

Scientific Computing

Wednesday, April 1

Announcements

- * HW 5 assigned, due Friday, April 10, 11:59pm
- * No class or office hours Friday or Monday (Easter break)

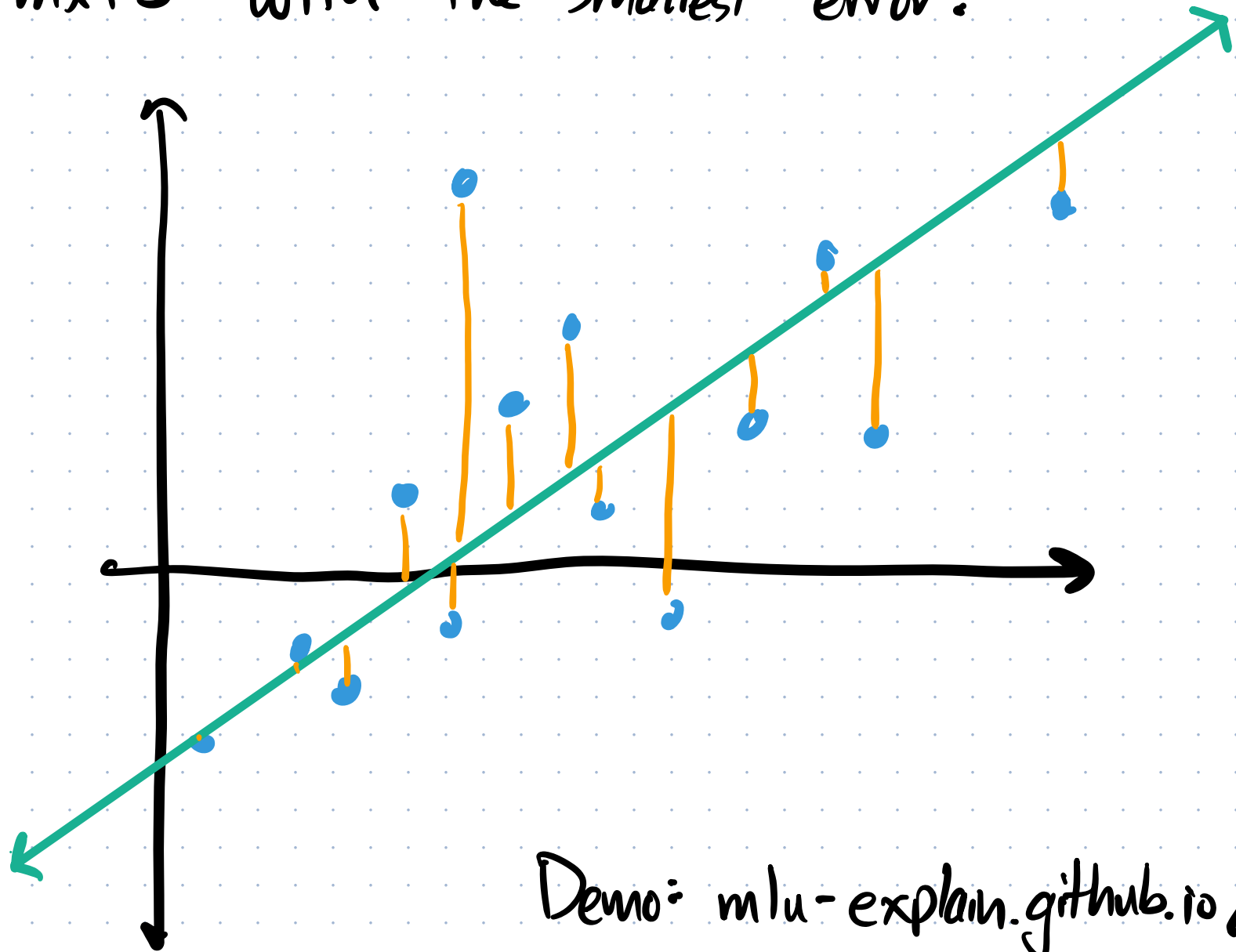
Office Hours:

Mon, 9:30-10:30

Fri, 2:00-3:00

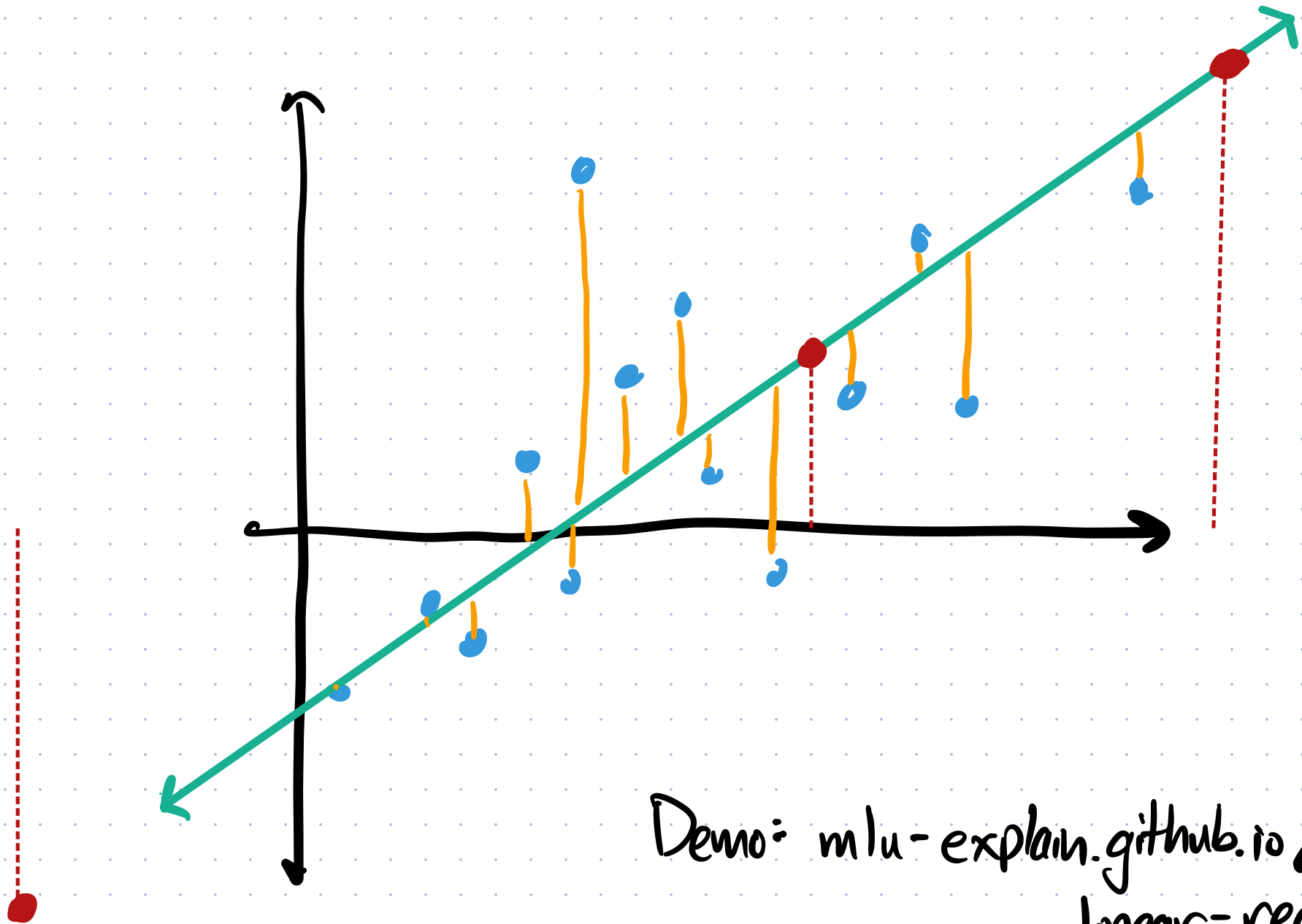
Cudahy 307

Problem: What values of m and b make the line $y = mx + b$ with the smallest error?



Demo: mlu-explain.github.io/linear-regression

Purpose: Estimate the output at new inputs.

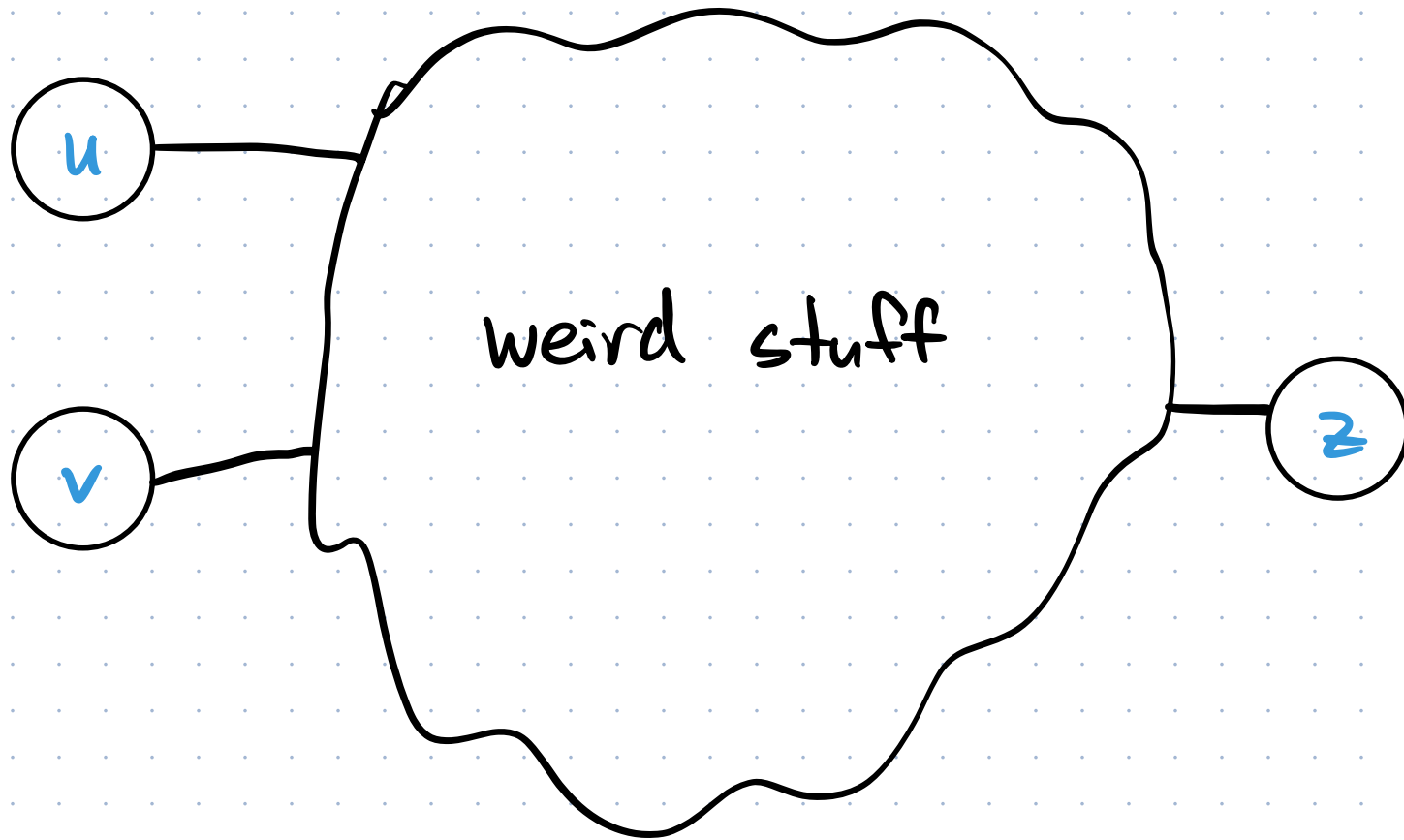


Demo: mlu-explain.github.io /
linear-regression

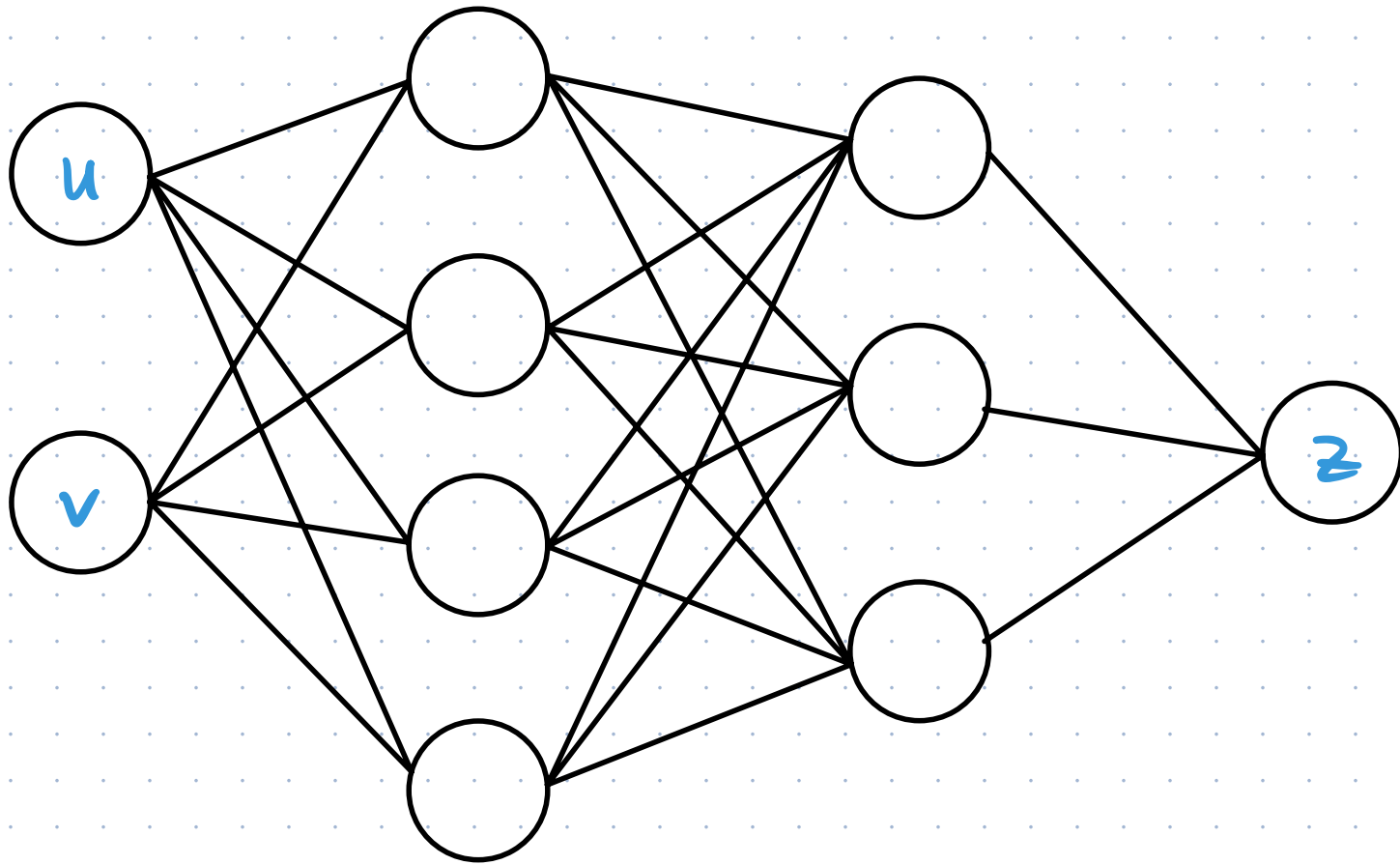
What is a neural network?

A fancy function defined in the following way.

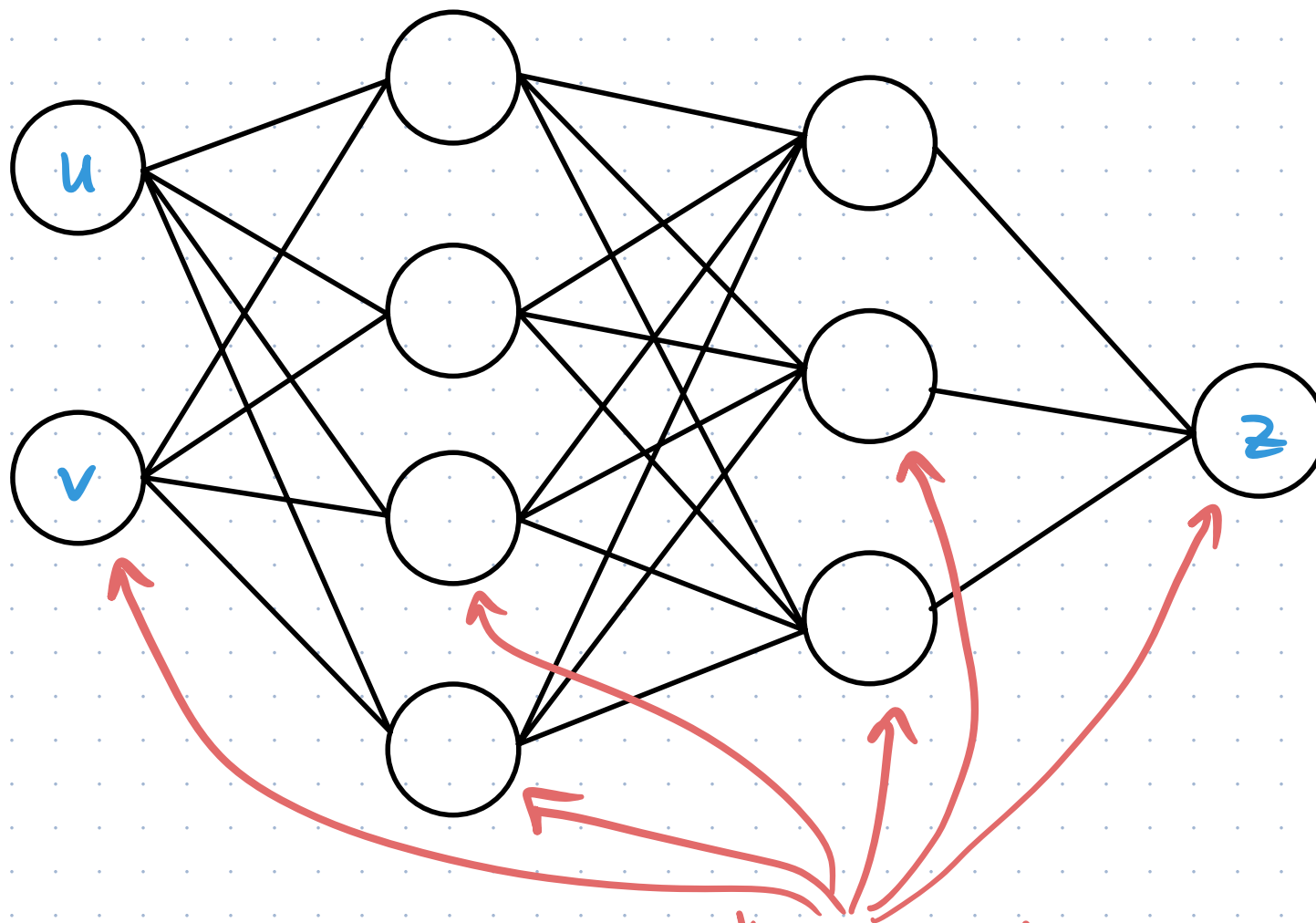
Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



