IT496: Introduction to Data Mining

Lecture 24

Introduction to Neural Networks

(Slides are created from the lecture notes of Dr. Derek Bridge, UCC, Ireland)

Arpit Rana

10^{th} October 2023

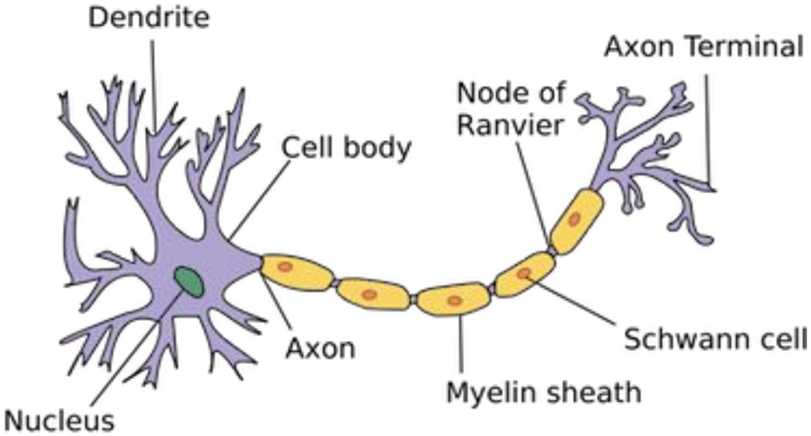
Introduction

Neural Networks are loosely inspired by what we know about our brains:

- Networks of neurons.
- However, they are not models of our brains.
  - E.g. there is no evidence that the brain uses the learning algorithm that is used by neural networks.

Biological Neurons

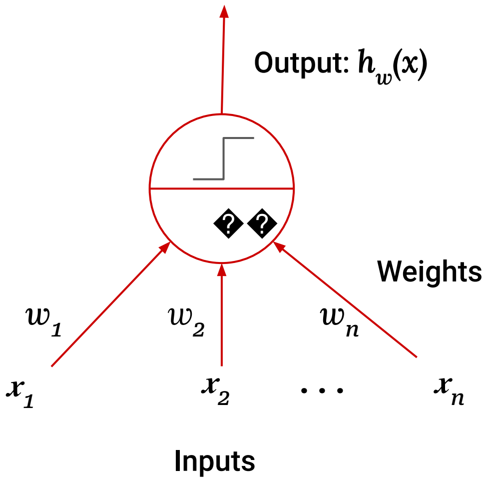
- Our brain is a network of about 10^{11} neurons, each connected to about 10^4 others
- Sufficient electrical activity on a neuron's dendrites causes an electrical pulse to be sent down the axon, where it may activate other neurons.



https://commons.wikimedia.org/wiki/File:Neuron.svg

Artificial Neuron

- A simple artificial neuron has n real-valued inputs, $x_1, \ldots, x_n$.
- The connections have real-valued weights, $w_1, \ldots, w_n$..
- The neuron also has a number b called the bias.



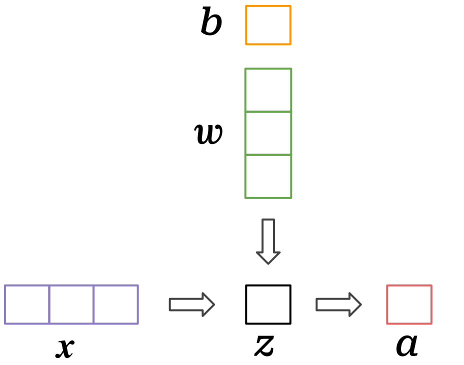
Artificial Neuron

- The neuron computes the weighted sum of its inputs and adds b:

$$z = b + w_1x_1 + w_2x_2 + \;.\;.\;.\; + w_nx_n$$

or if $\boldsymbol{x}$ is a row vector of the inputs and $\boldsymbol{w}$ is a (column) vector of the weights

$$z = b + xw$$



Although artificial neurons are inspired by real neurons, really all we're doing is the *dot product* of two vectors, followed by element-wise application of the *activation function*.

