Perceptrons, Part 3 – Limitations and XOR

Eckel, TJHSST AI2, Spring 2021

Explorations: What Can't Perceptrons Model?

Ok, so you've got perceptron training working. The next question is: which functions can't be modeled?

You may have noticed that the only two 2-bit functions that can't be modeled are #6 and #9, aka as XOR and XNOR.

To answer the question "why don't these work", let's first consider one that *does* work. We saw last week that AND can be modeled with this perceptron:

$A(t) = step(t)$ or, in other words, $A(t) = 1$ if $t > 0$ otherwise $A(t) = 0$.
$\vec{w} = \langle 1, 1 \rangle$
$b = -1.5$

I want to show you a different way of thinking about this. If we break apart the input and weight vectors and call the individual values in_1, in_2, w_1, and w_2 then we can rewrite the perceptron formula as follows:

$$A(w_1 \cdot in_1 + w_2 \cdot in_2 + b)$$

This time, the dot just means numerical multiplication. Now, when does the step function return "1"? Because we're using the step function, it returns 1 any time the expression inside the parentheses is greater than zero. So let's write this as an inequality, then – this perceptron returns "1", or "True", when the following condition is met:

$$w_1 \cdot in_1 + w_2 \cdot in_2 + b > 0$$

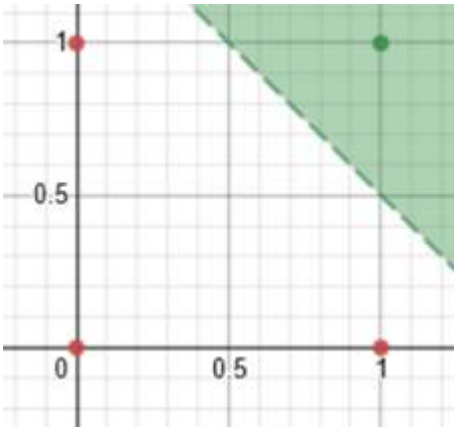
Or, to rewrite:

$$w_1 \cdot in_1 + w_2 \cdot in_2 > -b$$

This is a simple linear inequality on two variables, input 1 and input 2! For the specific AND perceptron above, it makes:

$$1 \cdot in_1 + 1 \cdot in_2 > 1.5$$

Let's visualize! Examine the following graph:

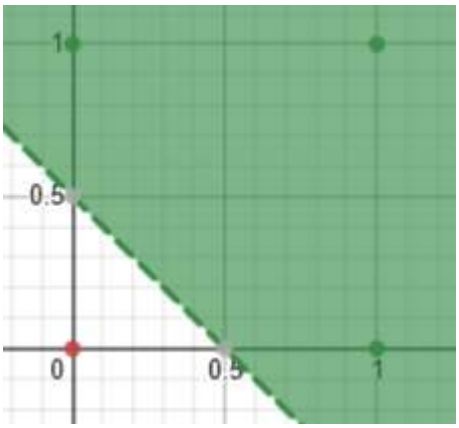


This graph contains two things:

- The inequality written above, graphed with in_1 on the x-axis and in_2 on the y-axis.
- The four input vectors to the AND Boolean function graphed as ordered pairs, with the color RED representing inputs that should be evaluated as "0" or "False", and the color GREEN representing inputs that should be evaluated as "1" or "True".

Notice that the inequality contains the green point and all the red points are outside its solution zone!

As another example, this diagram shows the four possible inputs evaluated using the OR function, and the inequality representation of a perceptron that models OR correctly:



Specifically, this is the inequality $1 \cdot in_0 + 1 \cdot in_1 > 0.5$.

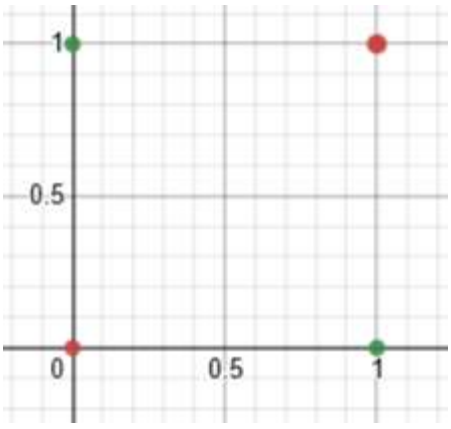
At this point, it should make sense that any perceptron with a step activation function is equivalent to a linear inequality. These diagrams are all on two variables, but the same logic applies to 3, 4, etc.

