UC DAVIS PRE-MED AMSA

vitalant Blood Drive

February 14th, 15th, and 16th

Earn $10 for donating ($20 if you donate on the 14th!)

Visit: https://www.vitalant.org/ucdavis

SCAN THE QR CODE TO DONATE

Eligibility: tinyurl.com/VitalantDonationEligibility

Interested in Volunteering? Sign up here:
http://tinyurl.com/BDVolunteerFeb2024

MEMBERSHIP POINT OPPORTUNITIES:

Donating whole blood → 5 points
Donating power red → 6 points
Volunteering → 4 points

UC DAVIS PRE-MED AMERICAN MEDICAL STUDENT ASSOCIATION

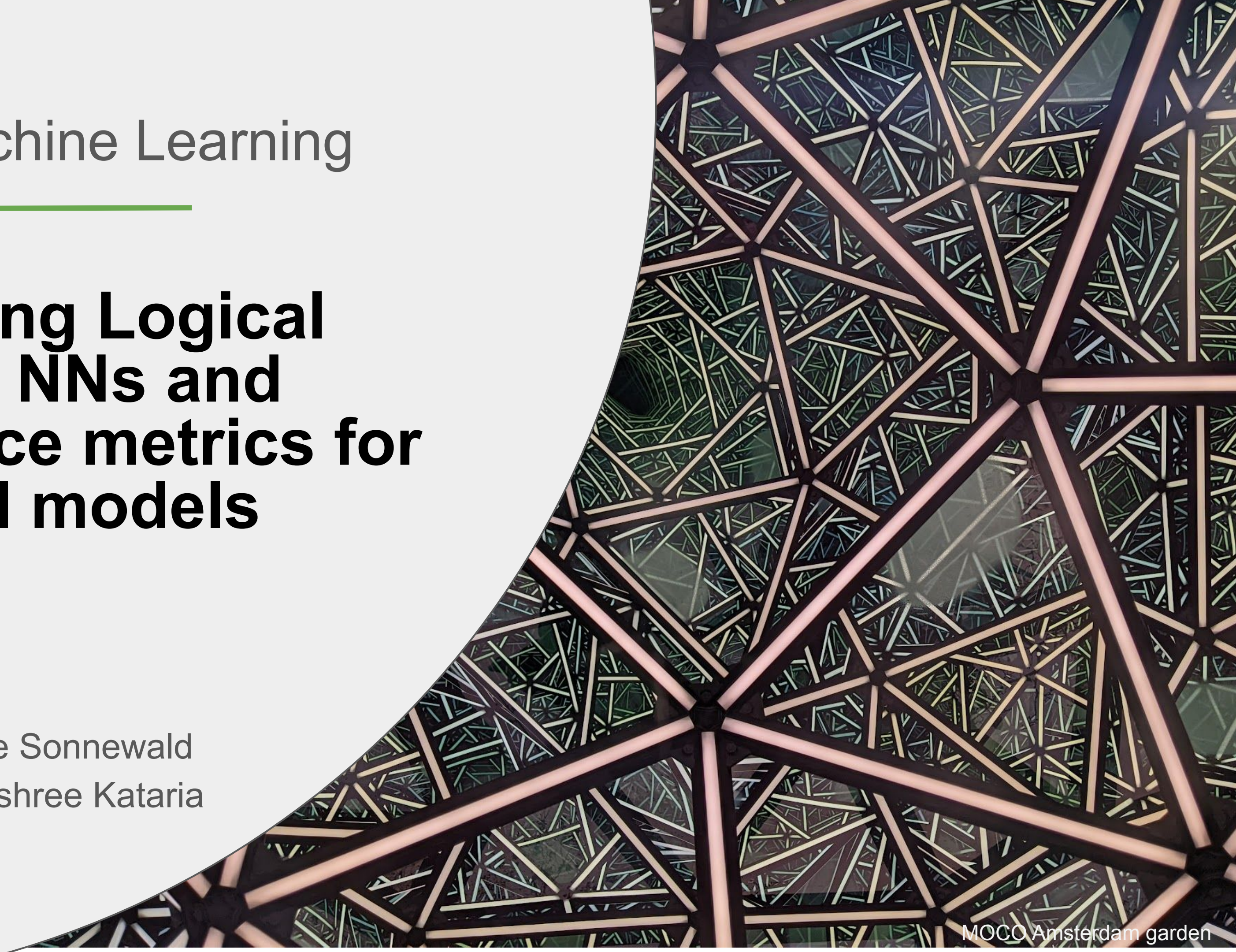
UC DAVIS STUDENT HEALTH AND COUNSELING SERVICES

QUESTIONS:
ucdblooddrives@gmail.com

ECS171: Machine Learning

L7 Emulating Logical Gates with NNs and performance metrics for categorical models

Instructor: Prof. Maike Sonnewald
TAs: Pu Sun & Devashree Kataria



MOCO Amsterdam garden

Intended learning outcomes

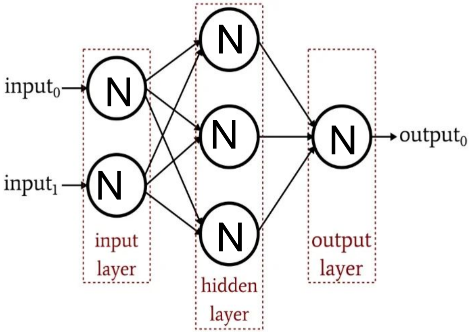
- Appreciate the concept of Boolean functions and how they relate to neural networks including perceptrons and multilayer perceptrons
  - Apply and explain the reasoning behind the AND and NOR gate, as well as XOR
- Know the various metrics used for determining classification skill and how various combinations are beneficial for different use cases
  - Be able to apply the calculations

Recall: Feed-Forward pass in a neural network

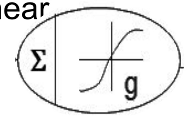
- Process input data through the network to receive output
- The data flows only ‘forwards’ from the input layer
- In a ‘fully-connected’ network, such as a normal Multilayer Perceptron, every node in one layer is connected to every node in the *next* layer with an assigned weight
- In each node, we take the Linear Combination of Inputs (aka weighted sum):

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

- The Linear Combination of Inputs z is passed through an ‘activation function’ to give the output from the given node
- This process is done for each node in a layer, and passed to the next layer



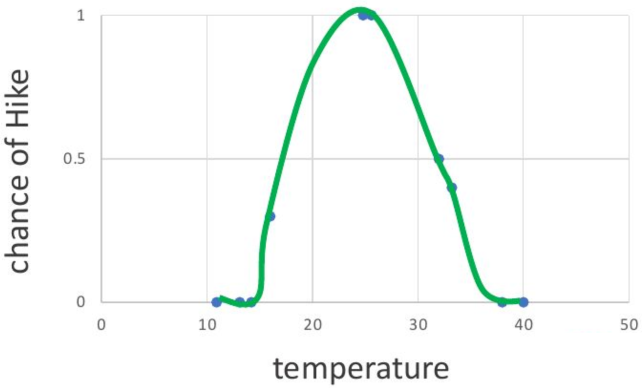
Inside one node ‘N’: each has the Linear Combination of Inputs (left) and activation (right)



Hike or no hike?

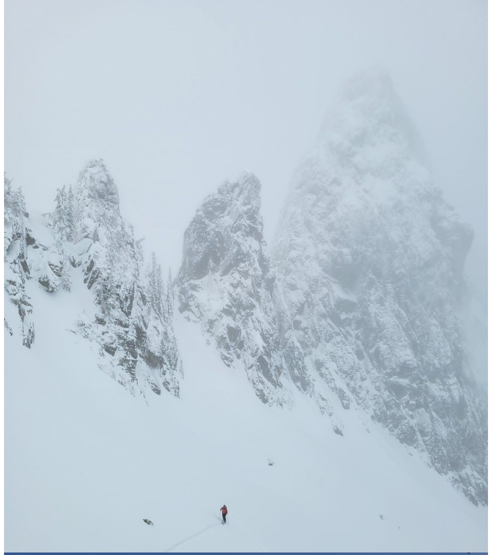
Input: Temperature (oC)