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

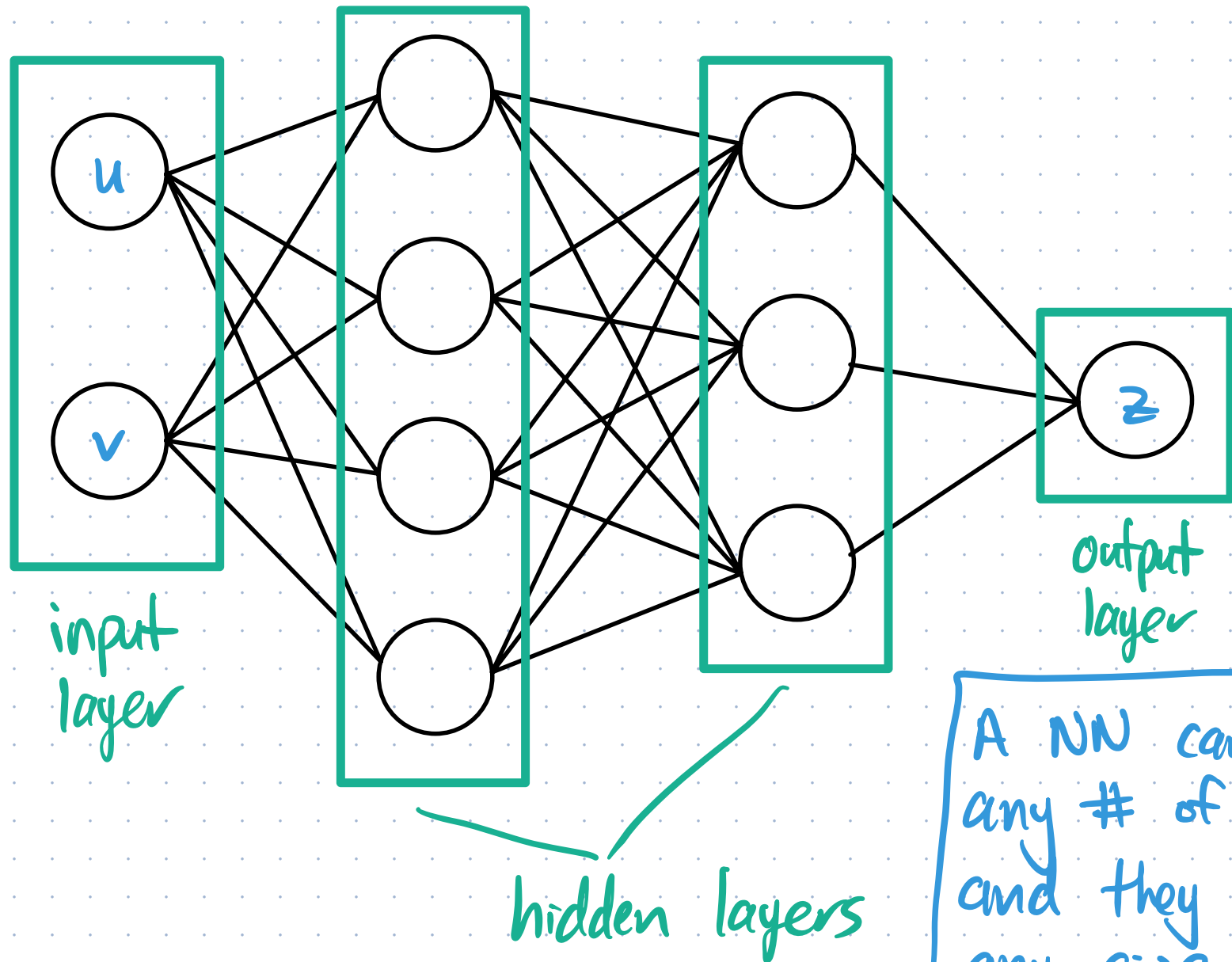


Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



each circle is a "neuron"

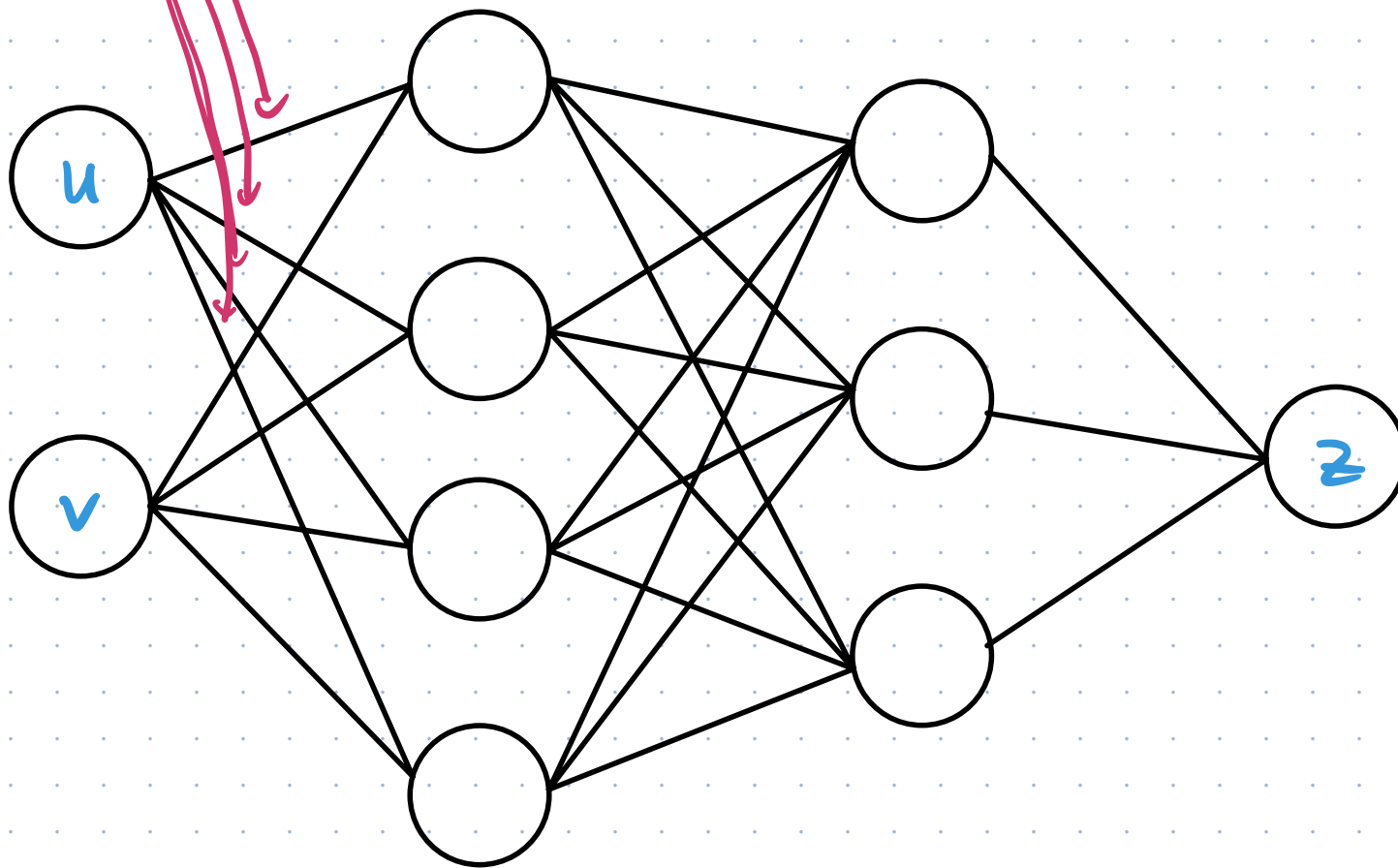
Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



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

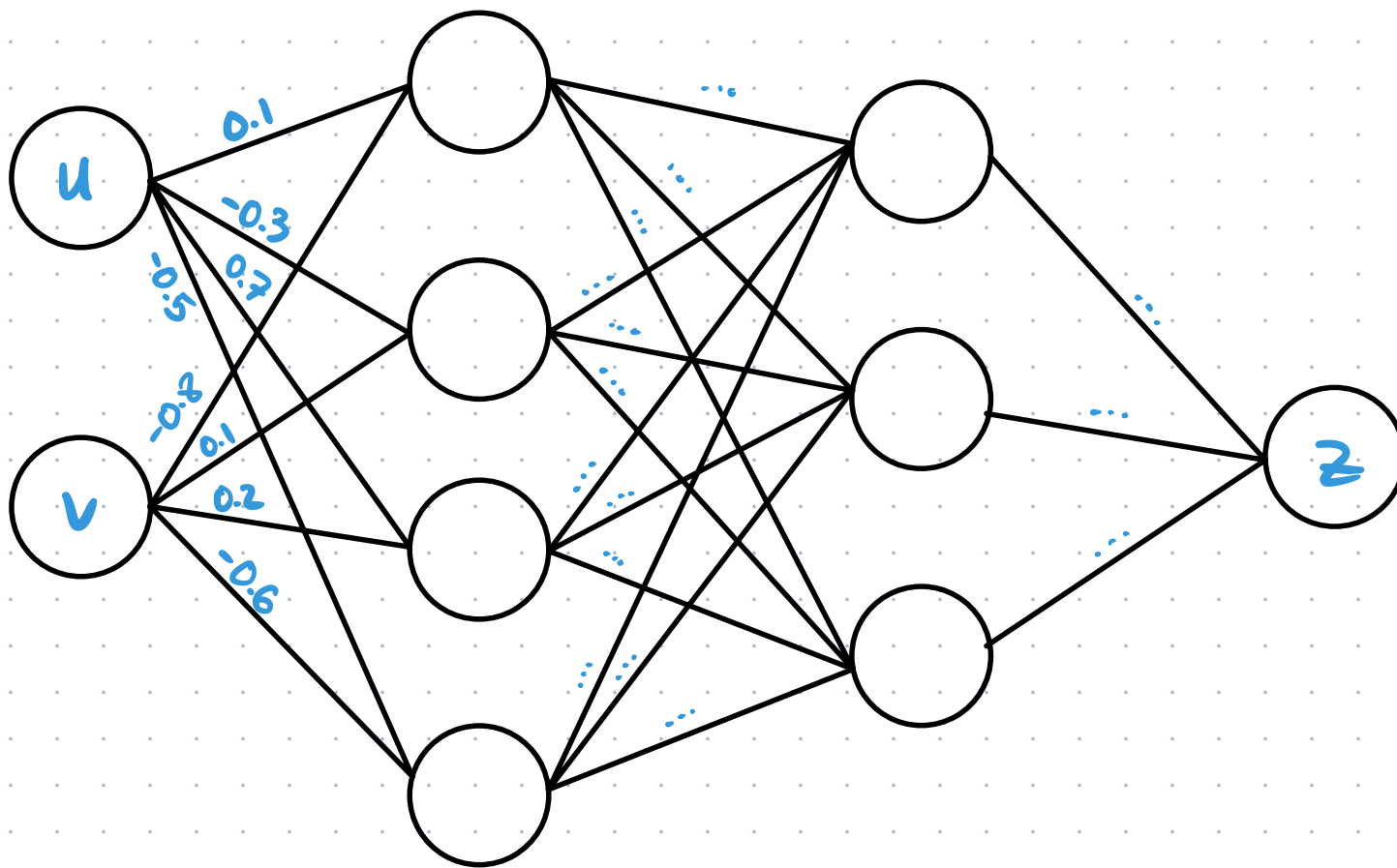
Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$

real #s (decimals)



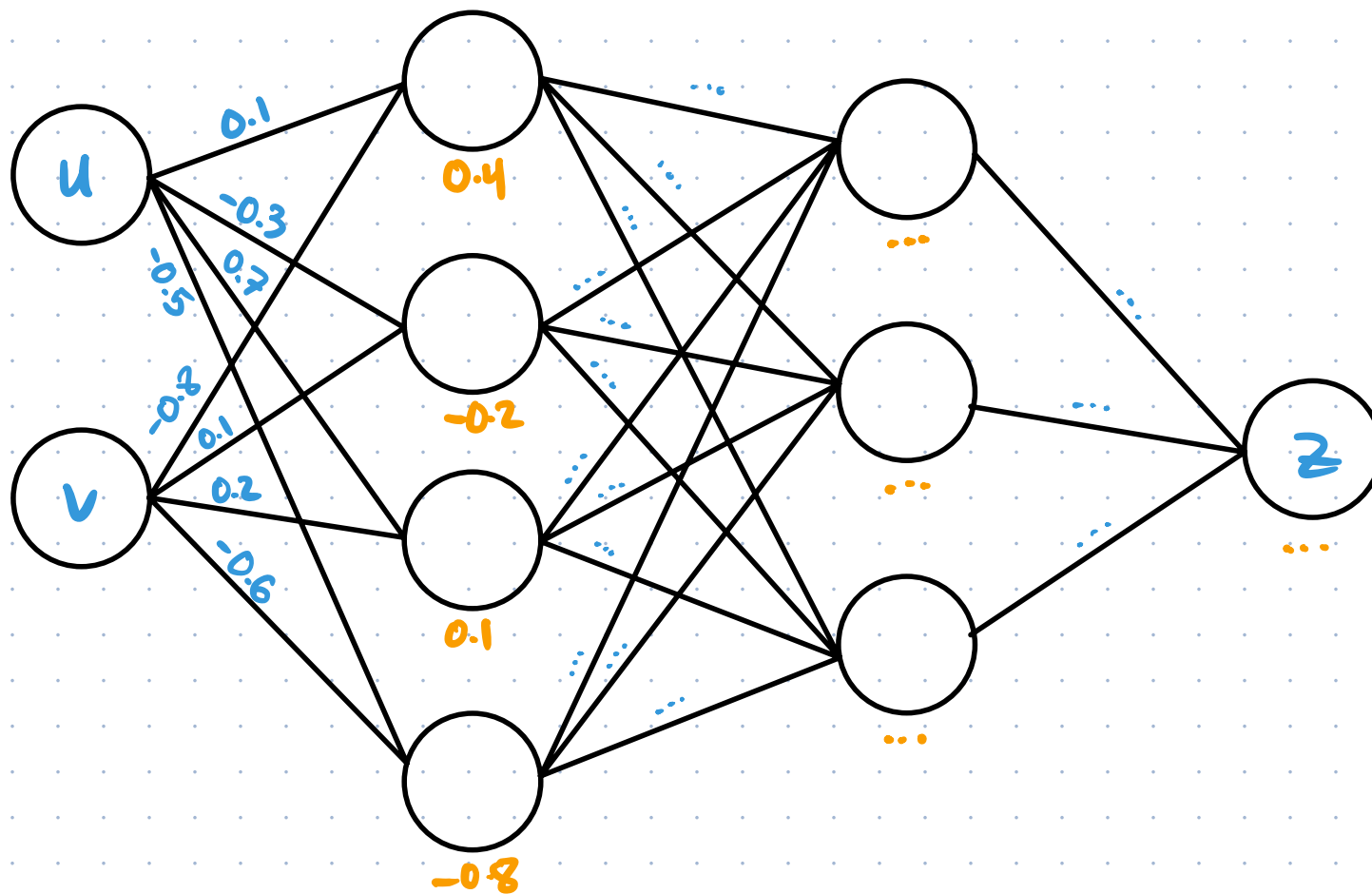
Every edge between neurons has a real # attached to it called its weight

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



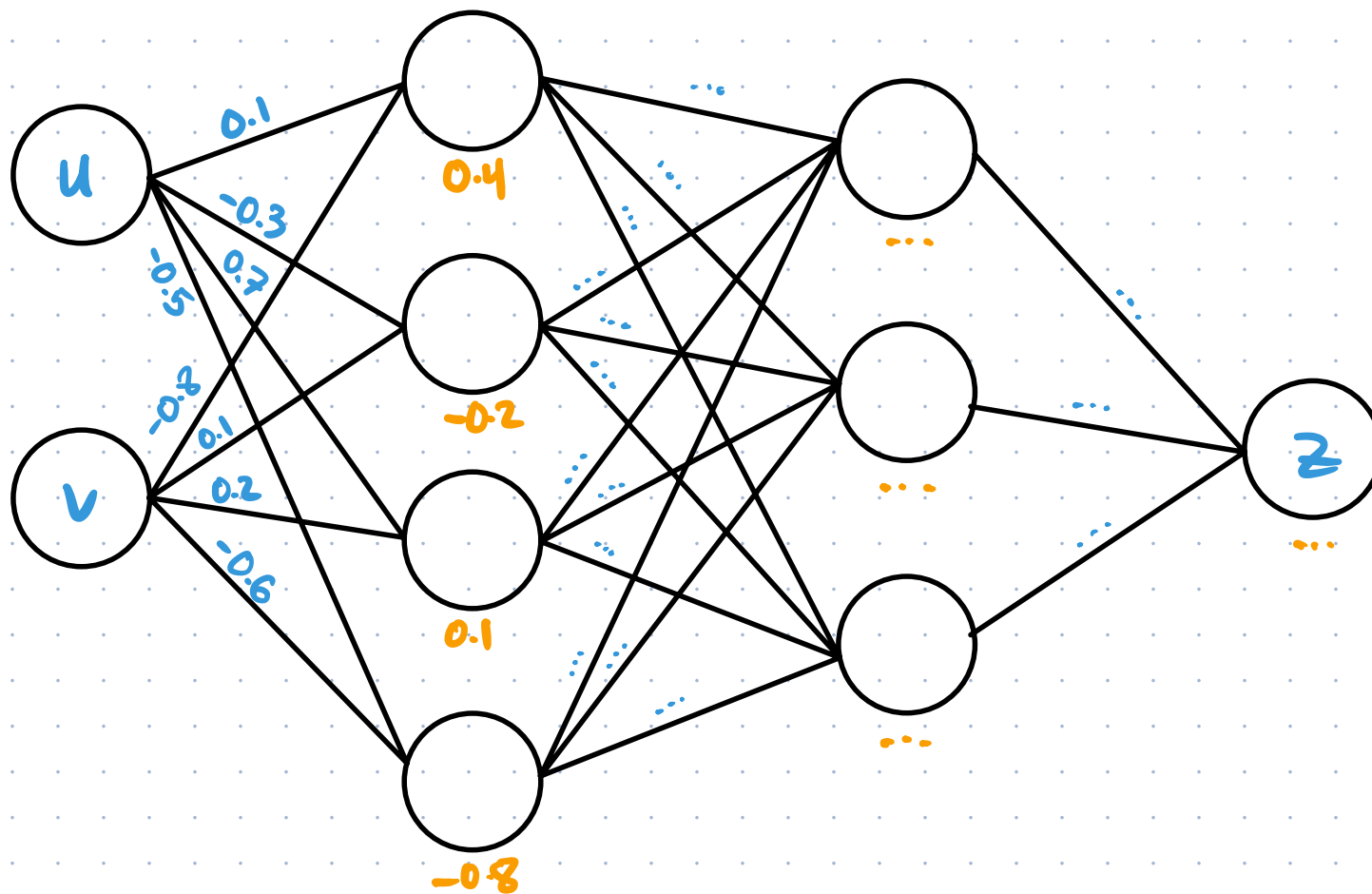
I'm using #'s to 1 decimal place for simplicity, but typically they are full floating point #'s.

