

# COMP9444

## Neural Networks and Deep Learning

### Term 2, 2025

## Week 1 Tutorial Solutions

This page was last updated: 06/08/2025 20:39:33

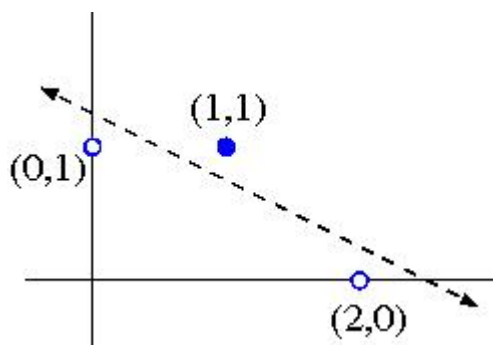
### 1. Introduce yourselves, get to know your fellow students and your tutor

### 2. Perceptron Learning

- a. Construct by hand a Perceptron which correctly classifies the following data; use your knowledge of plane geometry to choose appropriate values for the weights  $w_0$ ,  $w_1$  and  $w_2$ .

Training Example	$x_1$	$x_2$	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

The first step is to plot the data on a 2-D graph, and draw a line which separates the positive from the negative data points:



This line has slope  $-1/2$  and  $x_2$ -intercept  $5/4$ , so its equation is:

$$x_2 = 5/4 - x_1/2, \text{ i.e. } 2x_1 + 4x_2 - 5 = 0.$$

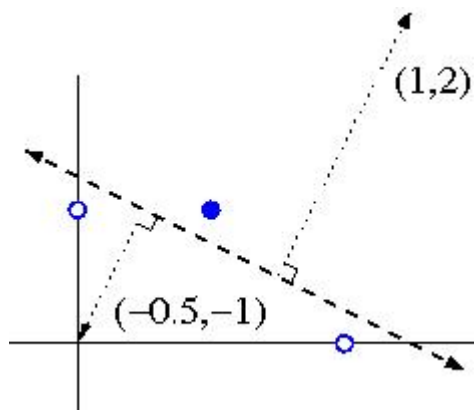
Taking account of which side is positive, this corresponds to these weights:

$$w_0 = -5$$

$$w_1 = 2$$

$$w_2 = 4$$

Alternatively, we can derive weights  $w_1=1$  and  $w_2=2$  by drawing a vector normal to the separating line, in the direction pointing towards the positive data points:



The bias weight  $w_0$  can then be found by computing the dot product of the normal vector with a perpendicular vector from the separating line to the origin. In this case  $w_0 = 1(-0.5) + 2(-1) = -2.5$

(Note: these weights differ from the previous ones by a normalizing constant, which is fine for a Perceptron)

- b. Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of

$$w_0 = -1.5$$

$$w_1 = 0$$

$$w_2 = 2$$

In your answer, you should clearly indicate the new weight values at the end of each training step. Continue the table until all items are correctly classified.

Iteration	$w_0$	$w_1$	$w_2$	Training Example	$x_1$	$x_2$	Class	$s=w_0+w_1x_1+w_2x_2$	Action
1	-1.5	0	2	a.	0	1	-	+0.5	Subtract
2	-2.5	0	1	b.	2	0	-	-2.5	None
3	-2.5	0	1	c.	1	1	+	-1.5	Add
4	-1.5	1	2	a.	0	1	-	+0.5	Subtract
5	-2.5	1	1	b.	2	0	-	-0.5	None
6	-2.5	1	1	c.	1	1	+	-0.5	Add
7	-1.5	2	2	a.	0	1	-	+0.5	Subtract
8	-2.5	2	1	b.	2	0	-	+1.5	Subtract
9	-3.5	0	1	c.	1	1	+	-2.5	Add

10	-2.5	1	2	a.	0	1	-	-0.5	None
11	-2.5	1	2	b.	2	0	-	-0.5	None
12	-2.5	1	2	c.	1	1	+	+0.5	None

### 3. Computing any Logical Function with a 2-layer Network

Recall that any logical function can be converted into **Conjunctive Normal Form** (CNF), which means a conjunction of terms where each term is a disjunction of (possibly negated) literals. This is an example of an expression in CNF:

$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$$

Assuming False=0 and True=1, explain how each of the following could be constructed. You should include the bias for each node, as well as the values of all the weights (input-to-output or input-to-hidden and hidden-to-output, as appropriate).

