Artificial Neural Networks (ANN) The basics

Michael Claudius, Associate Professor, Roskilde

22.04.2020; Revised 09.11.2020, 09.04.2022

What is Artificial Neural Networks(ANN) ?

- **A very *powerful* versatile algorithm for both classification and regression**
- **Predictions are based on a layered network with so called neurons**
  - **A neuron is cell computing a value to next layer neuron**
  - **Neurons are connected with weights**
  - **Training is based on iterations (epochs) over the dataset**

- **So its predicting something; but lets evaluate first!**

Evaluation of ANN?

- **Advantages**
  - **Very good for complex and huge/medium sized data sets**
  - **Scalable**
  - **Easy to use**
  - **Many API forms: Sequential, Functional, Subclassing**

- **Disadvantages**
  - **Slow**
  - **Black box; details unknown how it works**
  - **Complex, pipelining with scaling is needed**
  - **Greedy algorithm, (which must be stopped)**

History

- **Invented in 1943 based on propositional logic (binary on/off 1/0 values)**
- **1960’s development stopped couldn't solve some simple problems**
- **1980s new architectures**
- **1990’s development stopped, other ML (e.g. SVM) worked better**
- **2010’s ANN strikes back due**
    - **Computer speed and storage**
    - **Huge data**
    - **Image recognition**
    - **Speech recognition**
  - **Using different training algorithm**

Lets see how it looks!

Biological neurons

- **A neuron fires an electic signal producing chemical neurotransmitter to other neurons**
- **When neurotransmitters exceed a threshold then receive-neuron fire electric signals to other neurons**
- **Billions of neurons in network and each neuron is connected to thousands of neighbour neurons**

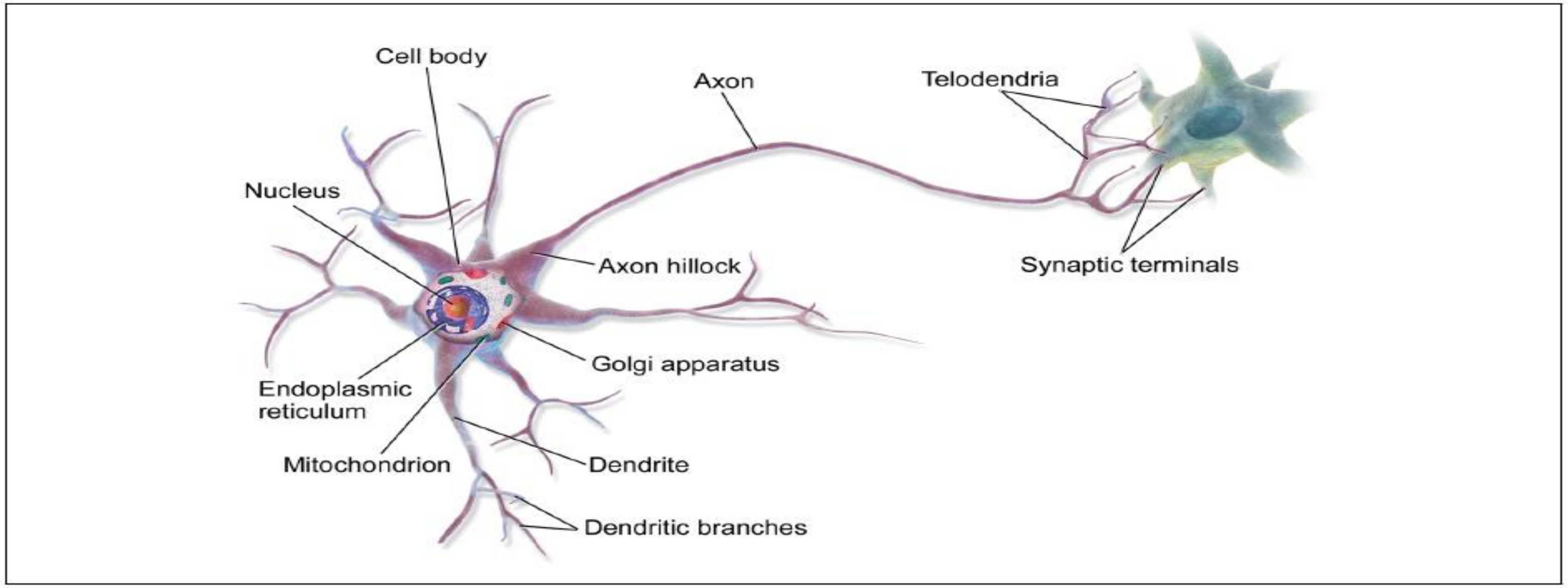

Figure 10-1. Biological neuron[4]

