Scientific Computing

April 11, 2025

Announcements

→ Homework 5 due Fri Apr 18, 11:59pm

→ No class Fri Apr 18, Mon Apr 21 (Easter Break)

Today

→ Intro to Neural Networks

Office Hours:

Mon + Fri

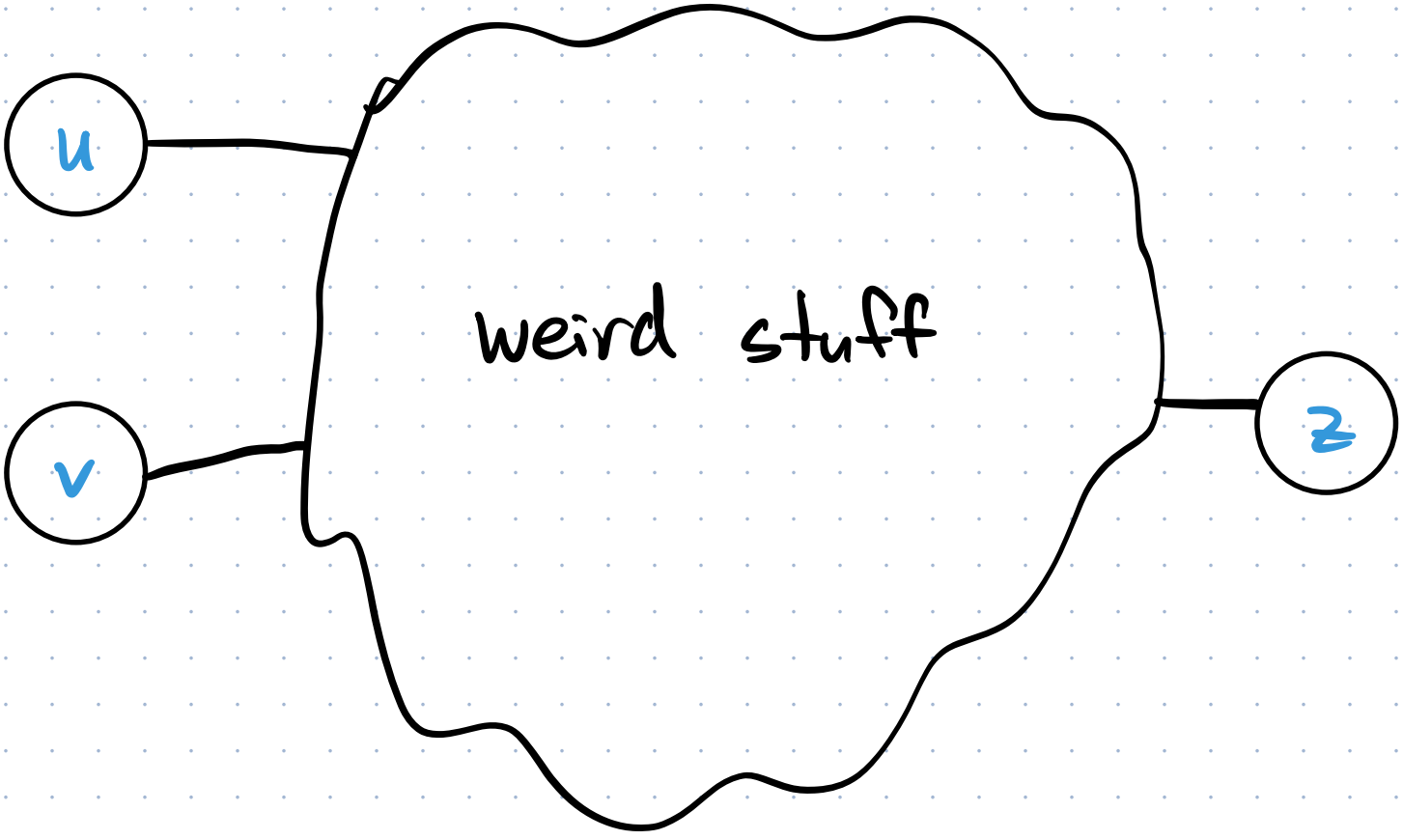
9:30am - 10:30am

Cudahy 307

What is a neural network?

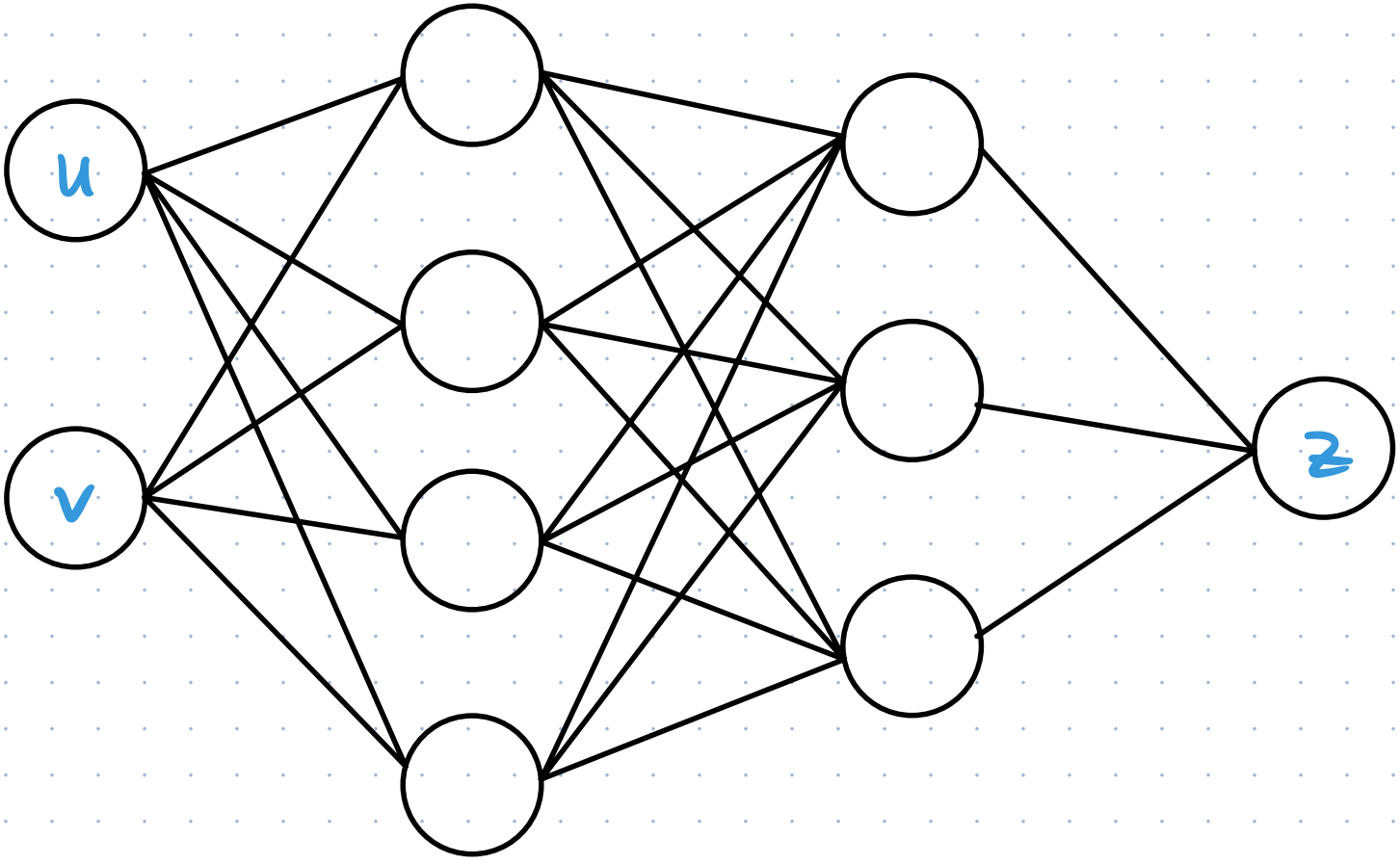
A fancy function defined in the following way.

Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



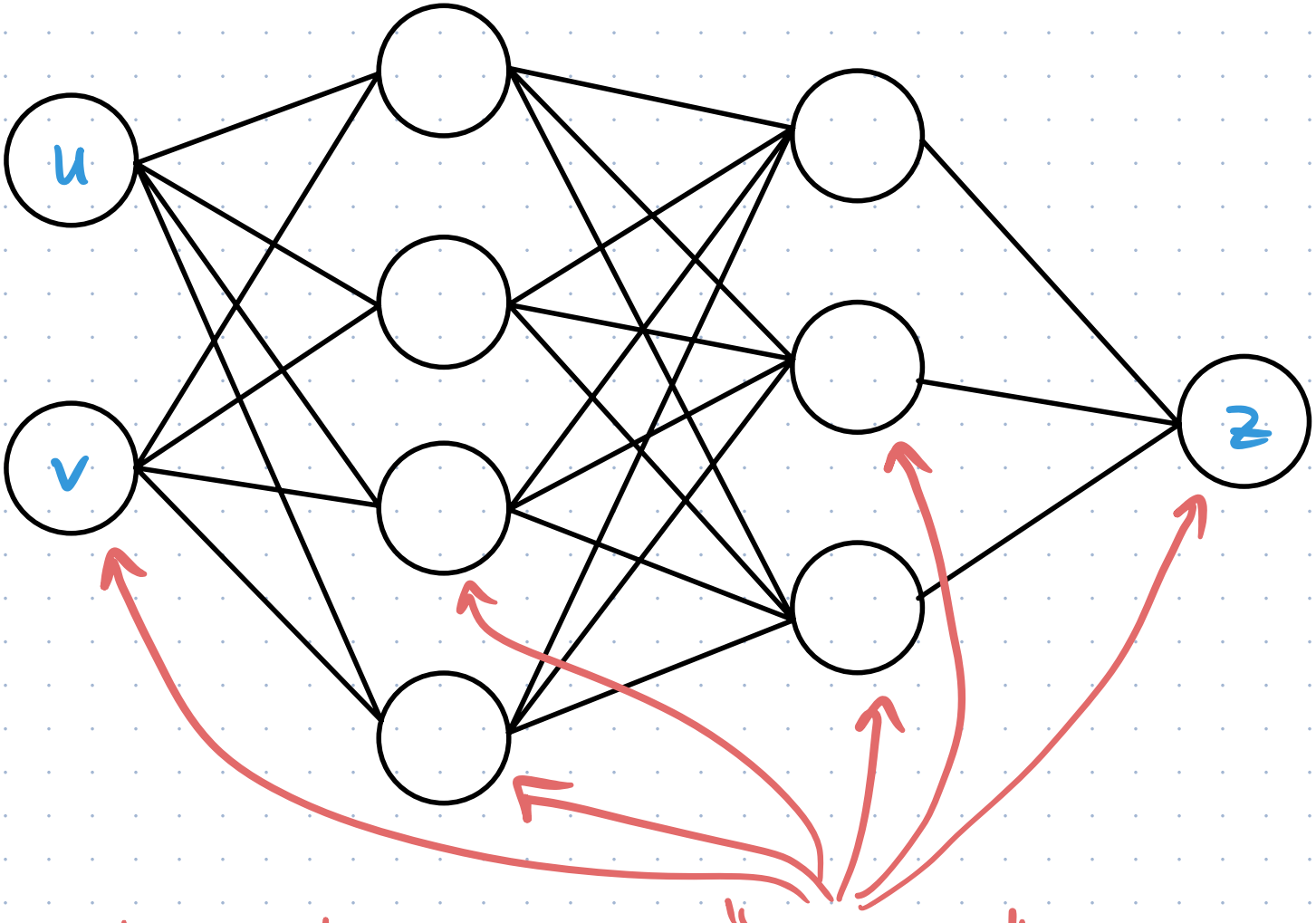
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



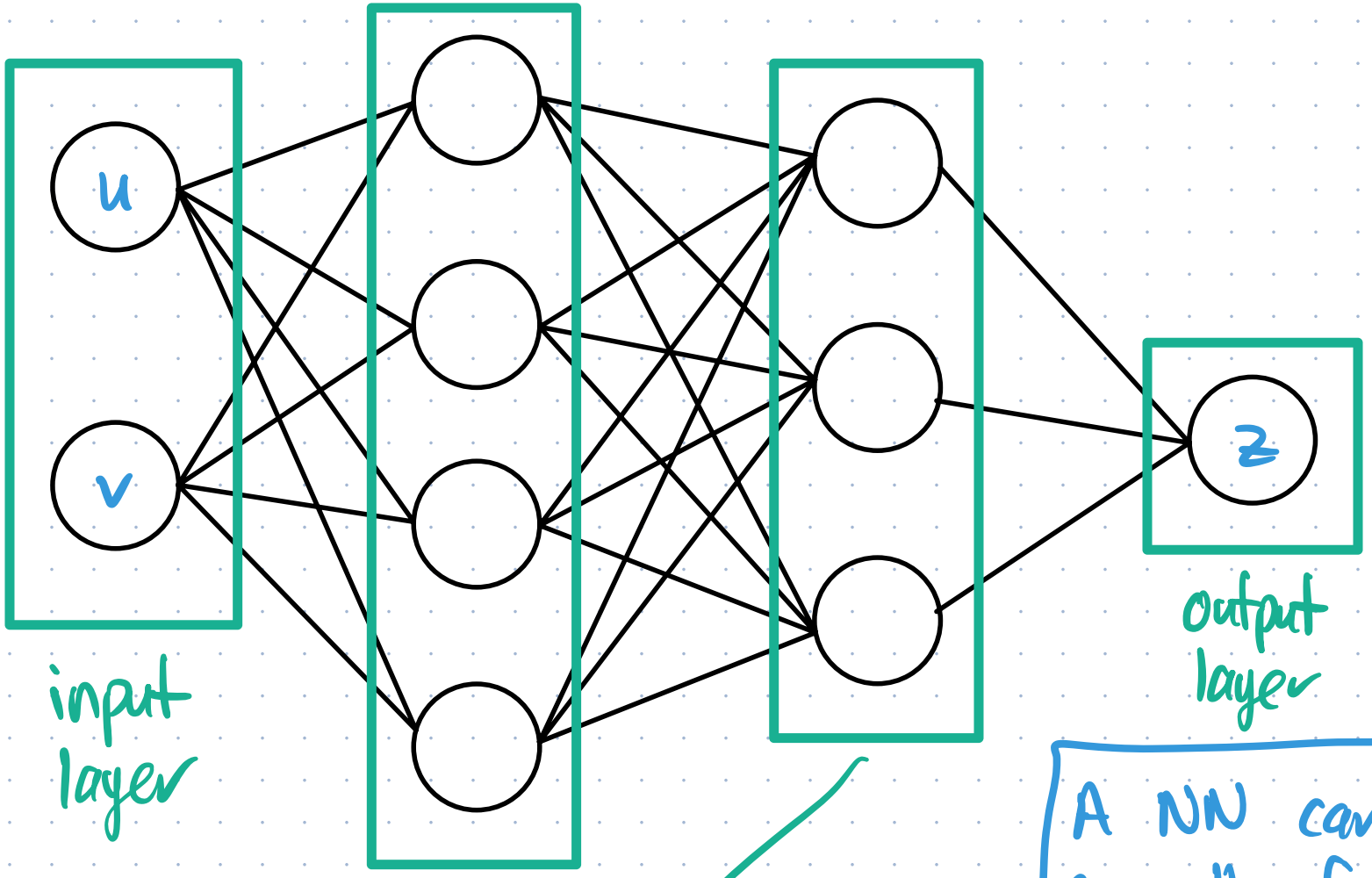
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



each circle is a "neuron"

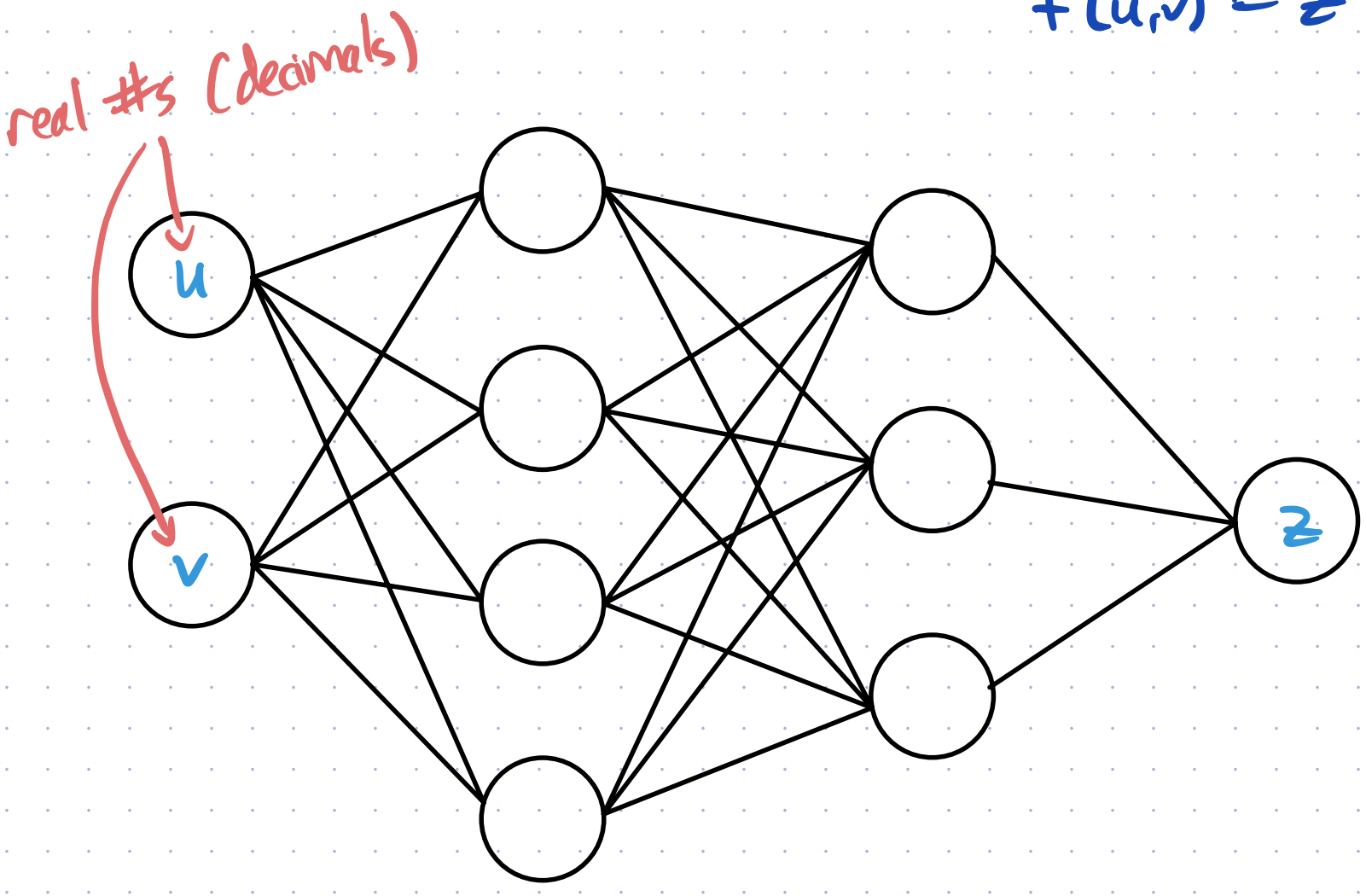
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



A NN can have any # of layers, and they can be any size.

