SLOPE AT 0
BUT RARELY
USED IN PRACTICE



SIGMOID ACTIVATION FUNCTION

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 $(-\infty, \infty)$ to the range in **(0,1)**.



Sigmoid Calculator
sigmoid = $1 / (1 + e^{-x})$

x:
Sigmoid
S(X) 0.8190612068479367
S'(X) 0.14819994628473815 sigmoid prime(X)

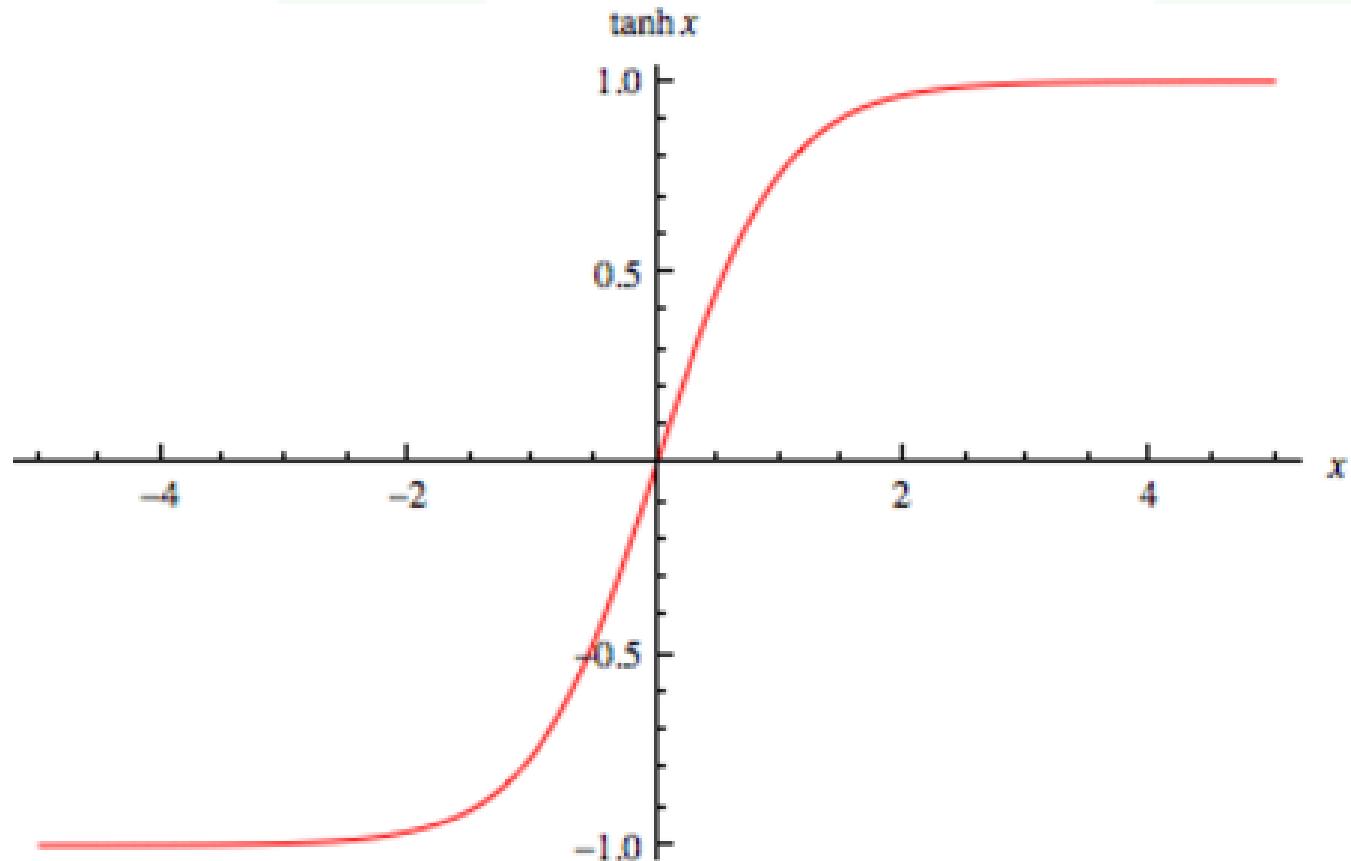
ReLU Calculator
ReLU = $\max(X, 0)$

x:
ReLU
S(X)
S'(X) ReLU prime(X)