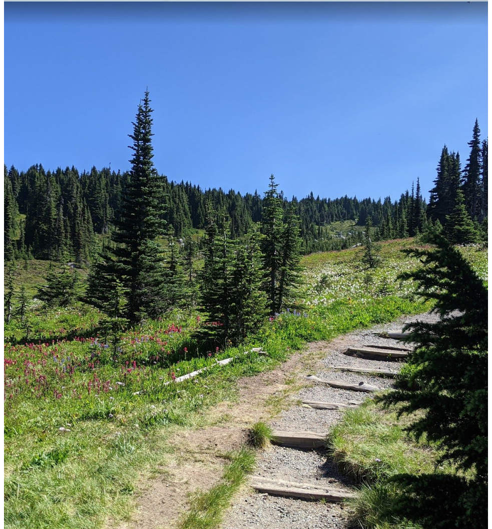
Output: Chance of hike



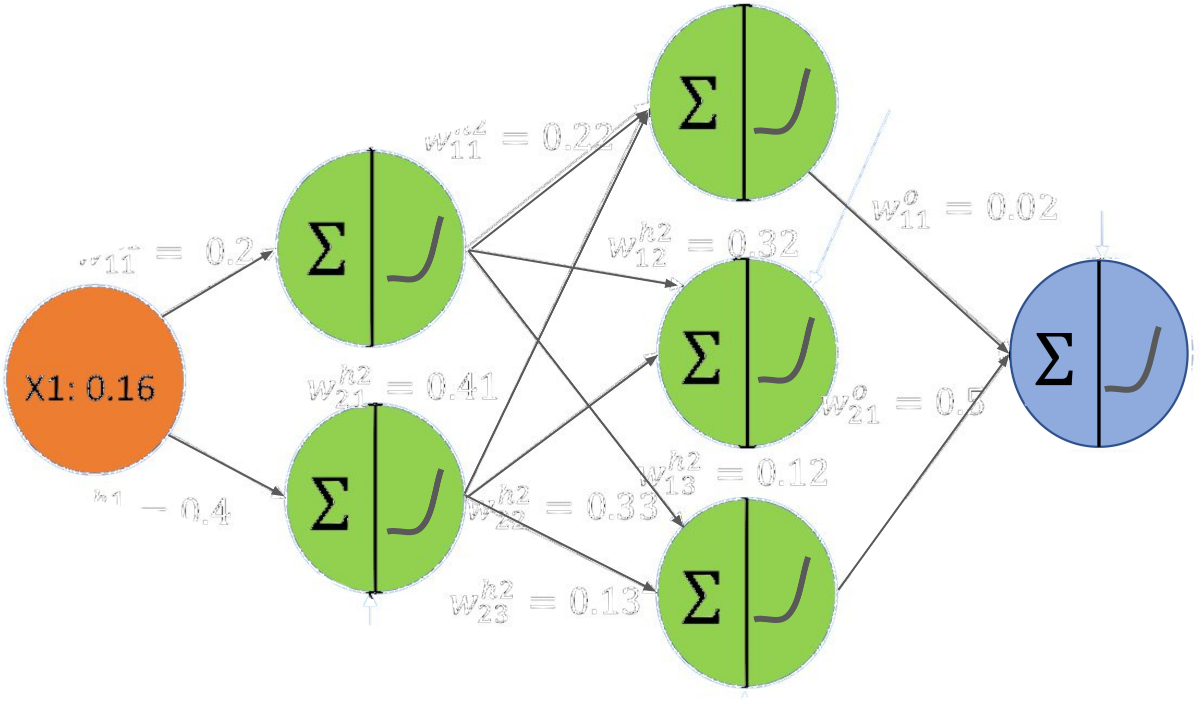
We can model the curve with a NN

Temperature	Hike
14.2	0
10.9	0
13.1	0
25.6	1
24.8	1
38	0
40	0
32	0.5
33.2	0.4
16	0.3





Are we going for a hike?

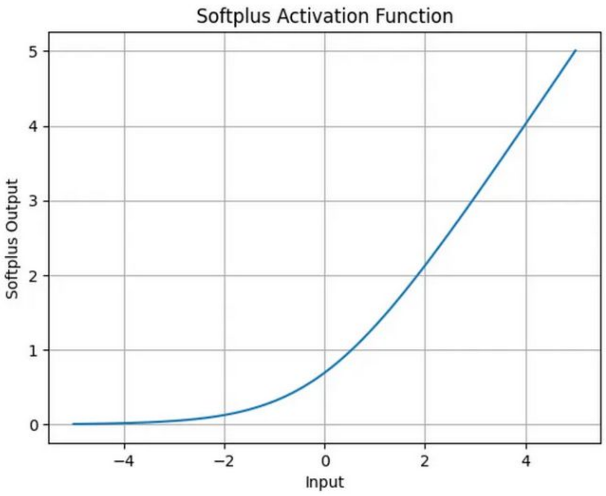


Are we going for a hike? ‘Softplus’ activation function

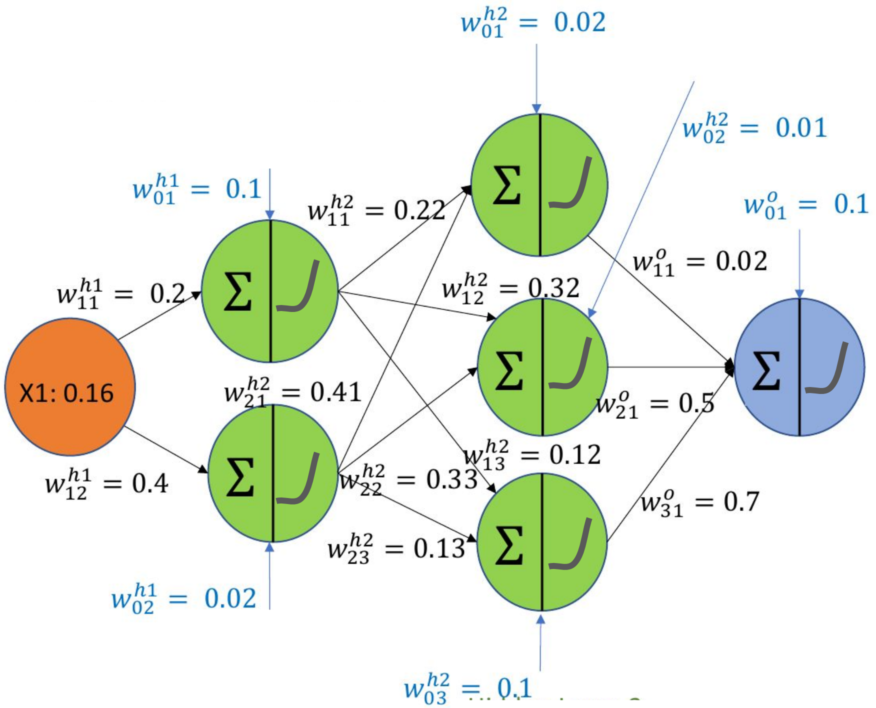
- Example using the ‘Softplus’
- The Softplus activation function is a smooth and continuously differentiable:

$$g(x) = \log(1-e^{z})$$

- Maps negative values to zero and keeps positive values unchanged
- Being continuously differentiable can be beneficial for training
- Note: Softplus function tends to saturate for large positive input values which can lead to a ‘vanishing gradient’



Are we going for a hike?



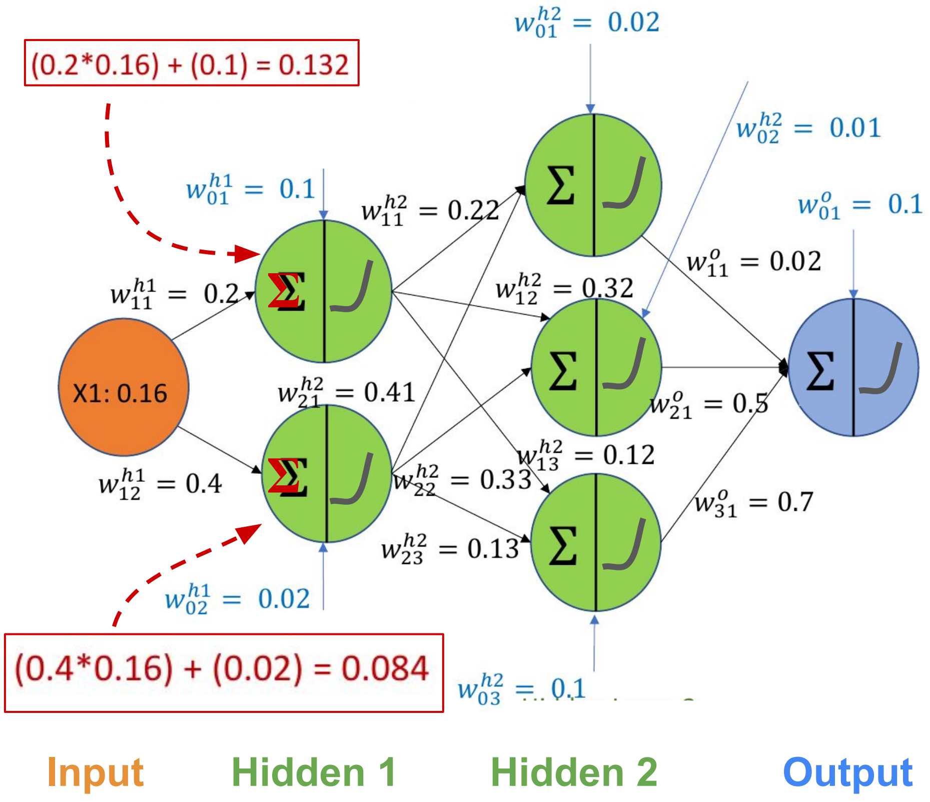
Input Hidden 1 Hidden 2 Output

Temperature	Hike
0.16	0
0.0	0
0.07560137	0
0.50515464	1
0.47766323	1
0.93127148	0
1.0	0
0.72508591	0.5
0.76632302	0.4
0.17525773	0.3

Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

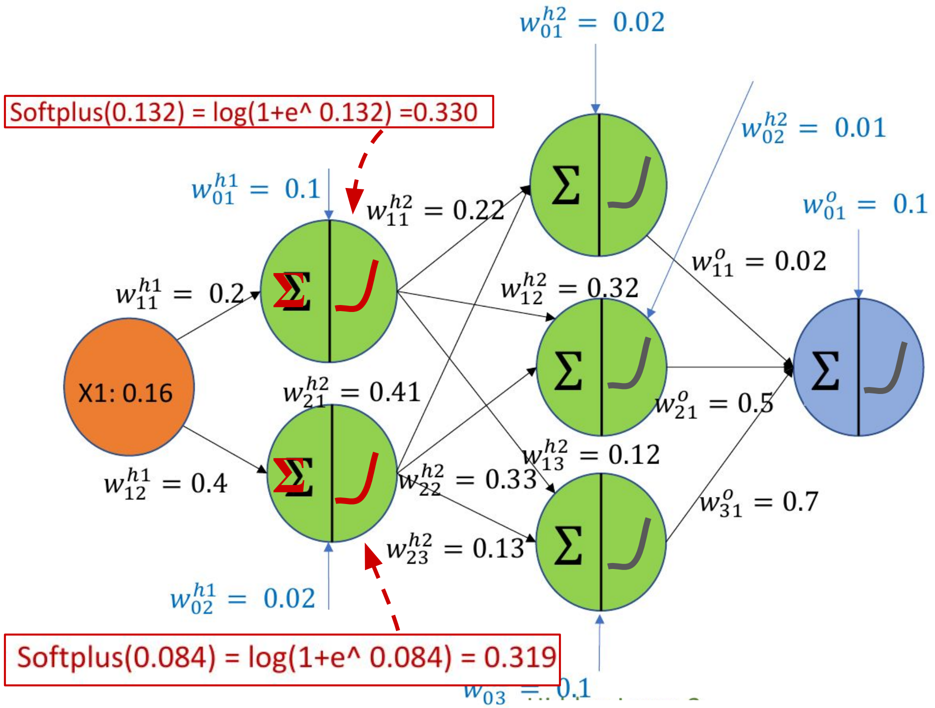
Are we going for a hike?

