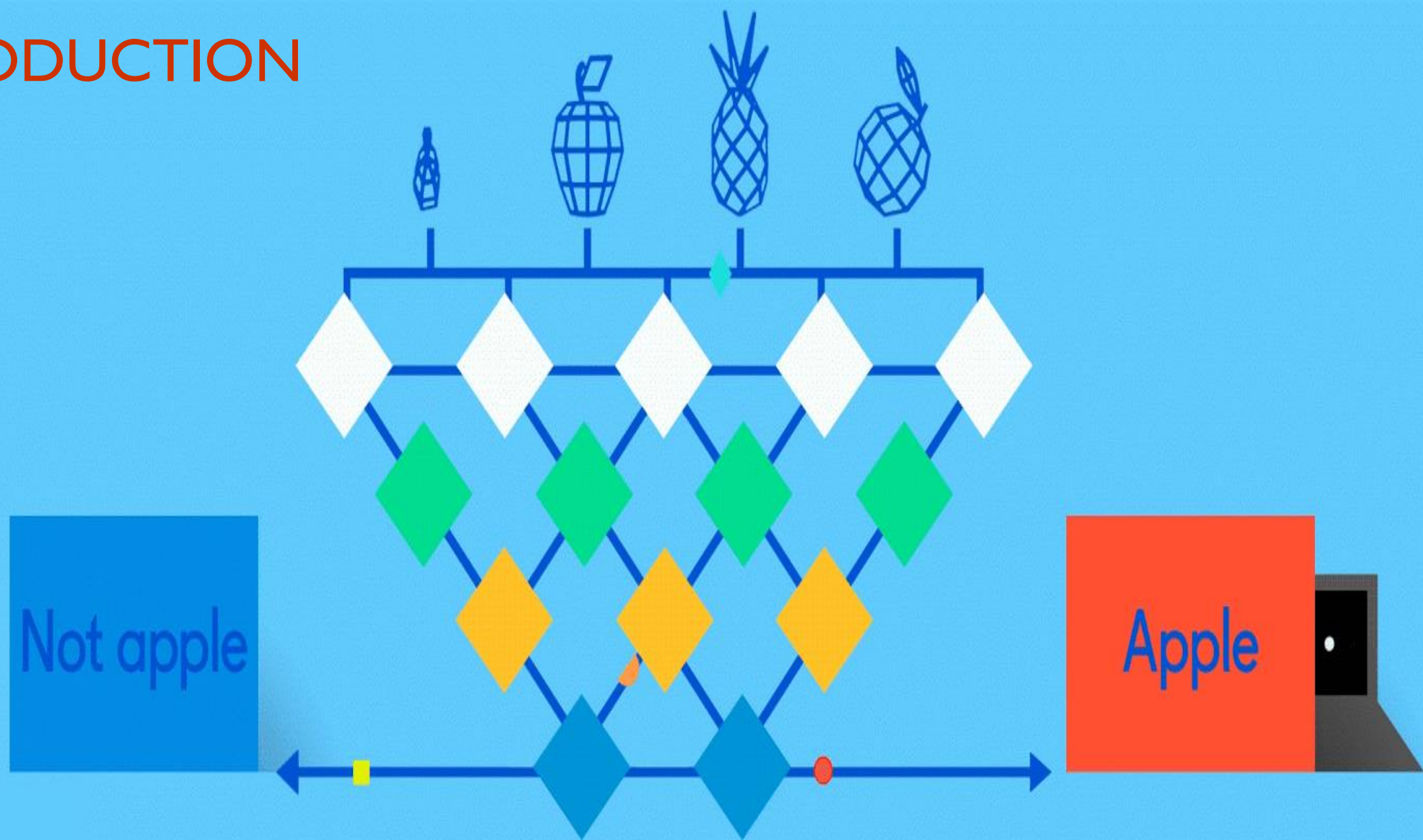


Introduction to Deep Learning Concepts (Intuitive Understanding)

INTRODUCTION



WHAT WE LEARN?

Challenges...

A big topic...difficult to know where to start...

Unfamiliar terminology...

Hard to abstract...

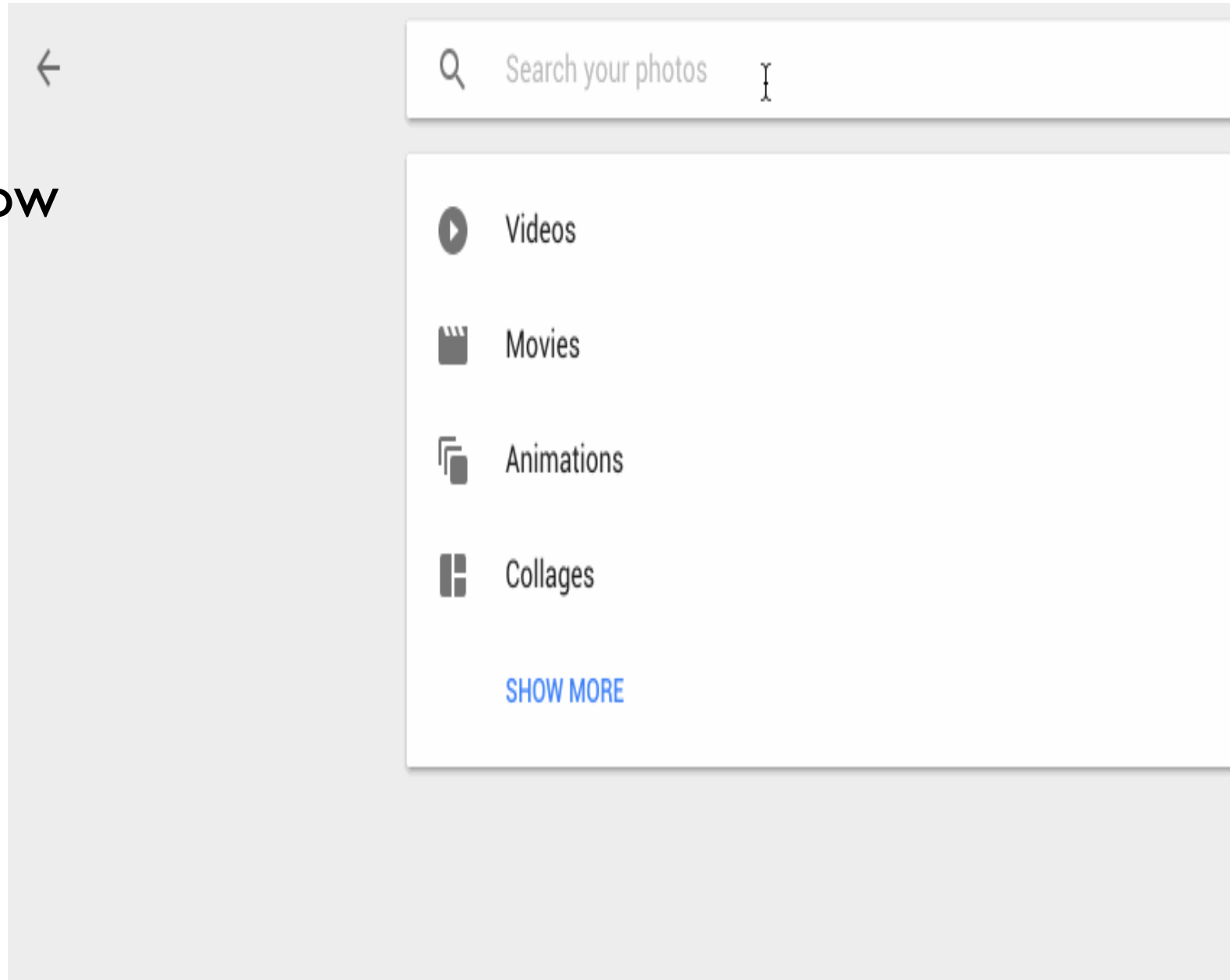
Approach...

Simplify...

Demystify...

Intuitions...

Miss out...

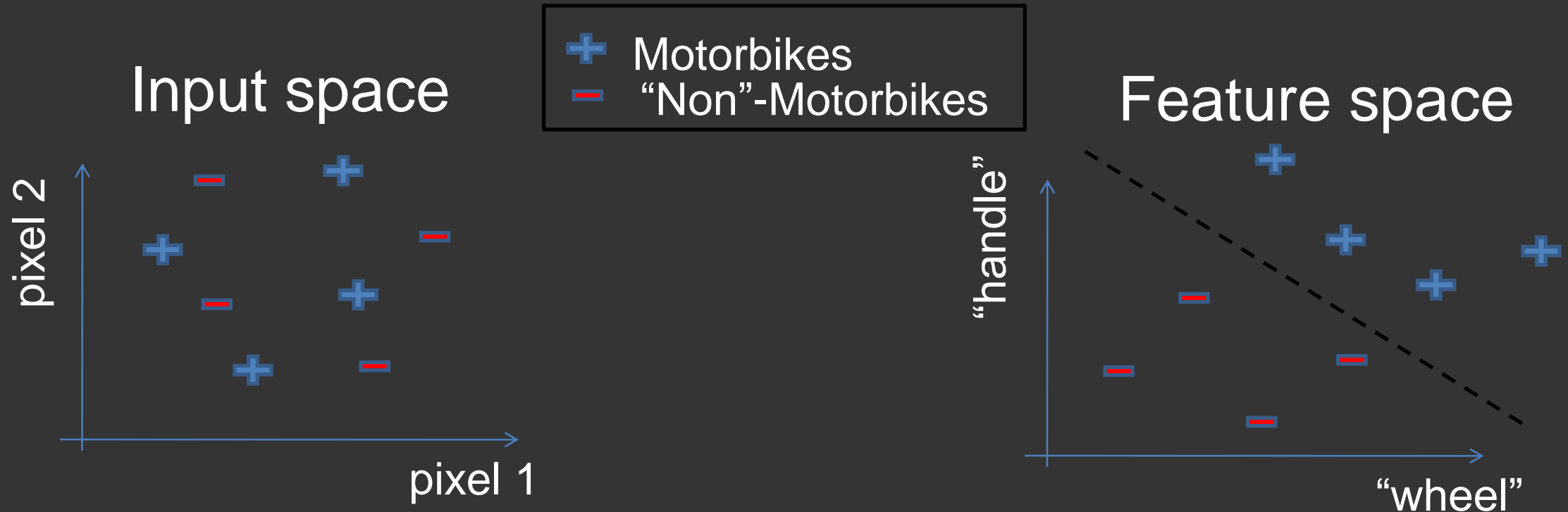


Deep learning isn't magic.



But, it is very good at finding patterns...to make predictions.

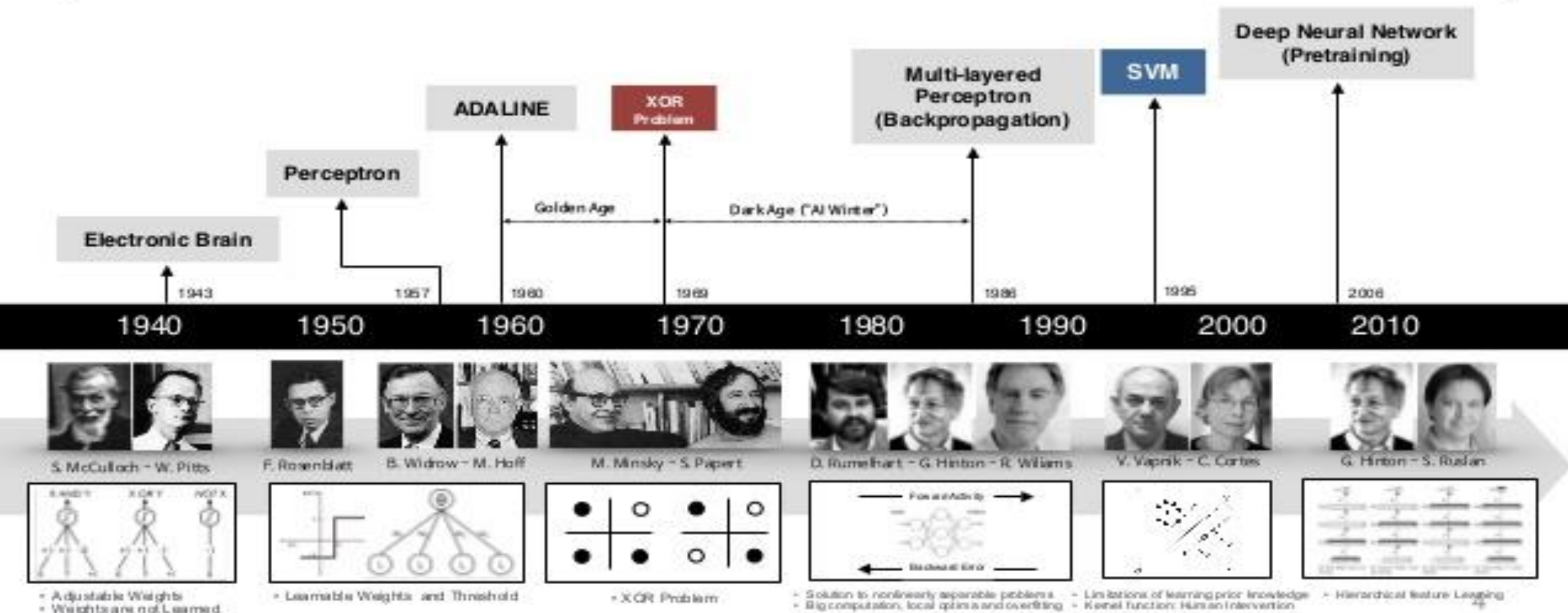
What is Machine Learning?



DEEP LEARNING

Brief History of Neural Network

DEVIEW
2015



DEEP LEARNING

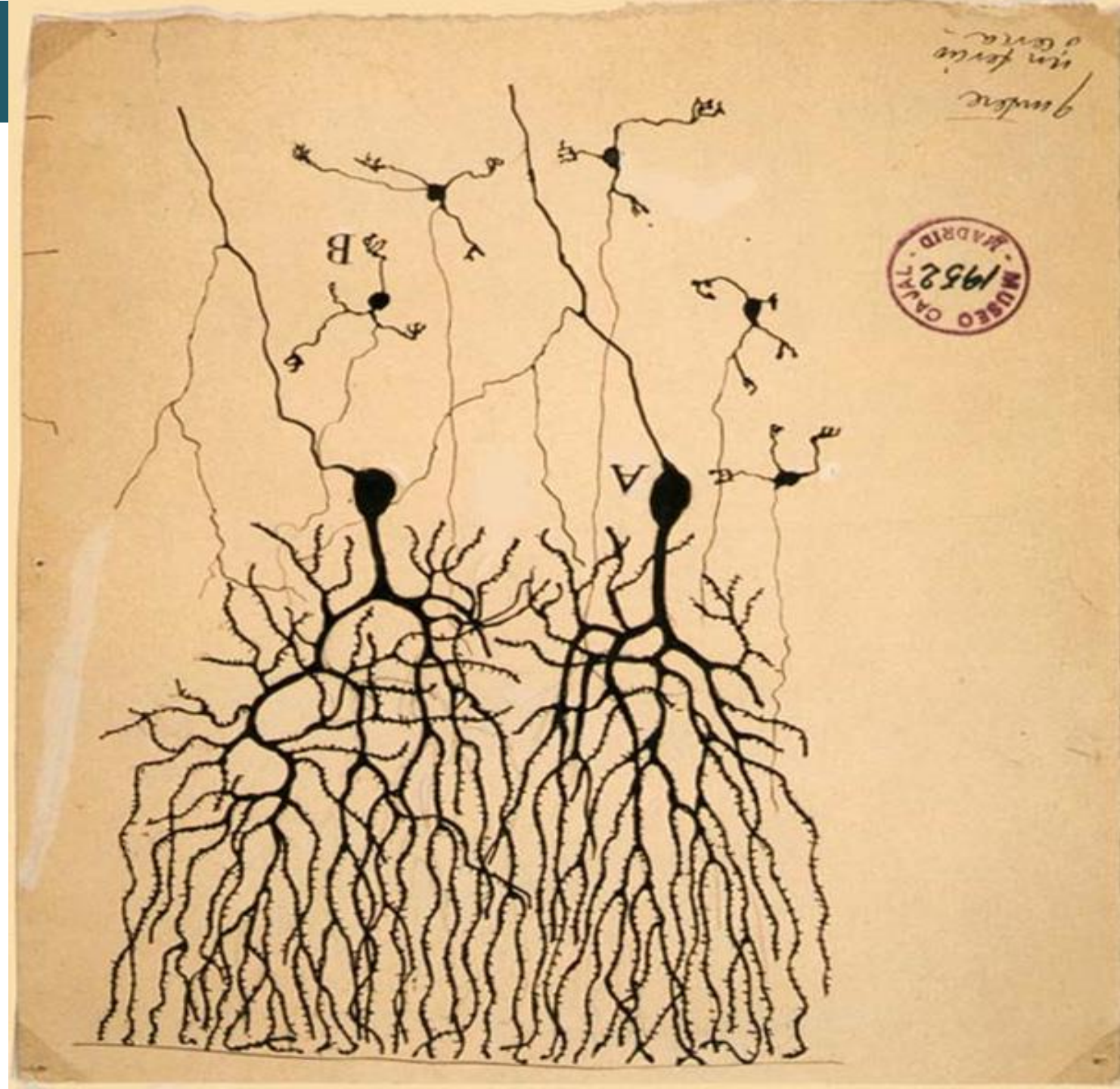
Dominant technology inspired by Biology...



NEURON



NEURAL NETWORK



NEURAL NETWORK (BIOLOGY)

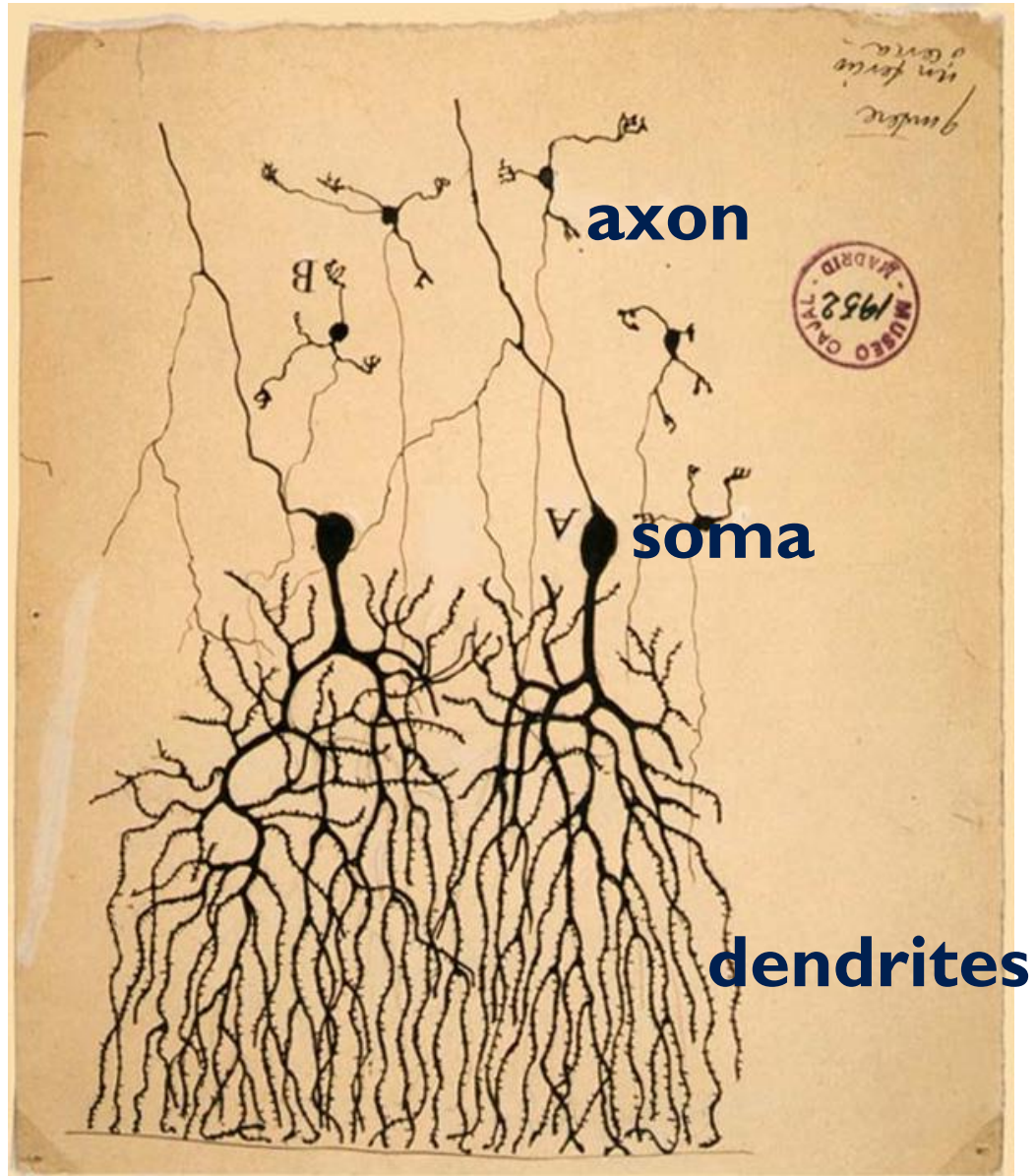
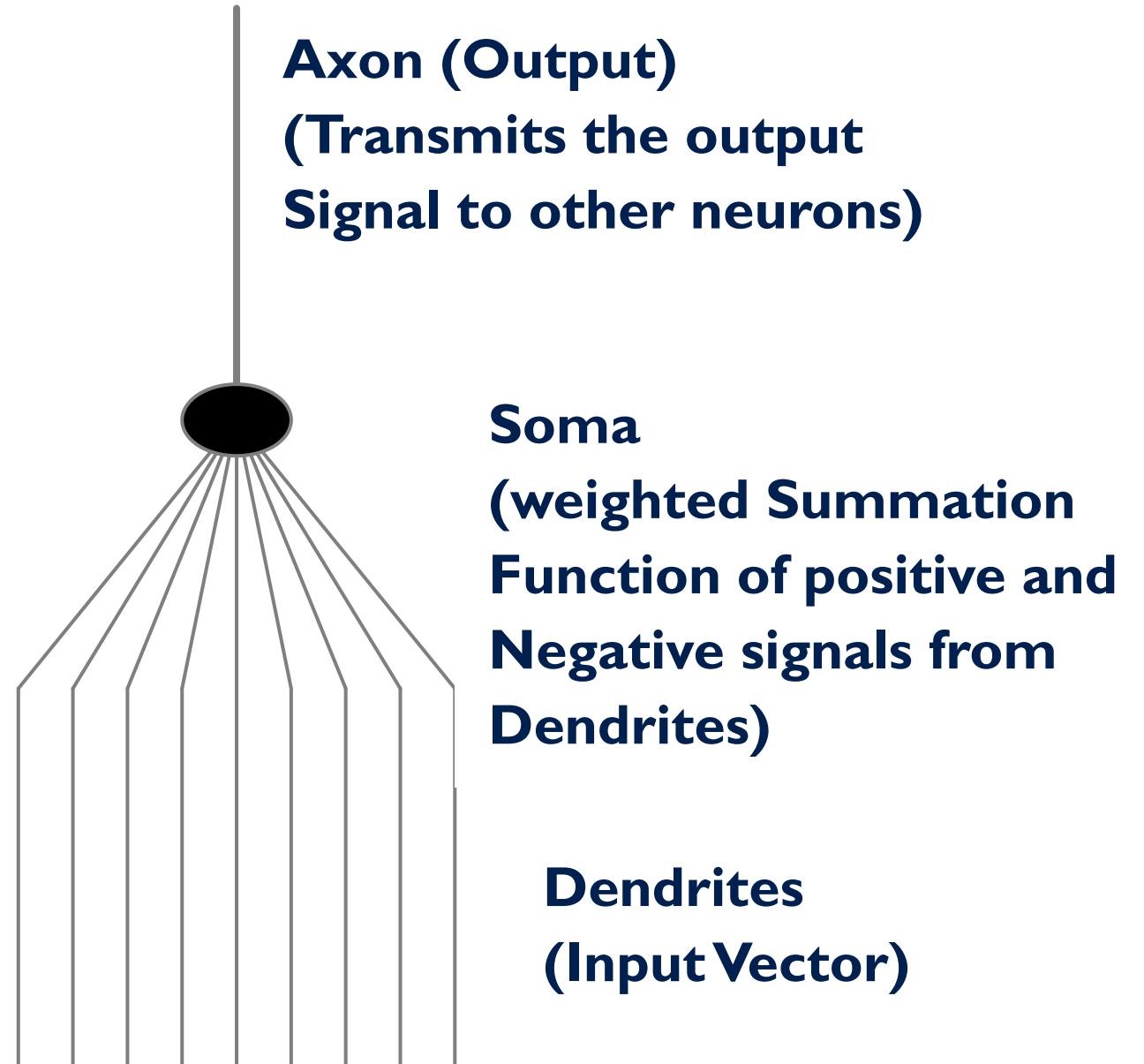
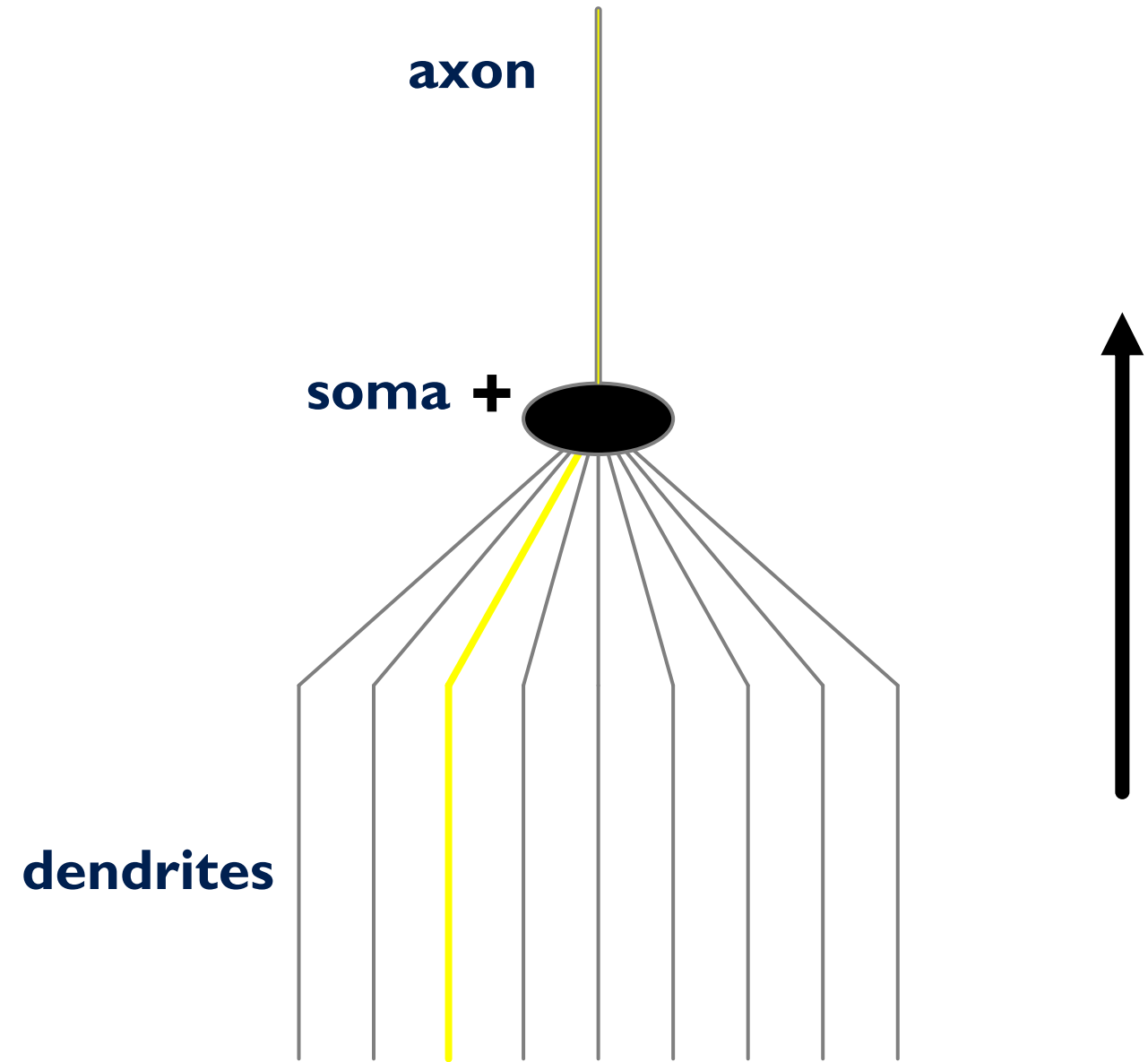