TanH ACTIVATION FUNCTION

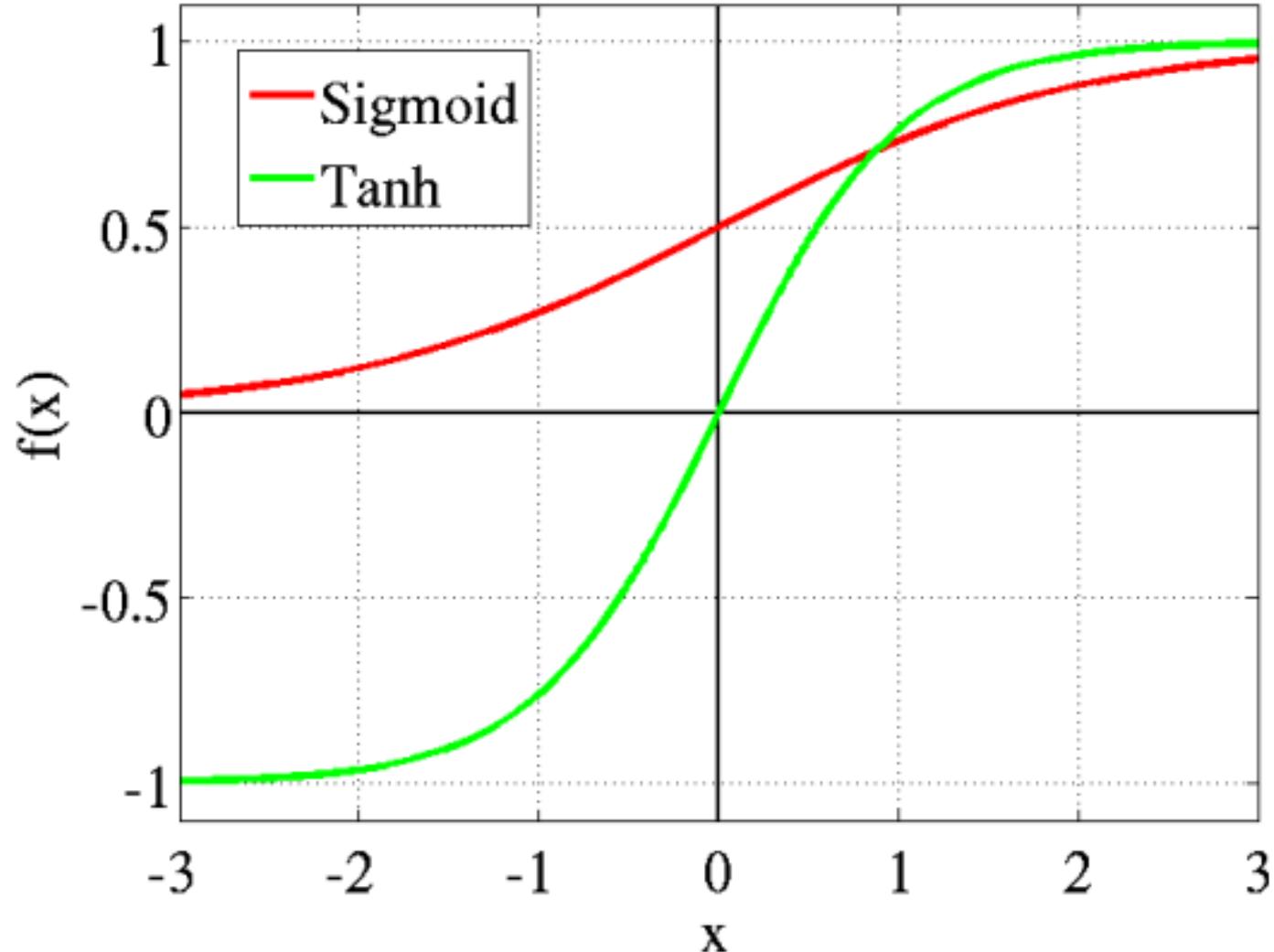
It is very similar to the sigmoid activation function and even has the same S-shape. The function takes any real value as input and outputs values in the range **-1 to 1.**





TanH vs Sigmoid ACTIVATION FUNCTION

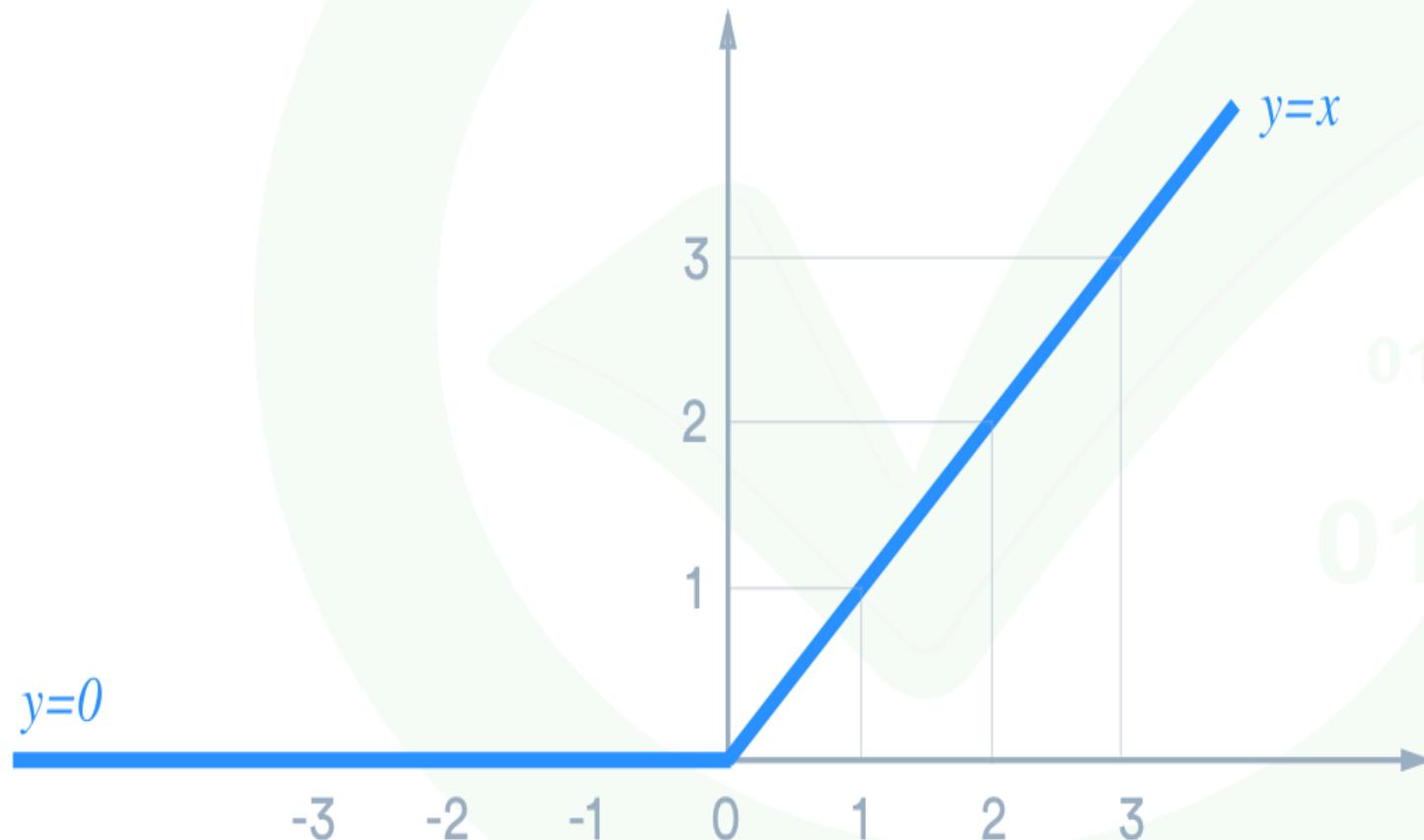
Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s-shaped).