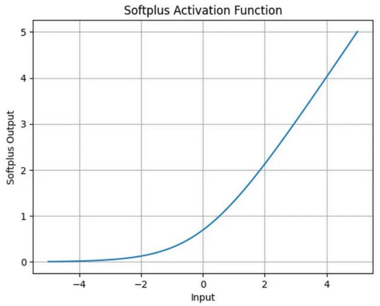


Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Are we going for a hike?

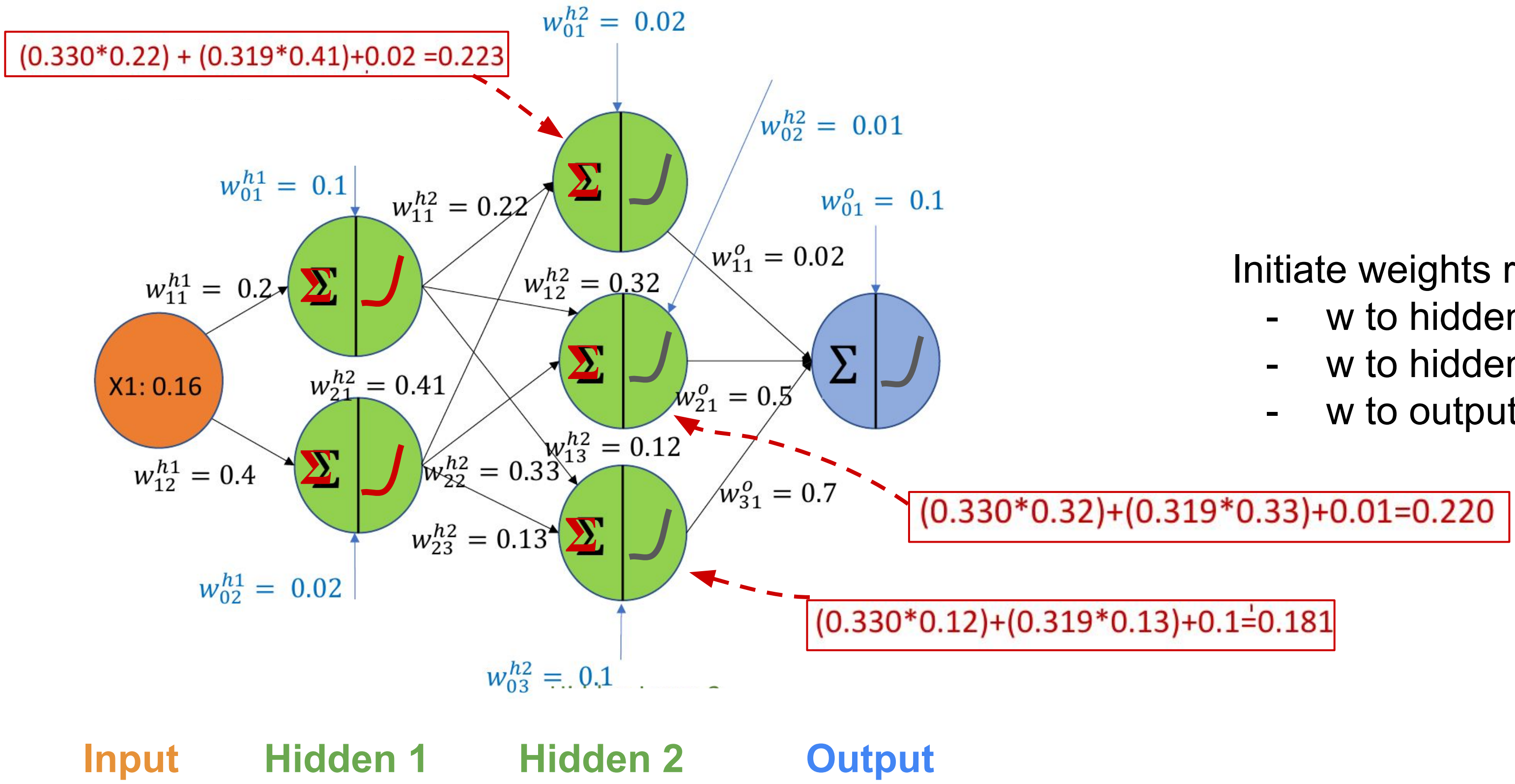




Initiate weights randomly
- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Input Hidden 1 Hidden 2 Output

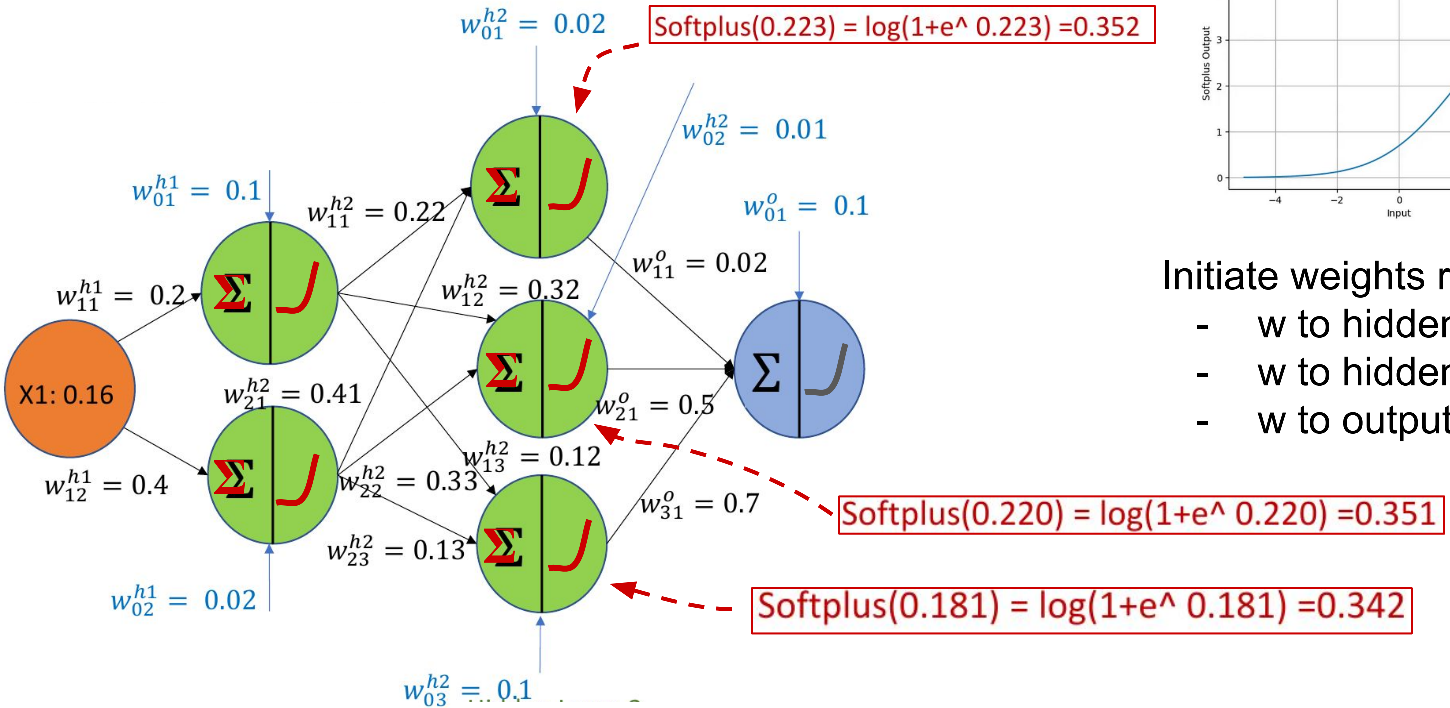
Are we going for a hike?



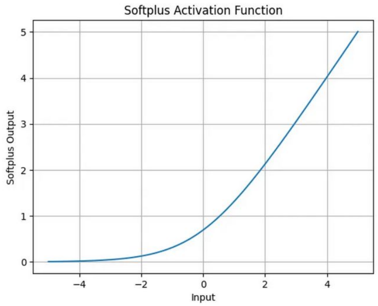
Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Are we going for a hike?



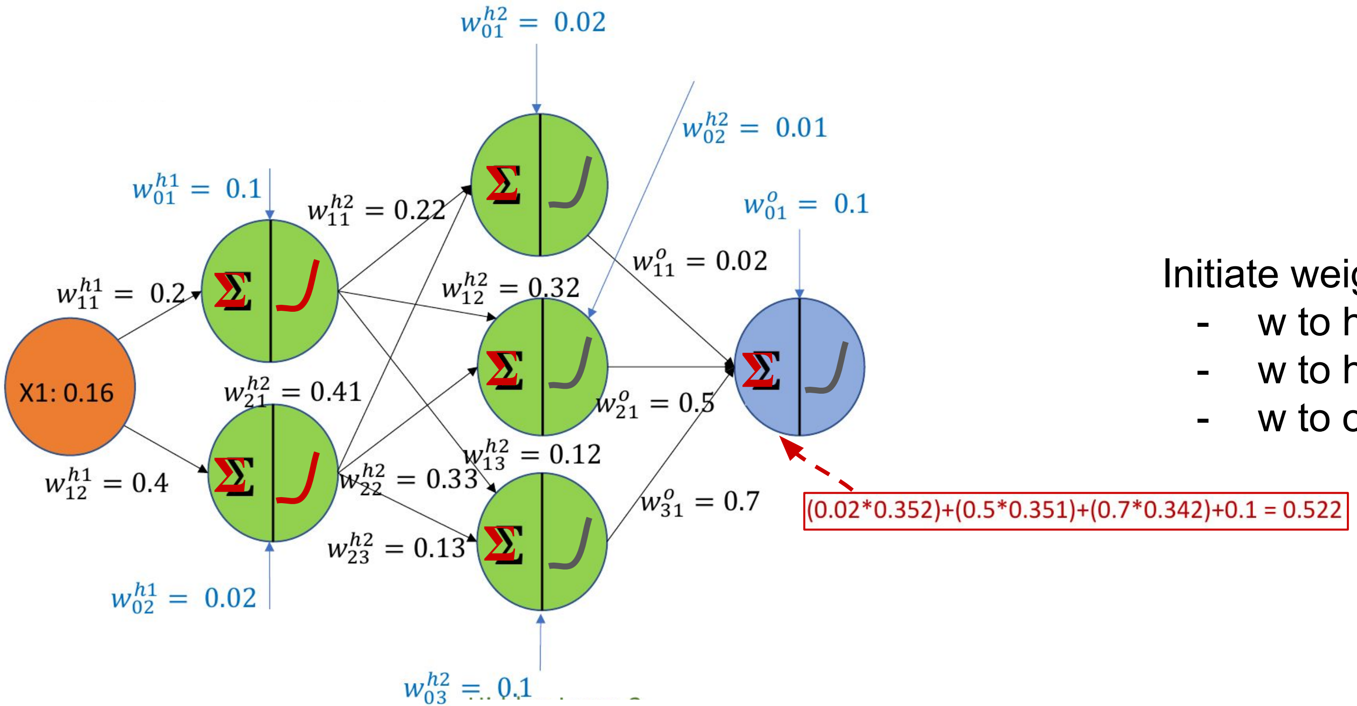
Input Hidden 1 Hidden 2 Output



Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Are we going for a hike?

