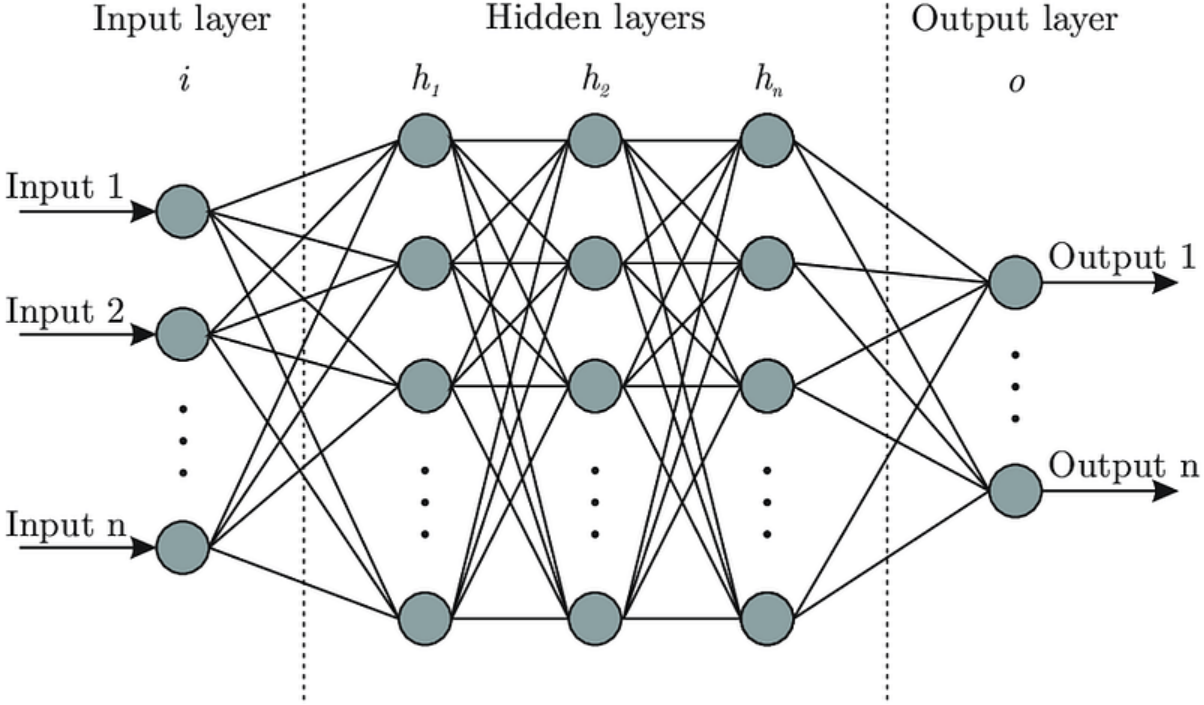
Theory behind Perceptrons: From Biological to Artificial Neurons

This notebook is predestined to Neural Networks and Deep Learning enthusiasts. We will explore the history and evolution of Artificial Neural Networks, from their creation in the 1940s by McCulloch and Pitts to the Perceptron learning algorithm proposed by pioneer Frank Rosenblatt.

As part of my exploration of this field, I will regularly share my discoveries and learnings in the form of articles and notebooks.
Stay tuned🔔 and follow me on this exciting adventure.

- **Check the Full Article:** Theory behind Perceptrons: From Biological to Artificial Neurons.
- **Link to Kaggle Notebook** Perceptrons: From Biological to Artificial Neurons

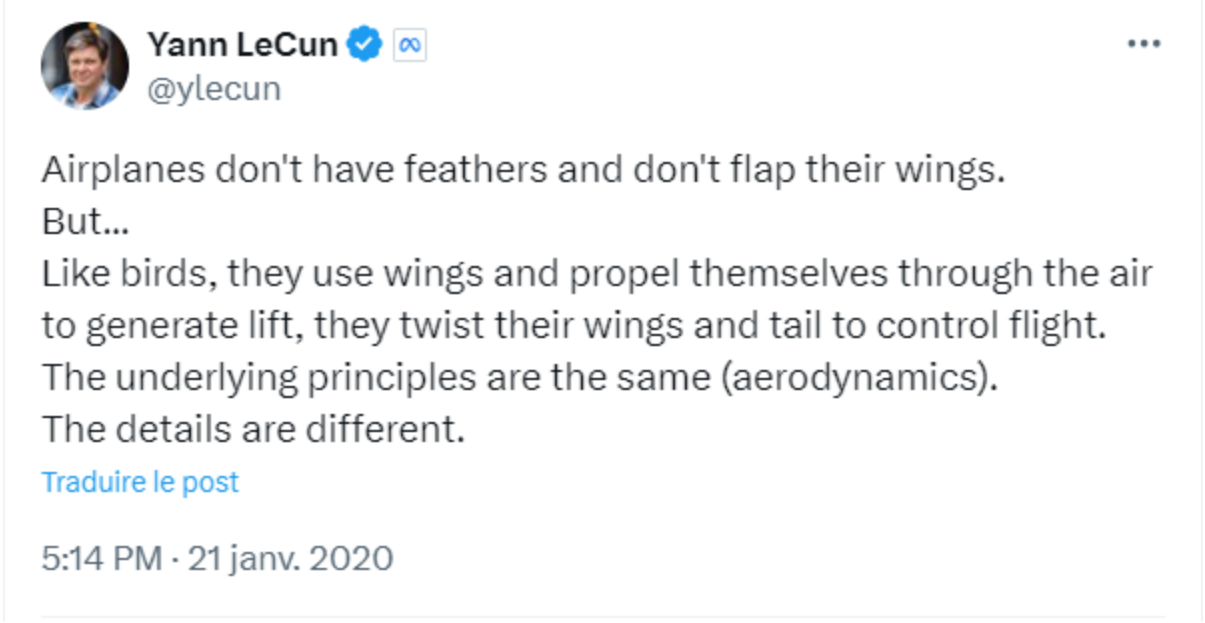
Artificial Neural Networks, or ANNs for short, are at the very core of the Deep Learning field. ANNs are computational models consisting of interconnected nodes, organized in layers and inspired by the structure and function of the human brain.