Example: A NN that takes 2 inputs and has 1 output

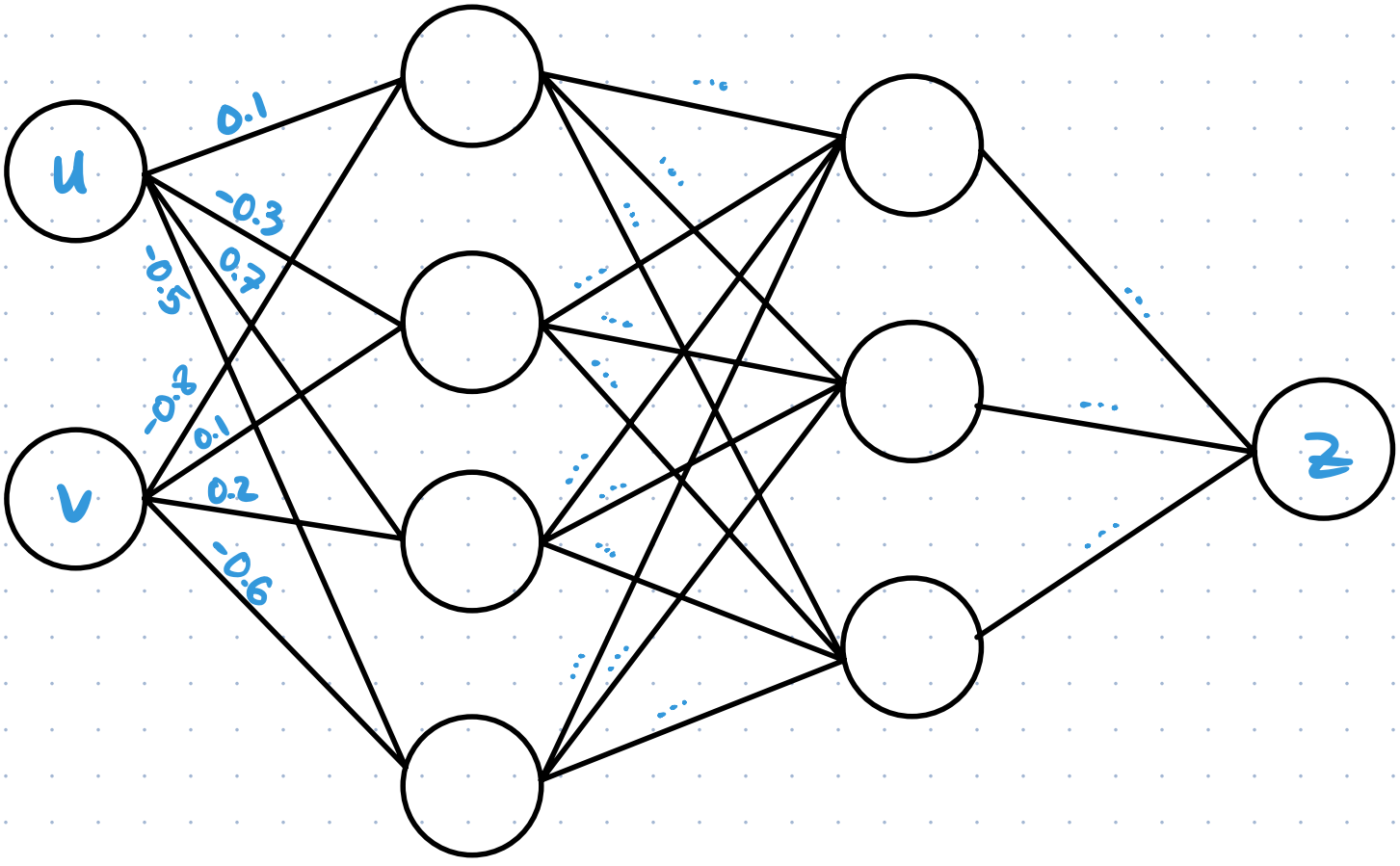
$f(u,v) = z$



Every edge between neurons has a real # attached to it called its weight

Example: A NN that takes 2 inputs and has 1 output

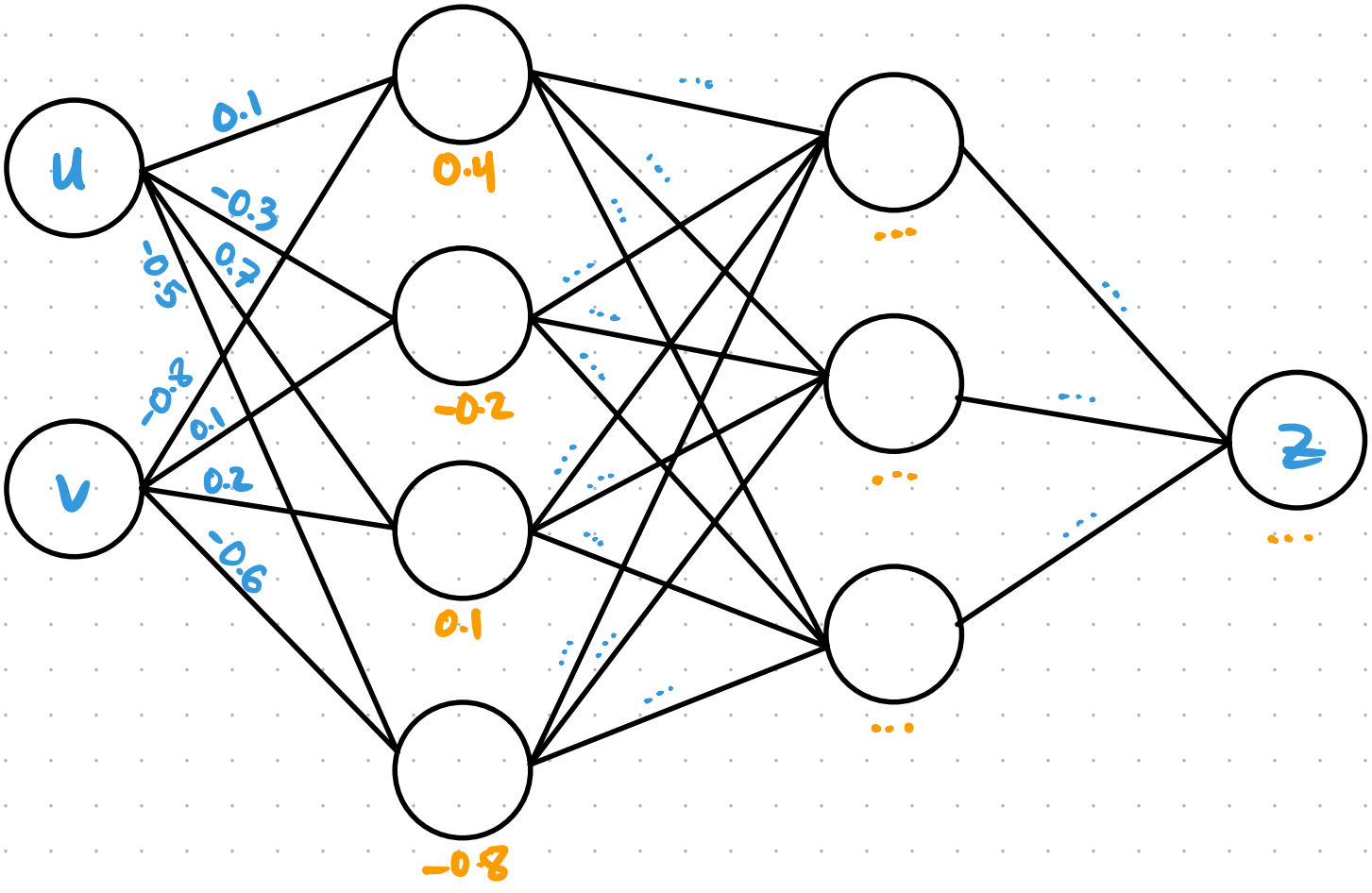
$f(u,v) = z$



I'm using #s to 1 decimal place for simplicity, but typically the are full floating point #s.

Example: A NN that takes 2 inputs and has 1 output

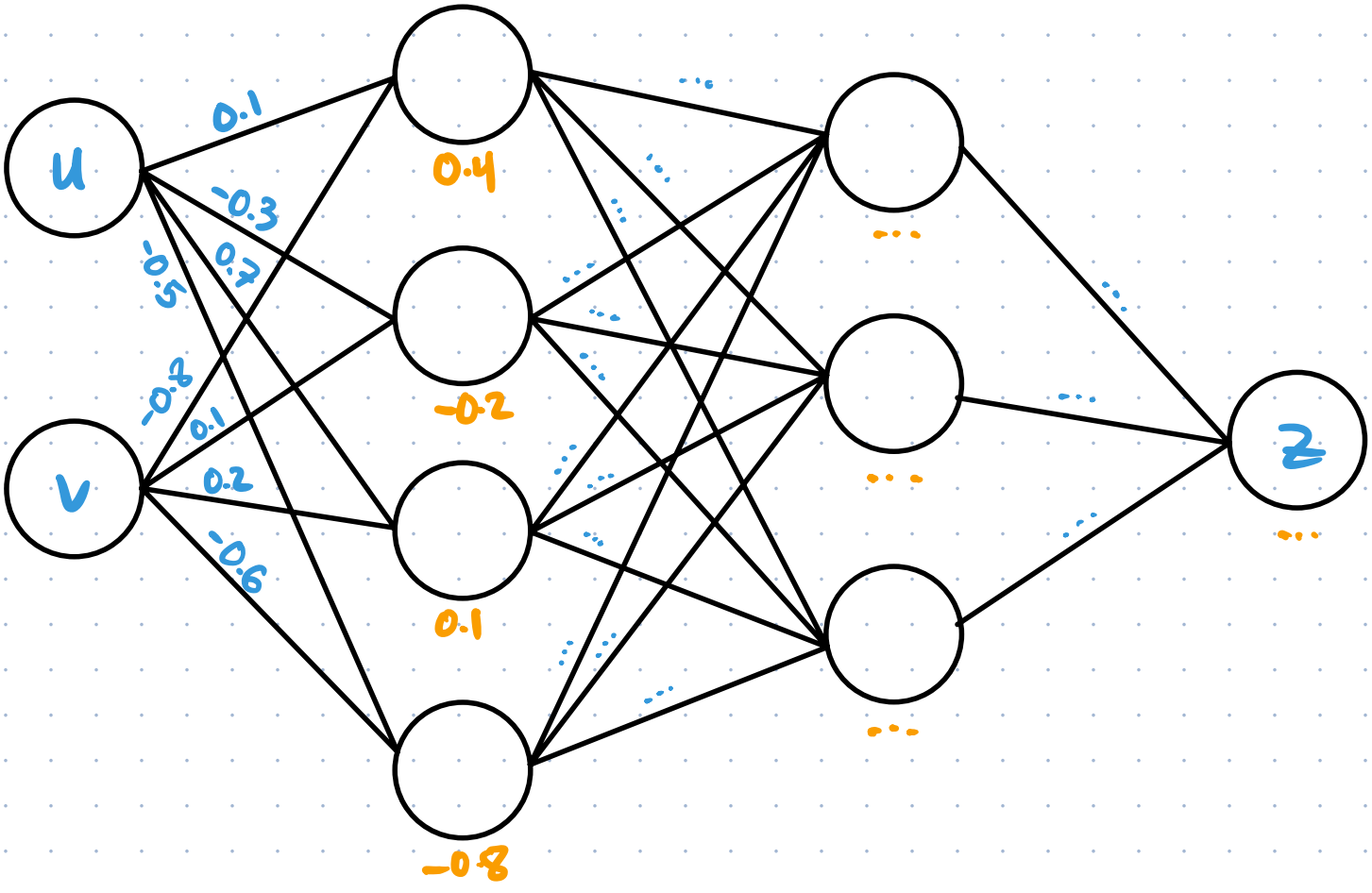
$f(u,v) = z$



Every neuron (except the input layer) has a # attached to it called its **bias**.

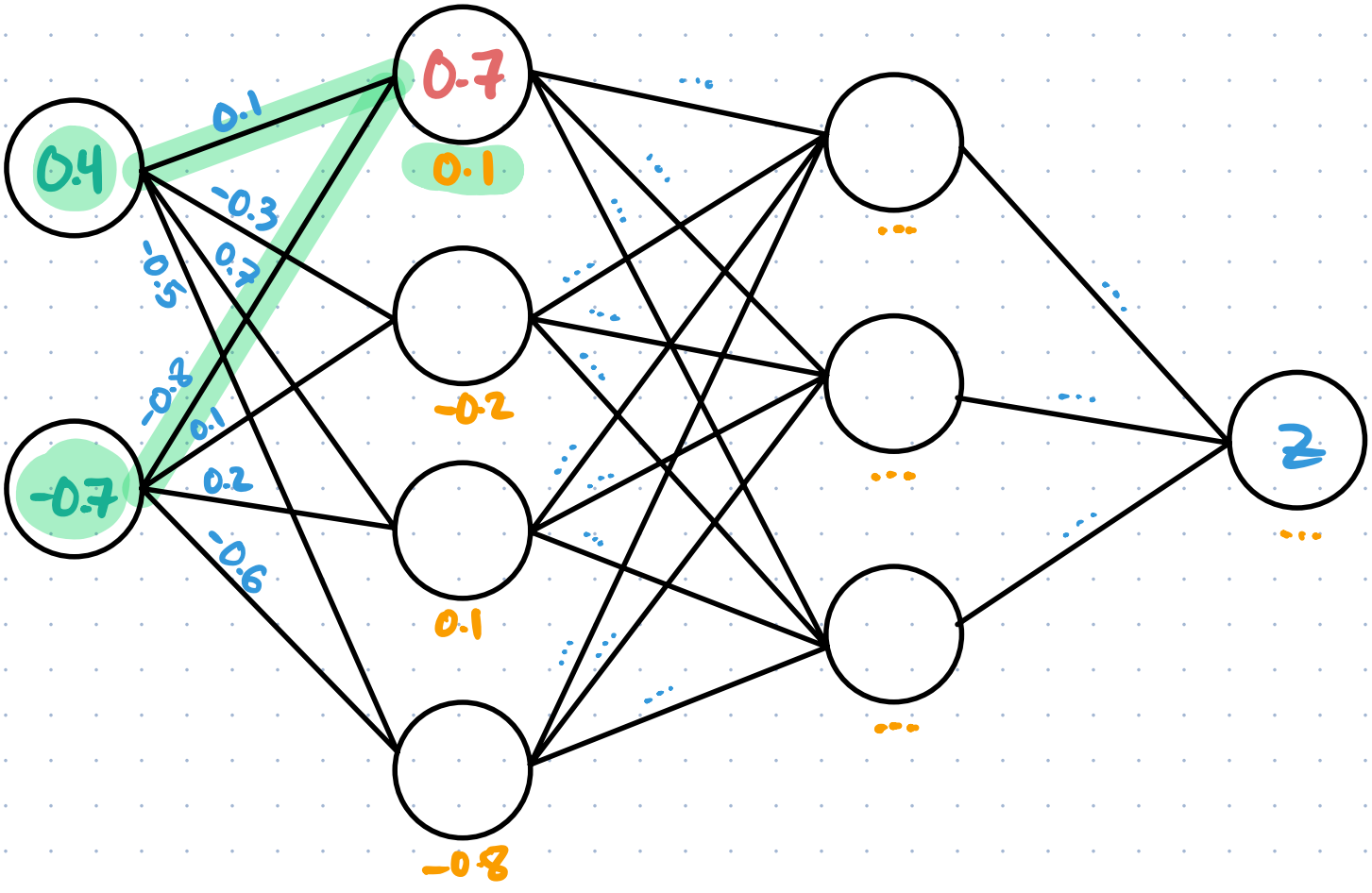
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



The #s u and v get "fed forward" to the neurons in the next layer.