Image Credits: <https://en.wikipedia.org/wiki/Neuron>



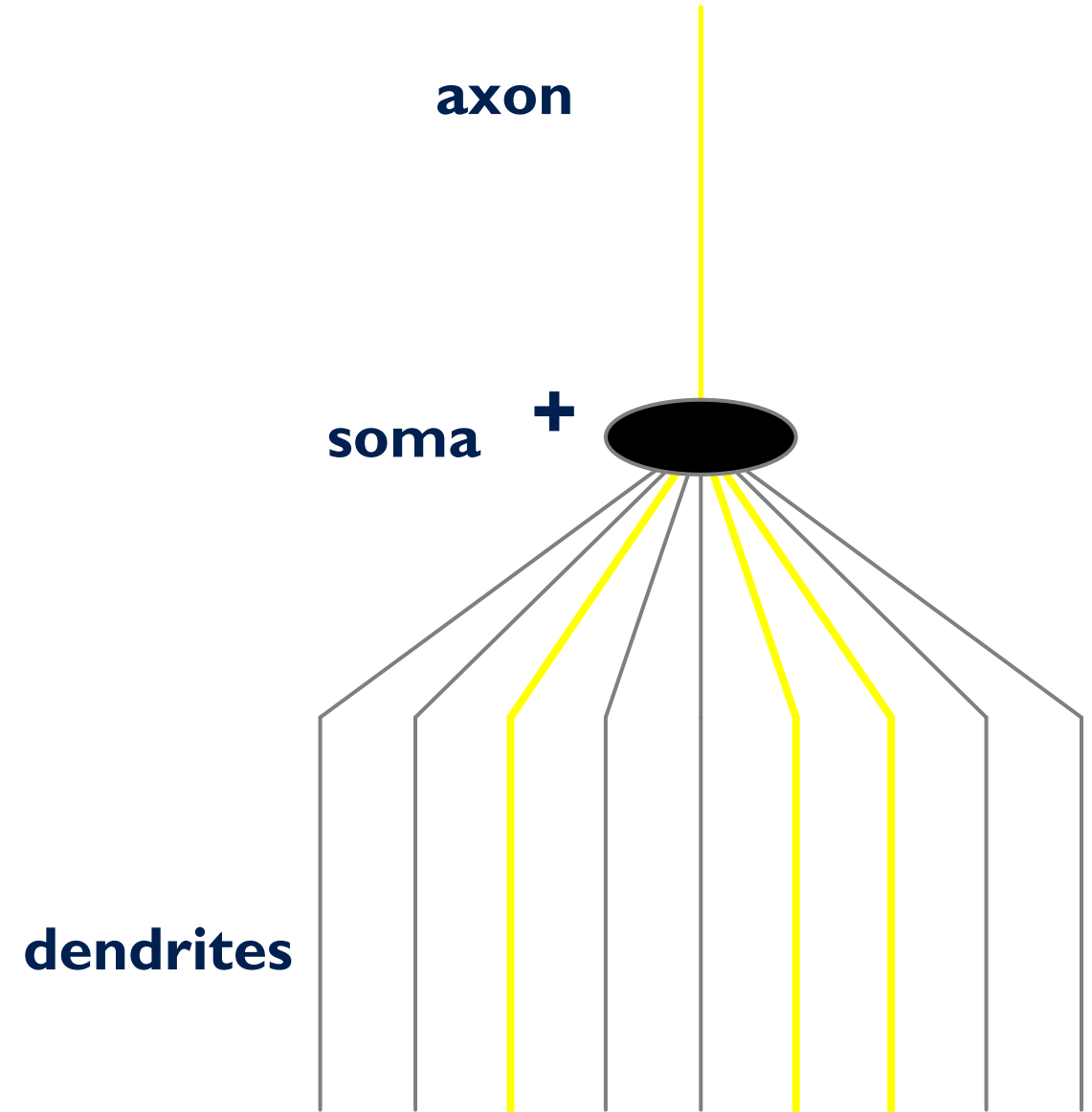
NEURAL NETWORK (BIOLOGY)...

“Soma” combines dendrites activities and passes it to axon.



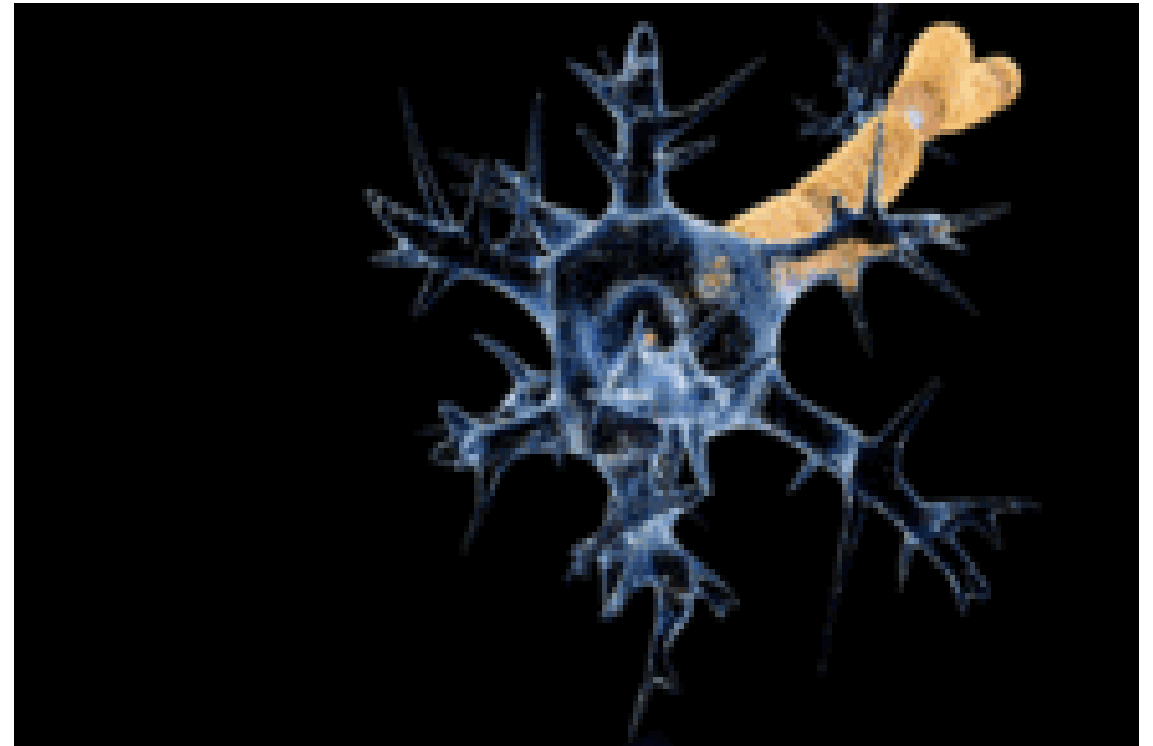
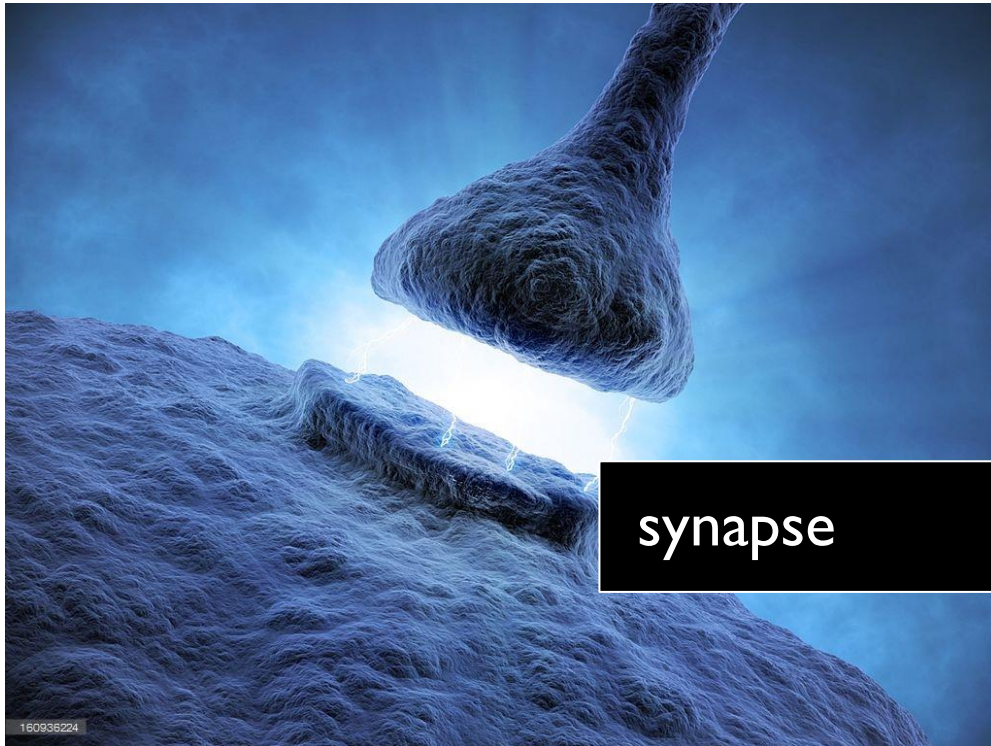
NEURAL NETWORK (BIOLOGY)...

More dendrite activity
→ more axon activity



NEURAL NETWORK (BIOLOGY)...

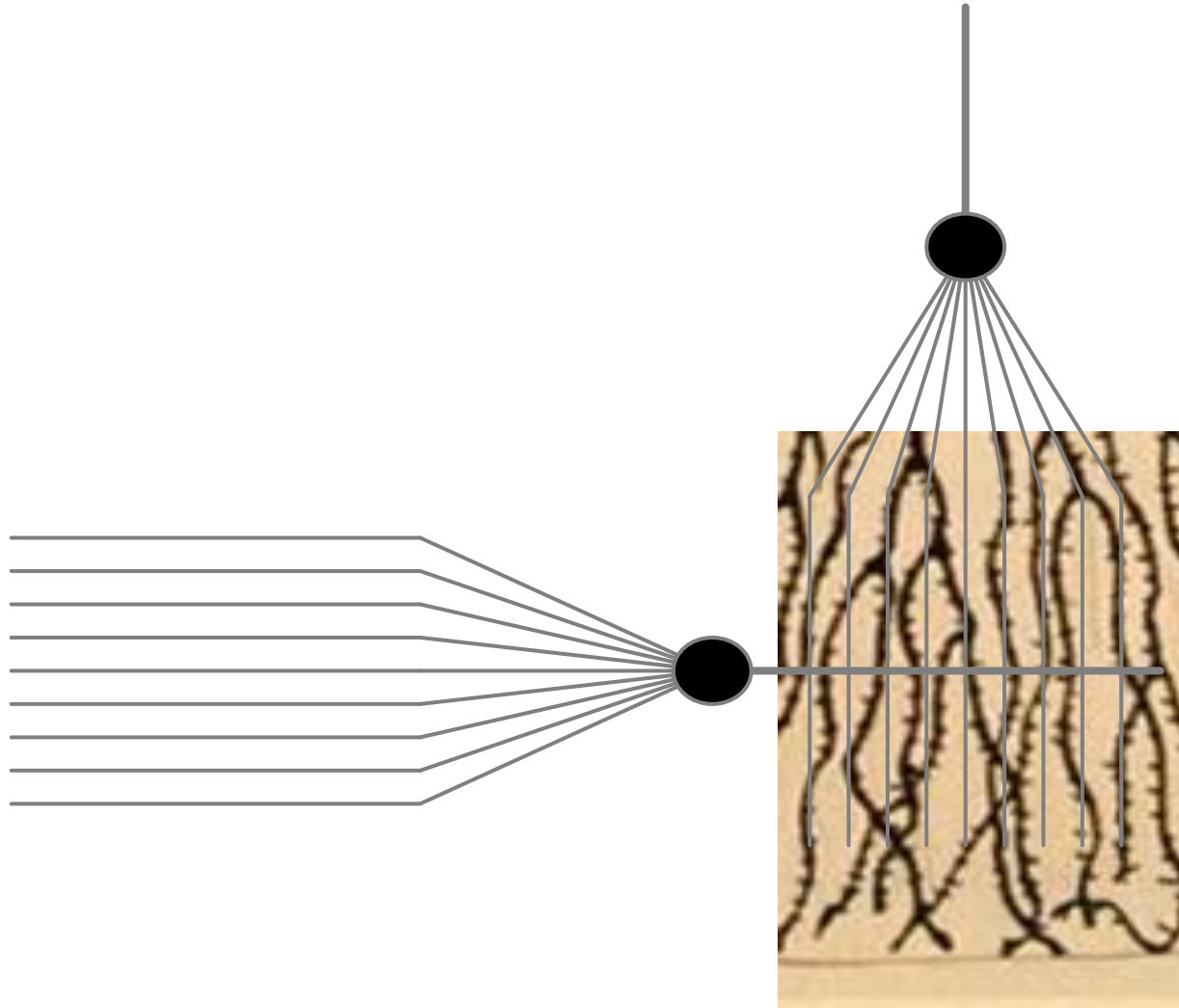
Connection between axon of one neuron and dendrites of another



Credits: <https://en.wikipedia.org/wiki/Neuron>

NEURAL NETWORK (BIOLOGY)...

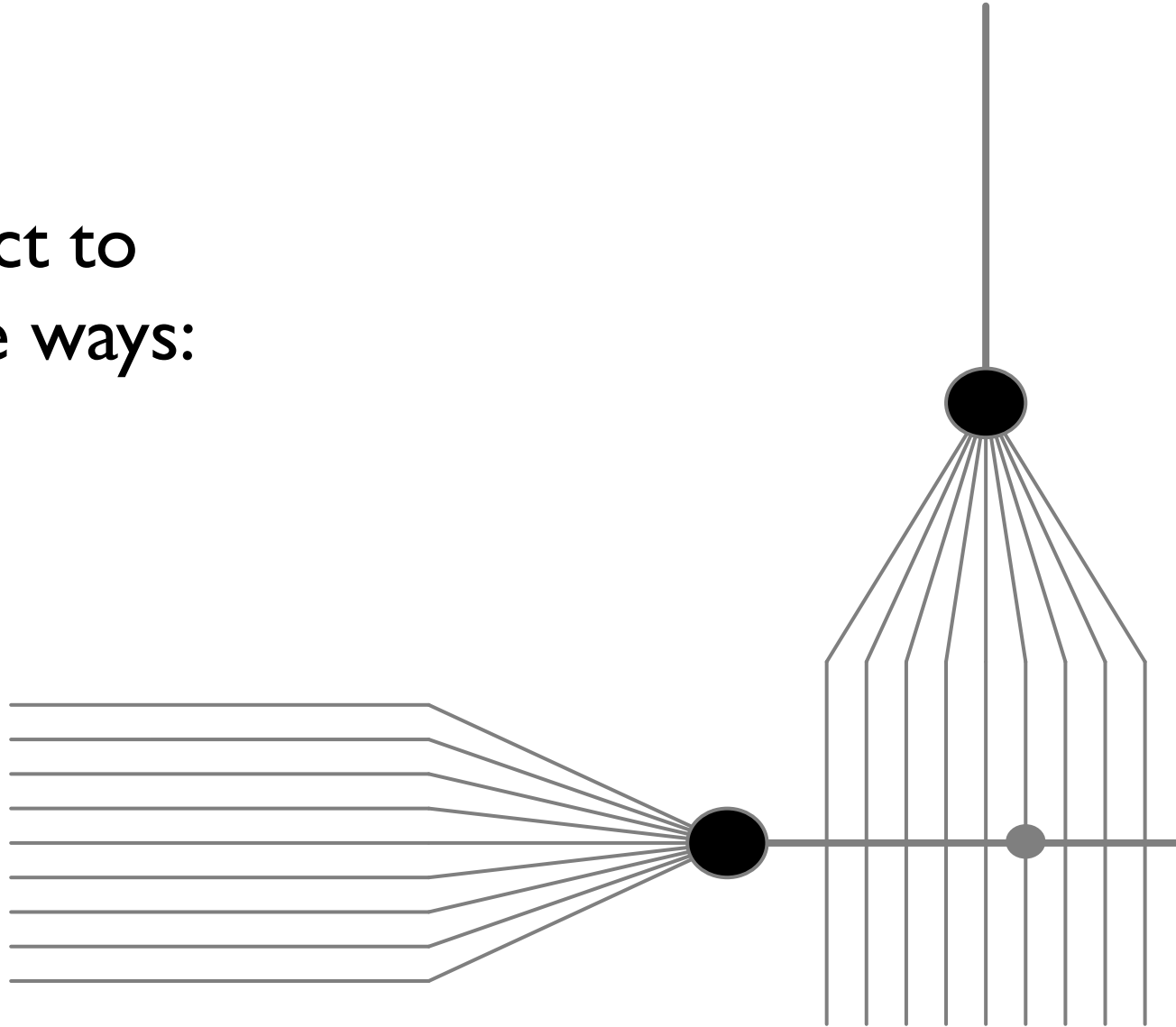
Connection between axon of one neuron and dendrites of another



NEURAL NETWORK (BIOLOGY)...

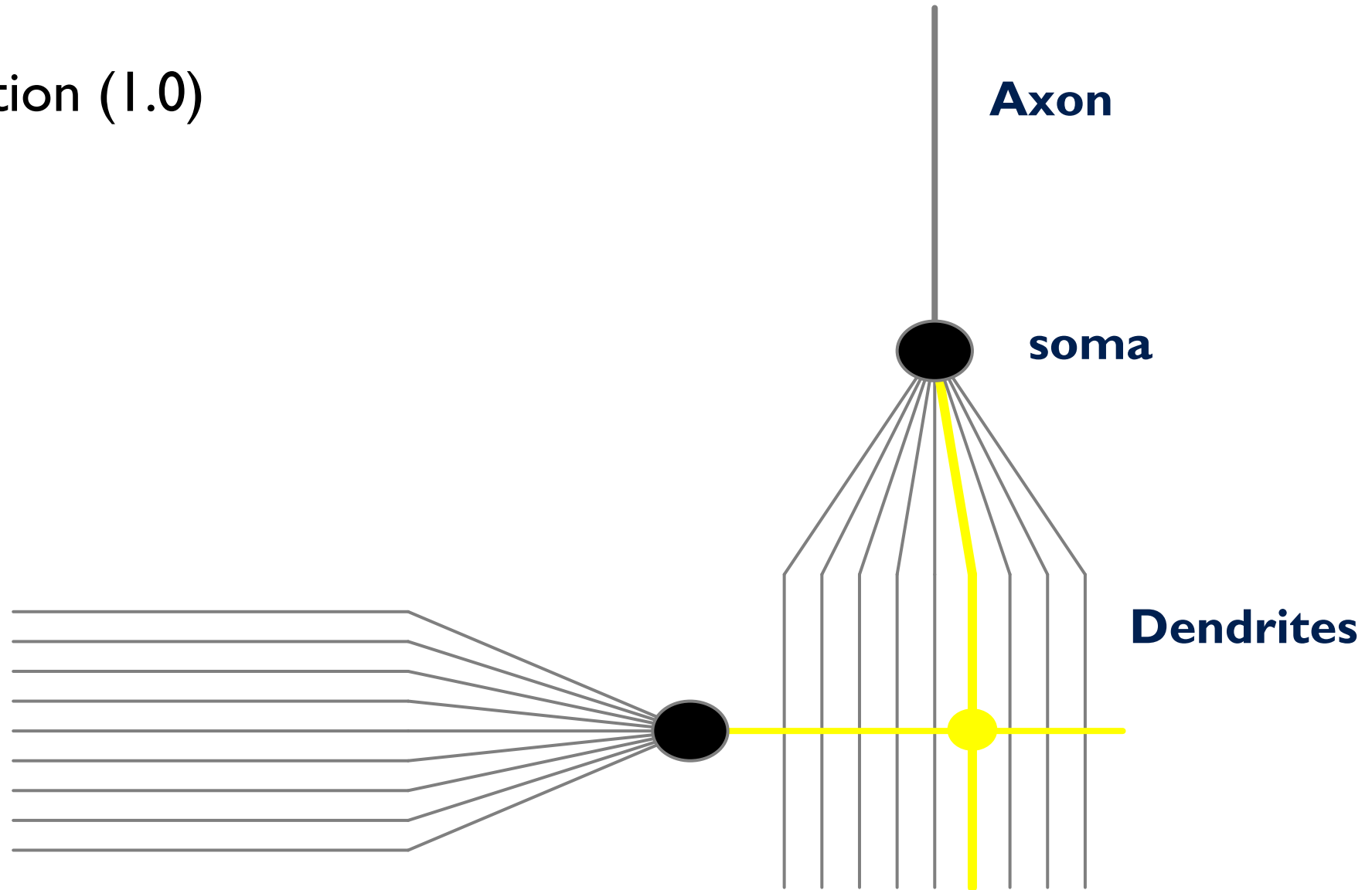
Axons can connect to dendrites in three ways:

- Strongly
- Weakly,
- Medium.



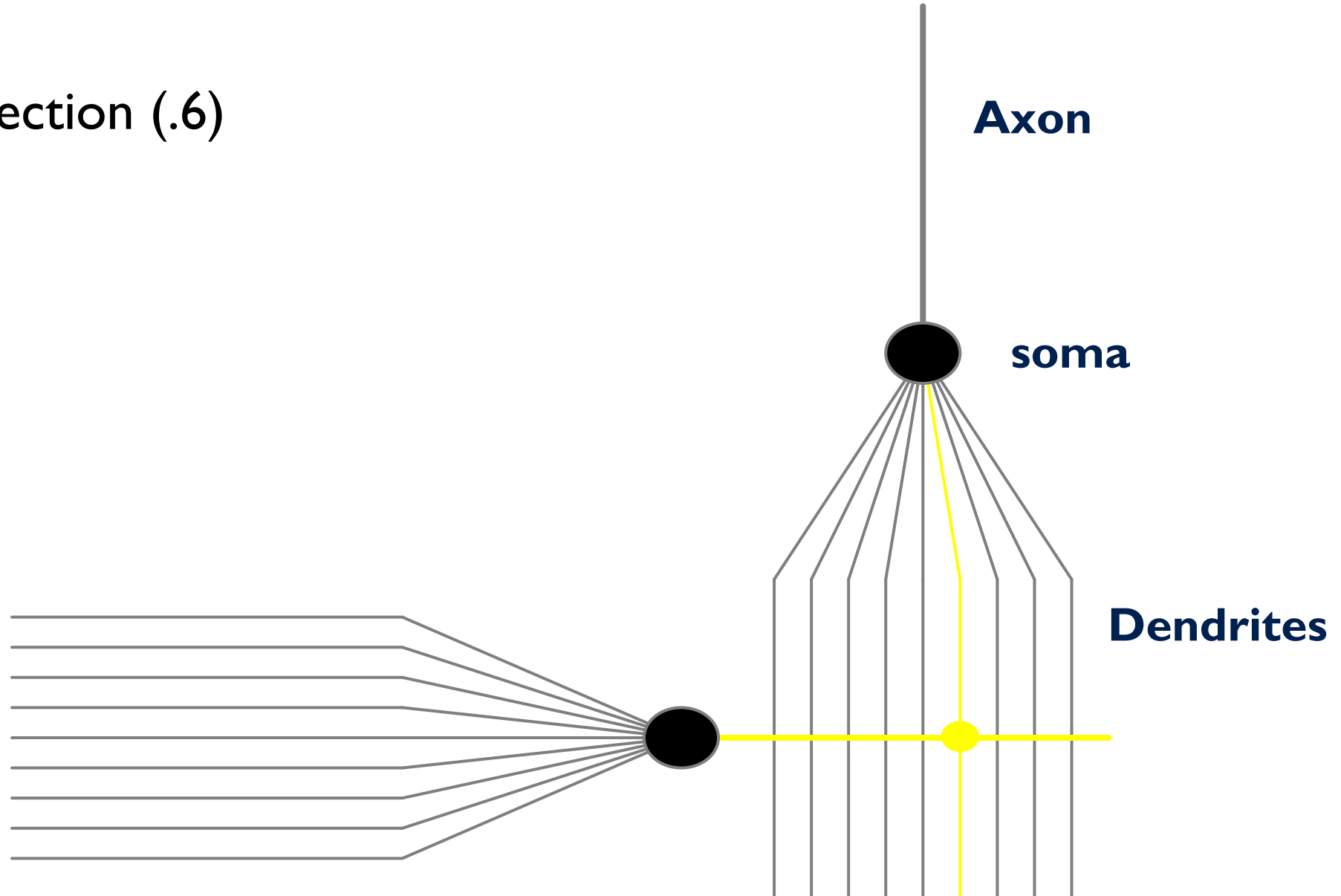
NEURAL NETWORK (BIOLOGY)...

Strong connection (1.0)



NEURAL NETWORK (BIOLOGY)...

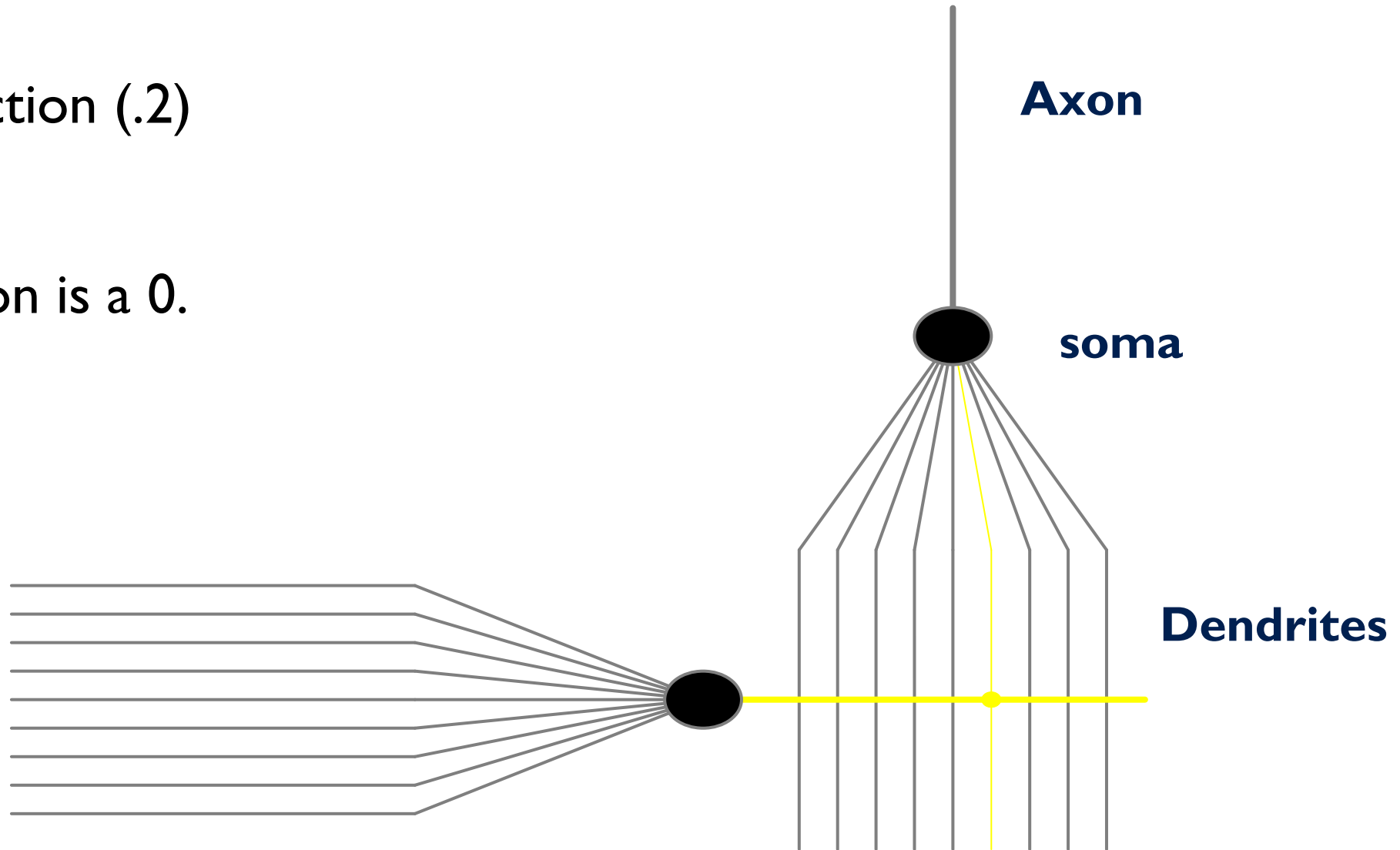
Medium connection (.6)



NEURAL NETWORK (BIOLOGY)...