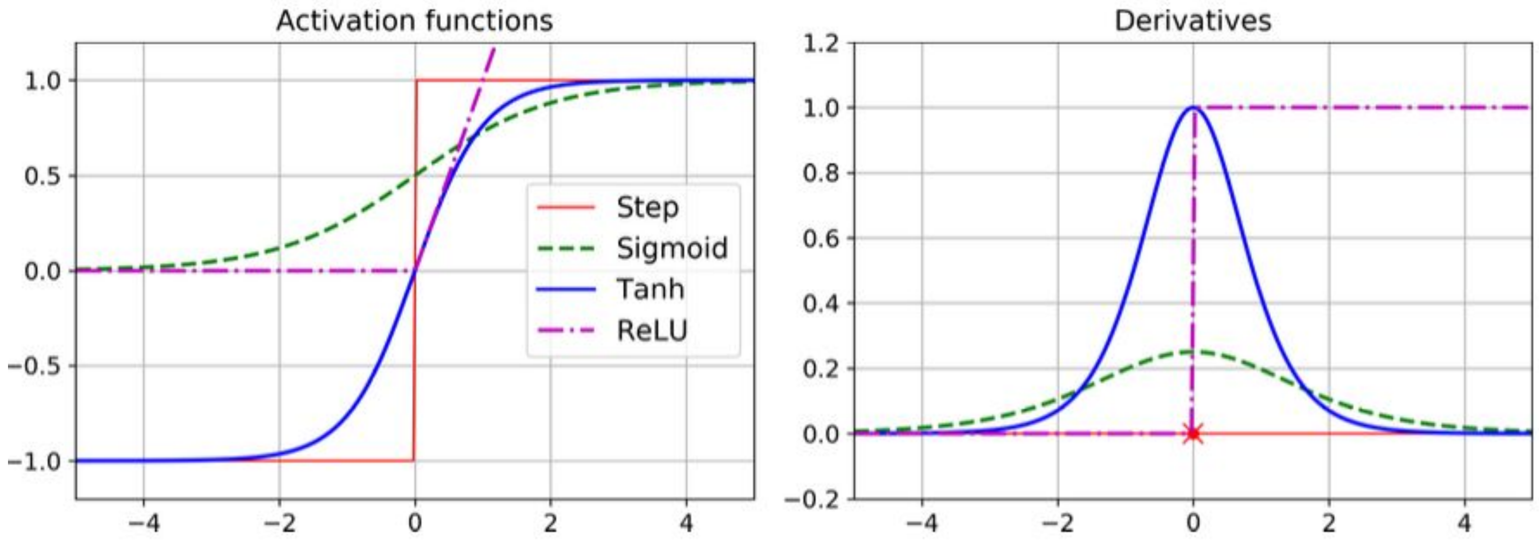
Artificial Neuron

Many activation functions have been proposed, including:

- ***linear*** activation function: $g(z) = z$
- ***step*** activation function: $g(z) = \begin{cases} 0 & z < 0, \\ 1 & z \geq 0 \end{cases}$
- ***sigmoid*** activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$
- ***tanh*** activation function (tanh is the hyperbolic tangent): $g(z) = \tanh z$
- ***ReLU*** activation function (ReLU stands for Rectified Linear Unit): $g(z) = \max(0, z)$

Apart from the *linear* activation function, these activation functions are *non-linear*, which is important to the power of neural networks.