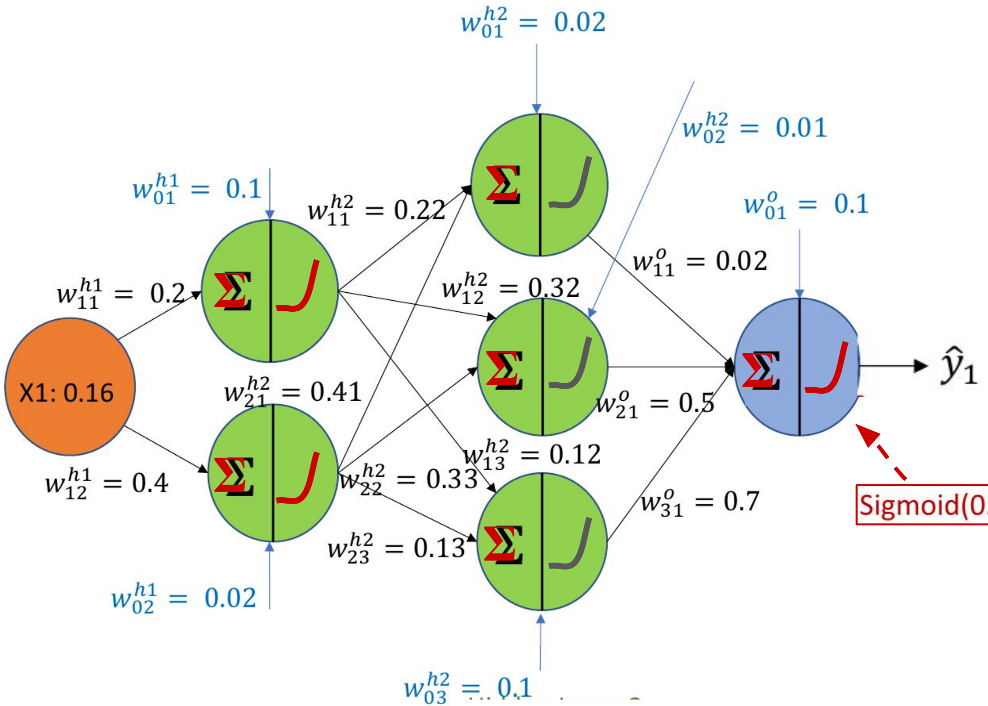


Input Hidden 1 Hidden 2 Output

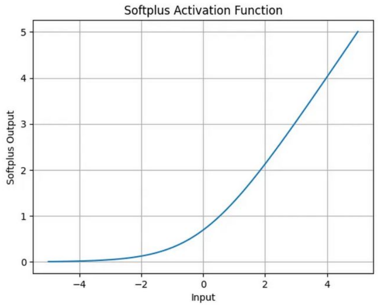
Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Are we going for a hike?



Sigmoid(0.522) = 1 / (1 + e^(- 0.522)) = 0.627

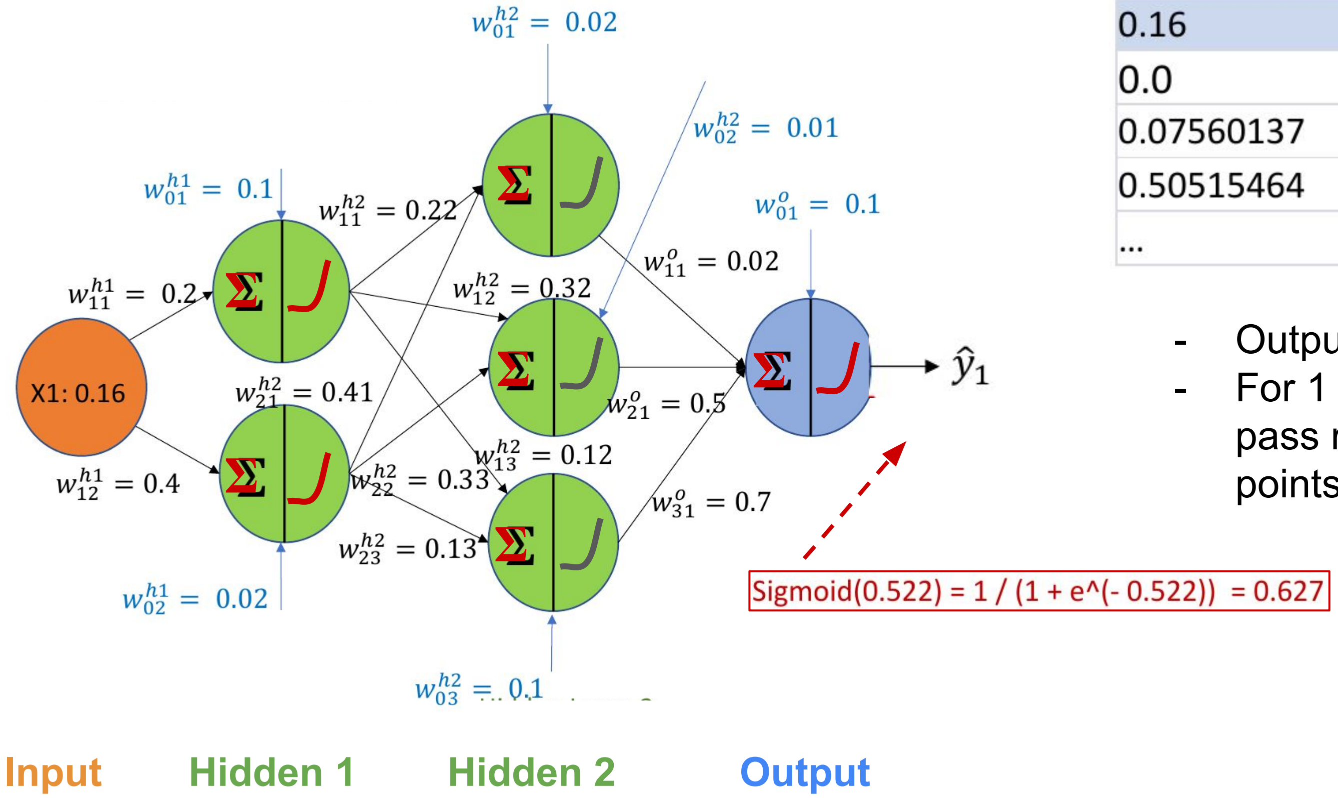


Initiate weights randomly

- w to hidden 1: h1
- w to hidden 2: h2
- w to output: o

Input Hidden 1 Hidden 2 Output

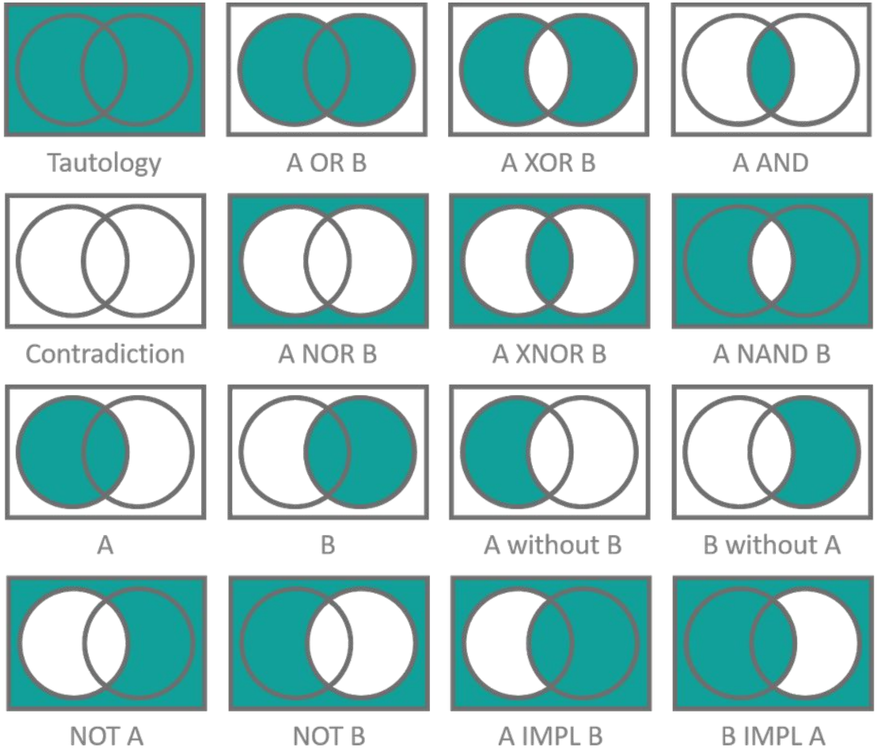
Are we going for a hike?



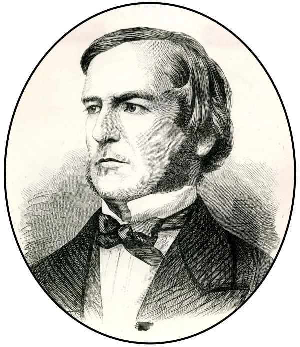
Temperature	Hike
0.16	0
0.0	0
0.07560137	0
0.50515464	1
...	...

