Introduction to Neural Networks

Two separate questions:

(1) What is a neural network?
What does it do?

(2) How do we produce a good one
for our task?

This lecture addresses (1).

Future lectures address (2).

Sources:

* Book: "Neural Networks From Scratch in Python"
* Lots of online resources and Youtube videos that I'll share as we go.

Goals:

* Understand the foundations of neural networks.

* See examples of training NNs to accomplish a task

  ↳ "training" = "find a good NN"

* Write our own Python code to create and train NNs (not using Machine Learning libraries)

Maybe:

* Learn about Python ML libraries (pytorch, tensorflow)

* Learn about specialized kinds of NNs for certain tasks.

Strong Analogy: Linear Regression

What is it?

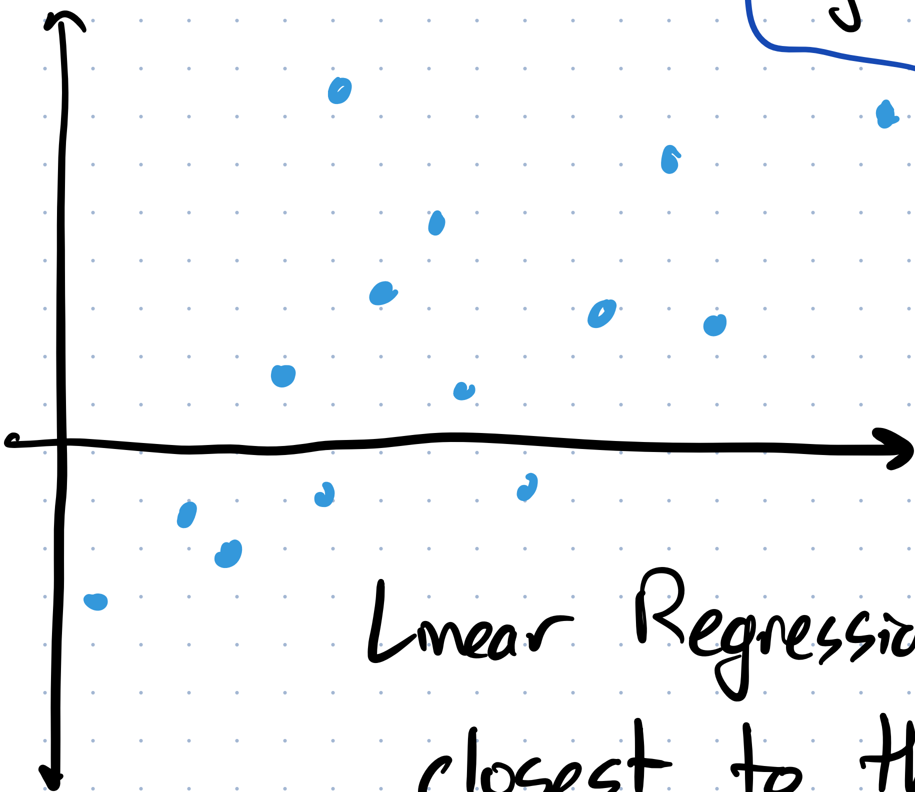
Simplest version: you have a bunch of (x,y) points and you want to find the line the best approximates them

Usually, the (x,y) points represent inputs (x) and outputs (y) of some unknown function.

x: time since Jan 1 this year
y: temperature on my outdoor thermometer

Strong Analogy: Linear Regression

x: time since Jan 1 this year
y: temperature on my outdoor thermometer



We only have sporadic readings.

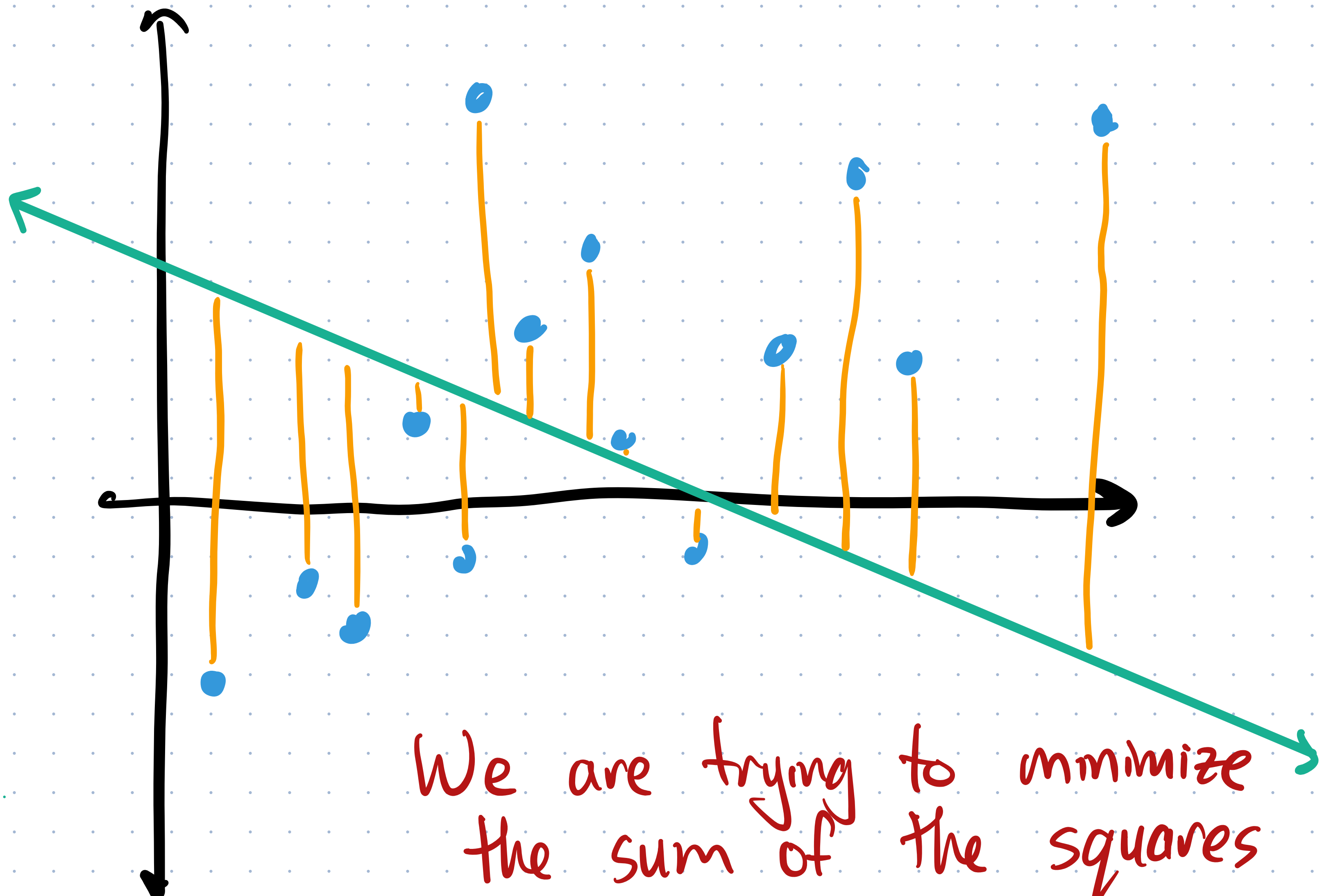
Linear Regression asks "what line is closest to these points?"

"Closest" means minimizing the sum of

$$([\text{actual } y \text{ value}] - [\text{predicted } y \text{ value}])^2$$

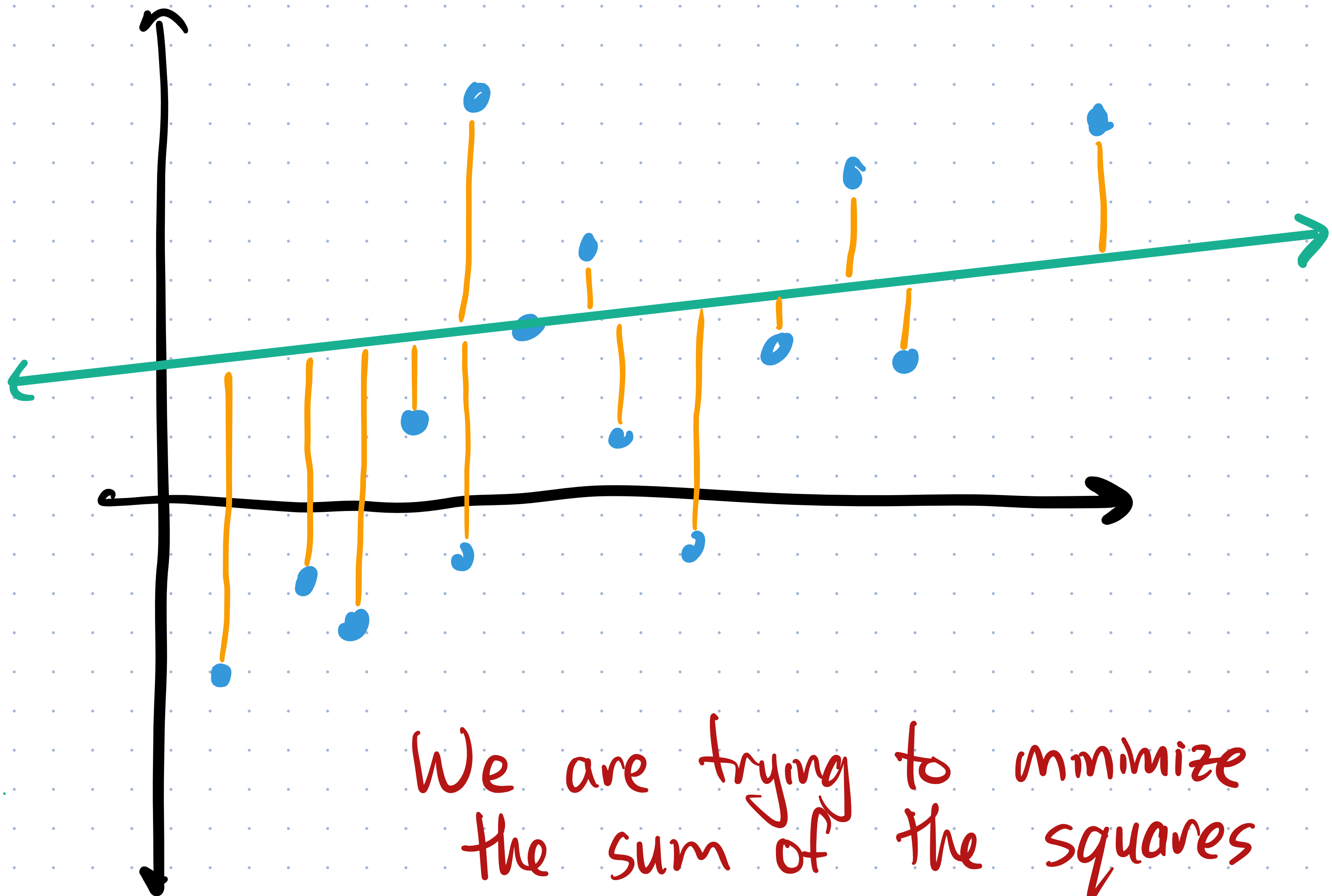
over all known points.