Activation functions and their derivatives



Relationship with Linear Models

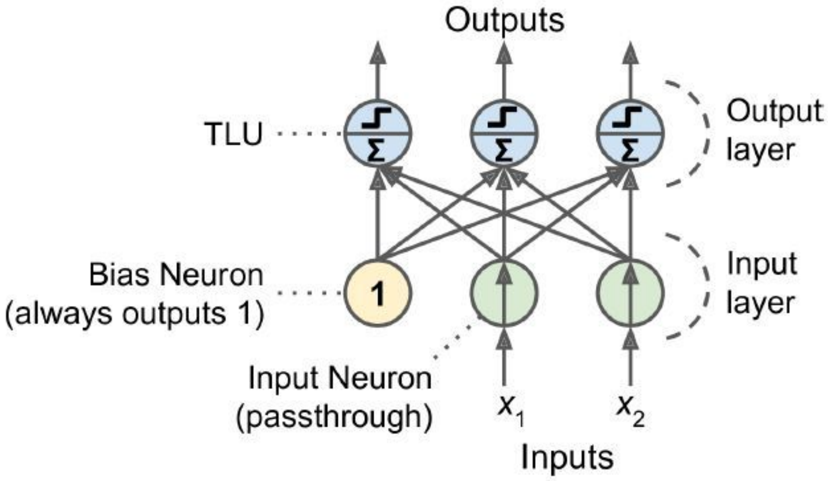
- A single artificial neuron that uses the *linear activation function* gives us the same linear models that we had in *Linear Regression*.
    - If we find the values of the weights and bias using MSE as our loss function, then we will be doing OLS regression.

- A single artificial neuron that uses the *sigmoid activation function* gives us the same models that we had when using *Logistic Regression* for binary classification.
    - We can set the weights using the binary cross-entropy function as our loss function.

Layers of Neurons

We don't usually have just one neuron. We have a layer, containing several neurons.

- For now let's consider what is called a *dense layer* (also a ***fully-connected layer***): *every input is connected to every neuron in the layer.*



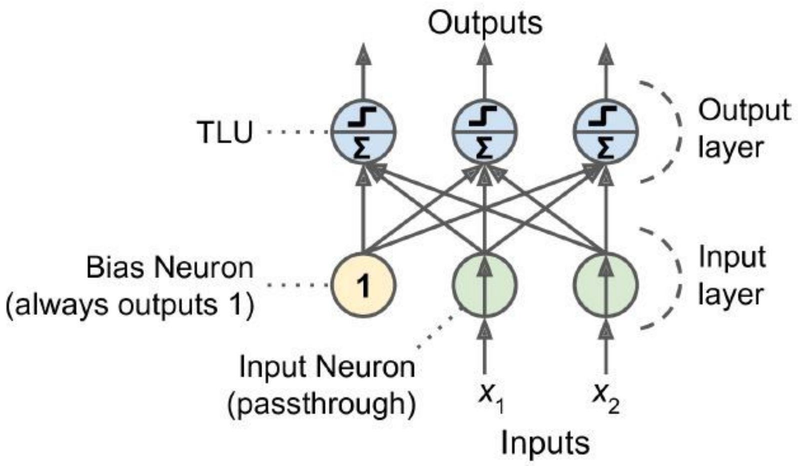
- So now we have more than one output, one per neuron, each calculated as before.

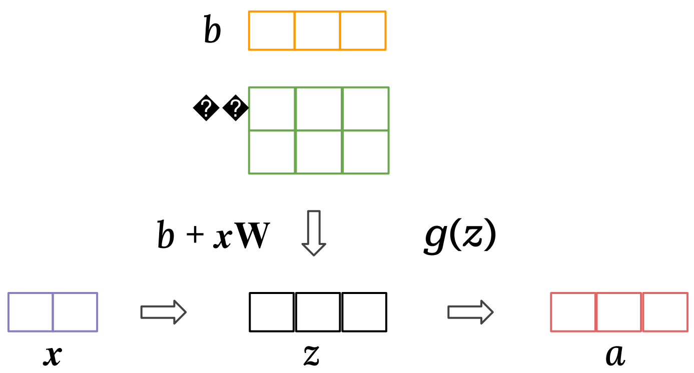
Image Source: Hands on Machine Learning by Aurelien Geron

Matrix Multiplication

Suppose there are m inputs and p neurons in a layer. We can put all the weights into a m x p matrix:

$m = 2, p = 3$



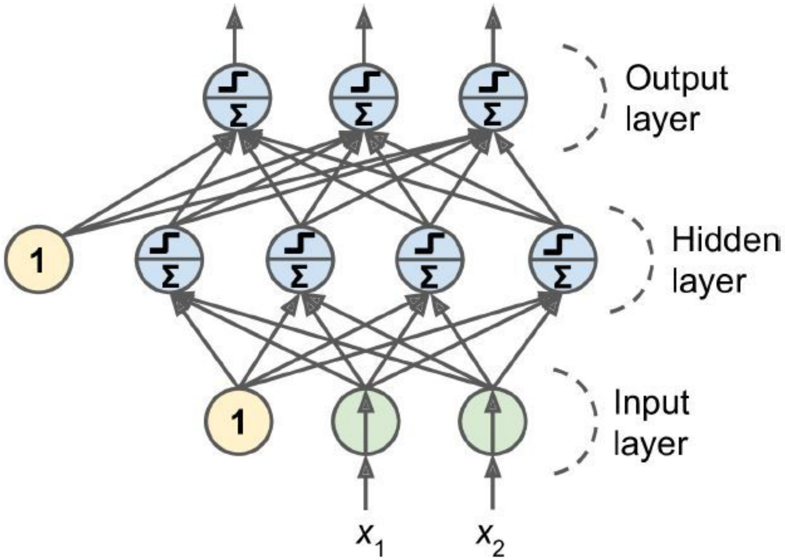


Multi-layer Neural Network

Let's assume we have multiple layers and they are also *dense* layers. These neural networks contain:

- an ***input layer*** (although this is not a layer of neurons);
- one or more ***hidden layers***;
- an ***output layer***.

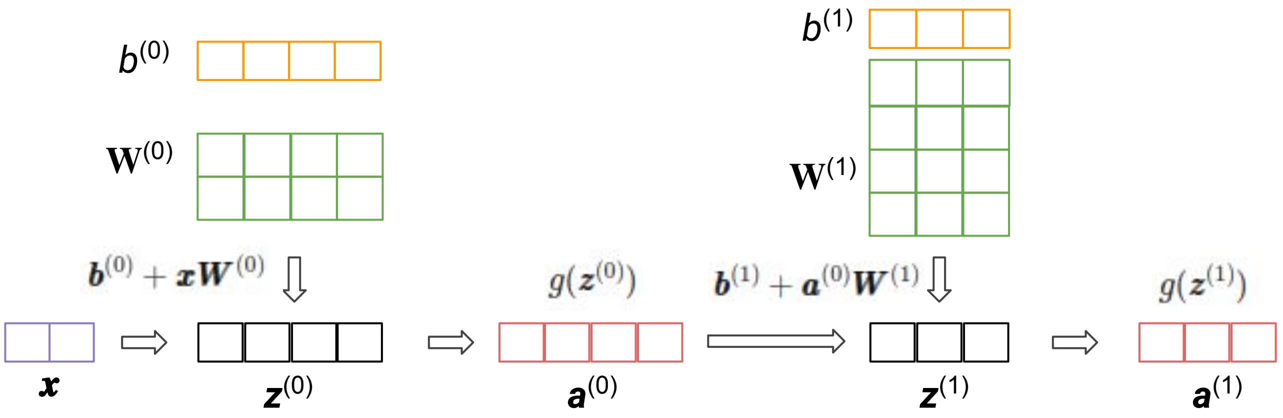
Every neuron has a *bias*.



- The network shown in the diagram is a ***layered***, ***dense***, ***feedforward*** network.
- The depth of a MLNN is simply the number of layers of neurons.

Matrix Multiplication Again

Similarly, we can obtain output for the second layer, and so on.



Matrix Multiplication Again

- When we make predictions for unseen examples, we often want predictions, not for a single object $\boldsymbol{x}$, but for a set of objects X.
  - This is also true during training, in the case of *Batch Gradient Descent* and *Mini-Batch Gradient Descent*.

$$\boldsymbol{Z}^{(0)} = \boldsymbol{b}^{(0)} + \boldsymbol{X}\boldsymbol{W}^{(0)} \text{ and } \boldsymbol{A}^{(0)} = g(\boldsymbol{Z}^{(0)})$$

$$\boldsymbol{Z}^{(1)} = \boldsymbol{b}^{(1)} + \boldsymbol{Z}^{(0)}\boldsymbol{W}^{(1)} \text{ and } \boldsymbol{A}^{(1)} = g(\boldsymbol{Z}^{(1)})$$

Matrix Multiplication Again

- This is all that a neural network consists of! They are just collections of:
  - matrix multiplications; and
  - element-wise activation functions.