- Output is a probability
- For 1 Feed-Forward pass repeat for all data points

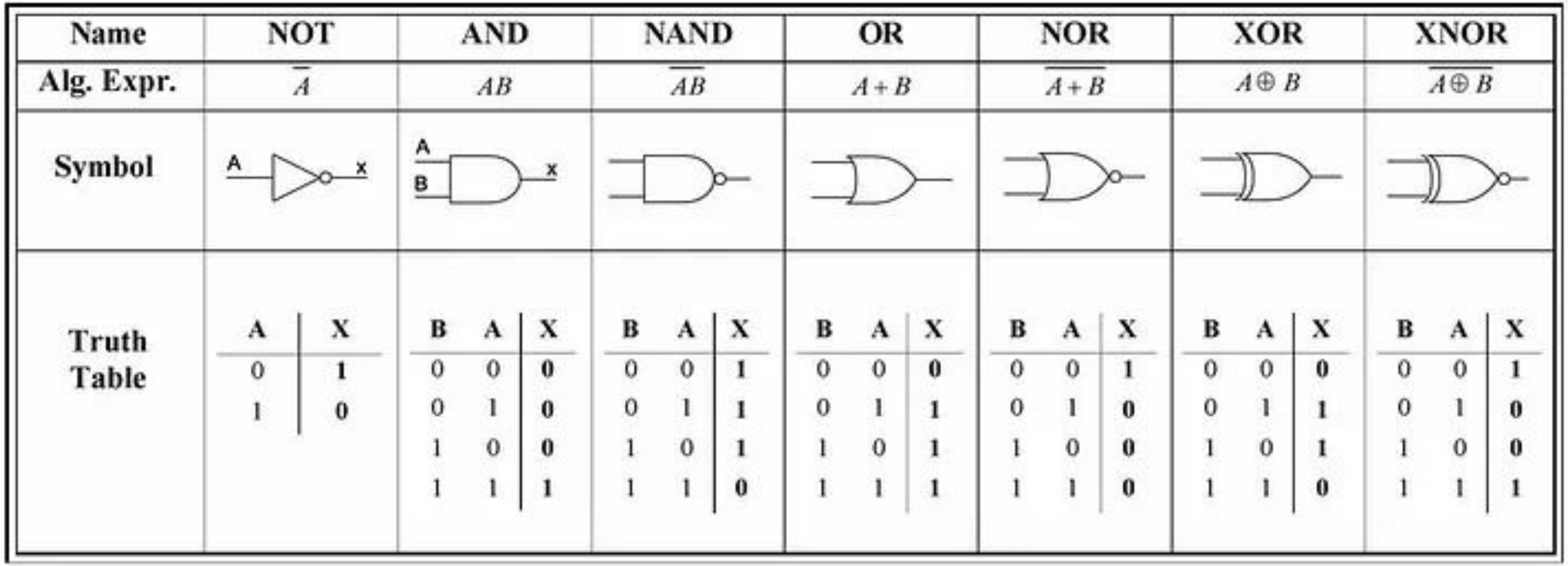
Boolean/logical venn diagrams



peakd.com


George Boole
(1800s UK)

Logical gates are also boolean functions

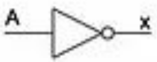


Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol	A x	A B x					
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1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

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

Logical gates are also boolean functions

Perceptron   Perceptron

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

Multilayer Perceptron

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

Logical gate for the AND

Perceptron

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

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

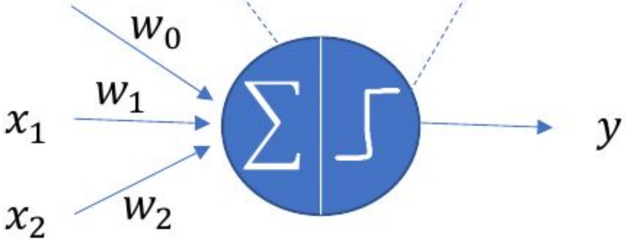
Neural Network as logical ‘AND’ gate

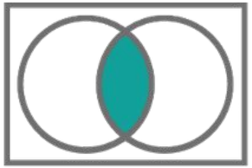
- A Perceptron is sufficient
- Goal: Find values for the weights such that the network acts like the AND gate as given by the ‘truth table’

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

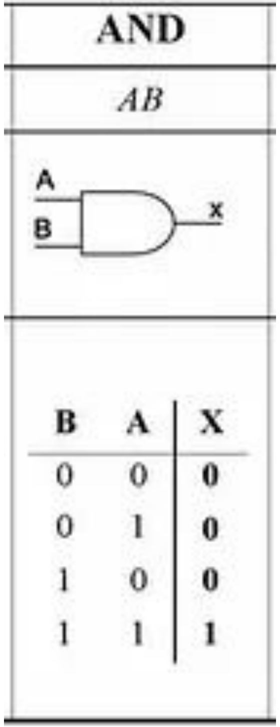
Activation function

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





A AND

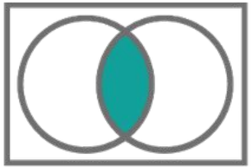


AND

AB

B	A	X
0	0	0
0	1	0
1	0	0
1	1	1

Neural Network as logical ‘AND’ gate



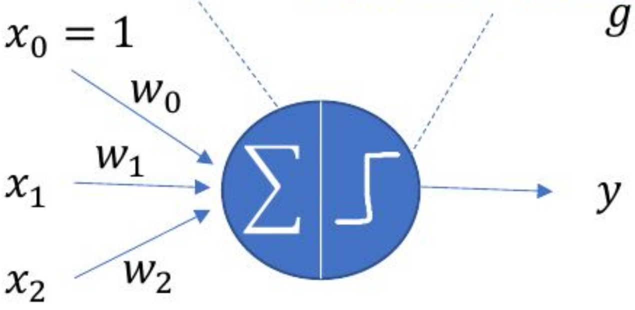
A AND

- A Perceptron is sufficient
- Goal: Find values for the weights such that the network acts like the AND gate as given by the ‘truth table’

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

Activation function

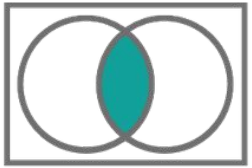
$$g(z) = \begin{cases} 0 \ ; if \ g(z) < 0.5 \\ 1 \ ; if \ g(z) \geq 0.5 \end{cases}$$



Truth table

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

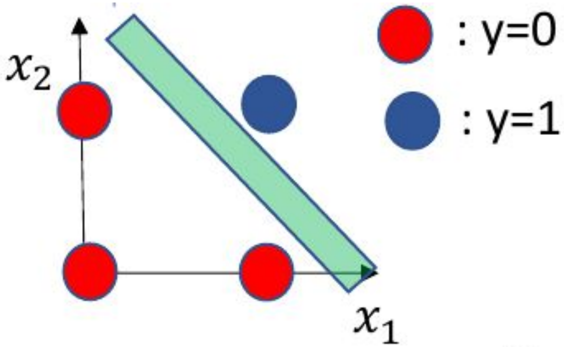
Neural Network as logical ‘AND’ gate

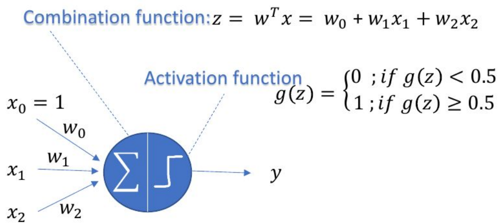

A AND

- A Perceptron is sufficient
- Goal: Find values for the weights such that the network acts like the AND gate as given by the ‘truth table’
- Visually interpreted, the decision boundary from the truth table can be seen as a straight line

Truth table

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





- NOTE: There is a linear decision boundary, thus the problem is lin separable

Neural Network as logical ‘AND’ gate

- To demonstrate, take a set of weights, w:

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

- Set up the four ‘cases’ in the truth table (recall the g(z)):

1) Case (0,0) : $w_0 + w_1x_1 + w_2x_2 = w_0 = -3$ ➔ y =0
2) Case (0,1) : $w_0 + w_2x_2 = -3 + (2)1 = -1$ ➔ y=0
3) Case (1,0) : $w_0 + w_2x_2 = -3 + (2)1 = -1$ ➔ y=0
4) Case (1,1) : $w_0 + w_1x_1 + w_2x_2 = -3 + (2)1 + (2)1 = 1$ ➔ y=1

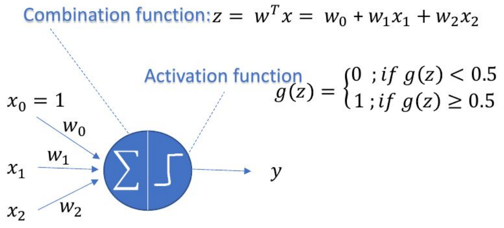
- The equation for the line separating the points is:

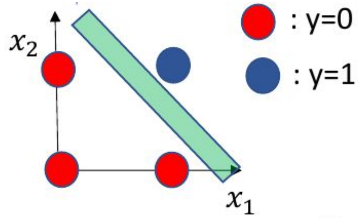
$$-3 + 2x_1 + 2x_2 = 0 \rightarrow x_1 + x_2 = 3/2$$

- You can take any values of the weights as long as the inequality is preserved

Truth table

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





Python implementation of AND gate

```python
# Implementing AND Gate

class AND_Perceptron_model:

    def __init__(self, weights, threshold):
        self.weights=weights
        self.threshold = threshold

    def combination(self, x, w):
        return sum(x_i*w_i for x_i, w_i in zip(x, w))


    def stepactivation(self, sumproduct):
        print("Threshold: " , self.threshold)
        return 1.0 if sumproduct >= self.threshold else 0.0


    def fit(self, train):
        for row in train:
            layer_1 = row[:-1] # all attributes except the last column (y)
            print("For input ")
            print(layer_1)

            sumproduct= self.combination(layer_1, self.weights)
            layer_2= self.stepactivation(sumproduct )
            print("the sum is" ,sumproduct, "the output is", layer_2)

initial_w=[-3,2,2]
dataset = [[1,0,0,0],[1,0,1,0],[1,1,0,0],[1,1,1,1]]

model = AND_Perceptron_model (initial_w,0.5)
model.fit(dataset)