Weak connection (.2)

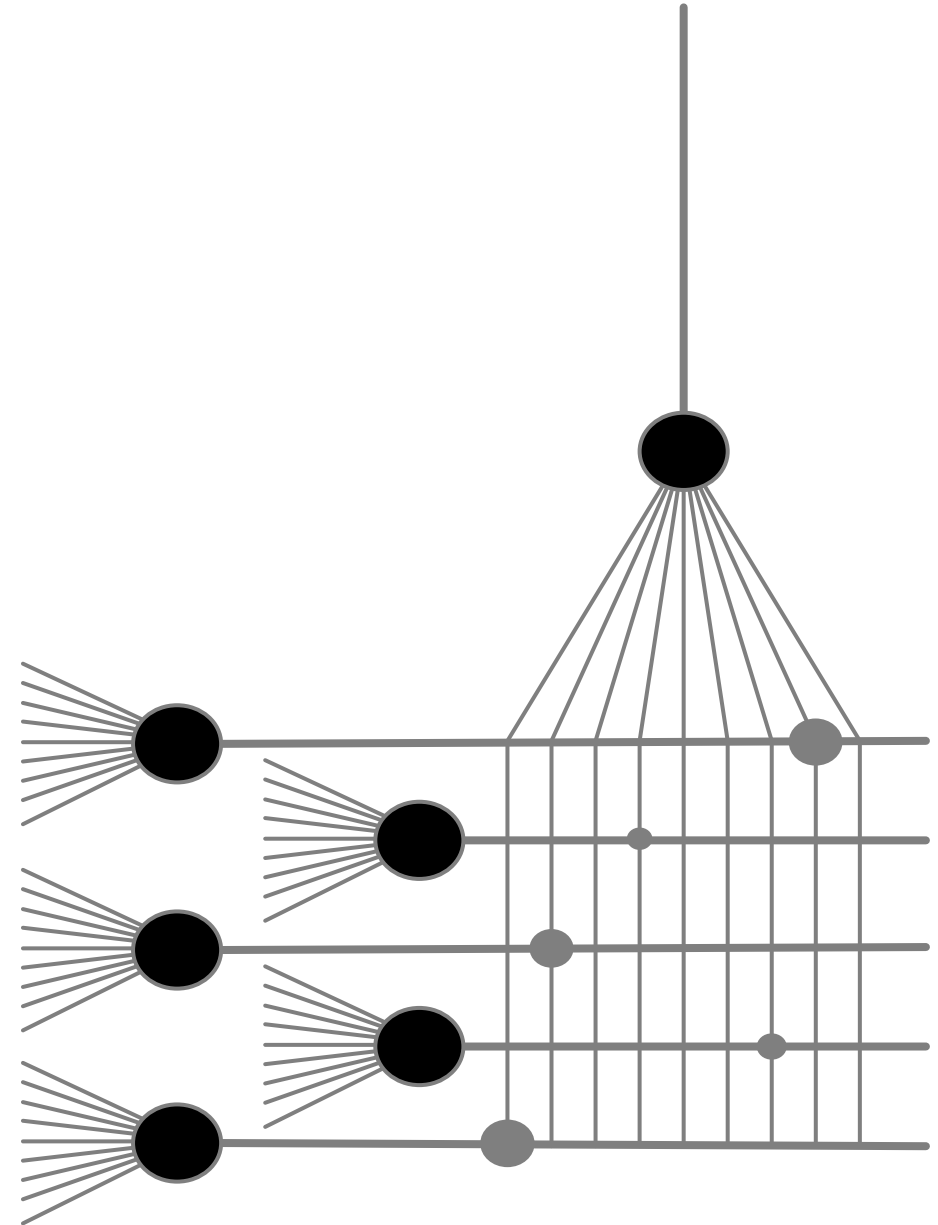
No connection is a 0.



NEURAL NETWORK (BIOLOGY)...

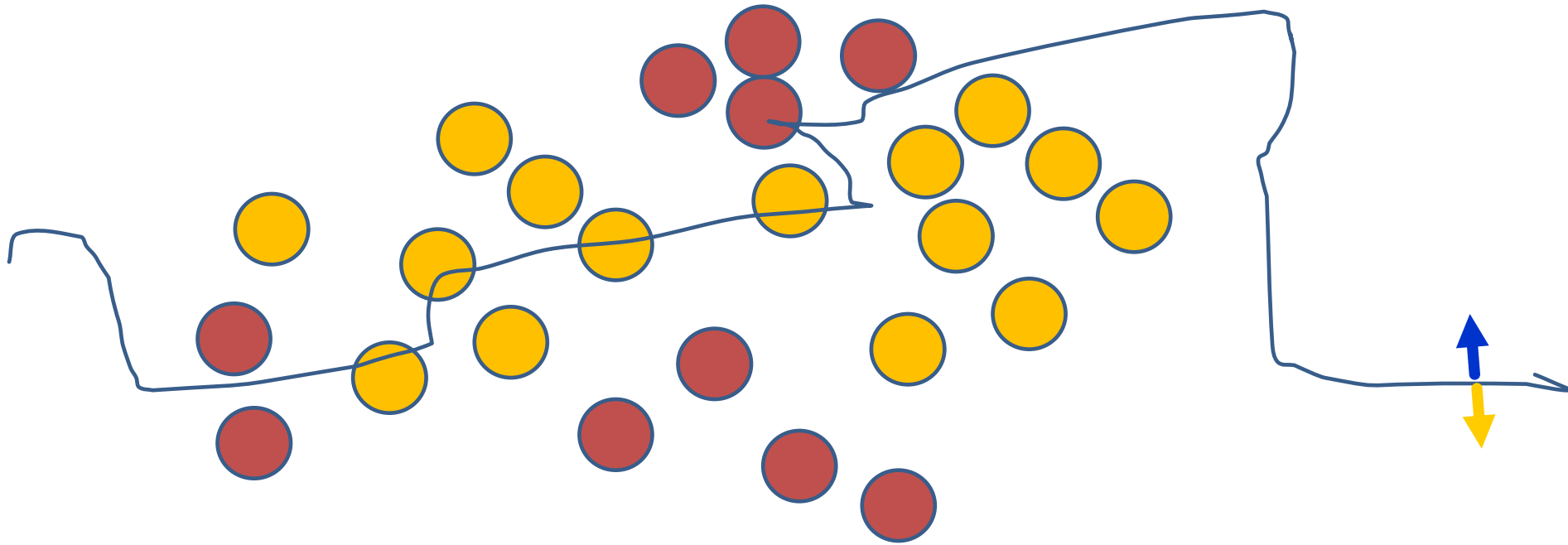
Lots of axons connect with the dendrites of one neuron.

Each has its own connection strength.



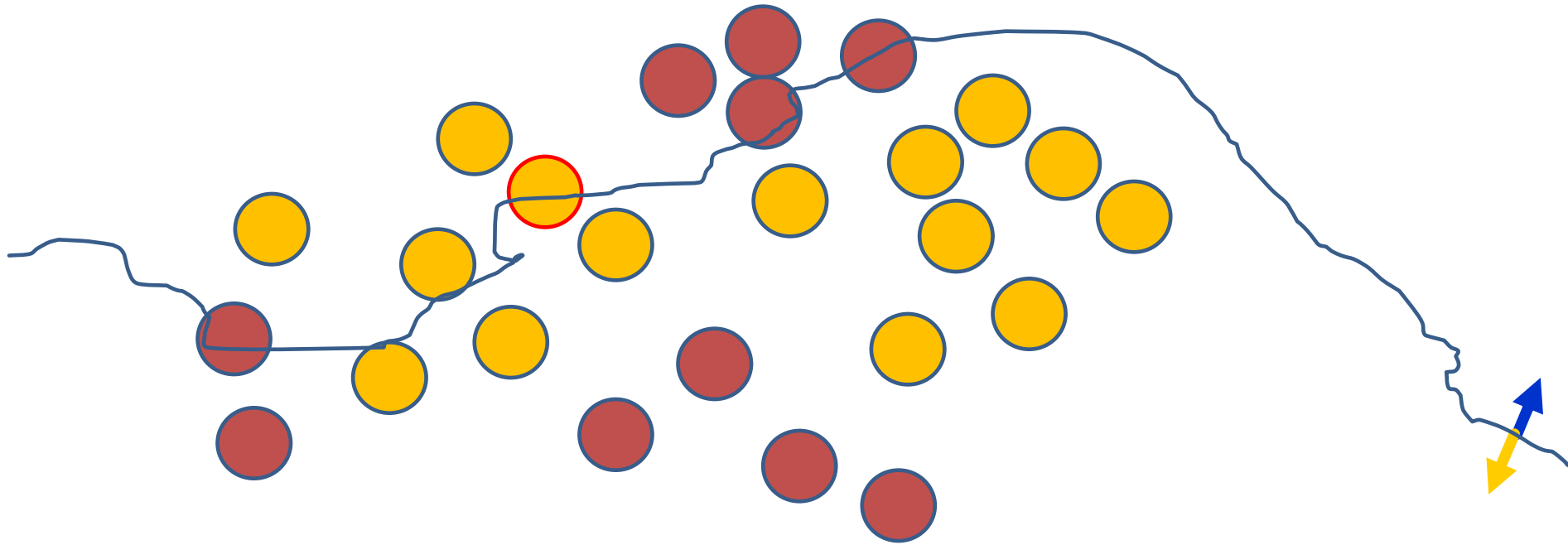
THE DECISION BOUNDARY PERSPECTIVE...

Initial random weights



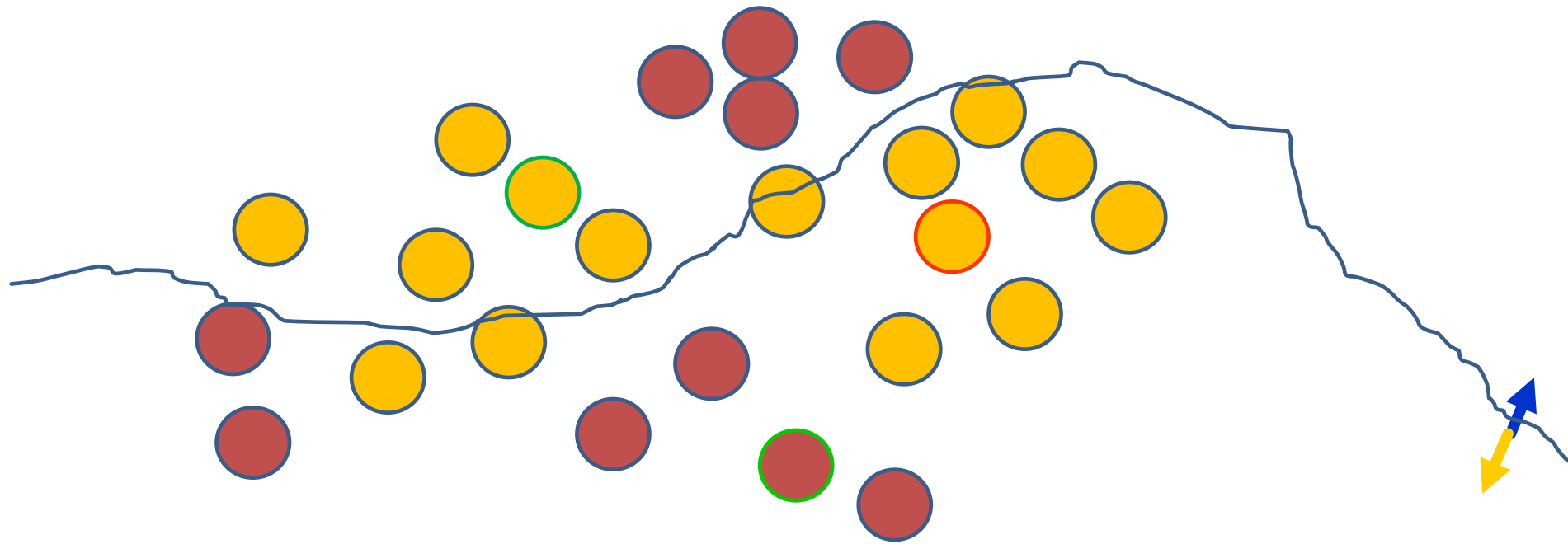
THE DECISION BOUNDARY PERSPECTIVE...

Present a training instance / adjust the weights



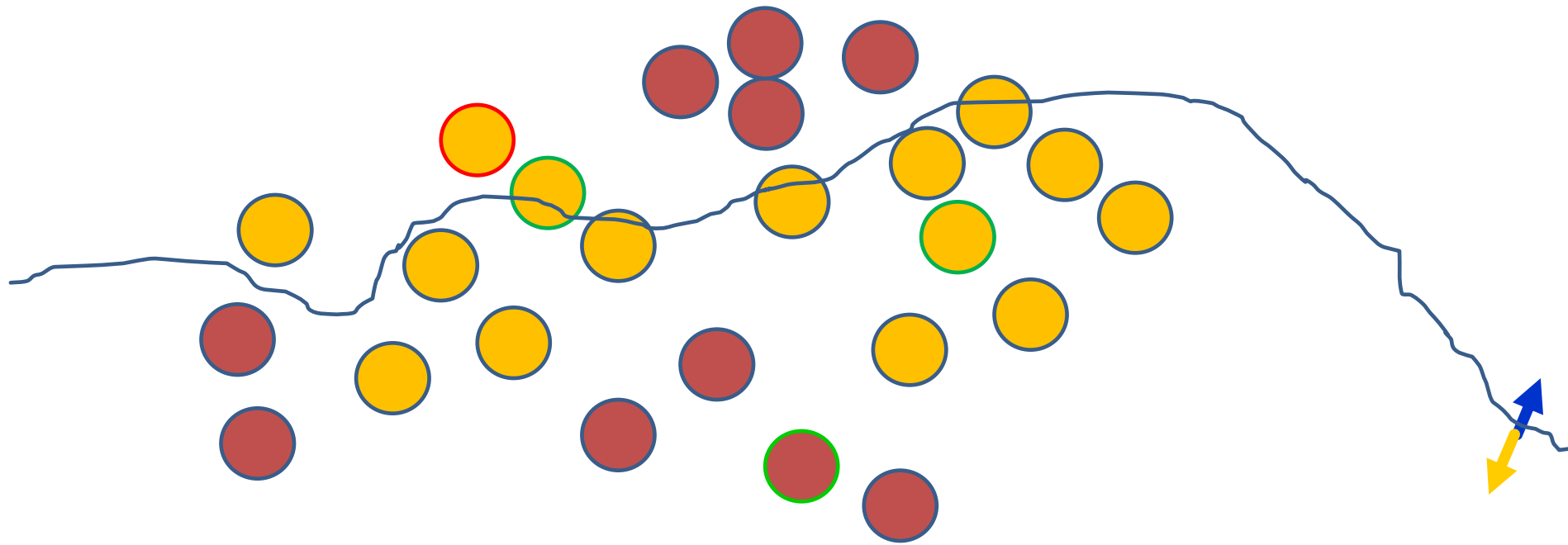
THE DECISION BOUNDARY PERSPECTIVE...

Present a training instance / adjust the weights



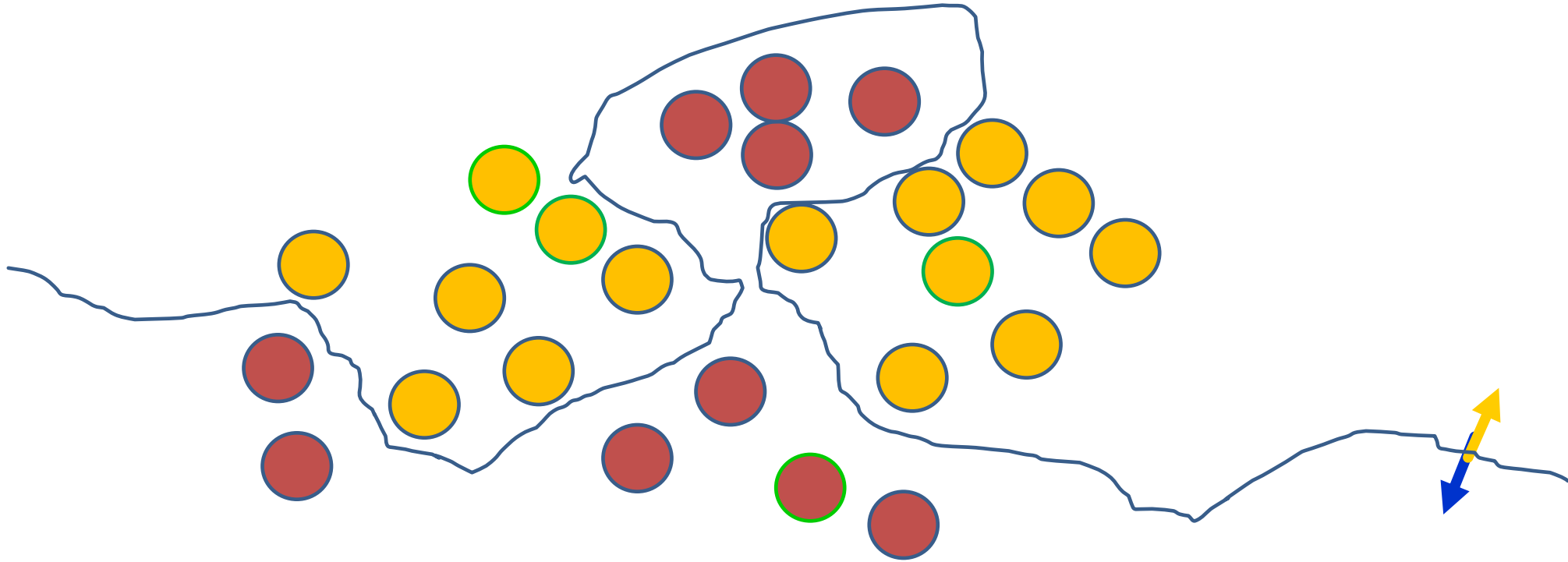
THE DECISION BOUNDARY PERSPECTIVE...

Present a training instance / adjust the weights



THE DECISION BOUNDARY PERSPECTIVE...

Eventually

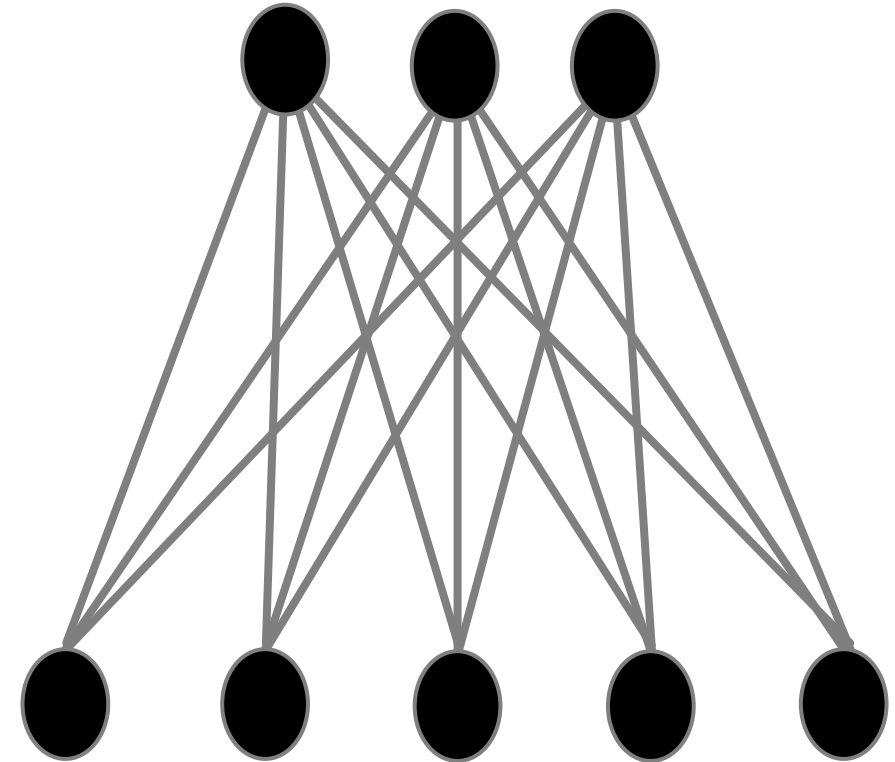


NEURAL NETWORK

Each node represents a pattern, a combination of the neurons on the previous layer.

first layer

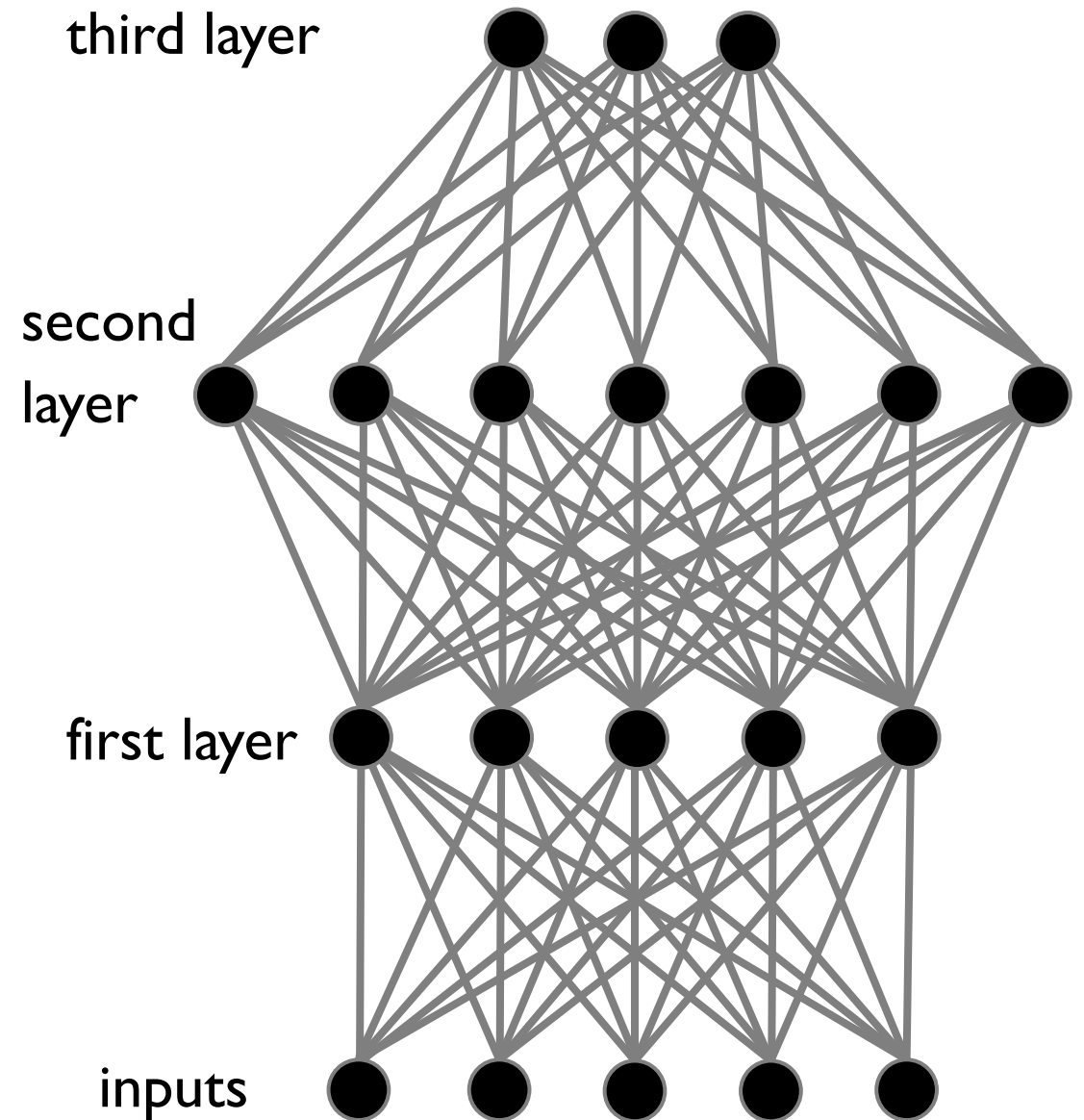
inputs



DEEP NEURAL NETWORK

If a network has more than three layers, **it's deep**.

Some 10 or more layers.



ALGORITHMS

- Single Perceptron
- Multi Layer Perceptron (MLP)
- Recurrent Neural Networks (RNN, LSTM, GRU)
- Convolution Neural Networks (CNN and its variants)
- Auto encoders (For Dimension Reduction, Unsupervised))

SINGLE PERCEPTRON

NEURAL NETWORKS .VS. TRADITIONAL COMPUTING

Recognize images of cats – Traditional Method

IF (furry) AND
IF (has tail) AND
IF (has 4 legs) AND
IF (has pointy ears) AND
Etc...