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
 $f(u, v) = z$

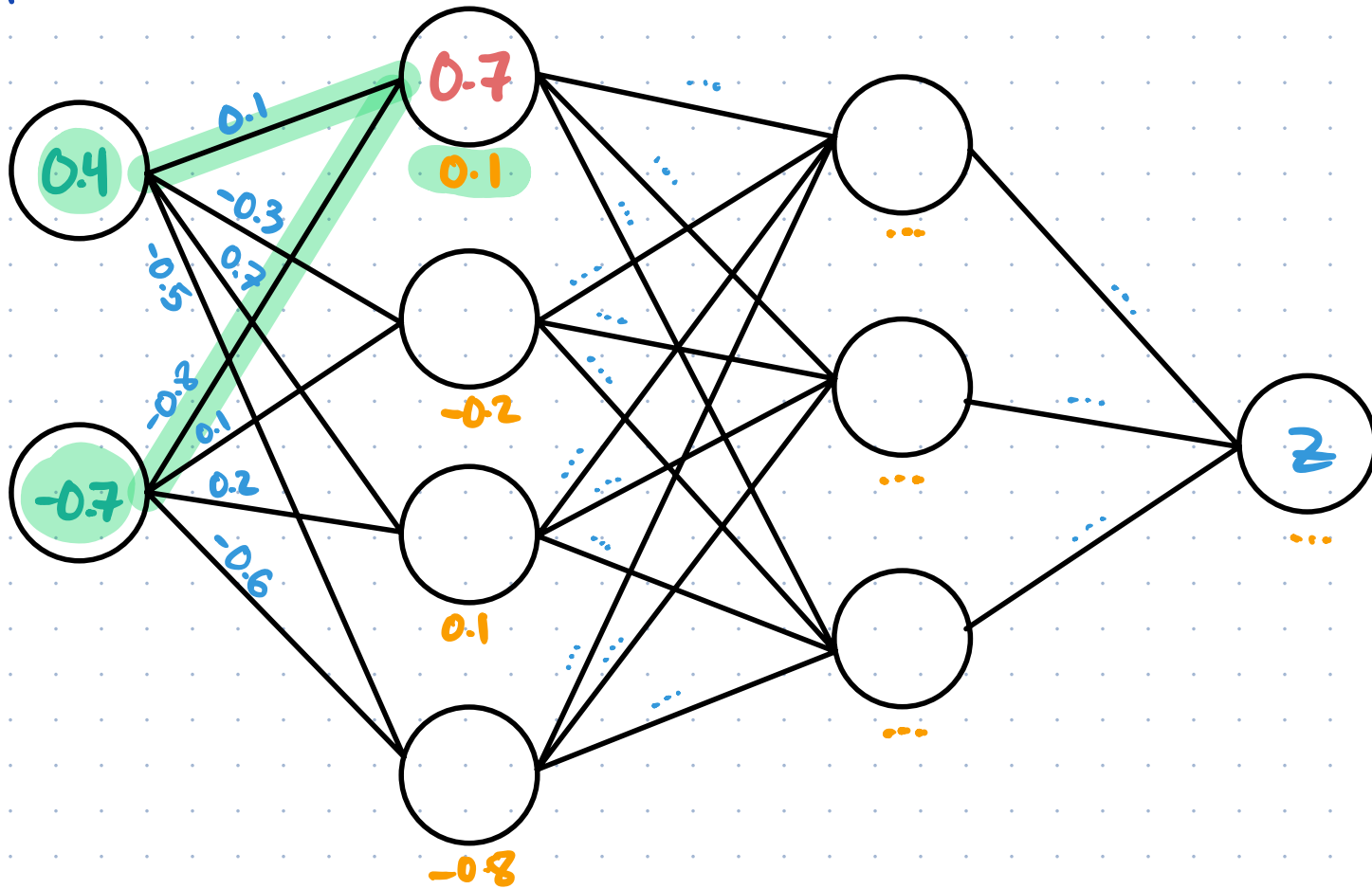


The #s u and v get "fed forward" to the neurons in the next layer.

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

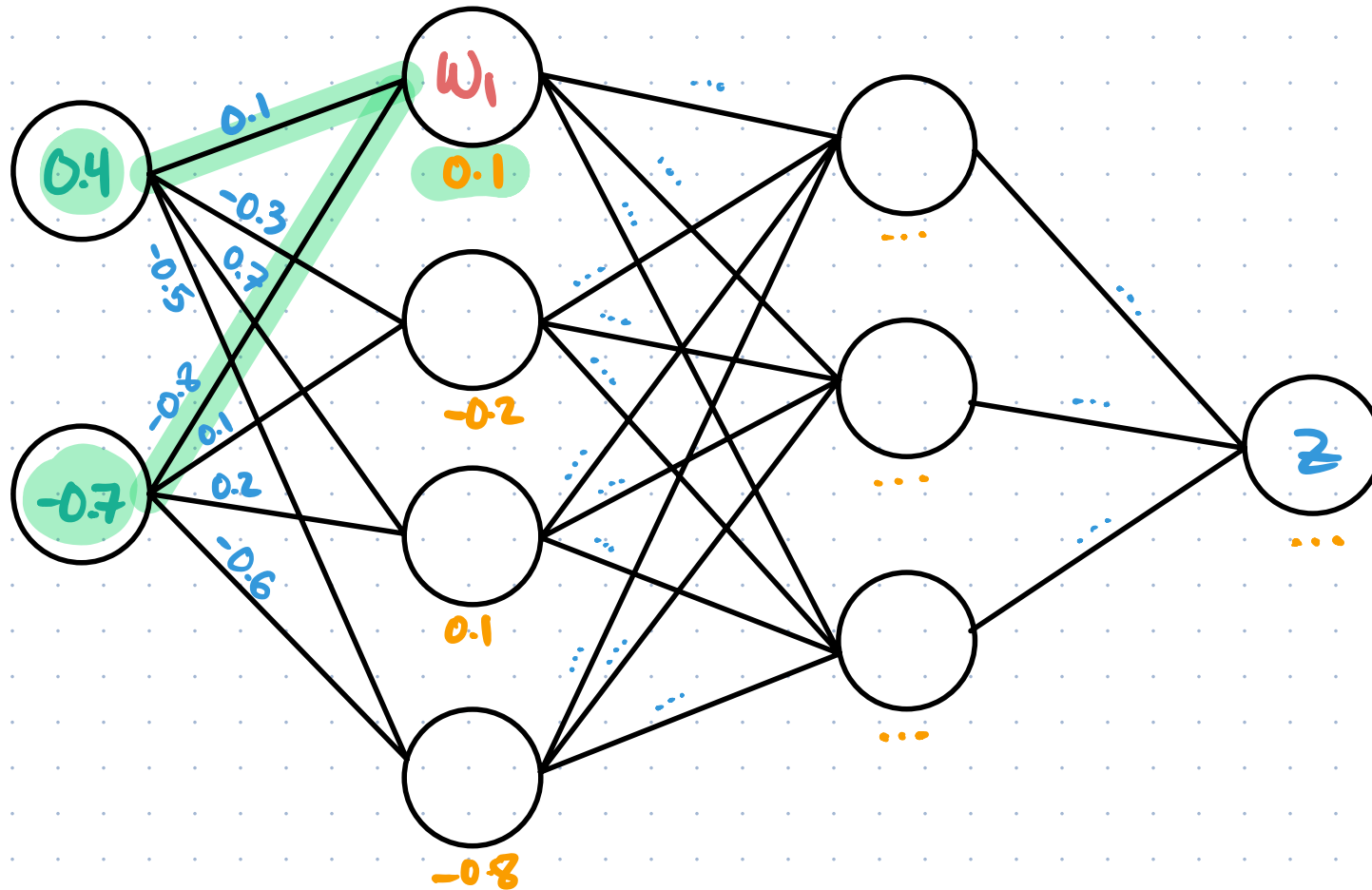
$$f(0.4, -0.7) = ?$$

$$f(u, v) = z$$



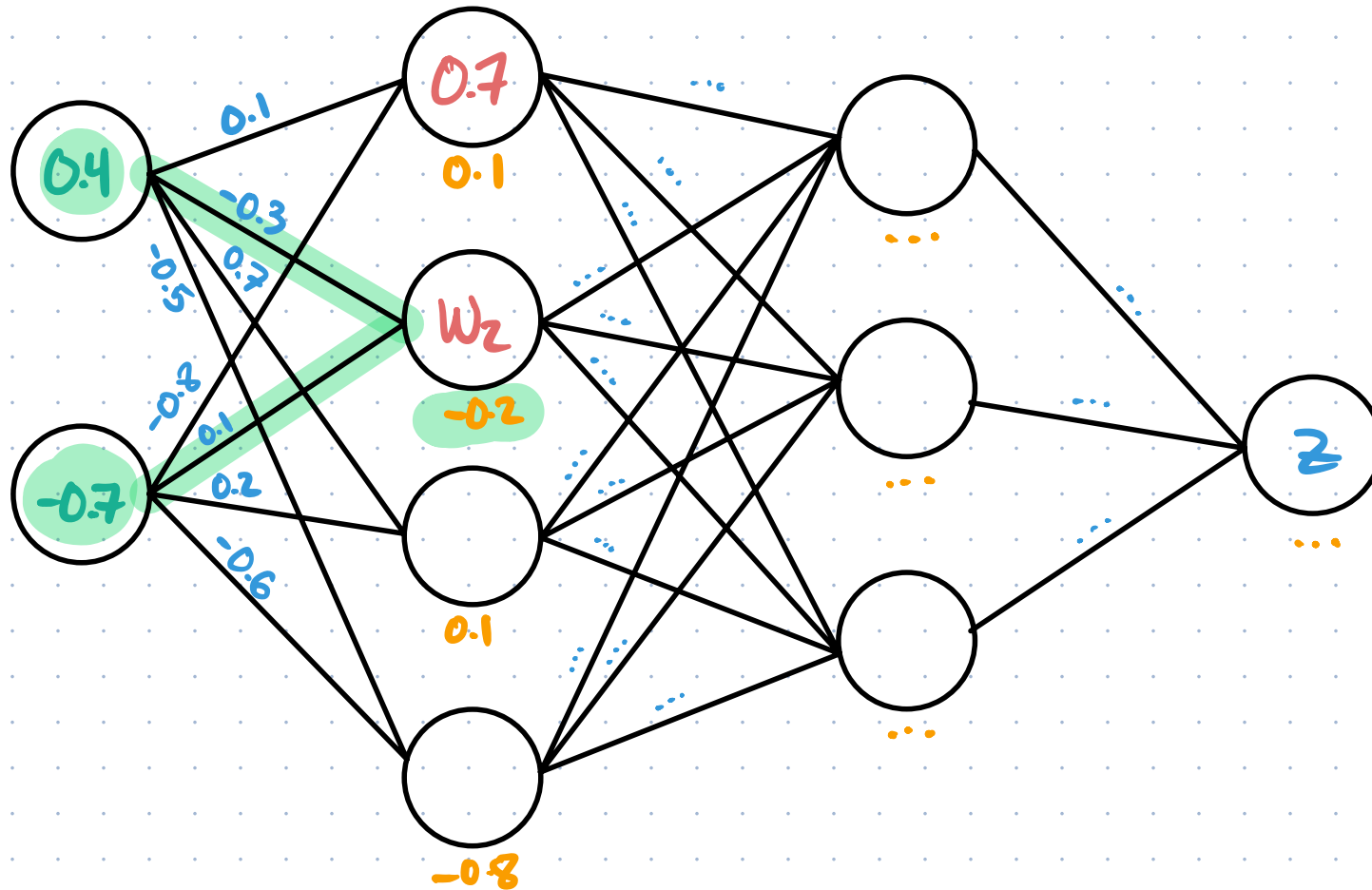
To calculate the value for a neuron, we multiply the value of each connected neuron from the last layer by the weight of the connection, add these all up, and add the bias

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



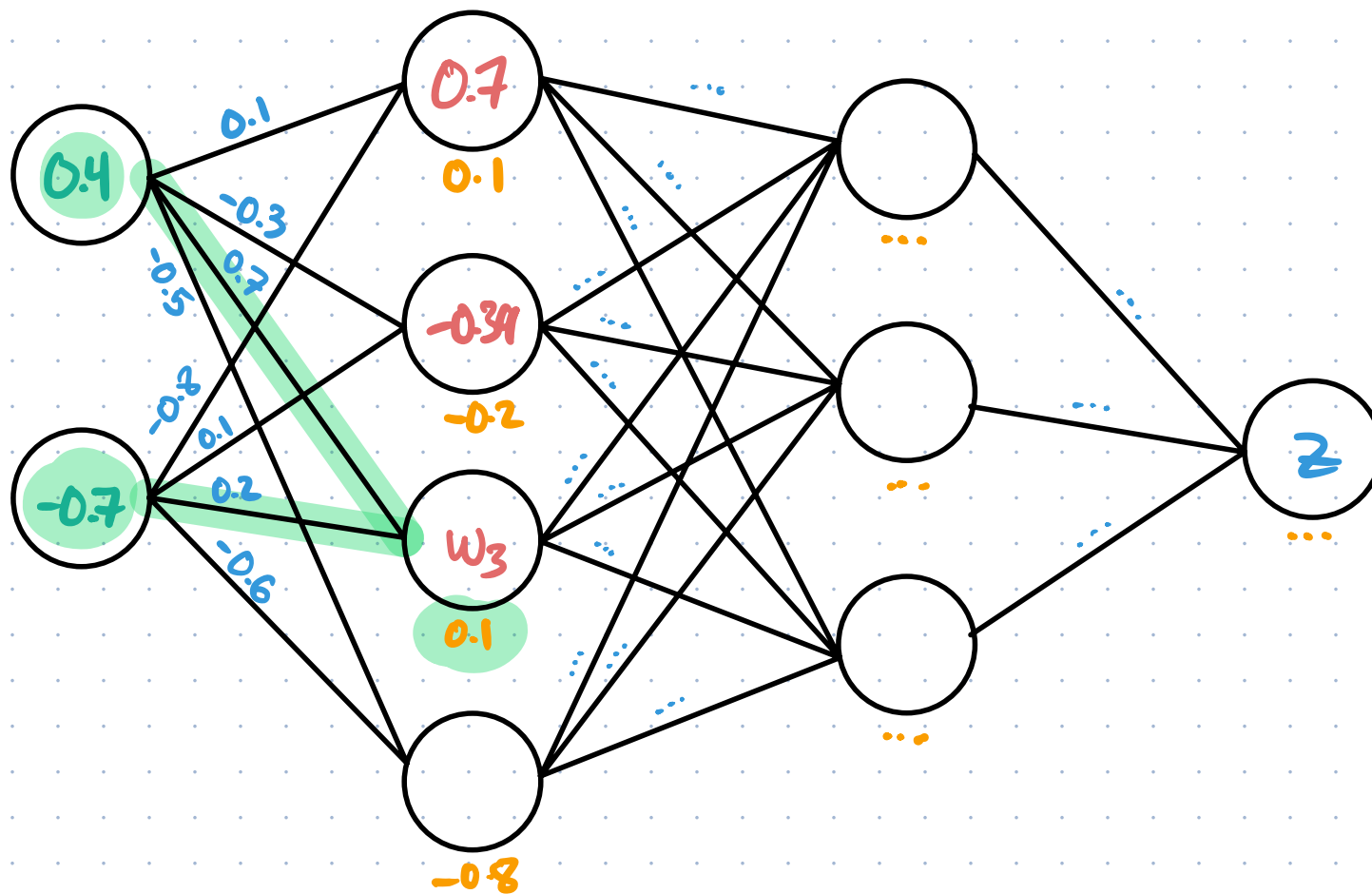
$$w_1 = (0.4)(0.1) + (-0.7)(-0.8) + 0.1$$
$$= 0.7$$

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



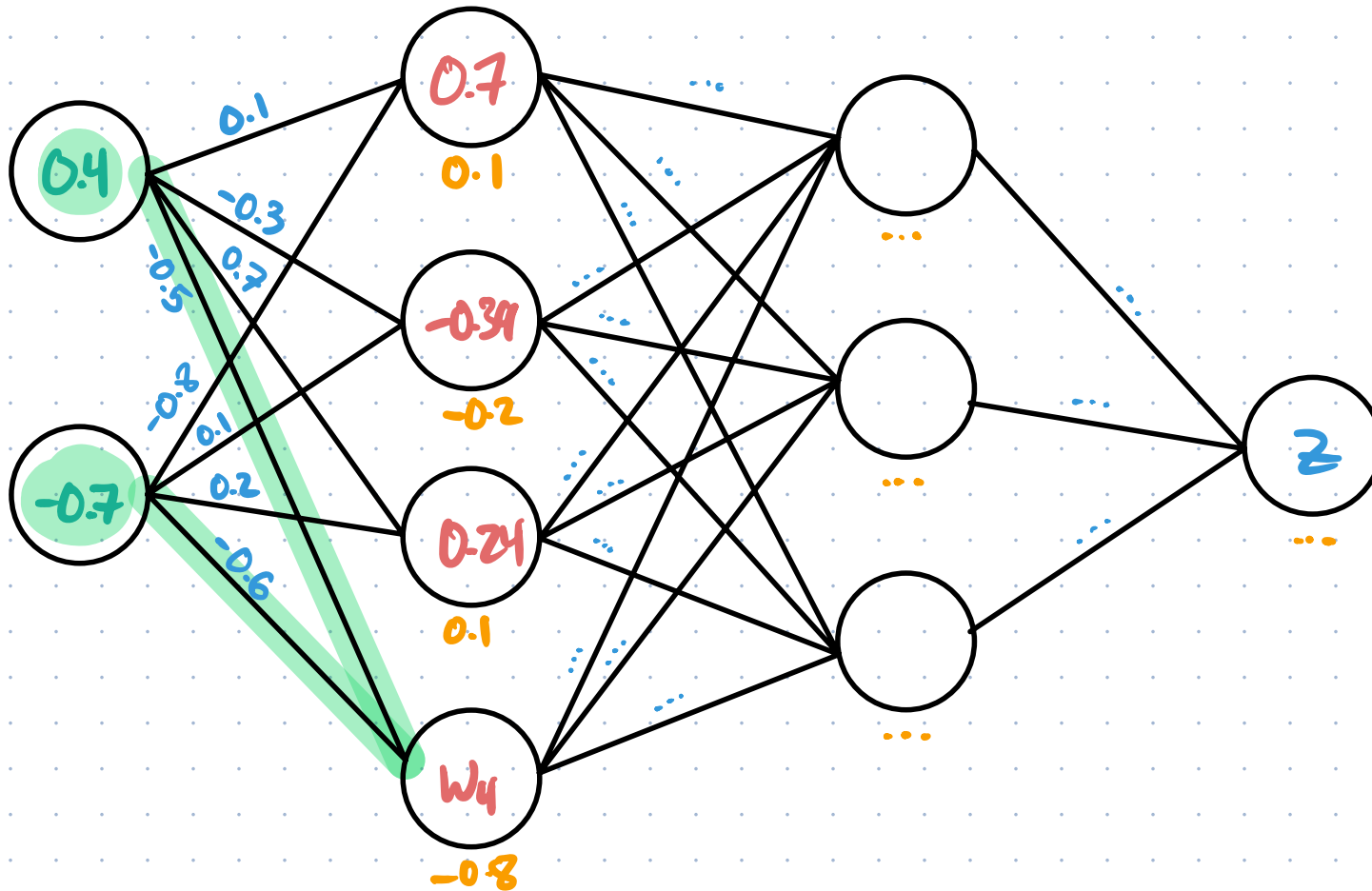
$$\begin{aligned} W_2 &= (0.4)(-0.3) + (-0.7)(0.1) + -0.2 \\ &= -0.39 \end{aligned}$$