```

Logical gate for the NOR

Perceptron

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

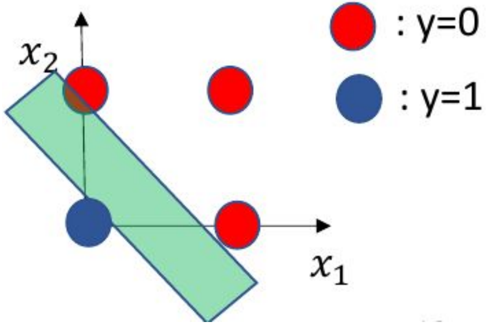
https://medium.com/autonomous-agents/how-to-teach-logic-to-your-neuralnetworks-116215c71a49

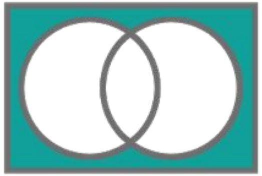
Neural Network as logical 'NOR' gate

- The implementation of the NOR gate is similar to the AND gate
- The truth table is now:

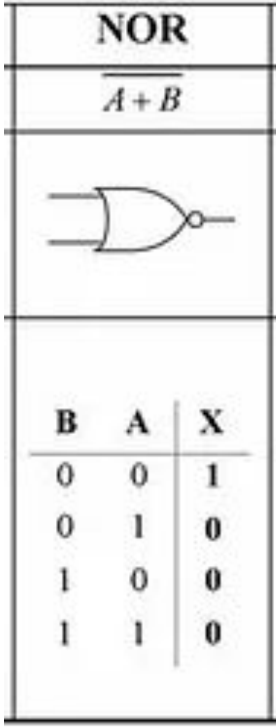
x_1	x_2	$y\colon g(x;w)$
0	0	1
0	1	0
1	0	0
1	1	0

- Which we can graphically represent as:





A NOR B



NOR

$\overline{A+B}$

B	A	X
0	0	1
0	1	0
1	0	0
1	1	0

Neural Network as logical 'NOR' gate

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

- Take the weights:

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ -4 \end{bmatrix}$$

- The four cases are now:
  1) Case (0,0) : $w_0 + w_1x_1 + w_2x_2$ = w_0 = 3 ➔ y =1
  2) Case (0,1) : $w_0 + w_2x_2$ = 3 + (-4)1 = -1 ➔ y=0
  3) Case (1,0) : $w_0 + w_2x_2$ = 3 + (-4)1 = -1 ➔ y=0
  4) Case (1,1) : $w_0 + w_1x_1 + w_2x_2$ = 3 + (-4)1 +(-4)1 = -5 ➔ y=0

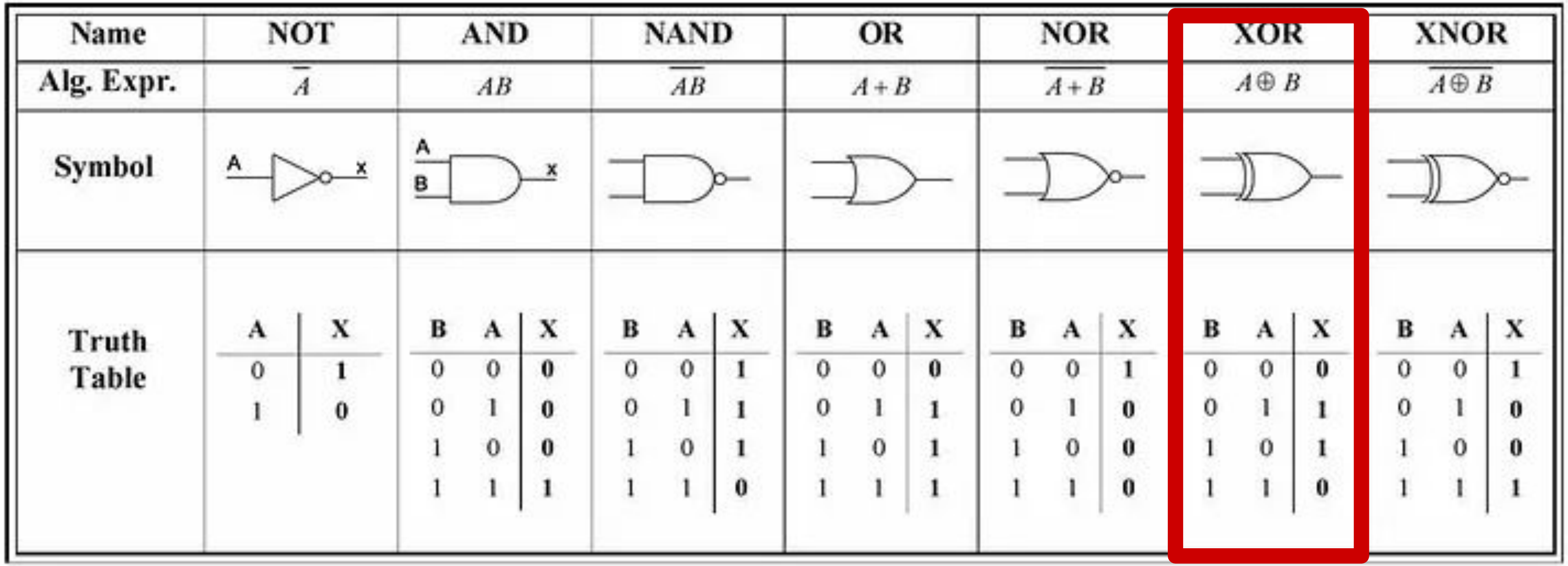
- The equation for the line separating the points is:

$$3 - 4x_1 - 4x_2 = 0 \Rightarrow x_1 + x_2 = 3/4$$

- You can take any values of the weights as long as the inequality is preserved

Logical gate for the XOR

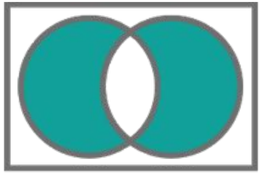
Perceptron



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	A x	A B x					
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

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

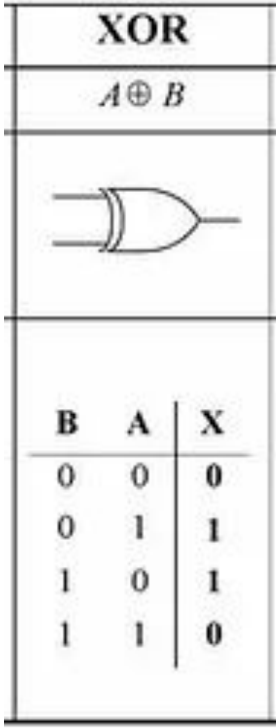
Neural Network as logical ‘XOR’ gate



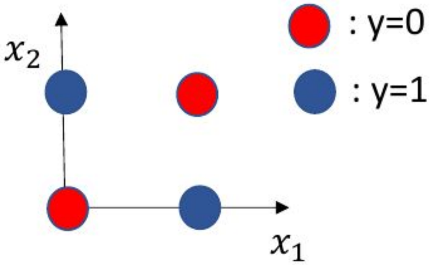
A XOR B

- The implementation of the XOR gate is different from the AND and NOR
- The truth table is now:

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



- The graphical representation is:



- There is no straight line determine to separate the points

Neural Network as logical ‘XOR’ gate: Problem!

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

- Work with weights:

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

- Set up the cases:

1) Case (0,0) : $w_0 + w_1x_1 + w_2x_2$ = w_0 = -0.5 ➔ y =0
2) Case (0,1) : $w_0 + w_2x_2$ = -0.5 + (1)1 = 0.5 ➔ y=1
3) Case (1,0) : $w_0 + w_2x_2$ = -0.5 + (1)1 = 0.5 ➔ y=1
4) Case (1,1) : $w_0 + w_1x_1 + w_2x_2$ = -0.5 + (1)1 +(1)1 = 1.5 ➔ $\boxed{\text{y=1}}$

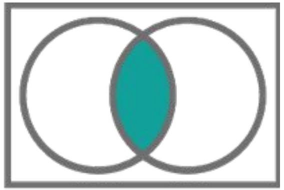
- Separability problem: We have no straight line separating the two…

Possible Solutions for Separability Problem

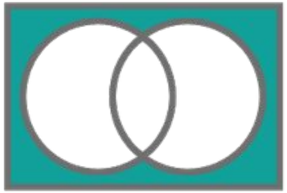
- We need to add the capacity to allow non-linear decision boundaries

1. Add a non-linear feature
    a. This is referred to as the ‘kernel trick’ (Covered with SVM)
2. Add extra layers to allow non-linearity
    a. Here we use a multilayer perceptron

The XOR gate is a combination of NOR and AND

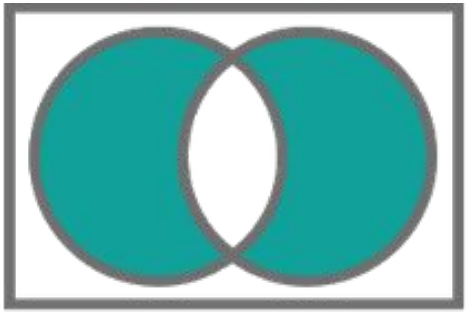


A AND



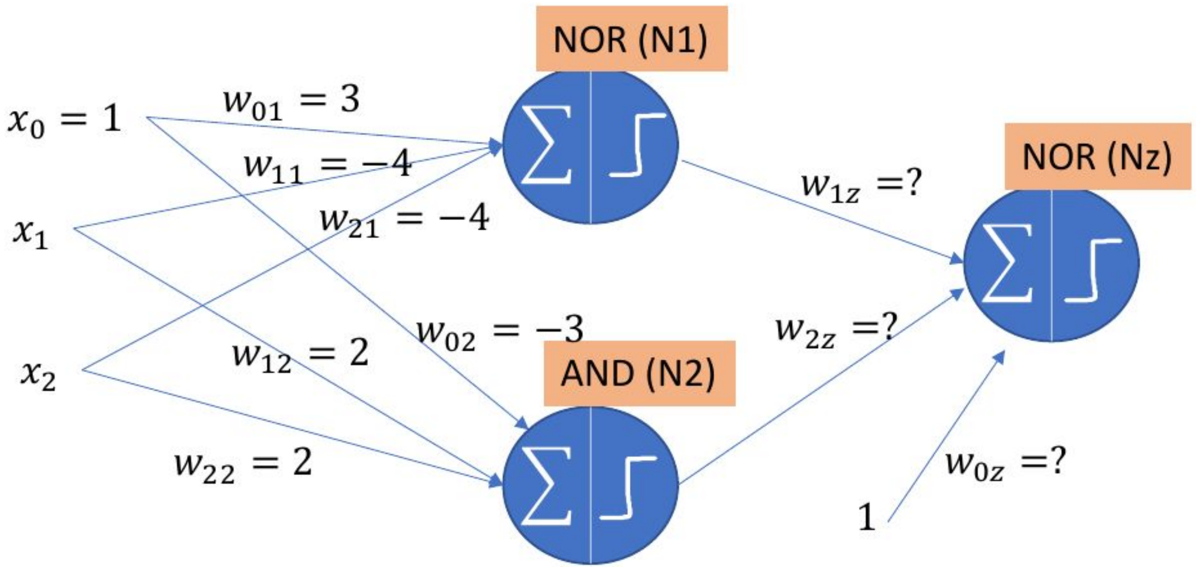
A NOR B

XOR(x1,x2) can be thought of as NOR(NOR(x1,x2),AND(x1,x2))