- explicitly programming the computer for what features to look for.
- However, it's pretty easy to see that **these features could also apply to a dog or a fox,**
- it would be tricky to account for unusual edge cases like a **cat without a tail or only three legs.**



NEURAL NETWORKS .VS. TRADITIONAL COMPUTING

Recognize images of cats – Neural Networks

show the computer a load of examples of **cats**



show the computer a load of examples of **not-cats**

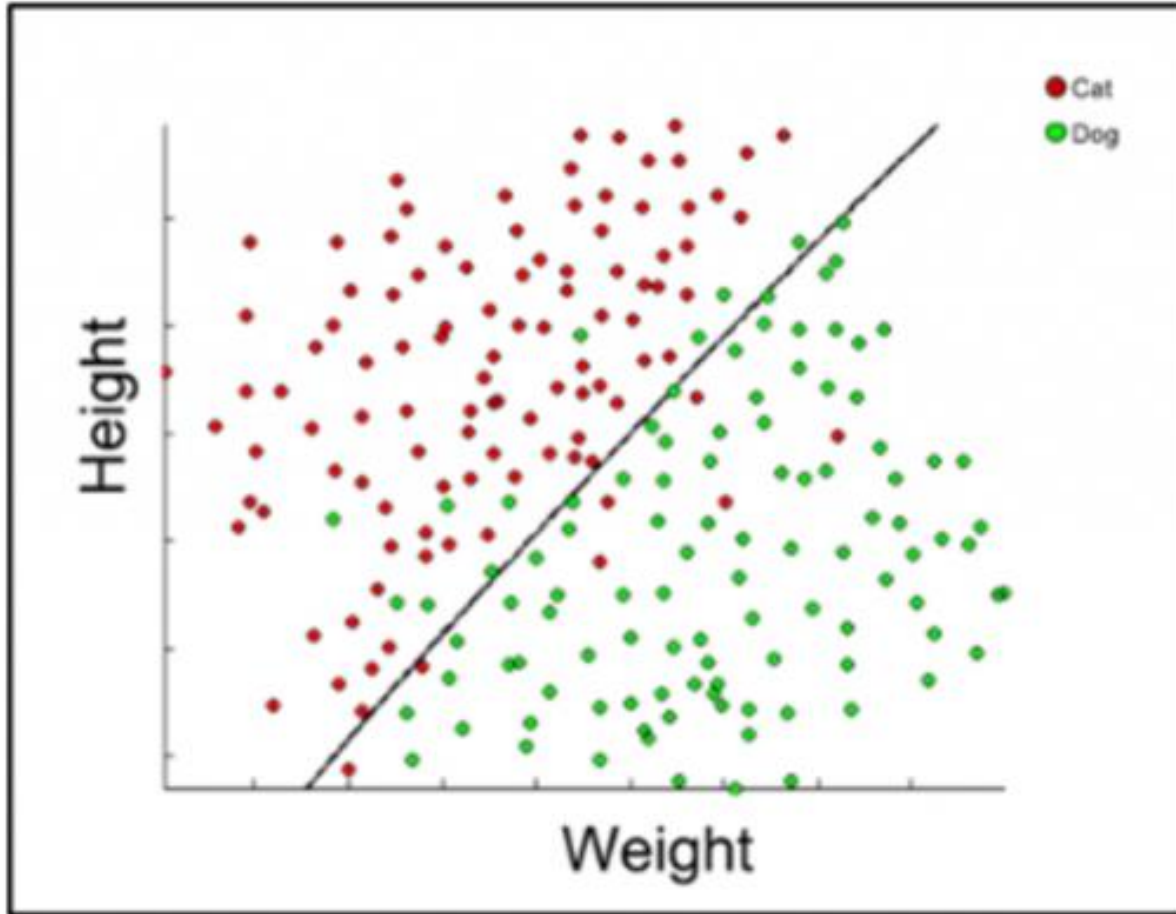


NEURAL NETWORKS .VS. TRADITIONAL COMPUTING

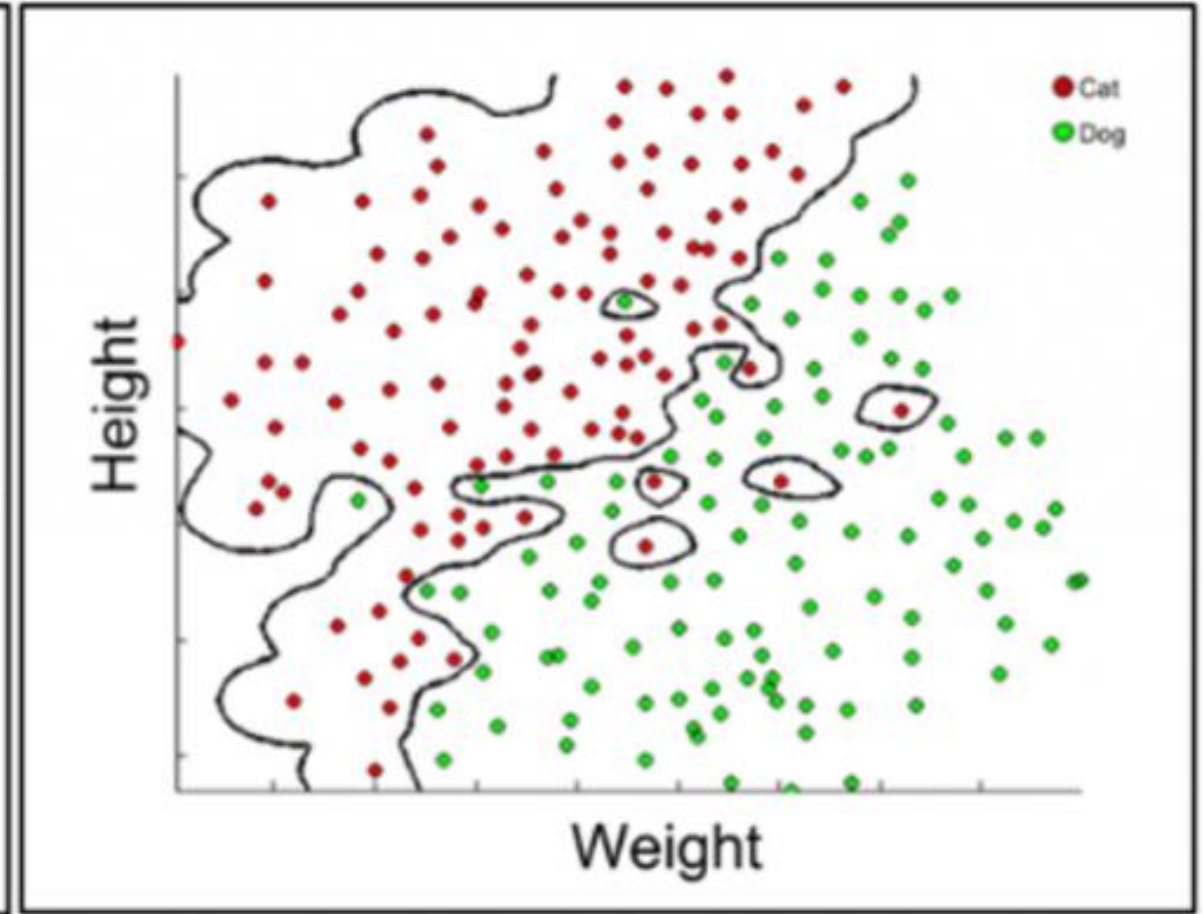
Recognize images of cats – Neural Networks

- Here, the computer works out:
 - what to look for, and
 - **what features are essential to ‘cat-ism’.**
- This is actually a lot closer to the way humans learn to distinguish objects, and is why we call them ‘neural’ networks’
 - they are **inspired by biology** and **how our own brains work.**

NEURAL NETWORKS .VS. TRADITIONAL COMPUTING



**Traditional Computing
(OR Simple Linear Models)**

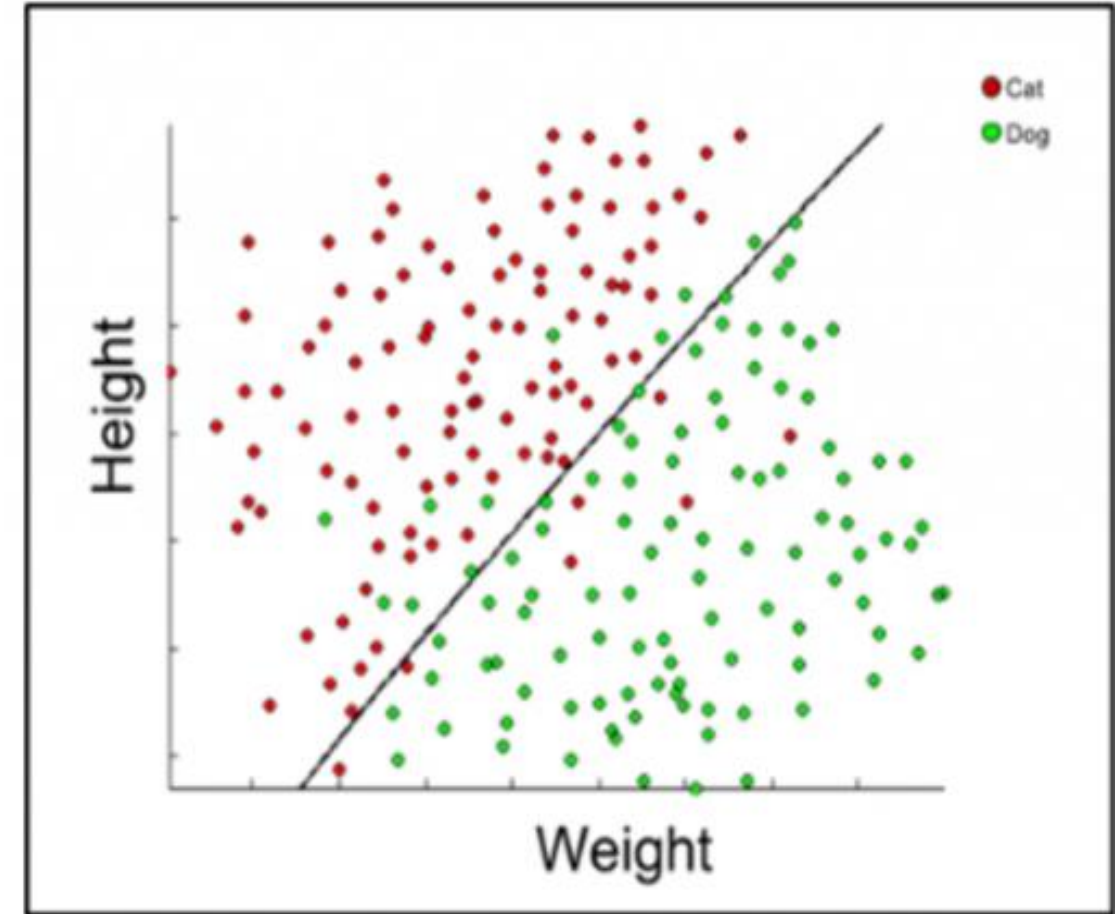


Deep Neural Networks

NEURAL NETWORKS .VS. TRADITIONAL COMPUTING

Traditional Computing:

- Linear Model
- straight line attempts to separate the red dots (representing cats) from the green dots (representing dogs).
- Does a relatively good job, still dots on the wrong side of the line.
- a very easy function to model using conventional computing,

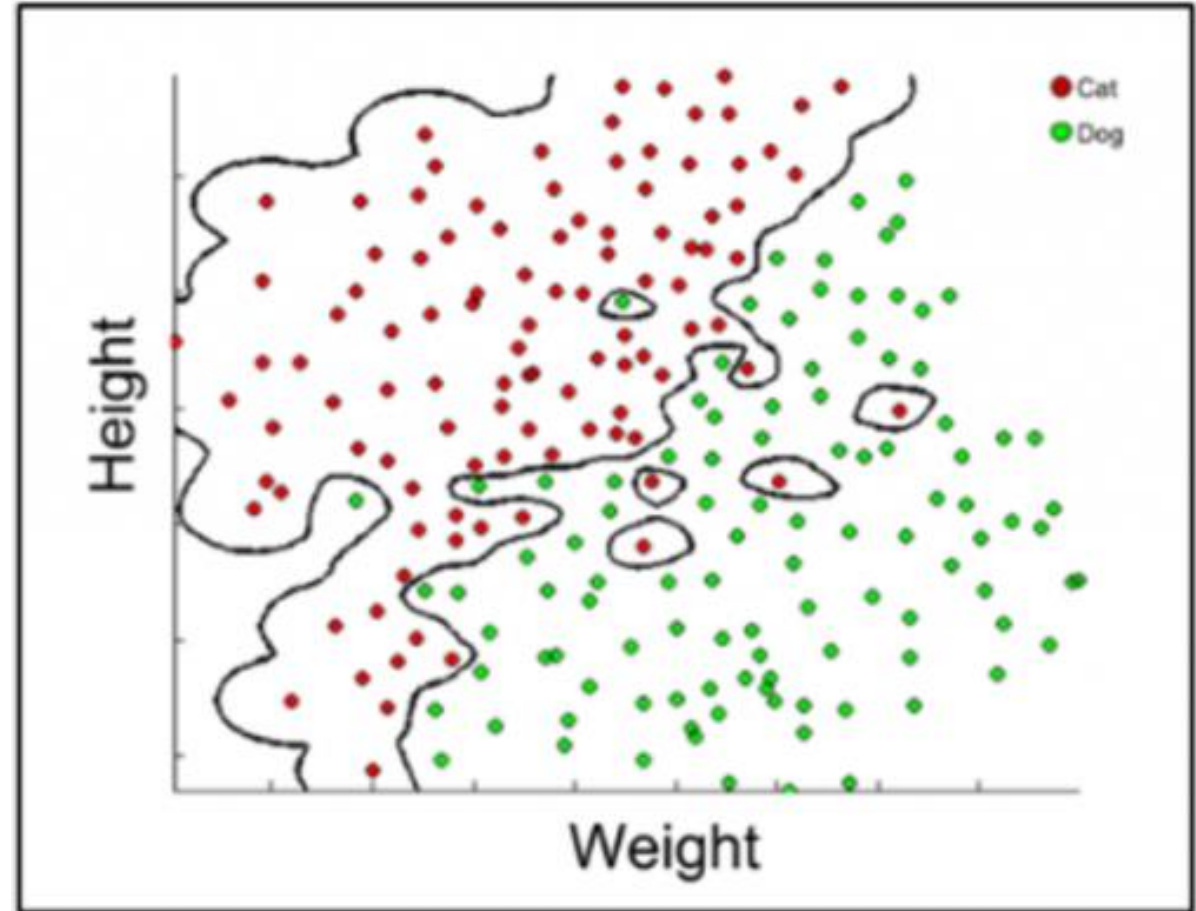


using a straight line equation of the form:
$$y = mx + c$$

NEURAL NETWORKS .VS. TRADITIONAL COMPUTING

Neural Networks

- **Complex Model**
- manages to **successfully partition the dots**, including weird outliers.
- incredibly hard to write a function to describe this 'line' using conventional computing
- **can handle many more features** to learn a complex, robust, multidimensional model.



using a straight line equation of the form:
 $y = f(x)$

WHAT HAPPENS IN A NEURON?

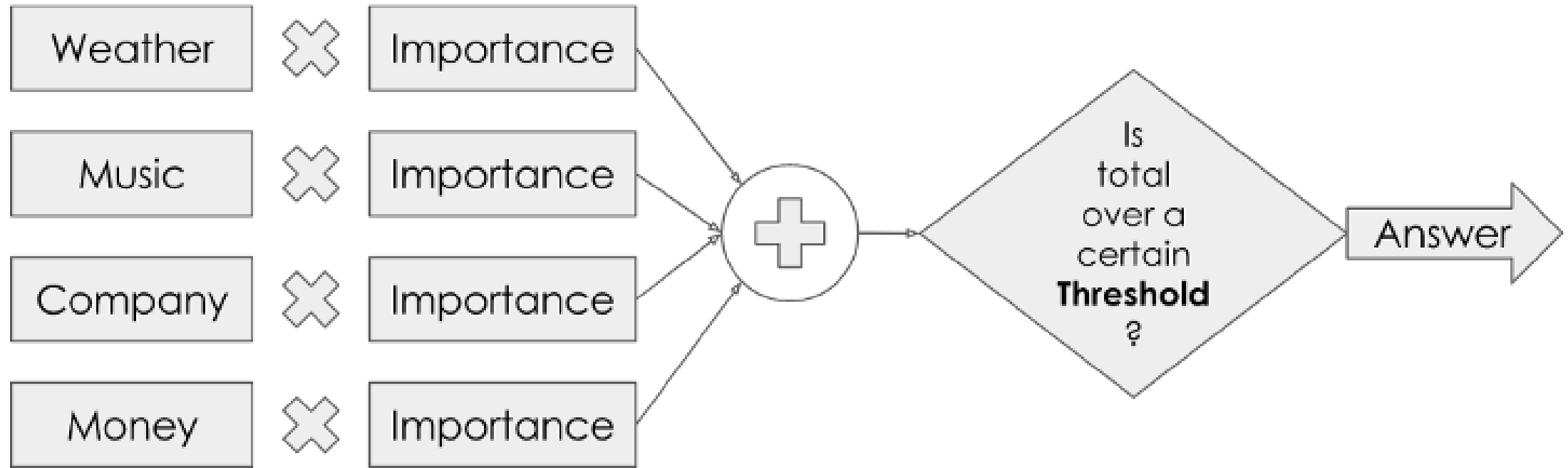
Problem: Whether to go to a cricket match or not?

- Will the weather be nice?
- What's the music like?
- Do I have anyone to go with?
- Can I afford it?
- Do I need to write my exam?
- Will I like the food in the food stalls at stadium?

**Just take the first four
for simplicity**

WHAT HAPPENS IN A NEURON?

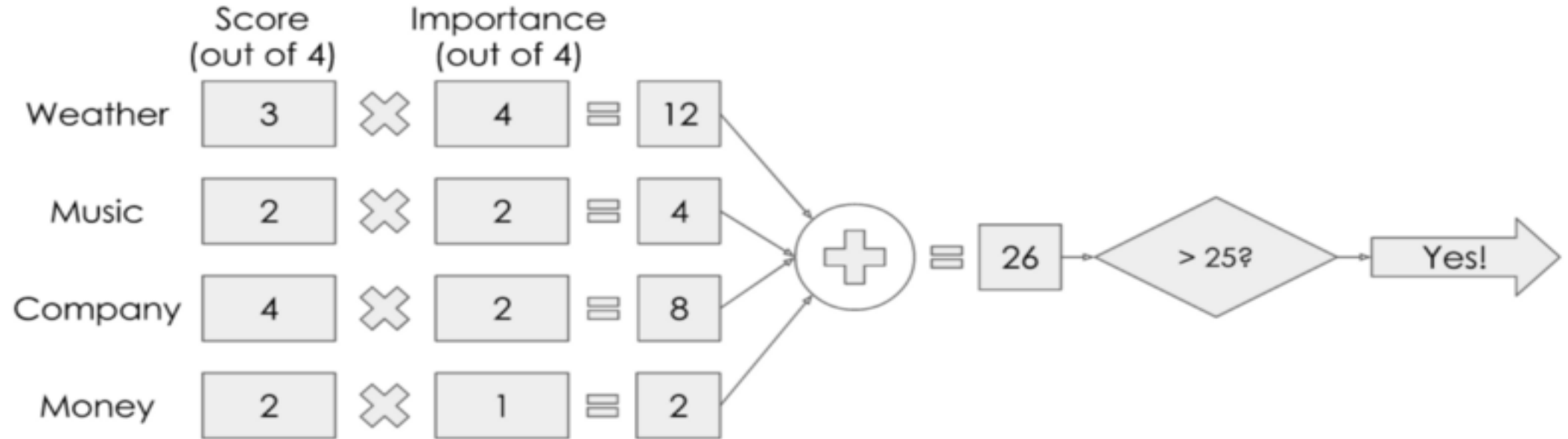
Problem: Whether to go to a cricket match or not?



To make my decision, **I would consider how important each factor was to me, weigh them** all up, and see if the result was over a certain threshold. If so, I will go to the match! It's a bit like the process we often use of **weighing up pros and cons to make a decision**.

WHAT HAPPENS IN A NEURON?

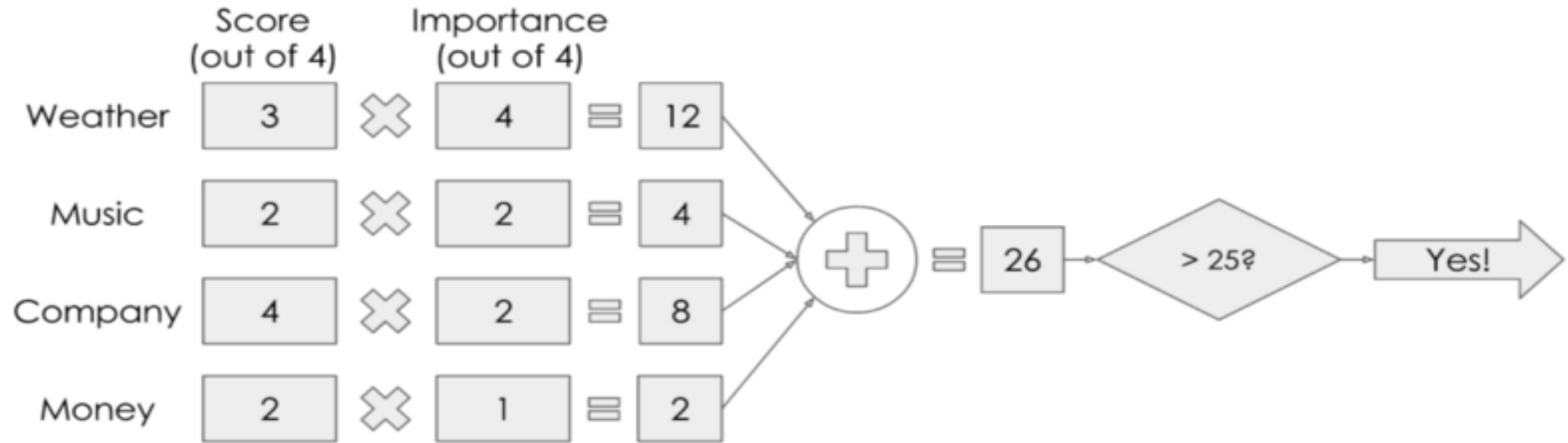
Problem: Whether to go to a cricket match or not?



Give some weights...and scores...

WHAT HAPPENS IN A NEURON?

Problem: Whether to go to a cricket match or not?

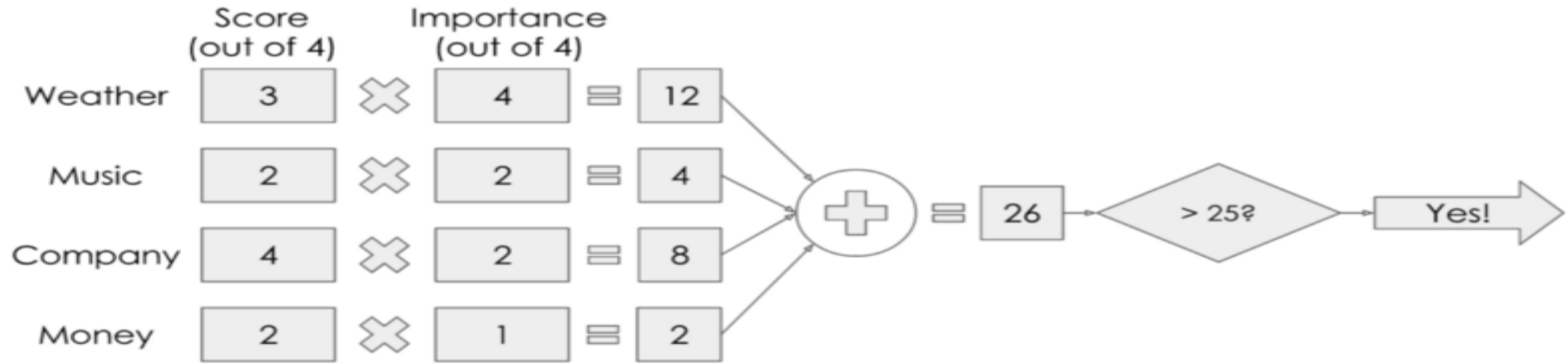


Scores:

- **weather** is looking pretty good but not perfect - 3 out of 4.
- **music** is ok but not my favourite - 2 out of 4.
- **Company** - my best friend has said she'll come with me so I know the company will be great, - 4 out of 4.
- **Money** - little pricey but not completely unreasonable - 2 out of 4.

WHAT HAPPENS IN A NEURON?

Problem: Whether to go to a cricket match or not?

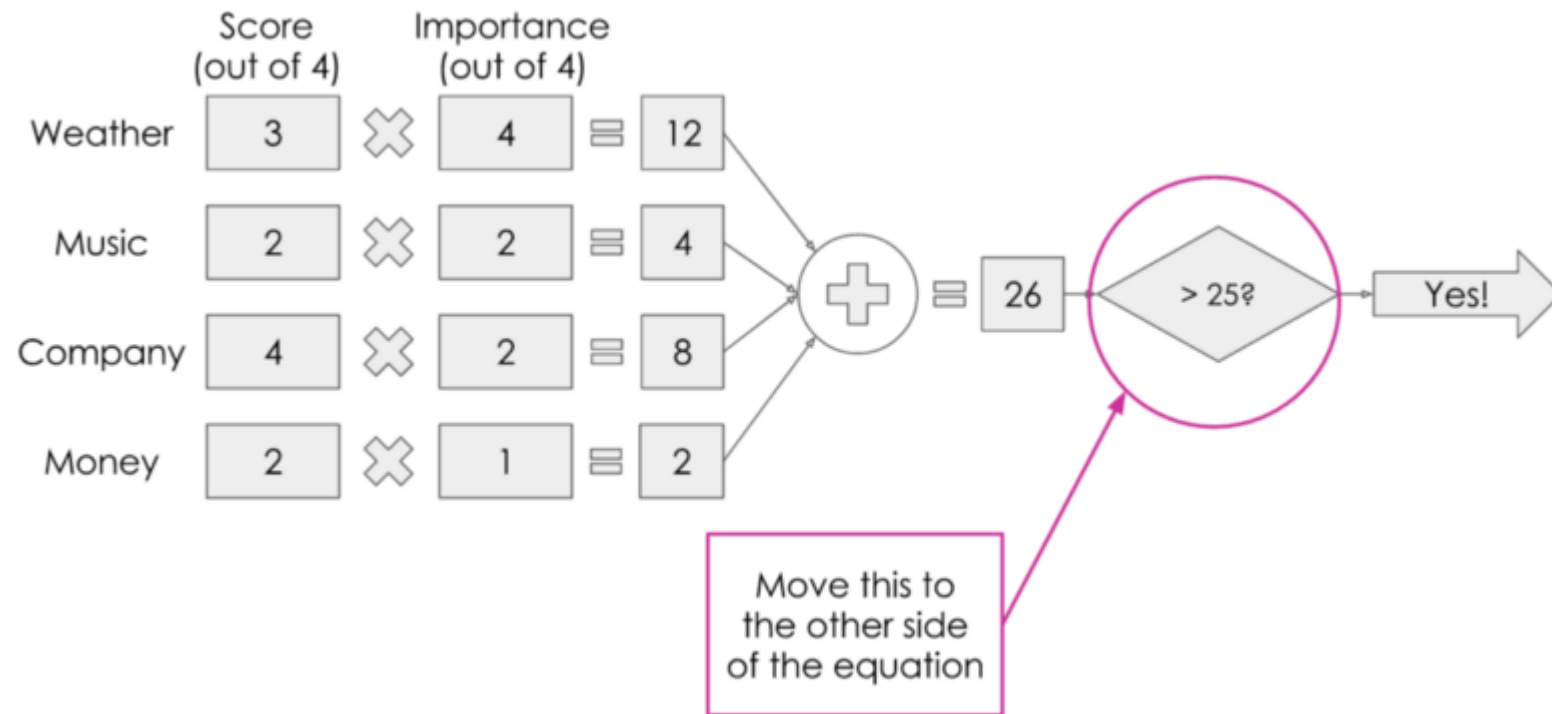


Now for the importance:

- **Weather** - I really want to go if it's sunny, very imp - I give it a full **4 out of 4**.
- **Music** - I'm happy to listen to most things, not super important - **2 out of 4**.
- **Company** - I wouldn't mind too much going to the match on my own, so company can have a **2 out of 4** for importance.
- **Money** - I'm not too worried about money, so I can give it just a **1 out of 4**.

WHAT HAPPENS IN A NEURON?

Problem: Whether to go to a cricket match or not?



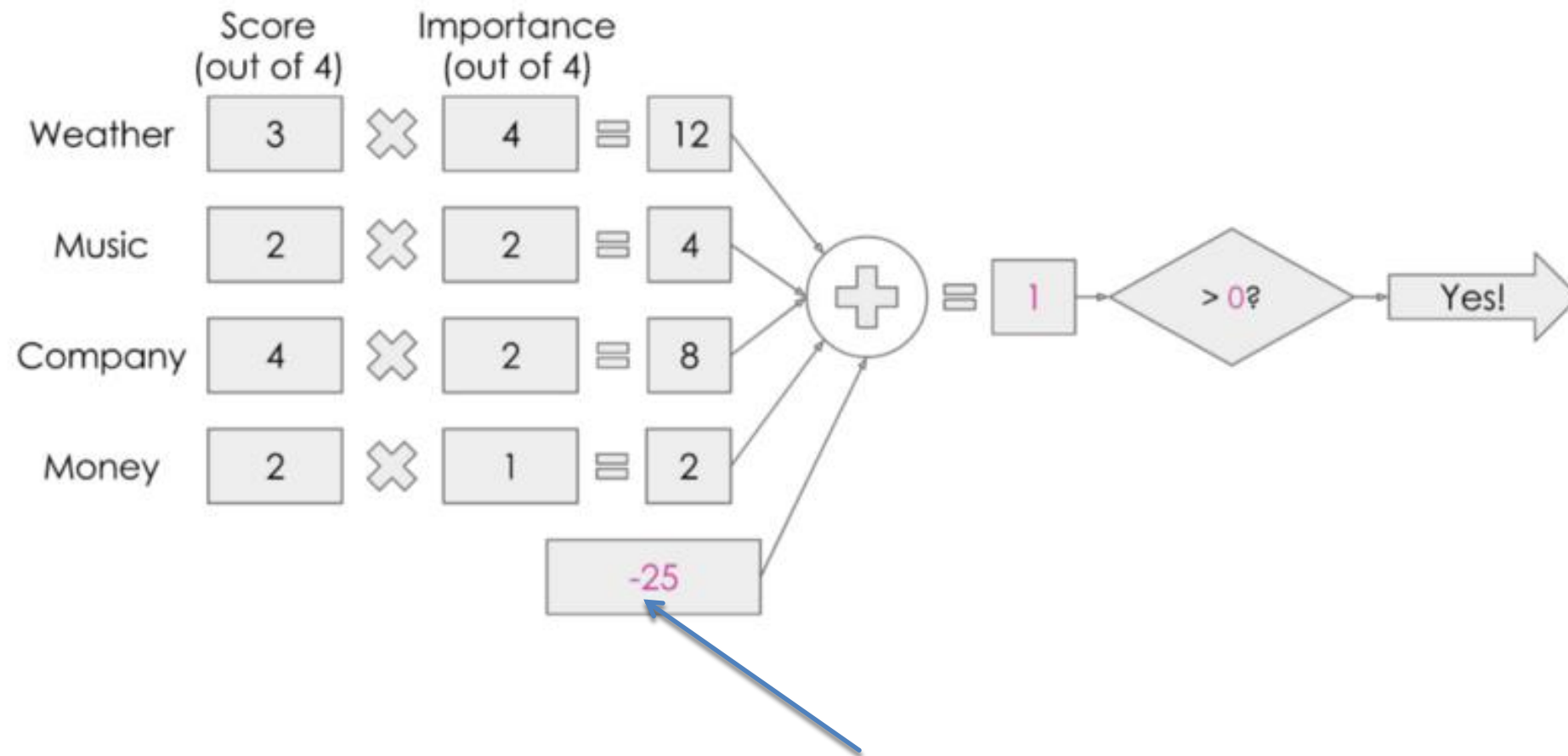
Total Score = Sum of (Score x Weight) = 26

Threshold = 25

Score (26) > Threshold (25) → Going to Cricket Match

WHAT HAPPENS IN A NEURON?