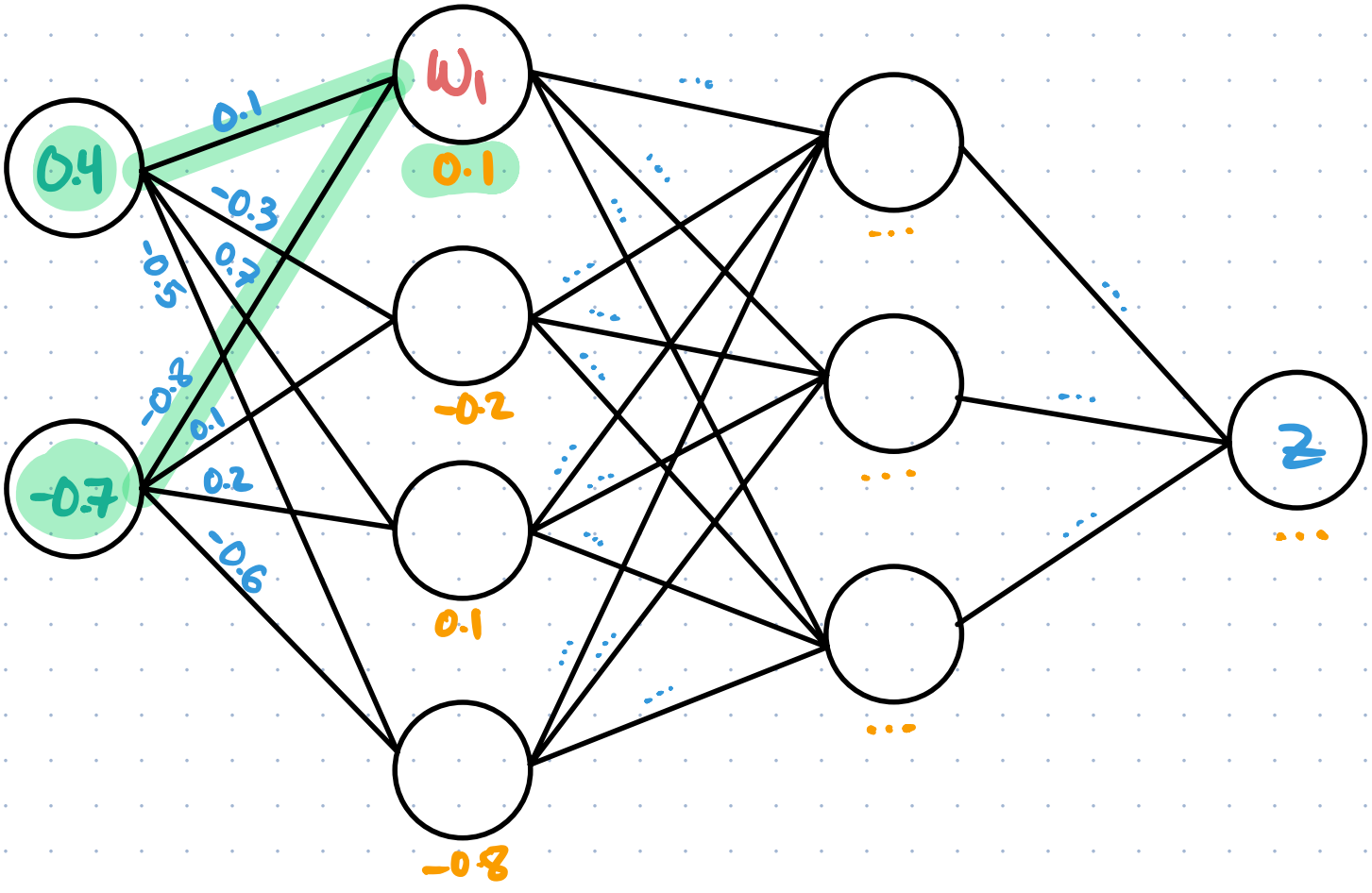
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



To calculate the value for a neuron, we multiply the value of each connected neuron from the last layer by the weight of the connection, add these all up, and add the bias

Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$

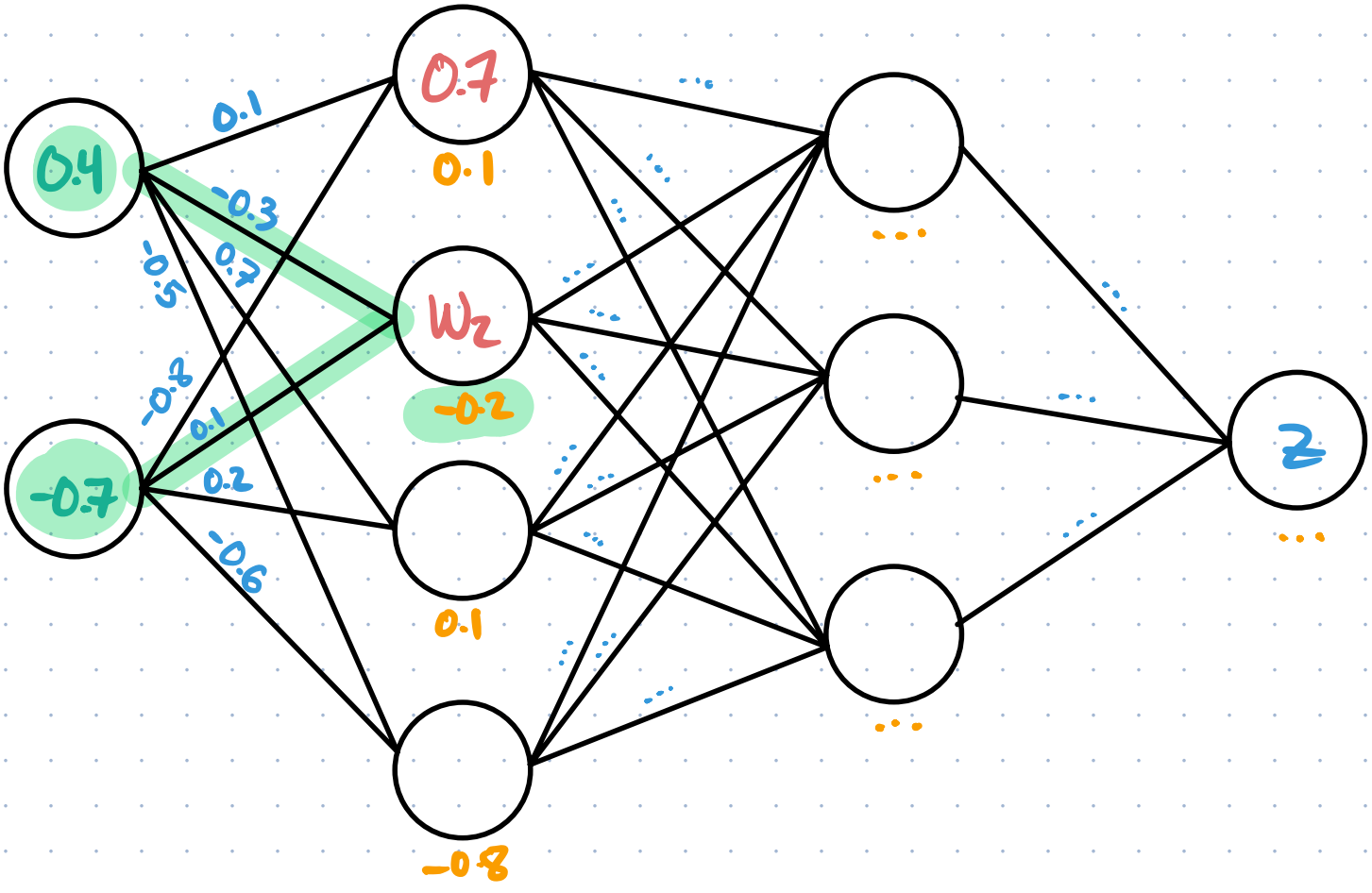


$W_1 = (0.4)(0.1) + (-0.7)(-0.8) + 0.1$

$= 0.7$

Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$

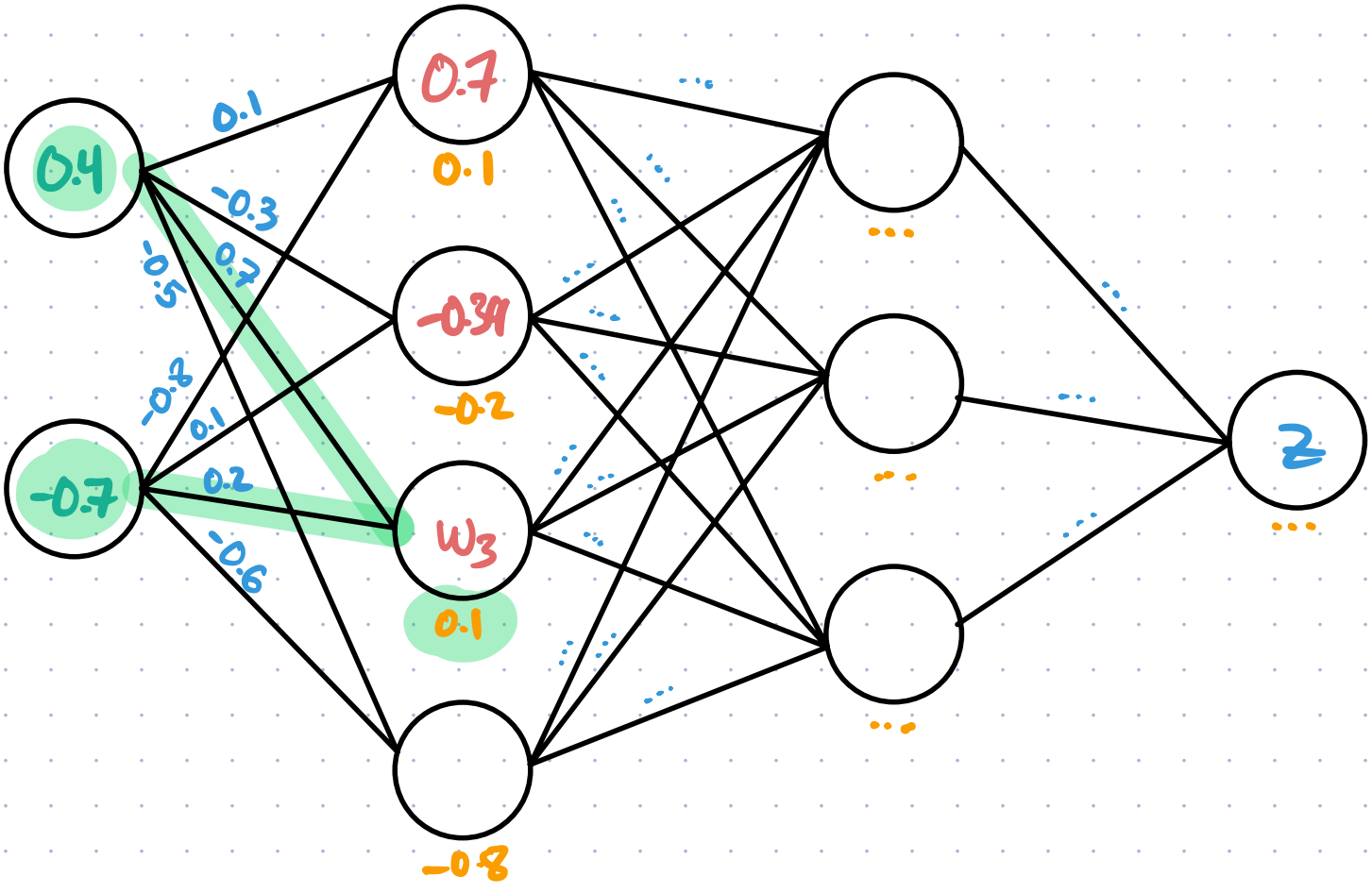


$W_2 = (0.4)(-0.3) + (-0.7)(0.1) + -0.2$

$= -0.39$

Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$

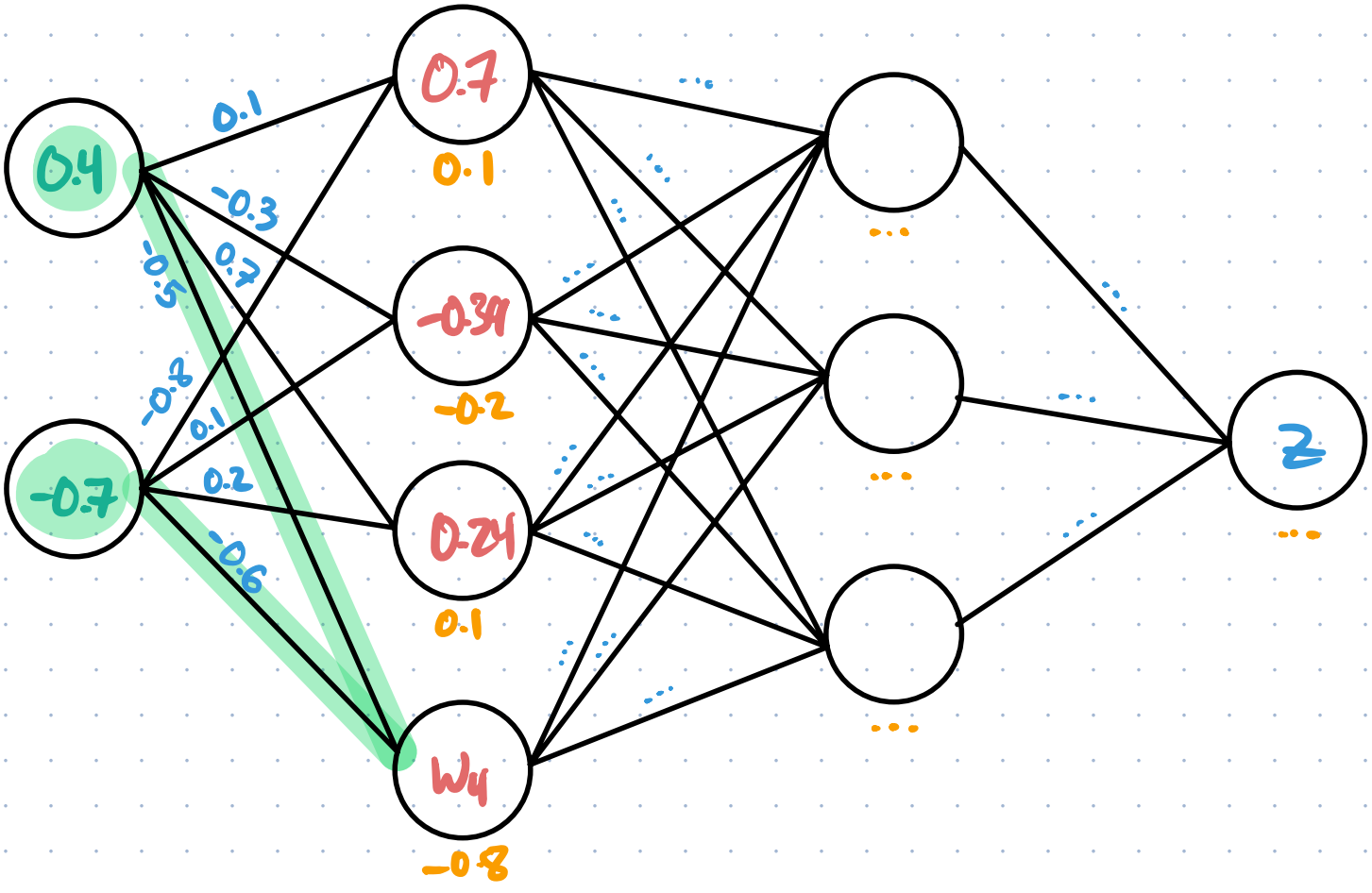


$W_3 = (0.4)(0.7) + (-0.7)(0.2) + 0.1$

$= 0.24$

Example: A NN that takes 2 inputs and has 1 output

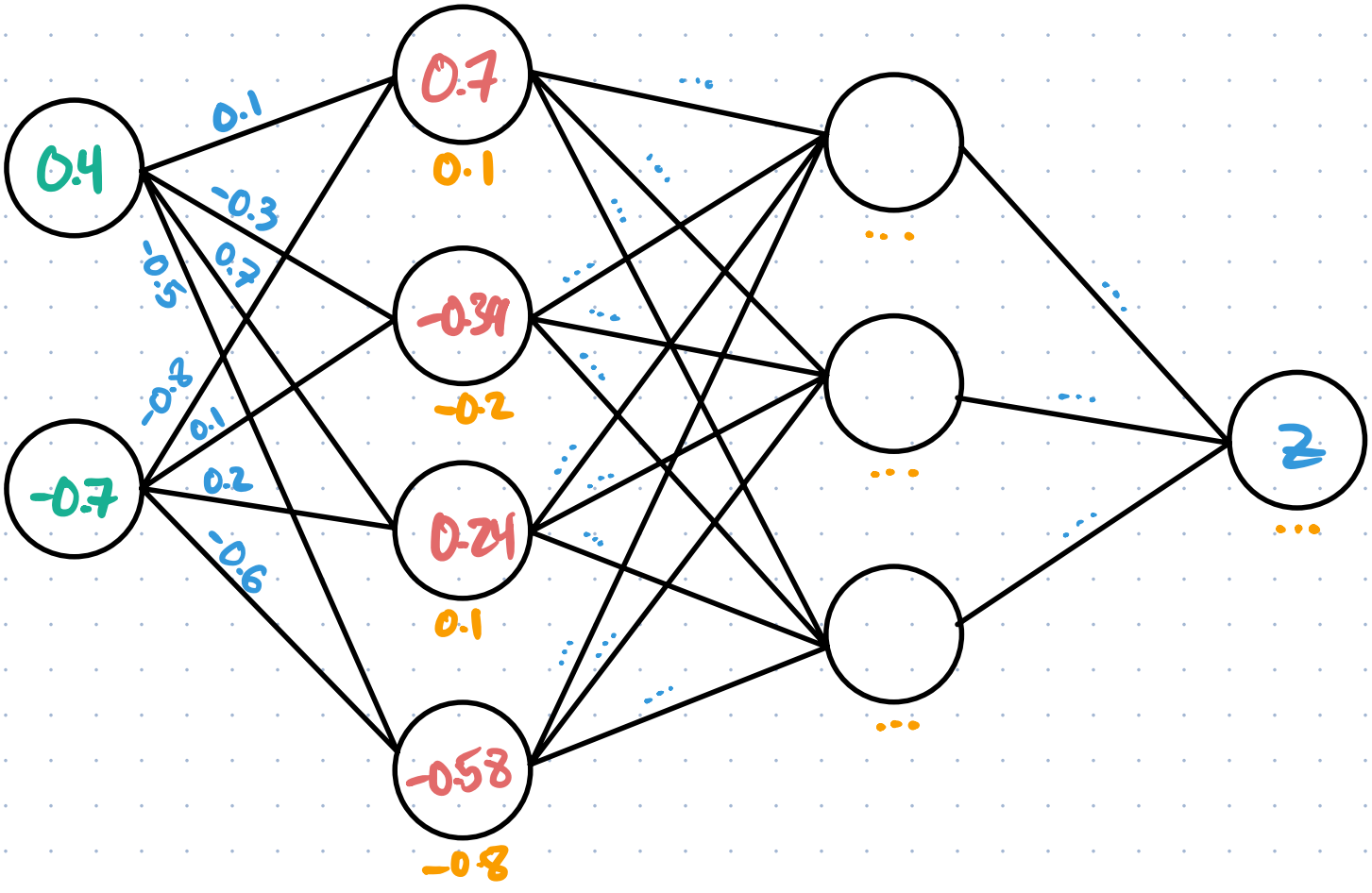
$f(u,v) = z$



$w_4 = (0.4)(-0.5) + (-0.7)(-0.6) + -0.8$

$= -0.58$

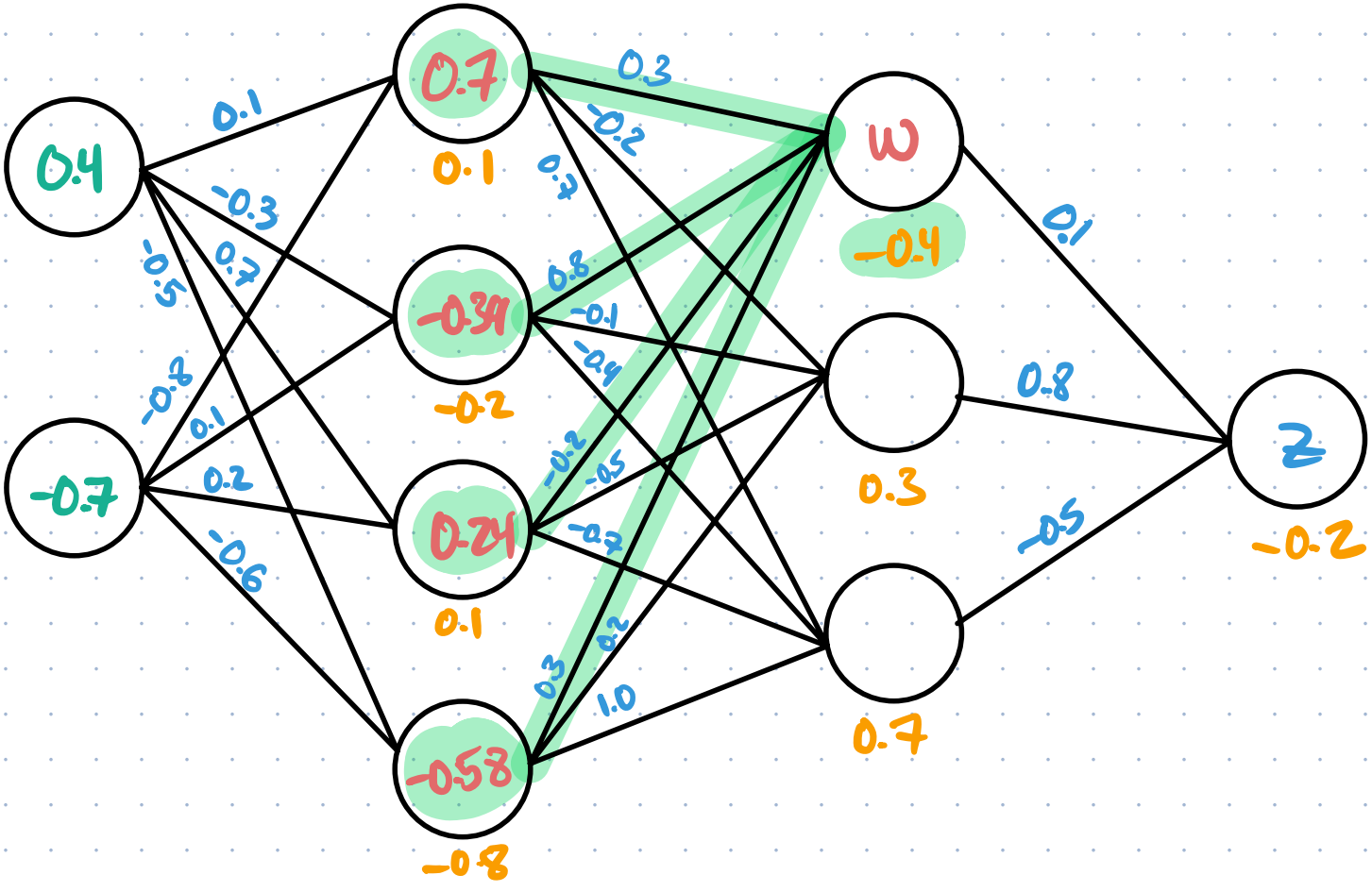
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



Then, the neurons from the first hidden layer feed forward to the next hidden layer.

Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$

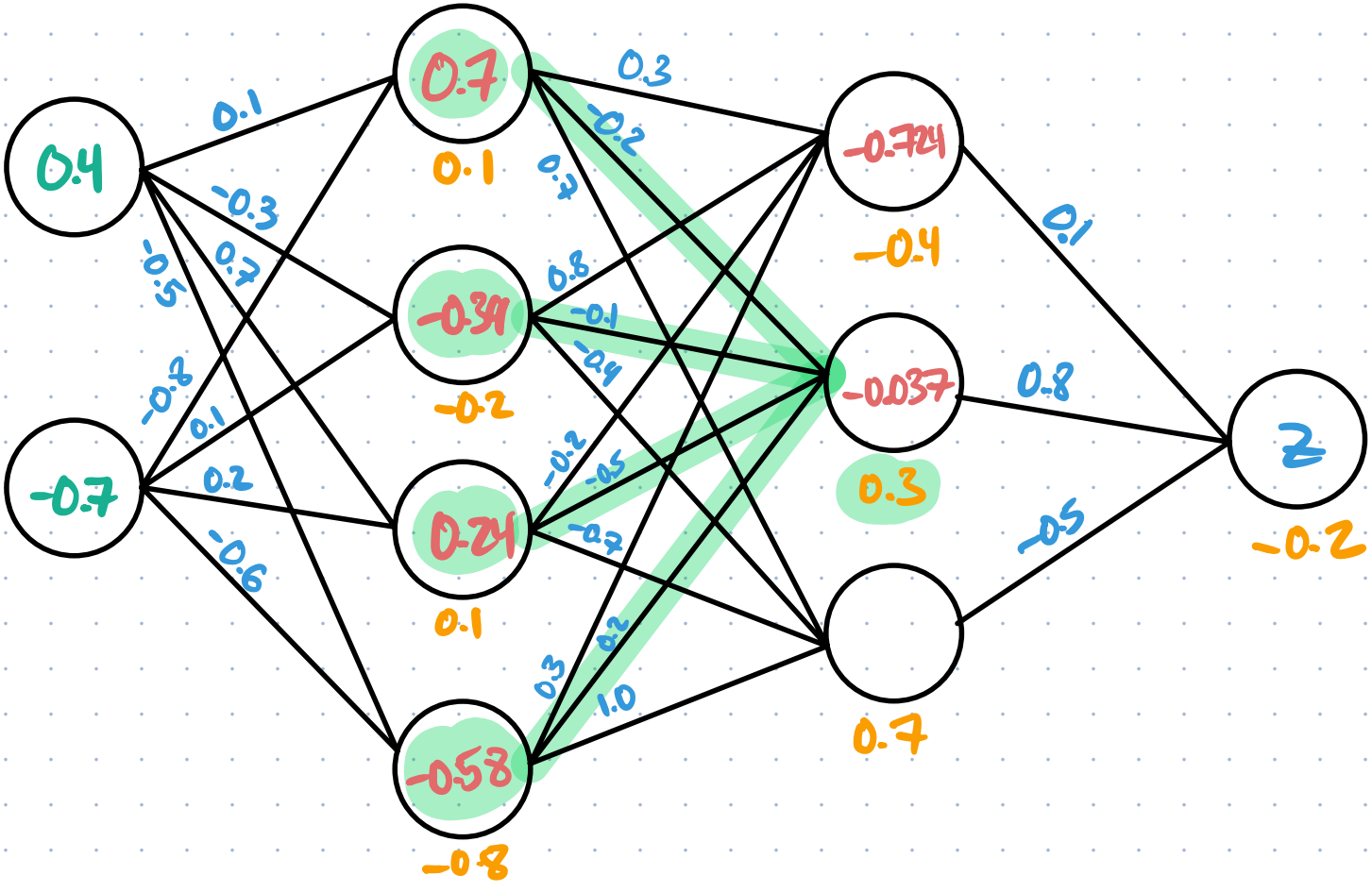


$w = (0.7)(0.3) + (-0.39)(0.8) + (0.24)(-0.2) + (-0.58)(0.3) + -0.4$

≈ -0.724

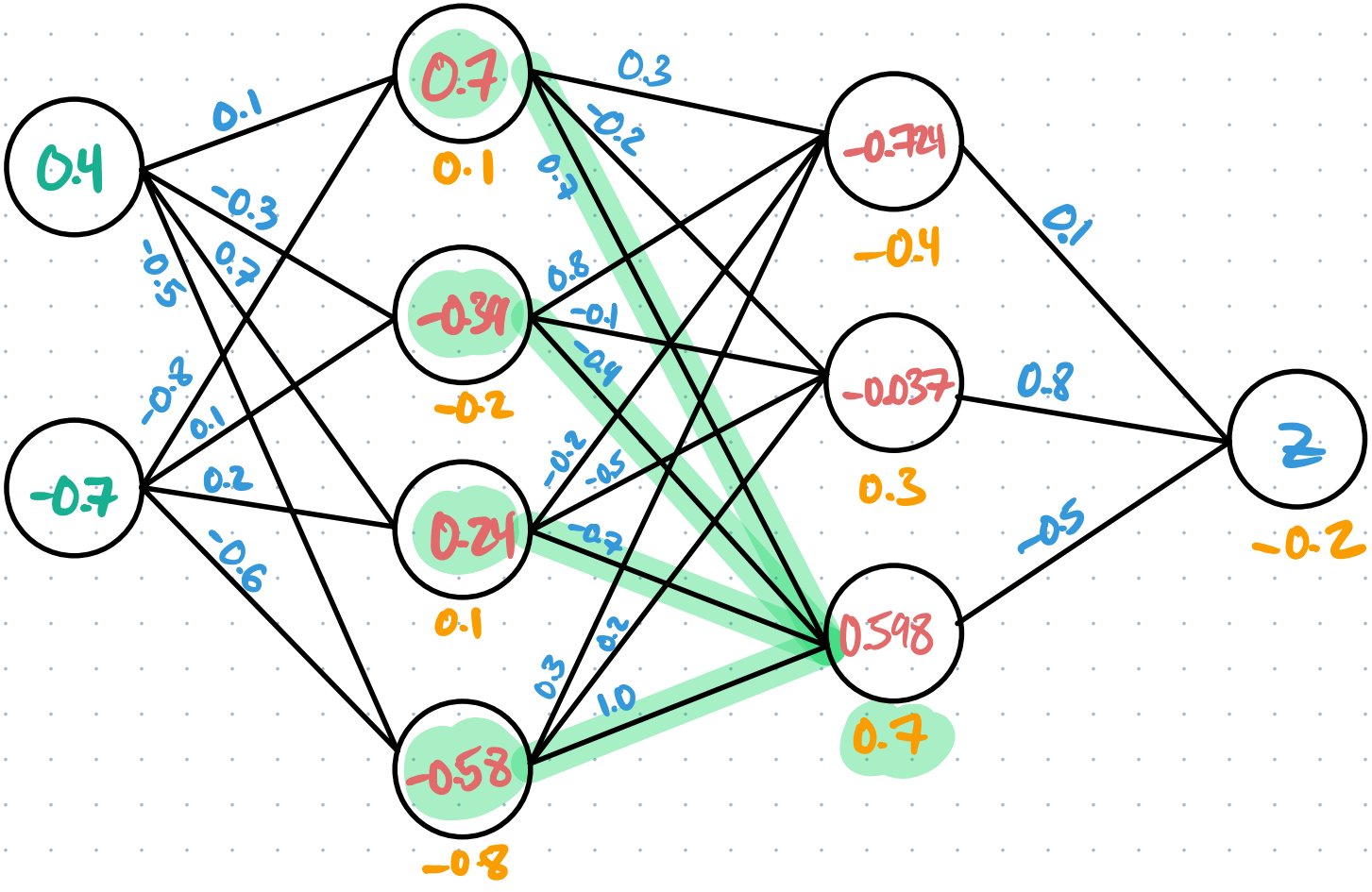
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



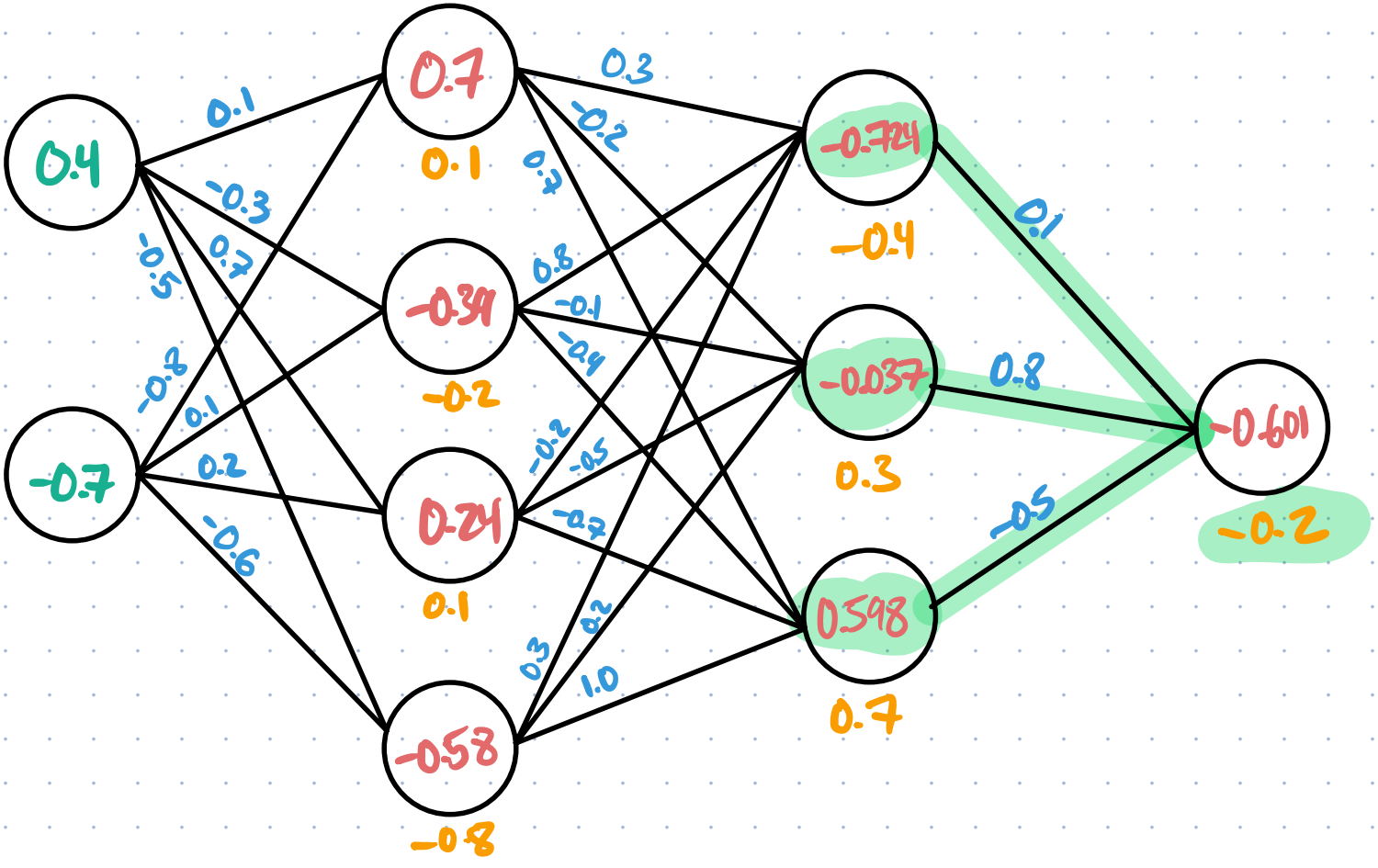
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