ReLU ACTIVATION FUNCTION

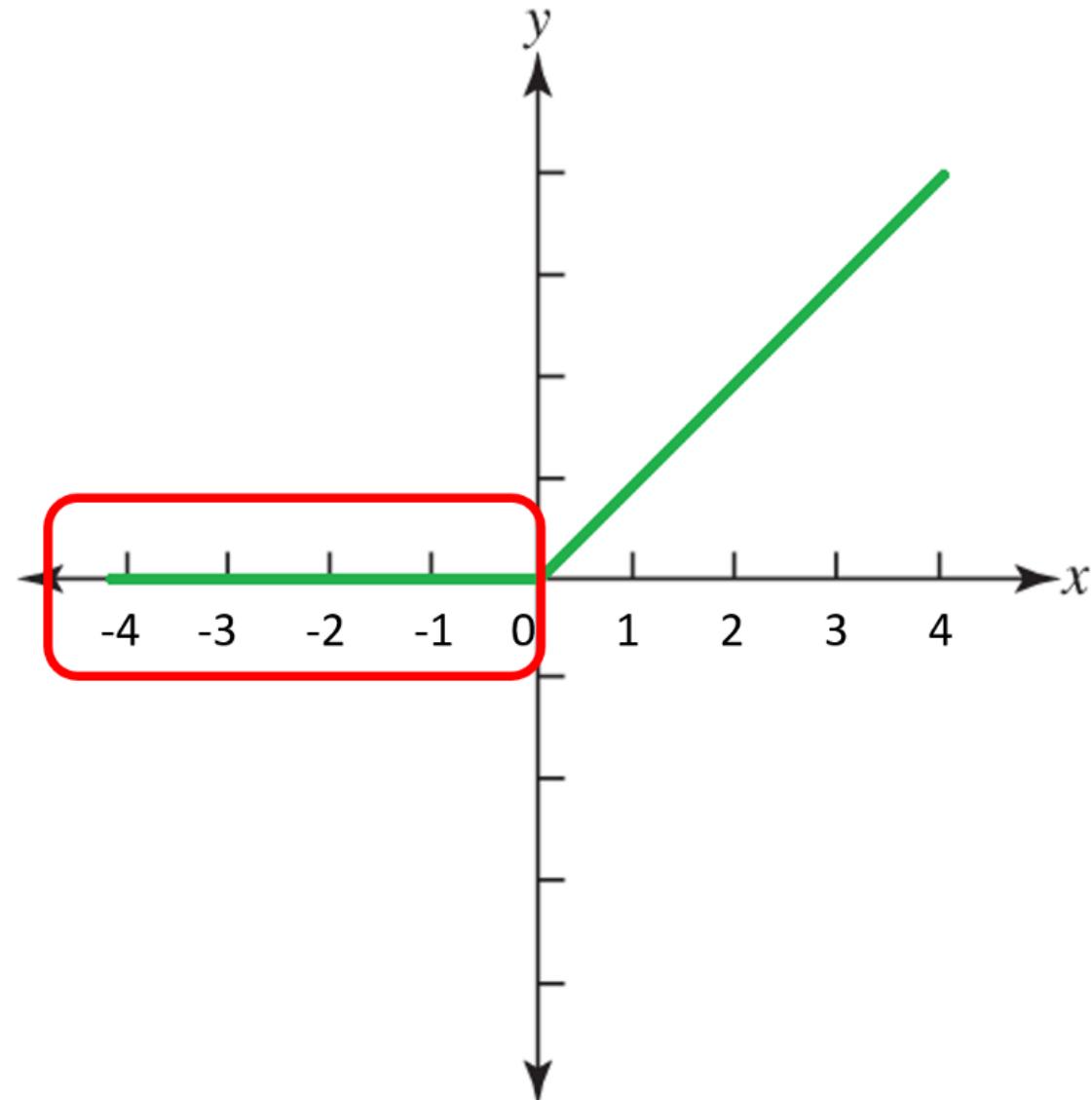
The ReLU (Rectified Linear Unit) is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.





DYING ReLU Problem

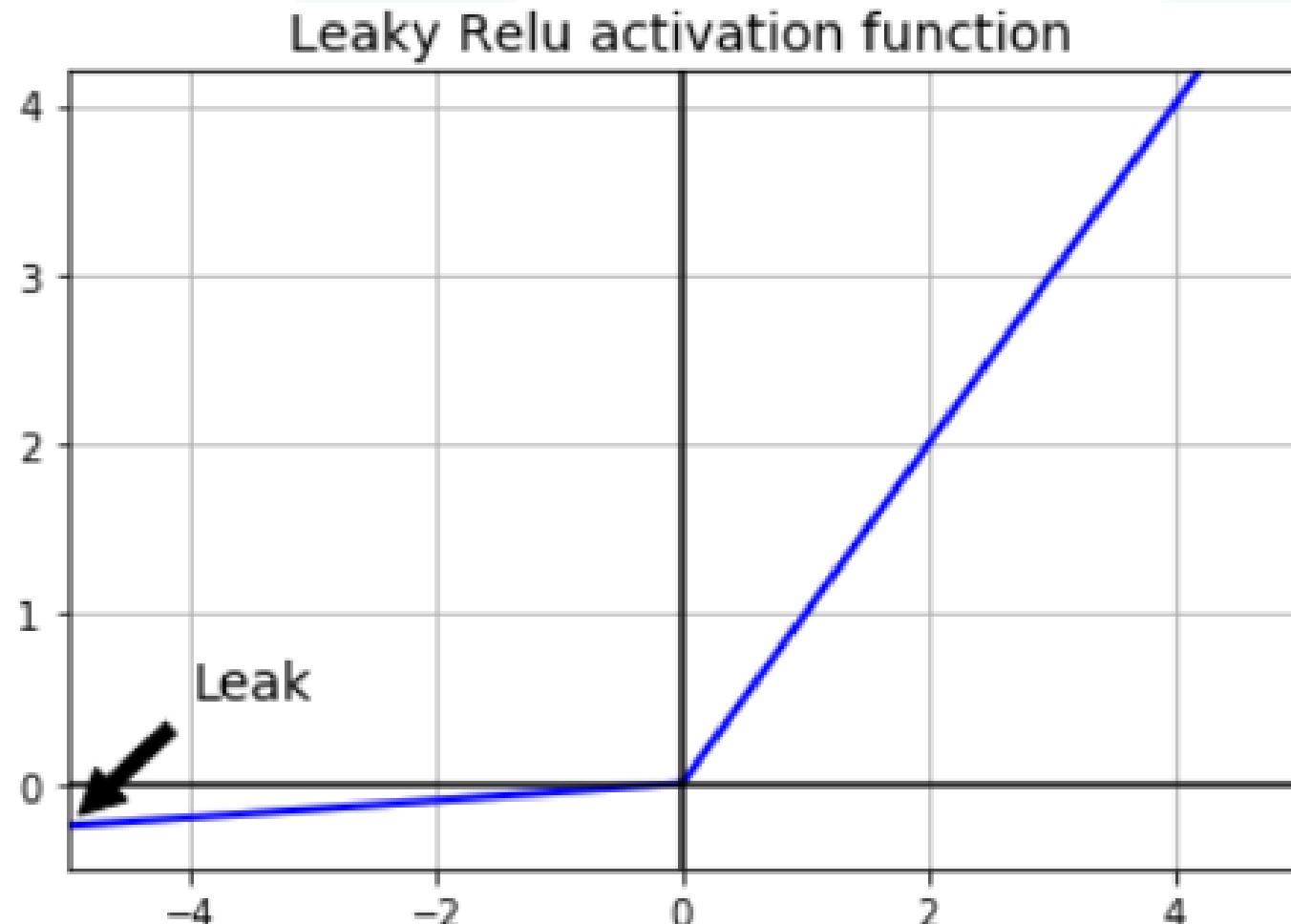
The dying ReLU refers to the problem when ReLU neurons become inactive and only output 0 for any input.