Strong Analogy: Linear Regression



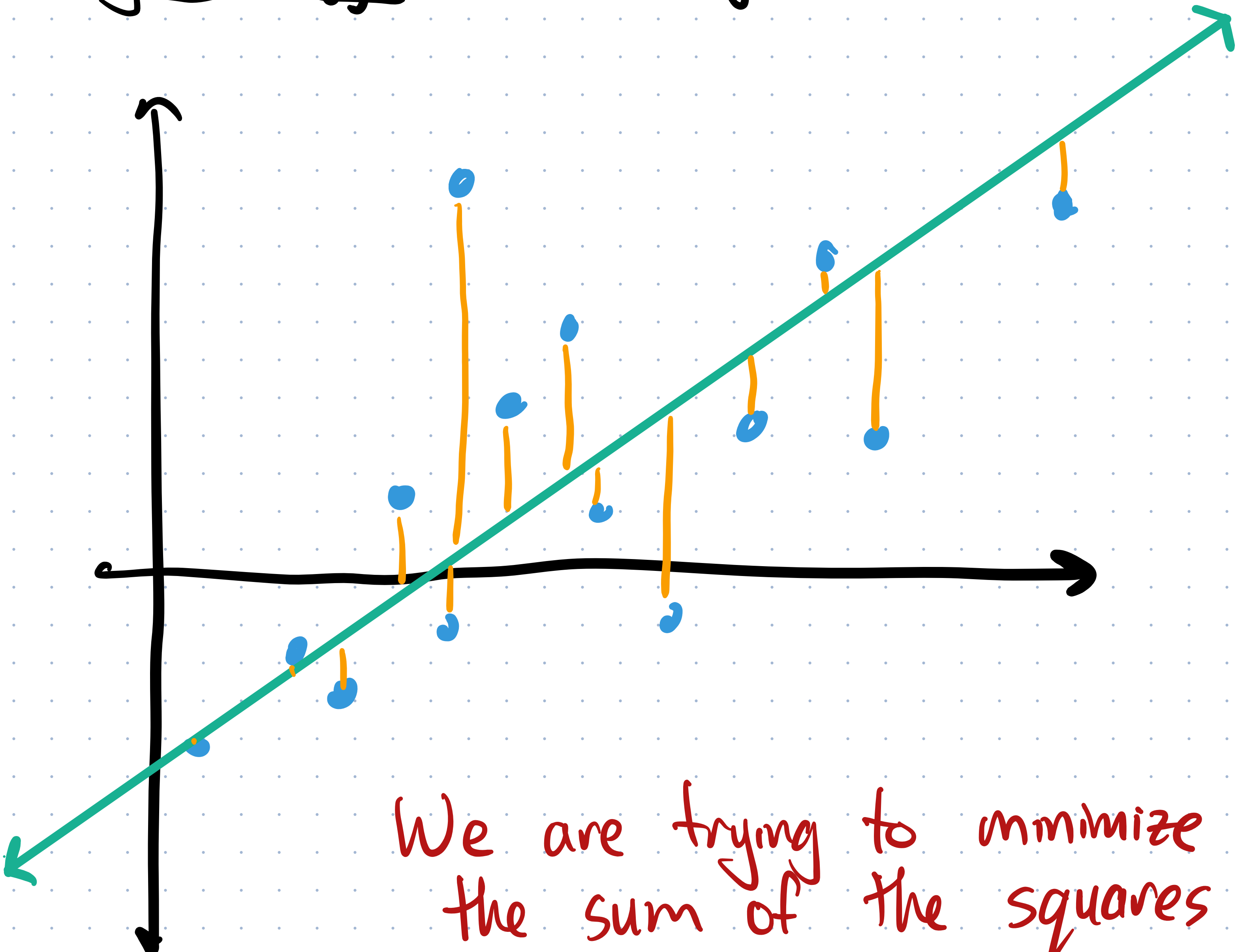
We are trying to minimize the sum of the squares of the lengths of the yellow lines

Strong Analogy: Linear Regression



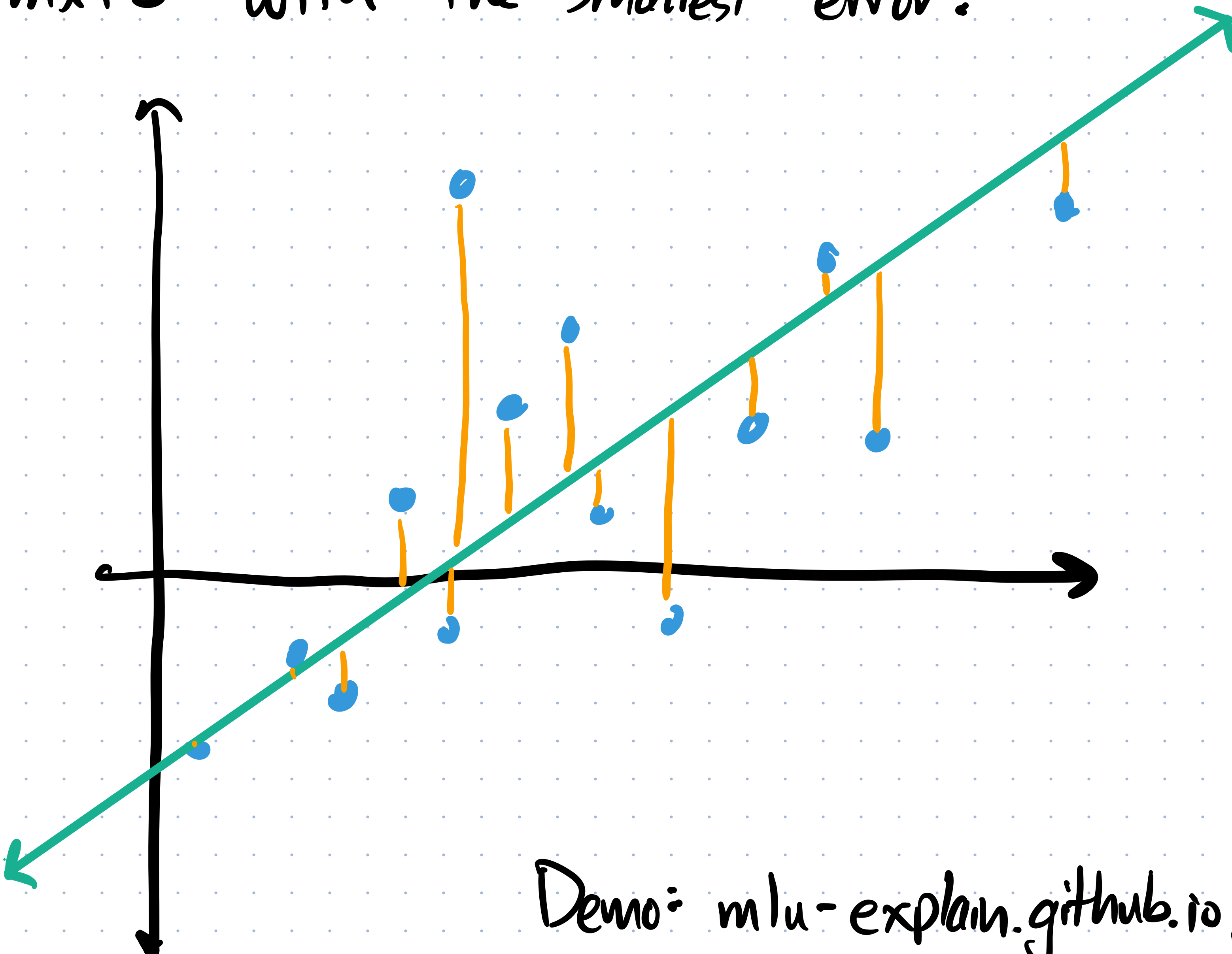
We are trying to minimize the sum of the squares of the lengths of the yellow lines

Strong Analogy: Linear Regression



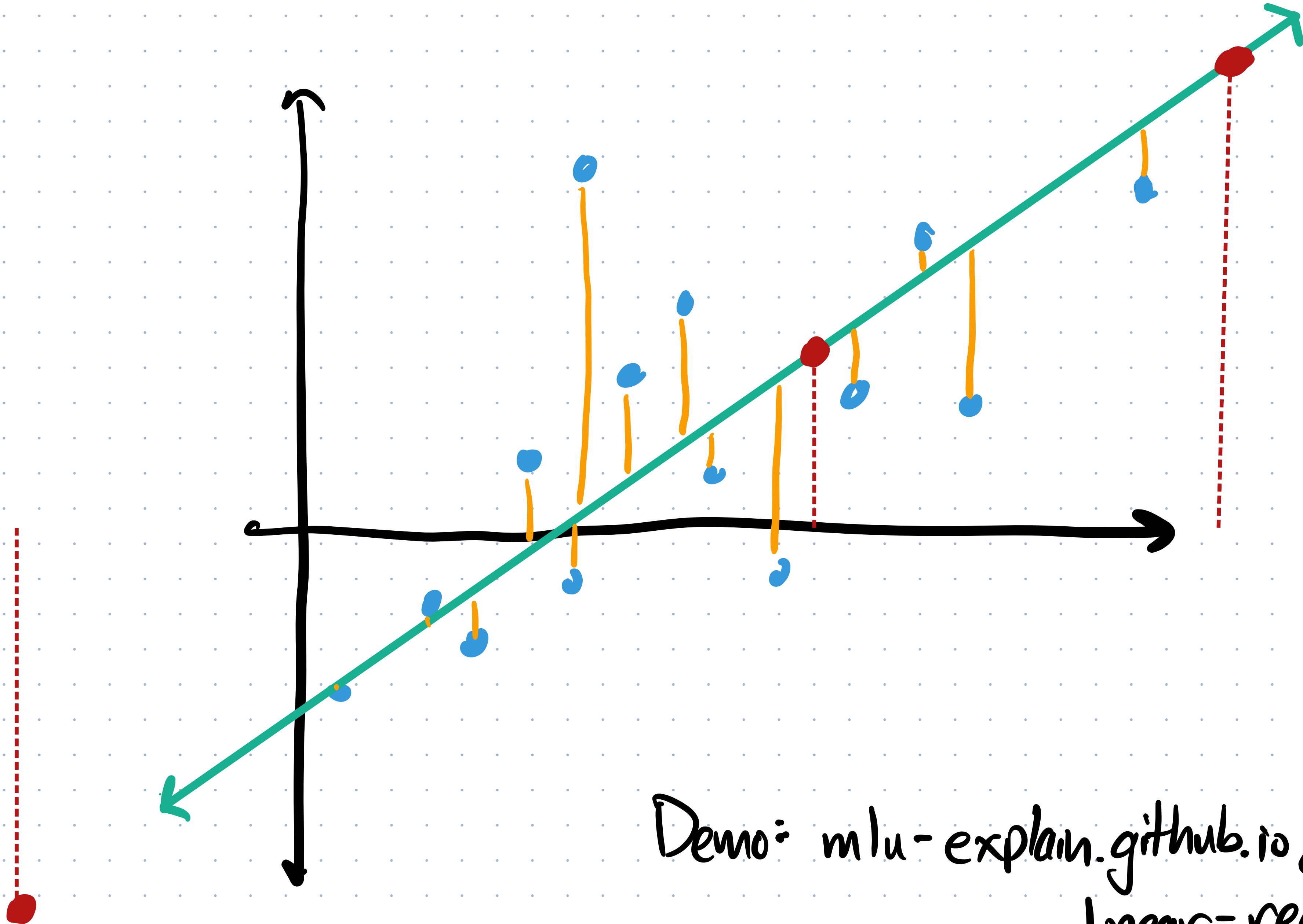
We are trying to minimize the sum of the squares of the lengths of the yellow lines

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.