They are versatile, powerful and extremely scalable structures that can **perform tasks that only living beings are initially capable of**, such as pattern recognition, text comprehension and generation, decision-making, etc... Their recent evolution may even lead us to claim that they have already surpassed man in certain tasks but that's a moot point we won't go into here.

> "Neural networks function in the same way as the human brain." -- A popular thought.

This is a popular preconceived but completely wrong idea that I often hear from people. **The human brain is far more complex and sophisticated than any existing ANN architecture.** And as the famous VP and Chief AI Scientist at Meta Yann LeCun once said, **although planes were inspired by birds, they don't have to flap their wings.** But of course that doesn't take away the fact that ANNs are heavily inspired by the way the human brain works.



Let's discover all about Perceptrons and, of course, gain an in-depth understanding of their inner workings.

Tables of contents

1. Brief history of ANNs

Check the full article for more details about this part.

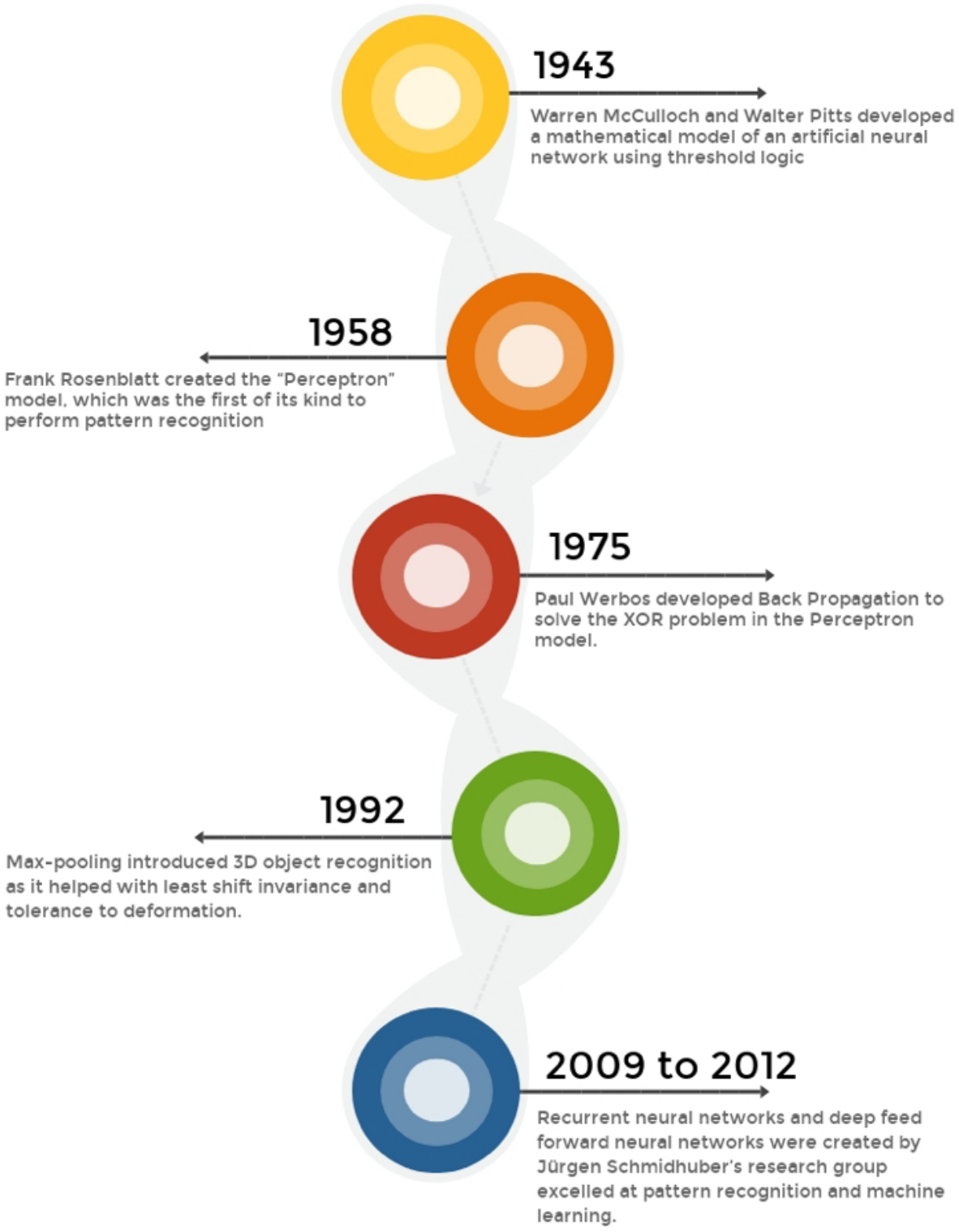


Image Credits: Allerin

Back to top

2. Functionning of Biological Neurons

It's important to understand how biologicals neurons works before diving deep into ANNs functionning and architectures. A little refresher course in Life Sciences that most people learned in high school wouldn't hurt :)

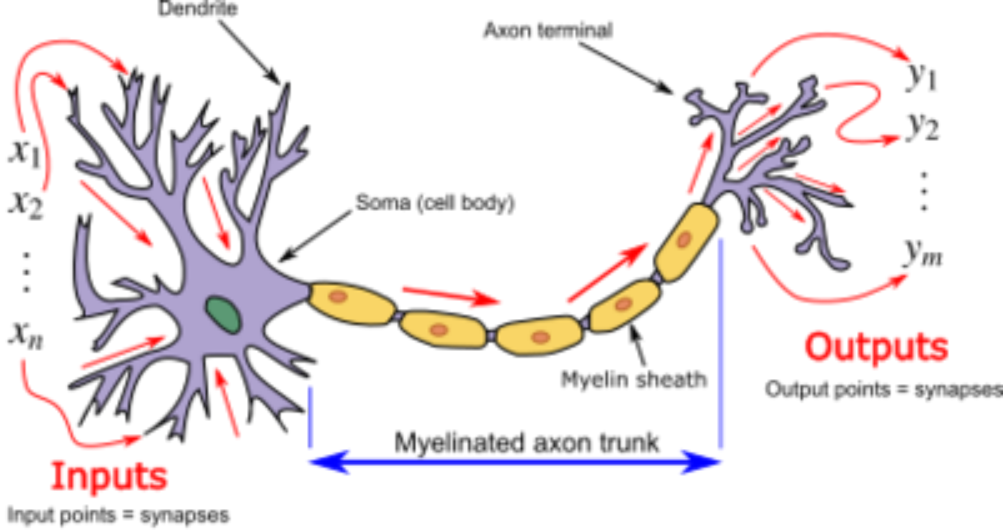
Definition of a neuron

According the biology dictionary, **a neuron is a single nervous system cell that receives, processes, and transmits electrochemical messages from and to other cells.**