Artificial neuron

- **Invented in 1943 based on propositional logic (binary on/off 1/0 values, and, or, not)**
- **Simple: binary input and binary output**
- **Performs simple logical computations**

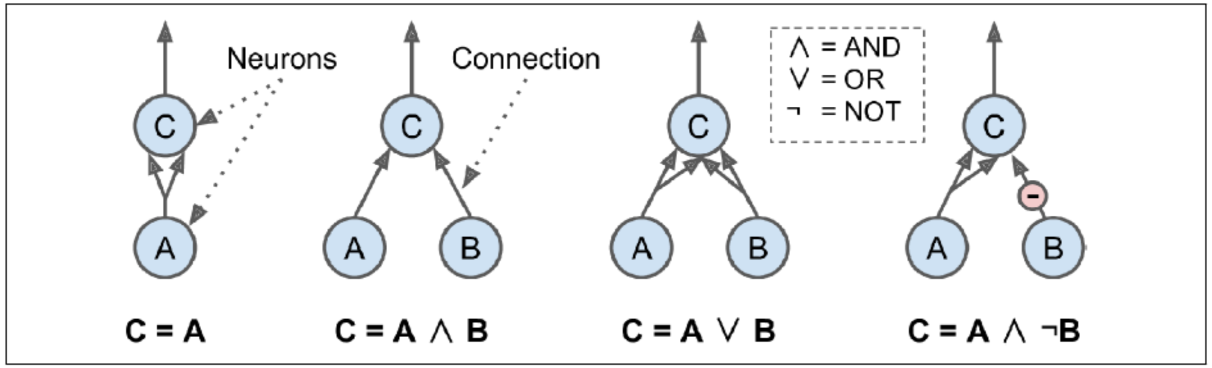

Figure 10-3. ANNs performing simple logical computations

Lets see how it looked in 1957, 14 years later!

Perceptron from 1957

- **Principle Threshold Logical Unit (TLU)**
- **Calculate $z = w_1 X_1 + w_2 X_2 + w_3 X_3 + \ldots w_n X_n$,**