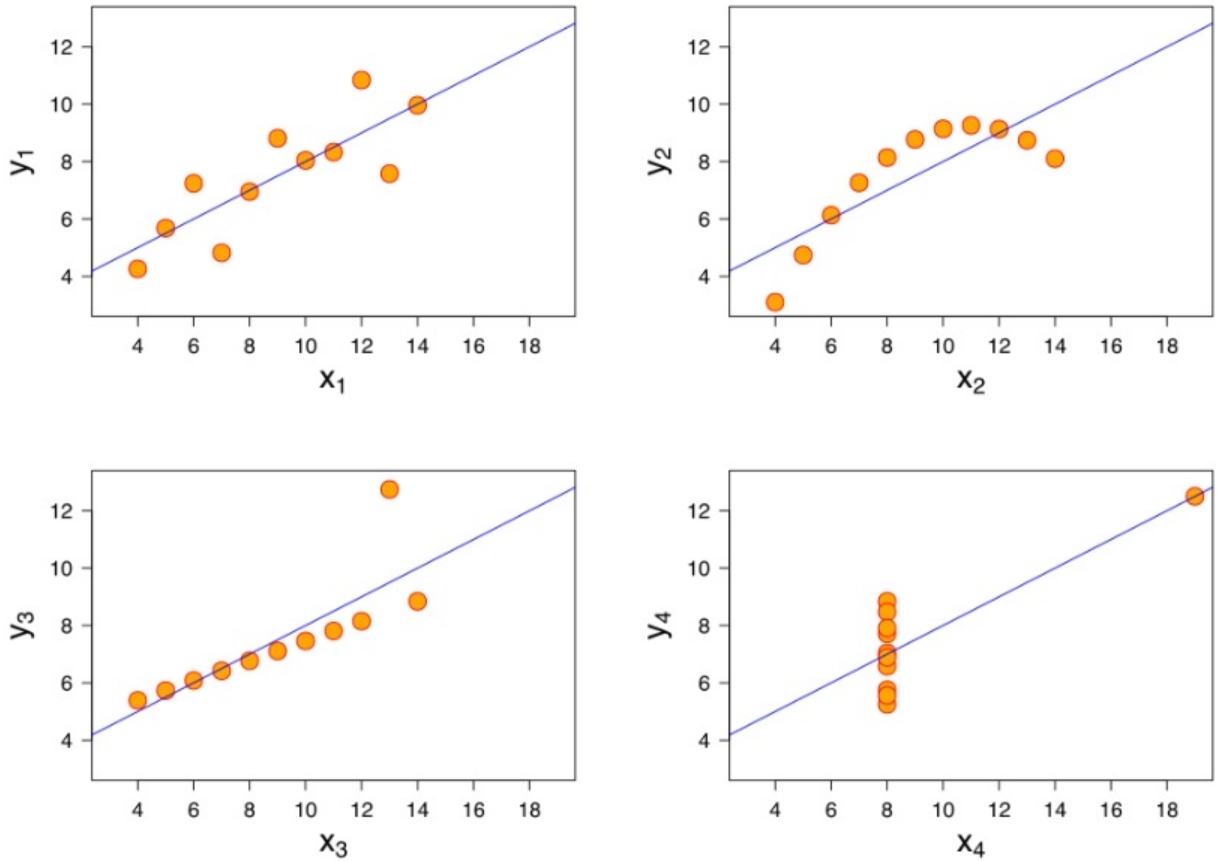
Demo: mlu-explain.github.io/linear-regression

This can be done with any # of input variables, and then you're finding planes and hyperplanes instead of lines. Same idea though.

How do you find the m, b that make the best line?

Formulas, Linear Algebra

The problem is a lot of data isn't linear.



The four datasets composing Anscombe's quartet. All four sets have identical statistical parameters, but the graphs show them to be considerably different

https://commons.m.wikimedia.org/wiki/File:Anscombe%27s_quartet_3.svg

Linear Regression: Approximate data with a line

Neural Network: Approximate data with <u>any function</u>

Pros: Can approximate data much better

Cons: Requires a lot of computation to produce

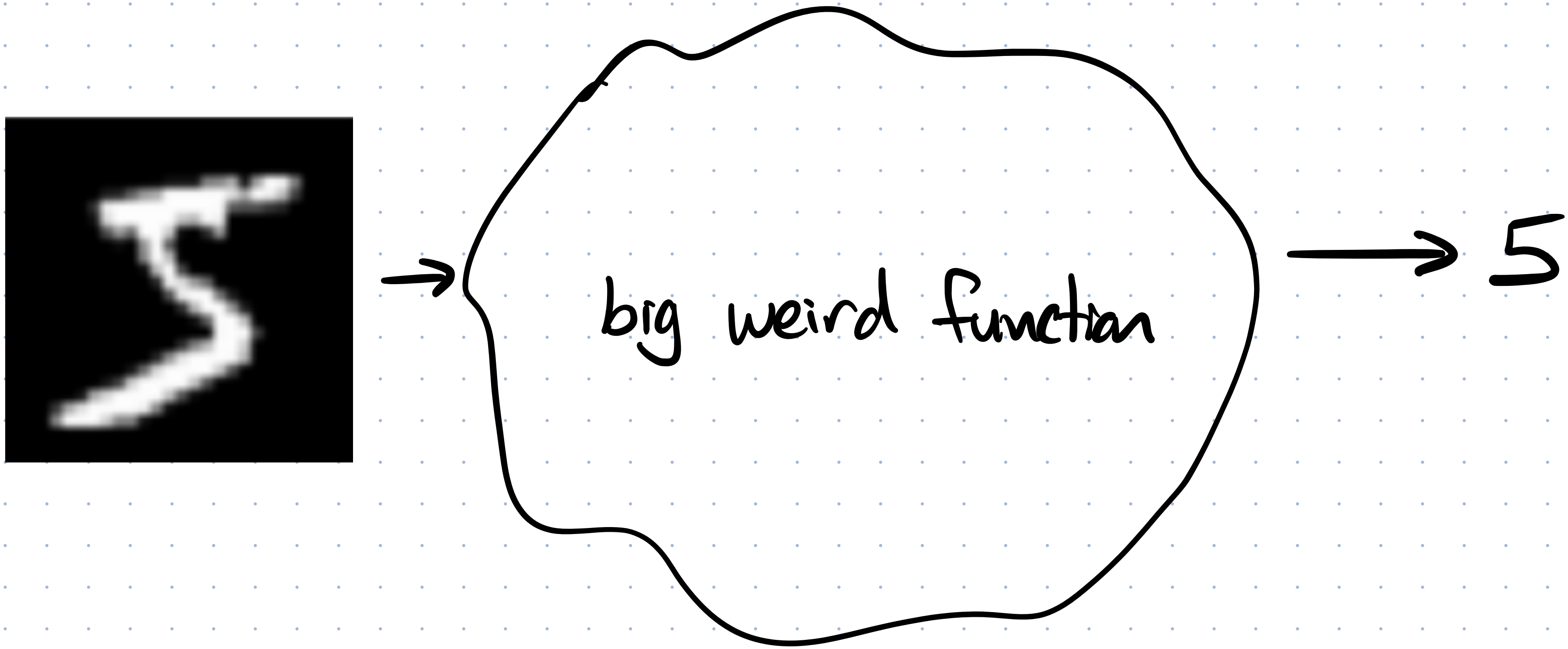
Big Picture:

* A neural network is just a fancy function.
* It takes numbers as inputs and produces numbers as outputs, just like any function from Calculus.
* Finding a good neural network that approximates your data is hard.

Also: Lots of interesting things can be framed as a (numerical inputs) → (numerical outputs) function

Example function:

Input: a 28 × 28 grayscale image
Output: what digit it is



How is this "numeric input"?

Example function:

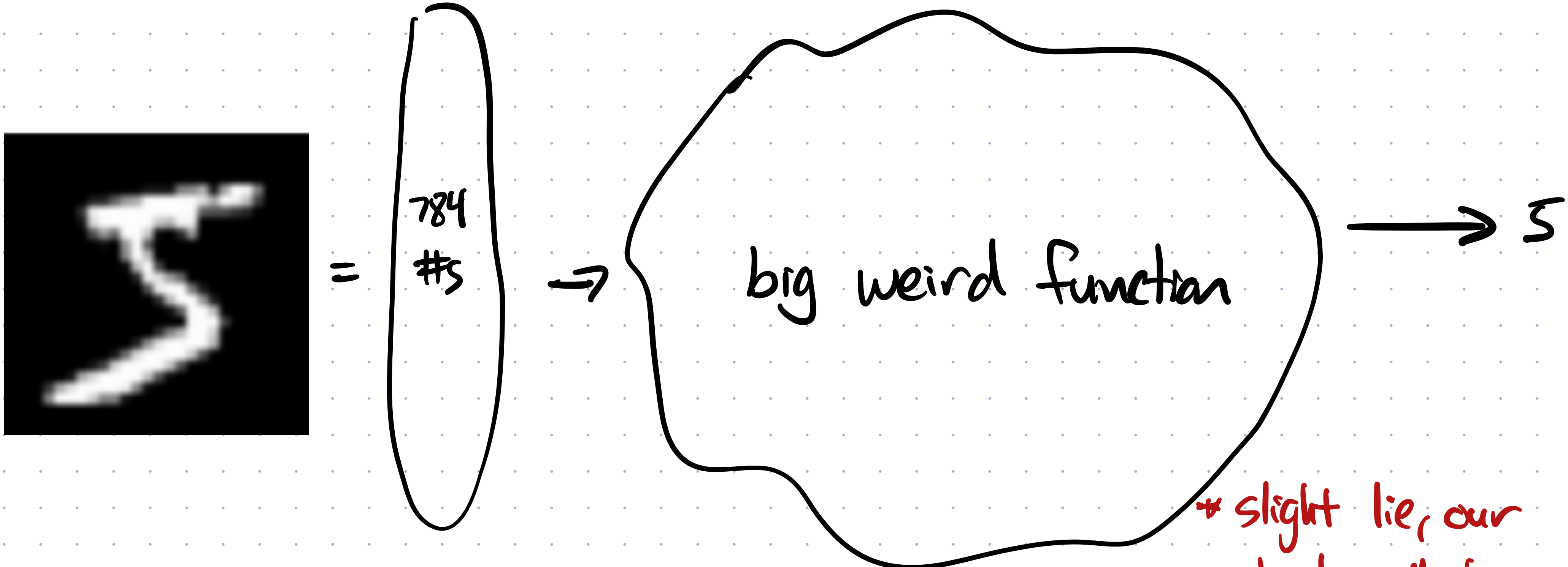
{ Input: a 28 × 28 grayscale image
Output: what digit it is }

* Each image is 28×28 = 784 pixels

Each pixel has a grayscale value from 0 (black) to 255 (white)

```
[[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253, 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35, 241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221, 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253, 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253, 195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,  11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0]],
```

(In practice, we divide by 255 to make a decimal between 0 and 1)



* slight lie, our output will be a little different

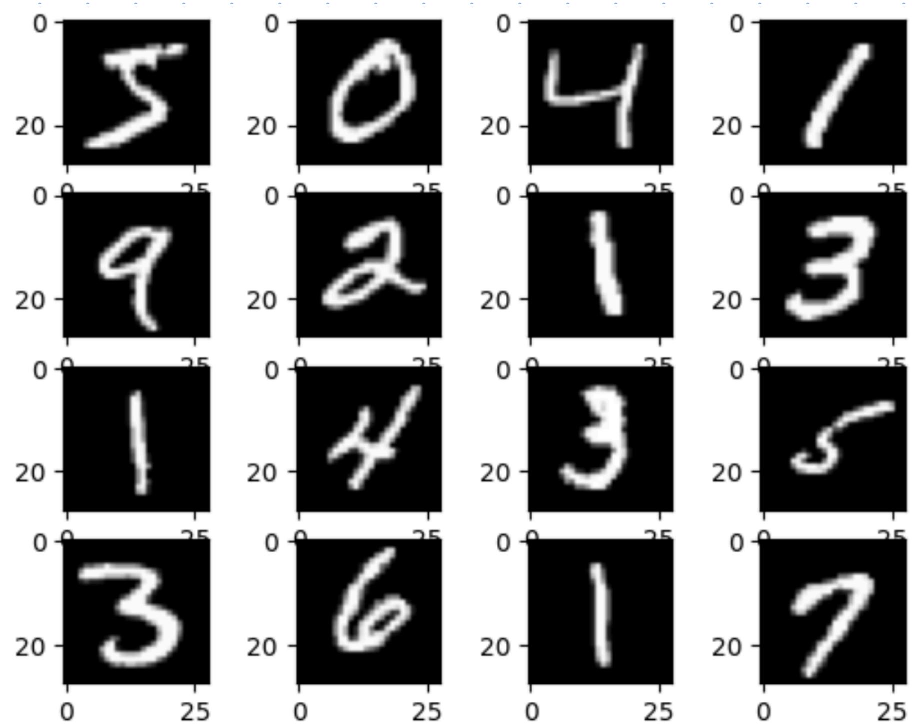
So this is a 784-dimensional function.