In other words, **it's the basic building block of the nervous system in living organisms, including humans that are responsible for transmitting electrical and chemical signals within the brain, spinal cord, and throughout the body.**

> "A cell that carries information between the brain and other parts of the body."--Cambridge dictionary

Anatomy of a neuron:

A neuron has three basic parts: a cell body, an axon and a dendrite.



- **The cell body (also known as the soma)** contains a nucleus that controls the cell's activities as well as other genetic materials.
- **The dendrites** are branch-like extensions that protrude from the cell body. They receive signals from other neurons or sensory receptors and transmit these signals toward the cell body.
- **The axon** looks like a long tail that extends from the cell body. It conducts electrical impulses, known as action potentials, away from the cell body and toward other neurons. At the end of the axon, there are small branches called axon terminals or synaptic terminals. These terminals form connections, called synapses, with dendrites or cell bodies of other neurons.

Inner working of a neuron

All neurons have three basic functions:
- **Receive signals** (or information),
- **Integrate incoming signals** (to determine whether or not the information should be passed along) and, - **Communicate signals to target cells.**

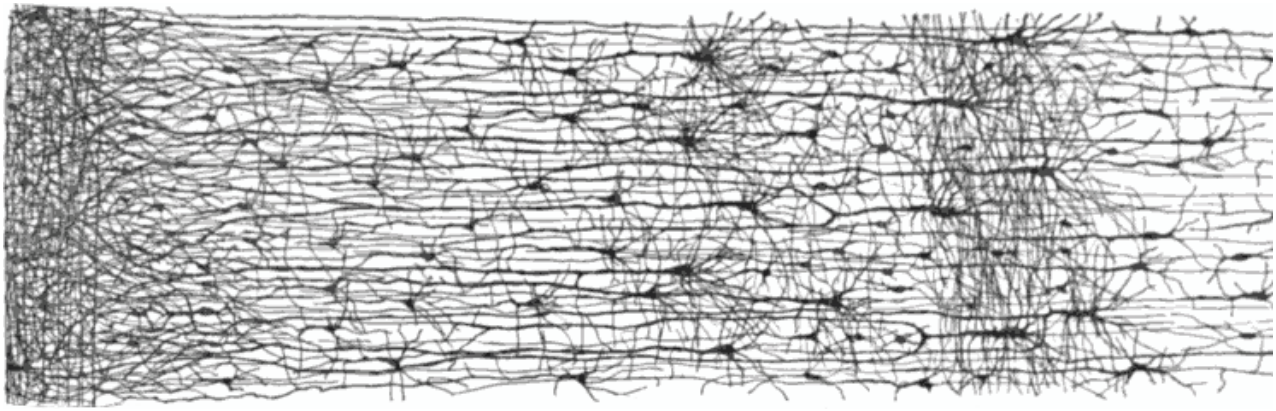
> "When a neuron receives a sufficient number of signals from other neurons within a few milliseconds, it fires its own signals."