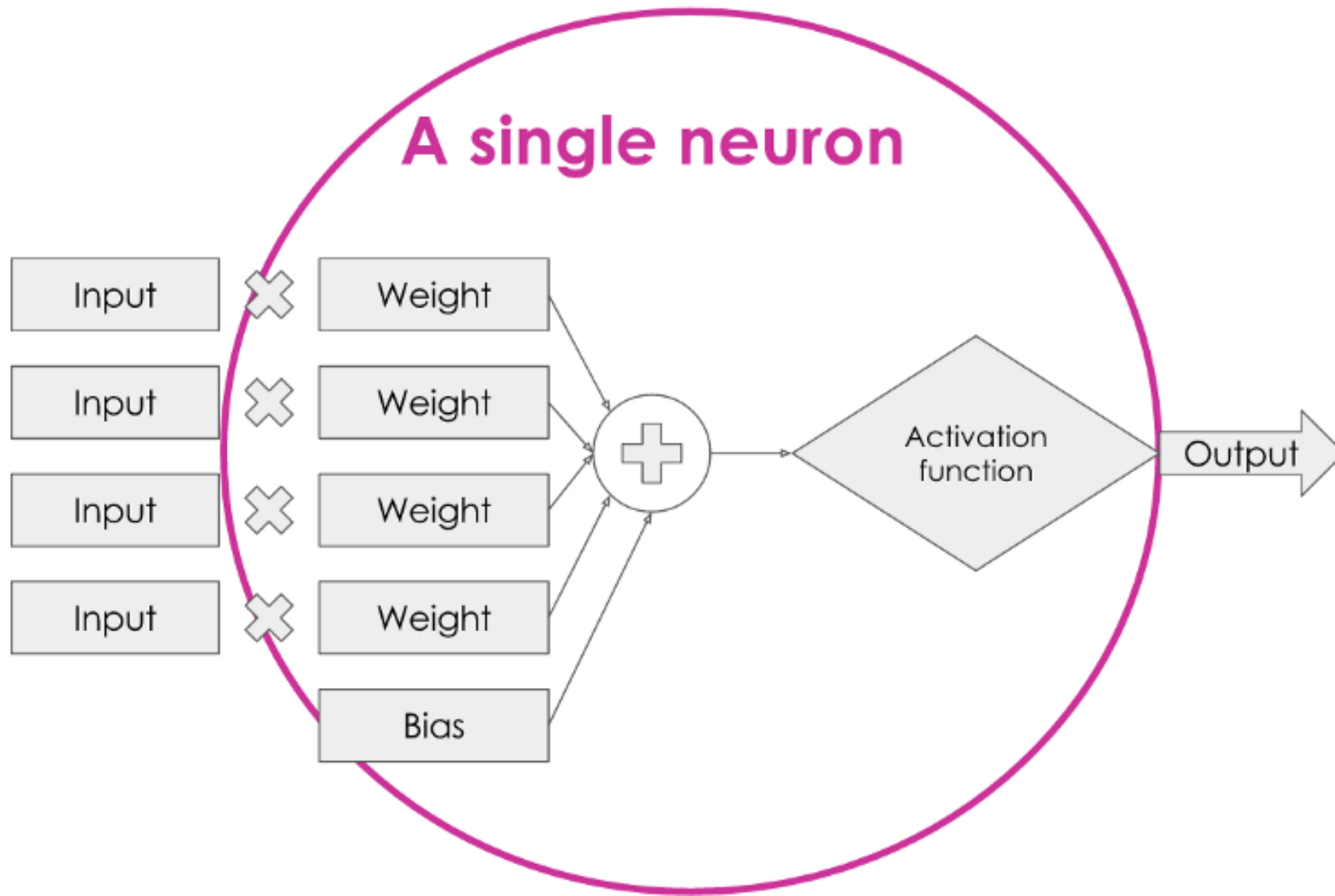
Problem: Whether to go to a cricket match or not?



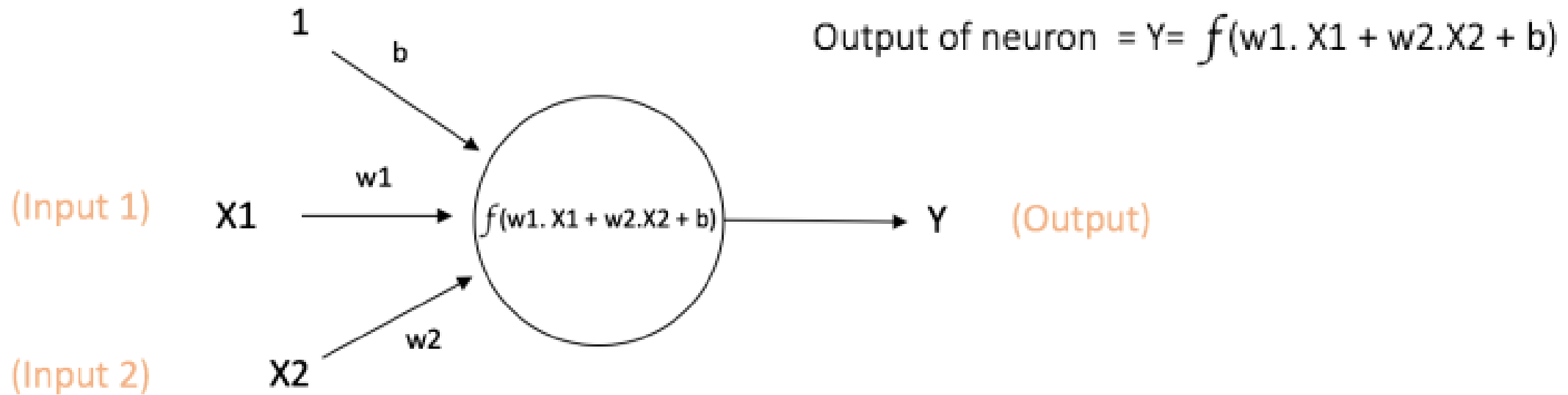
This number is what is known as the **'bias'**

WHAT HAPPENS IN A NEURON?

A general form of a Neuron/Single Perceptron

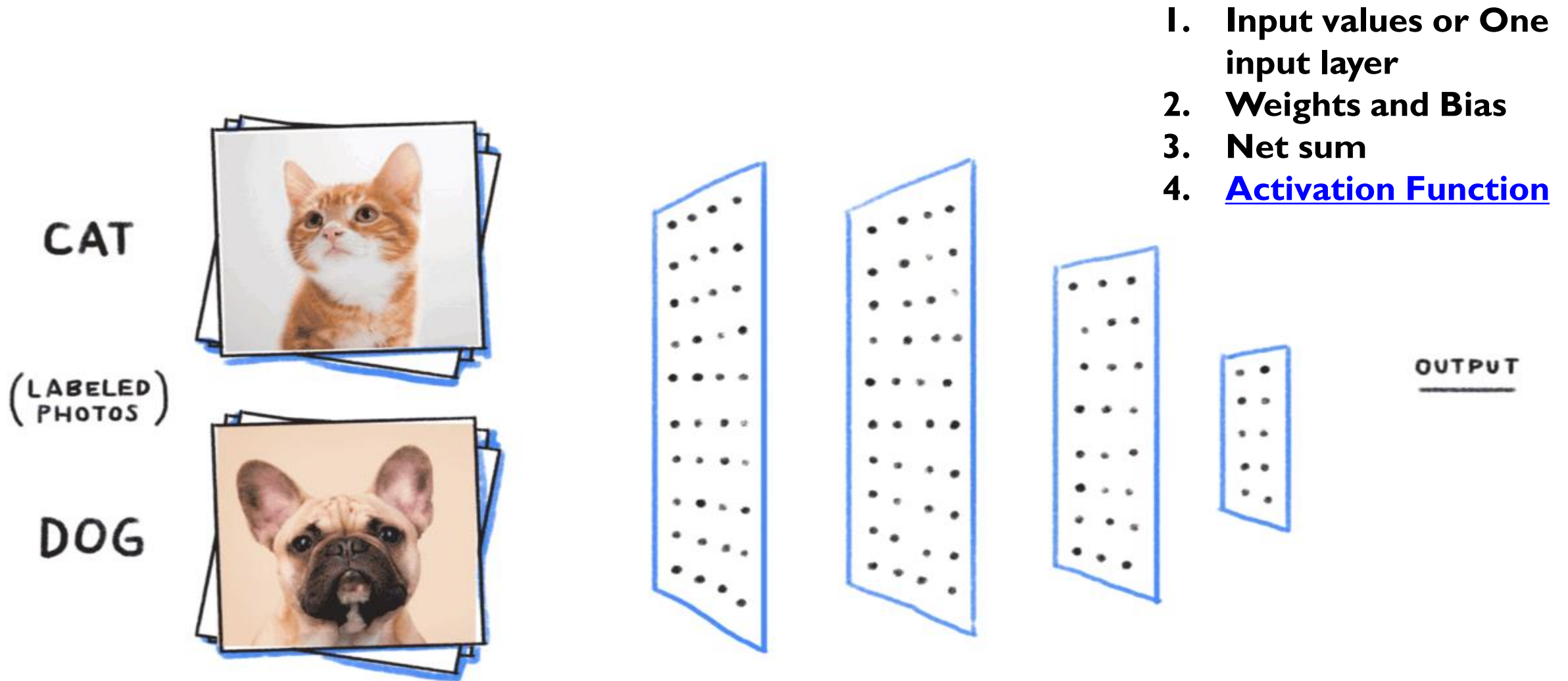


A PERCEPTRON



- The basic unit of computation in a neural network is the **neuron**, often called a **node** or **unit**.
- receives input from some other nodes, or from an external source and computes an output.
- Each input has an associated **weight** (w), which is assigned on the basis of its relative importance
- The node applies a function **f** (defined below) to the weighted sum of its inputs

A PERCEPTRON



- *A multi-layer perceptron is called Neural Network.*

A PERCEPTRON...HOW IT WORKS?

I. All the inputs x are multiplied with their weights w . Let's call it k .

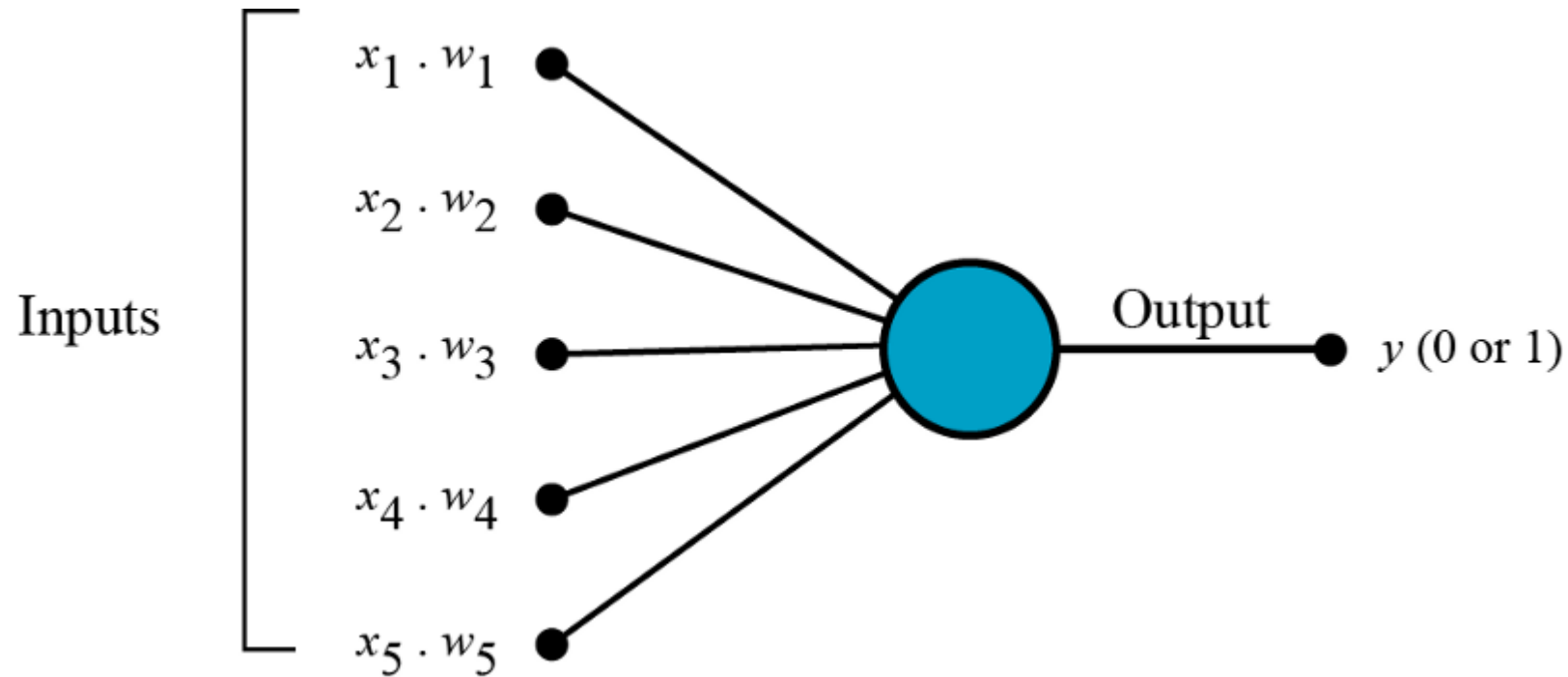


Fig: Multiplying inputs with weights for 5 inputs

A PERCEPTRON...HOW IT WORKS?

2. **Add** all the multiplied values and call them **Weighted Sum**.

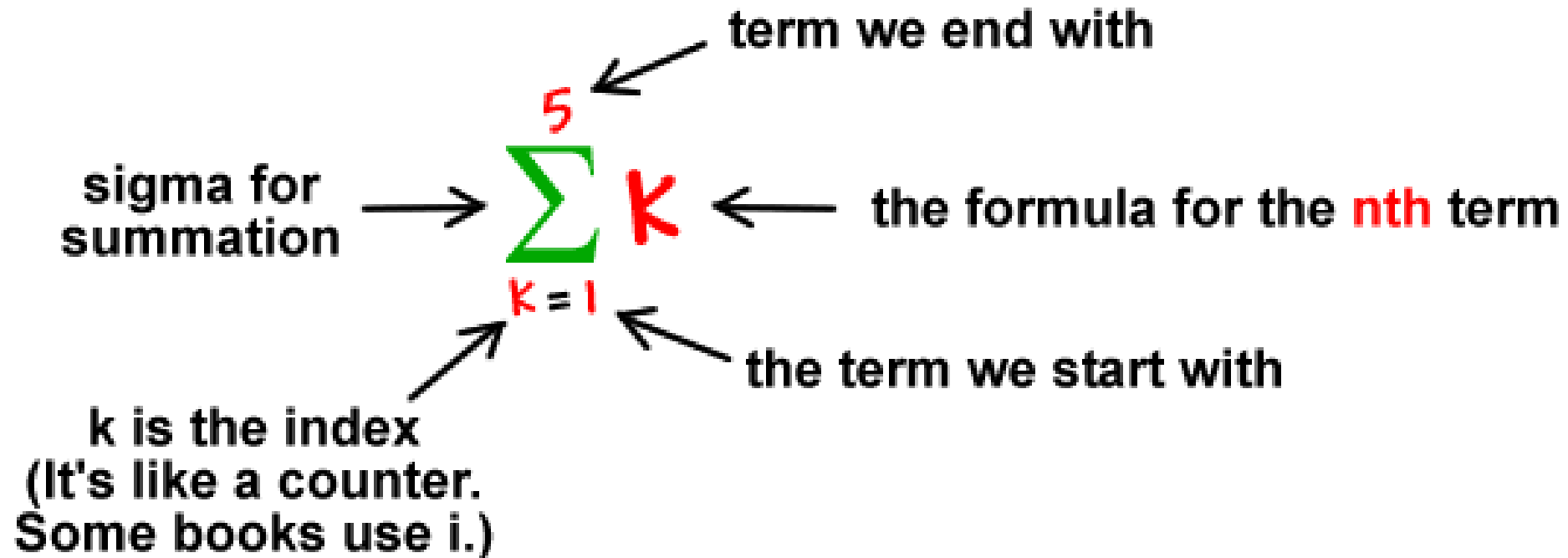
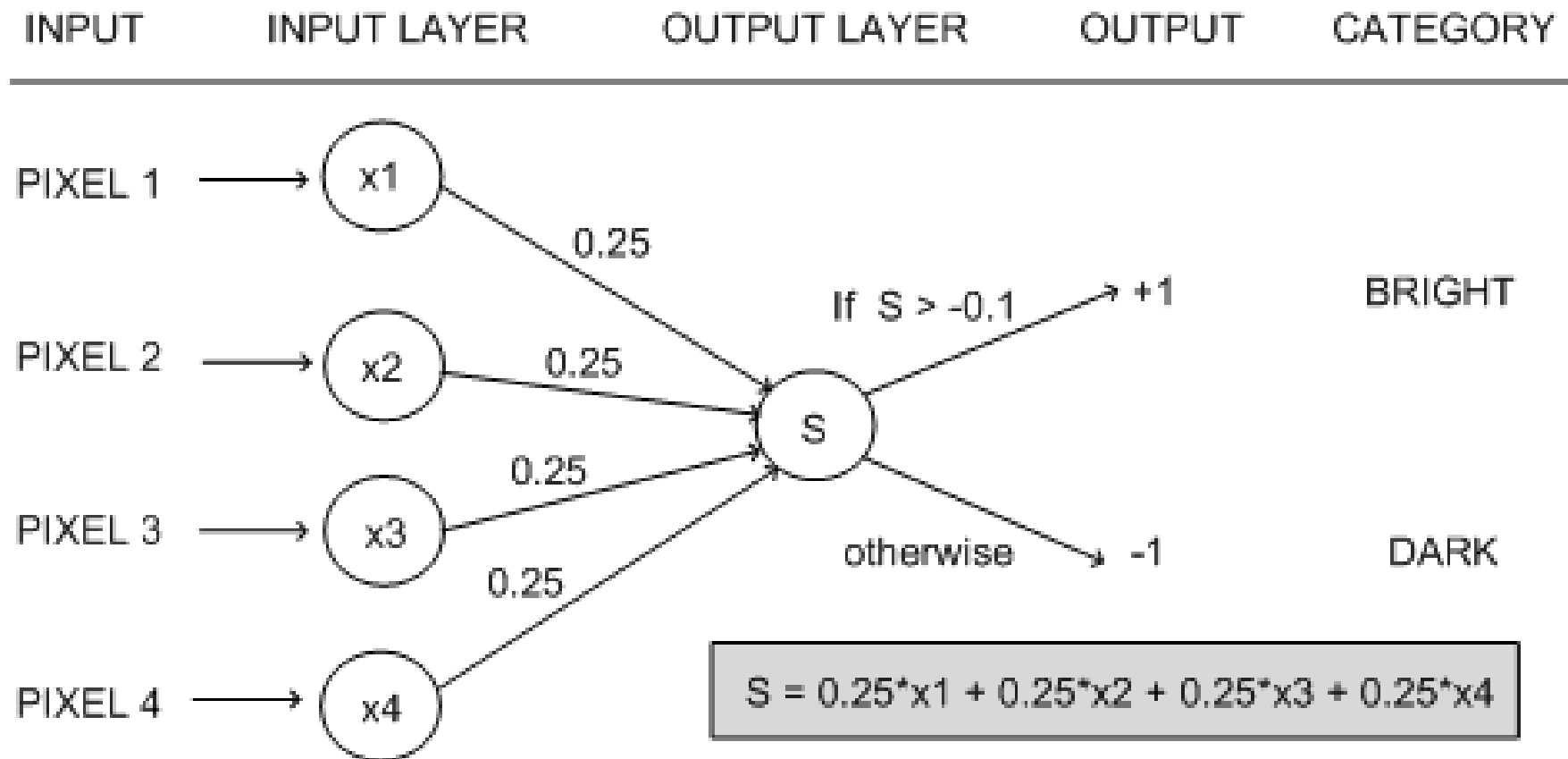


Fig:Adding with Summation

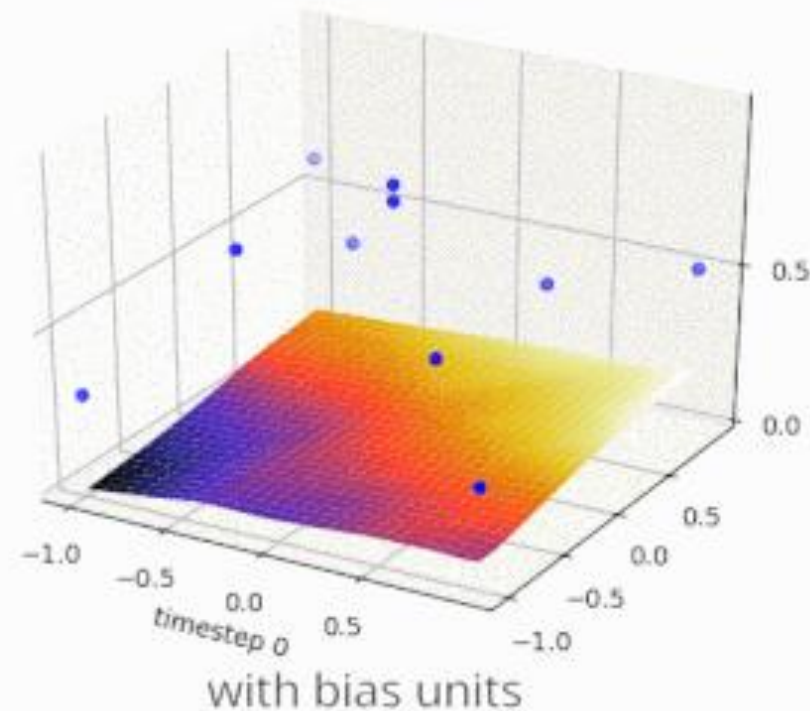
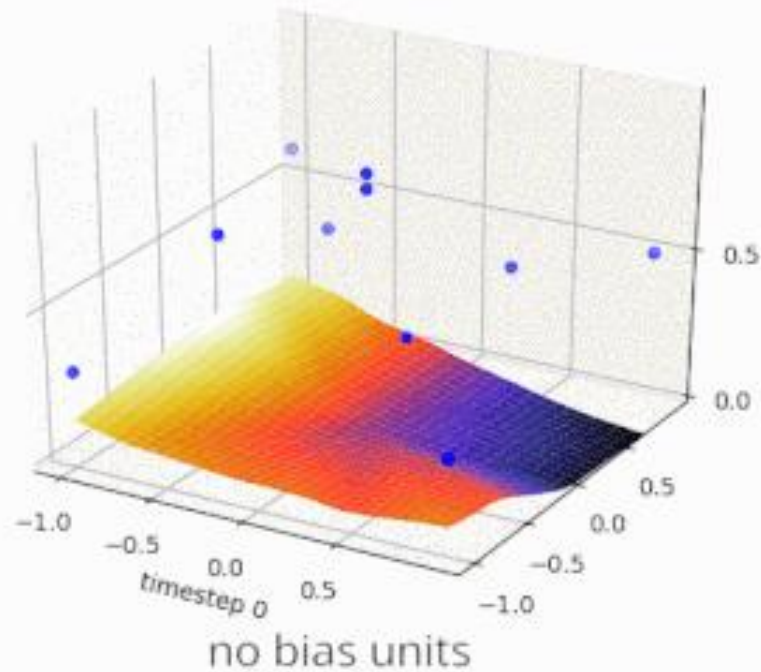
WHY DO WE NEED WEIGHTS AND BIAS?

- *Weights shows the strength of the particular node/input.*
- *A bias value allows you to shift the activation function to the left or right.*



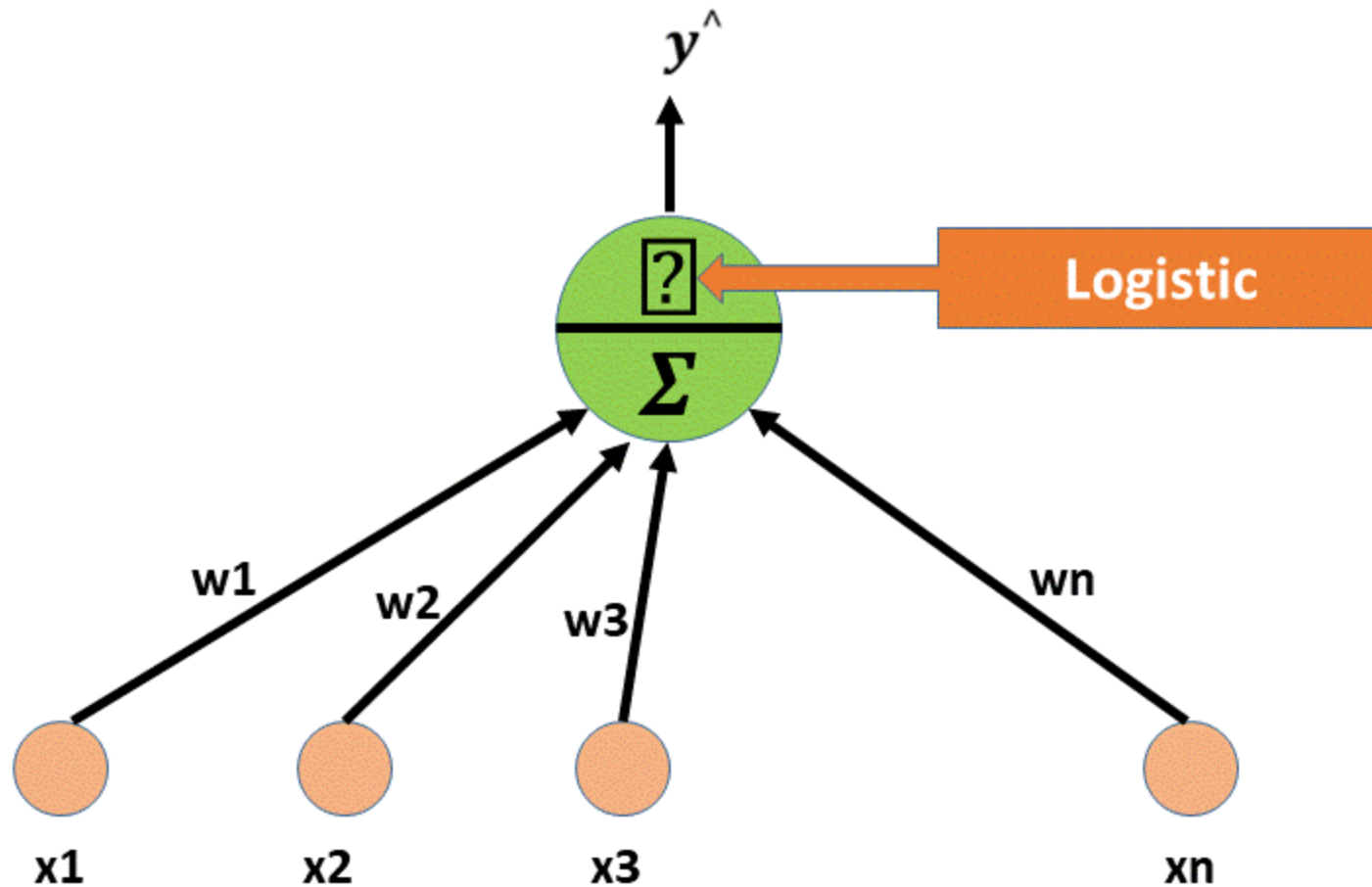
WHY DO WE NEED WEIGHTS AND BIAS?

