ECS 171 Machine Learning

Lecture5: ANN- Feed Forward Neural Network
Instructor: Dr. Setareh Rafatirad

MLE for Logistic Regression Recap.

1. Formulate Logistic regression using the sigmoid function
2. Formulate logistic regression as a MLE problem
3. Use Gradient Descent to find the optimal parameters of the model

$$p\left(y^{(i)} \middle| x^{(i)}; w\right) = g\left(x^{(i)}; w\right)^{y^{(i)}} \left(1 - g\left(x^{(i)}; w\right)\right)^{1-y^{(i)}}$$

Assumption: Samples are iid

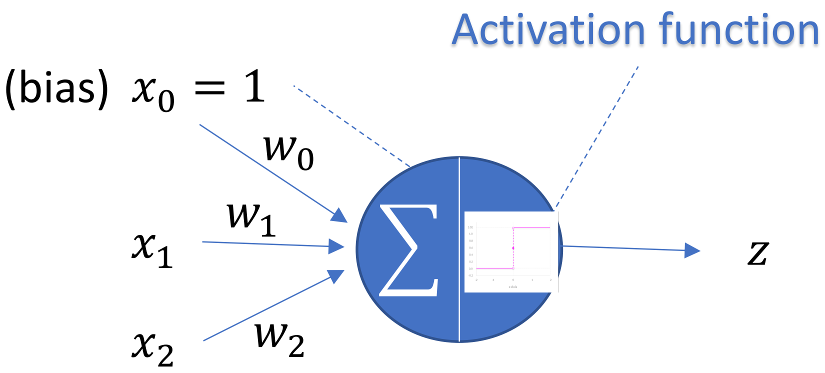
$$l(w) \triangleq \log p(D|w) = \sum_{i=1}^{N} \log p\left(y^{(i)} \middle| x^{(i)}; w\right)$$

$$= \sum_{i=1}^{N} \log\left(g\left(x^{(i)}; w\right)^{y^{(i)}} \left(1 - g\left(x^{(i)}; w\right)\right)^{1-y^{(i)}} \right)$$

$$= \boldsymbol{\sum_{i=1}^{N} \left(y^{(i)} \log g\left(x^{(i)}; w\right) + \left(1 - y^{(i)}\right) \log\left(1 - g\left(x^{(i)}; w\right)\right)\right)}$$

Perceptron Learning Algorithm Recap.

- The perceptron model takes an input , aggregates it that is calculates the weighted sum, and then returns 1 if the weighted sum is more than a threshold, or else returns 0.



- Perceptron is guaranteed to converge if the data is linearly separable and if the learning rate is sufficiently small.
- Perceptron uses the combination function, it does not use the sigmoid function (unlike logistic regression).
- Perceptron uses a more basic procedure compared to logistic regression. The output does not provide a measure of uncertainty.

Update Rule in Logistic Regression:

$$w_j = w_j + a\left(y^{(i)} - g\left(x^{(i)}; w\right)\right) x_j^{(i)}$$

Update Rule in Perceptron Learning:

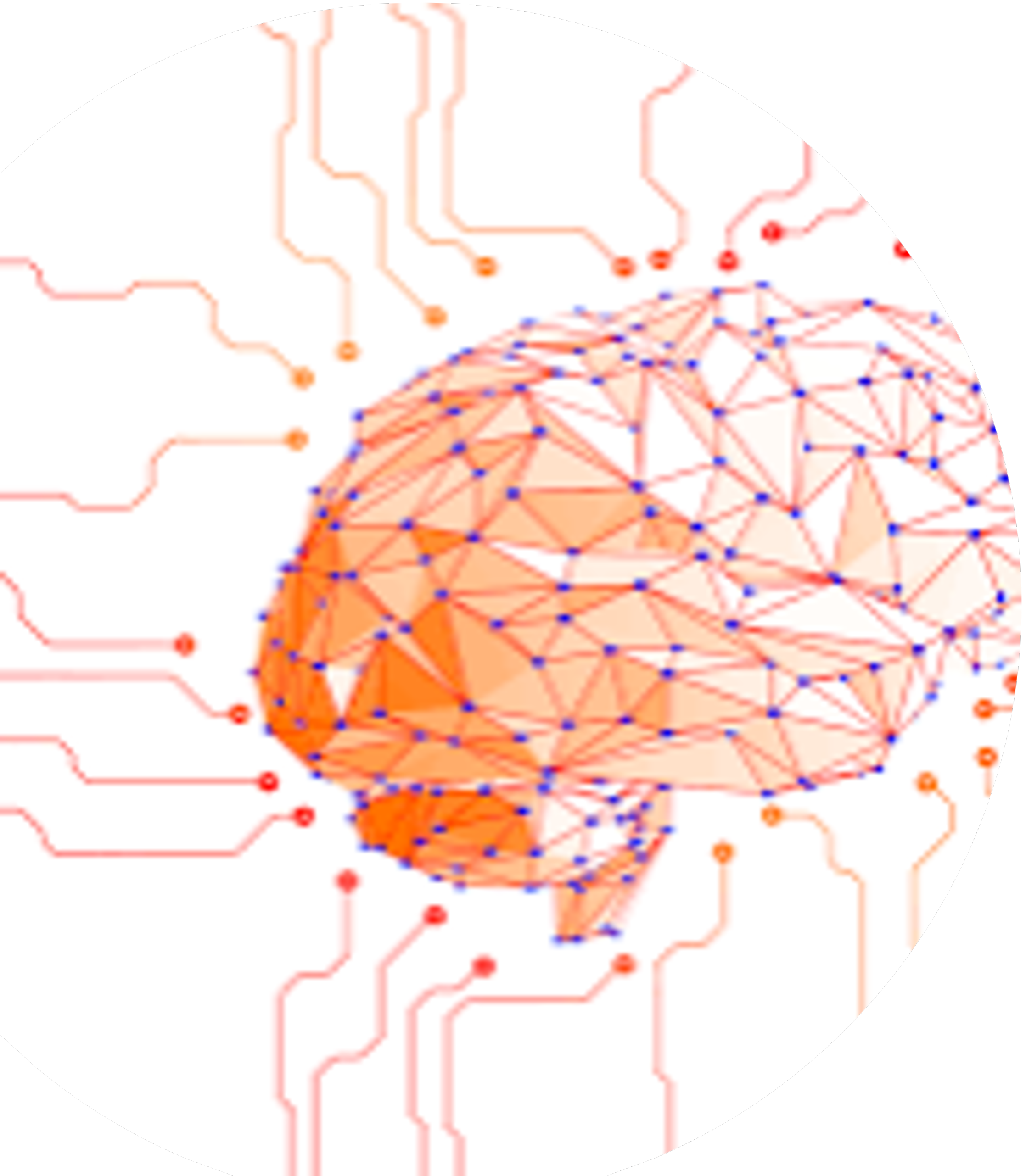
$$w_j = w_j + a\left(y^{(i)} - g(z)\right) x_j^{(i)}$$