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



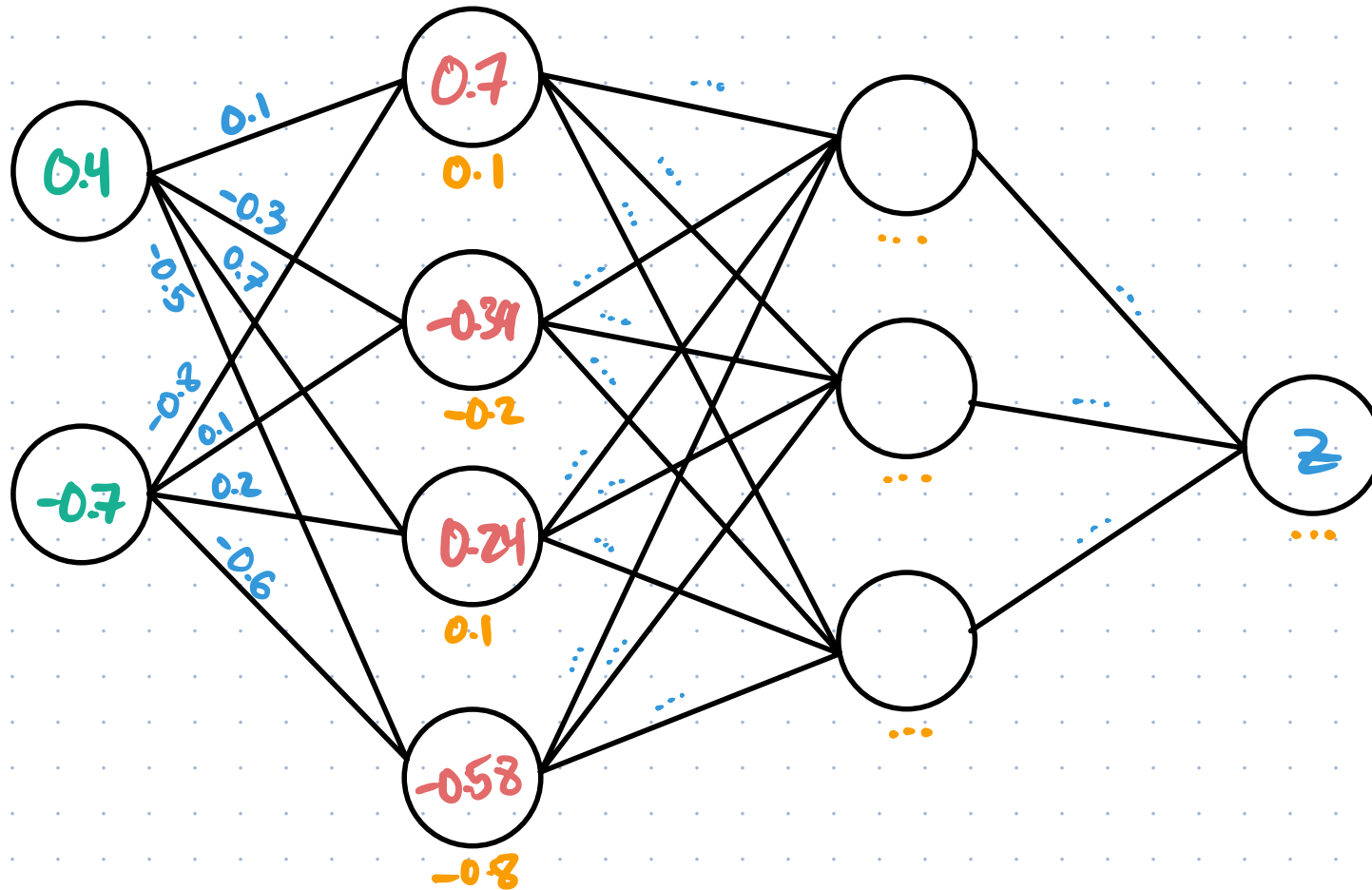
$$\begin{aligned} W_3 &= (0.4)(0.7) + (-0.7)(-0.2) + 0.1 \\ &= 0.24 \end{aligned}$$

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



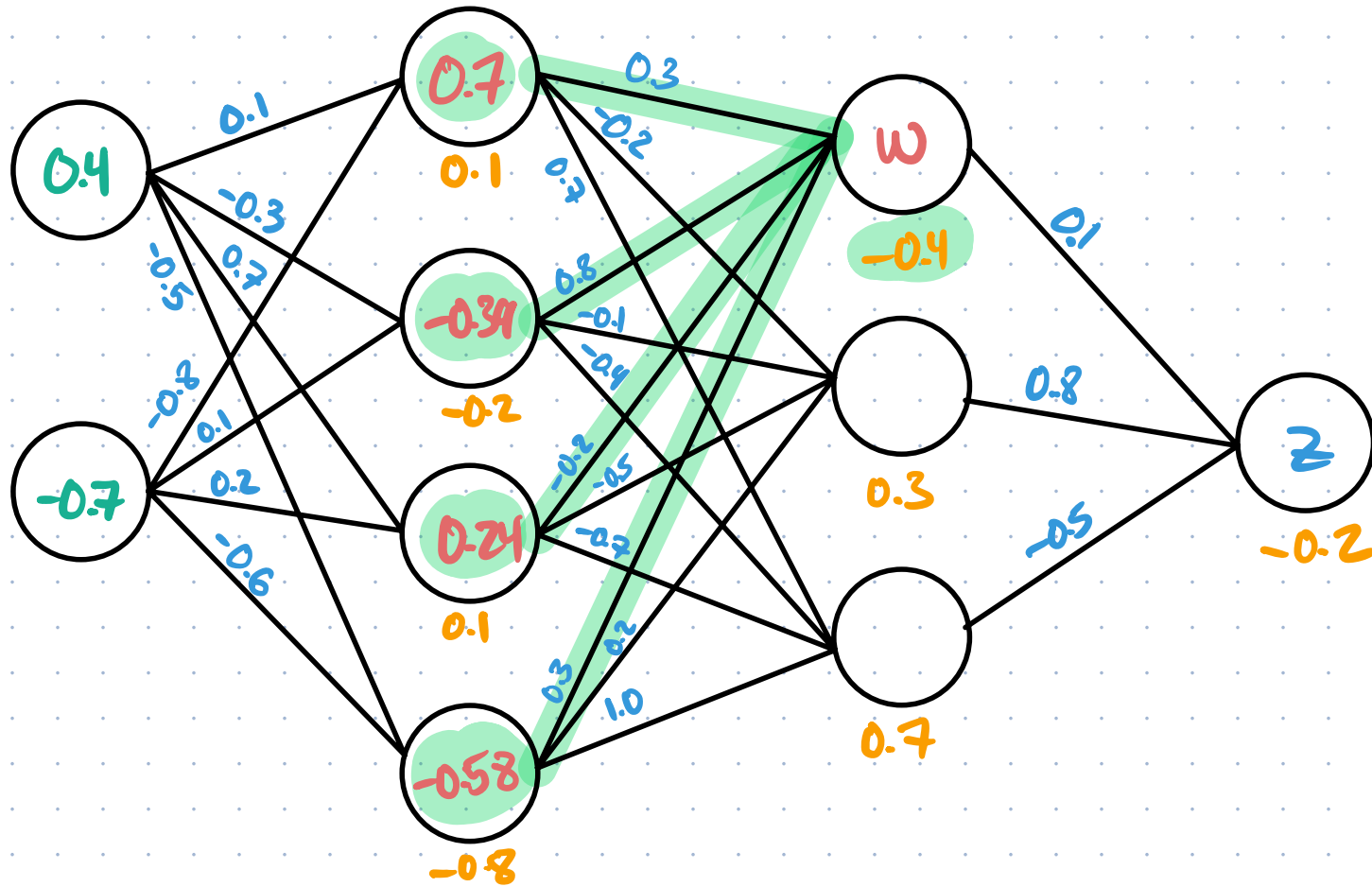
$$\begin{aligned} W_4 &= (0.4)(-0.5) + (-0.7)(-0.6) + -0.8 \\ &= -0.58 \end{aligned}$$

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.

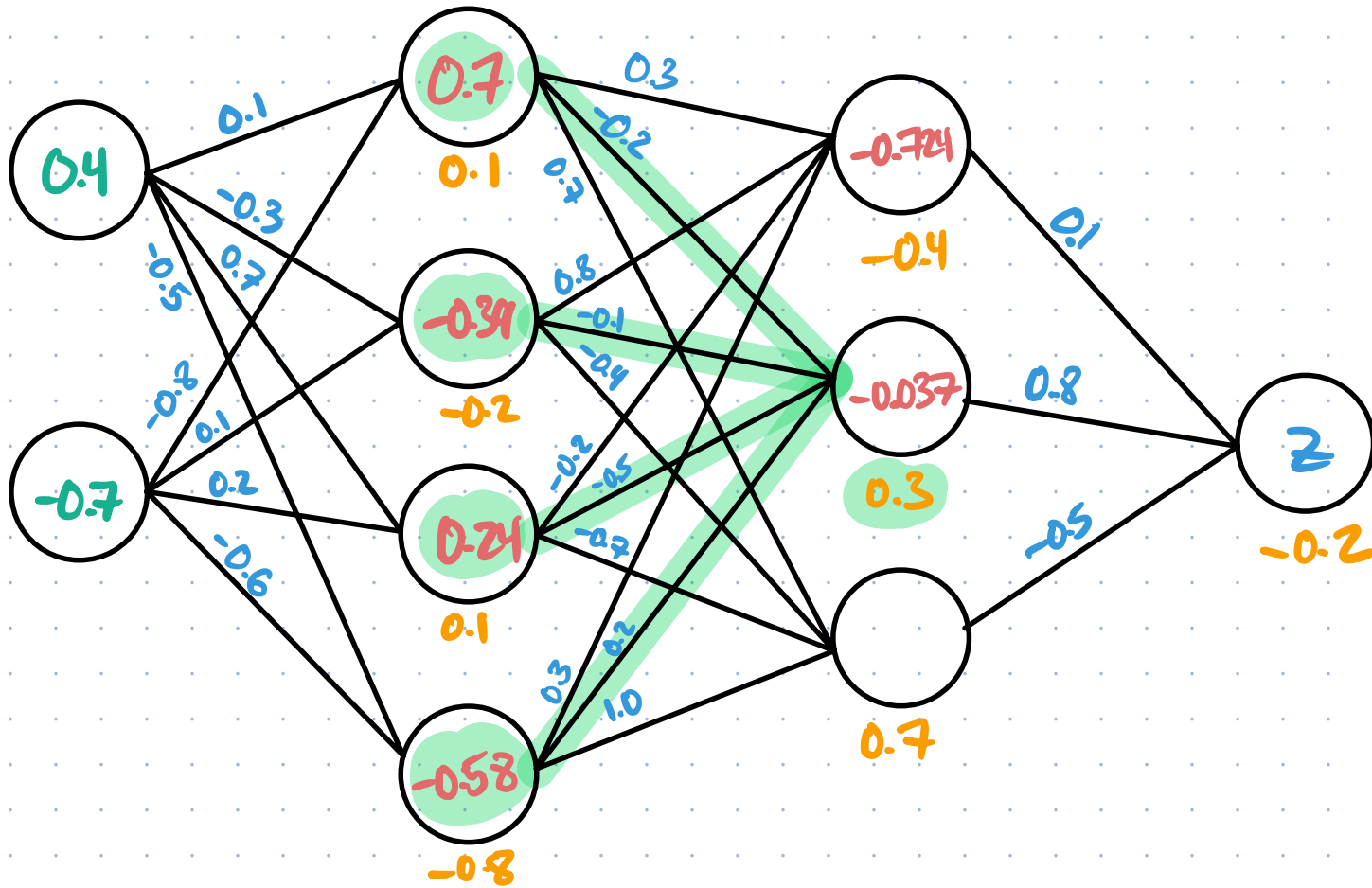
Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



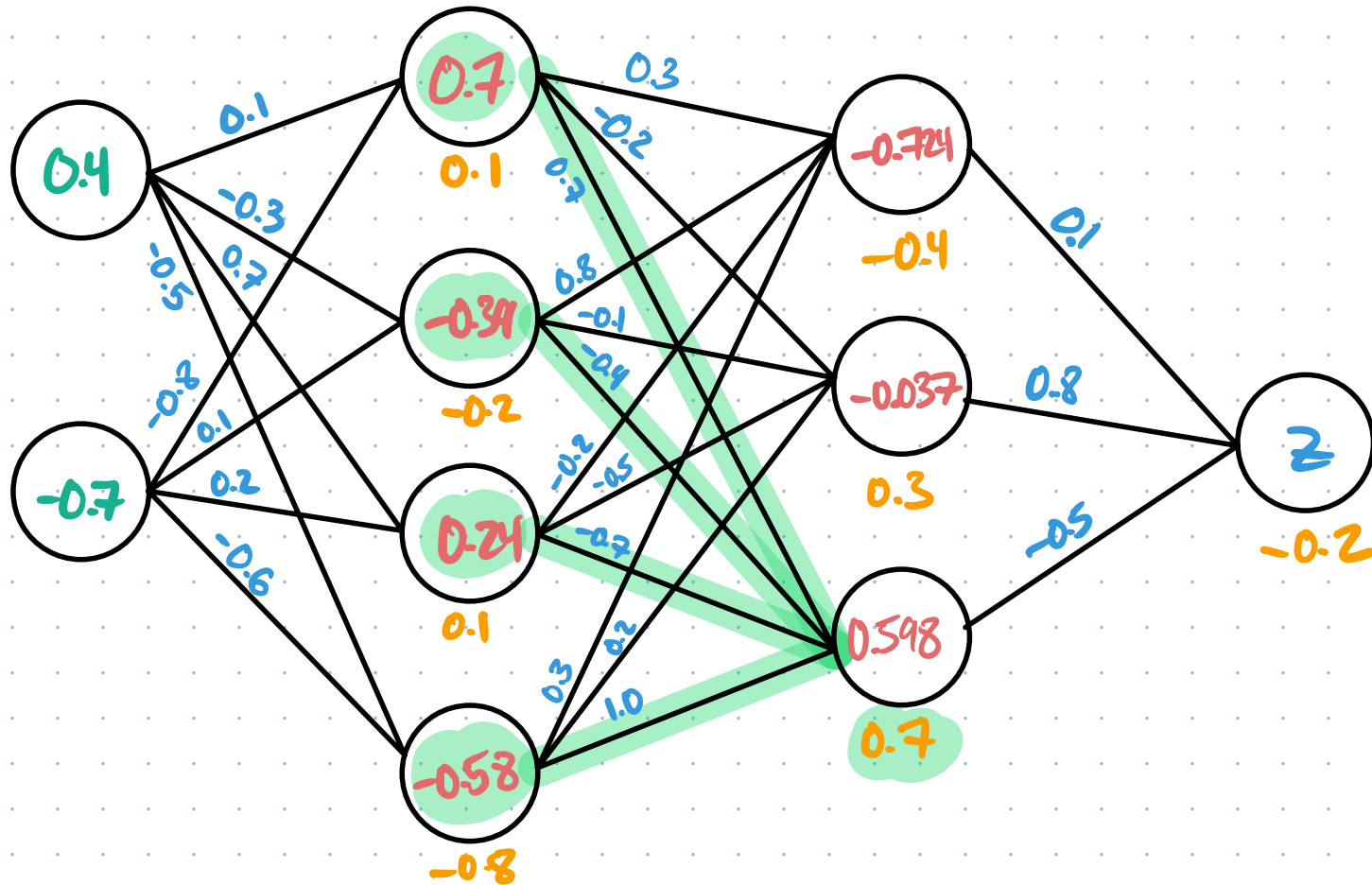
$$W = (0.7)(0.3) + (-0.39)(0.8) + (0.24)(-0.2) + (-0.58)(0.3) + -0.4$$

$$= -0.724$$

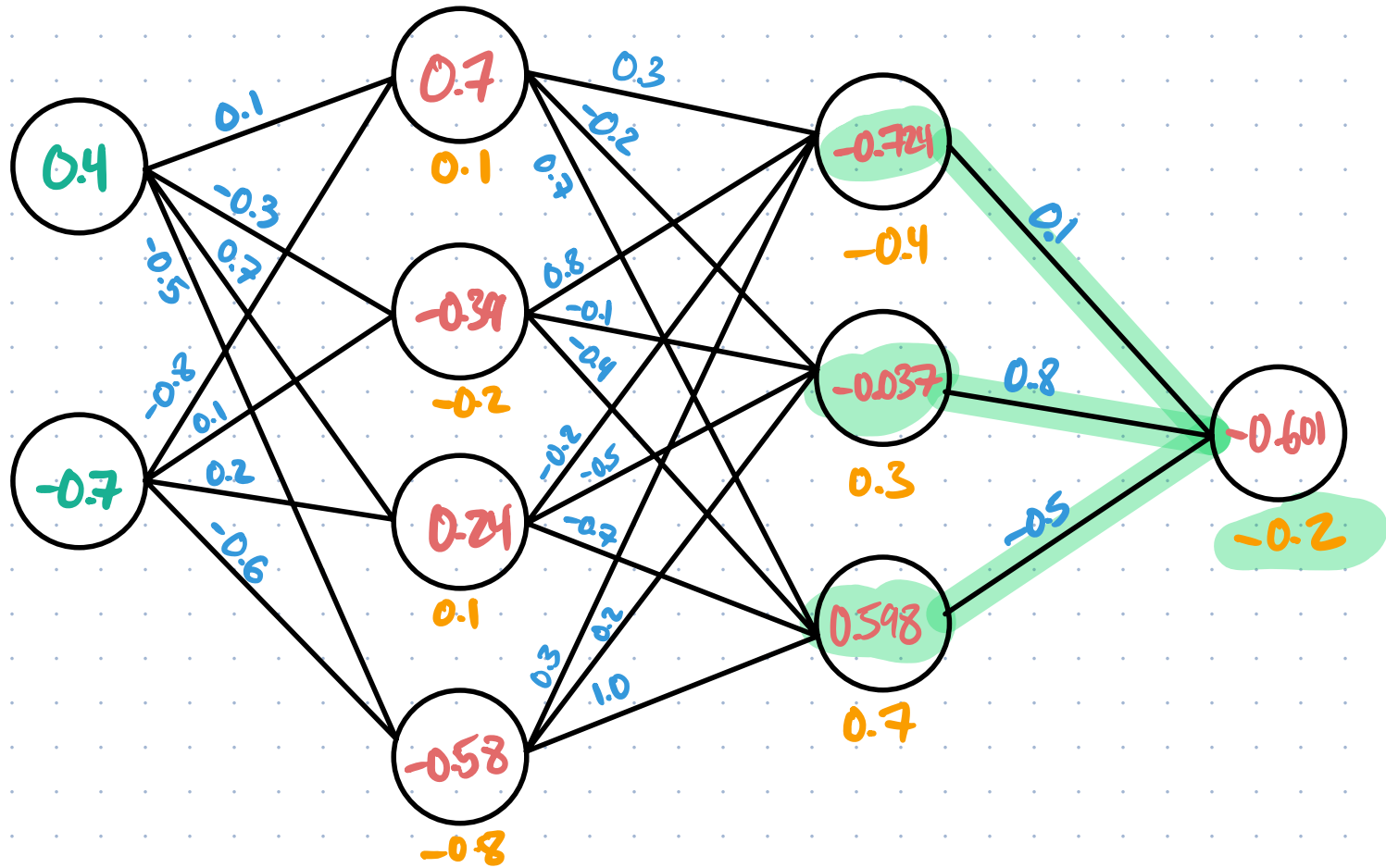
Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