And not really well-defined because many input pictures are not obviously a particular number

That's fine! We'll produce a neural network that takes 784 input #s (1 per pixel) and predicts the #.

We'll use "training data", known (input, output) pairs, to build the NN.

(just like using data to find a linear regression line)



MNIST data set

Big Picture:

* A neural network is just a fancy function.
* It takes numbers as inputs and produces numbers as outputs, just like any function from Calculus.
* Finding a good neural network that approximates your data is hard.

↳ We do this using lots of known (input, output) pairs.

Then we can use that good NN to predict new digits for us.

* Web applet demo

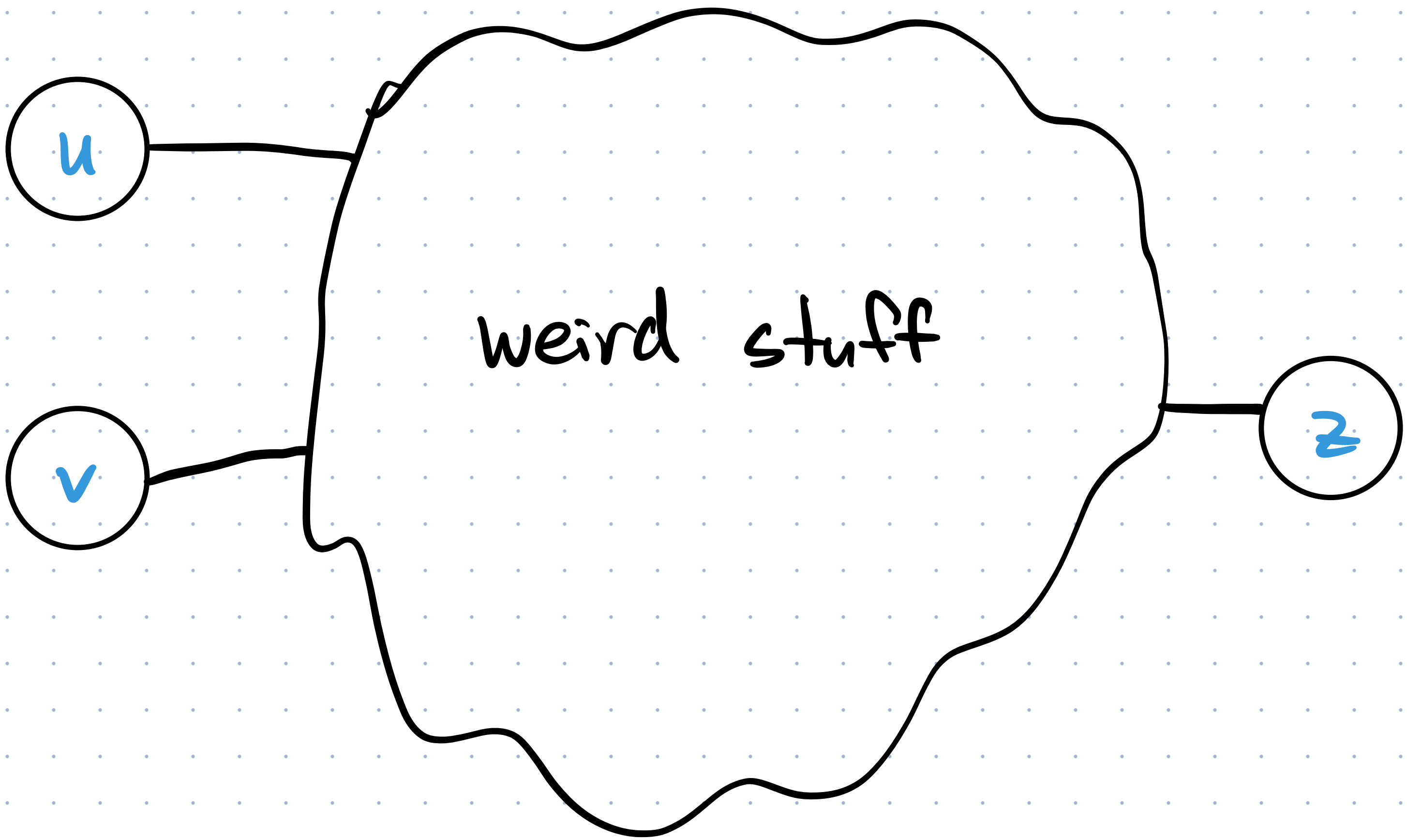
* Imagine if your job was to write an algorithm to look at the pixels manually & say what # it is! That would be so difficult.

What <u>is</u> a neural network?

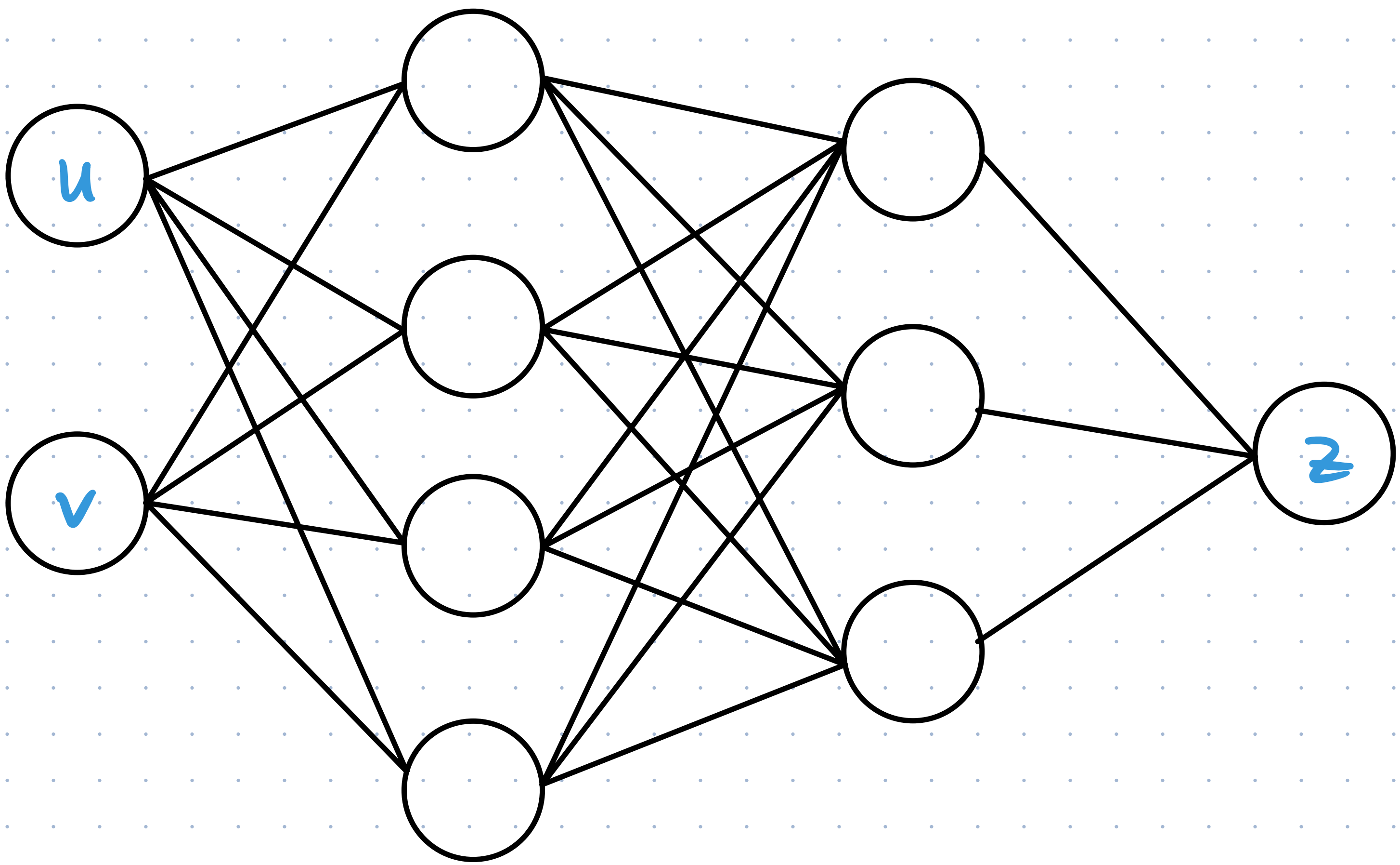
A fancy function defined in the following way.