- *Weights shows the strength of the particular node/input.*
- *A bias value allows you to shift the activation function to the left or right.*



without the bias the ReLU-network is not able to deviate from zero at (0,0).

Why do we need an activation function??



Why do we need an activation function??

Activation functions and neural networks are one of the beautiful ideas inspired from humans. When we feed lots of information to our brain, it tries hard to understand and classify the information between useful and not so useful information. In the same way, we need similar mechanism to classify the incoming information as useful and not useful in case of neural networks.

Only some part of information is much useful and rest may be some noise.

Network tries to learn the useful information. For that, we need activation functions.

Activation function helps the network in doing segregation. In simpler words, activation function tries to build the wall between useful



Why do we need an activation function??

Activation functions and neural networks are one of the beautiful ideas inspired from humans. When we feed lots of information to our brain, it tries hard to understand and classify the information between useful and not so useful information. In the same way, we need similar mechanism to classify the incoming information as useful and not useful in case of neural networks.

Only some part of information is much useful and rest may be some noise.

Network tries to learn the useful information. For that, we need activation functions.

Activation function helps the network in doing segregation. In simpler words, activation function tries to build the wall between useful

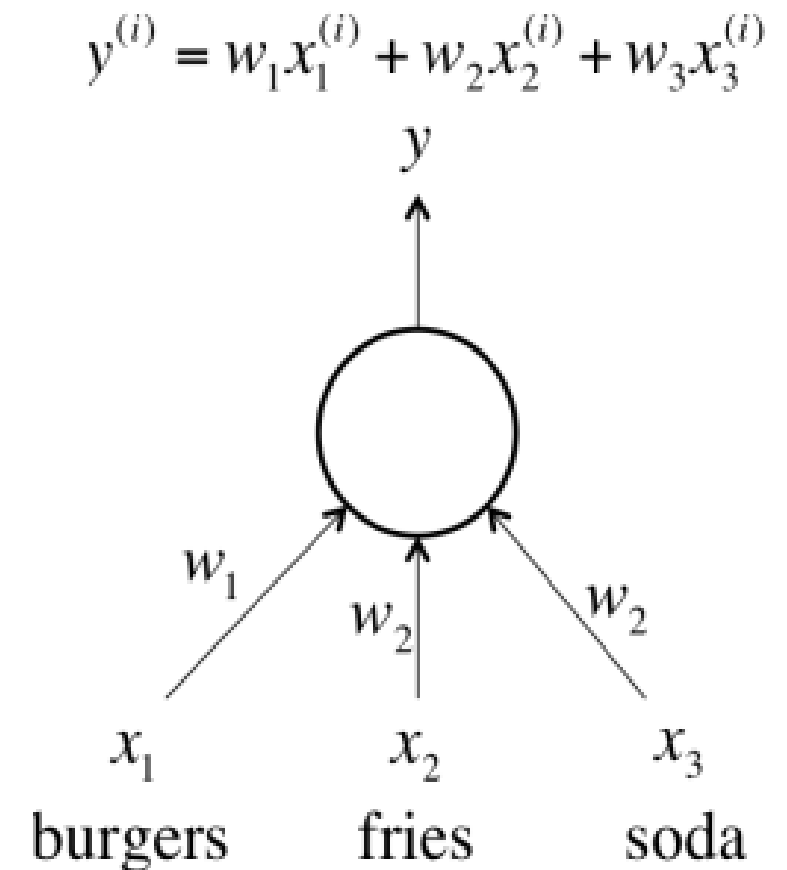


WHY DO WE NEED “ACTIVATION FUNCTION”?

For example, if we simply want to estimate the cost of a meal, we could use a neuron based on a linear function $f(z) = wx + b$. Using $(z) = z$ and weights equal to the price of each item, the linear neuron would take in the number of burgers, fries, and sodas and output the price of the meal.

The above kind of **linear neuron** is very easy for computation.

However, by just having an interconnected set of linear neurons, **we won't be able to learn complex relationships**: taking multiple linear functions and combining them still gives us a linear function, and we could represent the whole network by a simple matrix transformation.



WHY DO WE NEED “ACTIVATION FUNCTION”?

Real world data is mostly non-linear, so this is where the Activation functions comes for help.

Activation functions introduce non-linearity into the output of a neuron, and this ensures that **we can learn more complex functions by approximating every non-linear function as a linear combination of a large number of non-linear functions.**

Also, activation functions help us to **limit the output** to a certain finite value.

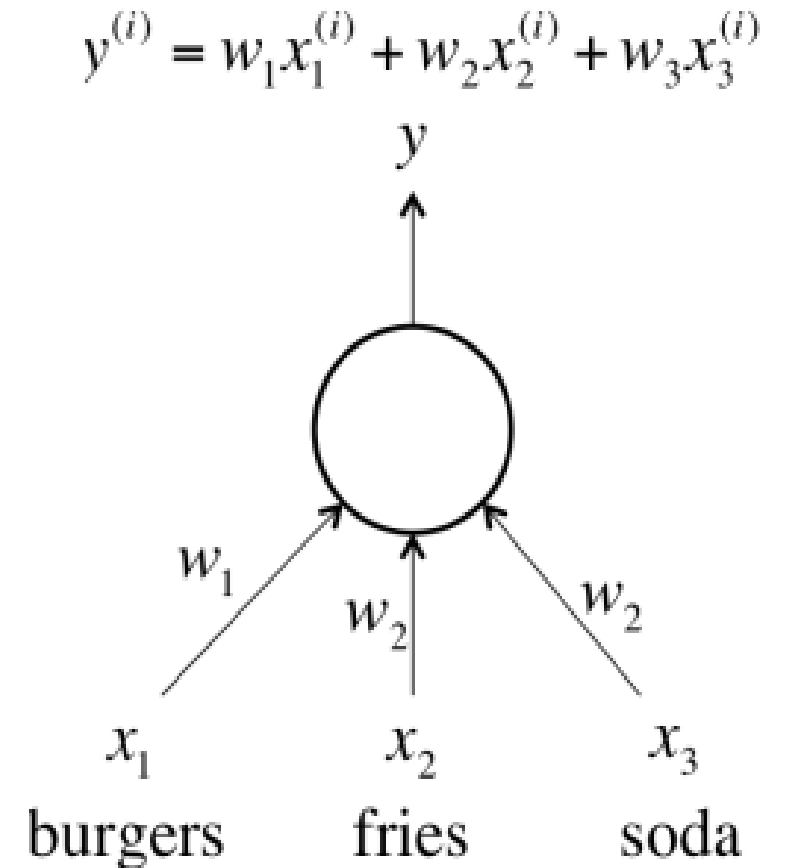


Image credits: Fundamentals of Deep Learnings

WHY DO WE NEED “ACTIVATION FUNCTION”?

- *Their main purpose is to **convert a input signal of a node in a A-NN to an output signal.***
- *In short, the activation functions are used to **map the input between the required values like (0, 1) or (-1, 1).***
- *It is also known as **Transfer Function.***
- *They **introduce non-linear properties** to our Network.*
- *That output signal now is used as a input in the next layer in the stack.*

The Activation Functions can be basically divided into 2 types-

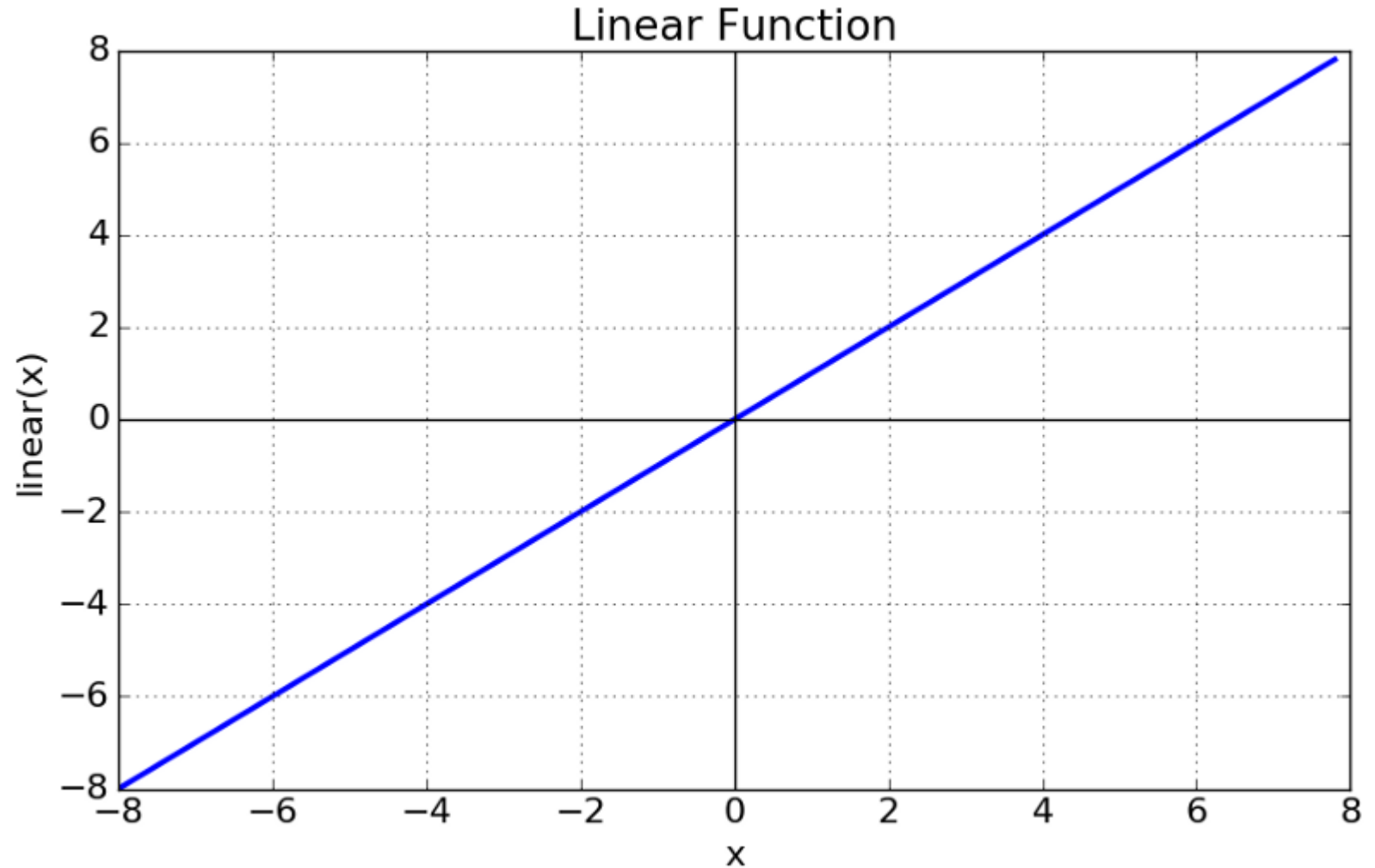
- Linear Activation Function
- Non-linear Activation Functions

LINEAR ACTIVATION FUNCTION

Equation : $f(x) = x$

Range : (-infinity to infinity)

It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.



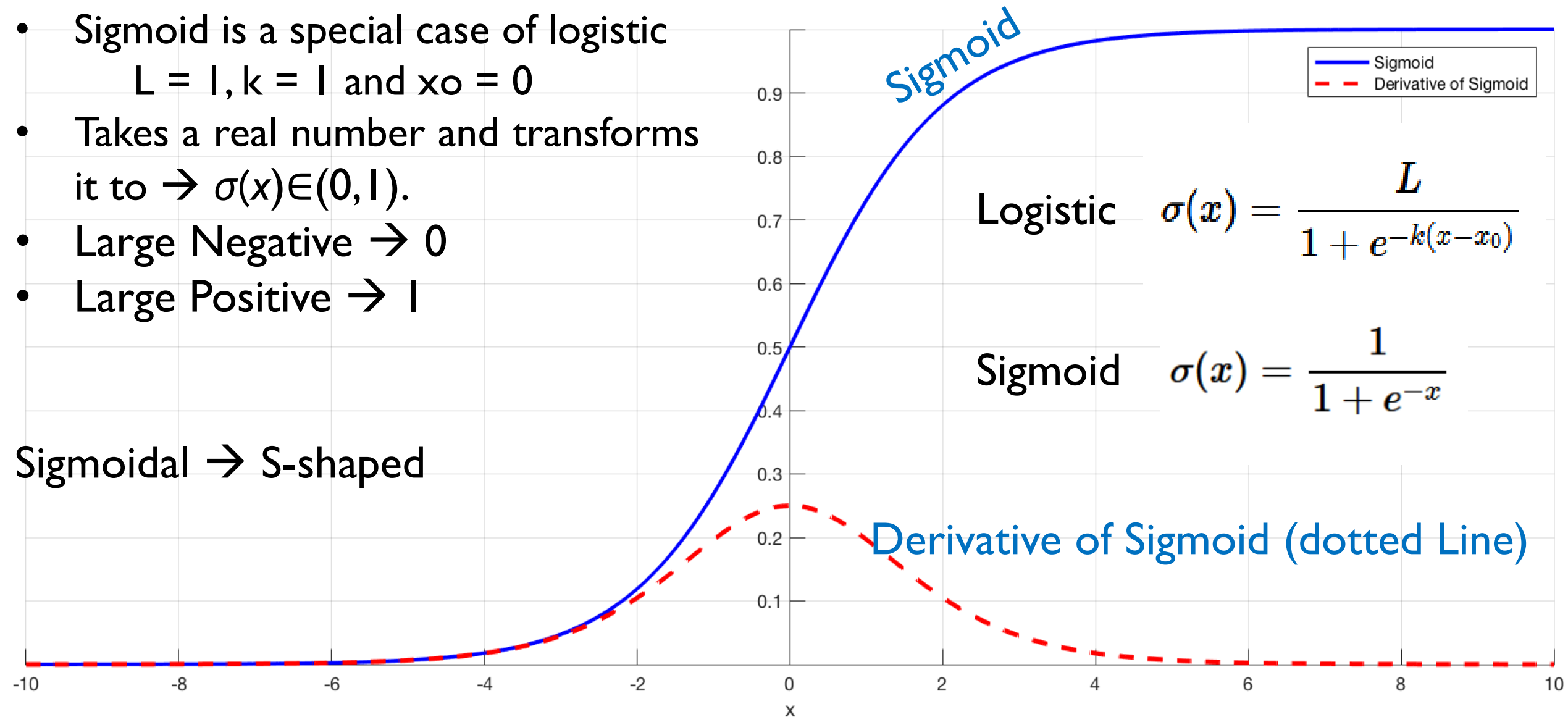
WHY CAN'T WE DO IT WITHOUT ACTIVATING THE INPUT SIGNAL?

- Linear Activation functions are just a function of a polynomial of Degree 1.
- They have less power to solve complex problems.
- A NN with linear activation function can be treated as a linear regression model
- without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.
- **Examples:**
 - Sigmoid or Logistic
 - *Tanh - Hyperbolic tangent*
 - *ReLu - Rectified linear units*

SIGMOID

- Sigmoid is a special case of logistic
 $L = 1, k = 1$ and $x_0 = 0$
- Takes a real number and transforms it to $\rightarrow \sigma(x) \in (0, 1)$.
- Large Negative $\rightarrow 0$
- Large Positive $\rightarrow 1$

Sigmoidal \rightarrow S-shaped



SIGMOID – DISADVANTAGES

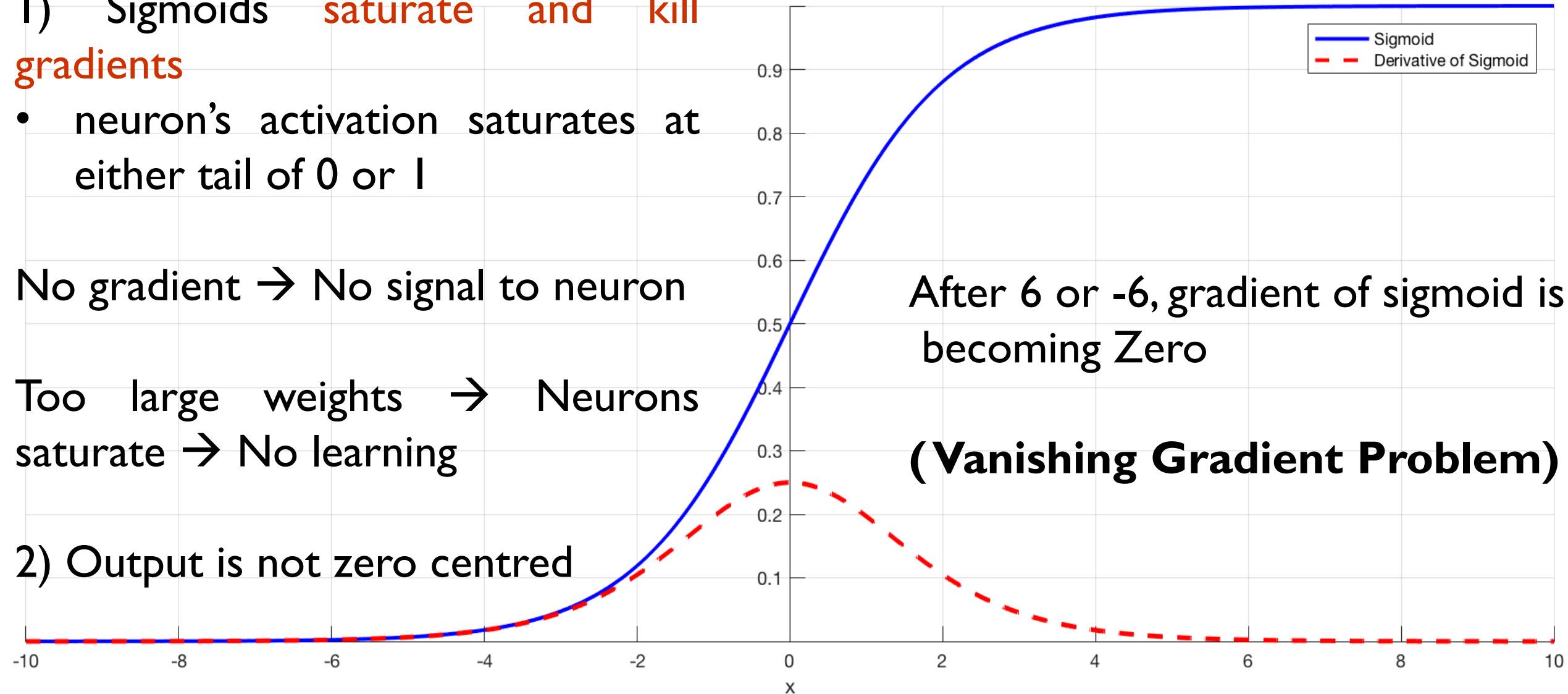
1) Sigmoids **saturate and kill gradients**

- neuron's activation saturates at either tail of 0 or 1

No gradient \rightarrow No signal to neuron

Too large weights \rightarrow Neurons saturate \rightarrow No learning

2) Output is not zero centred



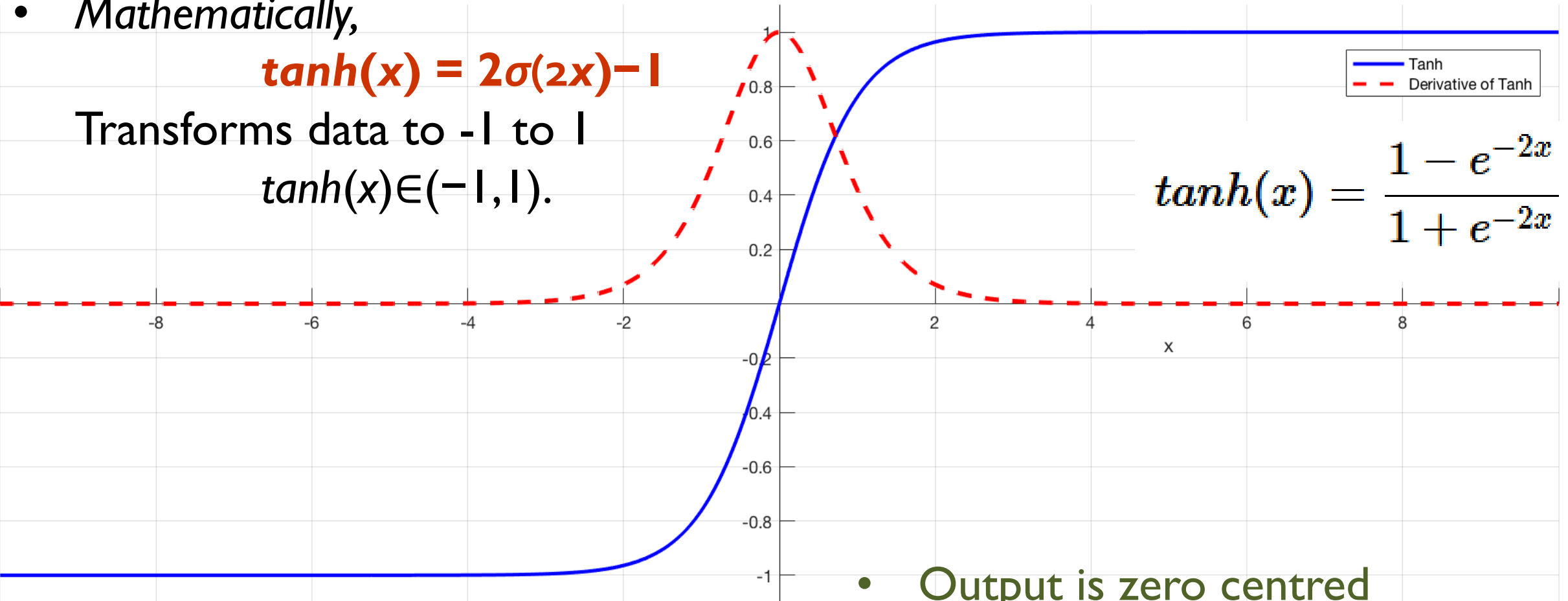
HYPERBOLIC TANGENT FUNCTION - TANH

- It is also sigmoidal
- Mathematically,

$$\tanh(x) = 2\sigma(2x) - 1$$

Transforms data to -1 to 1

$$\tanh(x) \in (-1, 1).$$



$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

- Output is zero centred
- It also saturates and kills gradient

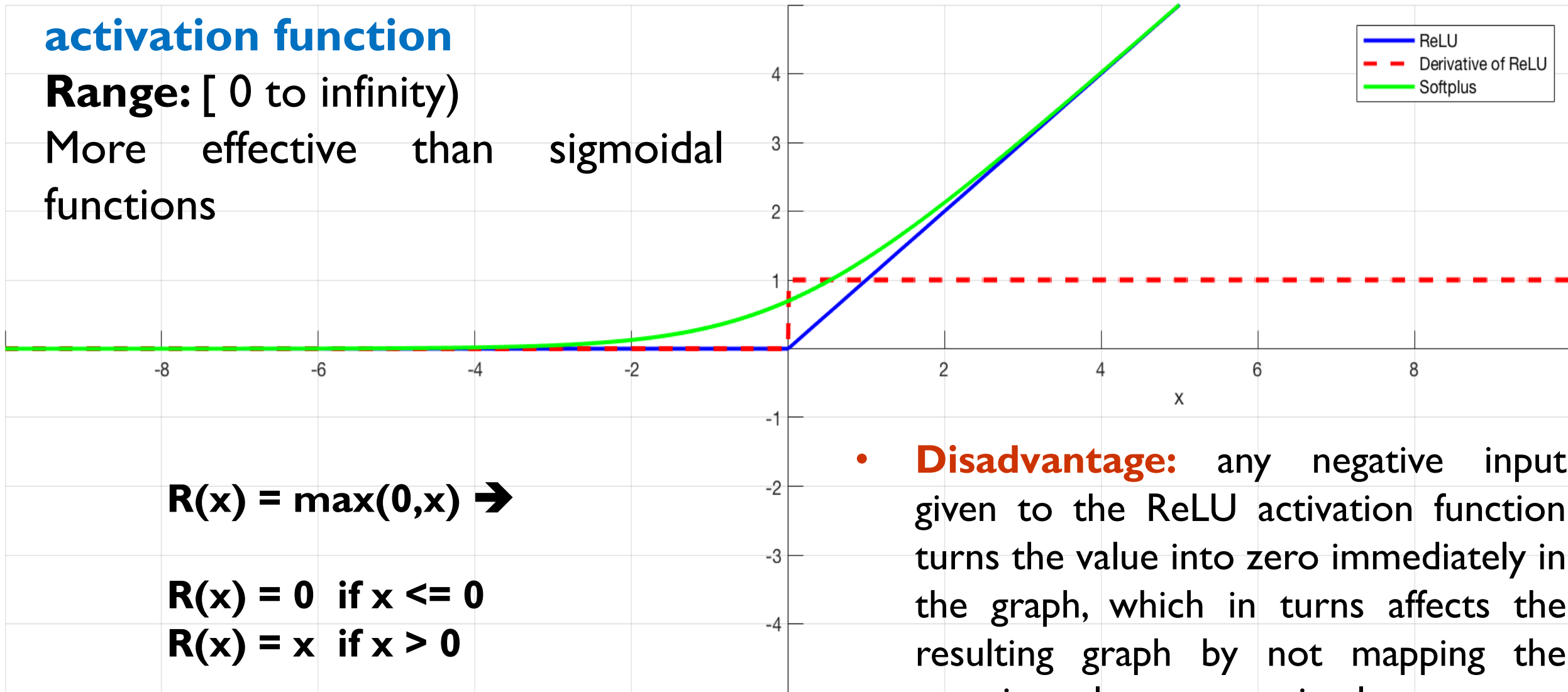
RELU (RECTIFIED LINEAR UNIT) ACTIVATION FUNCTION

- The ReLU is the most used activation function
- Range: [0 to infinity)
- More effective than sigmoidal functions

