Perceptrons 4: Matrices and Non-Linearity

Eckel, TJHSST AI2, Spring 2021

Background & Explanation

We're getting closer and closer to being ready to put perceptrons into huge neural networks that can train themselves to do complicated tasks. This assignment is about adding in the last couple of bits of understanding:

- Matrix representation of perceptron calculations, and familiarity with numpy so we can do them easily.
- Continuous nonlinearity in the activation function, and its benefits.

Let's work through a few challenges.

Matrix Representation of Perceptrons: Challenge 1, Recreating XOR

Watch the video linked on the course website. Then, download the numpy orientation Python file on the course website. Play around.

Then, recreate your XOR code. Instead of storing values individually, make a single list of the weight matrices (you should have a 2x2 matrix and a 2x1 matrix). Make a single list of the bias matrices (you should have a 1x2 matrix and a 1x1 matrix). Run your network with a single call to the p_net function described in the video, passing it the step function, an input vector, and the w_list and b_list that you've hardcoded.

This should match the previous assignment's specification for what occurs if the code is passed a single command line variable, but now the comment "XOR HAPPENS HERE" should be next to a single p_net call as described.

After you've recreated your XOR code, move on to the next challenge.

Matrix Representation of Perceptrons: Challenge 2, The Diamond

Imagine the rotated square made by connecting (1,0), (0,1), (-1,0), and (0,-1) on the coordinate plane. A diamond centered at the origin. Your next task is to make a 2-4-1 network that will correctly classify any point as being inside or outside this diamond.

Some notes:

- There is an easy way to tell if your network is correct. If the coordinates of a point are (x,y), then it is inside this diamond if and only if $|x| + |y| < 1$. In this way, it's easy to generate random (x,y) pairs and test your network.
- As a hint about architecture, this network is really seeing if your point satisfies four simultaneous inequalities (that is, being on the correct side of each of the four lines around the perimeter of the diamond). And each perceptron is a single linear inequality. This should at least tell you why I picked "4" for the number of perceptrons in the hidden layer!
- So… if the hidden perceptrons are checking each inequality individually, what should the last one do?

Your code should use a matrix representation of your network. That is, the work in specifying the network, as with the previous challenge, comes in hardcoding a w_list and a b_list. Then, you should be able to call your network with a single call to the p_net function, passing the step function, an input (x,y) pair of decimal values, and the w_list and b_list you've specified.

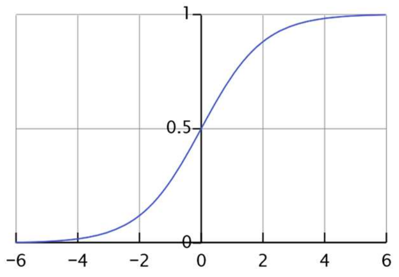
Get this so it's working perfectly, then move on to the next challenge.

Introducing Non-Linearity: Challenge 3, The Circle

Now, let's change the activation function. Instead of the step function, let's use the sigmoid function.

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

The graph of this is as follows:



I want to be completely clear here: **this will no longer output a 0 or 1 from each perceptron**. It will output something between 0 and 1, though. When it comes to perceptron network **outputs**, we can simply round to whichever is closest – 1 or 0 – and achieve a definitive output that way. But **within the network, we will not round**; that is to say, in any case where a perceptron feeds into another perceptron, **we will pass along the unrounded decimal values between 0 and 1**.

Why do we do this? Well, the sigmoid function isn't linear. Which means that it can be used… non-linearly. Now, instead of a single stark line on a graph, yes on one side and no on the other, we have a non-linear gradient, and when two non-linear gradients are overlapped their intersection may be *curved*! (It is also true that next week we'll discover that a differentiable activation function is necessary for training, but more on that later.)

Anyway, uh, check *this* out.

- Copy your previous diamond network so you can modify it for a new challenge. Instead of the diamond centered at (0,0), let us consider the *circle* centered at (0,0). Now say we want your network to correctly classify whether or not a point is inside this *circle*. (Again, easy to check if the network is correct – just see if $\sqrt{x^2 + y^2} < 1$.)
- Generate 500 random points where $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. Then, mess with the bias values on the hidden perceptrons to shift the lines to improve accuracy slightly. See how accurate you can get. You obviously can't get 100% accuracy! Four straight lines can't make a circle!
- Now, here's the fun part. Modify this network to use the *sigmoid function instead of the step function*. (If you've been doing this correctly, this should be a miniscule change – just change what function is passed in as argument A.)
- Modify the bias value in the output perceptron, and see how accurate you can get your network to be now. Without modifying any weights, just modifying biases, it should be possible to outperform the step function network by a noticeable margin – perhaps even get 100% accuracy, though 100% accuracy is not required. (You might even be able to write some code to figure out the ideal bias value for you…)

Just to be clear: **you should not modify the weight values from the previous challenge at all. You are only modifying biases.** Any of the 5 bias values is fair game, though it doesn't make sense to vary one hidden perceptron without varying the others; a symmetry argument should be pretty clear here! So you're really only playing with 2 values.

Save and <u>hardcode</u> the values of your best network.

Get Your Code Ready to Submit

There are **a lot of things I need to check** on this assignment, so please **follow these instructions carefully!**

Your code again needs to have multiple functionalities.

1. If you receive *one command-line input*, then it will be a string of a tuple containing a pair of Boolean inputs, for example "(1, 0)". In this case, run those inputs through your XOR network and print out the result. (Make sure you've commented "XOR HAPPENS HERE" again so I can see you're using matrices now!) This is how I will check **challenge 1**.
2. If you receive *two command-line inputs*, then each one will be a decimal value. Use the first as *x* and the second as *y*, and output "inside" or "outside" based on your *step-function diamond code* (not the circle challenge). This is how I will check **challenge 2**.
3. If you receive *zero command-line inputs*, then generate 500 random points as described in the circle challenge. Run them all through your hardcoded best results sigmoid circle network. Output the coordinates of any *misclassified* points, and print the percentage of the 500 that were classified correctly. This is how I will check **challenge 3**.

Specification

Submit **a single python script** 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 specifications above.