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

$f(u,v) = z$

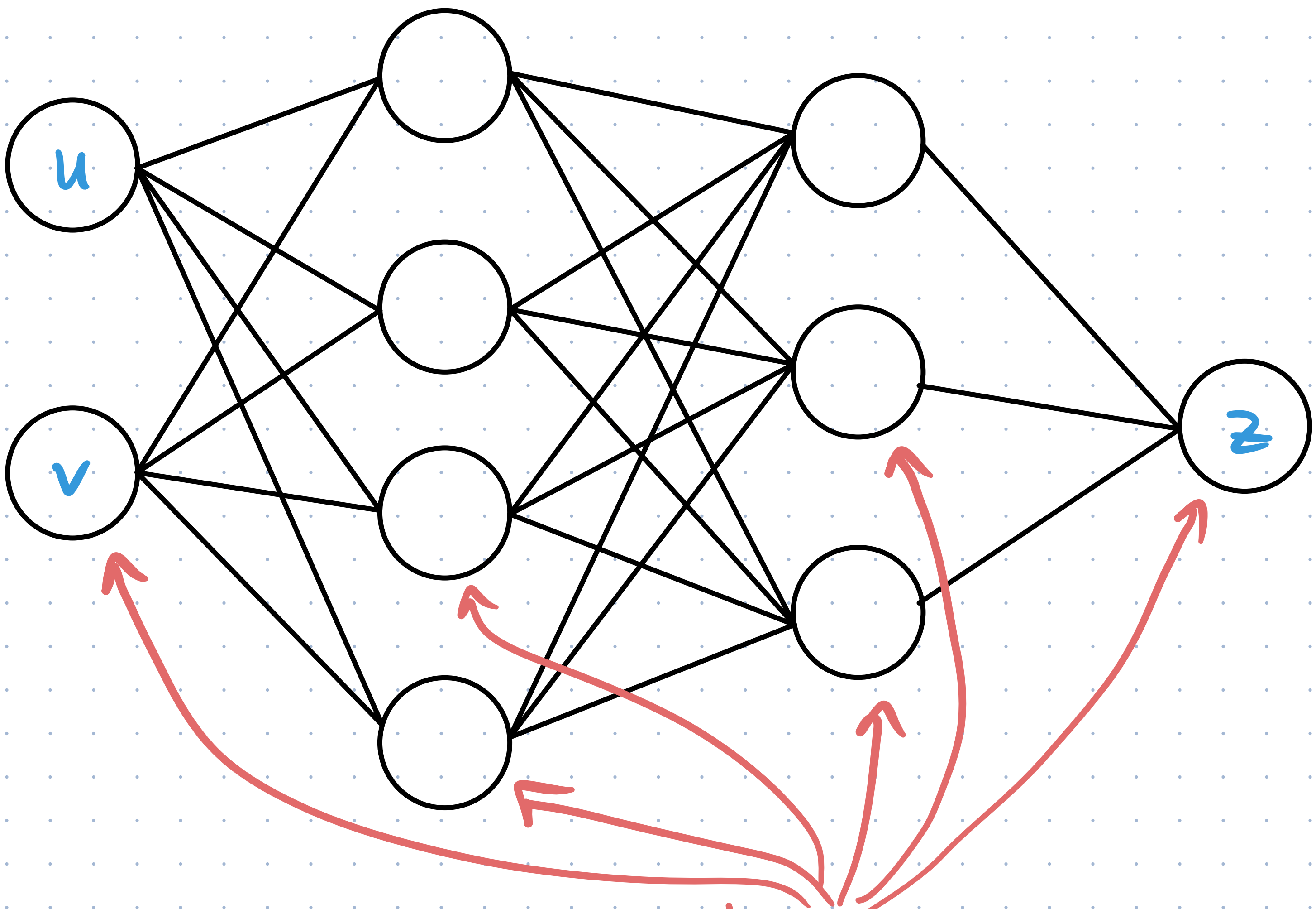


Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



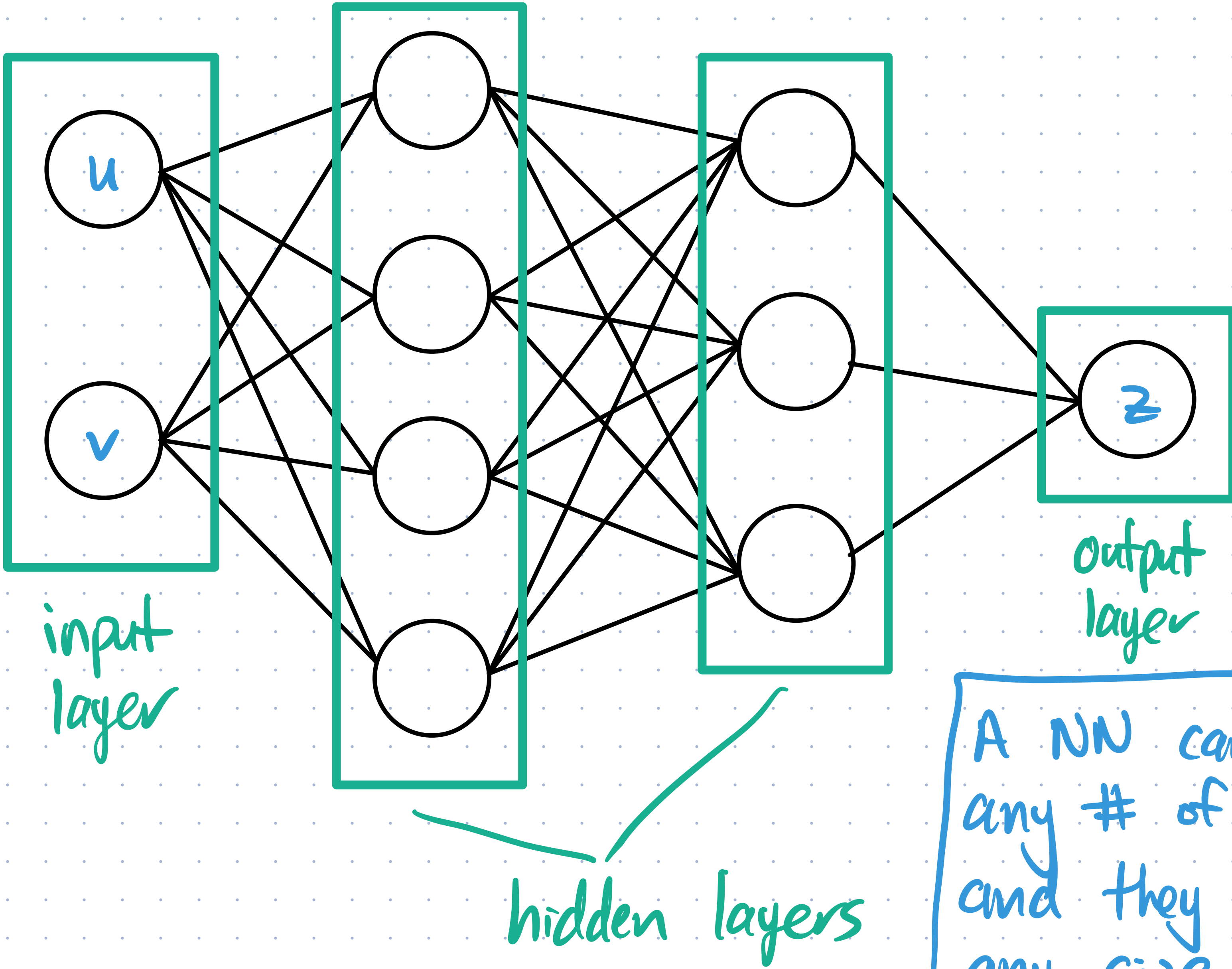
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



each circle is a "neuron"

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

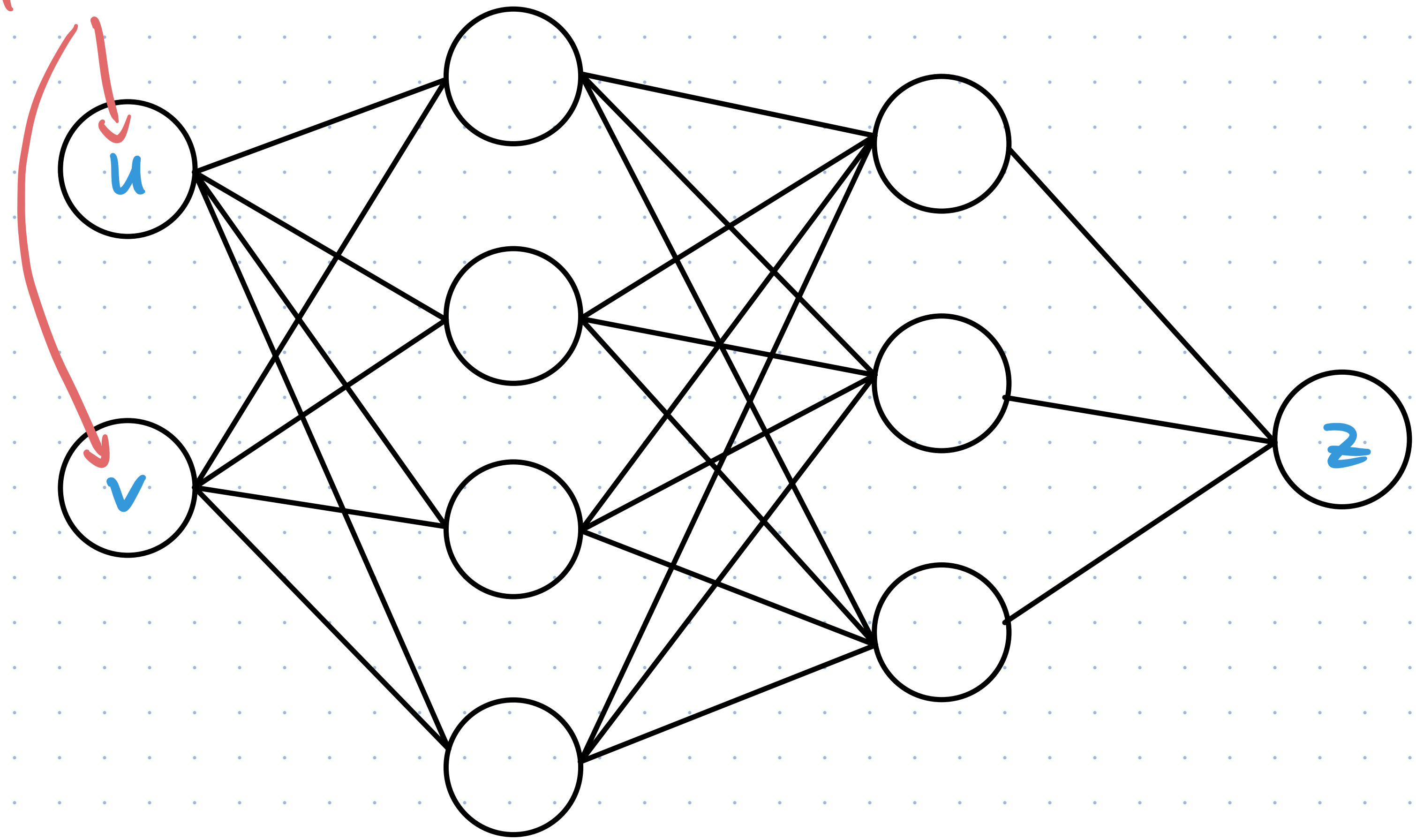


A NN can have any # of layers, and they can be any size.

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

$f(u,v) = z$

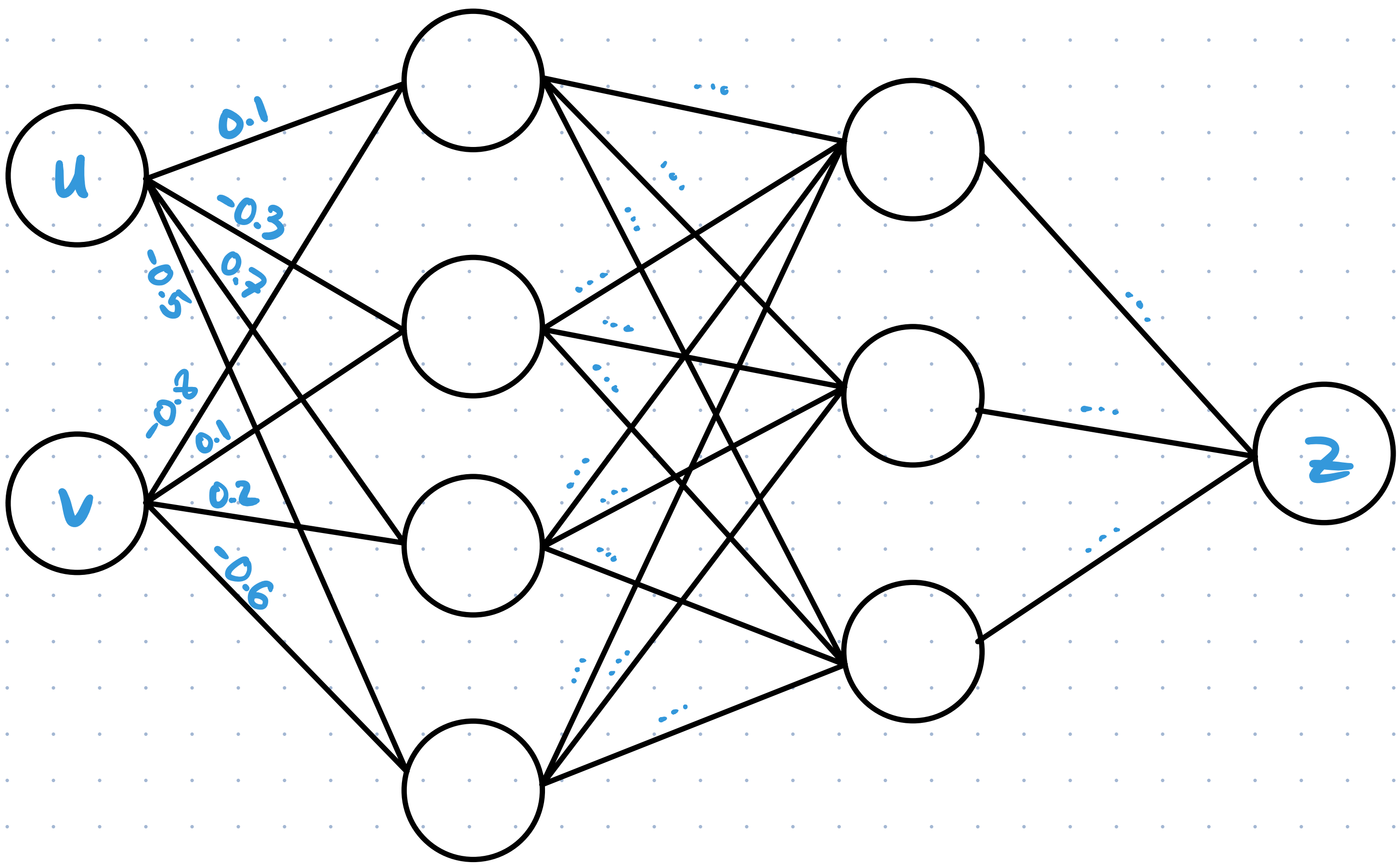
real #s (decimals)