Combination function:

$$z = w^T x = w_0 + w_1 x_1 + w_2 x_2$$

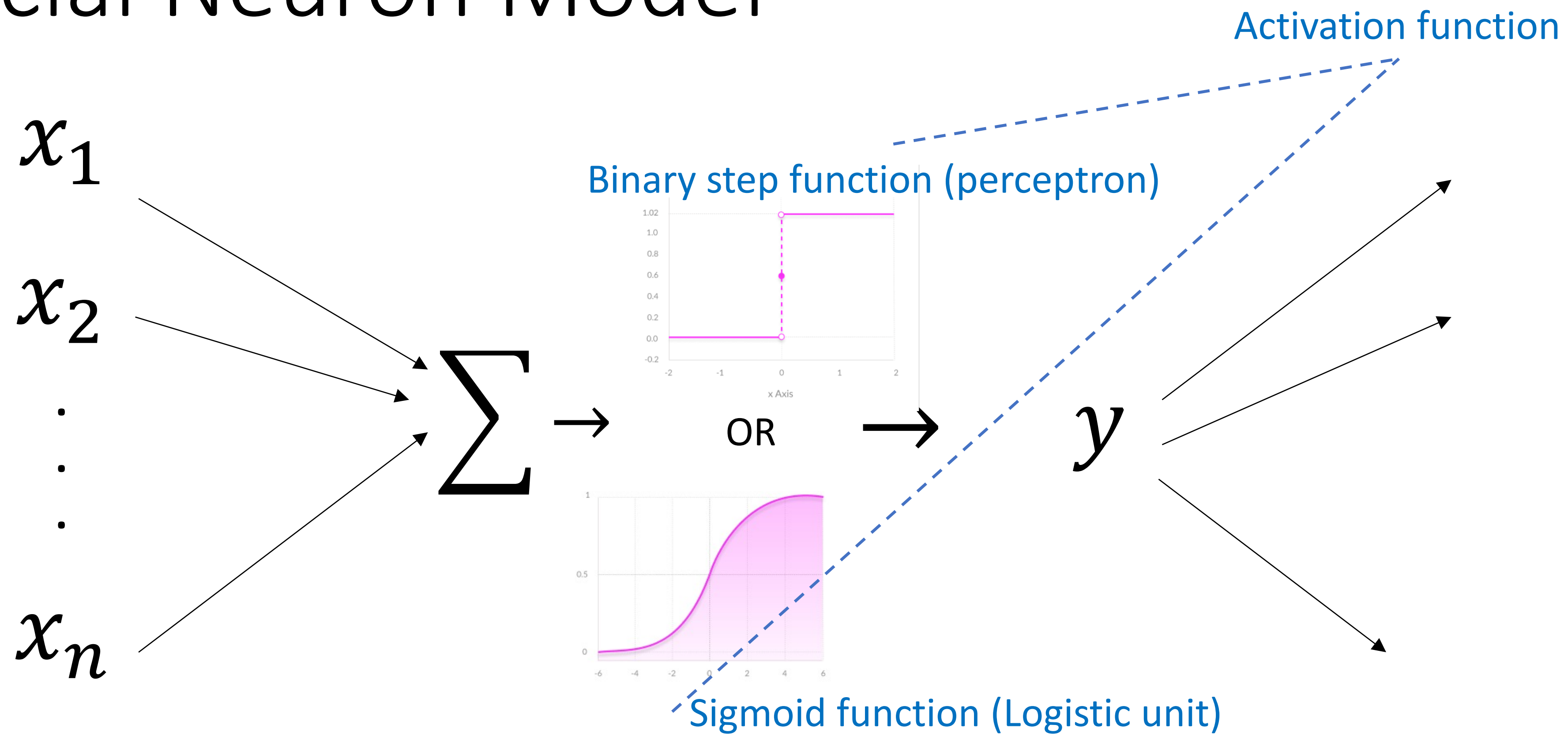
Threshold can be any scalar value (such as 0)

$$g(z) = \begin{cases} 0 \; if \;\; z < threshold \\ 1 \; if \;\; z \geq threshold \end{cases}$$



Introduction to Neural Networks

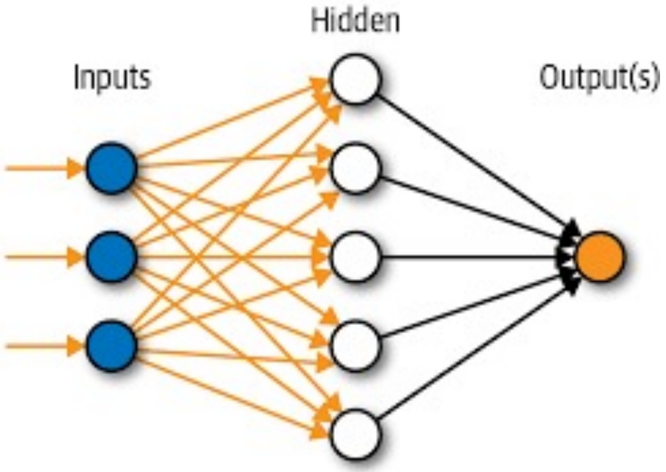
Artificial Neuron Model



A neuron in human combines information arriving from multiple neurons.