Biological Neural Networks

Individual biological neurons may seem simple on their own, but when you consider that there are millions of millions of these neurons, things get really interesting. **Each neuron is like a tiny piece of a big puzzle, and they communicate with thousands of other neurons.** Even though each neuron operates in a straightforward manner, when you link them up in a massive network, they can perform incredibly complex tasks.

Scientists are still actively studying the structure of Biological Neural Networks (BNN). While there's much we're still figuring out, some parts of the brain have been mapped, giving us a glimpse into how neurons are arranged.

It appears that these neurons often form layers, like stacking building blocks. Imagine it as if these layers are the different floors in a skyscraper where each floor contributes to the overall functioning of the brain similar to the layers illustrated in the following image:



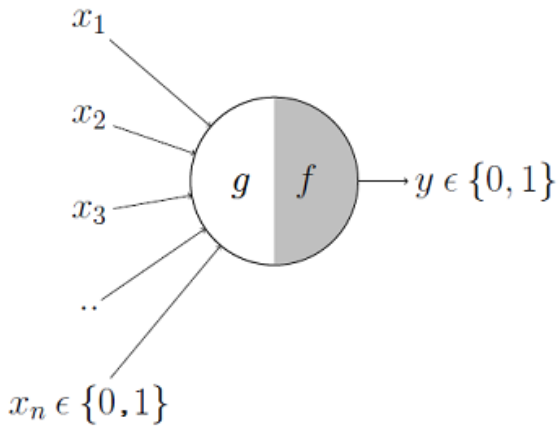
Now that we have a little bit of knowledge about the architecture and functioning of neurons and the human brain, let's discover how these are "implemented" artificially.

Back to top

3. Architecture of a Single-Layer Perceptron

Simplified McCulloch-Pitts Model

In the McCulloch-Pitts model, **artificial neurons are represented as binary threshold units.** Each neuron takes binary inputs (0 or 1 also on/off) from other neurons or external sources and produces a binary output (0 or 1) based on a predetermined threshold.
Simply put, the neuron's output is activated (set to 1) if more than a certain number of its inputs are active; otherwise, the output remains deactivated (set to 0).



$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

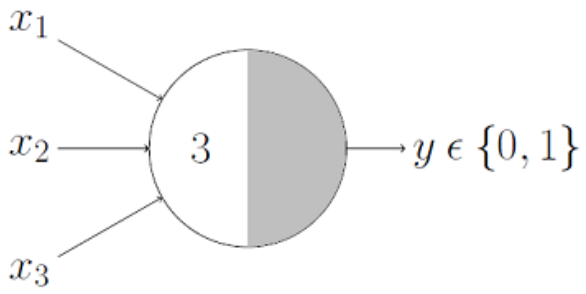
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

The process is splitted into two steps. In the first part, g takes an input (kind of like a message receiver - dendrite), and it adds everything up. Then, based on this added-up value, the second part, f makes a decision using a special number called theta. **This is**

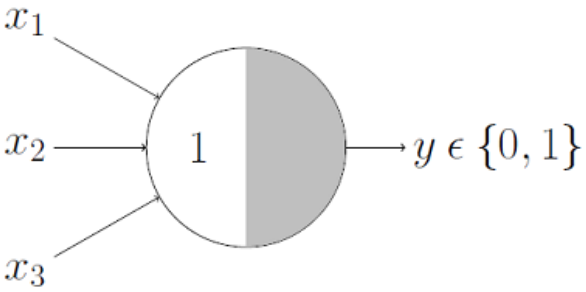
called the Thresholding Logic.

Note here the similarity of this model to the neuron diagram described above. Such simple model was primarily used to demonstrate the computational capabilities of simple network of artificial neurons since it's possible to compute almost any logical proposition you want.

Examples : ANNs performing simple logical computations. - **AND function:** neuron would only fire when all the inputs are activated(ON or 1) i.e., g(x) ≥ 3 here.



- **OR function:** neuron would fire if ANY of the inputs is activated (1) i.e., g(x) ≥ 1 here.

