Derived neural networks

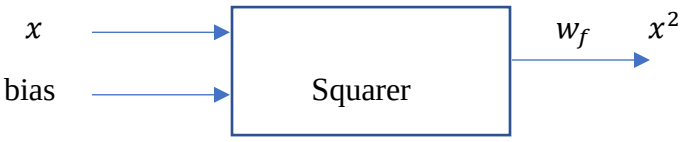
AI – Gabor – Spring 2021

Neural networks can be used as building blocks, to build or derive other neural networks for related tasks, similarly to how one uses functions as building blocks to create more complicated computer functions. Another analogy might be in assembling a model car from components – if you hook up components in just the right way, they can do even more complicated things.

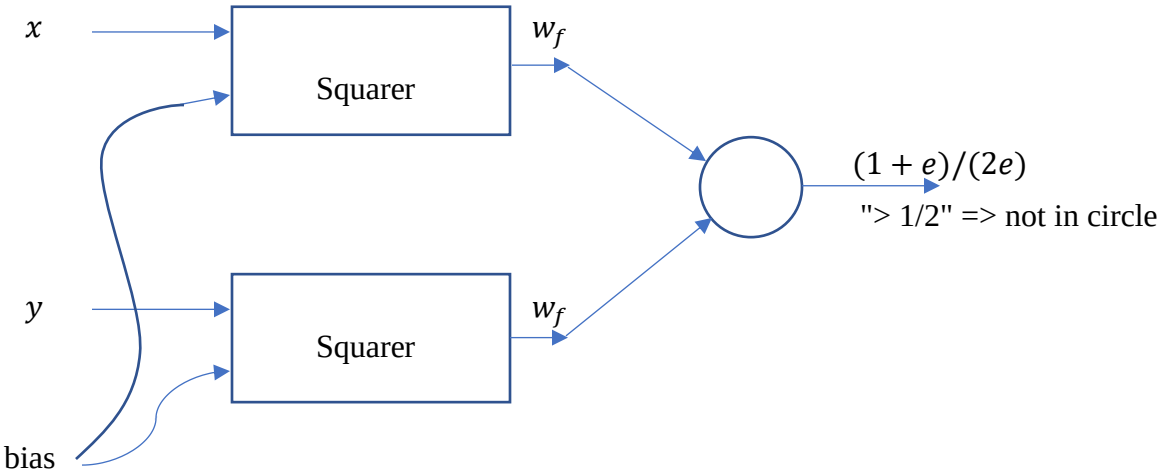
For example, if one had an NN implementing detection of whether a point (x, y) satisfies the inequality $x^2 + y^2 > 1$, then it is possible to modify the network in a straightforward way to create a derivative network that determines whether (x, y) satisfies the inequality $x^2 + y^2 > r^2$. One way to do this is by dividing each non-bias input weight by r. There is at least one other "standard" way by instead modifying weights near the output. By being careful near the output, it is also possible to modify the network to handle inequalities in the other direction, namely $x^2 + y^2 < r^2$.

In this lab, you will implement a script that takes in a squaring network and produces a network to determine whether a pt. (x, y) falls outside (or inside) a circle: $x^2 + y^2 > r^2$ ($x^2 + y^2 < r^2$). In particular, you will be given the squaring network and circle inequality and asked to come up with a derivative network.

The squaring network looks like the below (where w_f stands for "final weight of Squarer").



To check for (x, y) satisfying $x^2 + y^2 > 1$, you could do:



Whereas this is conceptually clear, there are a myriad of small details to get correct. First is the big question of how to marry the (weights of the) two squaring networks into a single network, then how to deal with an arbitrary radius of r, and then what to do if the inequality is in the other direction. We already covered one way to address the middle issue, so we'll consider here the main issue of how to marry the two squaring networks into one.

Clearly, the number of input signals goes from 2 to 3, since there is now a y supplementing the original x, but there is still just a single bias. All the other nodes (ie. what is inside the original rectangle – the hidden nodes) are replicated. That is to say, the number of cells in each hidden layer is doubled. An important thing to remember is that there is no cross-talk between the two embedded networks – the weights on the cross-talk wires are all 0s! The two squarer outputs in the 2nd diagram should be connected together to a single cell (the circle), so that there is again one output. That is, the sum $x^2 + y^2$ is passed through an activation function and then scaled.

Knowing the number of cells establishes the number of weights, so now let's take an example where the nodes are labeled by letters. Suppose the cells in our squaring network are:

	C			
x	D	I		
	E		K	x^2
B (for bias)	F	J		
	G			

That means that the architecture (layer counts) is: 2 5 2 1 1

Thus, the weights, in a weight file (with one line per layer of weights), would be listed as below. A pair of letters indicates the weight associated with the output of the node of the letter on the left to the node of the letter on the right.

```
xC BC xD BD xE BE xF BF xG BG
CI DI EI FI GI CJ DJ EJ FJ GJ
IK JK
K
```
x^2

In the combined network, the architecture (layer counts) is: 3 10 4 2 1 1

and the weights are:
xC 0 BC xD 0 BD xE 0 BE xF 0 BF xG 0 BG 0 xC BC 0 xD BD 0 xE BE 0 xF BF 0 xG BG

CI DI EI FI GI 0 0 0 0 0 CJ DJ EJ FJ GJ 0 0 0 0 0 0 0 0 0 0 CI DI EI FI GI 0 0 0 0 0 CJ DJ EJ FJ GJ

IK JK 0 0 0 0 IK JK

Kx^2 Kx^2

$\frac{1+e}{2e}$

NNBP3 - Lab specific details:

Your script receives two arguments: a file name and an inequality

The inequality has the same form as in NNBP2. The file contains the weights for a squaring network, one layer of weights per line of the file, in order. Any line in the file without numbers is considered a comment line to be ignored. The weight order within a line adheres to the conventions in NNFF and NNBP1 and NNBP2.

From the weights structure, the squaring network architecture may be derived (guaranteed two inputs, one being the bias, and one output), and hence an architecture for the circle problem is implied. The weights for the circle problem network are derived from the squaring network weights. Essentially, each weight from the squaring network appears twice in the circle's network while the remaining weights (other than the final one) are 0. These 0s are the cross-talk weights between the two embedded networks. The tricky part is putting the weights into the right places.

The output is in the same format as for NNBP2 (Layer counts followed by lines of weights). However, the grader will validate the output NN based on the underlying squaring network rather than throwing 100,000 points at the network. Your code will have 5 seconds to construct the network but should only need the smallest portion of that time. The entire test will take about 40 seconds, however, because the grader will take some time to generate a unique, though not precise, squaring network.

The score is out of 20, but the 20 items represent milestones on the way to success. This is essentially an all or nothing lab, meaning that if you almost derive the circle network, you still don't have a functioning circle network.

Example squarer weights file:

```
1.2 .3 0.4 .56 .789 1
 # This line should be ignored since there are no numbers
-.1, 0.23, 3.45, -.4567, , , .9866, and -0.321
Same with this line and the next, empty line

1.2 y -3.4
.654
```

There seven lines in the file of which four lines have numbers.
Four lines with numbers implies five numbers in the architecture.
The number of distinct weights is: 6, 6, 2, 1
The corresponding squarer's architecture is thus: 2 3 2 1 1