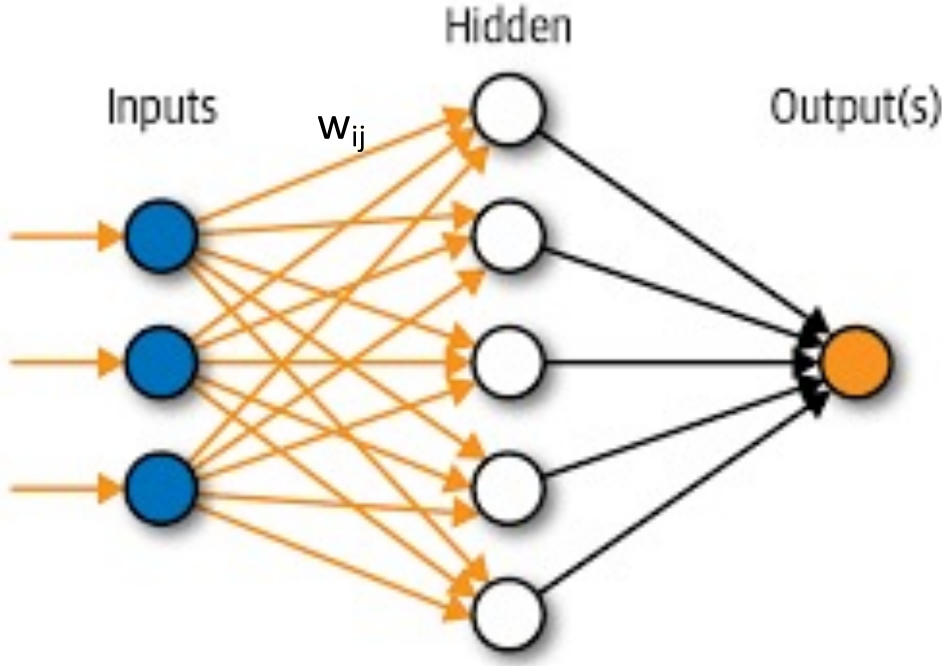
Neural Networks (NN): Number of Layers

- A NN consists of a <u>layered</u>, <u>feedforward</u>, <u>completely connected</u> network of neurons. Also called a Multi-Layer-Perceptron (MLP).
- Layers : input layer, hidden layer, output layer
- A Feed-Forward NN (FFNN) is composed of two or more layers, but mostly 3 layers, with activation functions usually step or logistic function.
- A NN without a hidden later, (i.e., with two layers of input and output) is a perceptron model. (AND, OR problems can be solved, but not XOR problem).
- The activation function in NN makes them non-linear regressors.
- A NN without an activation function is a linear regressor. So, The final layer can be another logistic regression/perceptron (such as sigmoid, tanh, or softmax) or a linear regression model (such as no activation function) depending whether it is a classification or regression problem.
- Some networks may have more than one hidden layers, but in general 1-2 hidden layers is sufficient.
- Too many hidden layers increases the training time, especially when the "error correction" is propagated backwards.

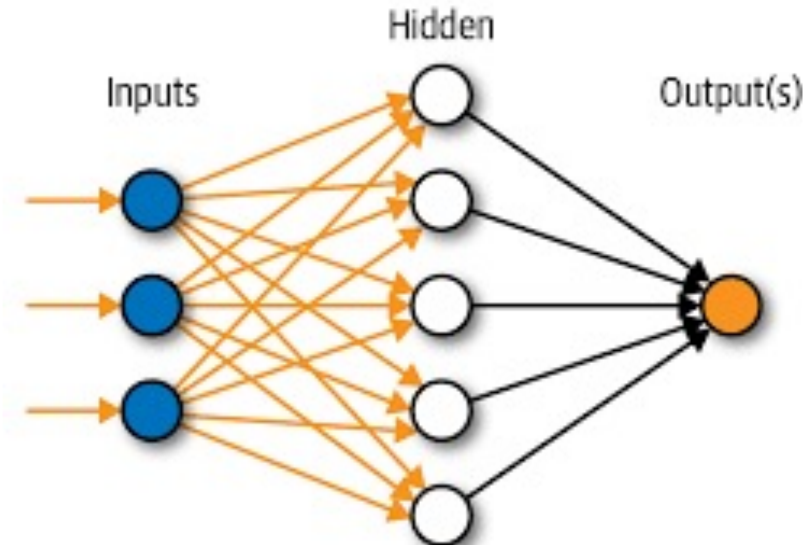


Feed-forward Neural Network (FFNN) : Number of Nodes per Layer

- Completely connected
- Weights are values between 0 and 1
- Number of nodes in the input layer depends on the number and types of dataset attributes
- The number of nodes (i.e., neurons) in the output layer may be more than 1 depending on the classification task.
- The number of nodes in the hidden layer depends on the complexity of the pattern. An overly large number of nodes can cause overfitting.
    - In case of overfitting, reduce the number of nodes in the hidden layer.
    - In case of low accuracy, increase the number of nodes in the hidden layer.
    - Determined by Trial-and-error



Neural Networks (NN): pros and cons



A NN consists of a layered, feedforward, completely connected network of neurons.

Pros:

- Robust and resilient to noise
- Can model non-linearity

Cons:

- Hard to interpret by humans
- Require relatively long training time
- All attributes (continuous and categorical variables) must be encoded in a standardized manner, taking values between 0 and 1 (sklearn.preprocessing.MinMaxScaler).

Examples

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
>>> scaler = MinMaxScaler()
>>> print(scaler.fit(data))
MinMaxScaler()
>>> print(scaler.data_max_)
[ 1. 18.]
>>> print(scaler.transform(data))
[[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
>>> print(scaler.transform([[2, 2]]))
[[1.5 0. ]]
```

Requirements of NN: Standardizing Attributes

- Continuous variables such as *"height"*
  - Apply min-max normalization
  - In Python, you can also use scikit-learn object ***MinMaxScaler***:

```
from sklearn.preprocessing import MinMaxScaler
```

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

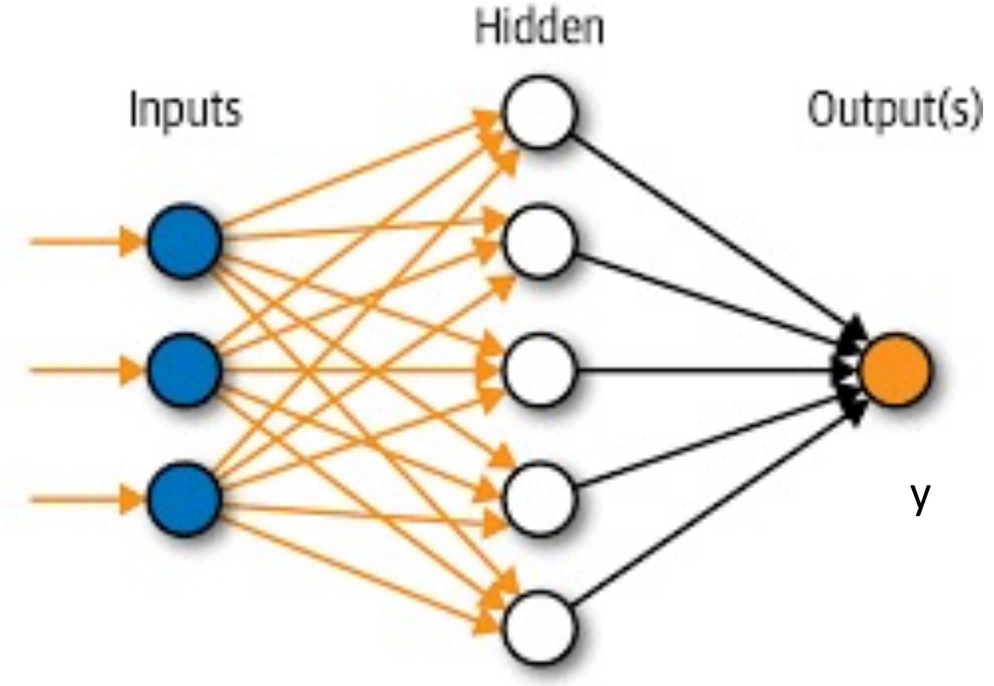
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Categorical variables such as *"gender"*
  - Flag variables if there aren't too many categorical variables!
  - Example: gender is an attribute that can take these labels: *male*, *female*, *unknown*. The number of labels is 3. You need (k-1 = 2) flag variables:

	male	female
Male sample	1	0
Female sample	0	1
Unknown sample	0	0

One-output Node Application

One output node (y) is good for binary classification problems Example: “if a team **wins** or **loses**”. Good for *binary classification*.



One output node (y) is also good <u>when the output classes are ordered</u>. e.g.,

$$\begin{aligned}
&if\ 0 \le output < threshold1 &&: classify\ 1st\ place \\
&if\ threshold1 \le output < threshold2 &&: classify\ 2nd\ place \\
&if\ threshold2 \le output &&: classify\ 3rd\ place
\end{aligned}$$

In this case, works for *multi-class classification*.

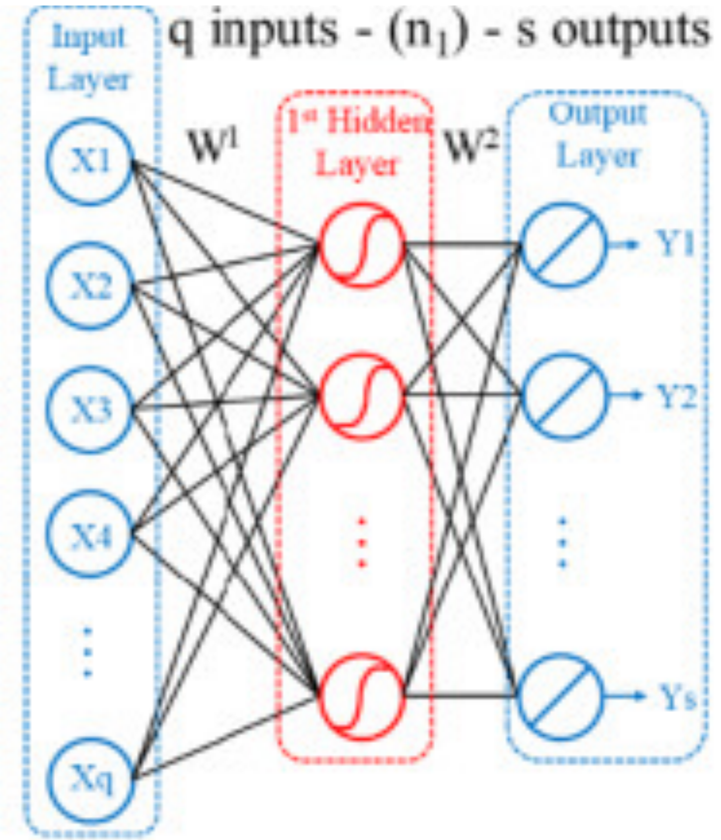
1-of-n output Encoding

There are more than one output nodes in the output layer.

Output classes are not ordered. e.g., gender: {male, female, unknown}

Each output node corresponds to one class label, quantified with a probability.

Benefit: it provides probabilities used as a measure of confidence in the classification.



Basic Example of FFNN

Nx4 input matrix
N: number of samples
4: number of attributes including x_0