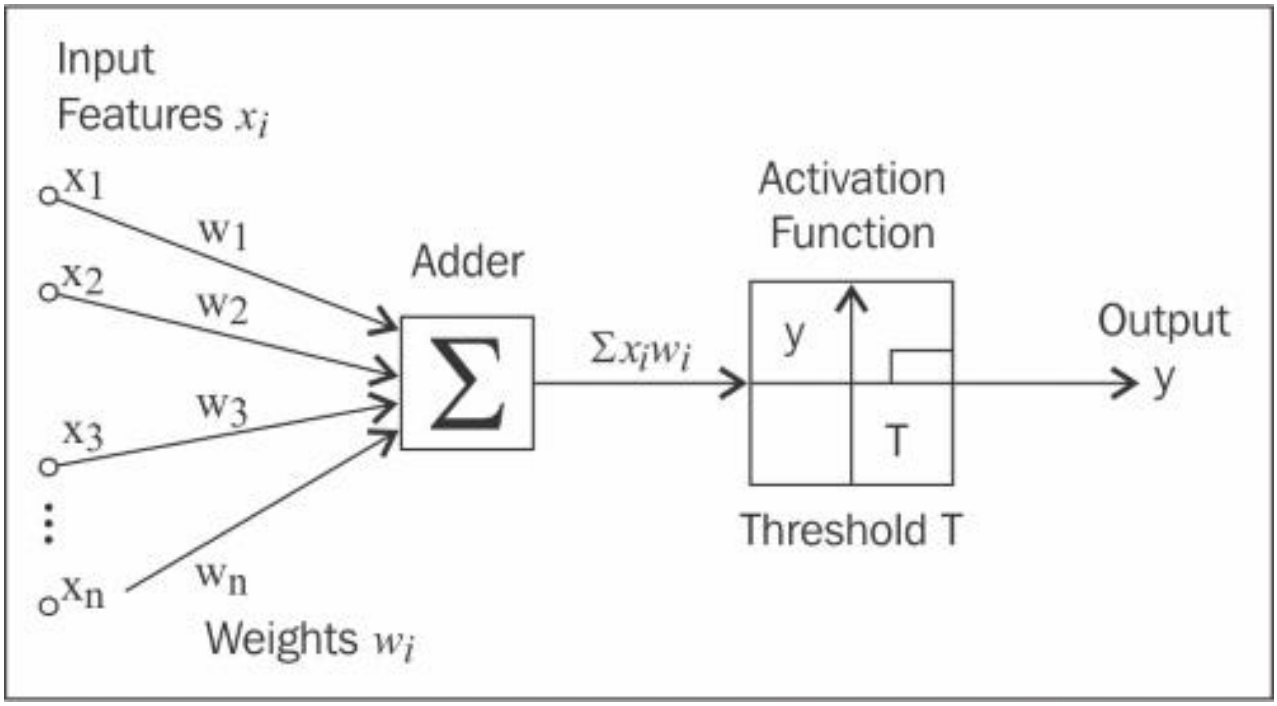


The Threshold Logic Units (TLUs)

Threshold Logic Units (TLUs) also known as Linear Threshold Units (LTU) were initially proposed by *Frank Rosenblatt in the late 1950s* as **the basic units of Perceptrons.**

These computational units are also based on a threshold activation function i.e., they apply a step function to the weighted sum of their

inputs and then produce outputs that are now numbers (instead of binary on/off values used in McCulloch-Pitts model).



There are two common step functions used in TLUs: **Heaviside and Sign.**

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

TLUs are capable of defining linear decision boundaries in input space. In two-dimensional space, for example, a single TLU can separate points into two classes (binary classification) using a straight line.

It simply computes a linear combination of the inputs: if the result exceeds a threshold, it outputs the positive class or else outputs the negative class (just like a Logistic Regression classifier). Training a TLU in this case means finding the right values for weights.

The Perceptron

Perceptrons are the simplest architecture of ANNs.

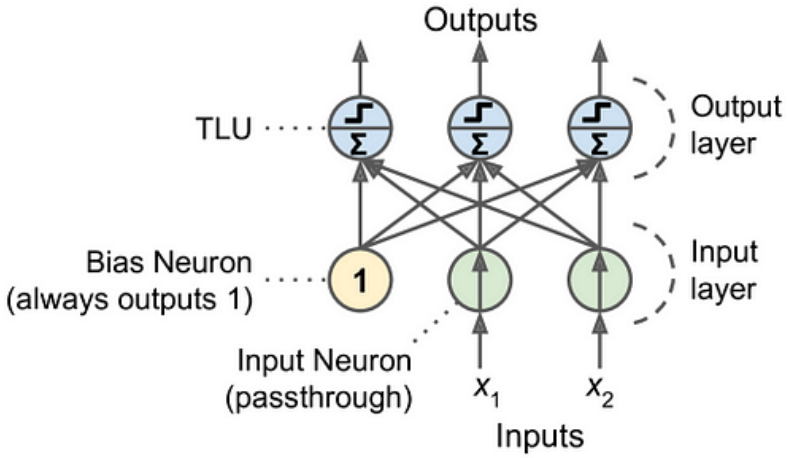
A Perceptron is simply composed of a single layer of TLUs, with each TLU connected to all the inputs forming a dense layer.

> "When all the neurons in a layer are connected to every neuron in the previous layer, it is called a fully connected layer or a dense layer."