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

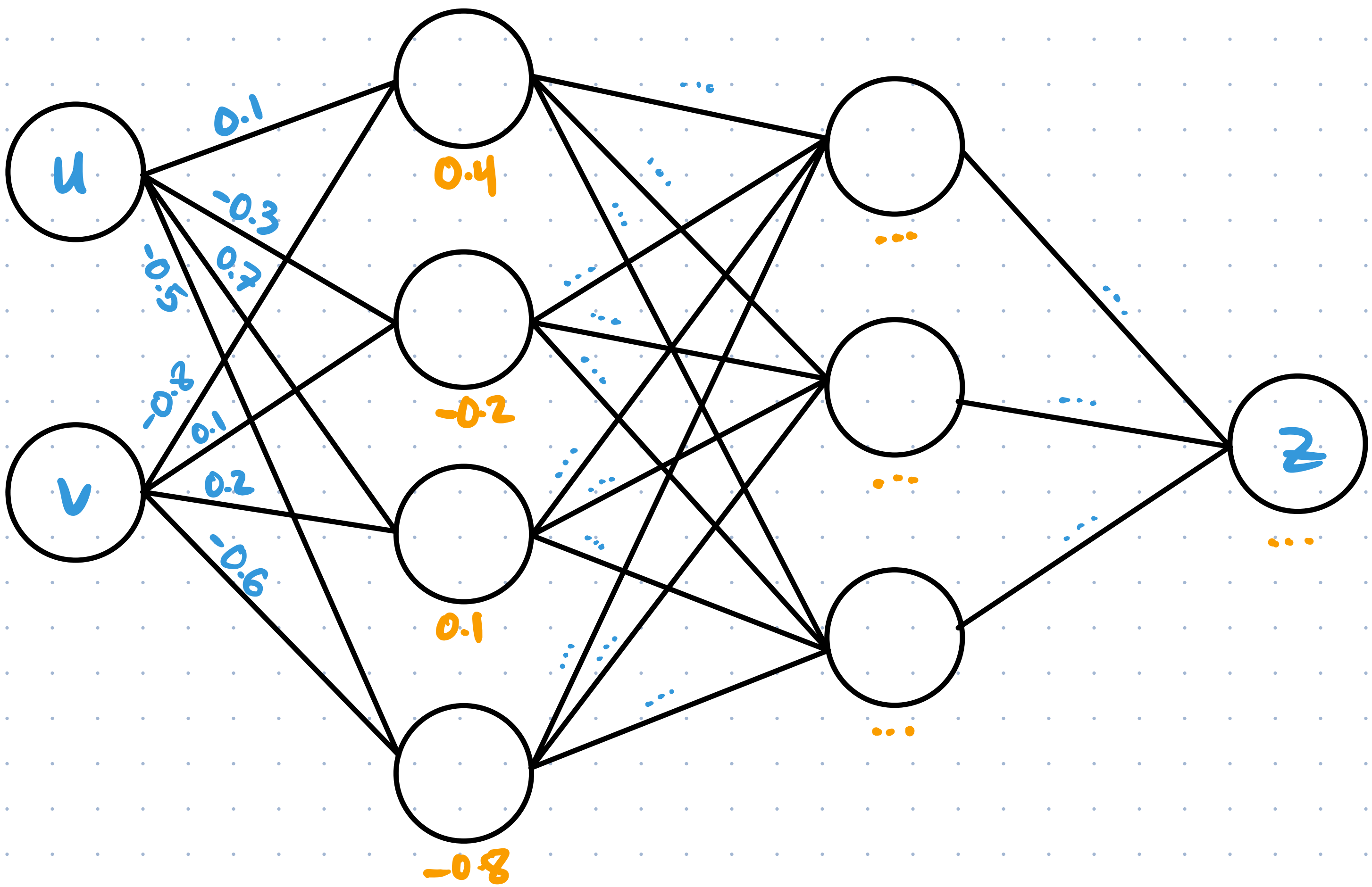
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



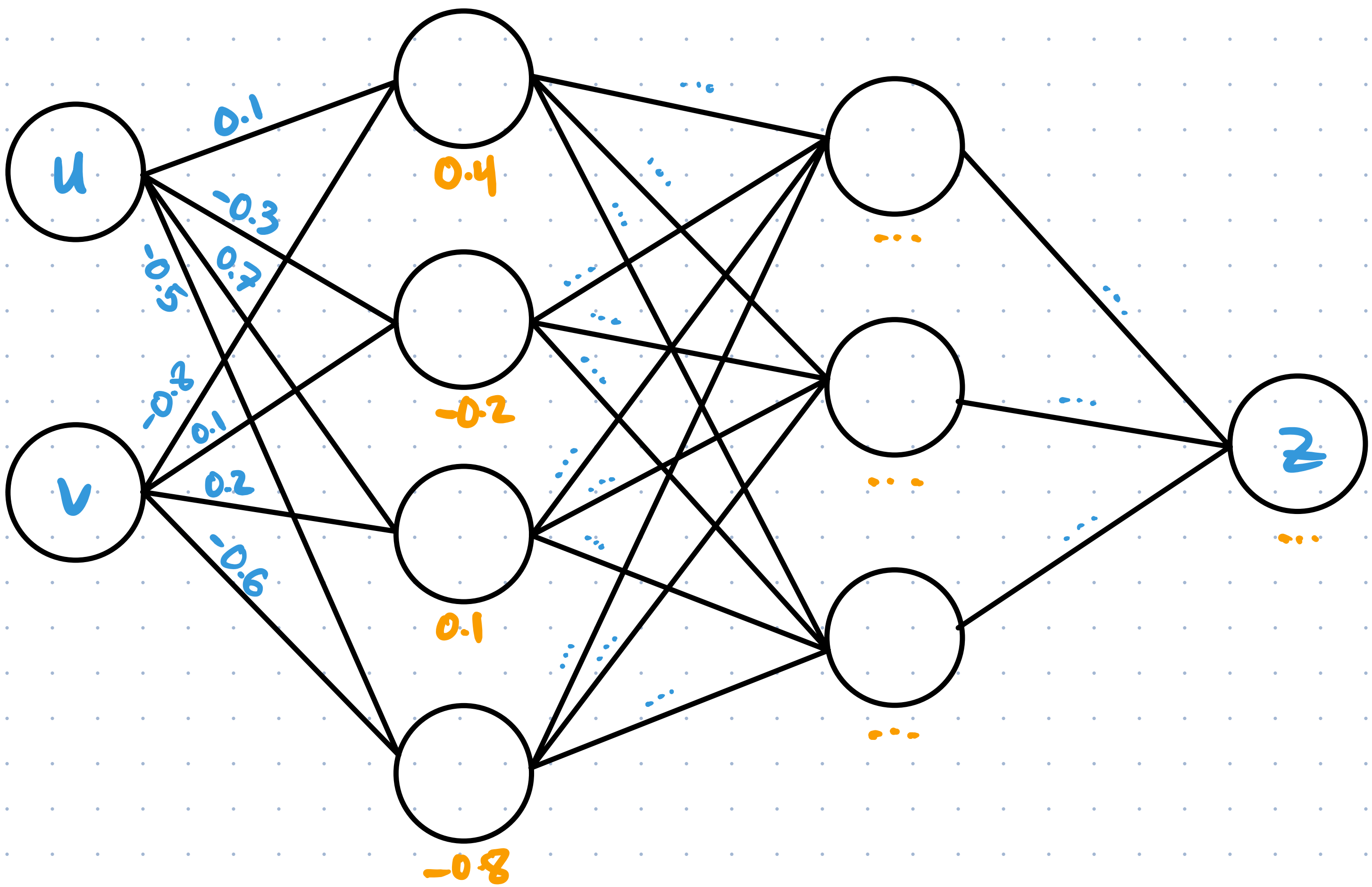
I'm using #s to 1 decimal place for simplicity, but typically the are full floating point #s.

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



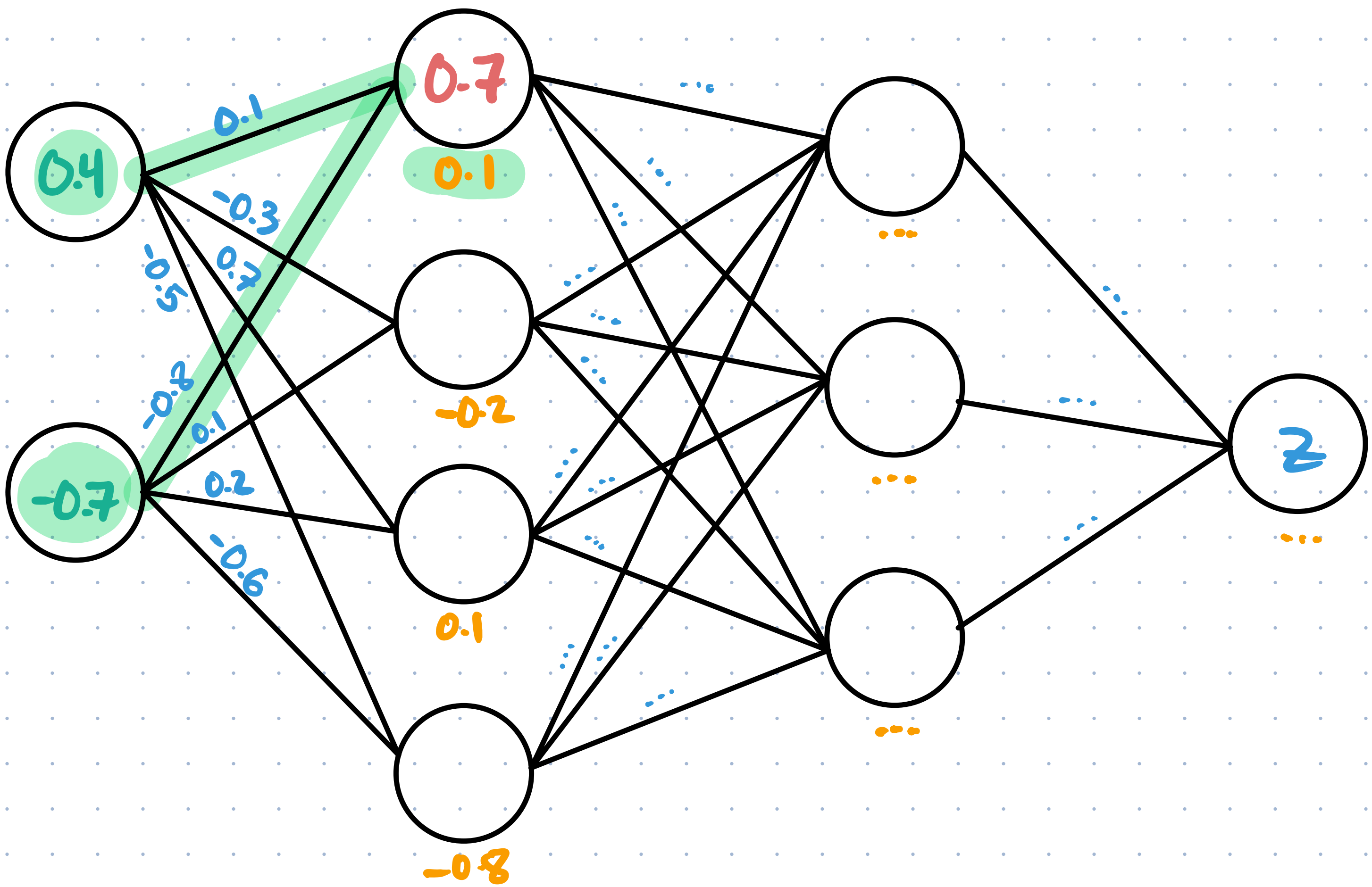
Every neuron (except the input layer) has a # attached to it called its **bias**.

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.