x_0 *is* 1 *by convention*

$$\begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & & \ddots & \vdots \\ x_0^{(n)} & & \cdots & x_3^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix}$$

$$y^{(i)} = \sum_{j=0}^{M} w_j x_j^{(i)} = \omega_0 + \omega_1 x_1^{(i)} + \omega_2 x_2^{(i)} + \omega_3 x_3^{(i)} + \cdots + \omega_n x_n^{(i)}$$

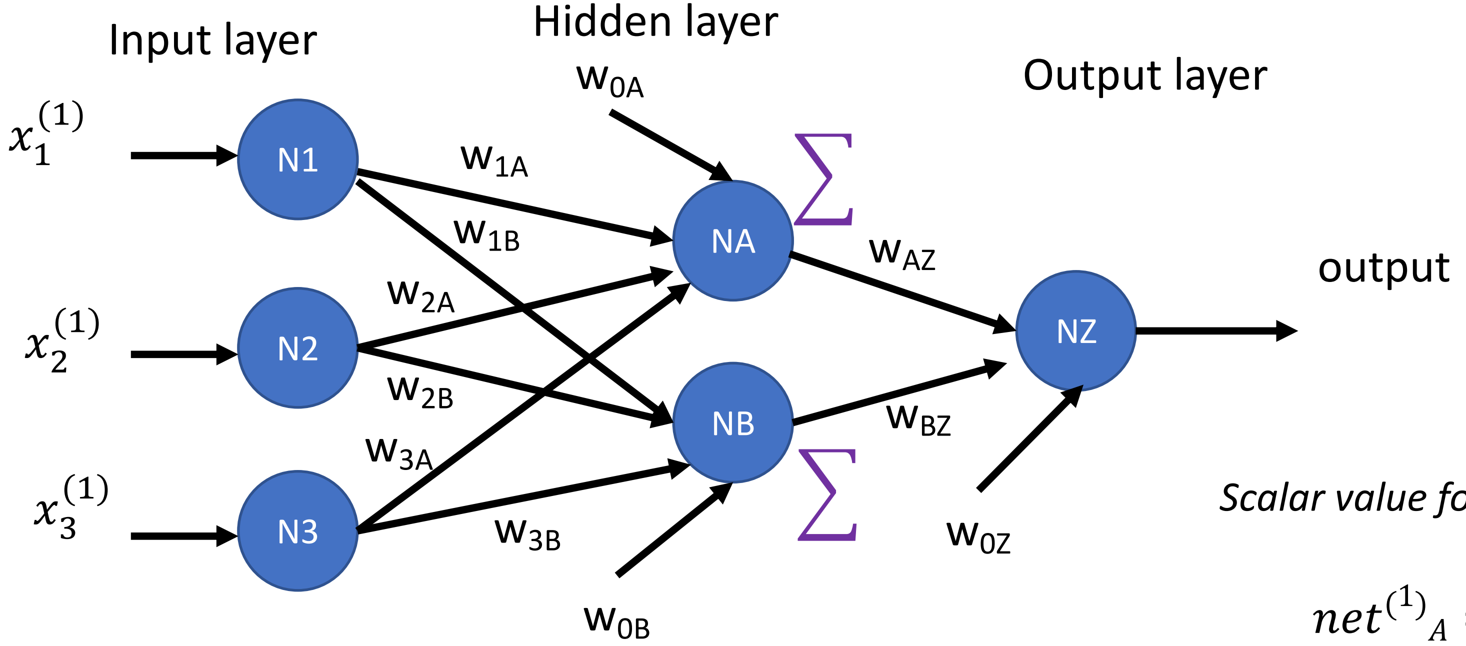
Basic Example of FFNN: Combination Function

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & \ddots & & \vdots \\ 1 & \cdots & & x_3^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0.4 & 0.2 & 0.7 \\ \vdots & \ddots & & \vdots \\ 1 & \cdots & & x_3^{(n)} \end{bmatrix}$$

Let's show the structure of hidden layer nodes and output layer nodes using the first sample in the dataset D.

w_{ij}: the weight associated with the i^{th} input to node j

x_{ij}: i^{th} input to node j



w_{0A} =0.5	w_{0B} =0.7	w_{0Z}=0.5
w_{1A} = 0.6	w_{1B} = 0.9	w_{AZ}=0.9
w_{2A}=0.8	w_{2B}=0.8	w_{BZ}=0.9
w_{3A}=0.6	w_{3B}=0.4	

Scalar value for node j: $net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)}$

Sigma is the combination function.

$$net^{(1)}{}_A = \omega_{0j} + \omega_{1j} x_{1j}^{(1)} + \omega_{2j} x_{2j}^{(1)} + \omega_{3j} x_{3j}^{(1)} = 1.32$$

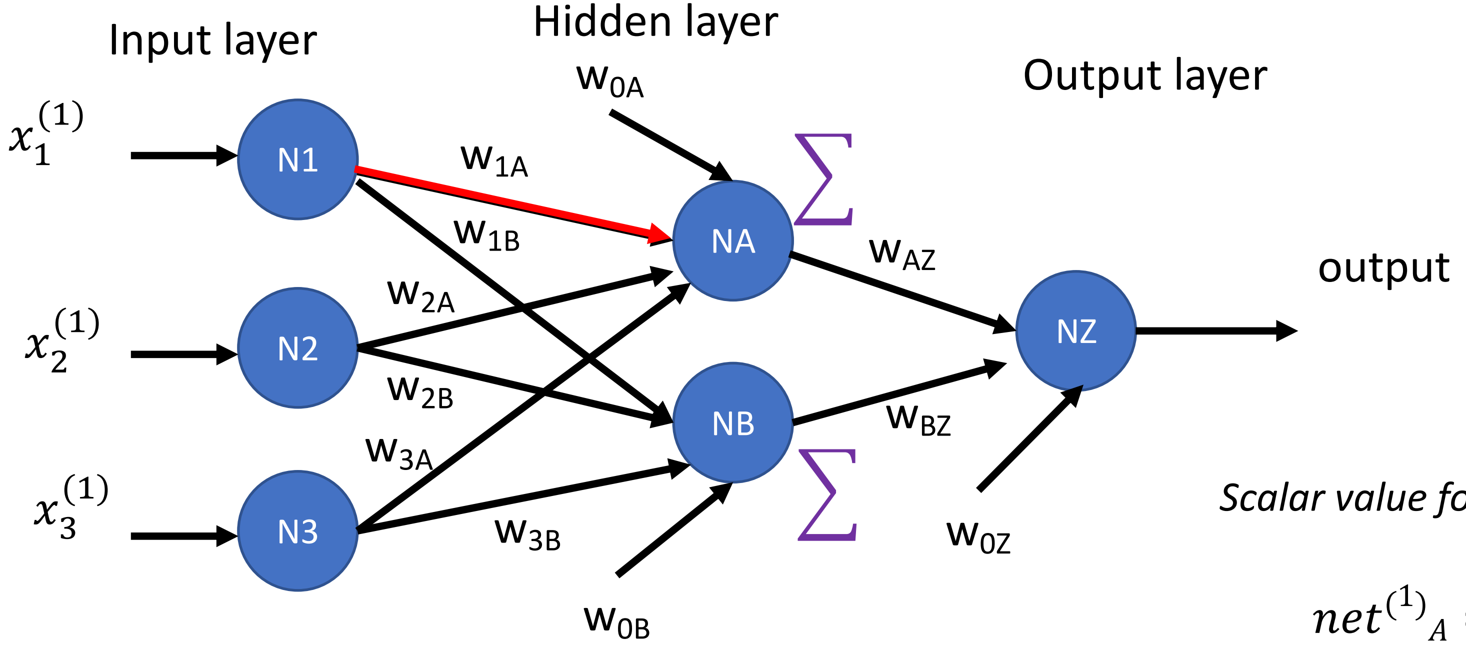
Source: Discovering Knowledge in Data D. Larose

Basic Example of FFNN: Combination Function

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0.4 & 0.2 & 0.7 \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix}$$