To show that every input goes to each TLU, we often draw special **"input neurons."** These neurons just pass on whatever they receive as input. All these input neurons together form what we call the **input layer.**

Additionally, we usually include an extra feature called a bias. It's like a constant factor that's always there. We represent this bias using a special kind of neuron called a **"bias neuron,"** which always outputs 1.

Now, let's imagine a Perceptron with two inputs and three outputs.



With this setup, the Perceptron can classify things into three different categories all at once, which means it's a multi-output classifier.

Equation 10-2. Computing the outputs of a fully connected layer

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{XW} + \mathbf{b})$$

- **X represents the matrix of input features.** Each row represents one instance, and each column represents one feature.

- **W contains all the connection weights** between input neurons and artificial neurons in the layer. Each row corresponds to one input neuron, and each column corresponds to one artificial neuron.
- **The bias vector b contains the connection weights** between the bias neuron and the artificial neurons. Each element of this vector represents the bias term for one artificial neuron.
- **The function "phi" is the activation function.**

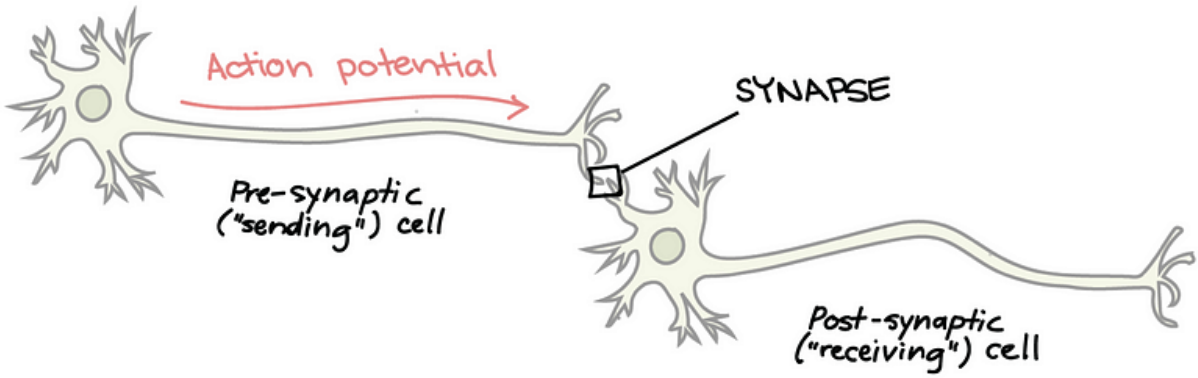
Back to top

4. The Perceptron training algorithm

Hebb's rule

Hebb's rule or Hebbian learning, proposed by Canadian psychologist Donald Hebb in his book "*The Organization of Behavior*" *published in 1949*, is a foundational principle in the field of neuroscience and Neural Network theory.
Hebb's postulate states:

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."



Simply put, Hebb's rule can be summarized as:

> "Neurons that fire together, wire together."--Hebb's Law.

This rule suggests that the strength of the connection between two neurons increases if they are activated simultaneously. If neuron A repeatedly triggers neuron B, the connection between them strengthens over time. This mechanism is fundamental to the process of learning and memory formation in the brain.

Perceptron learning algorithm

The Perceptron training algorithm proposed by Frank Rosenblatt was largely inspired by Hebb's rule. Perceptrons are trained using a variant of this rule that considers the network's errors and strengthens connections that help minimize these errors. Here's how it works:

- **We train the Perceptron one instance at a time.**
- **For each instance, the Perceptron makes predictions.**
- **If any of the predictions are wrong, the connections from the inputs that could have led to the correct prediction are reinforced.** This helps the Perceptron learn from its mistakes

and improve its accuracy over time.

Equation 10-3. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

> "If the training instances are linearly separable, Rosenblatt demonstrated that this algorithm would converge and find a solution."--The Perceptron convergence theorem.

5. Implementing Perceptron from scratch:

Implementing Perceptron from scratch helps us truly to understand how it works under the hood and we'll be able to customize and optimize it later for specific tasks depending on our needs.

The Dataset

The dataset we're going to use here is the **Breast Cancer Wisconsin (Diagnostic) dataset that is a collection of information gathered from digitized images of breast cancer cells.** It contains features like the size, shape, and texture of cell

nuclei, along with other characteristics computed from the images. The dataset is used to predict whether a detected breast tumor is malignant (cancerous) or benign (non-cancerous).