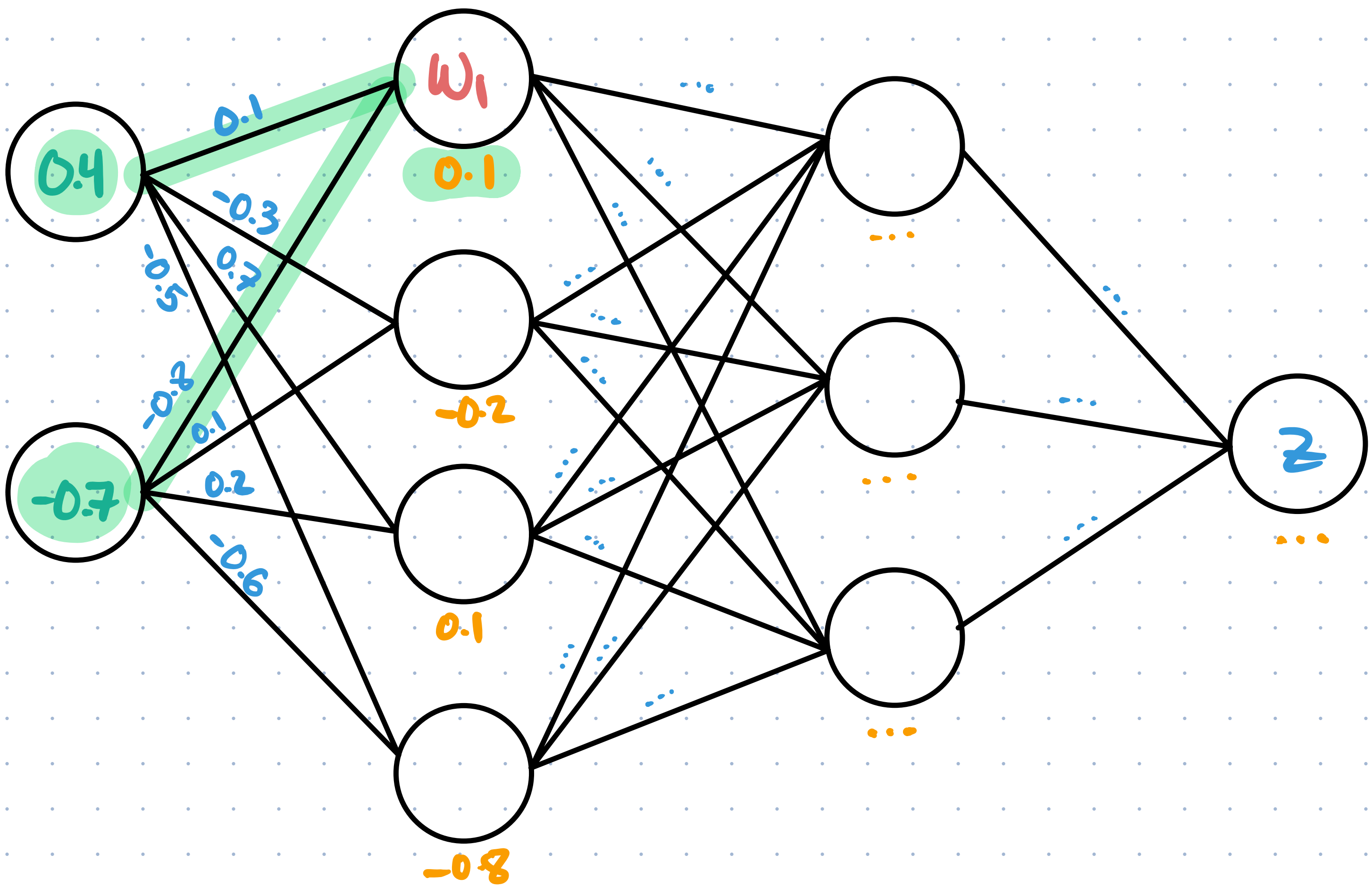
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



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

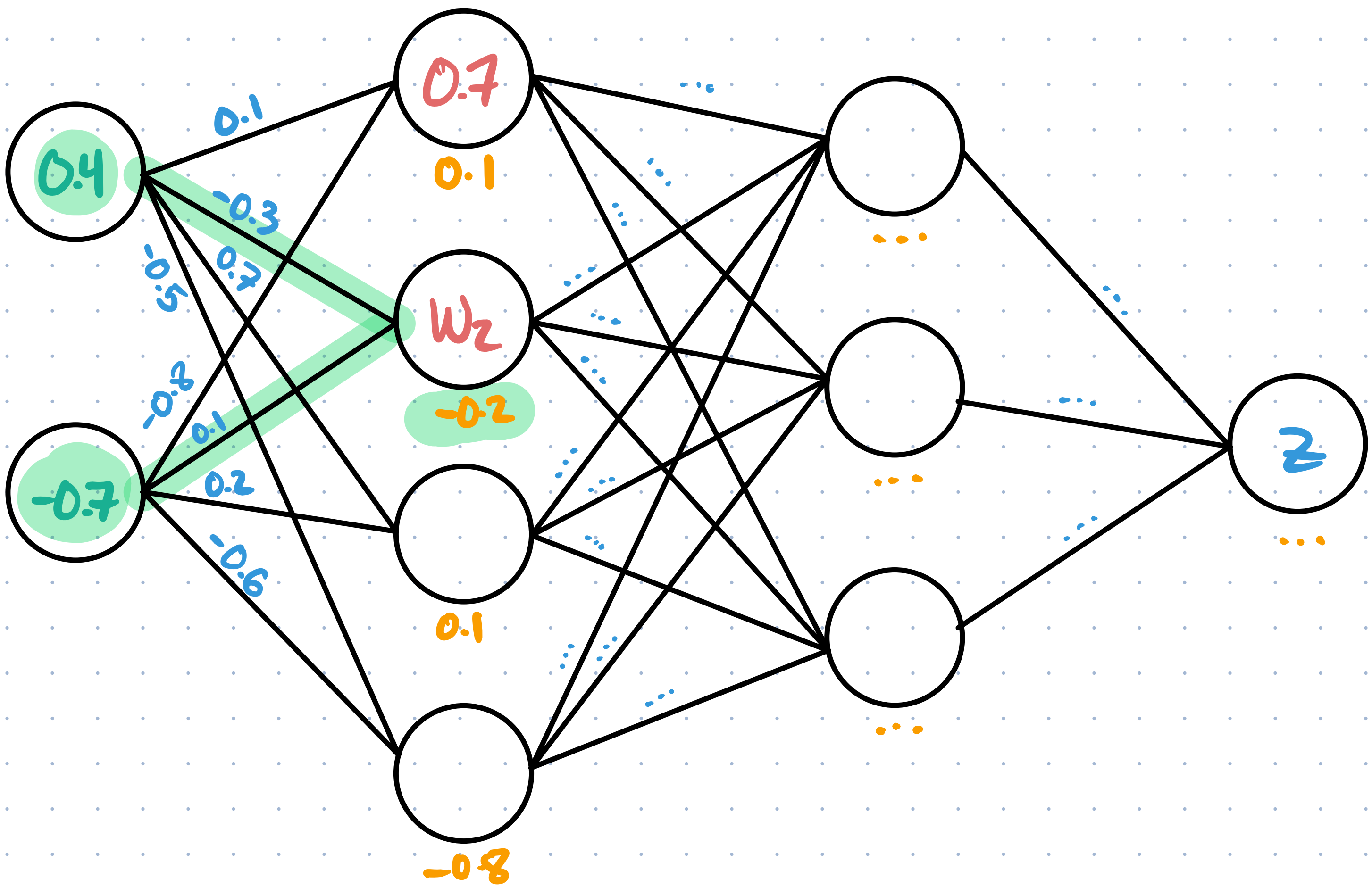
$f(u,v) = z$



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

$= 0.7$

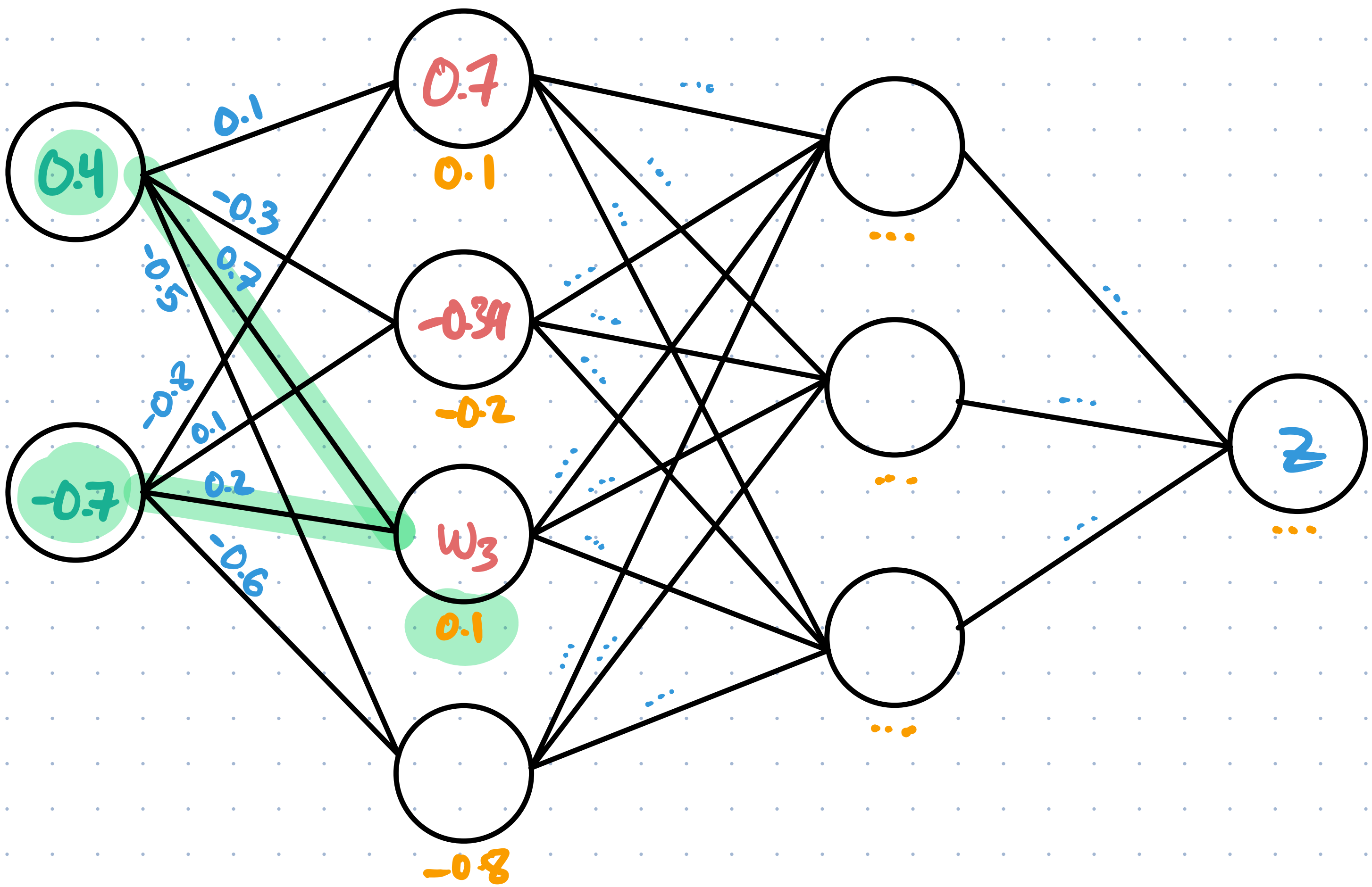
Example: A NN that takes 2 inputs and has 1 output $f(u,v) = z$