Let's show the structure of hidden layer nodes and output layer nodes using the first sample in the dataset D.

w_{ij}: the weight associated with the i^{th} input to node j
x_{ij}: i^{th} input to node j



w_{0A} =0.5	w_{0B} =0.7	w_{0Z}=0.5
w_{1A} = 0.6	w_{1B} = 0.9	w_{AZ}=0.9
w_{2A}=0.8	w_{2B}=0.8	w_{BZ}=0.9
w_{3A}=0.6	w_{3B}=0.4	

Scalar value for node j: $net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)}$

Sigma is the combination function.

$$net^{(1)}{}_A = \omega_{0j} + \omega_{1j} x_{1j}^{(1)} + \omega_{2j} x_{2j}^{(1)} + \omega_{3j} x_{3j}^{(1)} = 1.32$$

Source: Discovering Knowledge in Data D. Larose

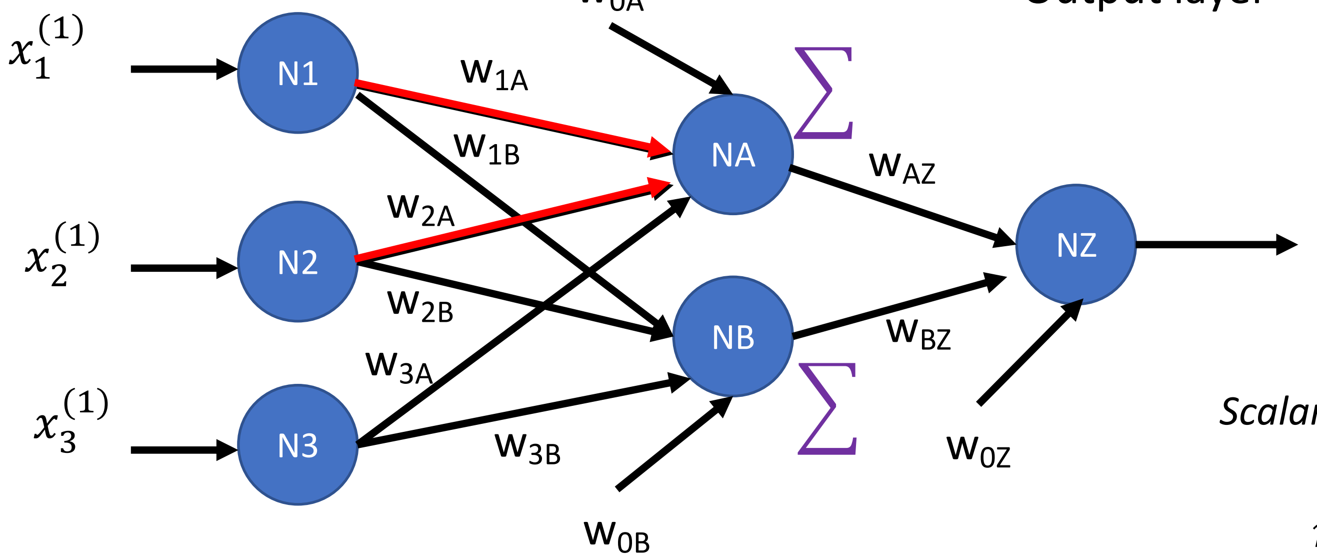
Basic Example of FFNN: Combination Function

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0.4 & 0.2 & 0.7 \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix}$$

Let's show the structure of hidden layer nodes and output layer nodes using the first sample in the dataset D.

w_{ij}: the weight associated with the i^{th} input to node j

x_{ij}: i^{th} input to node j



w_{0A} =0.5	w_{0B} =0.7	w_{0Z}=0.5
w_{1A} = 0.6	w_{1B} = 0.9	w_{AZ}=0.9
w_{2A}=0.8	w_{2B}=0.8	w_{BZ}=0.9
w_{3A}=0.6	w_{3B}=0.4	

Scalar value for node j: $net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)}$

Sigma is the combination function.

$$net^{(1)}{}_A = \omega_{0j} + \omega_{1j} x_{1j}^{(1)} + \omega_{2j} x_{2j}^{(1)} + \omega_{3j} x_{3j}^{(1)} = 1.32$$

Source: Discovering Knowledge in Data D. Larose

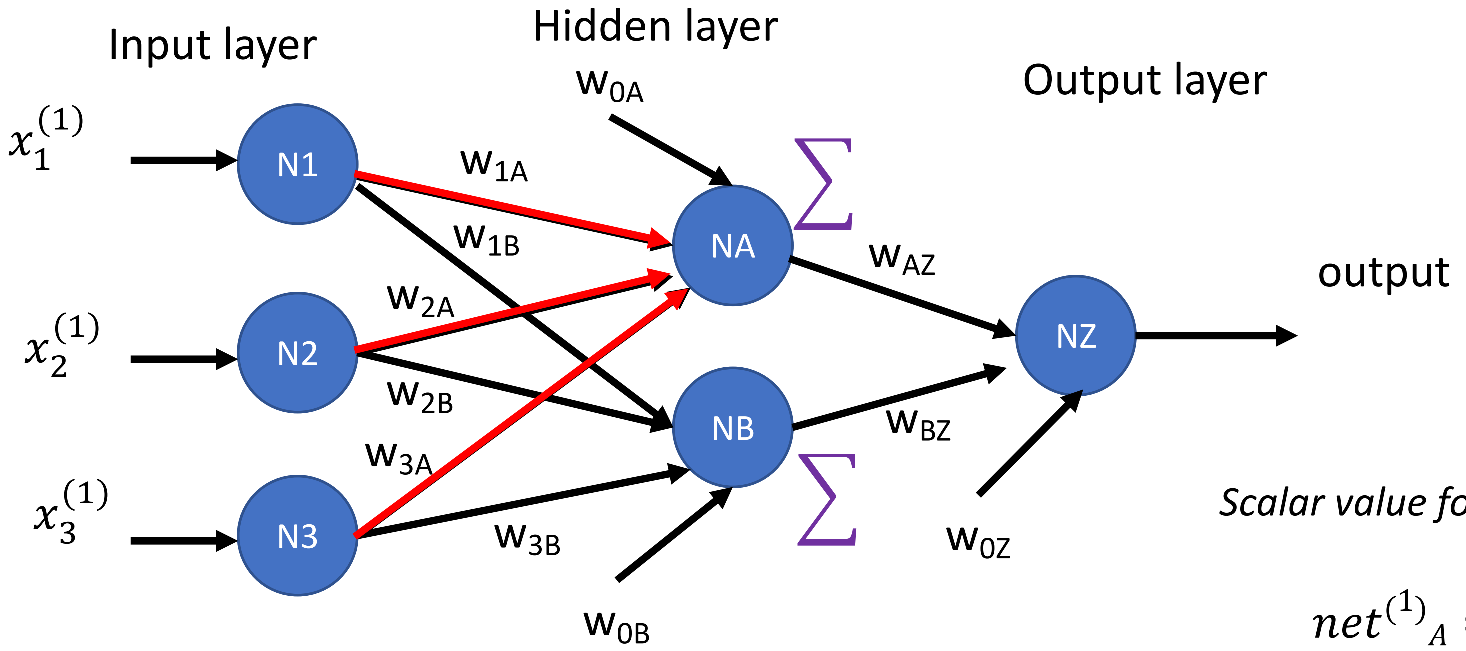
Basic Example of FFNN: Combination Function