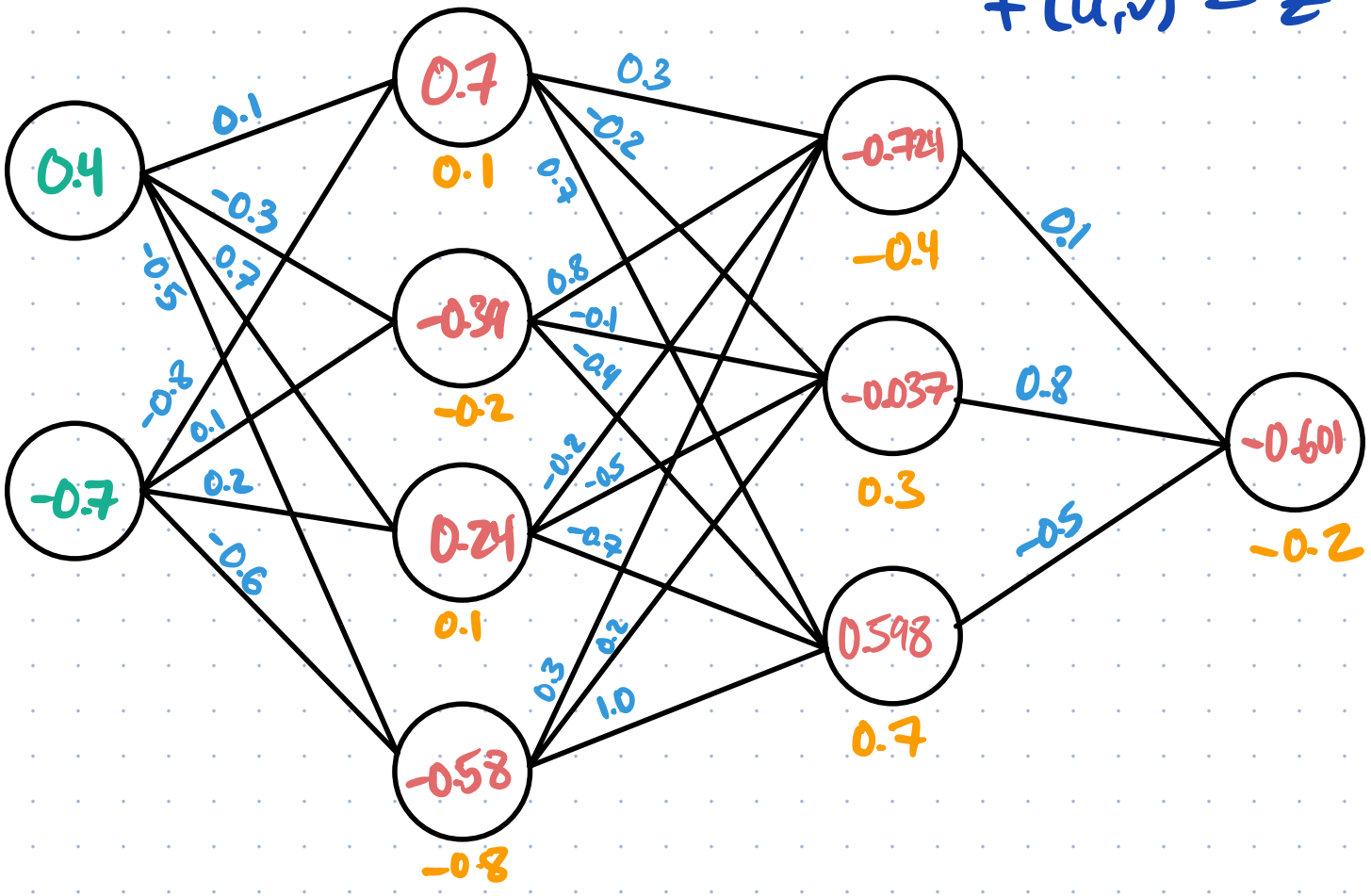
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$

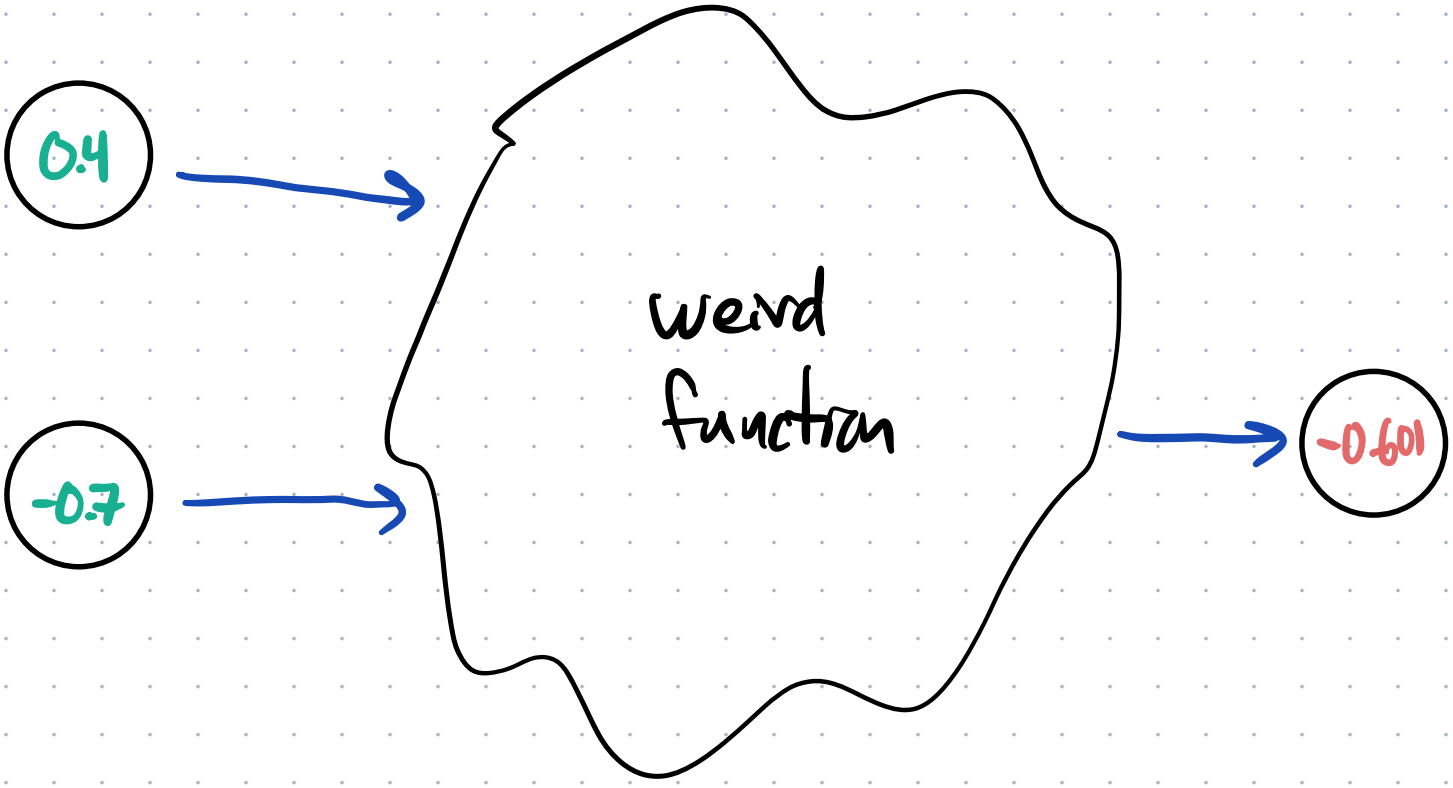


So, for this particular neural network, with these weights and biases, the inputs $(0.4, -0.7)$ produce output -0.601

$$f(0.4, -0.7) = -0.601$$

Example: A NN that takes 2 inputs and has 1 output

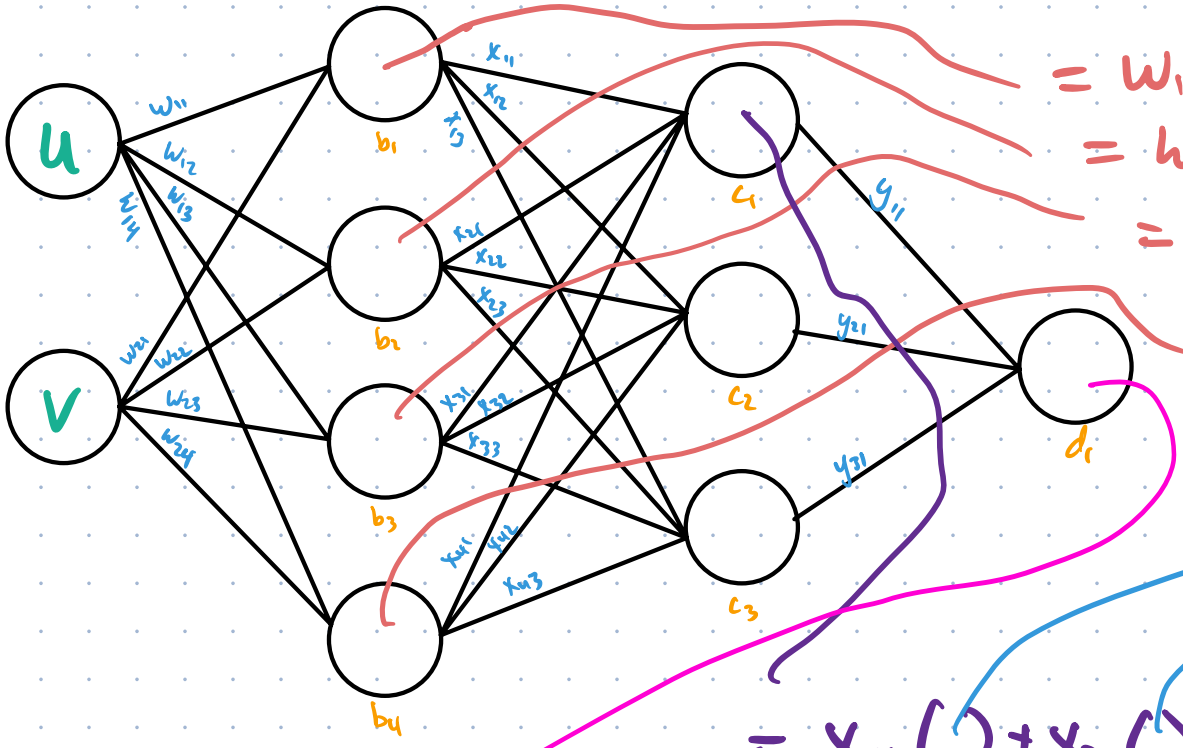
$f(u,v) = z$



So, for this particular neural network, with these weights and biases, the inputs $(0.4, -0.7)$ produce output -0.601

* NNs can have any # of layers
* The layers can have any # of neurons in them
* If every neuron in a layer is connected to every neuron in the next layer, then the network is "fully connected"

* But we're missing a really key component so far.



$= w_{11}u + w_{21}v + b_1$

$= w_{12}u + w_{22}v + b_2$

$= w_{13}u + w_{23}v + b_3$

$= w_{14}u + w_{24}v + b_4$

$= x_{11}(\ ) + x_{21}(\ ) + x_{31}(\ ) + x_{41}(\ ) + c_1$

$= (x_{11}w_{11} + x_{21}w_{12} + x_{31}w_{13} + x_{41}w_{14}) \cdot u$

$+ (x_{11}w_{21} + x_{21}w_{22} + x_{31}w_{23} + x_{41}w_{24}) \cdot v$

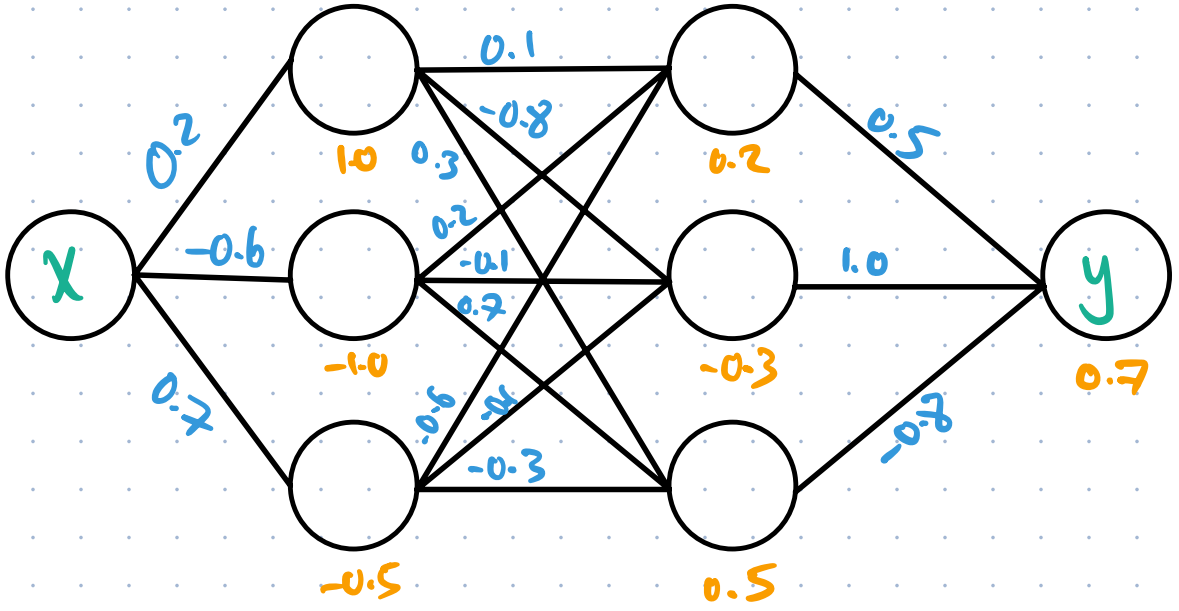
$+ (b_1 + b_2 + b_3 + b_4)$

=

```
(w11*x11*y11 + w11*x12*y21 + w11*x13*y31 + w12*x21*y11 + w12*x22*y21 + w12*x23*y31 + w13*x31*y11 + w13*x32*y21 + w13*x33*y31 + w14*x41*y11 + w14*x42*y21 + w14*x43*y31) * u
 + (w21*x11*y11 + w21*x12*y21 + w21*x13*y31 + w22*x21*y11 + w22*x22*y21 + w22*x23*y31 + w23*x31*y11 + w23*x32*y21 + w23*x33*y31 + w24*x41*y11 + w24*x42*y21 + w24*x43*y31) * v
 + b1*x11*y11 + b1*x12*y21 + b1*x13*y31 + b2*x21*y11 + b2*x22*y21 + b2*x23*y31 + b3*x31*y11 + b3*x32*y21 + b3*x33*y31 + b4*x41*y11 + b4*x42*y21 + b4*x43*y31 + c1*y11 + c2*y21 + c3*y31 + d1
```

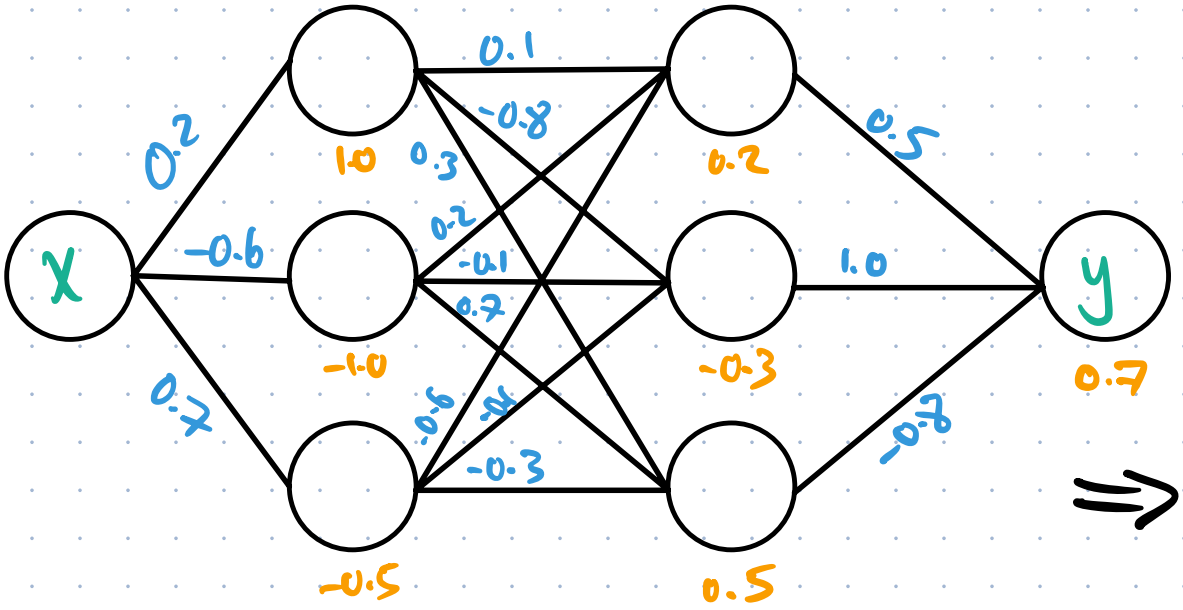
As we've built them NNs are still just (big) linear functions, so this is no better than linear regression.