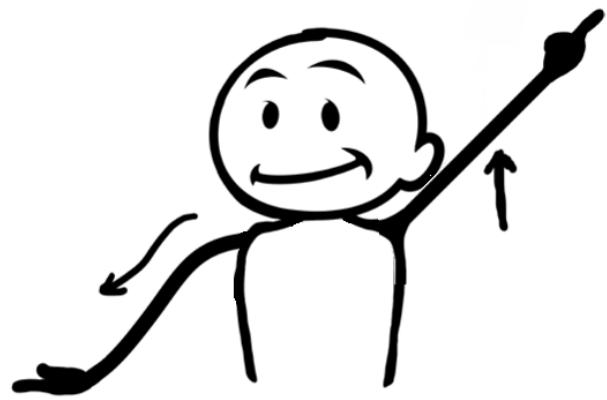
Leaky ReLU ACTIVATION FUNCTION

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

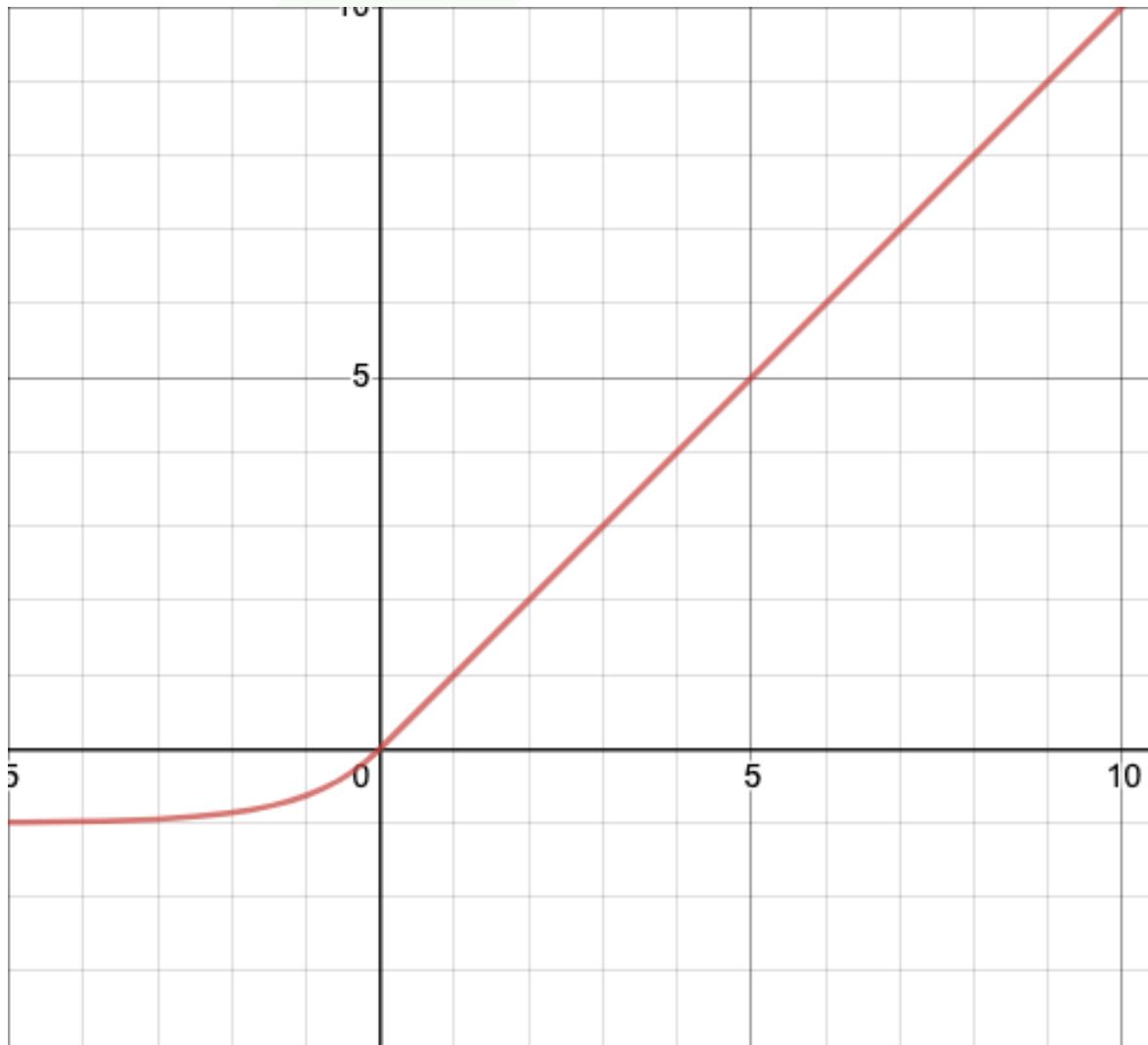




ELU ACTIVATION FUNCTION



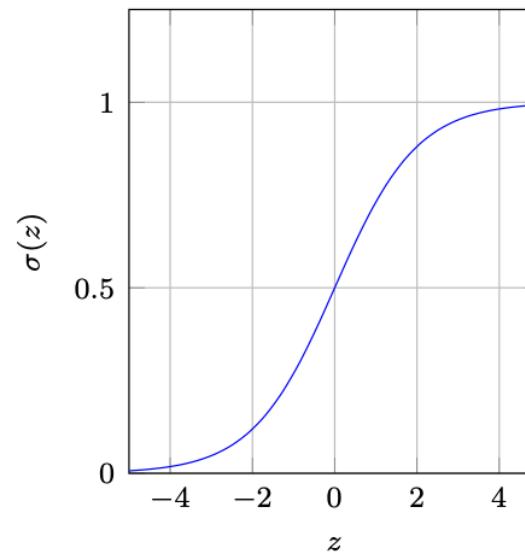
$$y = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$