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & \ddots & & \vdots \\ 1 & \cdots & & x_3^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0.4 & 0.2 & 0.7 \\ \vdots & & \ddots & \vdots \\ 1 & & \cdots & x_3^{(n)} \end{bmatrix}$$

Let's show the structure of hidden layer nodes and output layer nodes using the first sample in the dataset D.

w_{ij}: the weight associated with the i^{th} input to node j

x_{ij}: i^{th} input to node j



w_{0A} =0.5	w_{0B} =0.7	w_{0Z}=0.5
w_{1A} = 0.6	w_{1B} = 0.9	w_{AZ}=0.9
w_{2A}=0.8	w_{2B}=0.8	w_{BZ}=0.9
w_{3A}=0.6	w_{3B}=0.4	

Scalar value for node j: $net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)}$

Sigma is the combination function.

$$net^{(1)}{}_A = \omega_{0j} + \omega_{1j} x_{1j}^{(1)} + \omega_{2j} x_{2j}^{(1)} + \omega_{3j} x_{3j}^{(1)} = 1.32$$

Source: Discovering Knowledge in Data D. Larose

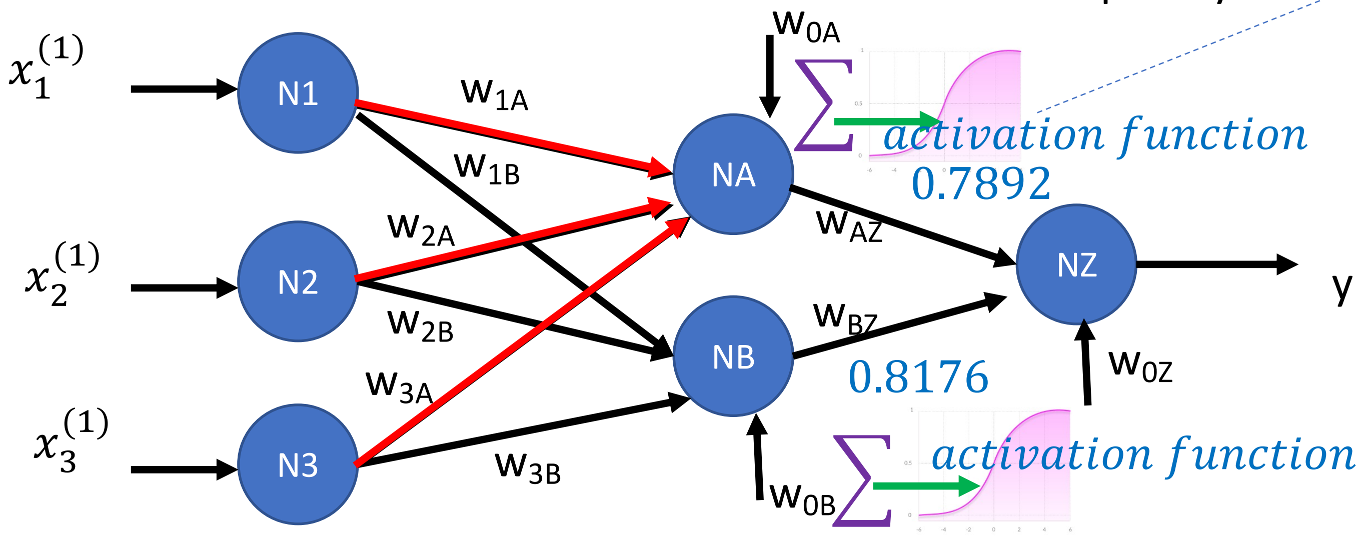
Basic Example of FFNN: Activation Function

$$net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)}$$

Input to the activation function

$$net^{(1)}{}_A = \omega_{0j} + \omega_{1j} x_{1j}^{(1)} + \omega_{2j} x_{2j}^{(1)} + \omega_{3j} x_{3j}^{(1)} = 1.32$$

$$predicted\ output\ y = \frac{1}{1+e^{-x}} = f(net_j)$$



$$f(net_A) = \frac{1}{1 + e^{-(1.32)}} = 0.7892$$

$$f(net_B) = 0.8176$$

There are various types of activation functions applied for classification (softmax, sigmoid, tanh) and regression (linear).

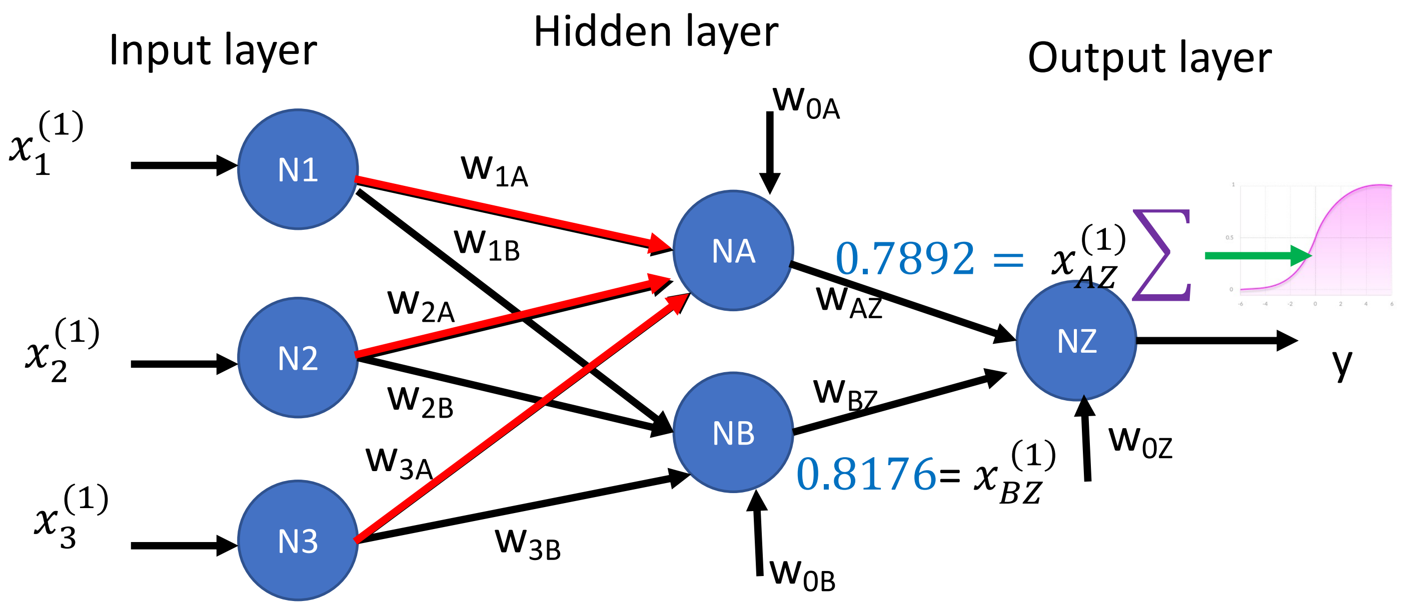
Source: Discovering Knowledge in Data D. Larose