In [1]:
```
# useful librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score

# load dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

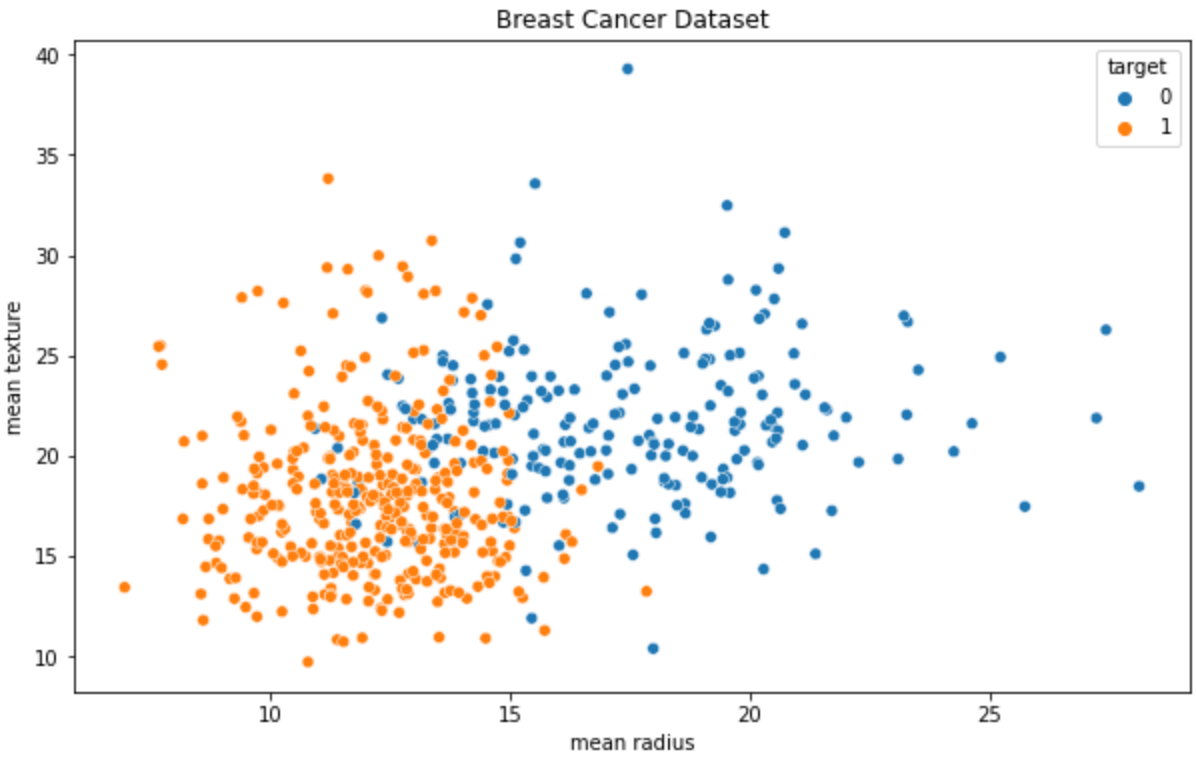
In [2]:
```
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

5 rows × 31 columns

In [3]:
```
# we'll only use the mean radius and mean texture features to plot the scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='mean radius', y='mean texture', hue='target')
plt.title('Breast Cancer Dataset')
plt.xlabel('mean radius')
plt.ylabel('mean texture')
plt.savefig("breast_cancer.png")
plt.show()
```



Perceptron Class Implementation

From the plot above we can see the two different classes but they are not totally linearly separable. Let's see how a Perceptron can classify them.

In [4]:
```
# From scratch Perceptron class
class Perceptron:
    def __init__(self, numb_feats, learning_rate=0.1, numb_epochs=150):
        self.numb_feats = numb_feats
        self.learning_rate = learning_rate
        self.numb_epochs = numb_epochs
        self.weights = np.zeros(numb_feats)
        self.bias = 0
        self.predictions = None

    def activation(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x):
        result = np.dot(self.weights, x) + self.bias
        return self.activation(result)

    def train(self, X, y):
        for _ in range(self.numb_epochs):
            for i in range(len(X)):
                pred = self.predict(X[i])
                self.weights += self.learning_rate * (y[i] - pred) * X[i]
                self.bias += self.learning_rate * (y[i] - pred)

    def make_predictions(self, X):
        if self.predictions is None:  # Predictions haven't been made yet
            self.predictions = [self.predict(x) for x in X]
        return self.predictions
```

In [5]:
```
X = df.iloc[:, [0, 1]].values
y = df.iloc[:, -1].values

# train the perceptron
perceptron = Perceptron(numb_feats=2)
perceptron.train(X, y)
```

In [6]:
```
# make prediction on the same set and see the accuracy
preds = perceptron.make_predictions(X)

print("Our Perceptron's accuracy:", accuracy_score(y, preds))
```

```
Our Perceptron's accuracy: 0.8558875219683656
```

6. Implementing Perceptron with Scikit-Learn

Scikit-Learn provides a Perceptron class that implements a single TLU network defining as shown in the official documentation:

sklearn.linear_model.Perceptron

class **sklearn.linear_model.Perceptron**(*, *penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

The below code is just an example of how it can be implemented:

In [7]:
```
from sklearn.linear_model import Perceptron

# perceptron class
sklearn_percept = Perceptron(max_iter=150,tol=0.1)
sklearn_percept.fit(X, y)
```

Out[7]:
```
Perceptron(max_iter=150, tol=0.1)
```

In [8]:
```
# make prediction on the same set and see the accuracy
sklearn_preds = sklearn_percept.predict(X)

print("Sklearn Perceptron's accuracy:", accuracy_score(y, sklearn_preds))
```

```
Sklearn Perceptron's accuracy: 0.6625659050966608
```

The accuracy is about 0.6625. Still figuring out why :) **But what is certain is that for this binary classification problem, Perceptron is not well suited and we're going to see why!**

7. Limitations of Single-Layer Perceptrons

Single-layer Perceptrons can be used in many scenarios particularly where the data is linearly separable and the problem is relatively simple including binary classification, some pattern recognition, etc...