$$R(x) = \max(0, x) \rightarrow$$

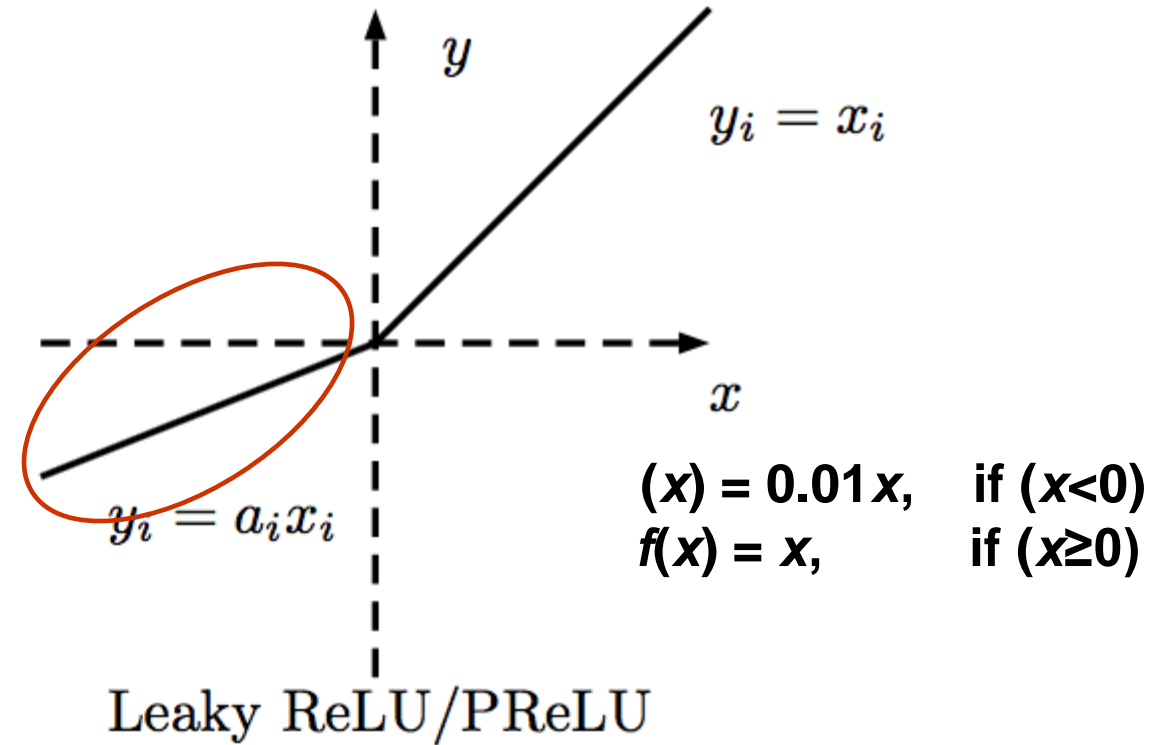
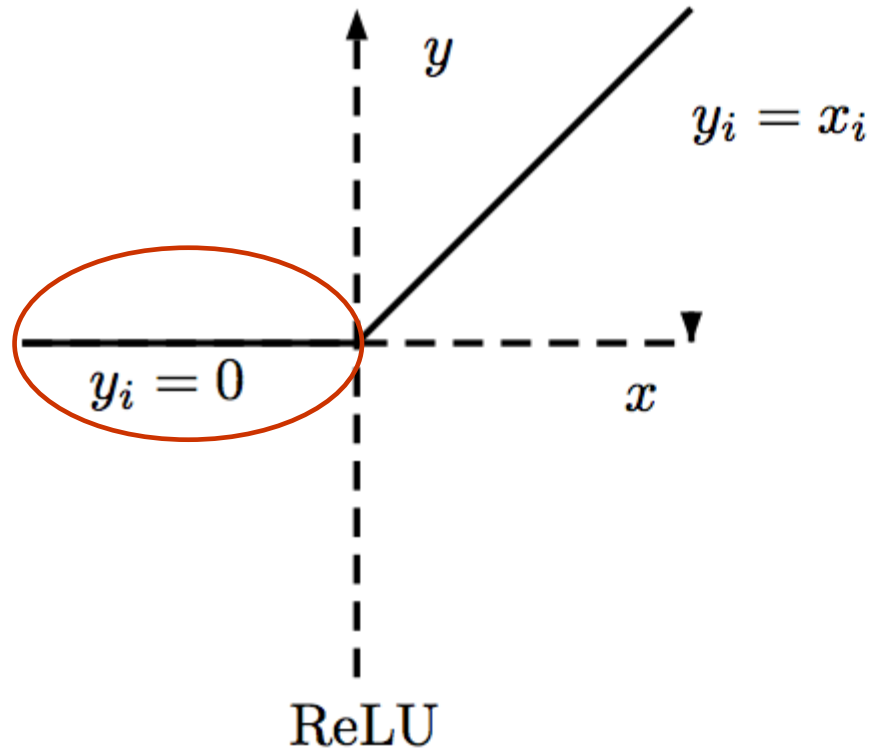
$$R(x) = 0 \quad \text{if } x \leq 0$$

$$R(x) = x \quad \text{if } x > 0$$



- **Disadvantage:** any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

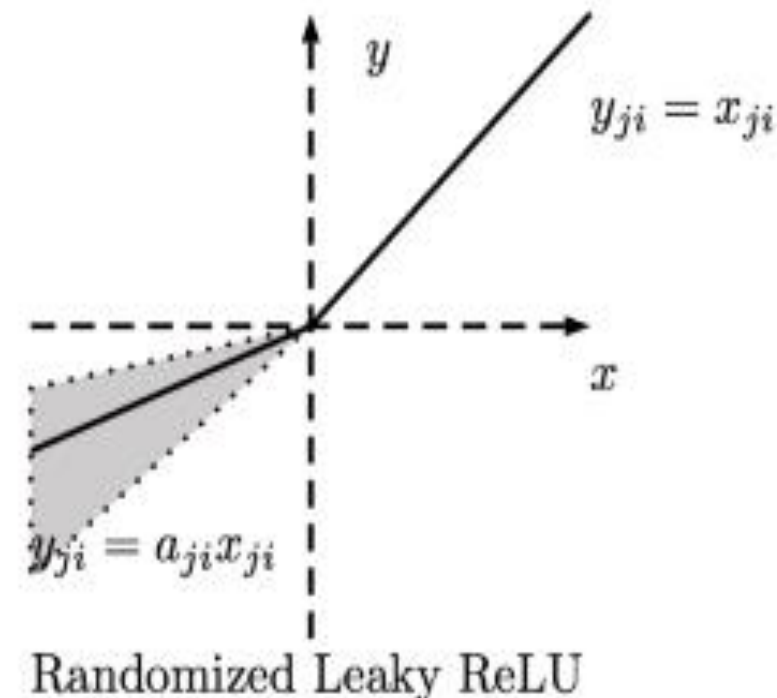
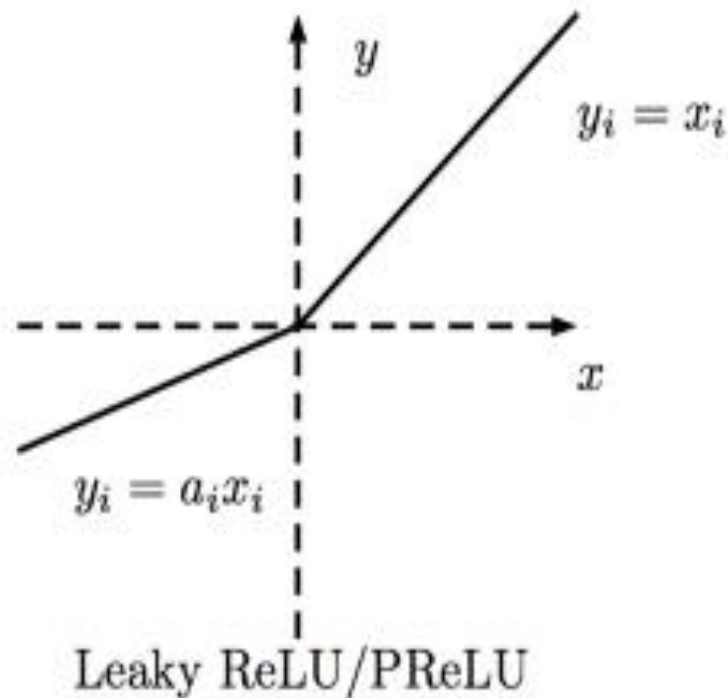
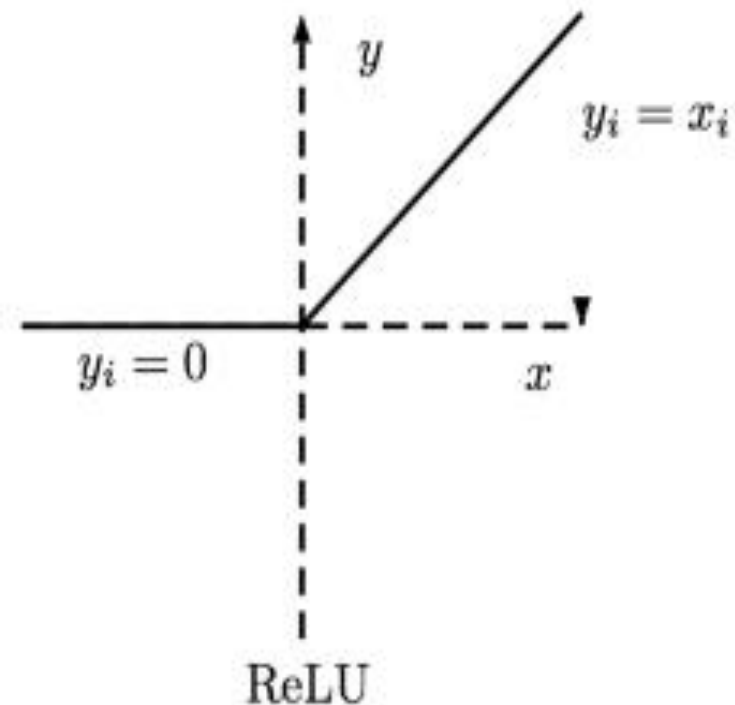
LEAKY RELU



fix the “dying ReLU” problem by
introducing a
Small negative slope (0.01 or so)

RANDOMIZED RELU

- Fix the problem of dying neurons
- It introduces a small slope to keep the updates alive.
- The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.
- **When a is not 0.01 then it is called Randomized ReLU.**

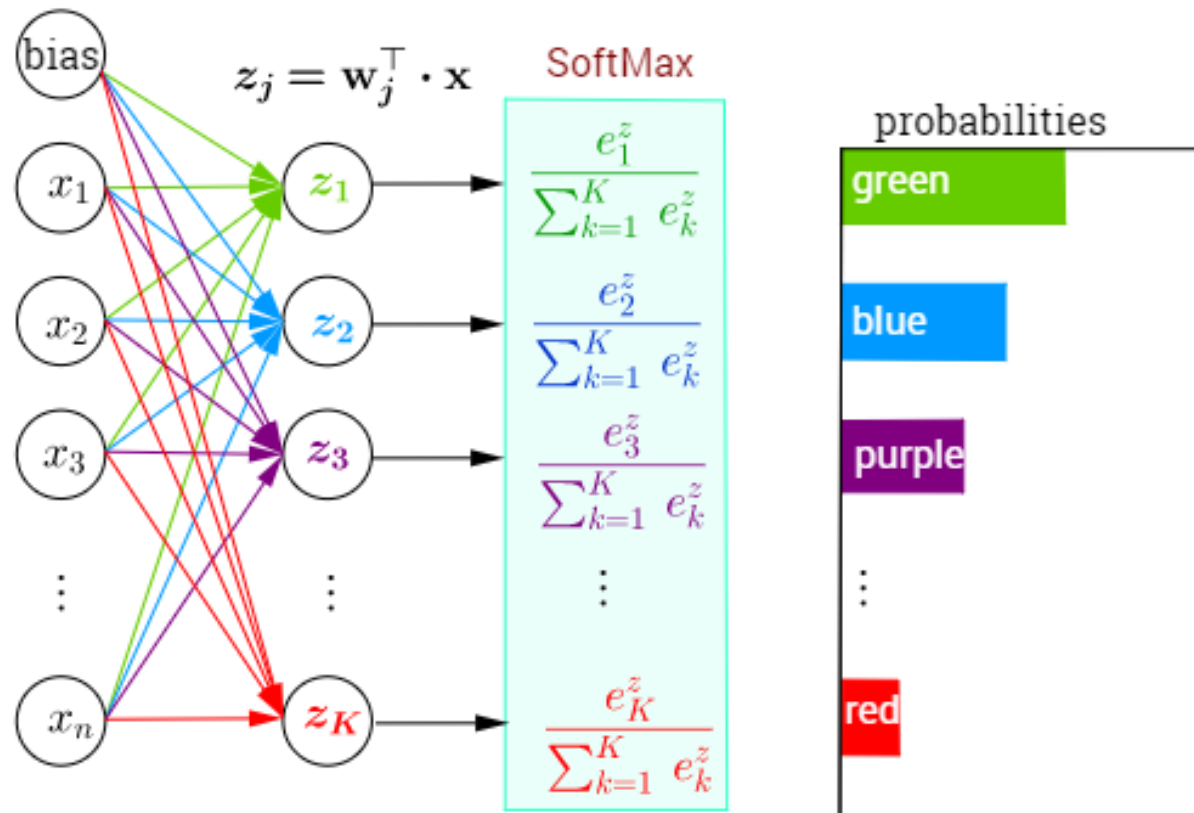


SOFTMAX

- Transforms a real valued vector to $[0, 1]$
- Output of softmax is the list of probabilities of k different classes/categories

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



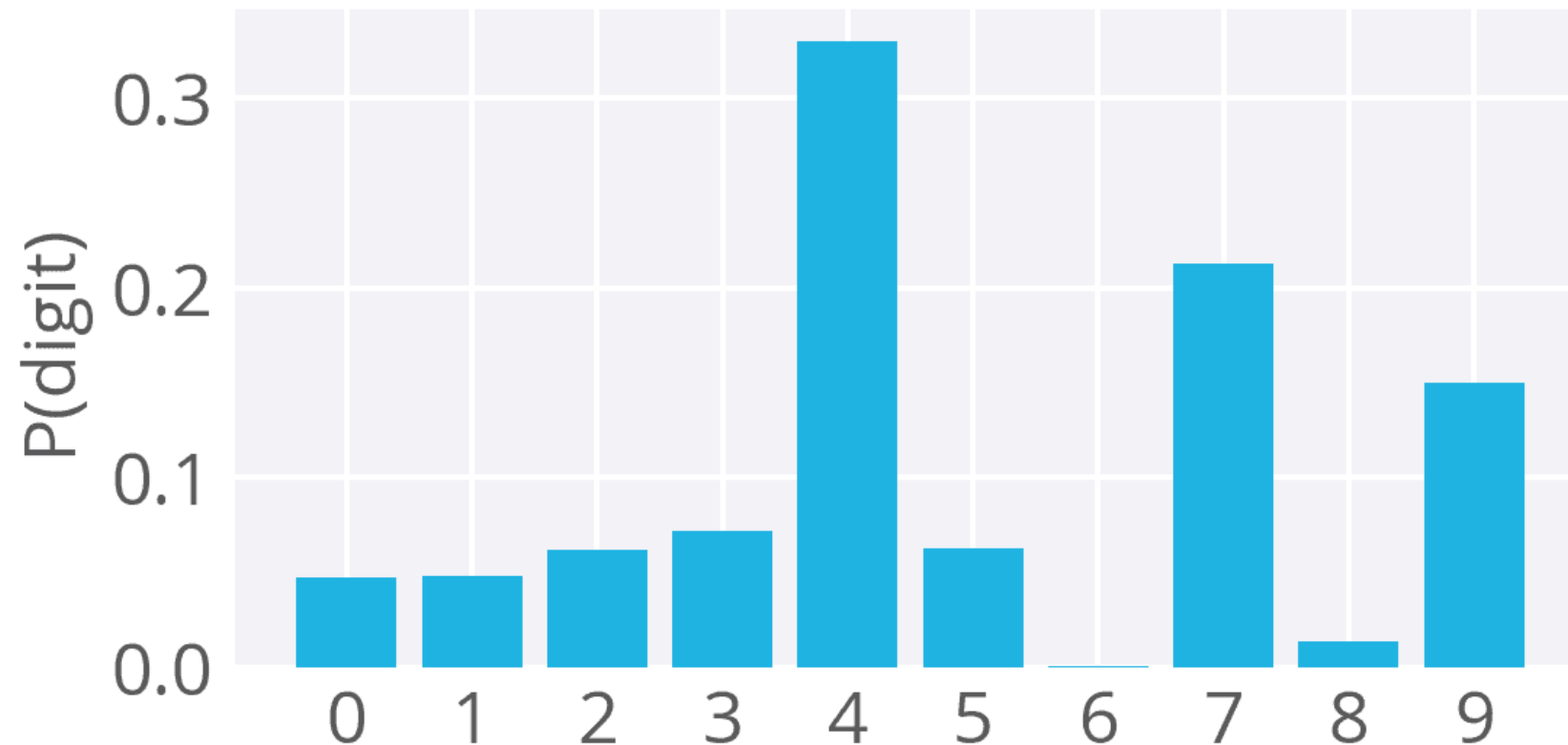
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

$j = 1, 2, \dots, K.$

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$



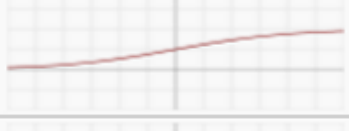
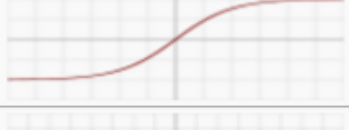

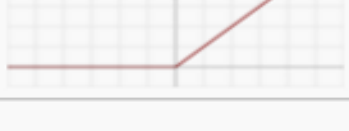



SOFTMAX – CLASSIFICATION OF DIGITS

- Transforms a real valued vector to $[0, 1]$
- Output of softmax is the list of probabilities of 10 digits



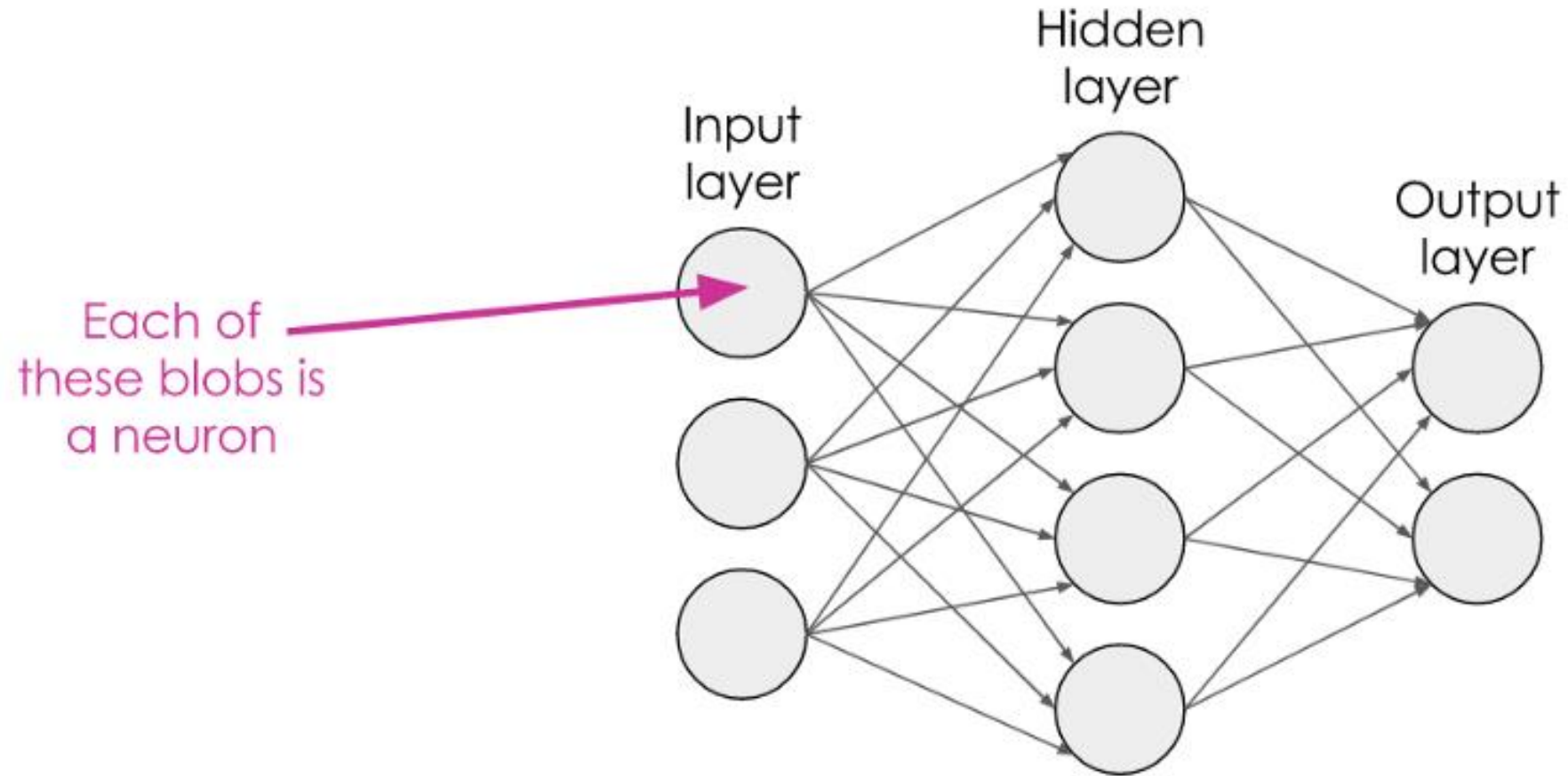
WHICH ACTIVATION FUNCTION TO USE?

- use **ReLU** which should only be applied to the hidden layers.
- And if our Model suffers from dead neurons during training we should use **leaky ReLU**.
- It's just that *Sigmoid and Tanh* should not be used nowadays due to the **vanishing Gradient Problem** which causes a lot of problems to train a Neural Network Model.
- **Softmax** is used in the last layer (Because it gives output probabilities).

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

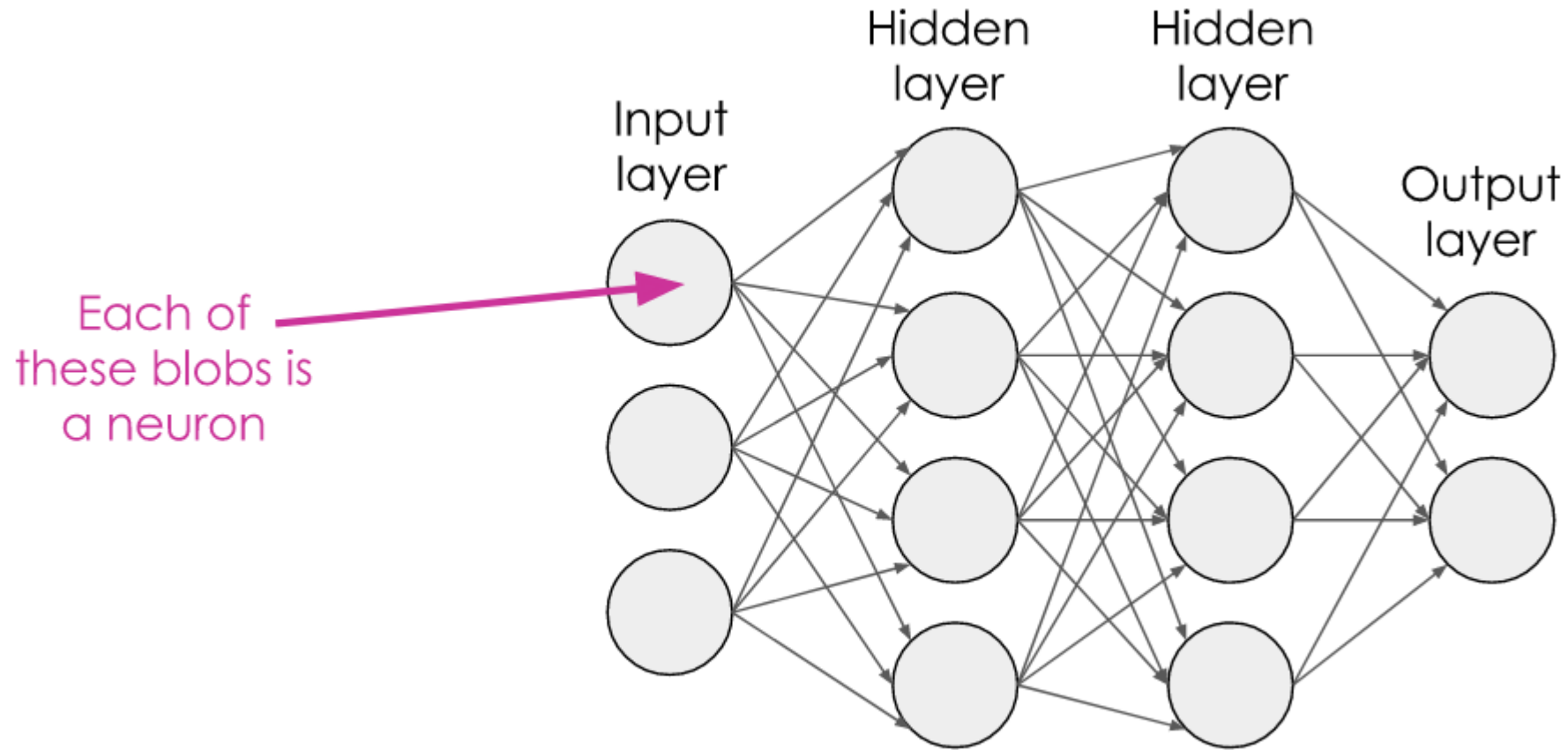
SINGLE NEURON TO A NETWORK - NEURAL NETWORK

SINGLE NEURON TO A NETWORK



- The outputs of the neurons in one layer flow through to become the inputs of the next layer, and so on.

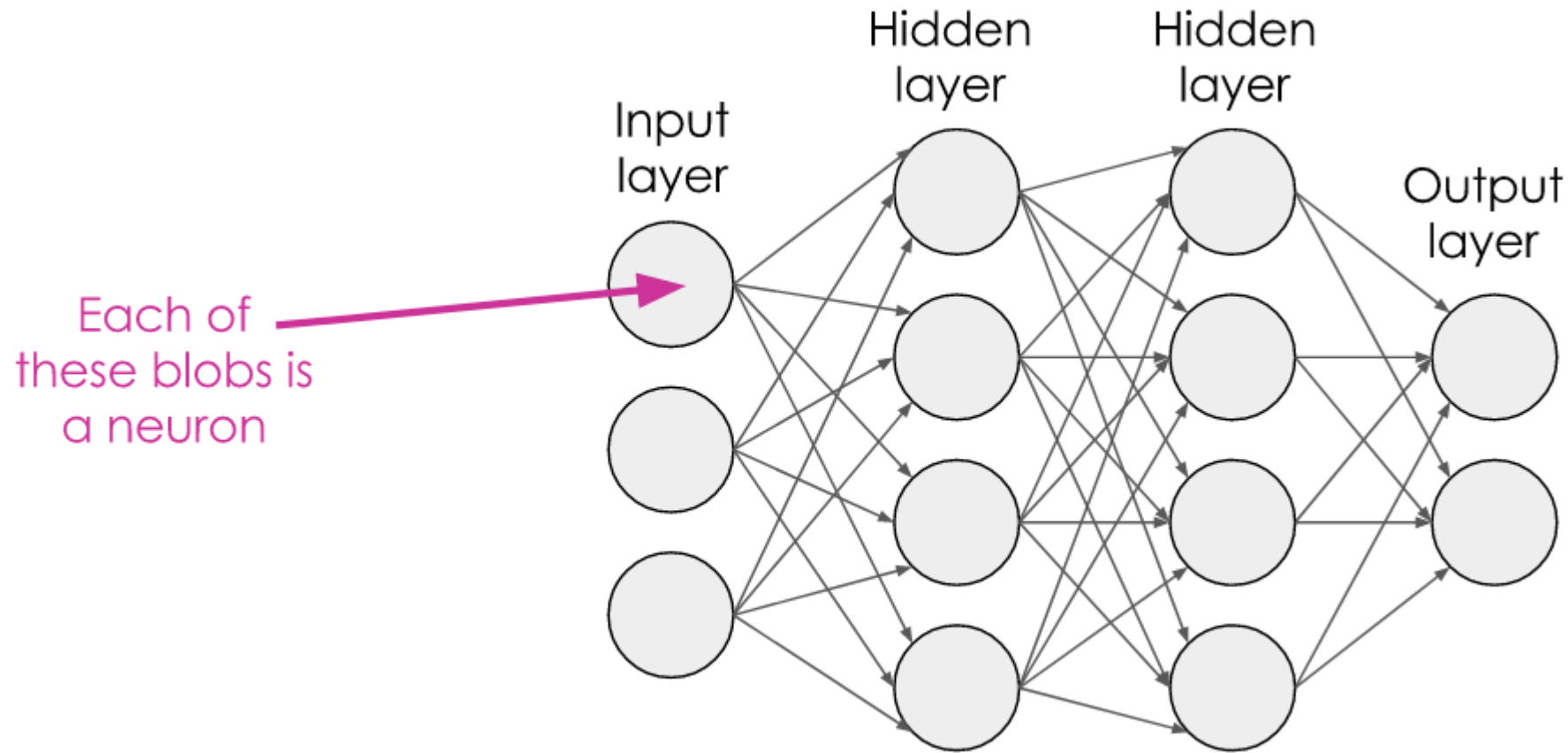
A NETWORK TO A DEEP NETWORK



Add another hidden layer....

a neural network can be considered 'deep' if it contains more hidden layers...