The problem is even more obvious when there's only one input and one output.



$$\Rightarrow y = 0.026x - 0.25$$

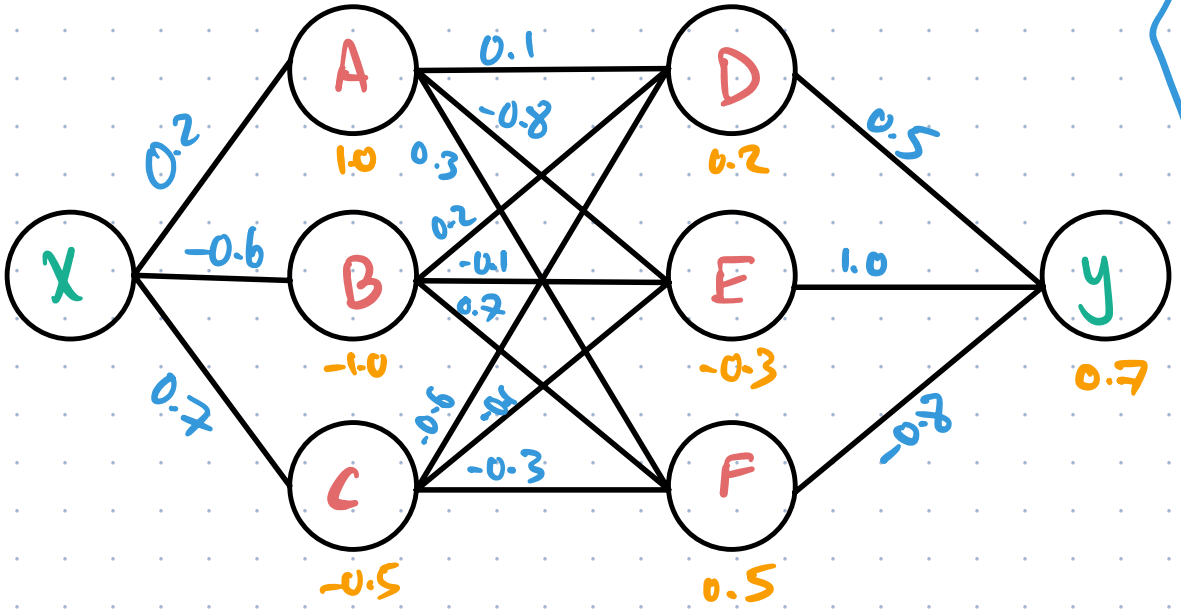
Just a line defined in a complicated way!



$$\Rightarrow y = 0.026x - 0.25$$

$(0.026)(0.3) - 0.25 = -0.2422$ ✓

Understanding check: Let's feed forward $x = 0.3$



$A = (0.2)(0.3) + 1.0 = 1.06$

$B = (-0.6)(0.3) - 1.0 = -1.18$

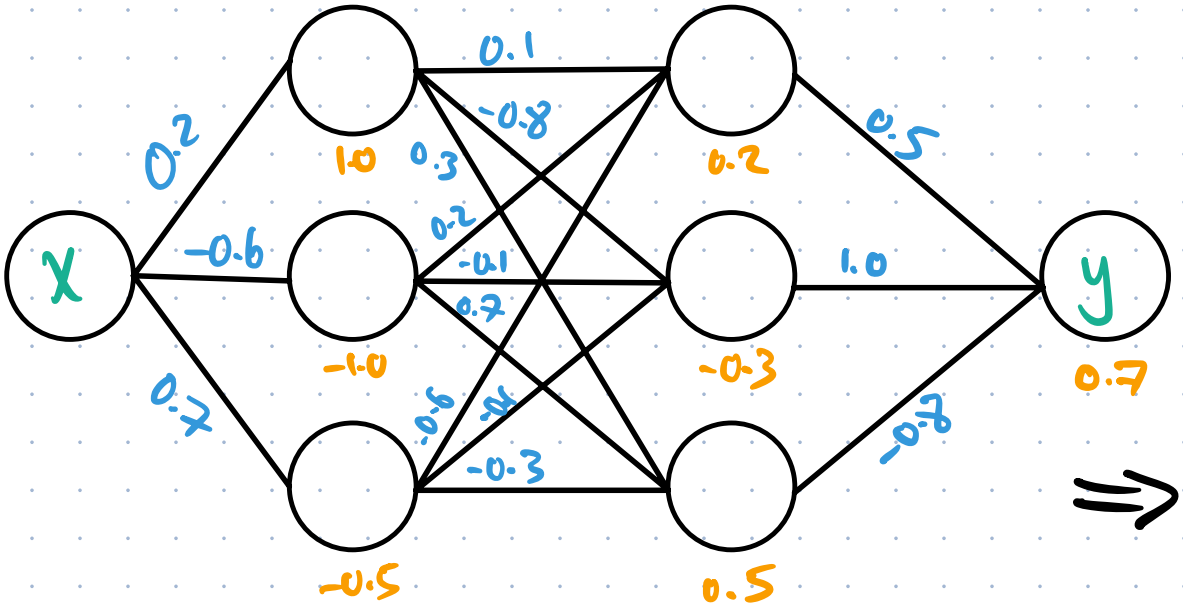
$C = (0.7)(0.3) - 0.5 = -0.29$

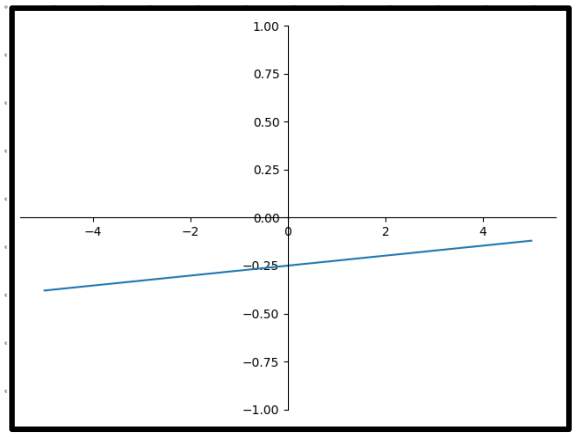
$D = 0.244$

$E = -1.001$

$F = 0.079$

$y = -0.2422$





$\Rightarrow y = 0.026x - 0.25$

Understanding check:

* x is the input variable, it changes
* y is the output variable, it changes as x changes

The weights and biases stay fixed. They are what define this particular function, like "m" and "b" in a line $y = mx + b$. If we change them, that's a different NN.

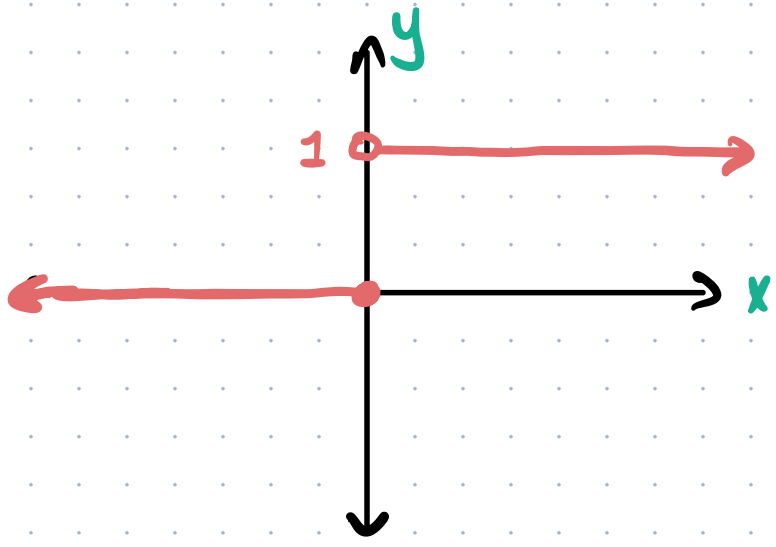
Activation Functions

In order to make NNs capable of representing non-linear functions, we pass the output of each neuron through an "activation function" before passing it on to the next layer.

The activation functions themselves are non-linear.

AF #1: Step Function (old school, not used often anymore)

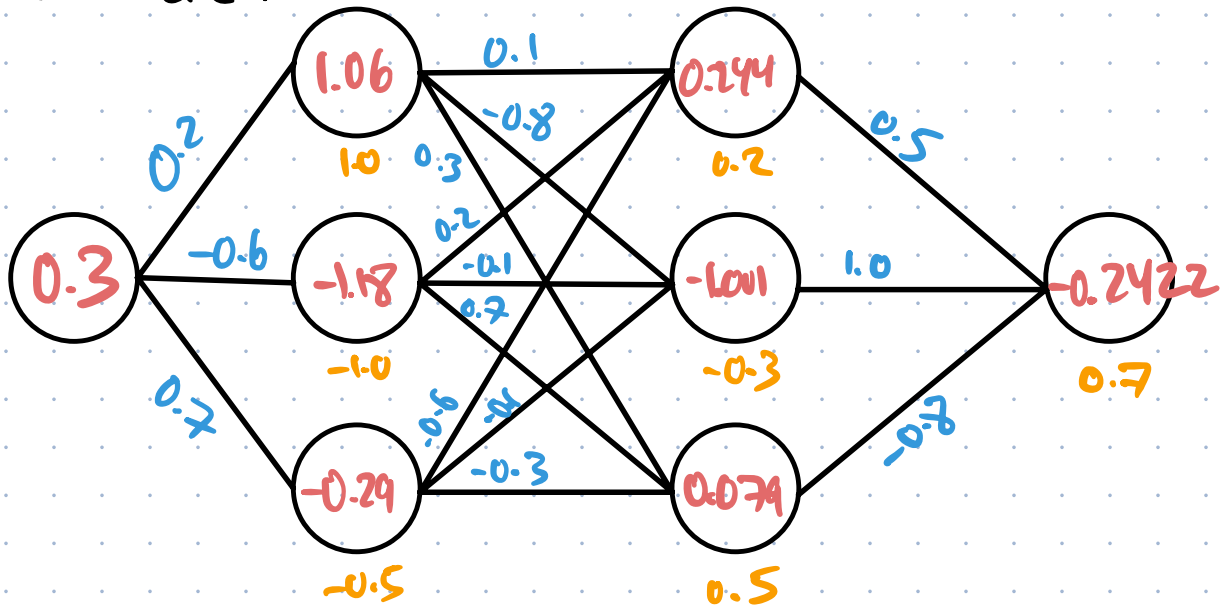
$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$



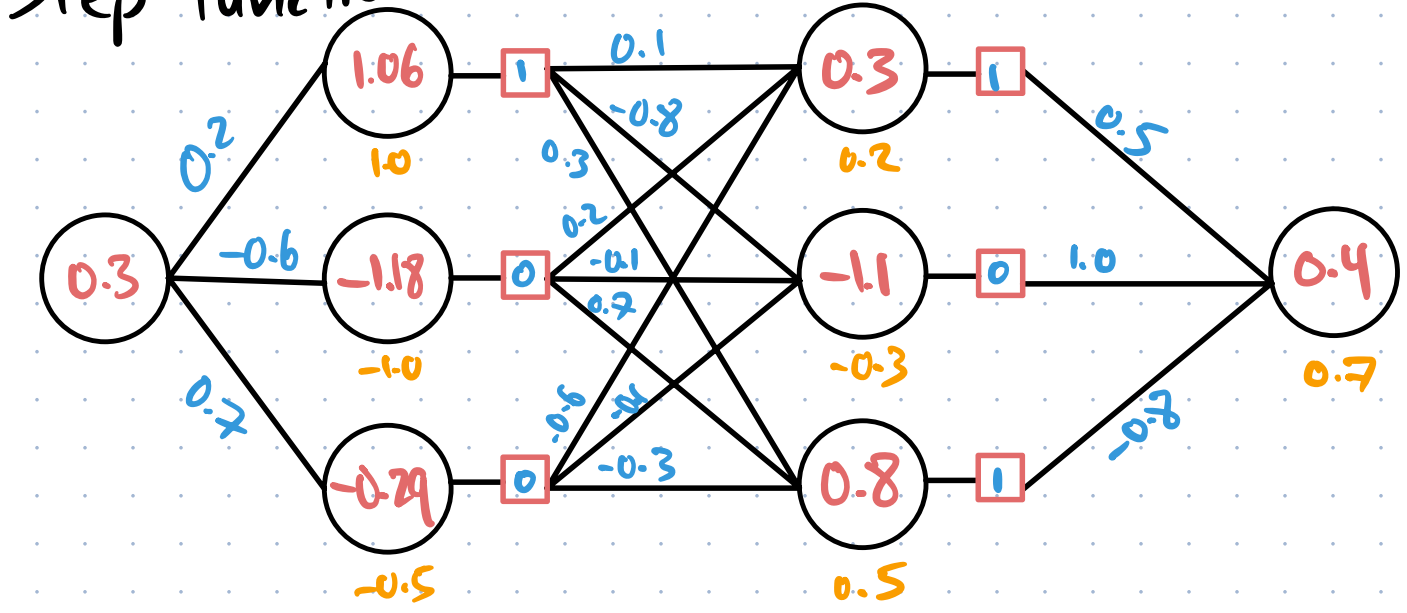
With this AF each neuron only outputs 0 or 1, not any decimal.

(off → 0, on → 1)

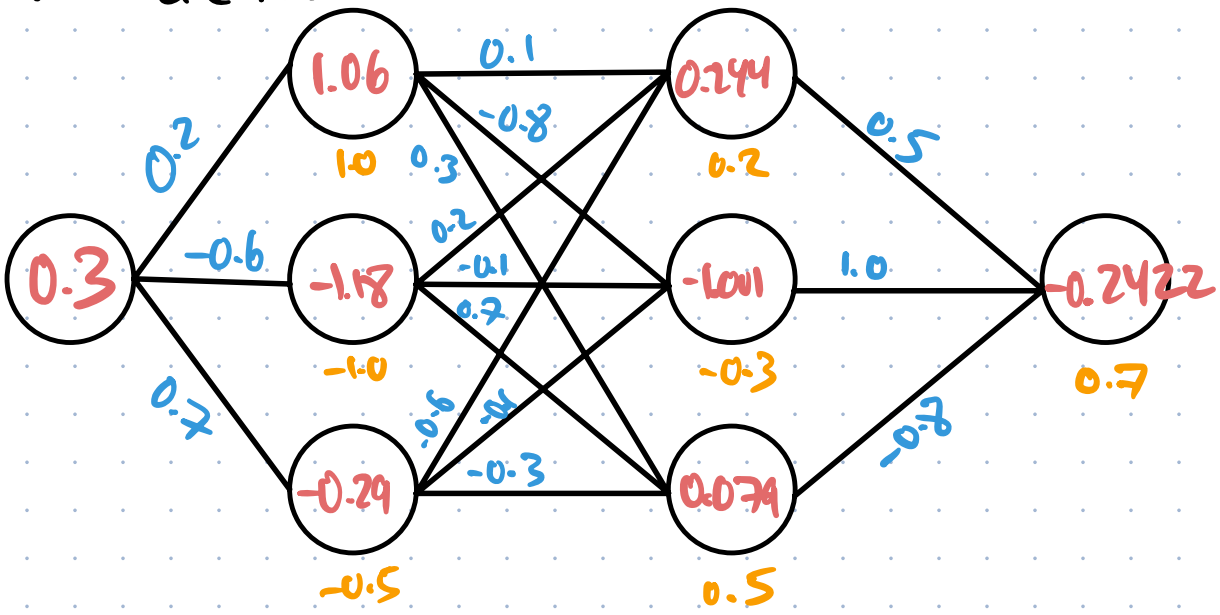
no activation:

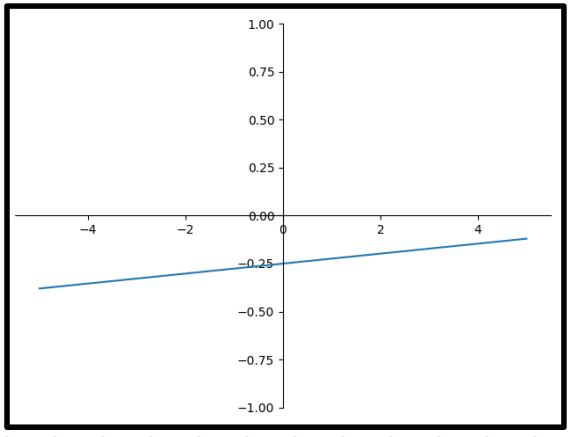


step function:

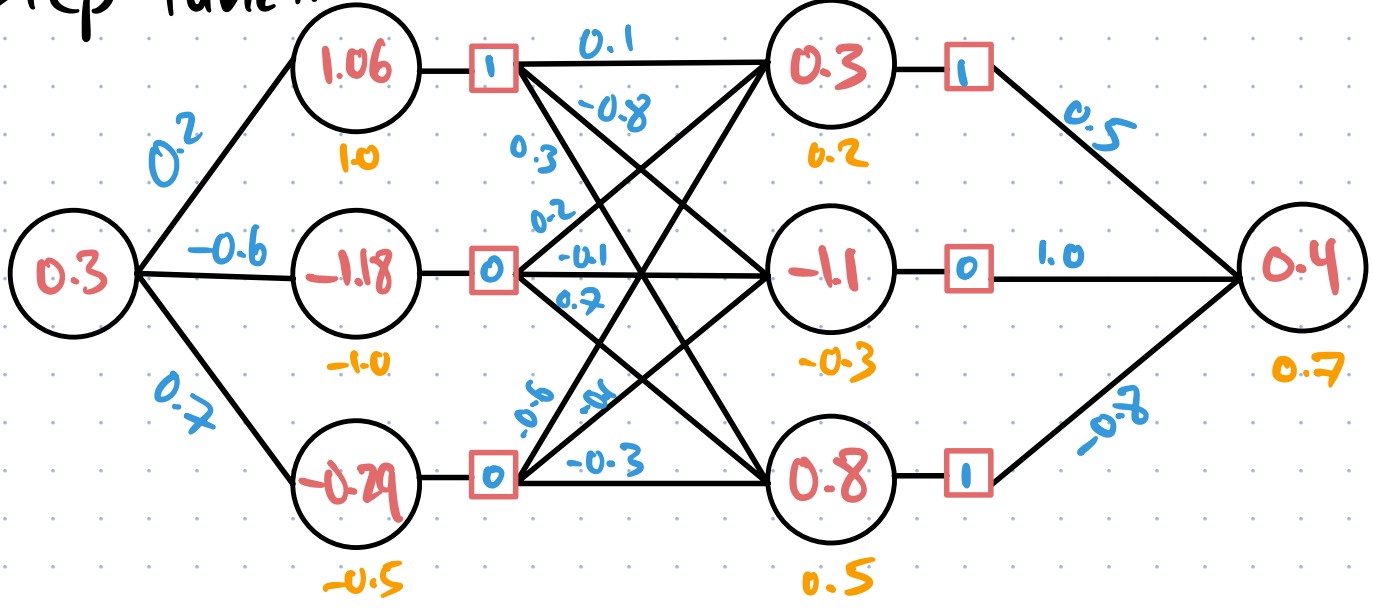


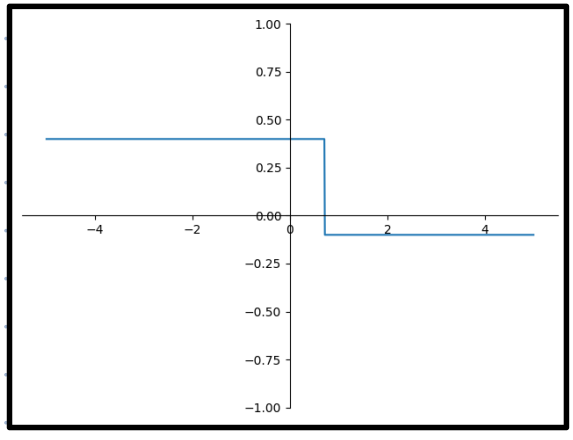
no activation:



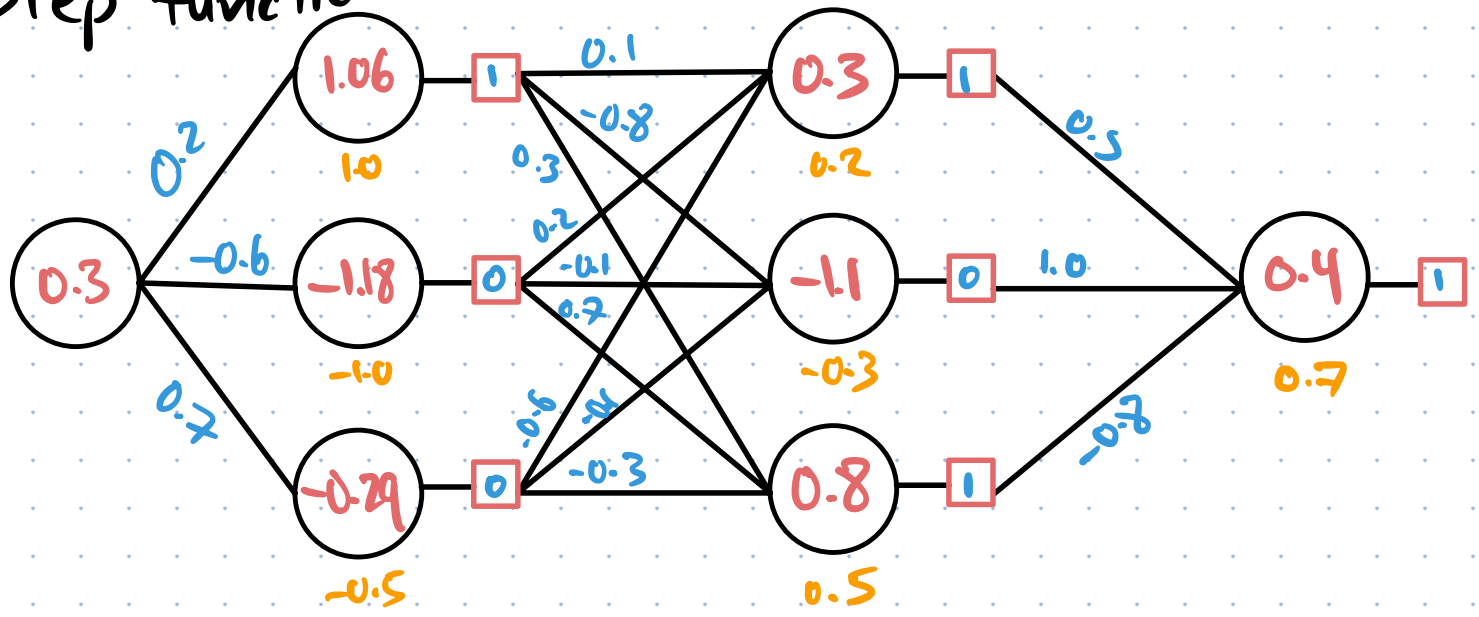


step function:

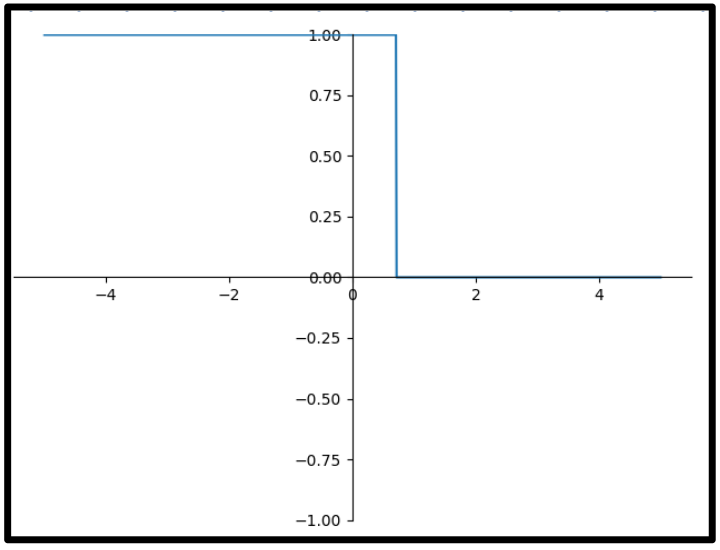




step function:



We can also add the AF to the output neuron, but usually the output layer is treated differently, depending on our goal.

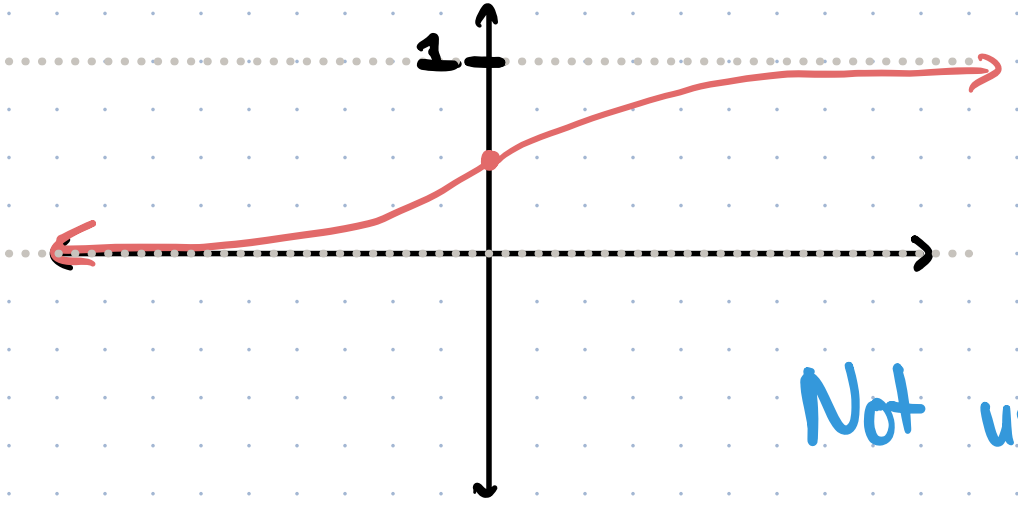


AF #2 : Sigmoid

A problem with the step function ⎍ is that it throws away a lot of information.

$f(0.001) = f(100) = 1$

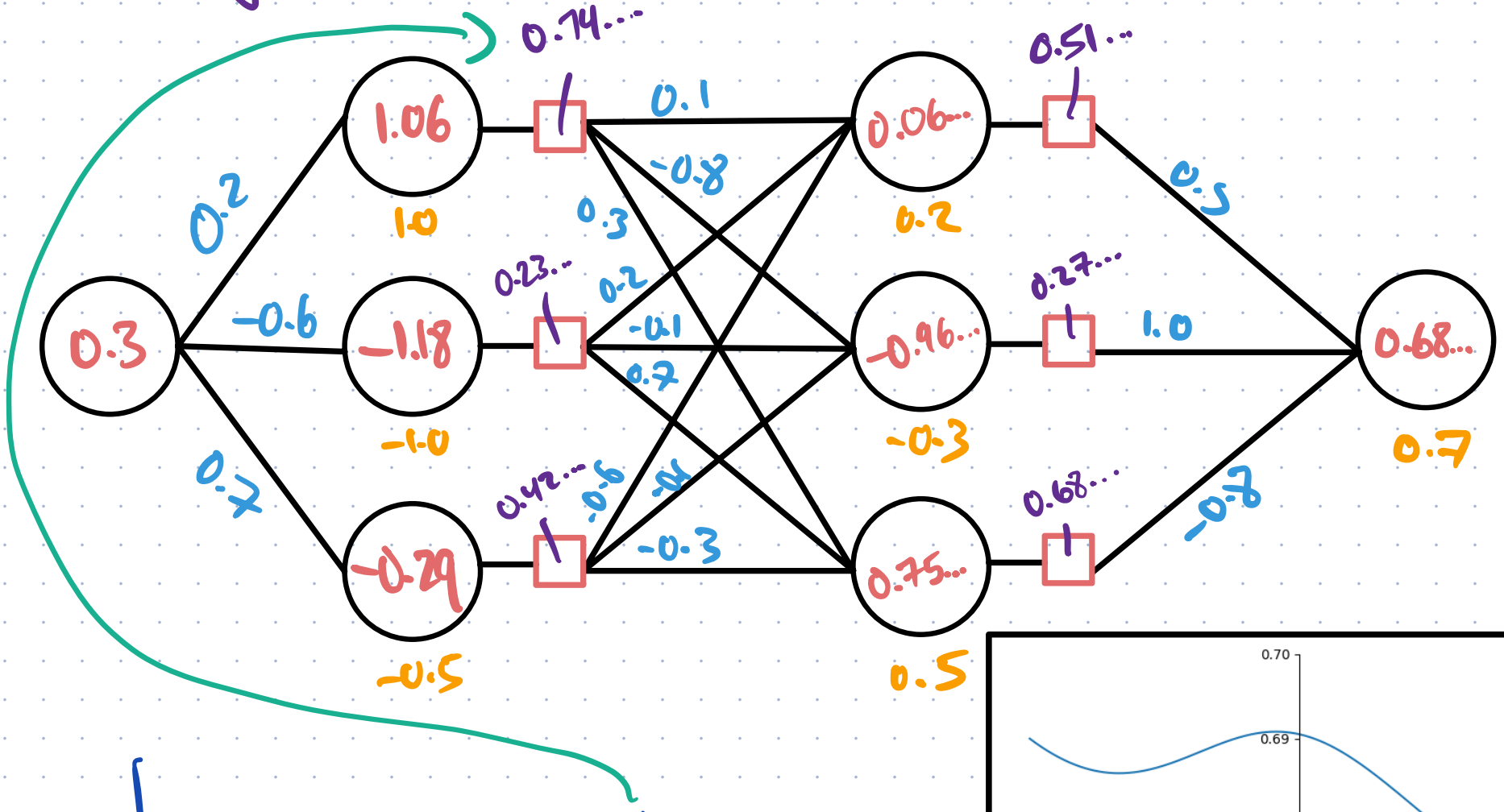
Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$



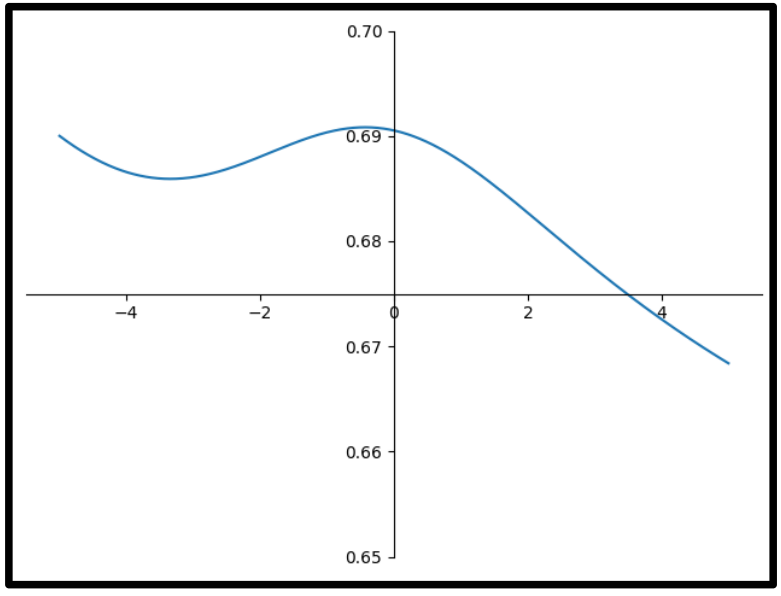
More gentle, and not as lossy.

Not used much anymore

With sigmoid activation

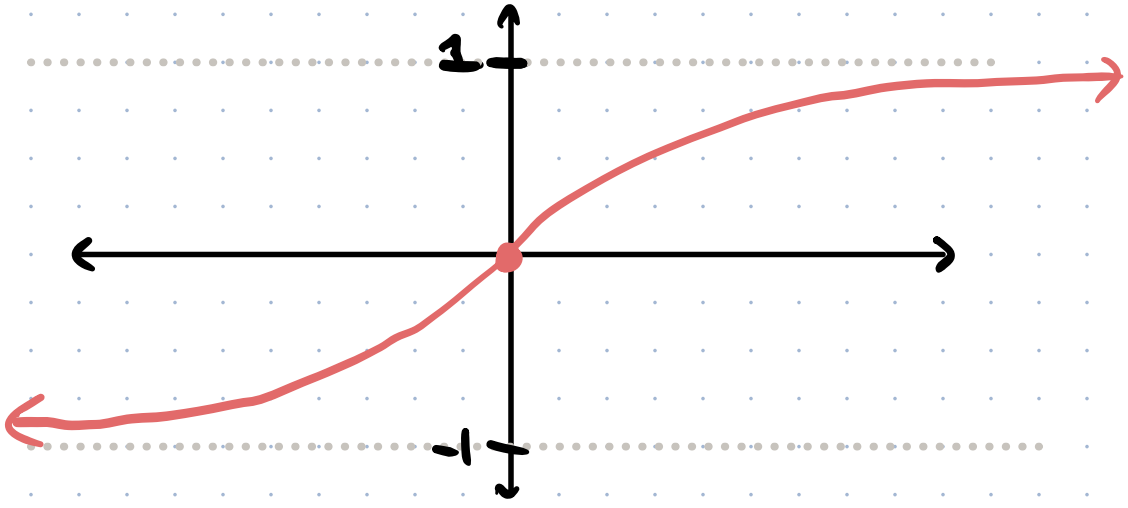


$$\frac{1}{1+e^{-1.06}} \simeq 0.74$$



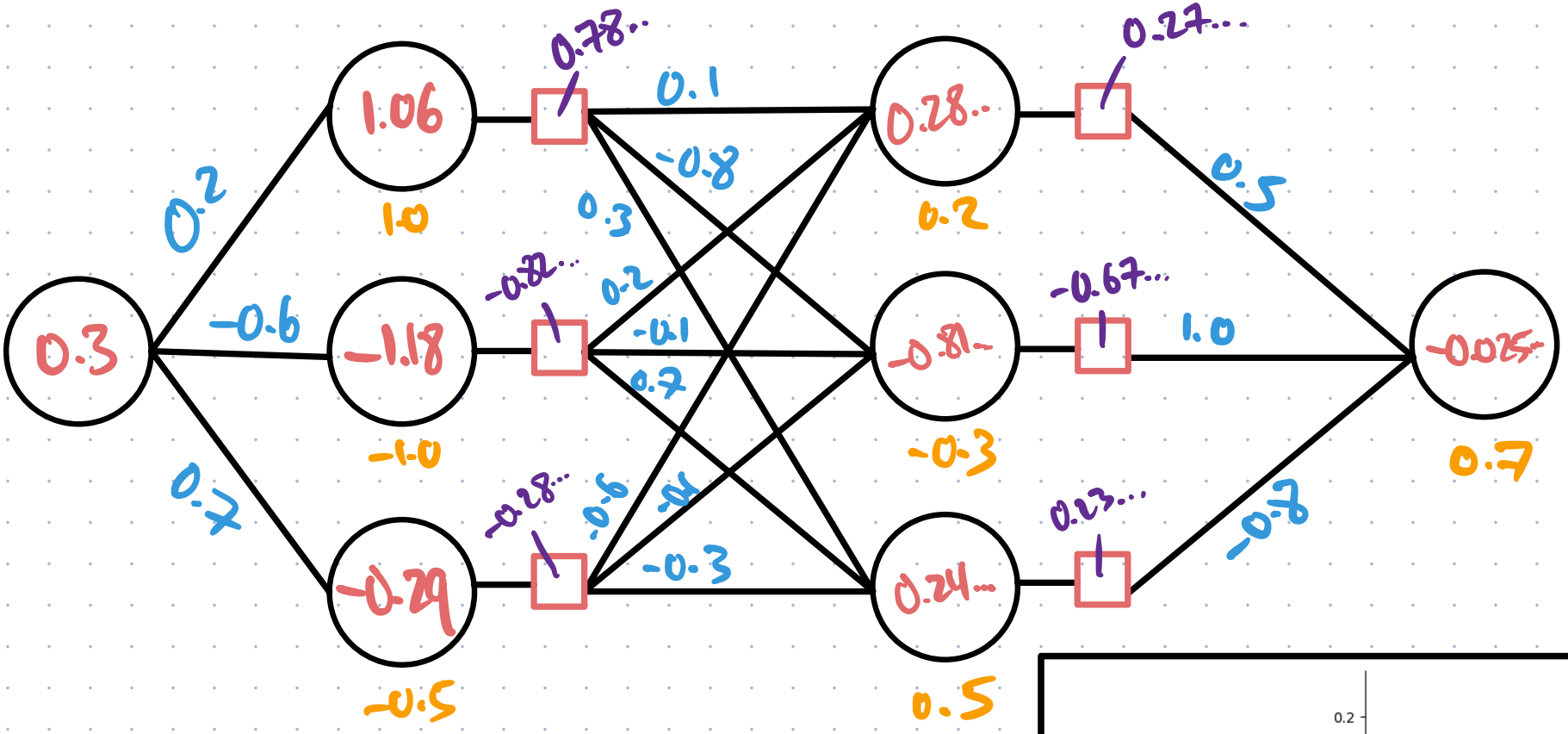
AF #3: Hyperbolic Tangent (tanh)