A XOR B

Neural Network as a Logical XOR gate: Extra Layers

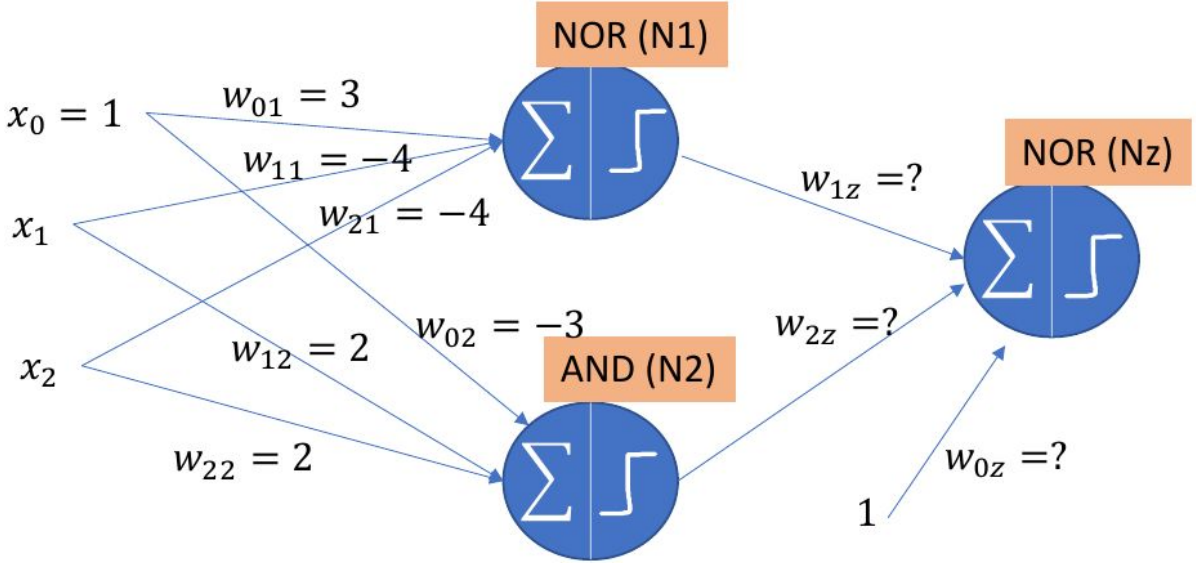


Truth Table for the network

x_1	x_2	$N1$	$N2$	Nz
0	0	?	?	0
0	1	?	?	1
1	0	?	?	1
1	1	?	?	0

We now need to determine what the weights from the NOR node (N1) and AND node (N2) to the NOR node (Nz) need to be

Neural Network as a Logical XOR gate: Extra Layers



Truth Table for the network

x_1	x_2	$N1$	$N2$	Nz
0	0	?	?	0
0	1	?	?	1
1	0	?	?	1
1	1	?	?	0

We now need to determine what the weights from the NOR node (N1) and AND node (N2) to the NOR node (Nz) need to be

Neural Network as a Logical XOR gate: 3-layer MLP

- Recall: We have determined the weights from the input to the now hidden layer to be:

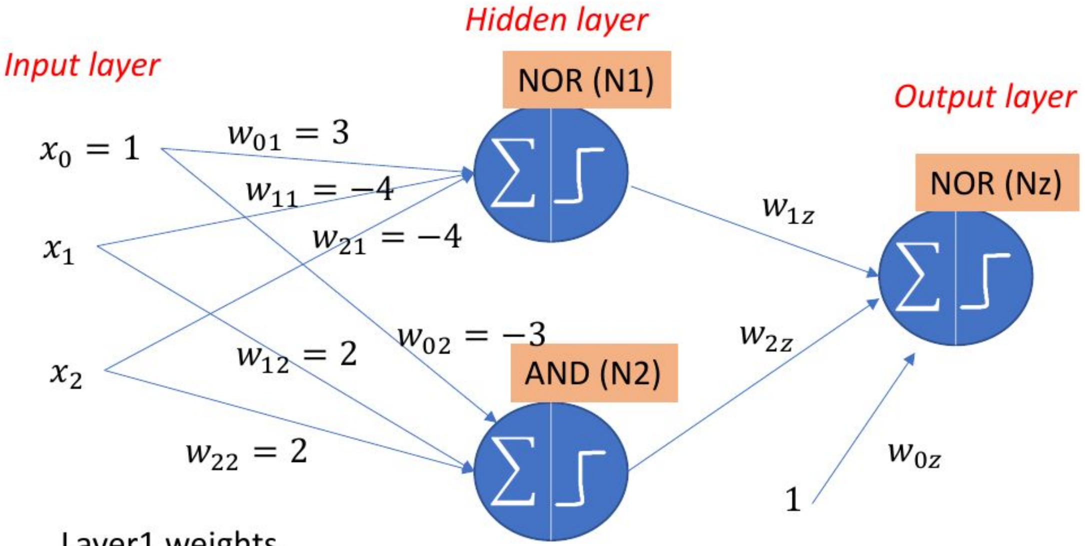
<u>Layer1 weights</u>

$w_{01} = 3$
$w_{11} = -4$
$w_{21} = -4$
$w_{02} = -3$
$w_{12} = 2$
$w_{22} = 2$

- We wish to determine the weights for the AND and NOR nodes to combine to the XOR
- We use the same activation function:

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

Neural Network as a Logical XOR gate: 3-layer MLP



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

Truth Table for the network

x_1	x_2	$N1$	$N2$	(XOR) Nz
0	0	1	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0

<u>Layer1 weights</u>

$w_{01} = 3$
$w_{11} = -4$
$w_{21} = -4$
$w_{02} = -3$
$w_{12} = 2$
$w_{22} = 2$

<u>Layer2 weights</u>

$w_{1z} = -2$
$w_{2z} = -2$
$w_{0z} = 1$

1) Case (1,0) :w_{0z} + w_{1z}= 1 + (-2) ➔ y =0
2) Case (0,0) : w_{0z} = 1 ➔ y=1
3) Case (0,0) : w_{0z} = 1 ➔ y=1
4) Case (0,1) : w_{0z} + w_{2z} = 1 + (-2) ➔ y=0

Classification metrics: How good is my classifier?

- We can use metrics of skill together with a loss function for training

Classification Metrics (Categorize data into labels)

- **Accuracy**: The proportion of total predictions that were correct.
- **Precision**: The proportion of positive identifications that were actually correct.
- **Recall (Sensitivity)**: The proportion of actual positives that were identified correctly.
- **F1 Score**: The harmonic mean of precision and recall.
- **Confusion Matrix**: A table used to describe the performance of a classification model, showing the actual vs. predicted values.
- **ROC-AUC**: The area under the receiver operating characteristic curve, measuring the trade-off between true positive rate and false positive rate.
- **Precision-Recall Curve**: Focuses on the performance with respect to the positive (minority) class.

Terminology for classification metrics

- True Positive (TP): The model predicts a positive class, and the actual class is also positive
- False Positive (FP): The model predicts a positive class, but the actual class is negative
- False Negatives (FN): The model predicts a negative class, but the actual class is positive
- True Negatives (TN): The model predicts a negative class, and the actual class is also negative

TP (True Positive)

FP (False Positive)

FN (False Negative)

TN (True Negative)

Metrics

- Accuracy (ACC): Ratio of true predictions (TP+TN) to total predictions

$$\text{ACC} = \frac{TP + TN}{TP + FP + TN + FN}$$

- Recall (R): aka ‘sensitivity’ and True Positive Rate (TPR) is the fraction of relevant instances retrieved

$$TPR = \frac{TP}{TP + FN}$$

- Specificity: or False Positive Rate (FPR)

$$FPR = \frac{FP}{FP + TN}$$

- True Negative Rate (TNR):

$$TNR = \frac{TN}{TN + FP}$$

- Precision (P): or Positive Predictive Value (PPV), the fraction of retrieved instances that are relevant

$$PPV = \frac{TP}{TP + FP}$$

- F1 score: harmonic mean of precision and recall

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

TP (True Positive)

FP (False Positive)

FN (False Negative)

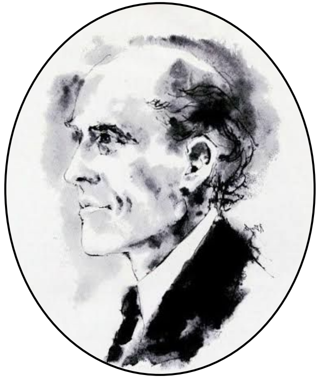
TN (True Negative)

The ‘confusion’ matrix

Contingency Table -> Confusion Matrix

Count the number of actual positives and negatives (from labels) vs the predicted positives and negatives

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)



Karl Pearson (1800s UK) established mathematical statistics

Confusion Matrix Terminology: TP

- The model predicts a positive class, and the actual class is also positive
- Here, we use an example of if a trojan is present (dog as reference)

<——Actual——>

<——Predicted——>	Has Trojan	Does Not have Trojan
Has Trojan	True Positives	
Does Not have Trojan		

NB: Trojan here is ‘dog’

TP (True Positive)

FP (False Positive)

FN (False Negative)

TN (True Negative)

Confusion Matrix Terminology: FP

- The model predicts a positive class, but the actual class is negative

Predicted \ Actual	Has Trojan	Does Not have Trojan
Has Trojan		**False Positives**
Does Not have Trojan		

NB: Trojan here is ‘dog’

TP (True Positive)

FP (False Positive)

FN (False Negative)

TN (True Negative)

Confusion Matrix Terminology: FN

- The model predicts a negative class, and the actual class is positive

		<——Actual——>	
		Has Trojan	**Does Not have Trojan**
<——Predicted——>	**Has Trojan**		
	Does Not have Trojan	**False Negatives**	

NB: Trojan here is ‘dog’

TP (True Positive)

FP (False Positive)

FN (False Negative)

TN (True Negative)

Confusion Matrix Terminology: TN

- The model predicts a negative class, and the actual class is also negative

<——Actual——>

<——Predicted——>

	Has Trojan	Does Not have Trojan
Has Trojan		
Does Not have Trojan		**True Negative**

NB: Trojan here is ‘dog’

TP (True Positive)

FP (False Positive)

FN (False Negative)

TN (True Negative)

More that two labels?