- a. Perceptron to compute the OR function of  $m$  inputs:

Set the bias weight to  $-1/2$ , all other weights to 1.

It makes sense for the input-to-output weights to be 1, because any of the inputs being True makes it more likely for the output to be True. In fact, the ONLY way the output can be False is if ALL the inputs are False. By setting the bias to  $-1/2$ , we insure that the linear combination is slightly negative when all of the inputs are False, but becomes positive when any of the inputs is True.

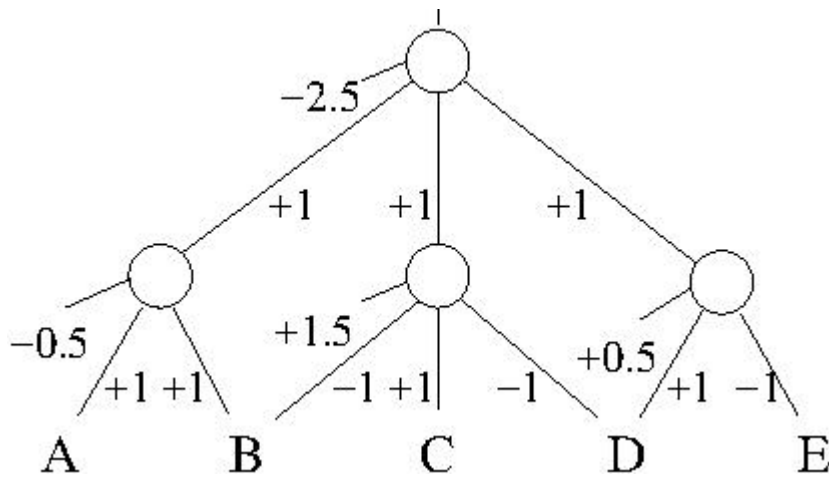
- b. Perceptron to compute the AND function of  $n$  inputs:

Set the bias weight to  $(1/2 - n)$ , all other weights to 1.

The ONLY way the conjunction can be True is if ALL the inputs are True. By setting the bias to  $(1/2 - n)$ , we insure that the linear combination is slightly positive when all of the inputs are True, but becomes negative when any of the inputs is False.

- c. Two-layer Neural Network to compute the function

$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E):$$



Each hidden node should compute one disjunctive term in the expression. The input-to-hidden weights are -1 for items that are negated, +1 for the others. The output node then computes the conjunction of all the hidden nodes, as in part (b).

---

With reference to this example, explain how a two-layer neural network could be constructed to compute any (given) logical expression, assuming it is written in Conjunctive Normal Form.

---

As in the example above, each hidden node should compute one disjunctive term in the expression; the output node then computes the conjunction of all these hidden nodes. The input-to-hidden weights should be -1 for items that are negated, +1 for the others. The bias for each hidden node should be  $(k - \frac{1}{2})$  where  $k$  is the number of items that are negated in the disjunctive term corresponding to that node.

For each disjunctive term, the ONLY situation where it would be False is when the input is 1 for all the negated items, and 0 for all the non-negated items. By setting the bias to  $(k - \frac{1}{2})$ , we ensure that the linear combination is slightly negative in this situation, but becomes positive with any change to the input values.

---

#### 4. XOR Network

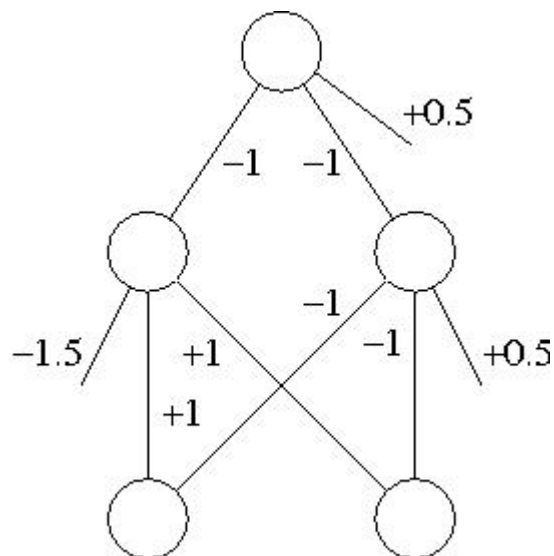
Construct by hand a Neural Network (or Multi-Layer Perceptron) that computes the XOR function of two inputs. Make sure the connections, weights and biases of your network are clearly visible.