Ok…so why can't we model XOR?

This is the truth table for XOR:

In_1	In_2	Out
1	1	0
1	0	1
0	1	1
0	0	0

…and when we graph those points, this is what we get:



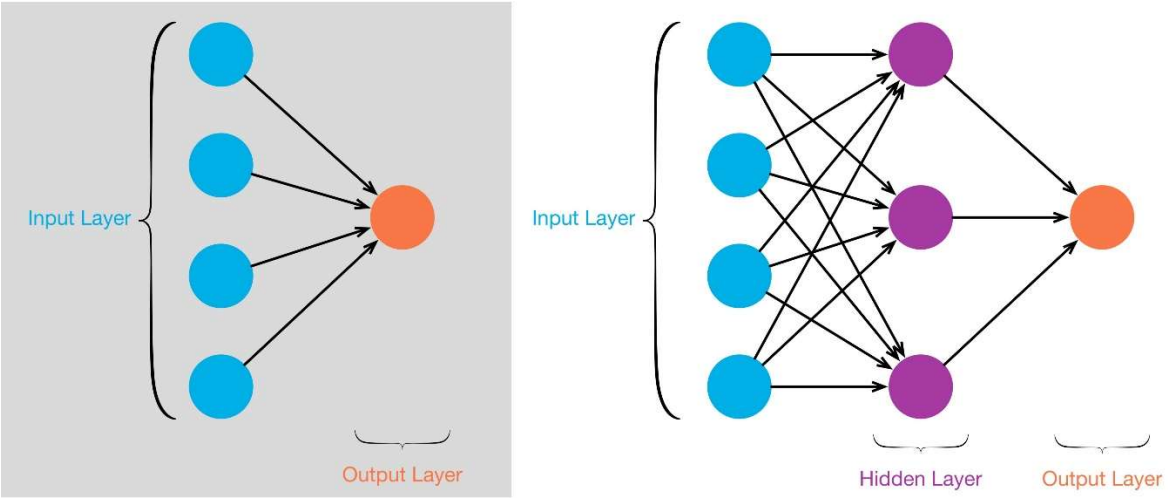
Do you see the problem? It's *impossible* to write a single linear inequality that will place the green points on one side and the red points on the other side!

This is a **CRUCIAL** concept to understanding the capabilities of a perceptron. We say that the outputs of the AND function are **linearly separable**, and the outputs of the XOR function are **not linearly separable**. (If you'd like further reading, there's a whole Wikipedia article on "linear separability", which I highly recommend!)

Ok… So… What next?

Well, if *one* perceptron won't model XOR correctly, we'll have to use more than one. Specifically, we'll have to use a **perceptron network.**

A visual aid:



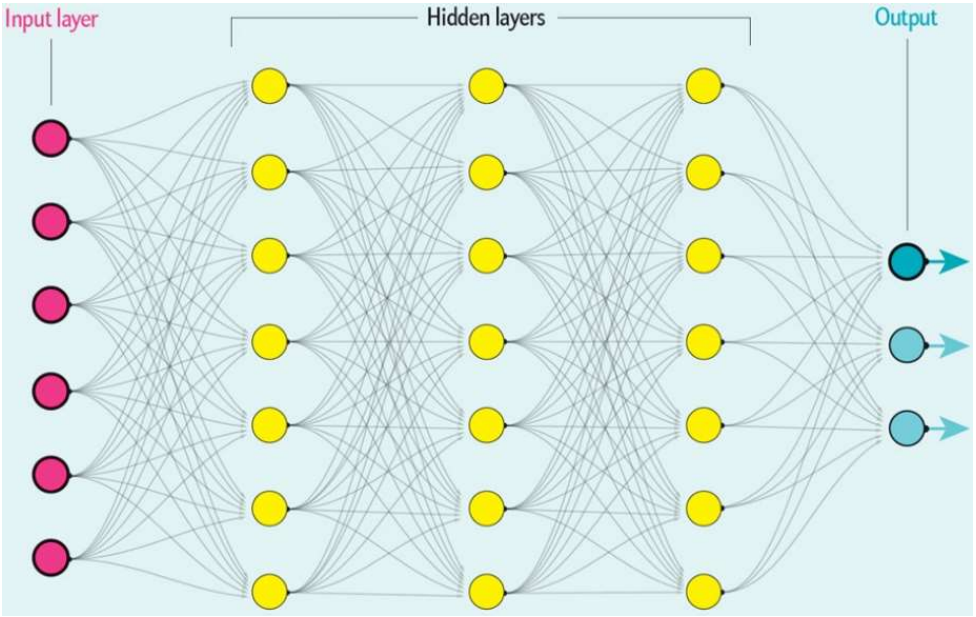
(Source: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f)