- In general, they are collections of:
  - *affine transformations* (matrix multiplication being one example of an affine transformation, which are linear operations); and
  - element-wise functions (activation functions being one example).

- Looking at neural networks in this way also helps us realise that a neural network simply defines a function as a composite of other functions.

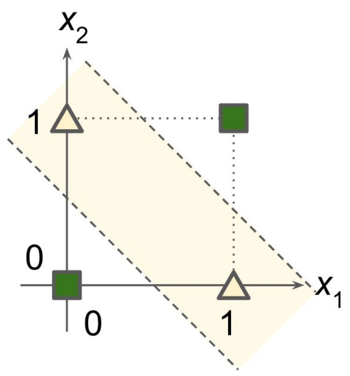
- In the example above, the whole network computes the following:

$$g^{(1)}(g^{(0)}(\boldsymbol{X}\boldsymbol{W}^{(0)} + \boldsymbol{b}^{(0)})\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})$$

Why Do We Need More Layers?

- A single neuron (or layer of neurons) gives us linear models.

- With linear models, there are problems we cannot solve, e.g., we cannot build a classifier that correctly classifies *exclusive-or*:

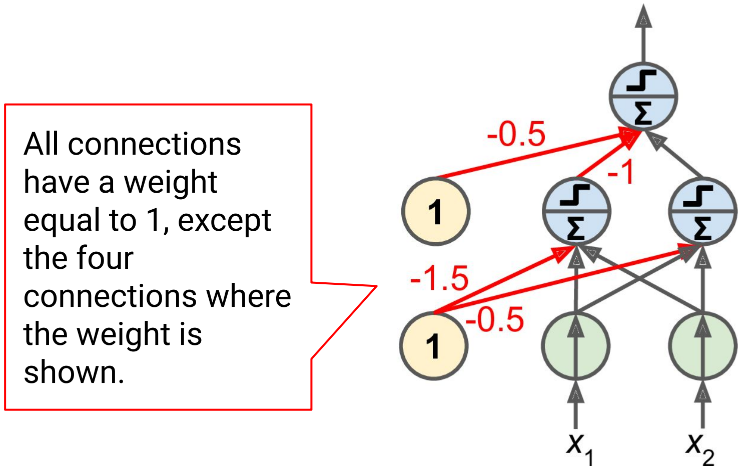


x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Note: A recent paper in Science Magazine claims that a single layer of biological neurons can compute exclusive-or. If true, this confirms what we said earlier: artificial neural networks are inspired by the human brain, but they are not a model of the human brain.

Why Do We Need More Layers?

But, a two-layer network that can correctly classify *exclusive-or*.



x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

So, with multiple layers of neurons and the non-linearities of their activation functions, we can eliminate these limitations.

Why Do We Need More Layers?

In general, MLNN has the following advantages:

- Other things being equal, each extra hidden layer enlarges the set of hypotheses that the network can represent: increasing complexity.
- In fact, the *universal approximation theorem* states that a feed-forward network with a finite (but arbitrarily large) single hidden layer can approximate any continuous function (to any desired precision), under mild assumptions on the activation function.

Training a Neural Network

Neural networks learn by modifying the values of the weights and biases.

- It is our job to decide on the neural network ***architecture*** (structure).
- It is our job to choose the values of numerous ***hyperparameters*** that we will encounter.
  - The hyperparameters of a neural network are the number of layers, number of neurons in each layer, activation function, loss function, optimizer, learning rate, batch size, etc.
- But, we use a dataset and a learning algorithm to find the values of the network's ***parameters***.
  - The parameters of a neural network are its *weights* and *biases*.

Training a Neural Network

- A lot of this is done using *supervised learning*:
  - So we need a *labeled dataset*;
  - a *loss function*; and
  - a learning algorithm known as *backpropagation* (or *backprop*) that uses some variant of *Gradient Descent.*

Next lecture

Neural Network Examples

17^{th} October 2023