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

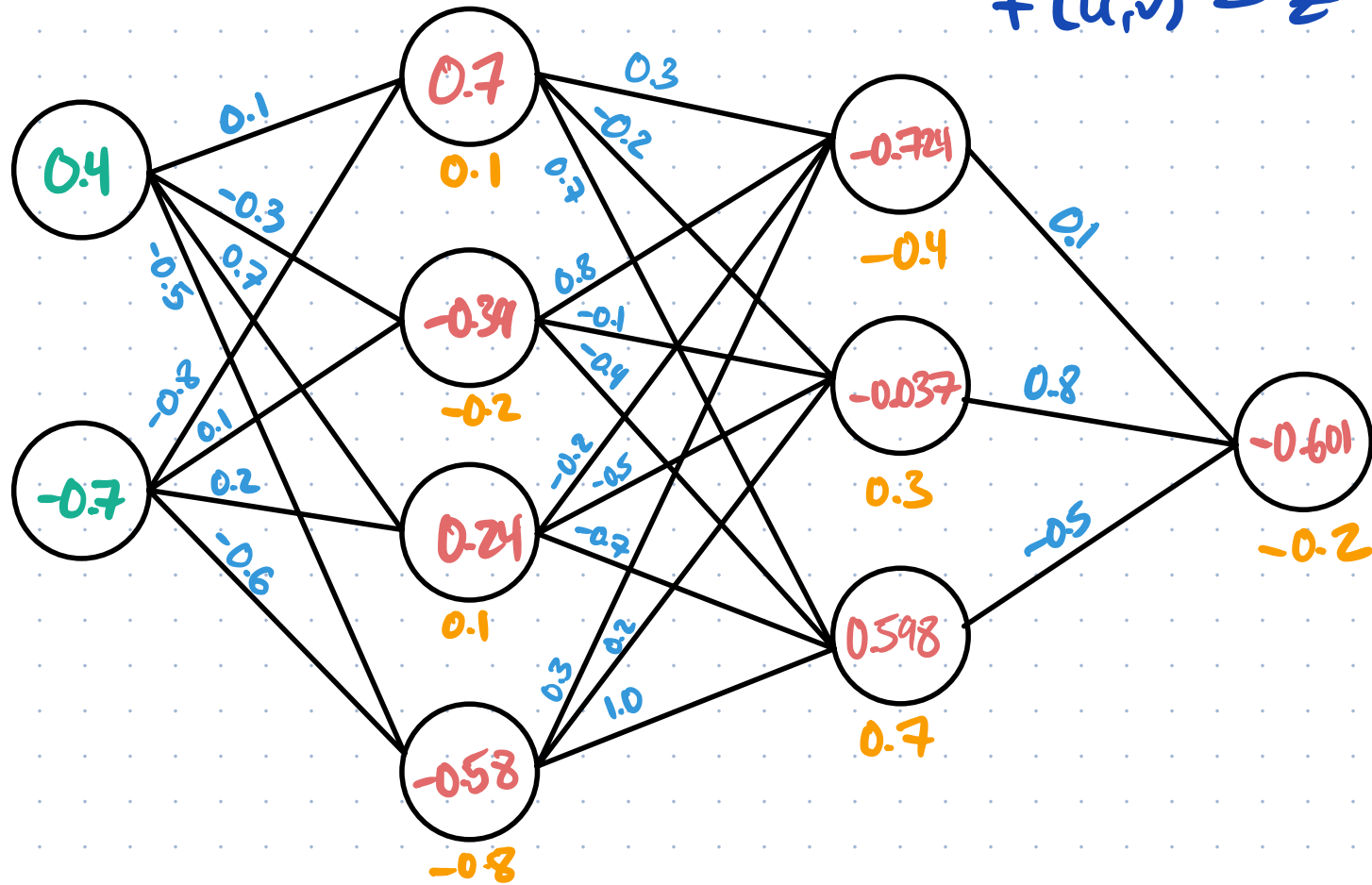


Example: A NN that takes 2 inputs and has 1 output
 $f(u, v) = z$



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

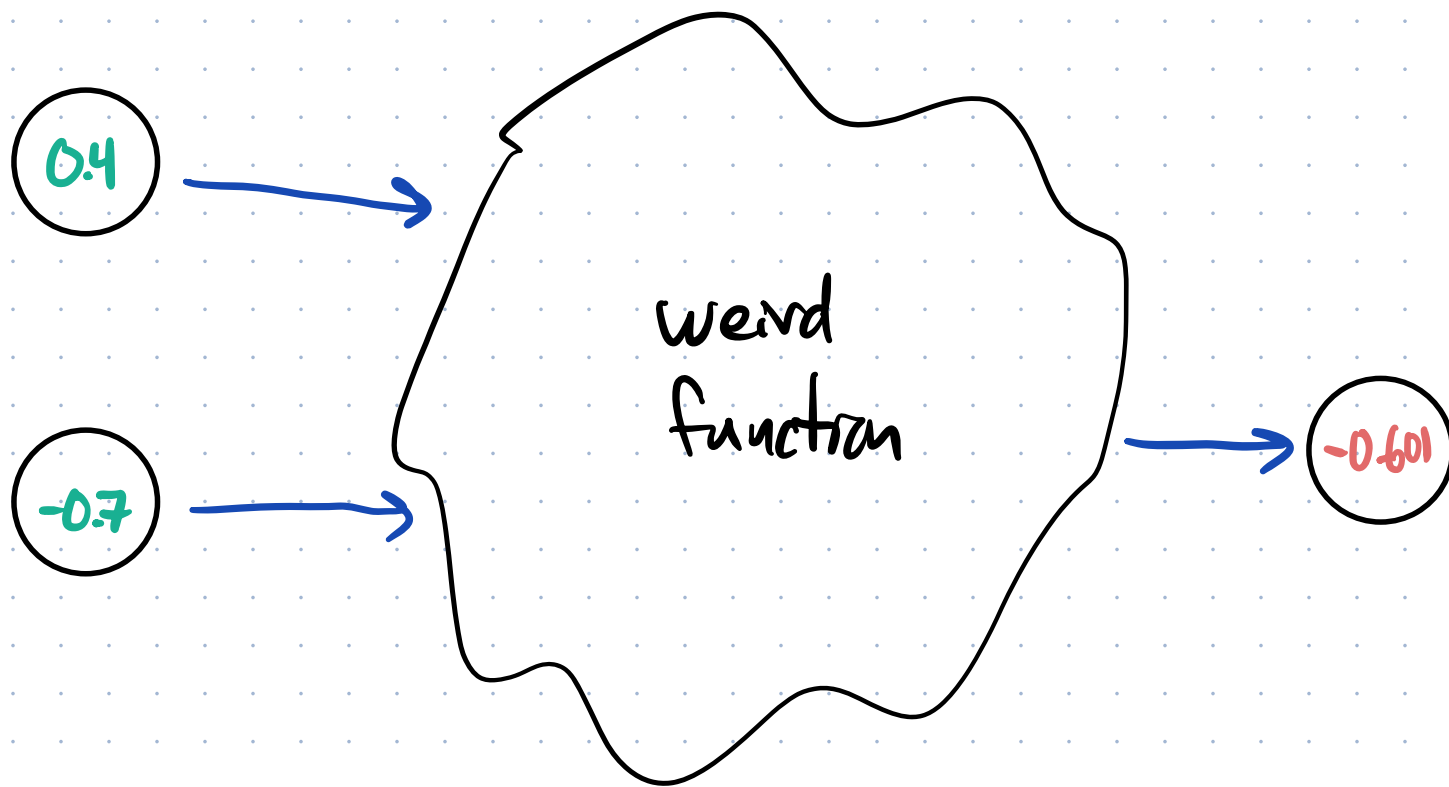
Example: A NN that takes 2 inputs and has 1 output
 $f(u,v) = z$



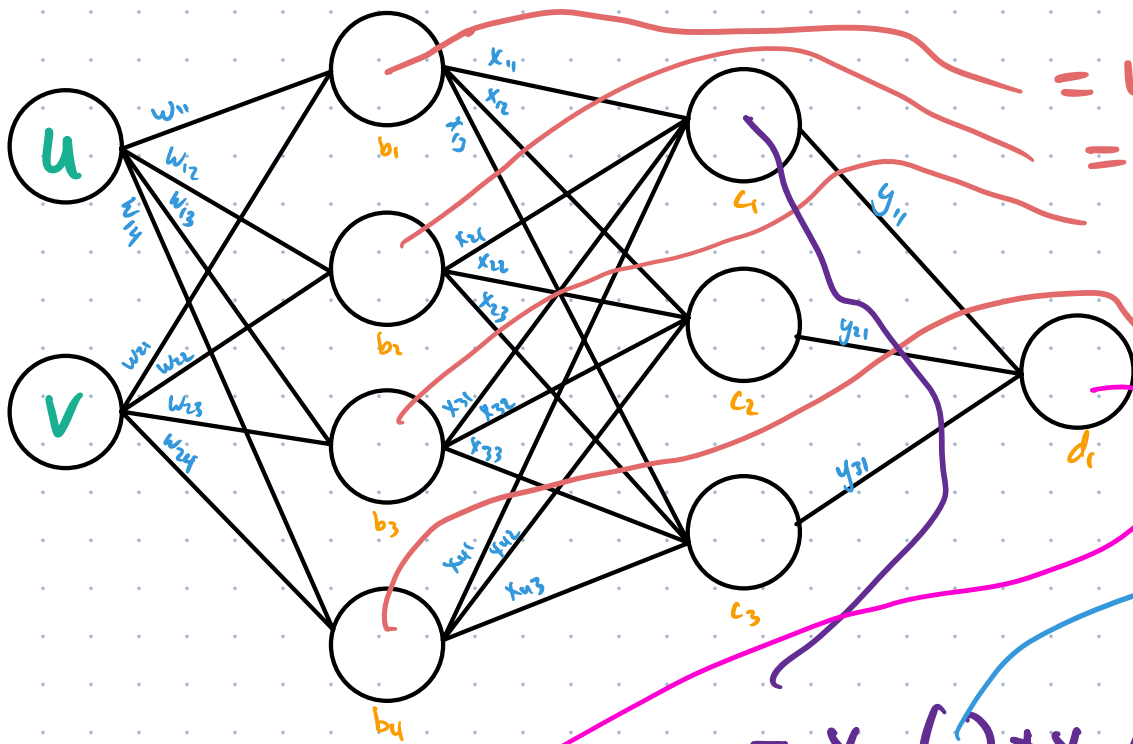
So, for this particular neural network, with these weights and biases, the inputs (0.4, -0.7) produce output -0.601

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

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



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

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

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

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

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

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

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

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

$$w_{11}x_{11}y_{11} + w_{11}x_{12}y_{21} + w_{11}x_{13}y_{31} + w_{12}x_{21}y_{11} + w_{12}x_{22}y_{21} + w_{12}x_{23}y_{31} + w_{13}x_{31}y_{11} + w_{13}x_{32}y_{21} + w_{13}x_{33}y_{31} + w_{14}x_{41}y_{11} + w_{14}x_{42}y_{21} + w_{14}x_{43}y_{31} \cdot u$$

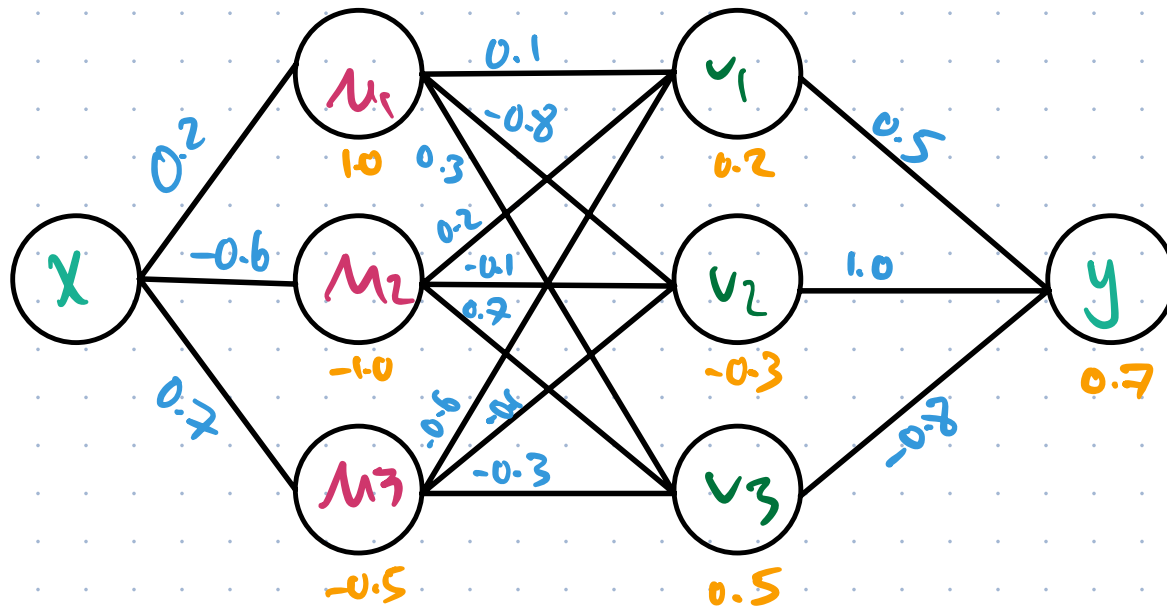
$$+ (w_{21}x_{11}y_{11} + w_{21}x_{12}y_{21} + w_{21}x_{13}y_{31} + w_{22}x_{21}y_{11} + w_{22}x_{22}y_{21} + w_{22}x_{23}y_{31} + w_{23}x_{31}y_{11} + w_{23}x_{32}y_{21} + w_{23}x_{33}y_{31} + w_{24}x_{41}y_{11} + w_{24}x_{42}y_{21} + w_{24}x_{43}y_{31}) \cdot v$$

$$+ b_1x_{11}y_{11} + b_1x_{12}y_{21} + b_1x_{13}y_{31} + b_2x_{21}y_{11} + b_2x_{22}y_{21} + b_2x_{23}y_{31} + b_3x_{31}y_{11} + b_3x_{32}y_{21} + b_3x_{33}y_{31} + b_4x_{41}y_{11} + b_4x_{42}y_{21} + b_4x_{43}y_{31} + c_1y_{11} + c_2y_{21} + c_3y_{31} + d_1$$

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

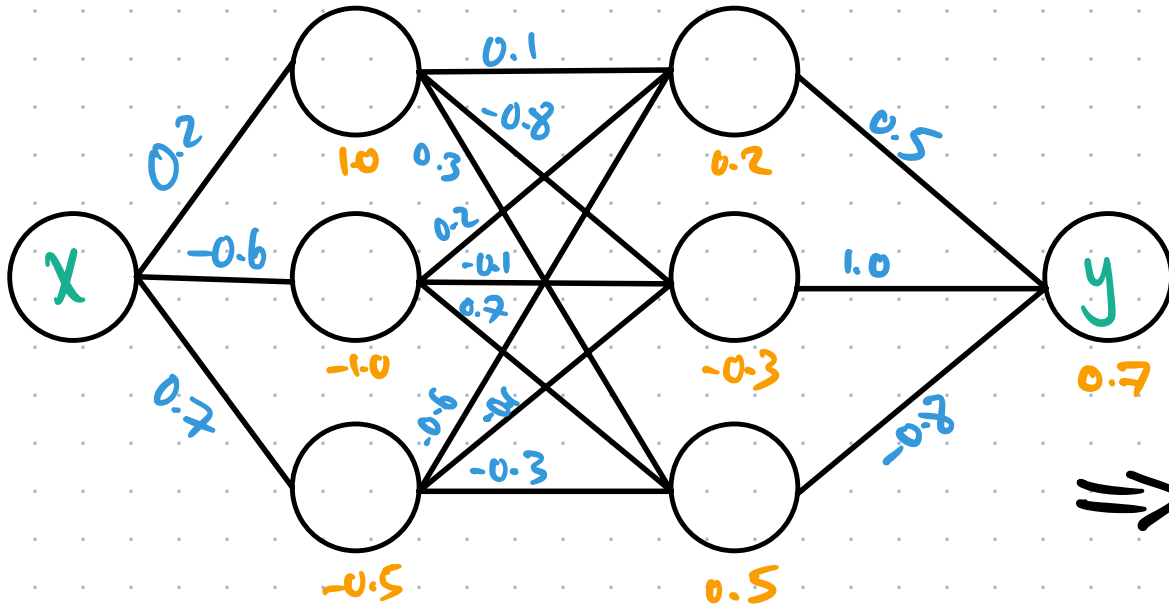
The problem is even more obvious when there's only one input and one output.

$$y = 0.5v_1 + 1.0v_2 - 0.8v_3 + 0.7$$



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

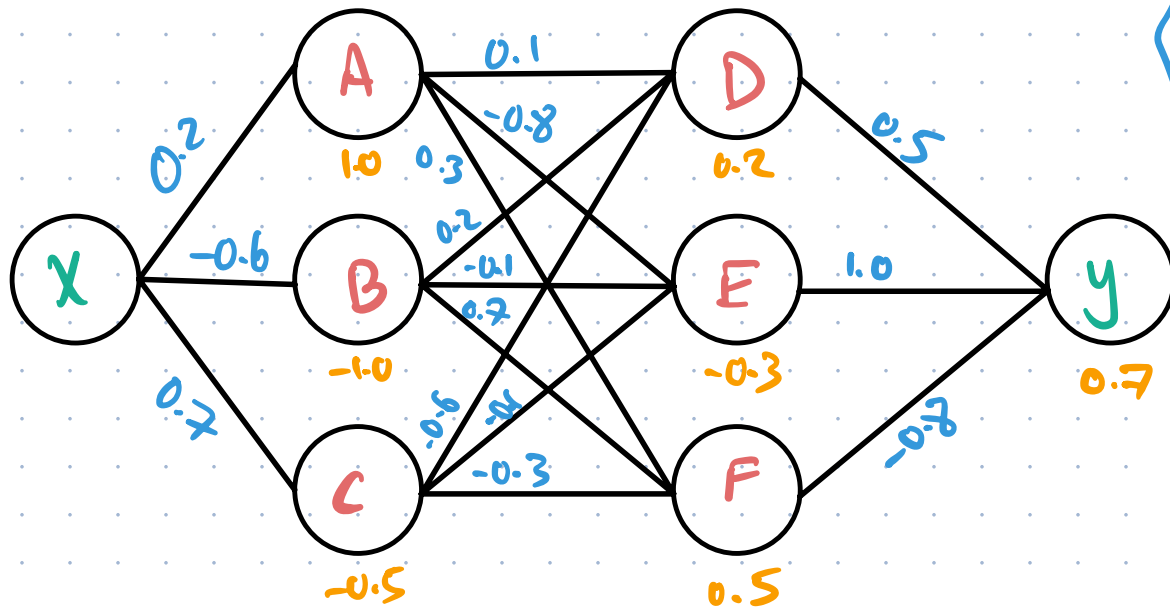
Just a line defined in a complicated way!