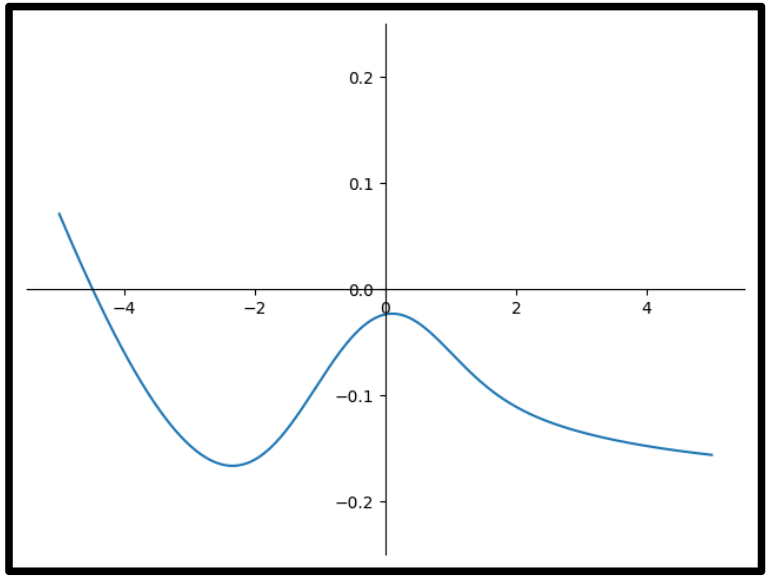
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



a bit like sigmoid, but can be negative

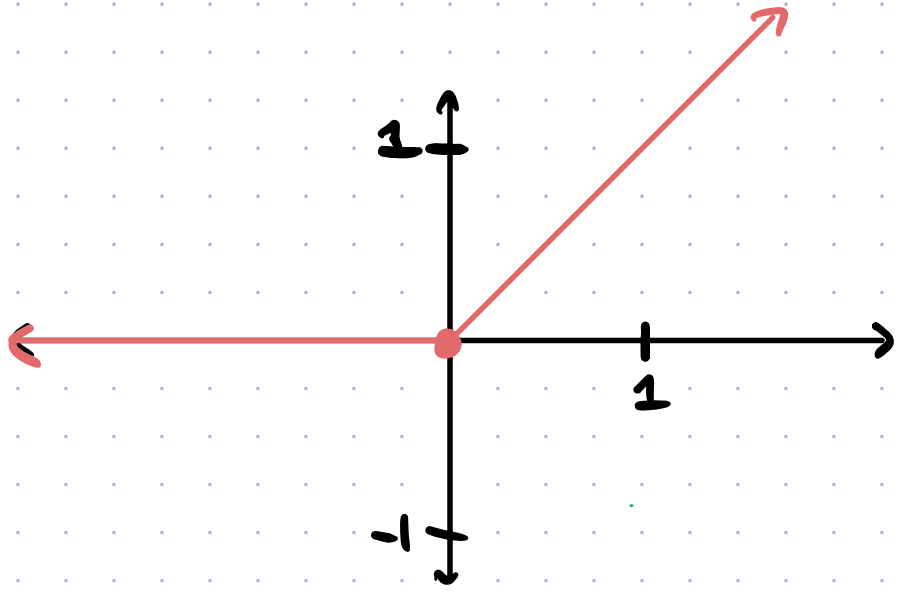
With tanh activation





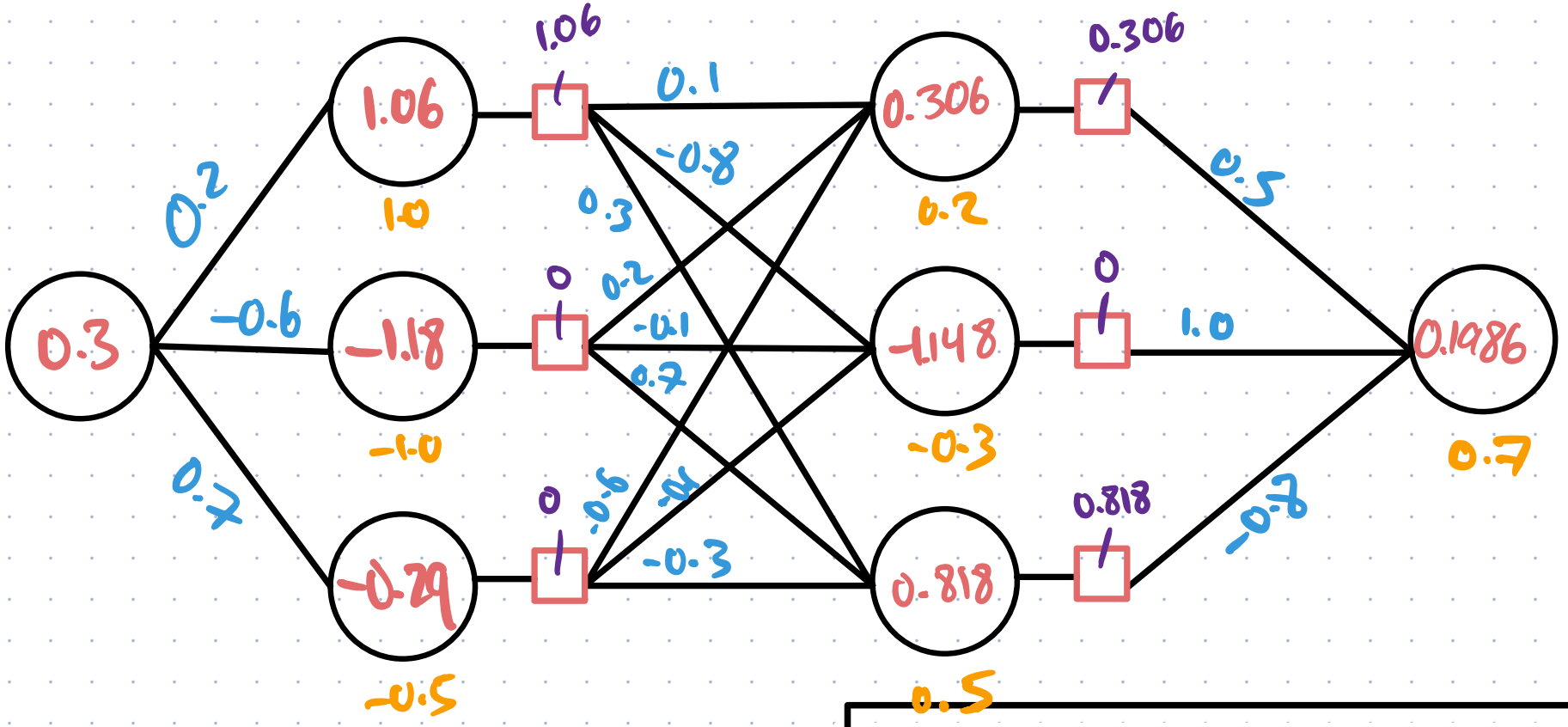
AF #4: Rectified Linear Unit (ReLU)

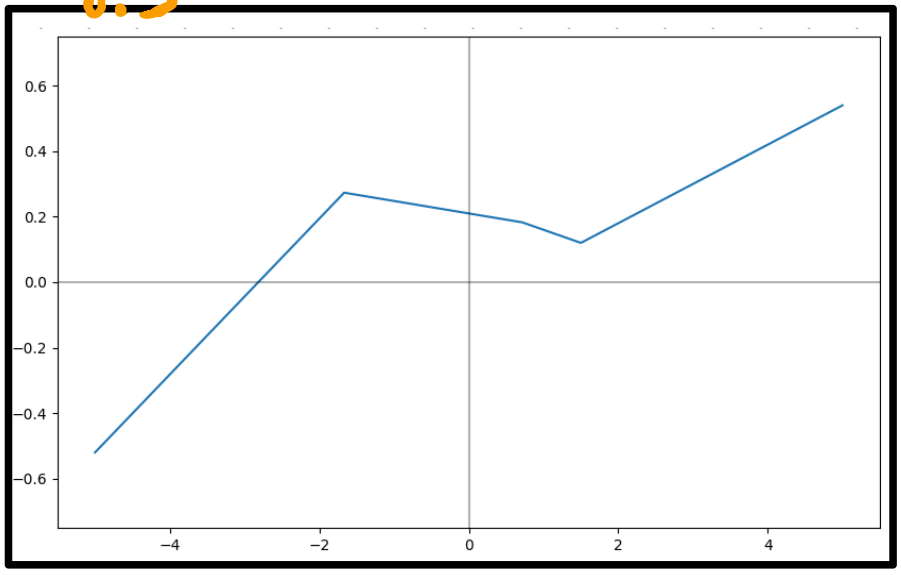
$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$



Neuron is "on" if ≥ 0, and keeps its value, and "off" if < 0, and returns no value

With ReLU activation



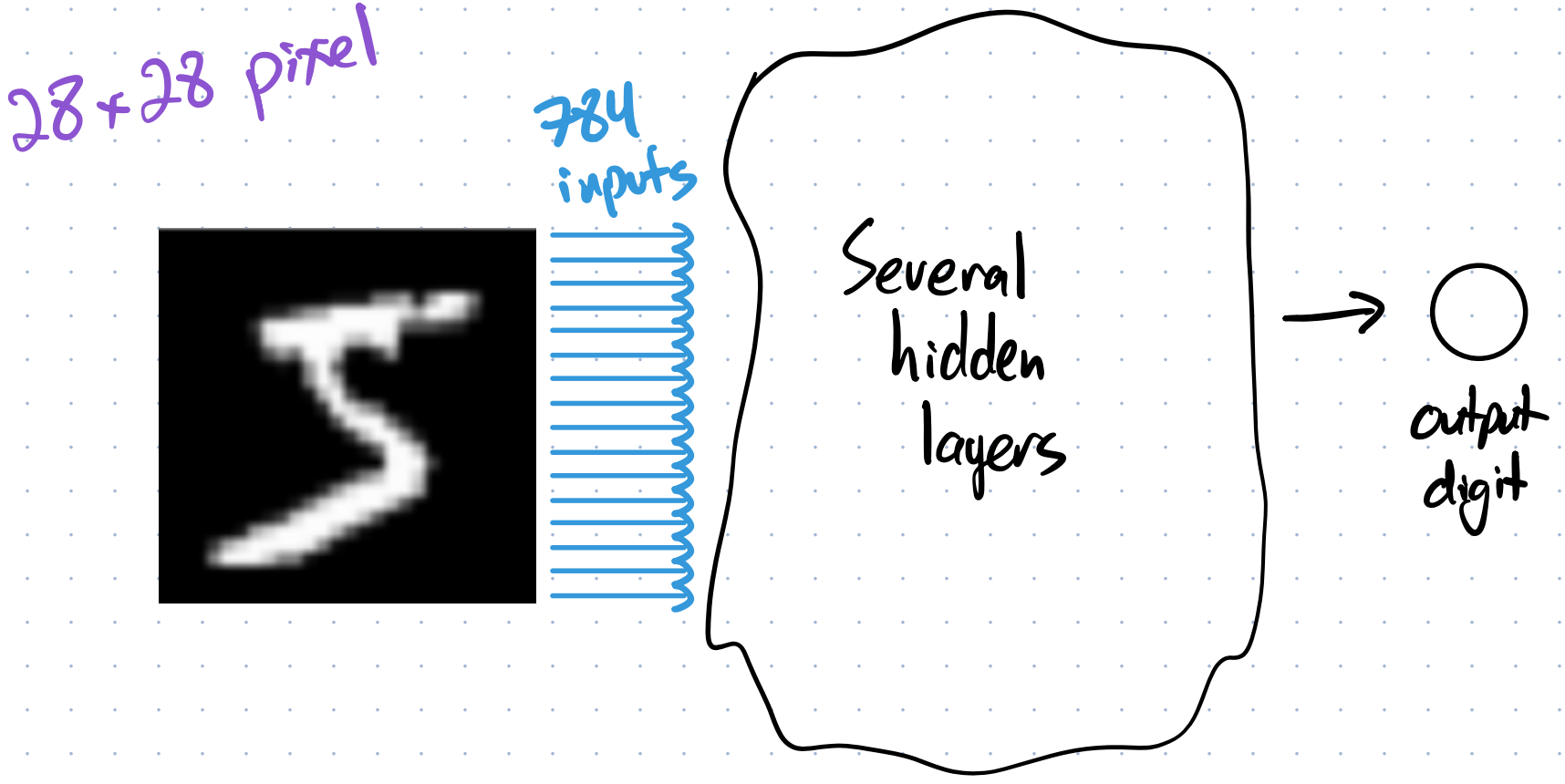


Feed forward Demo:

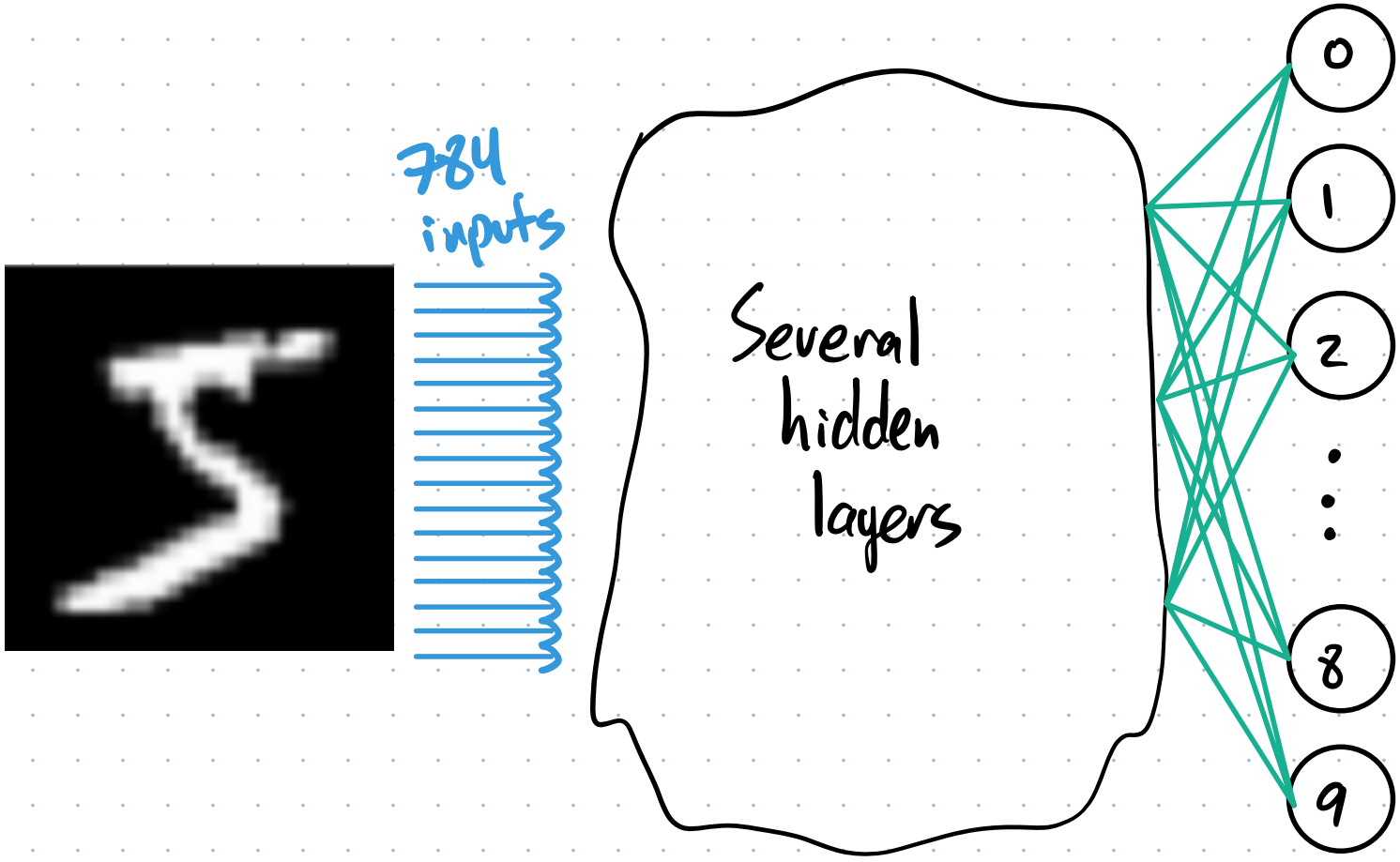
* Different activation functions
* Graphs on each neuron

Now we understand how inputs into the neural network are converted into outputs.

Revisiting MNIST:



Actually it's better to do the output differently.



output neuron i represents how confident we are that the input digit is i.

This is a classification problem. We are deciding which category the input belongs to. More on this later.

Linear Algebra

* You may have noticed that the operations done during the forward pass feel like dot products and matrix multiplications.

* This is why GPUs work so well with NNs.

* Let's explore.

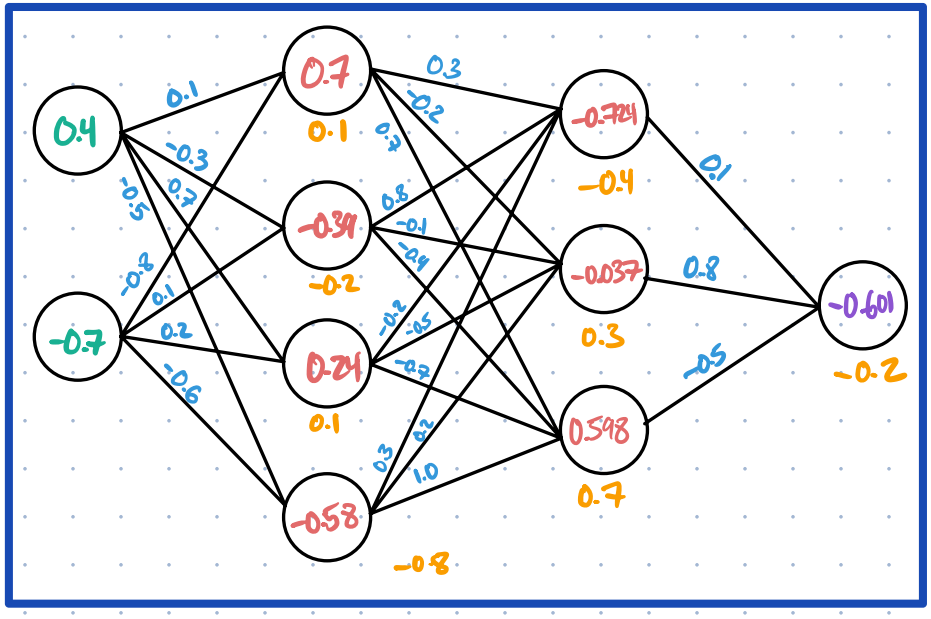
Useful to us: we'll use a python package called "numpy"

Think of each layer as a vector of values.

$$\begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} \quad \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix} \quad \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix} \quad \begin{bmatrix} -0.601 \end{bmatrix}$$

How is each layer computed from the previous one?
Make matrices for the weights between each layer and vectors for the biases.

$$M_1 = \begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \qquad b_1 = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix}$$



Rearrange!

(ignoring activation functions!)

$$\begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} \qquad \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \qquad b_1 = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix}$$