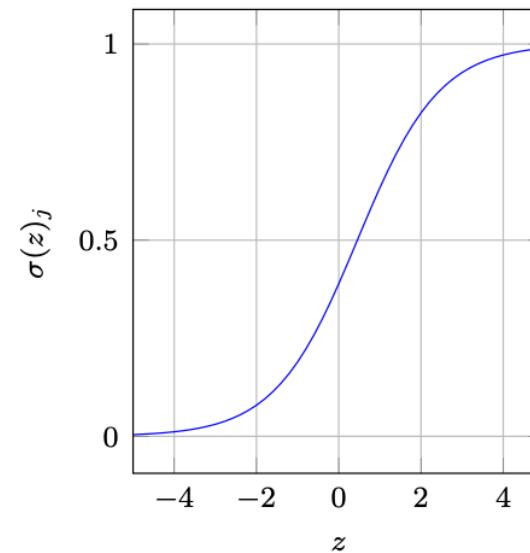


SOFTMAX ACTIVATION FUNCTION

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



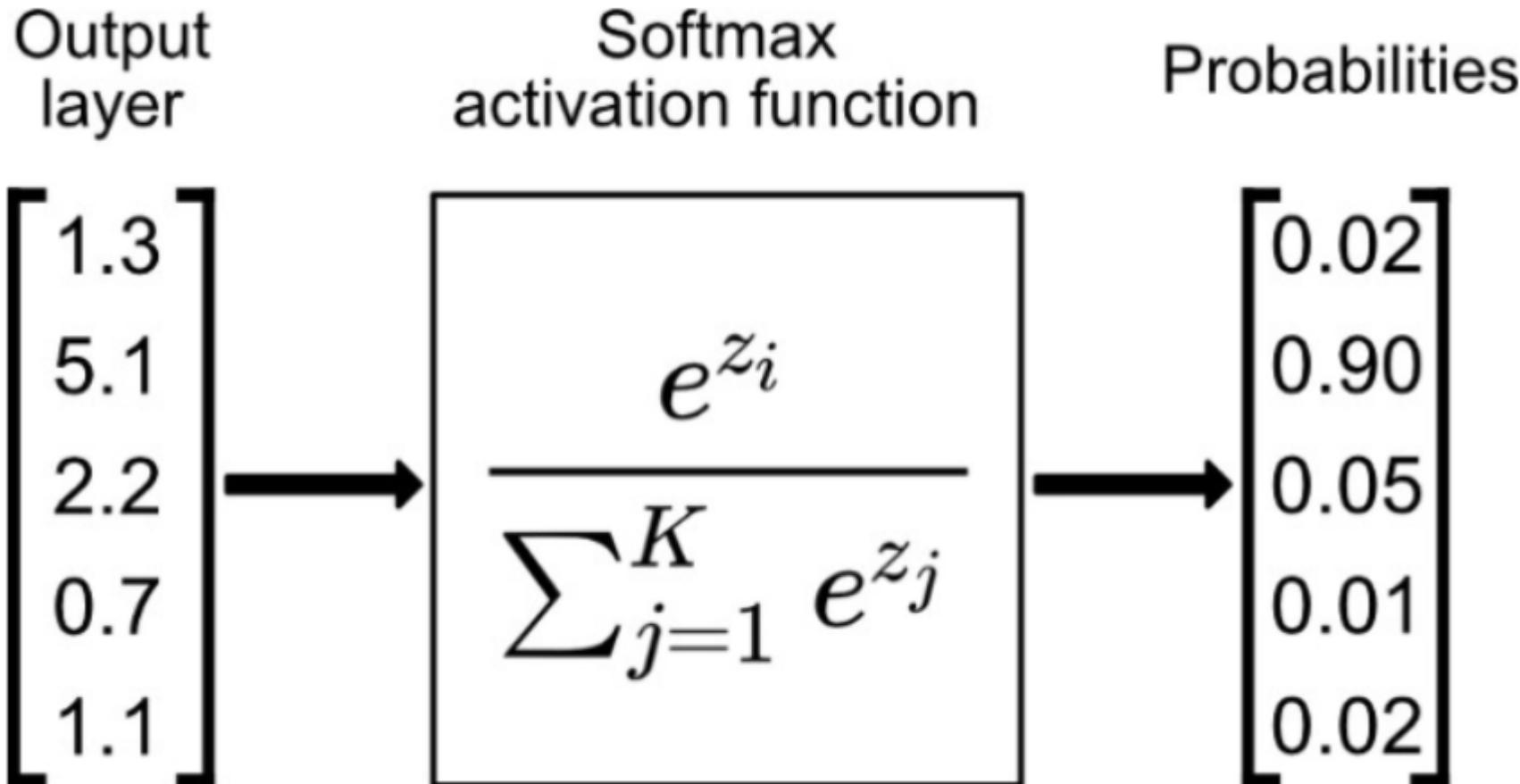
(a) Sigmoid activation function.



(b) Softmax activation function.