Basic Example of FFNN: Output Node

$$net^{(1)}{}_Z = \sum_k w_{kZ} x_{kZ}^{(1)} = \omega_{0Z} + \omega_{AZ} x_{AZ}^{(1)} + \omega_{BZ} x_{BZ}^{(1)} = 0.5 + 0.9(0.7892) + 0.9(0.8176) = 1.9461$$

Input to the activation function

$$y = \frac{1}{1+e^{-x}} = f(net_j)$$



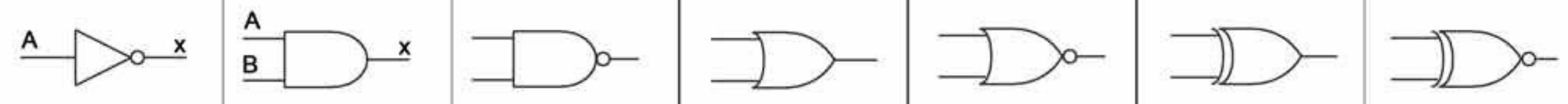
$$f(net_Z) = \frac{1}{1 + e^{-(1.9461)}} = 0.8750$$

Output from the NN for pass 1 through the network , and is the predicted value for the first observation in the dataset D.

Source: Discovering Knowledge in Data D. Larose

Emulating Boolean Functions with a NN

Logical Gates

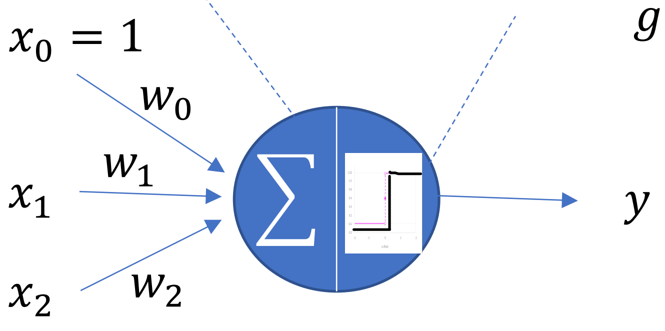
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Source: https://medium.com/autonomous-agents/how-to-teach-logic-to-your-neuralnetworks-116215c71a49

Neural Network as a Logical AND Gate

Combination function: $z = w^T x = w_0 + w_1 x_1 + w_2 x_2$

Activation function $g(z) = \begin{cases} 0 \ ; if \ g(z) < 0.5 \\ 1 \ ; if \ g(z) \geq 0.5 \end{cases}$



Goal: find the value of weights which will enable the network to act as a particular gate.

AND Gate Truth Table

x_1	x_2	y: $g(x; w)$
0	0	0
0	1	0
1	0	0
1	1	1

Weights for the input layer

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \\ 2 \end{bmatrix}$$

1) Case (0,0) : $w_0 + w_1 x_1 + w_2 x_2 = w_0$ = -3 ➔ y =0
2) Case (0,1) : $w_0 + w_2 x_2$ = -3 + (2)1 = -1 ➔ y=0
3) Case (1,0) : $w_0 + w_2 x_2$ = -3 + (2)1 = -1 ➔ y=0
4) Case (1,1) : $w_0 + w_1 x_1 + w_2 x_2$ = -3 + (2)1 +(2)1 = 1 ➔ y=1

$-3 + 2x_1 + 2x_2 = 0$ ➔ $x_1 + x_2 = 3/2$