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

$$\Rightarrow (0.026)(0.3) - 0.25 = -0.2422 \quad \checkmark$$

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



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

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

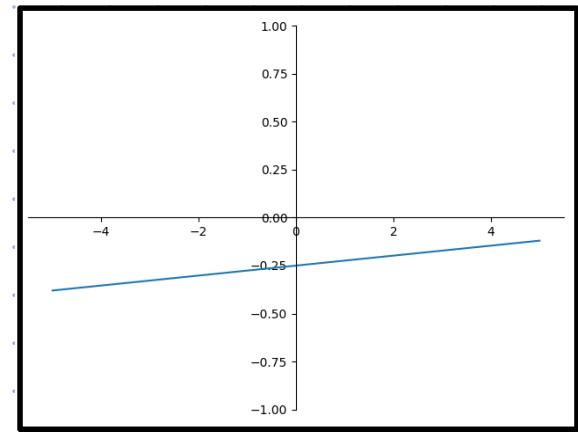
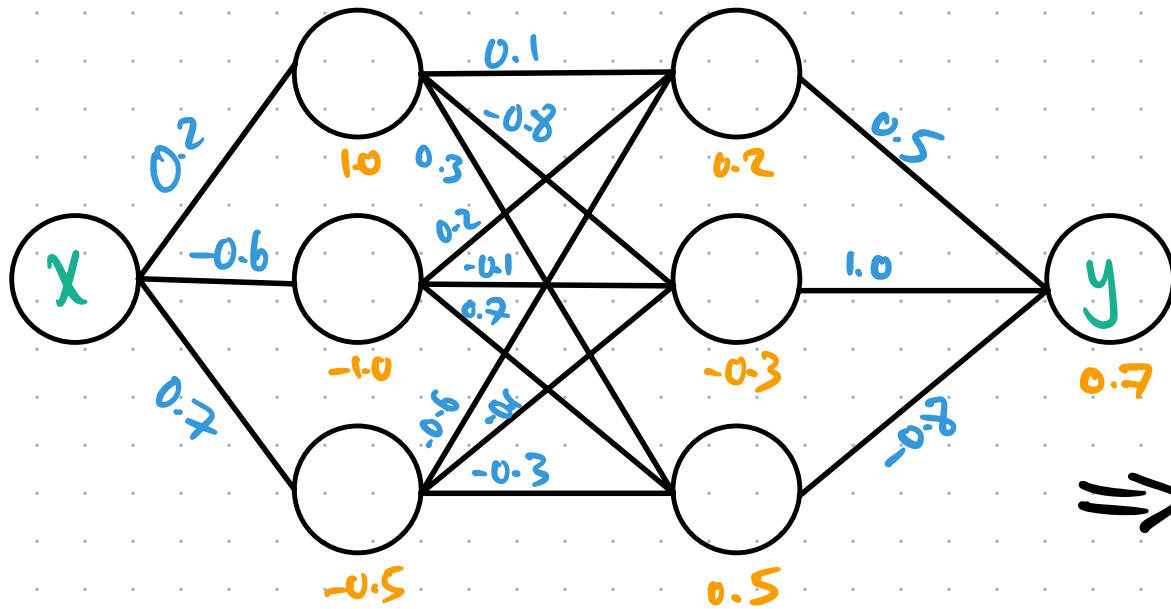
$$C = (0.7)(0.3) - 0.5 = -0.29$$

$$D = 0.244$$

$$E = -1.001$$

$$F = 0.079$$

$$y = -0.2422$$



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

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

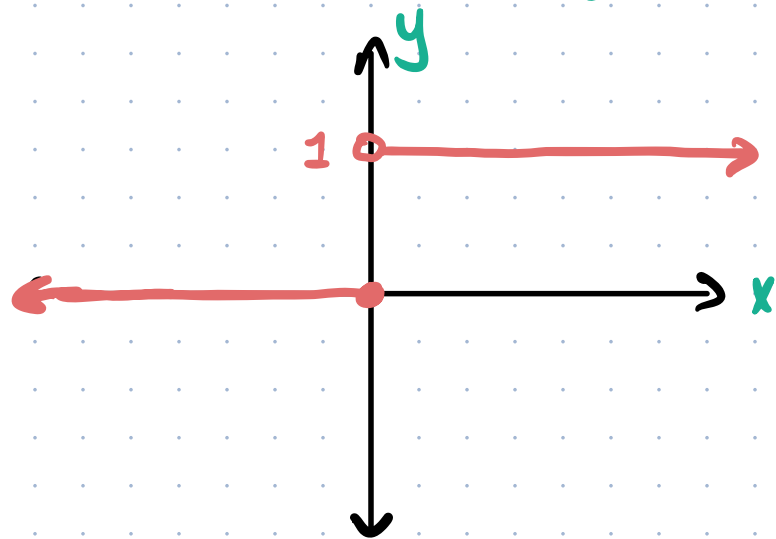
In order to make NNs capable of representing non-linear functions, we pass the output of each neuron through an "activation function" before passing it on to the next layer.

The activation functions themselves are non-linear.

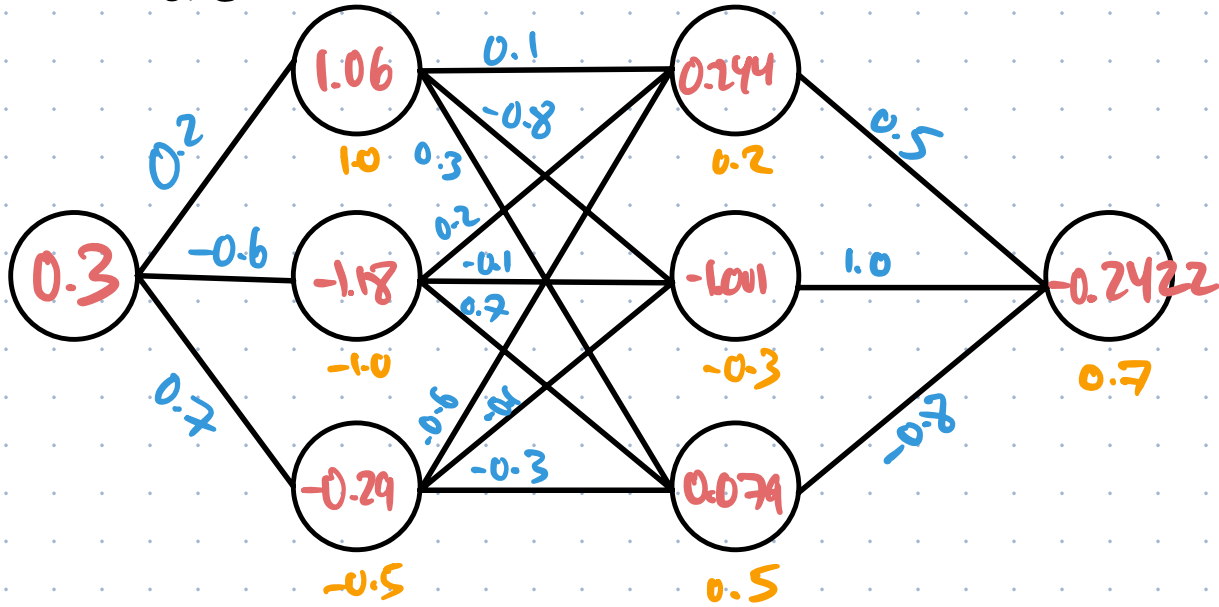
AF #1: Step Function (old school, not used often anymore)

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

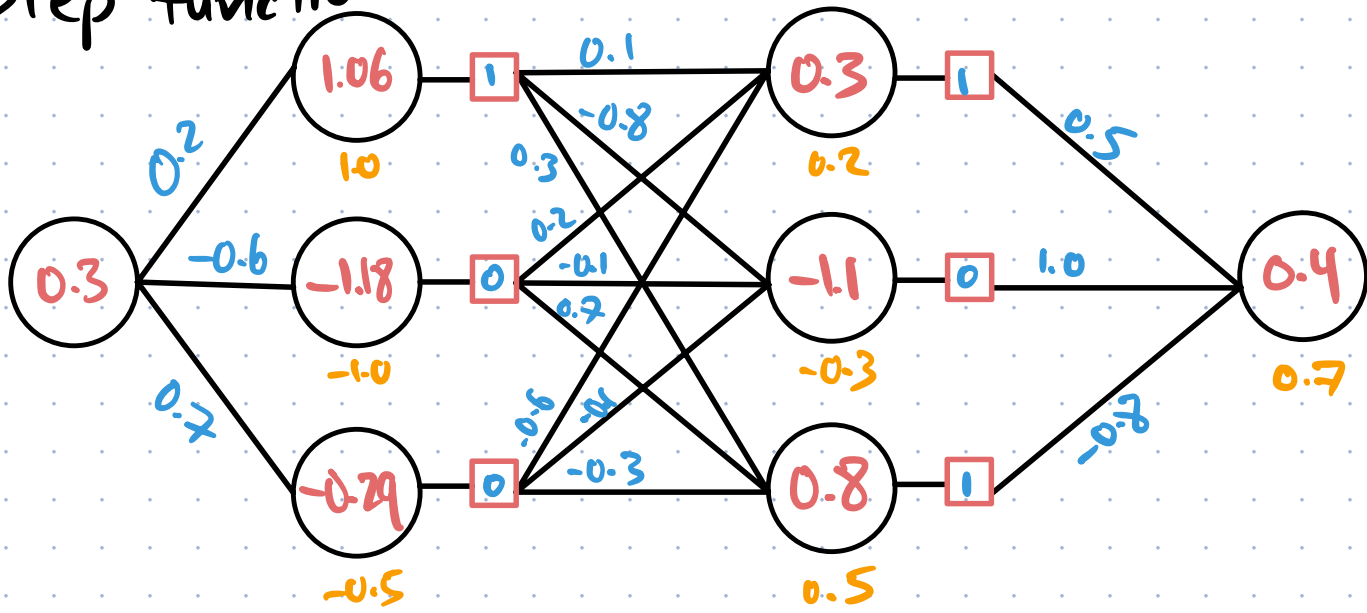
With this AF each neuron only outputs 0 or 1, not any decimal.
(off or on)



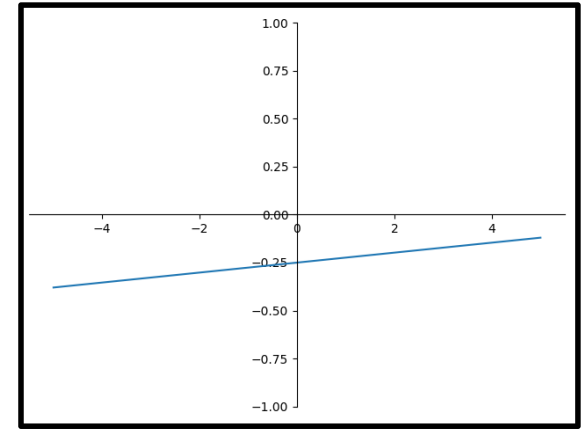
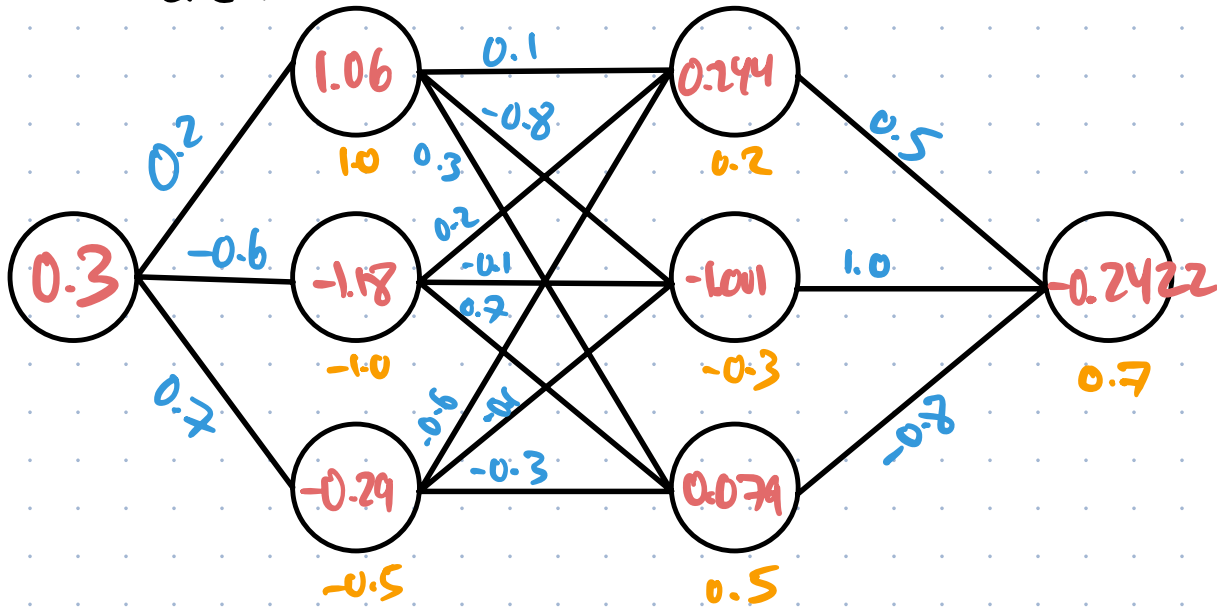
no activation:



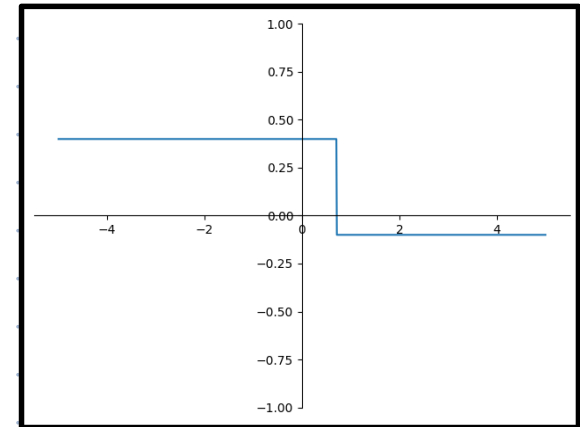
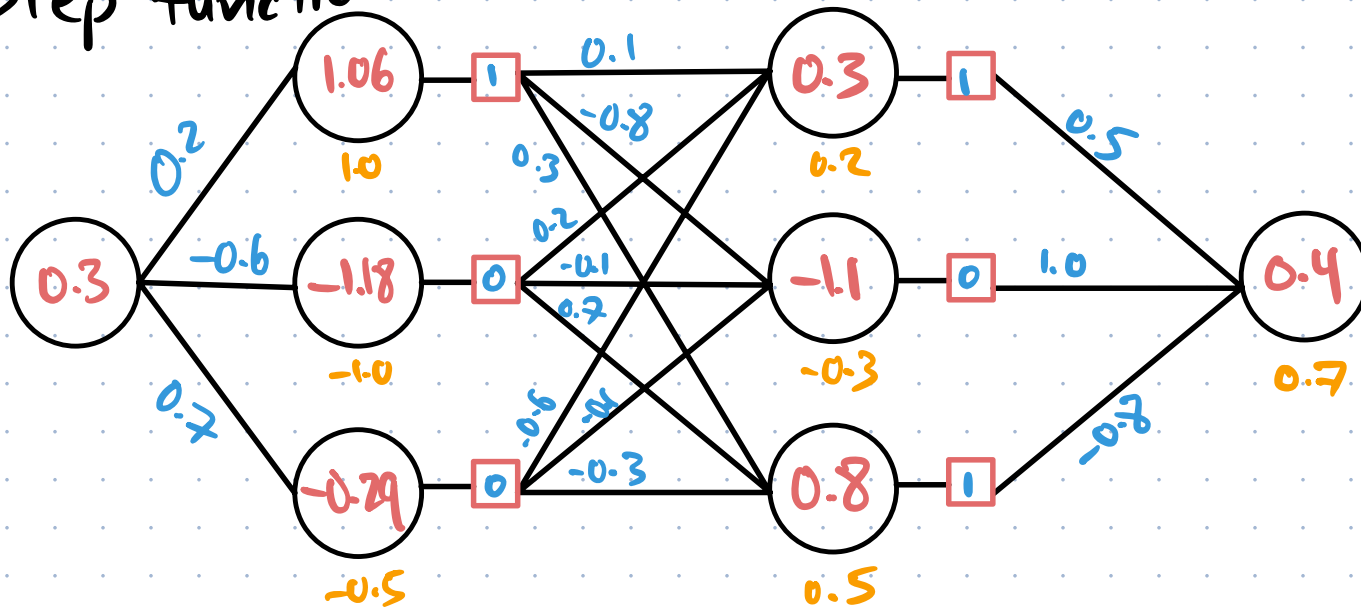
Step function:



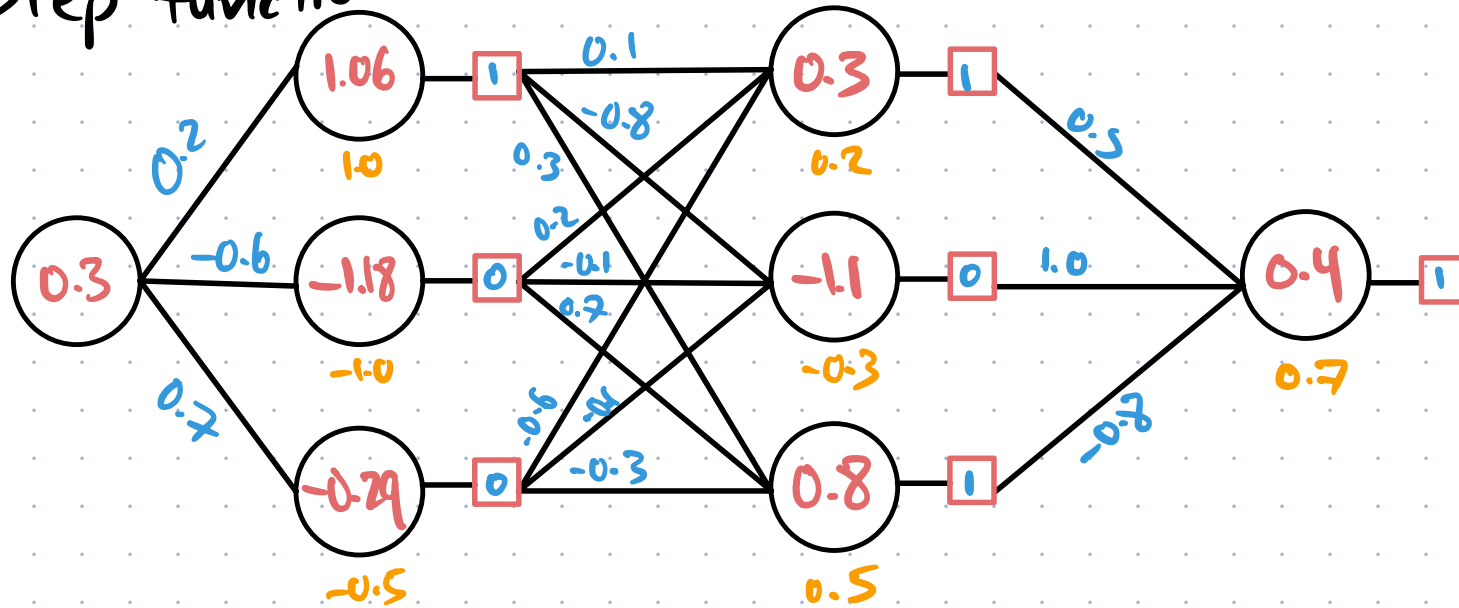
no activation:



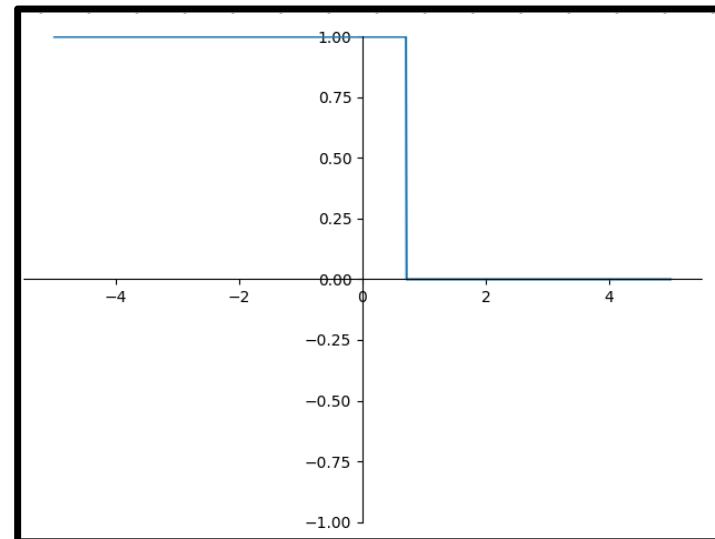
Step function:



Step function:



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

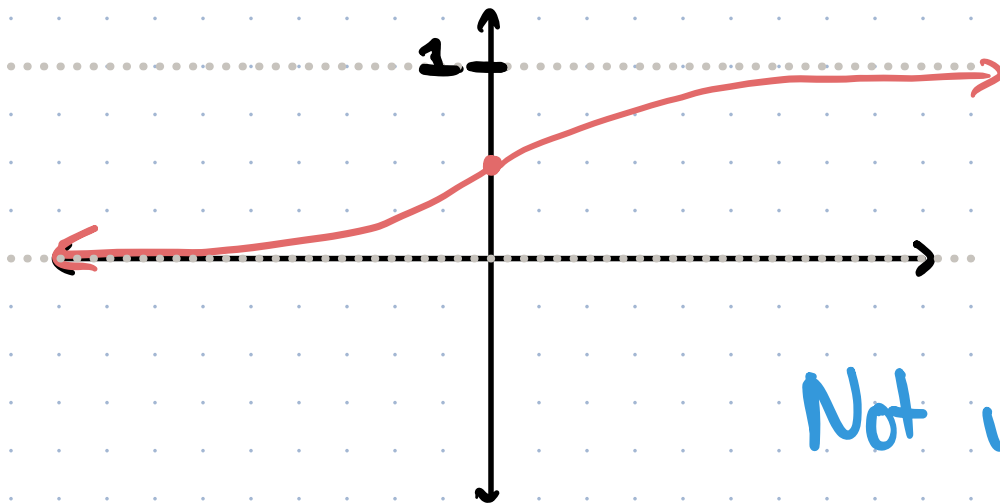


AF #2: Sigmoid

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

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

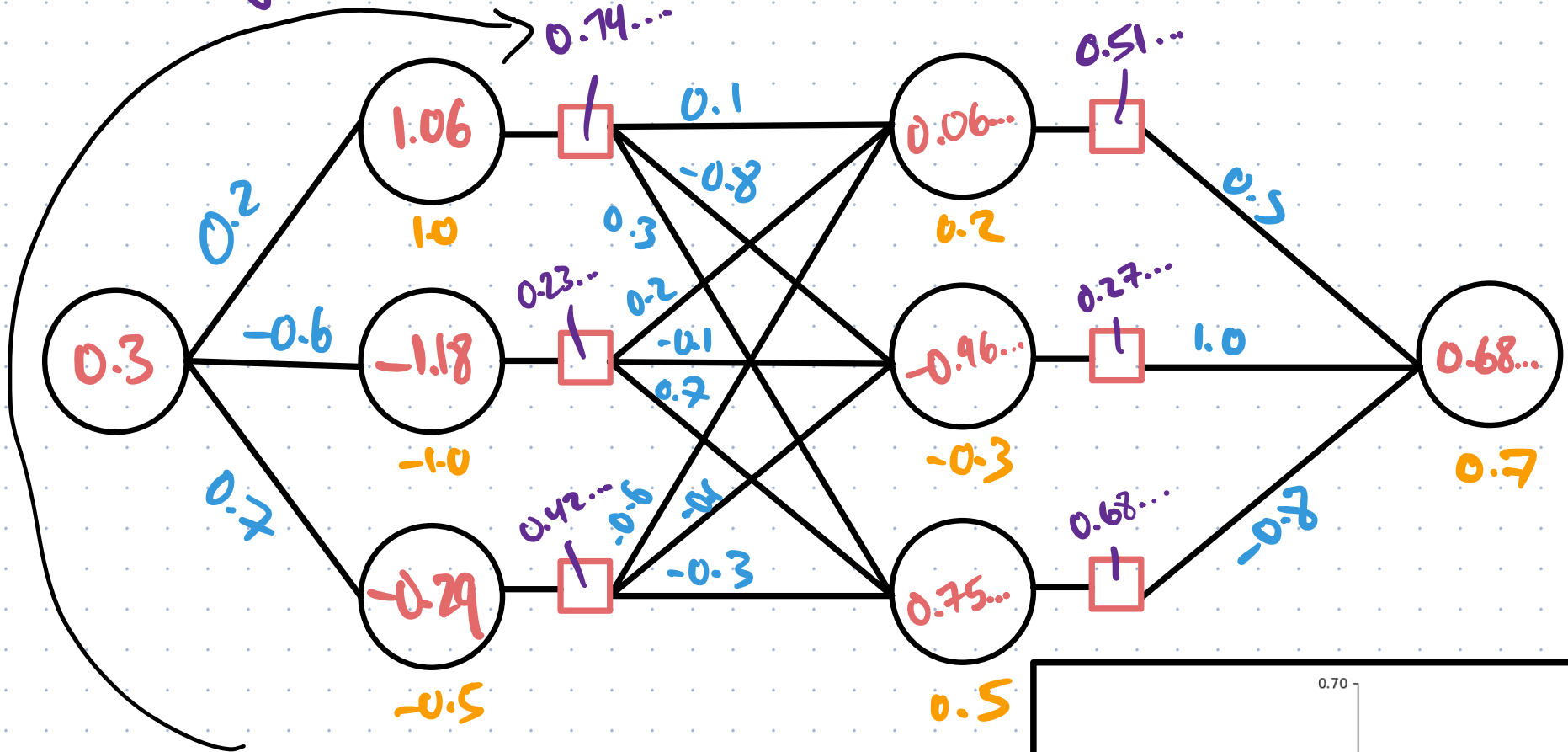
Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$



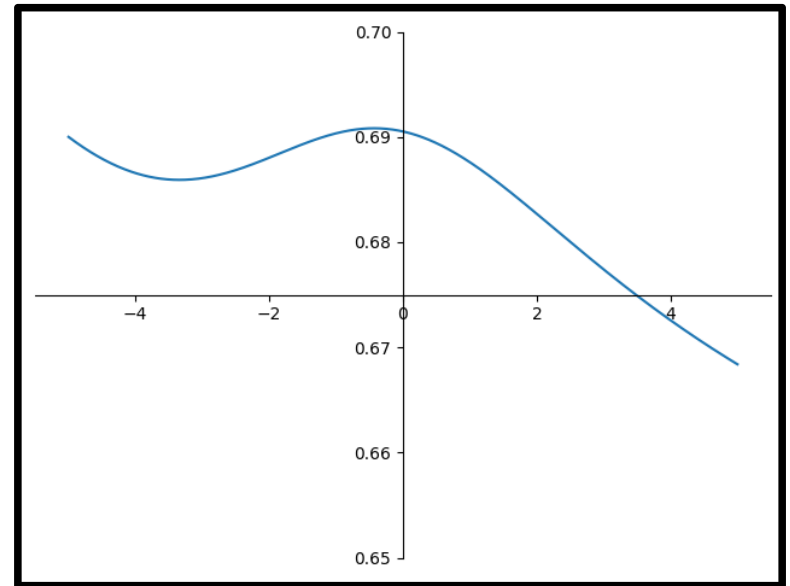
More gentle, and not as lossy.

Not used much anymore

With sigmoid activation

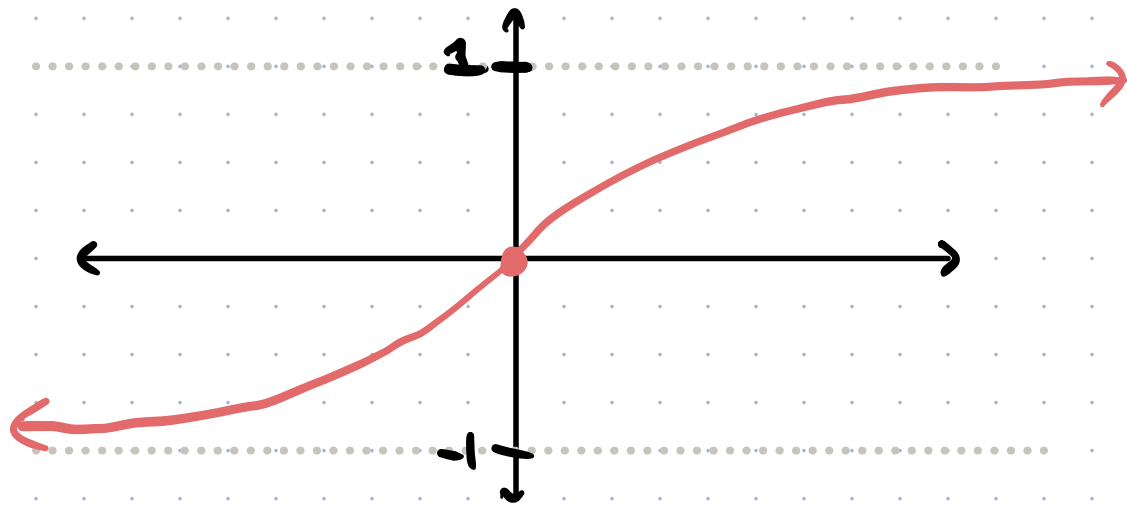


$$\frac{1}{1 + e^{-1.06}} \approx 0.74$$



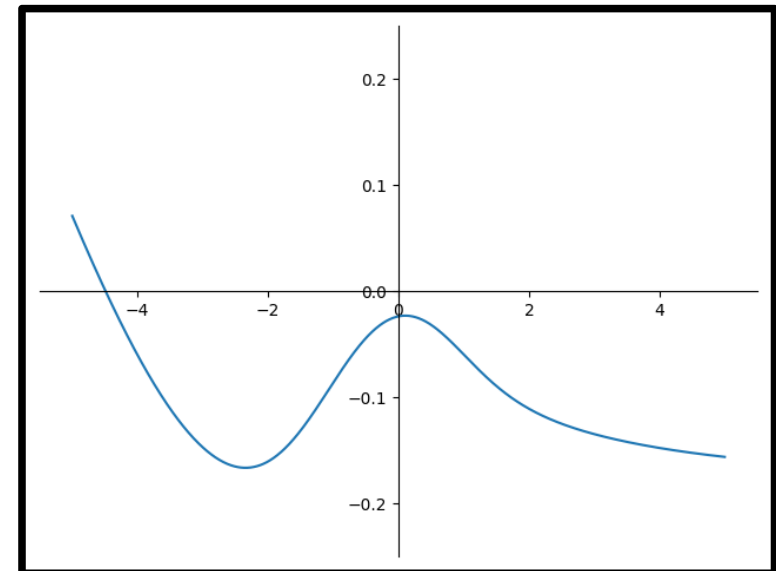
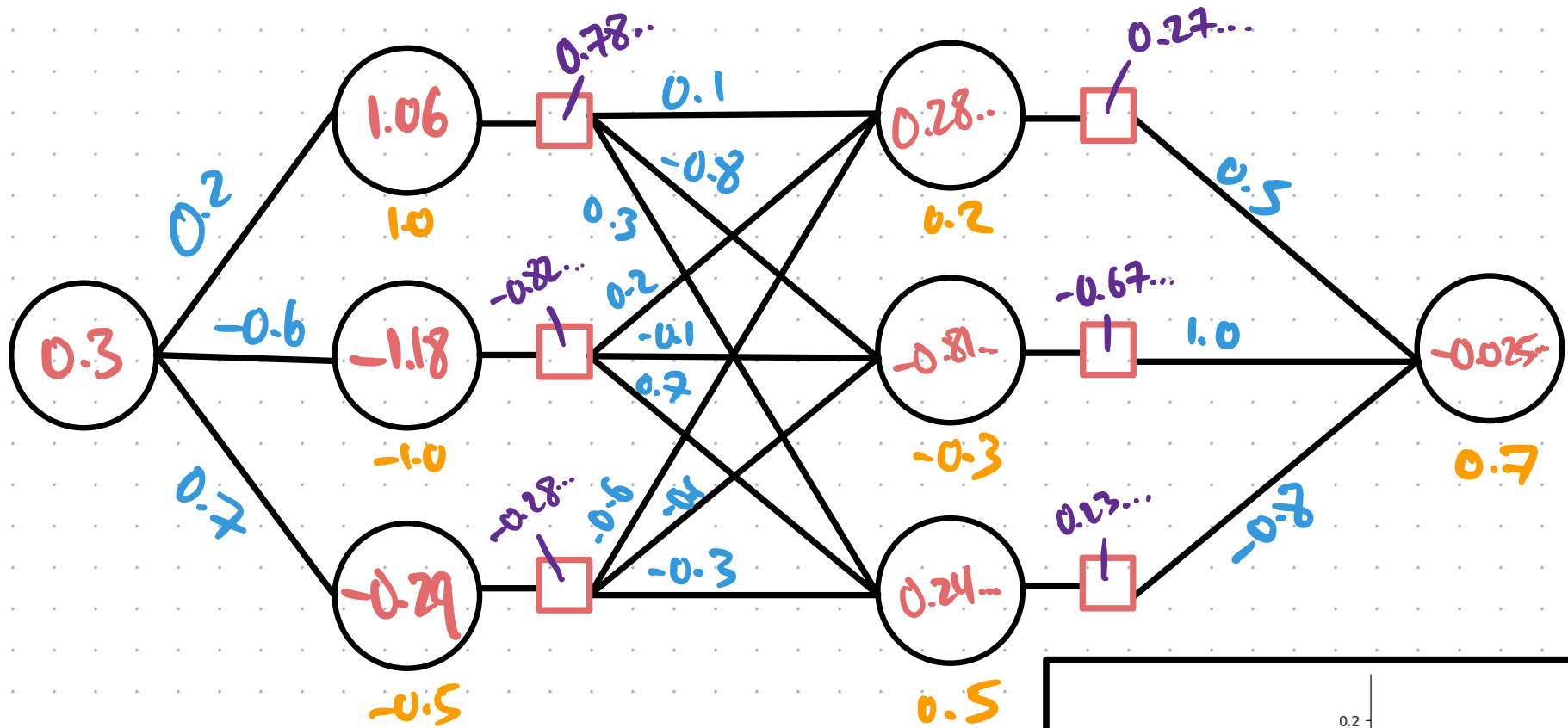
AF #3: Hyperbolic Tangent (tanh)

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



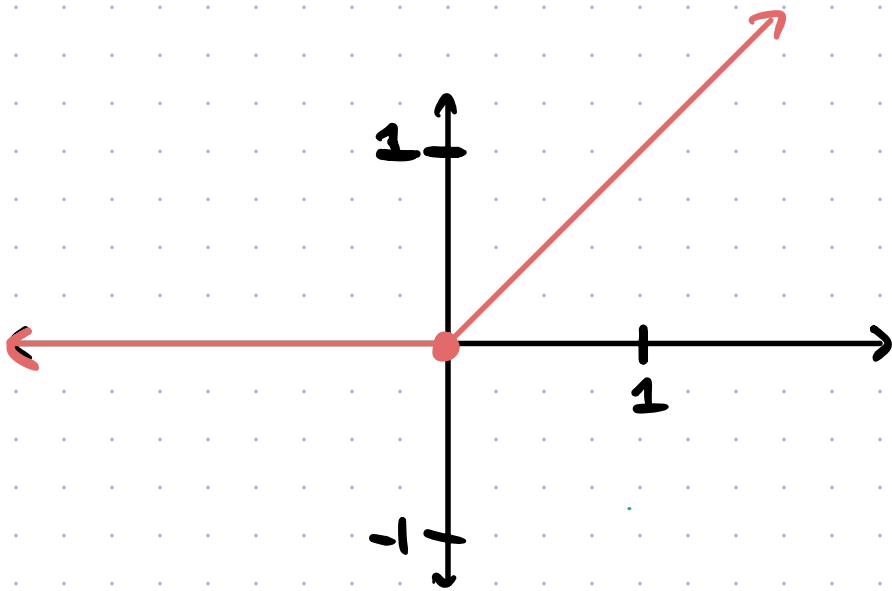
a bit like sigmoid,
but can be
negative

With tanh activation



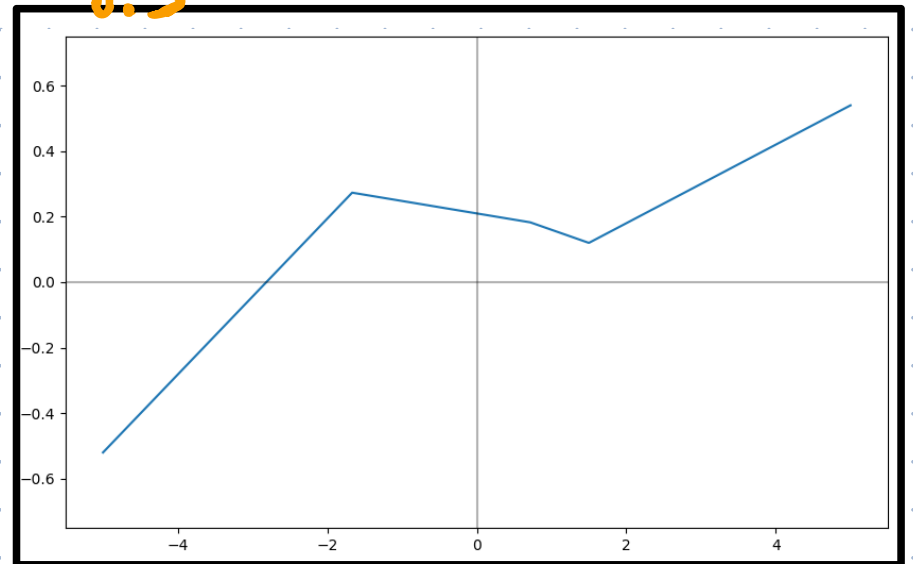
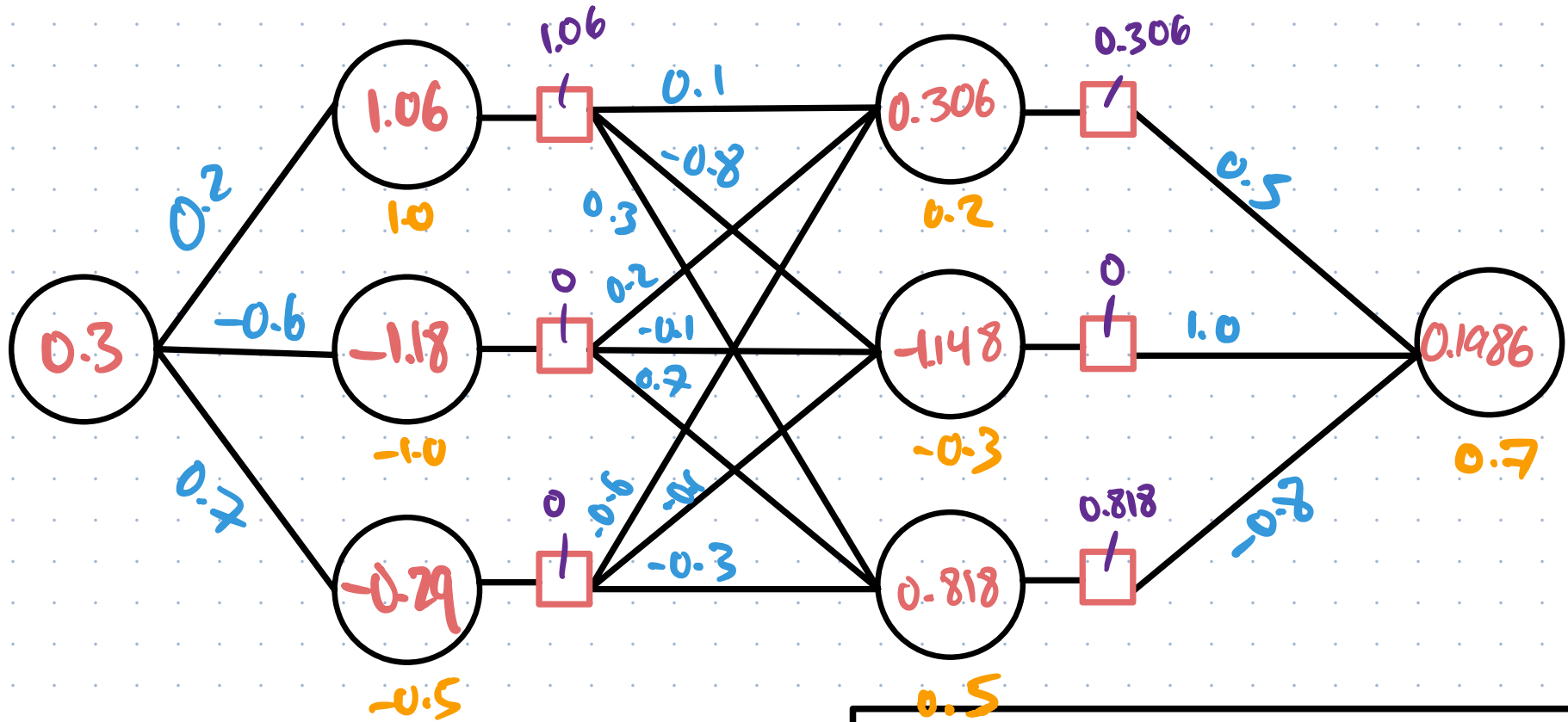
AF #4: Rectified Linear Unit (ReLU)