SOFTMAX ACTIVATION FUNCTION





ACTIVATION FUNCTIONS

| AKTİVASYON FONKSIYONU | 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 | $f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 & \text{için } f(x) = x \\ \beta \rightarrow \infty & \text{için } f(x) = 2\max(0, x) \end{cases}$ | $(-\infty, \infty)$ |

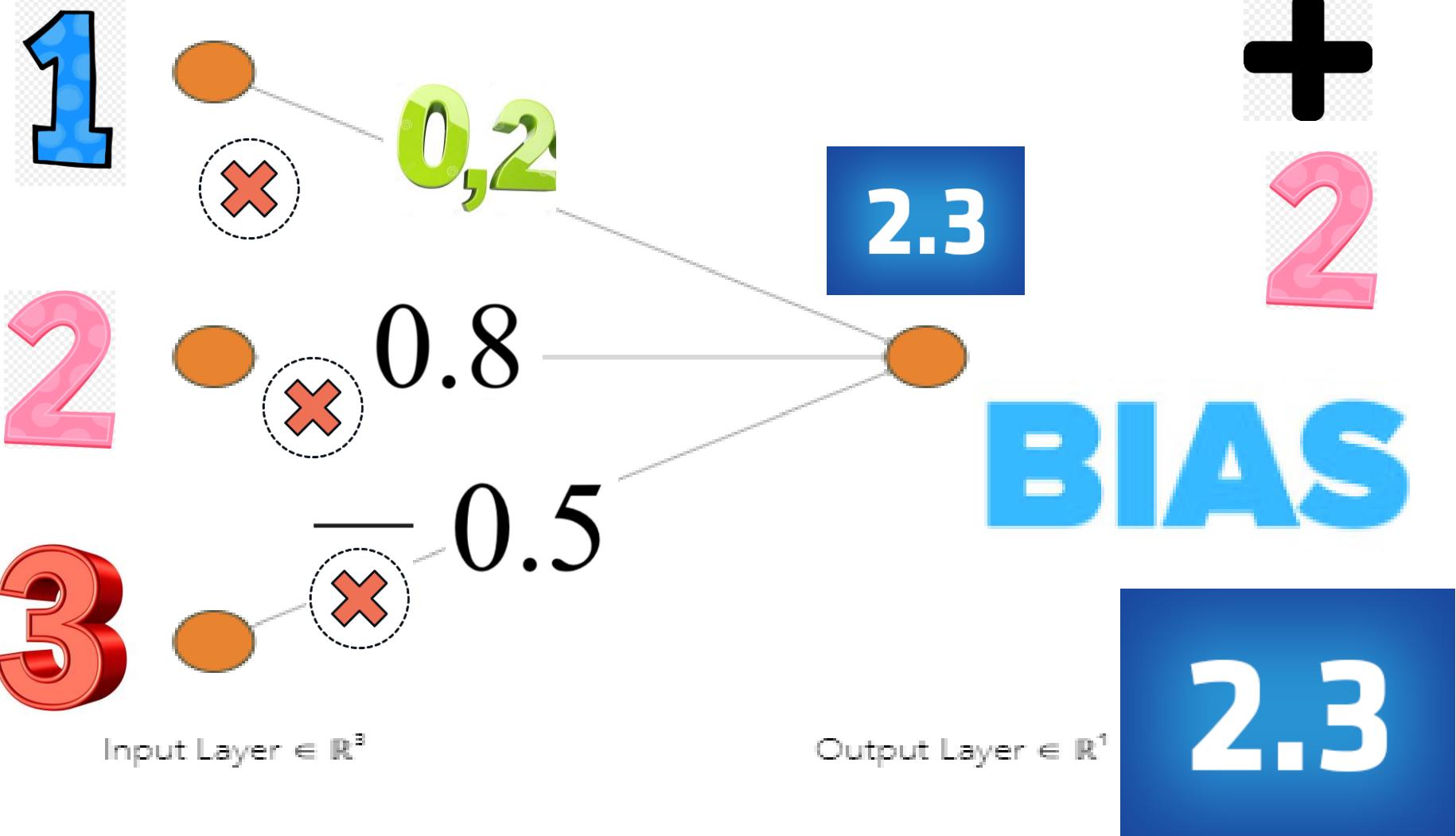
| Aktivasyon Fonksiyonu | En Çok Kullanıldığı Alan | Avantajları | Dezavantajları | Aralığı |
|-----------------------|---|--|--|---------|
| Sigmoid | Lojistik Regresyon, Binary Classification, GANs | 0-1 aralığında olması nedeniyle olasılık hesaplamalarında faydalıdır. | Gradient vanishing problemi 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 |