However, they are unable to model complex decision boundaries for non-linearly separable data. The inability to learn XOR (exclusive OR) is a classic example.

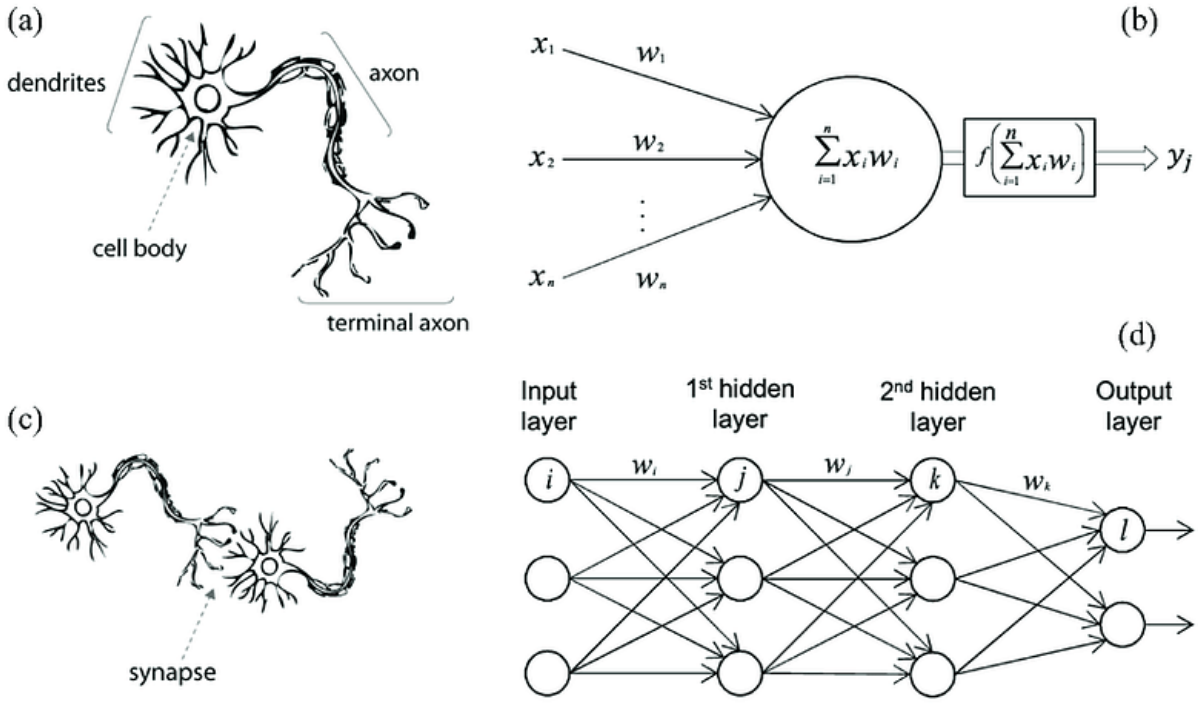
Also, they produce binary outputs (0 or 1) based on a threshold activation function. This binary output may not be suitable for tasks requiring continuous or probabilistic predictions.

Conclusion

The theory behind Perceptrons is fascinating, as they draw inspiration from the structure and function of the human brain, akin to interconnected puzzle pieces capable of learning and decision-making. Although Artificial Neural Networks (ANNs) may not reach the complexity of the human brain, they remain potent tools for various tasks. From their inception in the 1940s with McCulloch and Pitts to Rosenblatt's pioneering Perceptron learning algorithm, Perceptrons have profoundly impacted the field of Machine Learning.

However, for tackling more intricate problems requiring the comprehension of non-linear decision boundaries and complex data patterns, advanced neural network architectures like Multi-

Layer Perceptrons (MLPs) and Deep Neural Networks (DNNs) are preferred. In essence, while Perceptrons represent a significant milestone in the history of ANNs, they merely scratch the surface of a vast and evolving field.



References and Ressources :

- **Full article on medium:** Theory behind Perceptrons: From Biological to Artificial Neurons.
- **GitHub repo:** Hands on Deep Learning - From Scratch
- **Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow - Aurélien Géron:** This work is inspired from this excellent book that I'm still exploring and worth reading. I share this for learning purposes and to help others discover this fascinating field of Deep Learning.
- The Simplified McCulloch-Pitts Model part and The two examples are inspired from this article McCulloch-Pitts Neuron - Mankind's First Mathematical Model Of A Biological Neuron.
- Some images are from ResearchGate website. Hope I'll not be

blamed for copyrights.

- **Notebook Design:** PS4E2 : Visual EDA📈 | LGBM | Obesity Risk

Back to top

In [9]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download "perceptrons-from-biologica
```

```
[NbConvertApp] Converting notebook perceptrons-from-biological-to-artificial-neuron
s.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1439068 bytes to perceptrons-from-biological-to-artificial-ne
urons.pdf
```