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



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

With ReLU activation

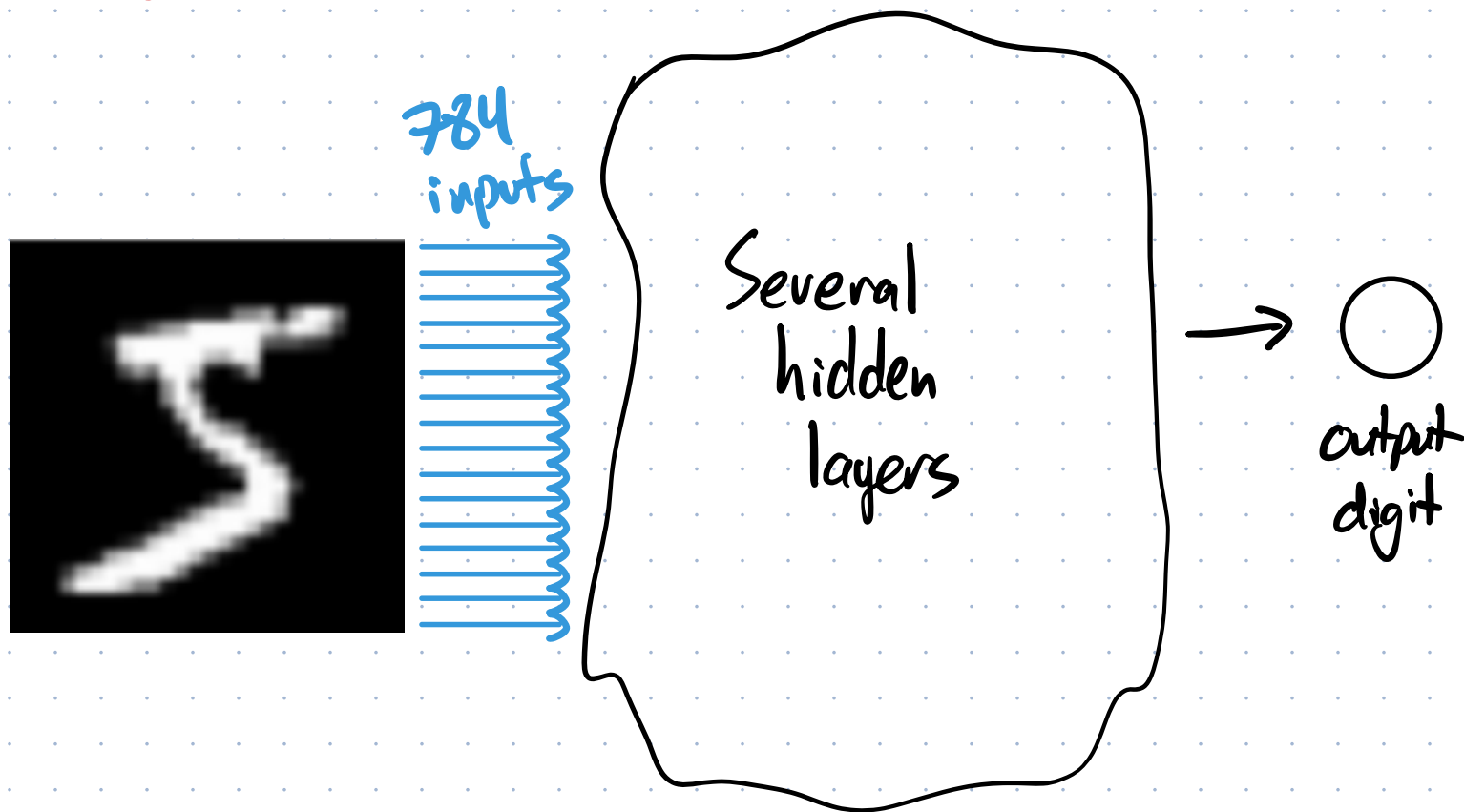


Feed forward Demo:

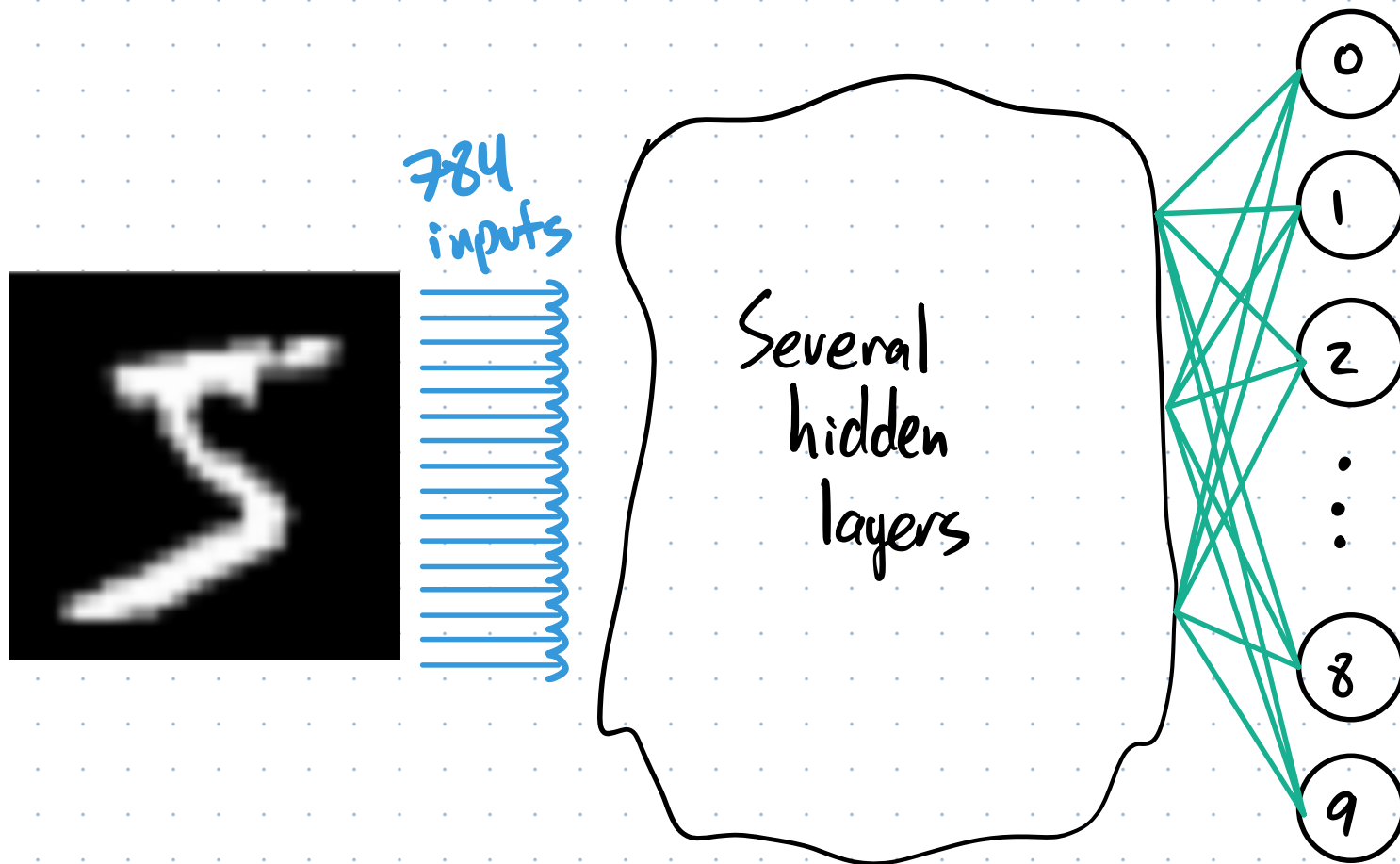
- * Different activation functions
- * Graphs on each neuron

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.

Think of each layer as a vector of values.

$$\begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix}$$

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

$$\begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix}$$

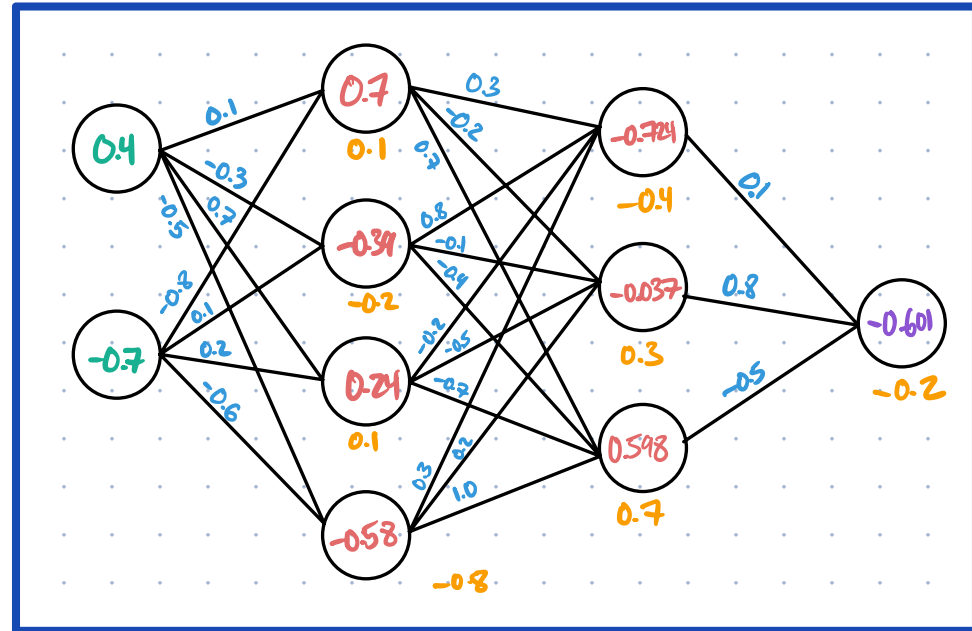
$$[-0.601]$$

How is each layer computed from the previous one?

Make matrices for the weights between each layer and vectors for the biases.

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

$$b_1 = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix}$$



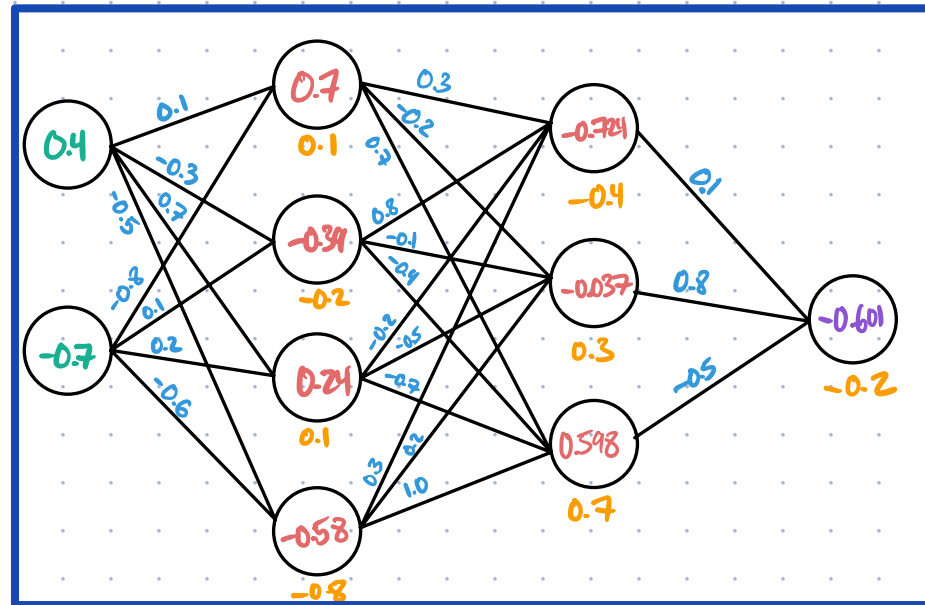
Rearrange!

(ignoring activation functions!)

$$\begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \cdot \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix}$$

$M_1 =$

$b_1 =$

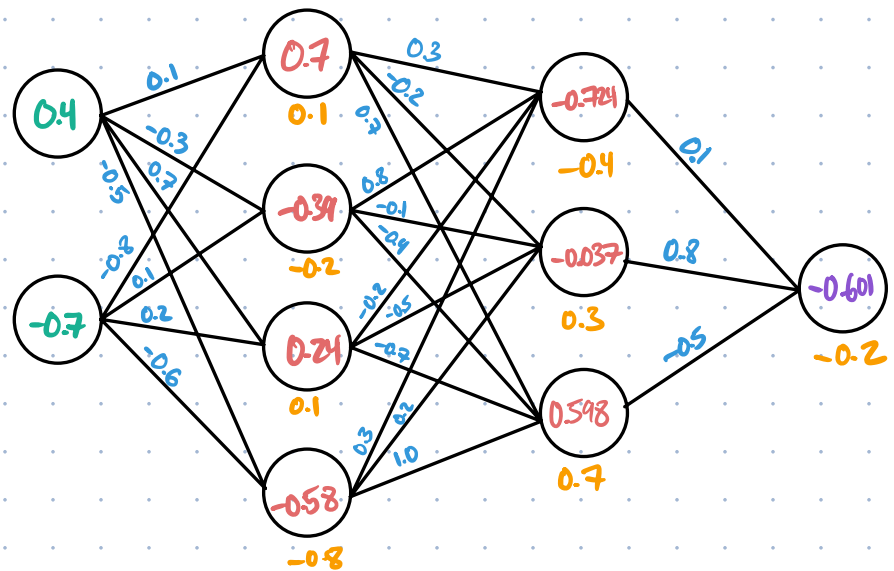


$$\begin{bmatrix} 0.1 & -0.8 \\ -0.3 & 0.1 \\ 0.7 & 0.2 \\ -0.5 & -0.6 \end{bmatrix} \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix}$$

(ignoring activation functions!)

$$\begin{bmatrix} 0.3 & 0.8 & -0.2 & 0.3 \\ -0.2 & -0.1 & -0.5 & 0.2 \\ 0.7 & -0.4 & -0.7 & 1.0 \end{bmatrix} \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix} + \begin{bmatrix} -0.4 \\ 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.8 & -0.5 \end{bmatrix} \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix} + \begin{bmatrix} -0.2 \end{bmatrix} = \begin{bmatrix} -0.601 \end{bmatrix}$$



$$\begin{bmatrix} 0.3 & 0.8 & -0.2 & 0.3 \\ -0.2 & -0.1 & -0.5 & 0.2 \\ 0.7 & -0.4 & -0.7 & 1.0 \end{bmatrix} \begin{bmatrix} 0.7 \\ -0.39 \\ 0.24 \\ -0.58 \end{bmatrix} + \begin{bmatrix} -0.4 \\ 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} -0.724 \\ -0.037 \\ 0.598 \end{bmatrix}$$

$$Wx_1 + b = x_2$$

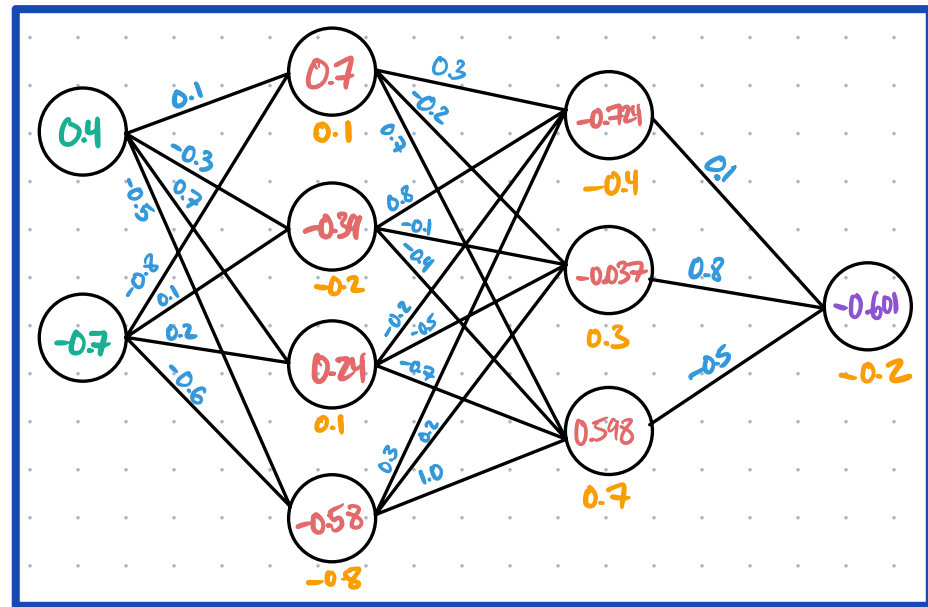
What about activation function?

Suppose our activation function is σ .

We'll use the notation

$$\sigma \left(\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix} \right) = \begin{bmatrix} \sigma(v_1) \\ \sigma(v_2) \\ \vdots \\ \sigma(v_k) \end{bmatrix}$$

$$\sigma(Wx_1 + b) = x_2$$



So this neural network, with activation function σ for each layer, is the function:

$$f(x) = \sigma(W_3 \sigma(W_2 \sigma(W_1[x] + b_1) + b_2) + b_3)$$