---

There are a number of ways to express XOR as a combination of simpler functions that are linearly separable. For example, using NOR as an abbreviation for "NOT OR",  $(x_1 \text{ XOR } x_2)$  can be written as:

$$(x_1 \text{ AND } x_2) \text{ NOR } (x_1 \text{ NOR } x_2)$$

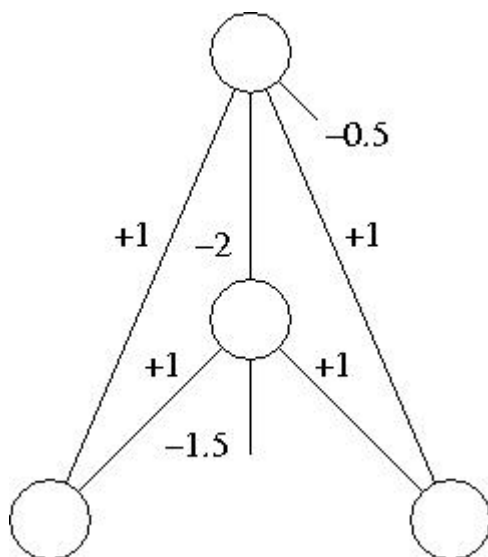
This decomposition allows us to compute XOR with a network like this:



Challenge: Can you construct a Neural Network to compute XOR which has only one hidden unit, but also includes shortcut connections from the two inputs directly to the (one) output.

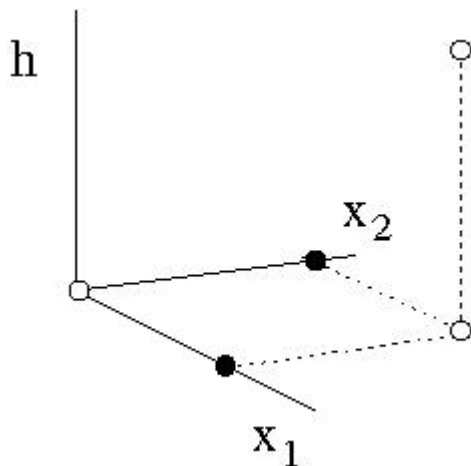
Hint: start with a network that computes the inclusive OR, and then try to think of how it could be modified.

Exclusive OR (XOR) is very similar to normal (inclusive) OR, except for the case where both inputs are True, i.e. where  $(x_1 \text{ AND } x_2)$  is True. We therefore introduce a single hidden unit which computes  $(x_1 \text{ AND } x_2)$ . This hidden unit is connected to the output with a negative weight, thus forcing the overall output to be False when the output from this hidden node is positive.



The addition of this hidden "feature" creates a 3-dimensional space in which the points can be linearly separated by a plane. The weights for the output

unit  $(+1, +1, -2)$  specify a vector perpendicular to the separating plane, and its distance from the origin is determined by the output bias divided by the length of this vector.




---

## 5. Checkpoint #1: Python Refresher and Tensor Basics

Checkpoints are designed to make sure you are keeping up with weekly tasks and consistently taking actions to learn the technical material, and the skills required for the successful completion of the assignment and the group project. Each checkpoint is worth 1% of your final grade. You will need to show your work to your tutor during the tutorial or mentoring session to get the weekly checkpoint mark.

In the last 45 minutes of the Week 1 Tutorial, you will work on refreshing Python and learning basics of NumPy Arrays and PyTorch Tensors, by following these links:

- [Python Refresher](#)
- [Python Exercises](#)
- [Numpy Refresher](#)
- [Getting Started with Matplotlib](#)
- [PyTorch Tensors](#)
- [Getting Started with Google Colab](#)

## 6. Implications of Deep Learning

What potential benefits and dangers might Deep Learning pose for education, entertainment, the economy, and society in general?

---