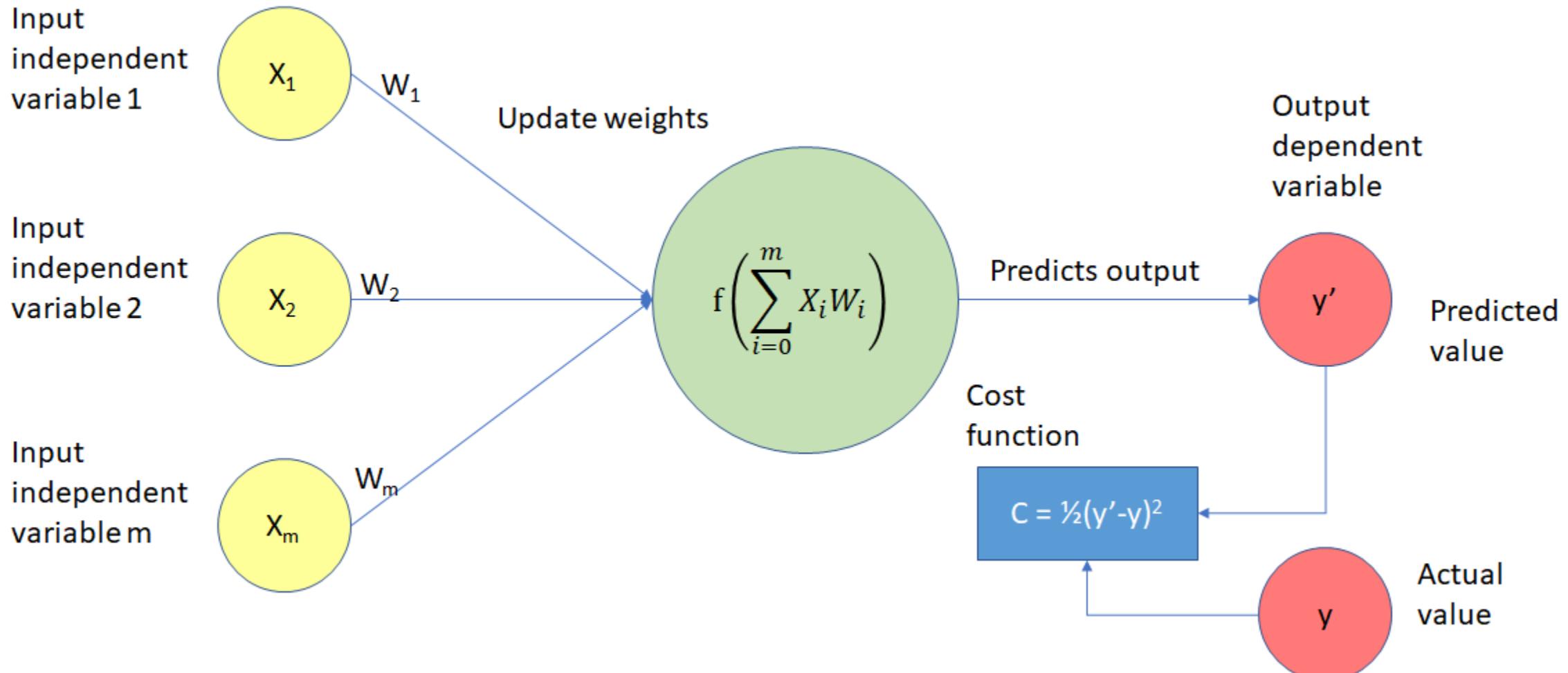
0,2
1.6 = 1.5 = 0.3





COST FUNCTION

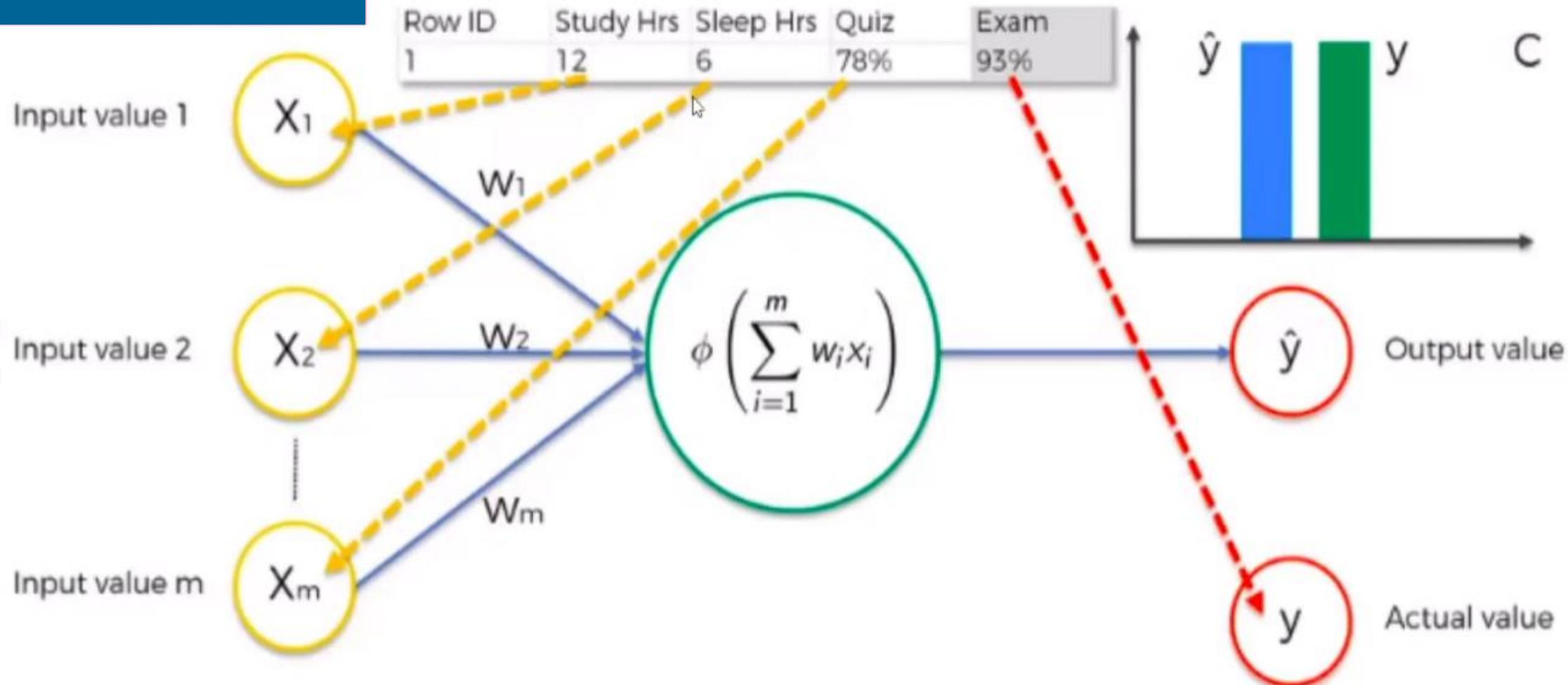
COST FUNCTION





COST FUNCTION

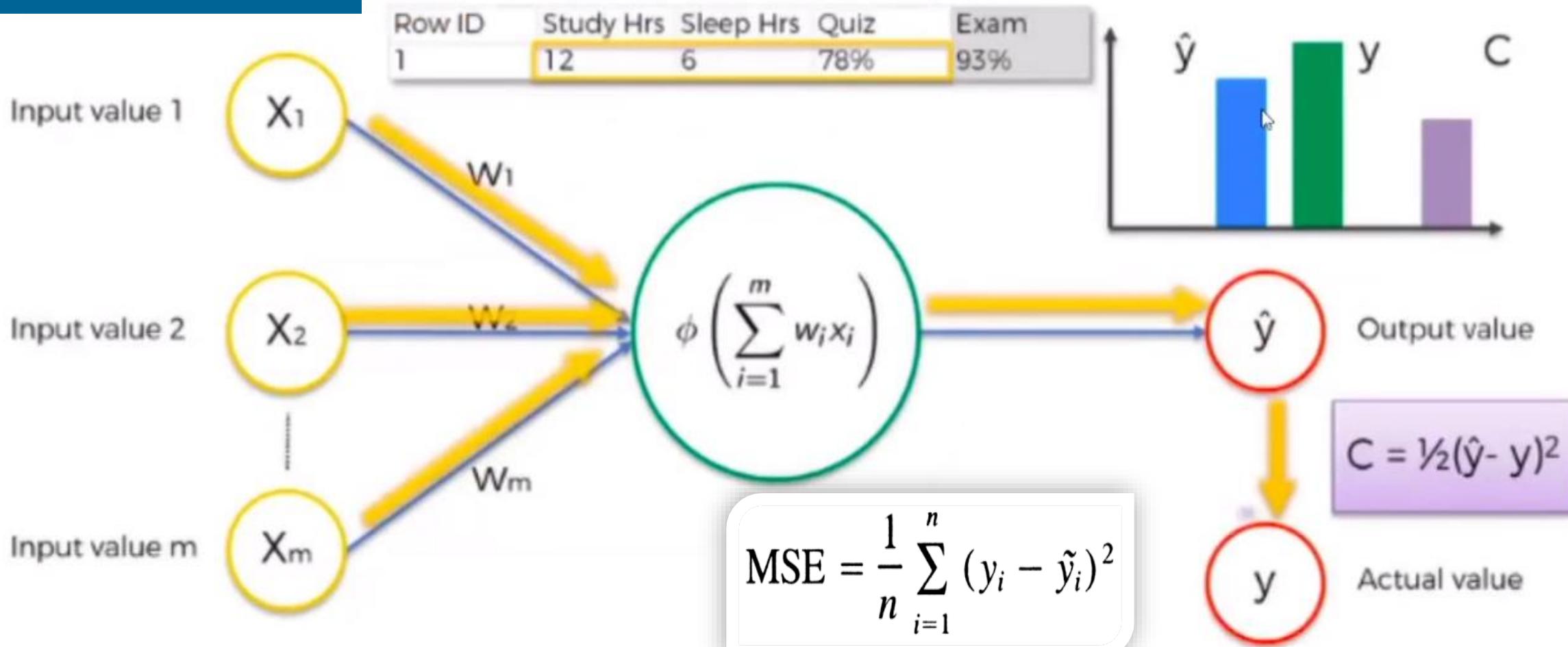
EXAMPLE





COST FUNCTION

EXAMPLE





COST FUNCTION

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



COST FUNCTION

| 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 |

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

Burada:

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

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

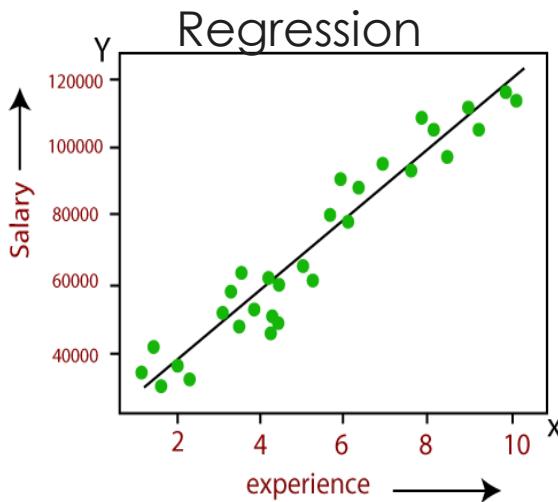
Burada:

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

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

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



```
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))

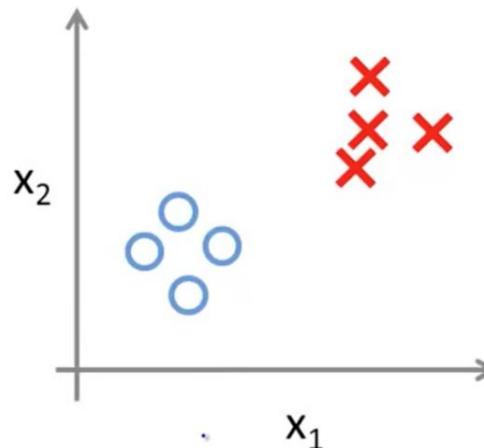
model.compile(optimizer = 'adam', loss = 'mse')
```

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"])
```

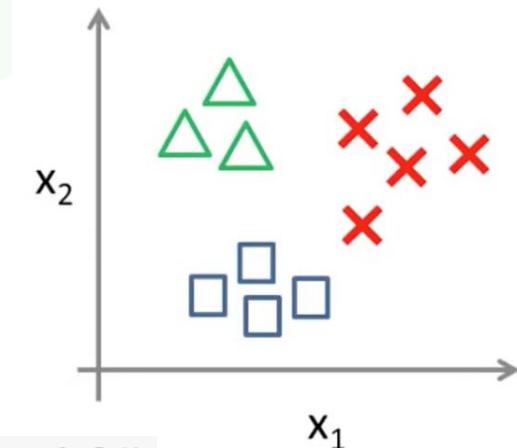
Aktivasyon **SIGMOID** olmalıdır!

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** olmalıdır!

Loss= **categorical_crossentropy**

Optimizer=Adam

Metrics= **Recall/Accuracy**



NOTEBOOKS

Regression Loss Functions

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

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



Binary Classification Loss Functions

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

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



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',
```



```
Epoch 1/600
426/426 [=====] - 0s 1ms/sample - loss: 0.6807 - val_loss: 0.6623
Epoch 2/600
426/426 [=====] - 0s 108us/sample - loss: 0.6498 - val_loss: 0.6330
Epoch 3/600
426/426 [=====] - 0s 80us/sample - loss: 0.6188 - val_loss: 0.6006
Epoch 4/600
426/426 [=====] - 0s 80us/sample - loss: 0.5878 - val_loss: 0.5806
#
```

decreases as
model learns

```
Epoch 99/600
426/426 [=====] - 0s 68us/sample - loss: 0.0485 - val_loss: 0.1003
Epoch 100/600
426/426 [=====] - 0s 73us/sample - 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
```



NOTEBOOKS

```
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
Epoch 7/50
```



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
```



I PROMPTED IT
TO PLAY
MOZART.

TRY
FINE-TUNING!