On the left, you see the kind of perceptron "networks" we've been building so far – some number of inputs go directly into a perceptron, which gives one output.

On the right, you see what we're going to make next – we have a **hidden layer** of perceptrons, and the outputs from the hidden layer feed into the final perceptron. The picture on the right is of a 4 – 3 – 1 network (4 inputs, 3 hidden perceptrons, 1 final perceptron).

Please note: **EVERY INPUT goes into EVERY HIDDEN PERCEPTRON**. This is a fundamental characteristic of perceptron networks, and it will continue onward when we have multiple hidden layers – every perceptron from each layer is connected to every perceptron in the next layer, always.

Our networks will look like this, eventually, though we don't have the notation or programming knowhow to model this efficiently yet. A brief vision of your future, then; specifically, this is a 6 – 7 – 7 – 7 – 3 perceptron network:



Required Task

Your next task is to **create a 2 – 2 – 1 perceptron network that models XOR correctly**. Specifically, that's 2 inputs (which should be expected), 2 hidden perceptrons, and 1 output perceptron.

I am NOT asking you to use the training algorithm on page 1 to accomplish this! I haven't taught you how to train a perceptron *network* yet… in fact, I haven't taught you how to do this at all! I'm asking you to figure it out in any way that makes sense to you. Feel free to work together with another student!

Ok. Before I can specify what we'll need, here, we need some better notation. Call the two inputs in_1 and in_2. Call the two hidden perceptrons $perceptron_3$ and $perceptron_4$. Call the output perceptron $perceptron_5$.

We need a weight vector and bias value for each perceptron; let's start with $perceptron_3$. The weight vector will contain specific values that correspond to the connection from in_1 to $perceptron_3$ and in_2 to $perceptron_3$. So this is the notation we'll use for those values – for $perceptron_3$, $\vec{w} = \langle w_{1,3}, w_{2,3} \rangle$. Or, in words, the weight vector of perceptron 3 is the weight from input 1 to perceptron 3 and the weight from input 2 to perceptron 3. Similarly, to distinguish perceptron 3's bias value from the other perceptron biases, we'll call it b_3.

To completely specify a 2 – 2 – 1 perceptron network, then, we need all of the following values:

- $w_{1,3}$
- $w_{2,3}$
- b_3
- $w_{1,4}$
- $w_{2,4}$
- b_4
- $w_{3,5}$
- $w_{4,5}$
- b_5

With that established, let me be specific. This is a paper challenge – either draw it and take a picture, or find a digital way to create a diagram – as well as a coding challenge. Your task comes in two pieces.

On paper, I need these things:

- A diagram / drawing of what the network looks like
- Every value above specified in a list near the diagram

In your code, I need these things:

- A function to run your XOR network that should visibly include three separate calls to the perceptron function specified in Perceptrons Part 1.
- A comment in your code that contains the phrase "XOR HAPPENS HERE" (so I can search easily) so I can find your code for the above bullet point.
- (Once again, your code can have all the values hardcoded; this is not a training challenge.)

Get Your Code & Answers Ready to Submit

You'll be submitting two files to me – a document and a .py file.

In the document, I need:

- A picture or screenshot of your 2 – 2 – 1 XOR network that includes all the specific values you used.

Your code will also be submitted:

- Your code will receive *one command-line input* – a string of a tuple containing a pair of Boolean inputs, for example "(1, 0)". Run those inputs through your XOR network and print out the result.
- Do not forget to add the comment specified on the previous page to your code! After seeing your results, I'll be opening your code to check that it's indeed a 2 – 2 – 1 network.

Specification

Submit **a single python script** and **a document file** to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code matches the specification above.