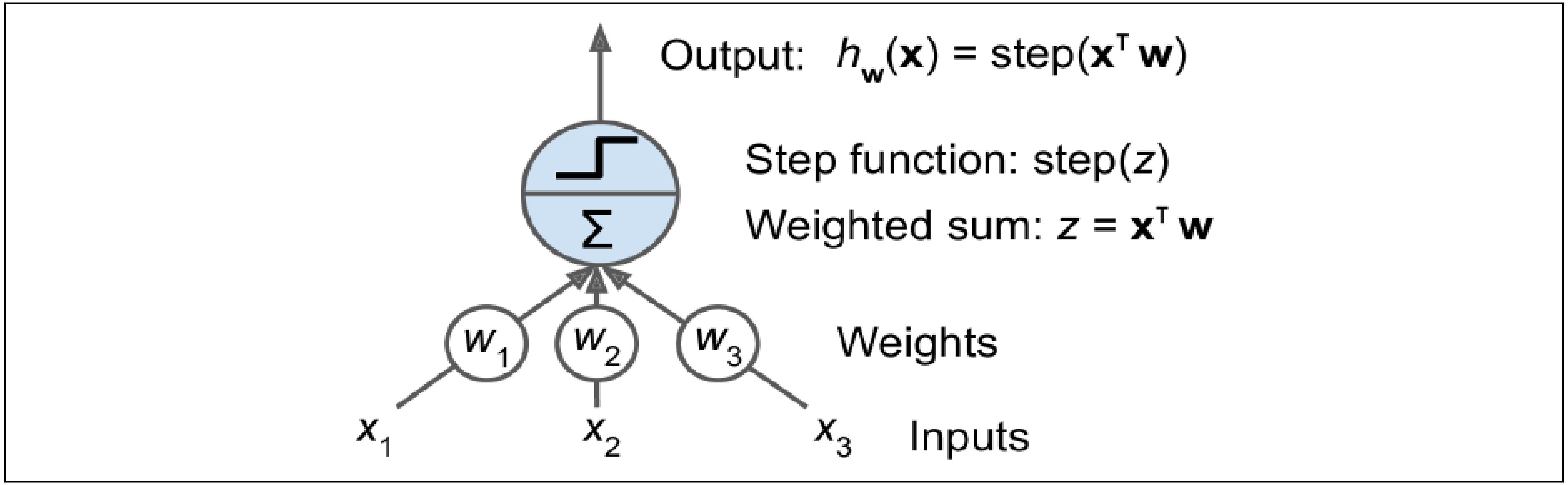

Figure 10-4. Threshold logic unit: an artificial neuron which computes a weighted sum of its inputs then applies a step function

Perceptron step activation function

- **Heaviside step function, h, works on the weighted sum z**
- **Output 0 or 1**

Equation 10-1. Common step functions used in Perceptrons (assuming threshold = 0)

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

Perceptron output function

- **Calculate output, h, based on activation function fi,**
- **b is the bias weight on the bias neuron**
- **Calculate $z = w_0 X_0 + w_1 X_1 + w_2 X_2 + w_3 X_3 + \ldots.. w_n X_n$, ;$w_0 X_0 = b$ is the bias connection weight as $X_0 = 1$**
- **φ is the activation function**

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

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

- **Heaviside function, h, works on the weighted sum z = φ(xw+b)**

Perceptron training

- **Fire and wire, using the learning rule. Start with random weights, then adjust.**
- **Take training instance one by one, lower weight for wrong prediction, increase weight for correct prediction**

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

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

In this equation:

- $w_{i,j}$ 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.

Perceptron example

- **One input layer, one TLU-layer, one output layer**
- **And some times impossible to use even on simple cases**

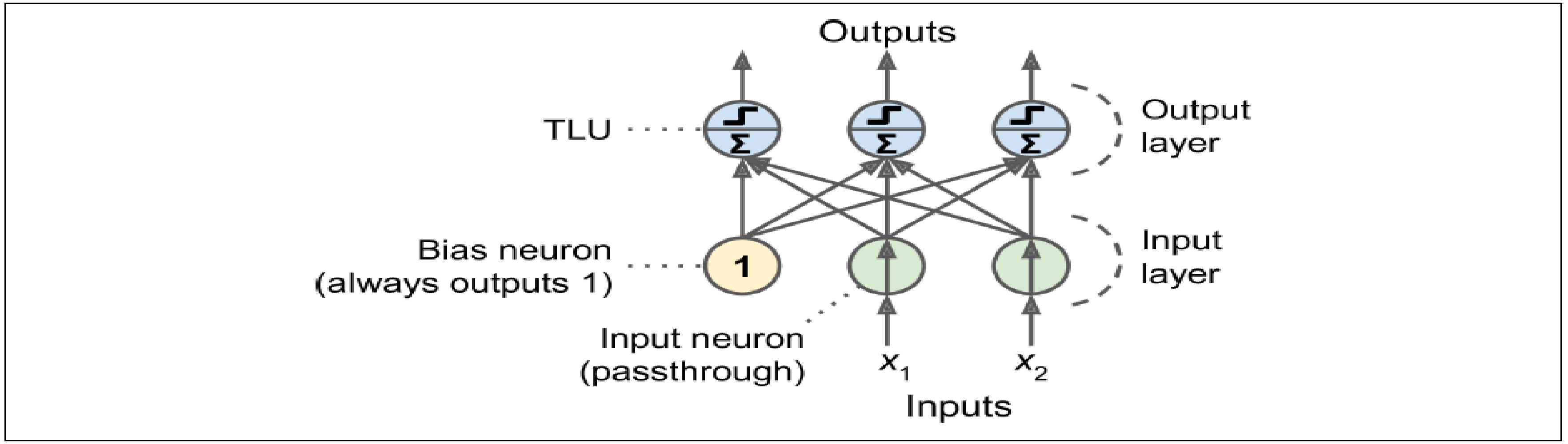

Figure 10-5. Architecture of a Perceptron with two input neurons, one bias neuron, and three output neurons

Perceptron code for Iris data set

- **Import libraries**
- **Make a Perceptron, train (fit-function)**

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)

per_clf = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

- **What about probability. NO! Cannot predict probability**

Assignments first round

- **It is time for discussion and solving a few assignments in groups**

- Tensorflow Installation
- Perceptron Iris Exercise