FEEDFORWARD NEURAL NETWORK



Add another hidden layer....

a neural network can be considered 'deep' if it contains more hidden layers...

TRAINING THE NETWORK

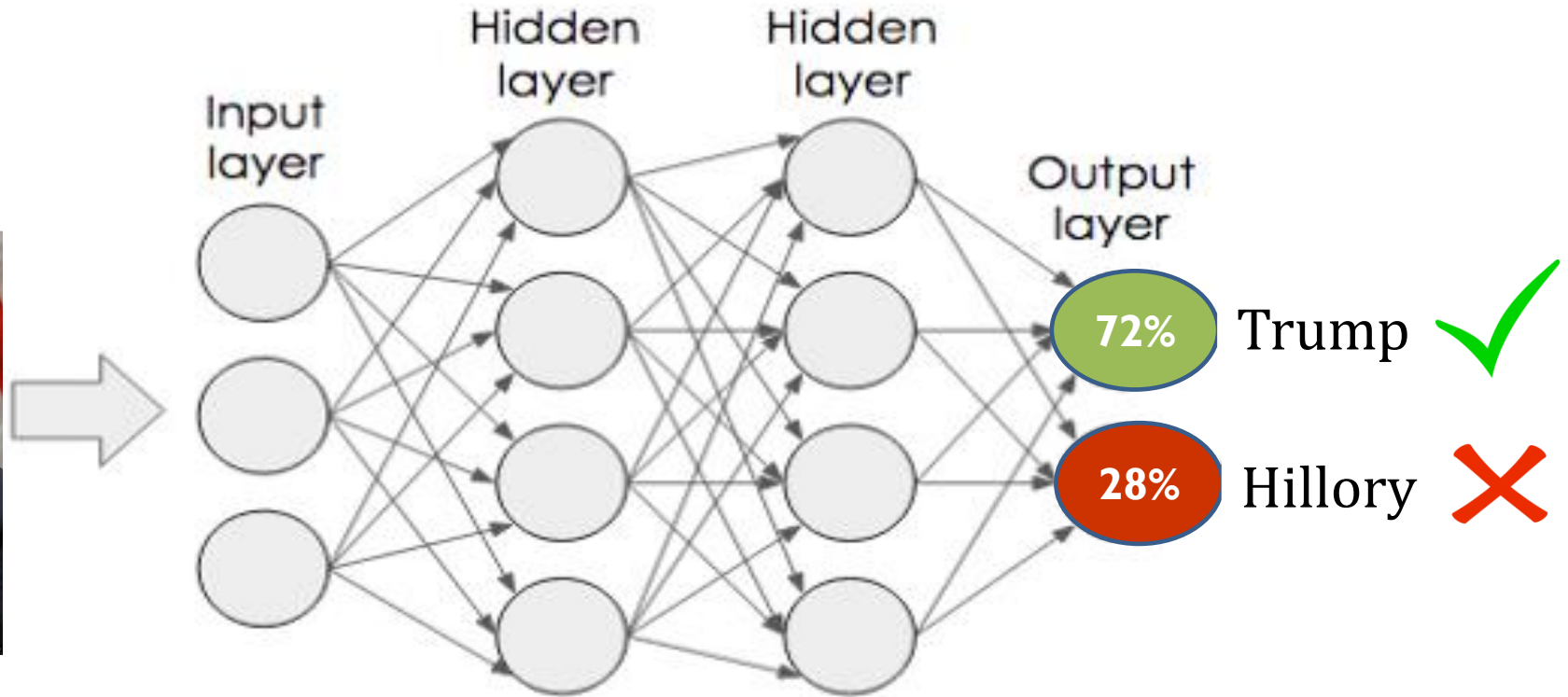
1. Randomly initialise the network weights and biases
2. **Training:**
 - Get a ton of labelled training data (e.g. pictures of cats labelled 'cats' and pictures of other stuff also correctly labelled)
 - For every piece of training data, feed it into the network
3. **Testing:**
 - Check whether the network gets it right (given a picture labelled 'cat', is the result of the network also 'cat' or is it 'dog', or something else?)
 - If not, how wrong was it? Or, how right was it? (What probability did it assign to its guess?)
4. **Tuning /Improving:** Nudge the weights a little to increase the probability of the network more probability getting the answer right.

EXAMPLE – CLASSIFY THE IMAGES – TRUMP & HILLORY



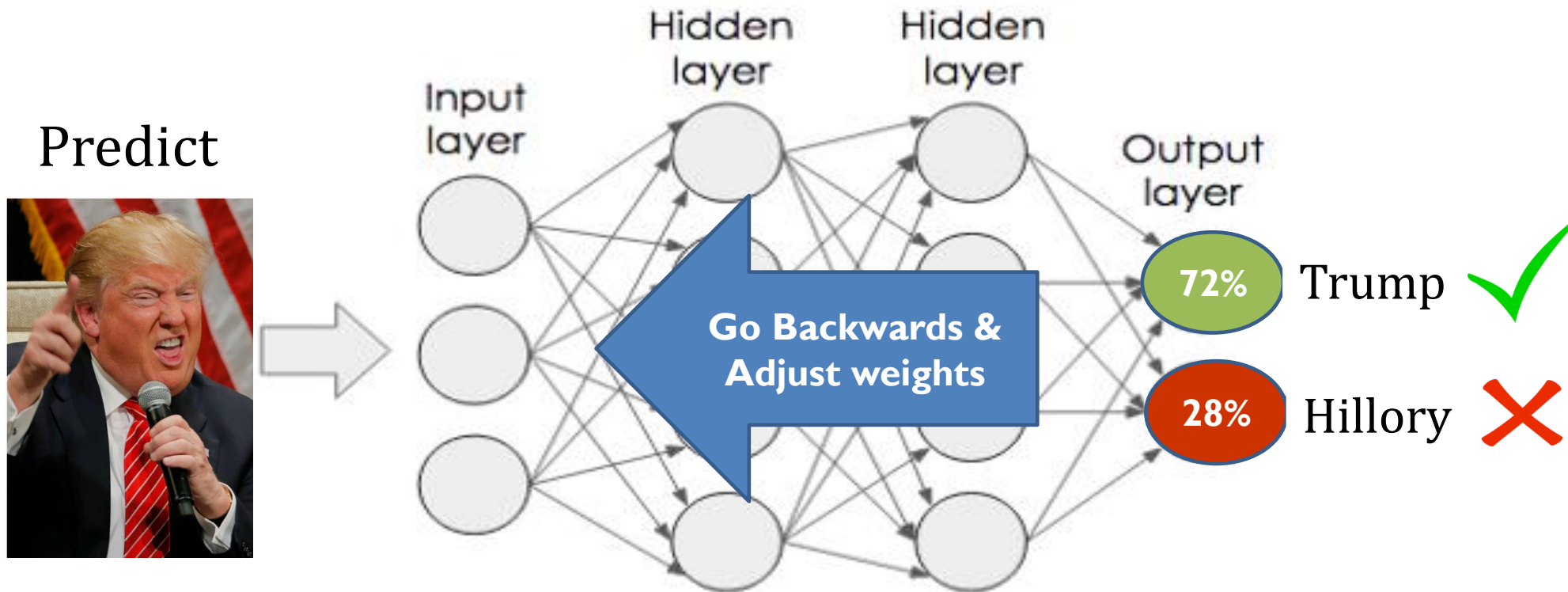
TRAINING THE NETWORK

Predict



1. Two output neurons (for each class one neuron)
2. O/P: A probability of **72%** that the image is "Trump" → 72 % is **Not sufficient**

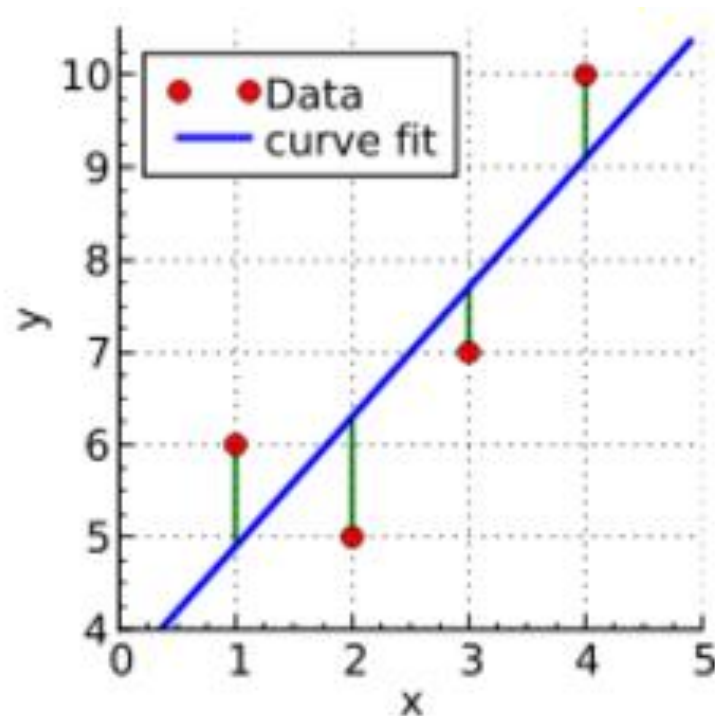
HOW TO INCREASE THE ACCURACY?



1. Two output neurons for each class
2. O/P: A probability of **72%** that the image is “Trump” → 72 % is **Not sufficient**

HOW TO INCREASE THE ACCURACY?

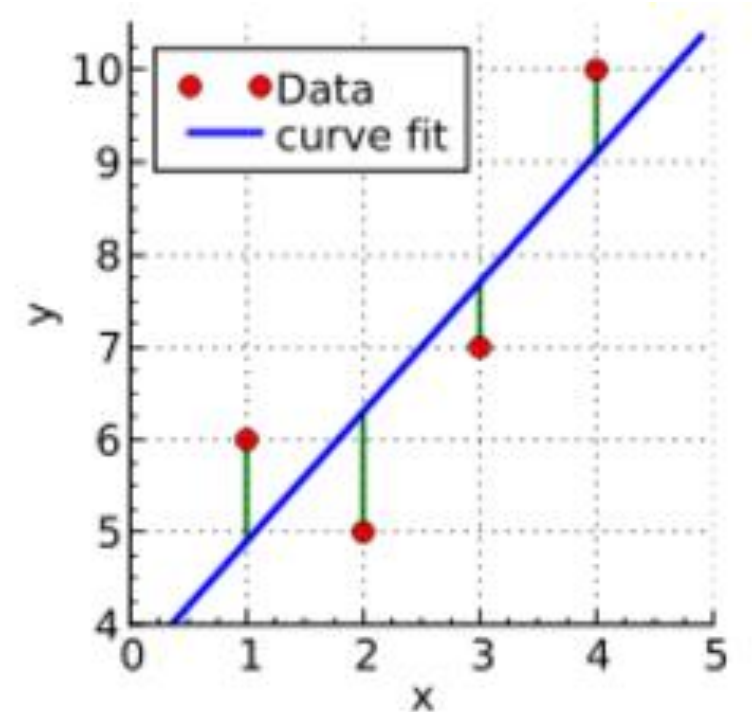
- Measure the network's output and the correct output using the '**loss function**'
- Which Loss Function??
 - Depends on the data
 - **Example:** Mean Square Error (MSE)



HOW TO INCREASE THE ACCURACY?

- Measure the network's output and the correct output using the '**loss function**'
- Which Loss Function??
 - Depends on the data
 - **Example:** Mean Square Error

The **goal of training** is to find the weights and biases that **minimizes the loss function**.

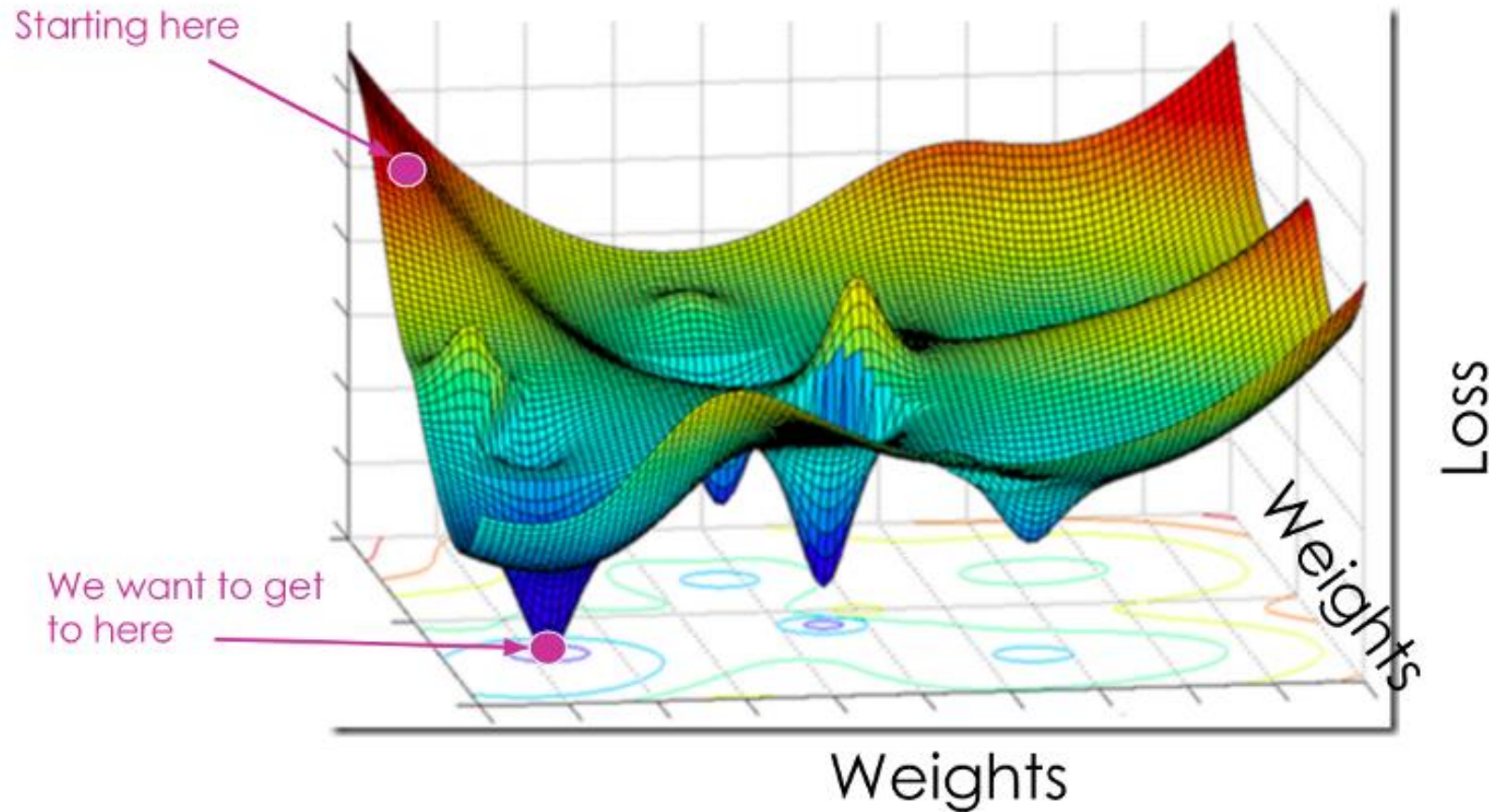


HOW TO INCREASE THE ACCURACY?

- We can plot the loss against the weights.
- To do this accurately, we would need to be able to visualise tons of dimensions, to account for the many weights and biases in the network.
- Because I find it difficult to visualise more than three dimensions,
- Let's pretend we only need to find two weight values. We can then use the third dimension for the loss.
- Before training the network, the weights and biases are randomly initialised so the loss function is likely to be high as the network will get a lot of things wrong.
- Our aim is to find the lowest point of the loss function, and then see what weight values that corresponds to. (See Next Slide).

HOW TO INCREASE THE ACCURACY?

- Here, we can easily see where the lowest point is and could happily read off the corresponding weights values.
- Unfortunately, it's not that easy in reality. The network doesn't have a nice overview of the loss function, it can only know what its current loss is and its current weights and biases are.



HOW TO INCREASE THE ACCURACY – GRADIENT DESCENT

Starting here



Network predictions



■ Cat
■ Dog

BACK PROPAGATION

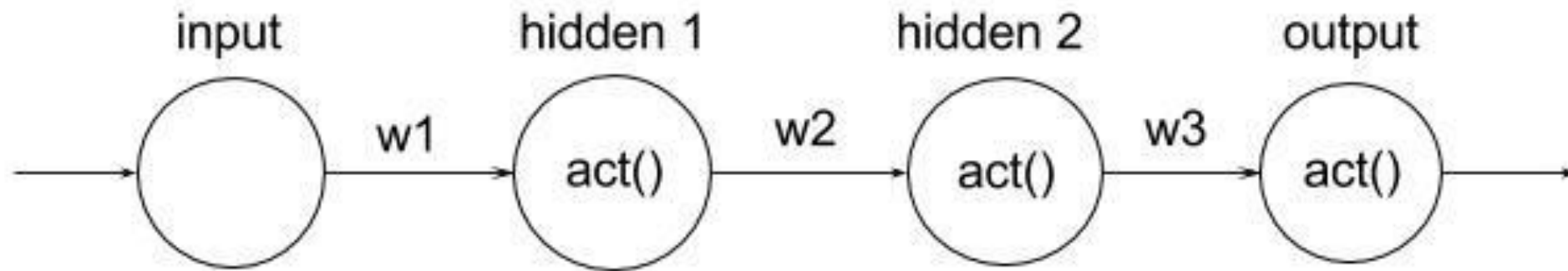
HOW A NEURON WORKS?

$$a_n = \text{act}(W_1x_1 + \dots + W_nx_n)$$

1. Each neuron **computes a weighted sum** of the outputs of the previous layer (which, for the inputs, we'll call \mathbf{x}),
2. **applies an activation function**, before forwarding it on to the next layer.
3. Each neuron in a layer has its own set of weights—so while each neuron in a layer is looking at the same inputs, their outputs will all be different.
4. The **final layer's activations are the predictions** that the network actually makes.

HOW A NETWORK WORKS?

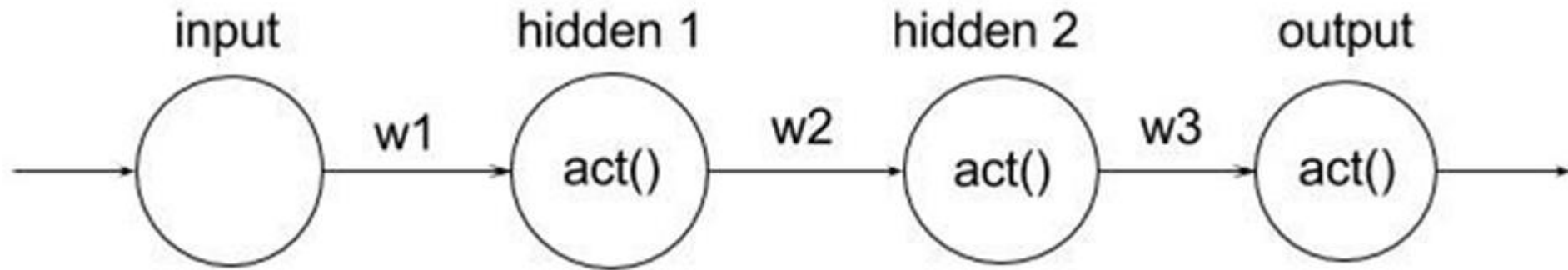
1. we want to find a set of weights \mathbf{W} that minimizes on the output of $J(\mathbf{W})$.
2. Consider a simple double layer Network.



3. each neuron is a function of the previous one connected to it. In other words, if one were to change the value of w_1 , both “hidden 1” *and* “hidden 2” (and ultimately the output) neurons would change.

HOW A NETWORK WORKS?

we can mathematically formulate the output as an extensive composite function:



$$output = act(w3 * hidden2)$$

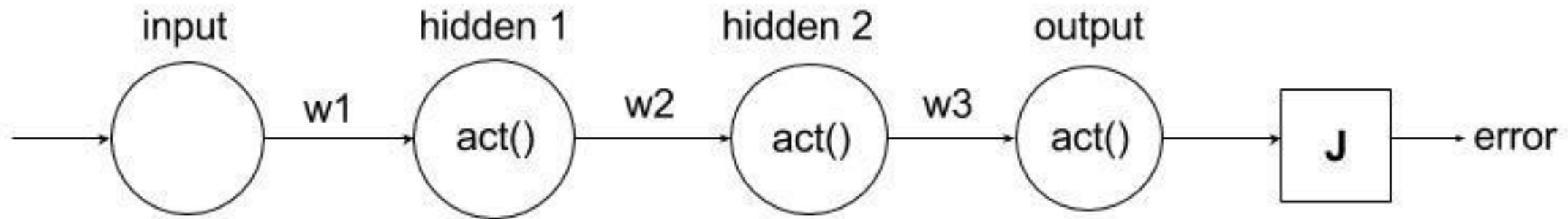
$$hidden2 = act(w2 * hidden1)$$

$$hidden1 = act(w1 * input)$$

$$output = act(w3 * act(w2 * act(w1 * input)))$$

INTRODUCE ERROR

we can mathematically formulate the **output as an extensive composite function**:



J is a function of \rightarrow input, weights, activation function and output

HOW TO MINIMIZE THE ERROR – BACK PROPAGATION

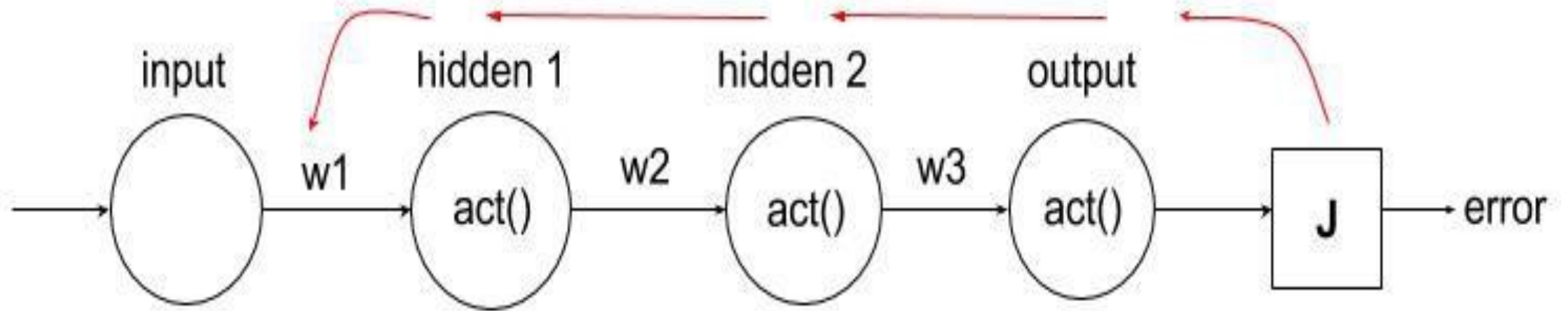
The error derivative can be used in gradient descent (as discussed earlier) to iteratively improve upon the weight.

We compute the error derivatives w.r.t. every other weight in the network and apply gradient descent in the same way.

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

BACK PROPAGATION

Find out the best “ w_1 ” by reducing the error from the last layer to first layer....



How Backpropagation works?

Refer the document – [backprop.docx](#)

TOOLS

MACHINE LEARNING TOOLS

Python



Java



.net



We need scalable Machine Learning/Data Mining Algorithms



Java, Scala



Spark

MLlib

Python, Scala



Statistical Analysis

C, FORTRAN Languages

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. Caffe is developed by the Berkeley Vision and Learning Center (BVLC), as well as community contributors and is popular for computer vision. Caffe supports [cuDNN](#) v5 for GPU acceleration.

Supported interfaces: C, C++, Python,.



Caffe2 is a deep learning framework enabling simple and flexible deep learning. Built on the original Caffe, Caffe2 is designed with expression, speed, and modularity in mind, allowing for a more flexible way to organize computation. Caffe2 supports [cuDNN](#) v5.1 for GPU acceleration.

Supported interfaces: C++, Python

MACHINE LEARNING TOOLS/Frameworks



The Microsoft Toolkit — previously known as CNTK— is a unified deep-learning toolkit from Microsoft Research that makes it easy to train and combine popular model types across multiple GPUs and servers. Microsoft Cognitive Toolkit implements highly efficient CNN and RNN training for speech, image and text data. Microsoft Cognitive Toolkit supports [cuDNN](#) v5.1 for GPU acceleration.

Supported interfaces: Python, C++, C# and Command line interface



TensorFlow is a software library for numerical computation using data flow graphs, developed by Google's Machine Intelligence research organization.

TensorFlow supports [cuDNN](#) v5.1 for GPU acceleration.

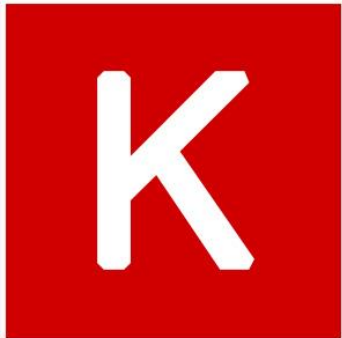
Supported interfaces: C++, Python

theano

Theano is a math expression compiler that efficiently defines, optimizes, and evaluates mathematical expressions involving multi-dimensional arrays.

Theano supports [cuDNN](#) v5 for GPU acceleration.

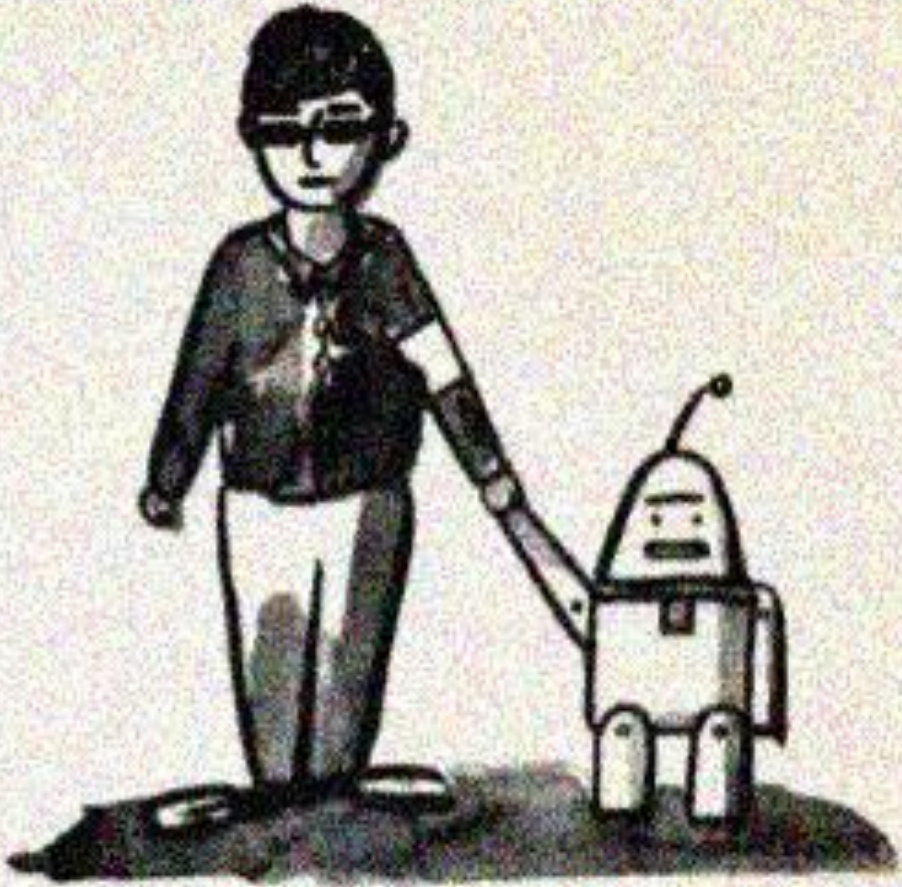
Supported interfaces: Python



Keras is a minimalist, highly modular neural networks library, written in Python, and capable of running on top of either TensorFlow or Theano. Keras was developed with a focus on enabling fast experimentation.

cuDNN version depends on the version of TensorFlow and Theano installed with Keras.

Supported Interfaces: Python



THANK YOU!

QUESTIONS?