Many datasets have multiple tables, e.g. not just ‘dog’ and ‘not dog’

For malware, we can have trojan A, B and C, giving us a 3 class confusion matrix

<——Actual——>

<——Predicted——>

	Trojan A	Trojan B	Trojan C
Trojan A	12	102	93
Trojan B	112	23	77
Trojan C	83	92	17

Network Files	Browser Files	Personal Files	System Files	Type of Trojan
12	68	233	1487	Trojan A
51	85	103	1596	Trojan B
14	69	98	1482	Trojan C
11	72	1000	1502	Trojan D

Comparing two models with their confusion matrices

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

Model B		
	Actual Malware	**Actual Normal**
Predicted Malware	125	30
Predicted Normal	25	150

	Actual Positive	**Actual Negative**
Predicted Positive	**True Positive (TP)**	**False Positive (FP)**
Predicted Negative	**False Negative (FN)**	**True Negative (TN)**

Comparing models with their confusion matrices: ACC

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

Model B		
	Actual Malware	**Actual Normal**
Predicted Malware	125	30
Predicted Normal	25	150

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

$$ACC_A = \frac{TP_A + TN_A}{Total} = \frac{(100) + (175)}{330} = 0.8333$$

$$ACC_B = \frac{TP_B + TN_B}{Total} = \frac{(125) + (150)}{330} = 0.8333$$

Comparing models with their confusion matrices: Recall

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

$$TPR_A = \frac{TP_A}{TP_A + FN_A}$$
$$= \frac{(100)}{(100) + (50)} = 0.6667$$

66.67% of *actual* malware samples were *correctly identified* as malware

Note: The True Positive Rate (TPR) is the recall (R)

Comparing models with their confusion matrices: True Negative Rate (TPR)

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

$$TNR_A = \frac{TN_A}{TN_A + FP_A}$$
$$= \frac{(175)}{(175) + (5)} = 0.9722$$

97.22% of *actual* normal samples were *correctly identified* as normal

Comparing models with their confusion matrices: Recall

- TPR and TNR expressed as probabilities

Model A		
	Actual Malware	Actual Normal
Predicted Malware	100	5
Predicted Normal	50	175

Model B		
	Actual Malware	Actual Normal
Predicted Malware	125	30
Predicted Normal	25	150

$TPR_A = 0.6667$ < $TPR_B = 0.8333$

$TNR_A = 0.9722$ > $TNR_B = 0.8333$

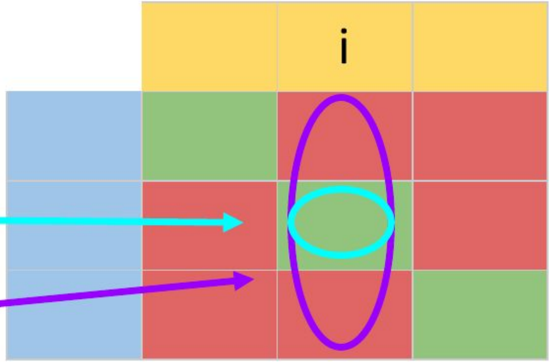
Which is *more likely* to correctly identify malware? Model B

Which is *more likely* to correctly identify normal? Model A

TPR/Recall (R) for several classes: Class i

$$TPR_i = \frac{TP_i}{TP_i + FN_i}$$

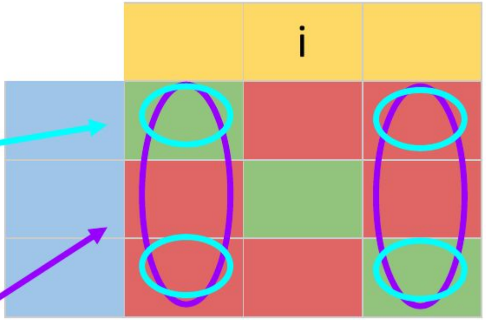
$$= \frac{M_{ii}}{\sum_{j=0}^{N} M_{ji}}$$



TPNR/Specificity for several classes: Class i

$$TNR_i = \frac{TN_i}{TN_i + FP_i}$$

$$= \frac{\sum_{j \neq i}^{N} \sum_{k \neq i}^{N} M_{jk}}{\sum_{l}^{N} \sum_{j \neq i}^{N} M_{lj}}$$



sum of all the values outside the i'th row and column, which are circled in cyan, divided by the sum of all the values outside the i'th column, which are circled in purple.

Precision (P) or Positive Predictive Value (PPV)

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

$$PPV_A = \frac{TP_A}{TP_A + FP_A}$$

$$= \frac{(100)}{(100) + (5)} = 0.9524$$

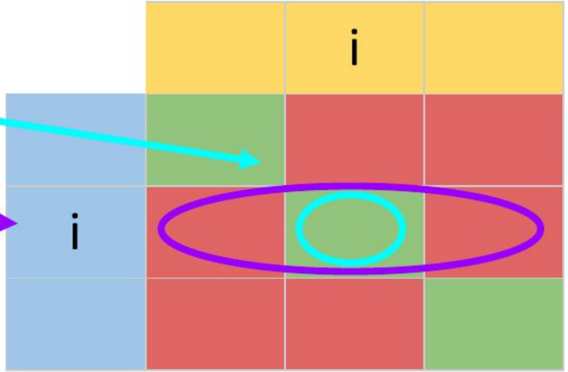
95% of the model's malware predictions are likely to be correct

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Precision (P) or Positive Predictive Value (PPV): Class i

$$PPV_i = \frac{TP_i}{TP_i + FP_i}$$

$$= \frac{M_{ii}}{\sum_j^N M_{ij}}$$



Precision is best if penalty for incorrect prediction is high

Model A		
	Actual Malware	**Actual Normal**
Predicted Malware	100	5
Predicted Normal	50	175

Model B		
	Actual Malware	**Actual Normal**
Predicted Malware	125	30
Predicted Normal	25	150

$$TPR_A = 0.6667$$
$$TNR_A = 0.9722$$
$$PPV_A = 0.9524$$

$$TPR_B = 0.8333$$
$$TNR_B = 0.8333$$
$$PPV_B = 0.8065$$

if there is a very high penalty for incorrectly predicting something as malware, then Precision might be our metric of interest, and we would want model A instead.

F1 score is preferable if we have imbalance datasets

Precision and Recall are not adequate for showing the performance of detection

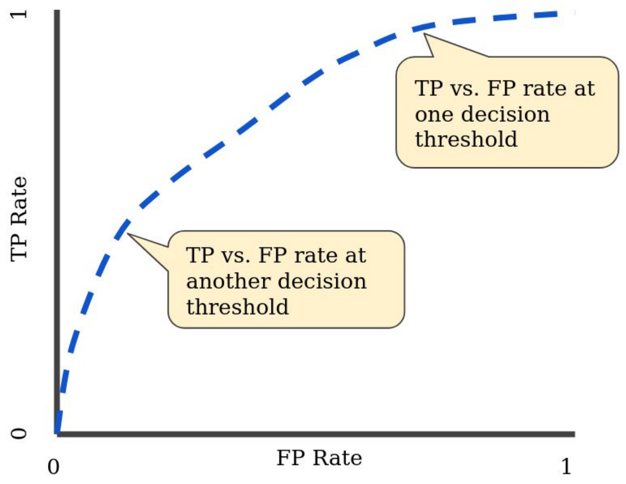
Unlike Accuracy, the F-score is resilient to imbalanced datasets

F-score is more comprehensive than Accuracy

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

Receiving Operating Characteristics (ROC) Curve

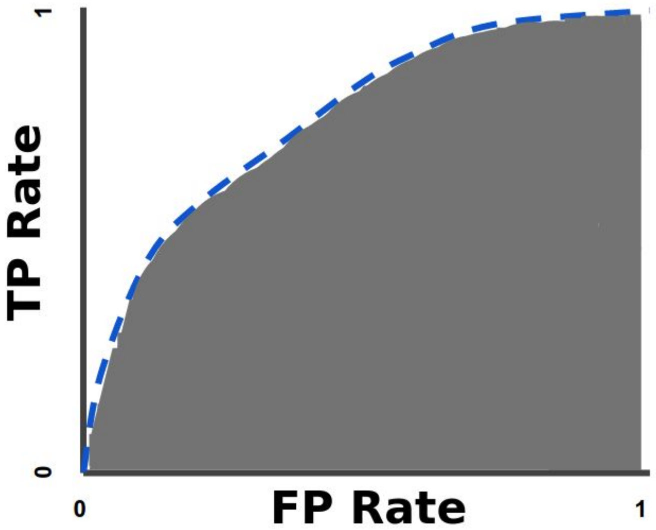
- An ROC curve plots TPR vs. FPR at different classification thresholds
- If we move the decision boundary in our model the TPR and FPR
- Lowering the classification threshold will classify more items as positive, thus increasing both False Positives and True Positives
- To compute the points in an ROC curve, we could e.g. evaluate a logistic regression model many times with different classification thresholds



developers.google.com

AUC: Area Under the ROC Curve

- The AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1)
- AUC provides an aggregate measure of performance across possible the classification thresholds
- One can interpret the AUC as the probability that the model ranks a random positive example more highly than a random negative example



developers.google.com

AUC considerations

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won’t tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.