$$W_2 = (0.4)(-0.3) + (-0.7)(0.1) + -0.2$$
$$= -0.39$$

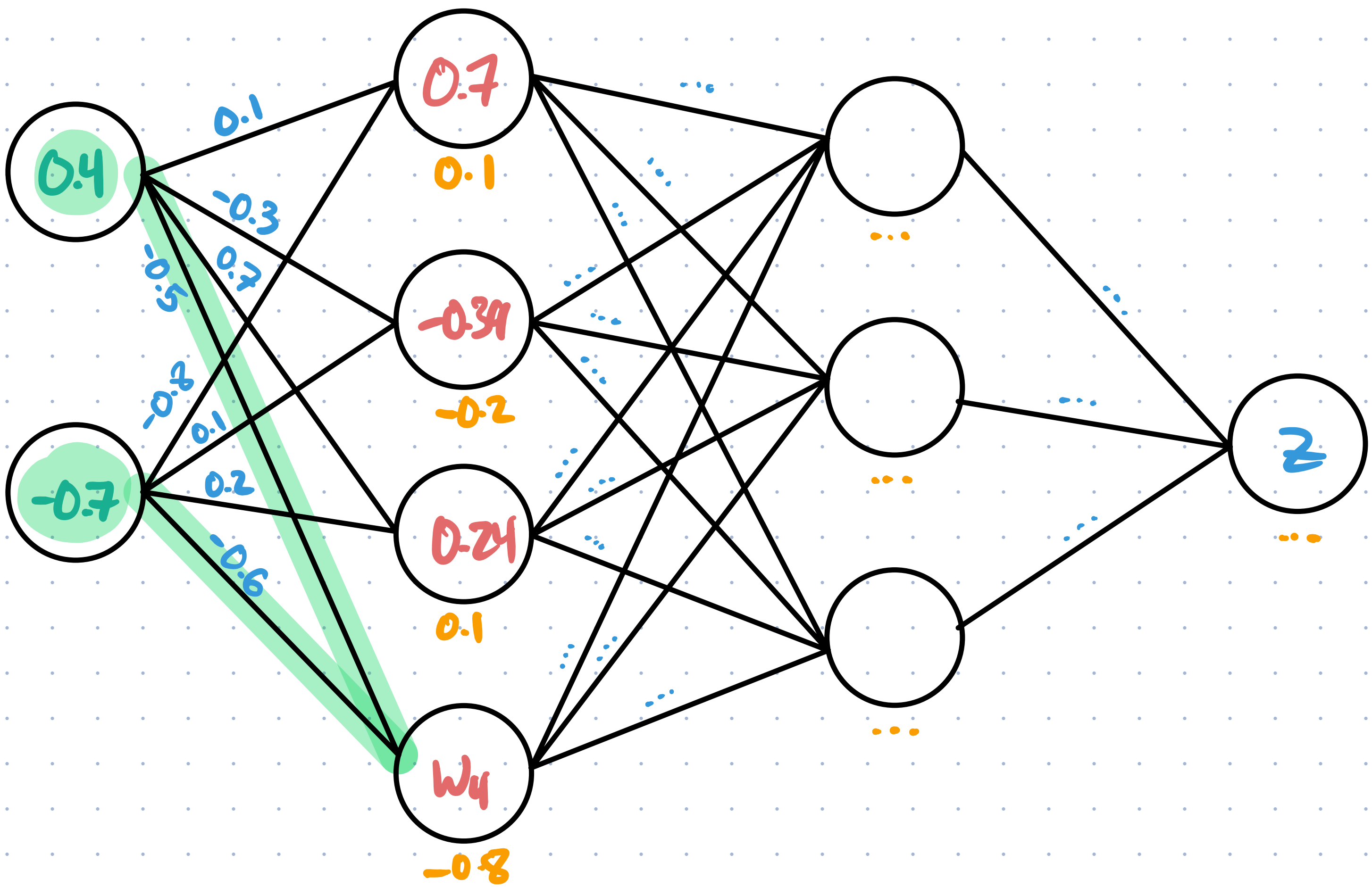
Example: A NN that takes 2 inputs and has 1 output

$f(u,v) = z$



$$w_3 = (0.4)(0.7) + (-0.7)(0.2) + 0.1$$
$$= 0.24$$

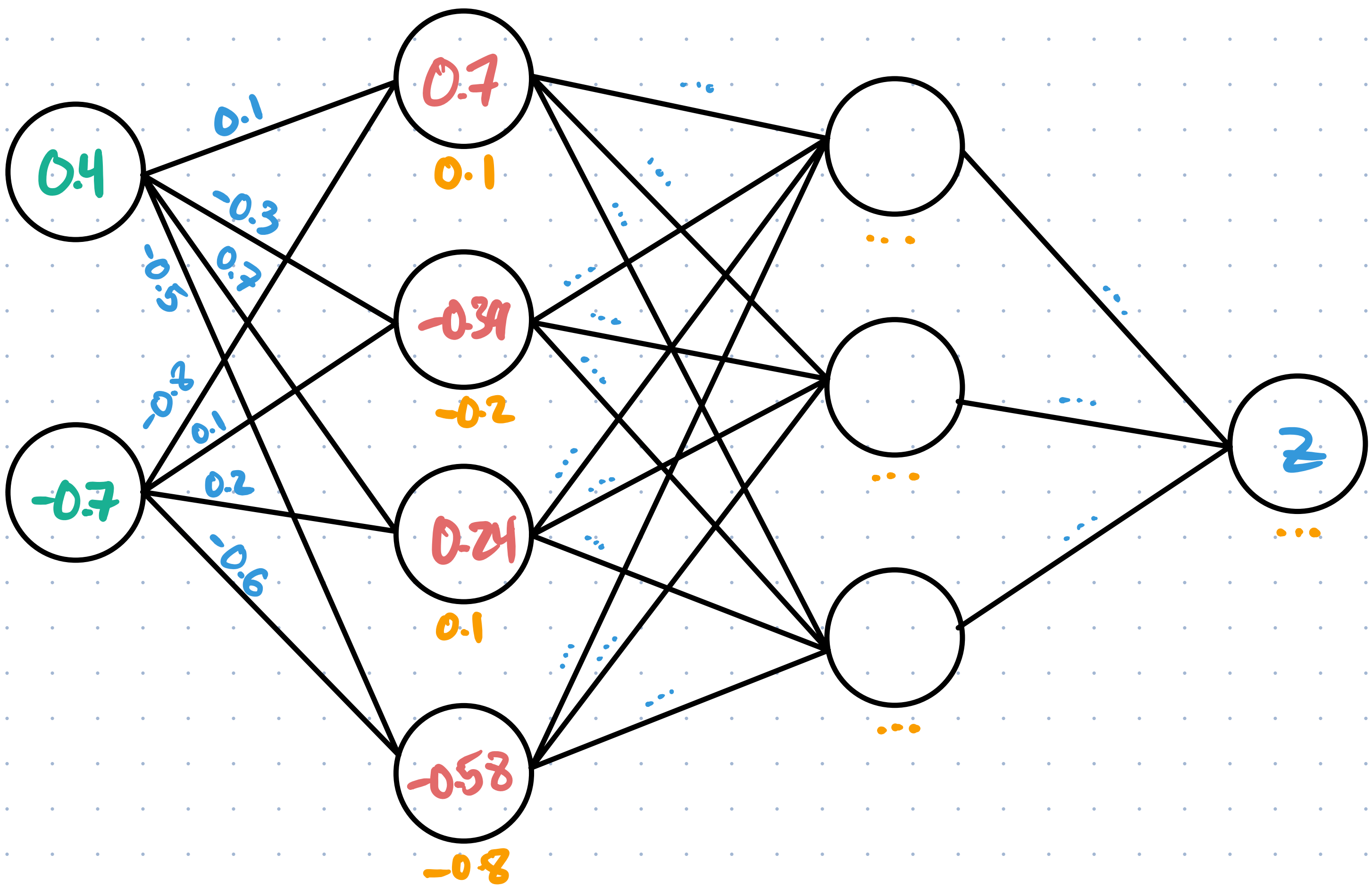
Example: A NN that takes 2 inputs and has 1 output $f(u,v) = z$



$$W_4 = (0.4)(-0.5) + (-0.7)(-0.6) + -0.8$$
$$= -0.58$$

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

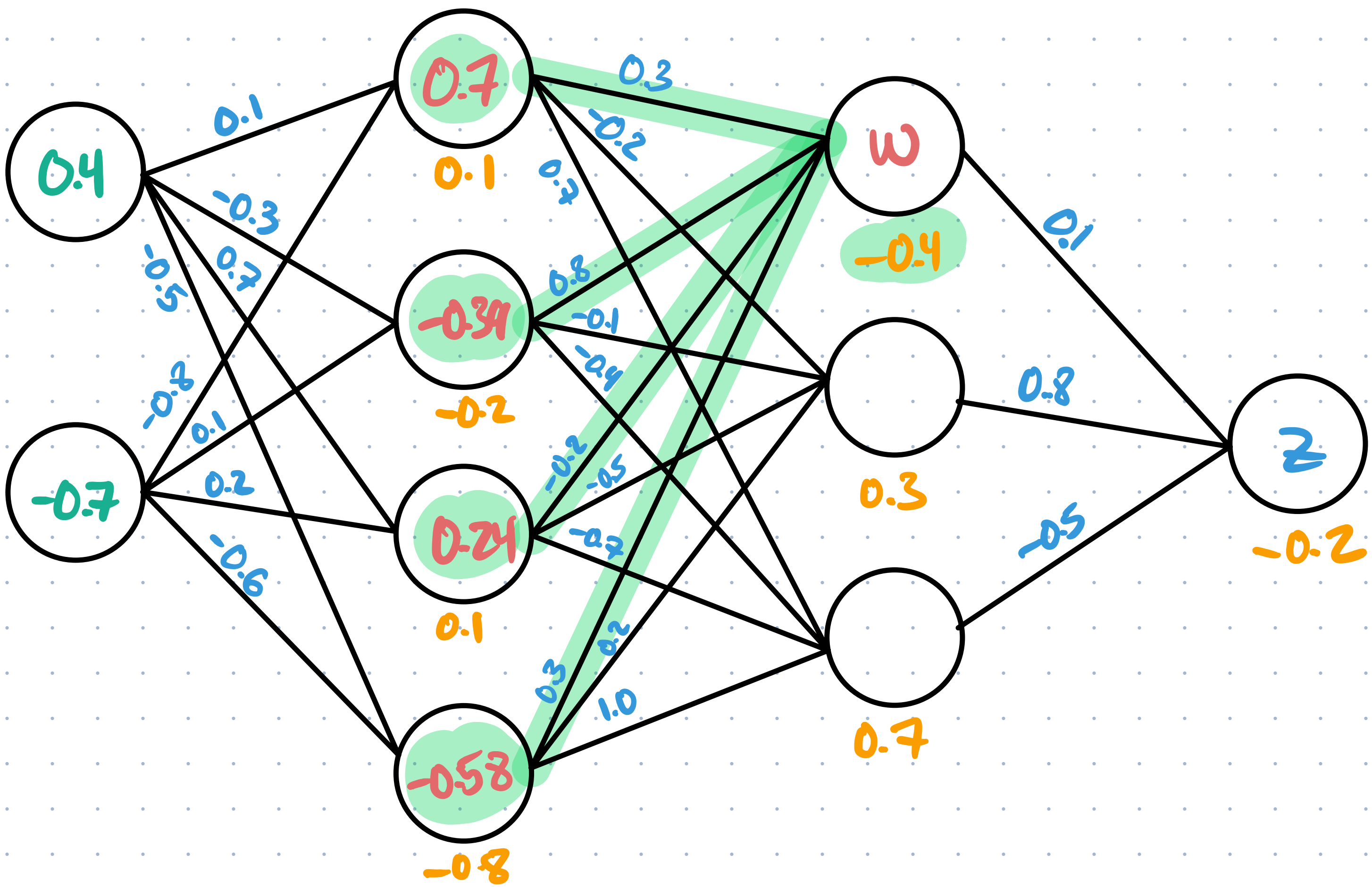
$f(u,v) = z$



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

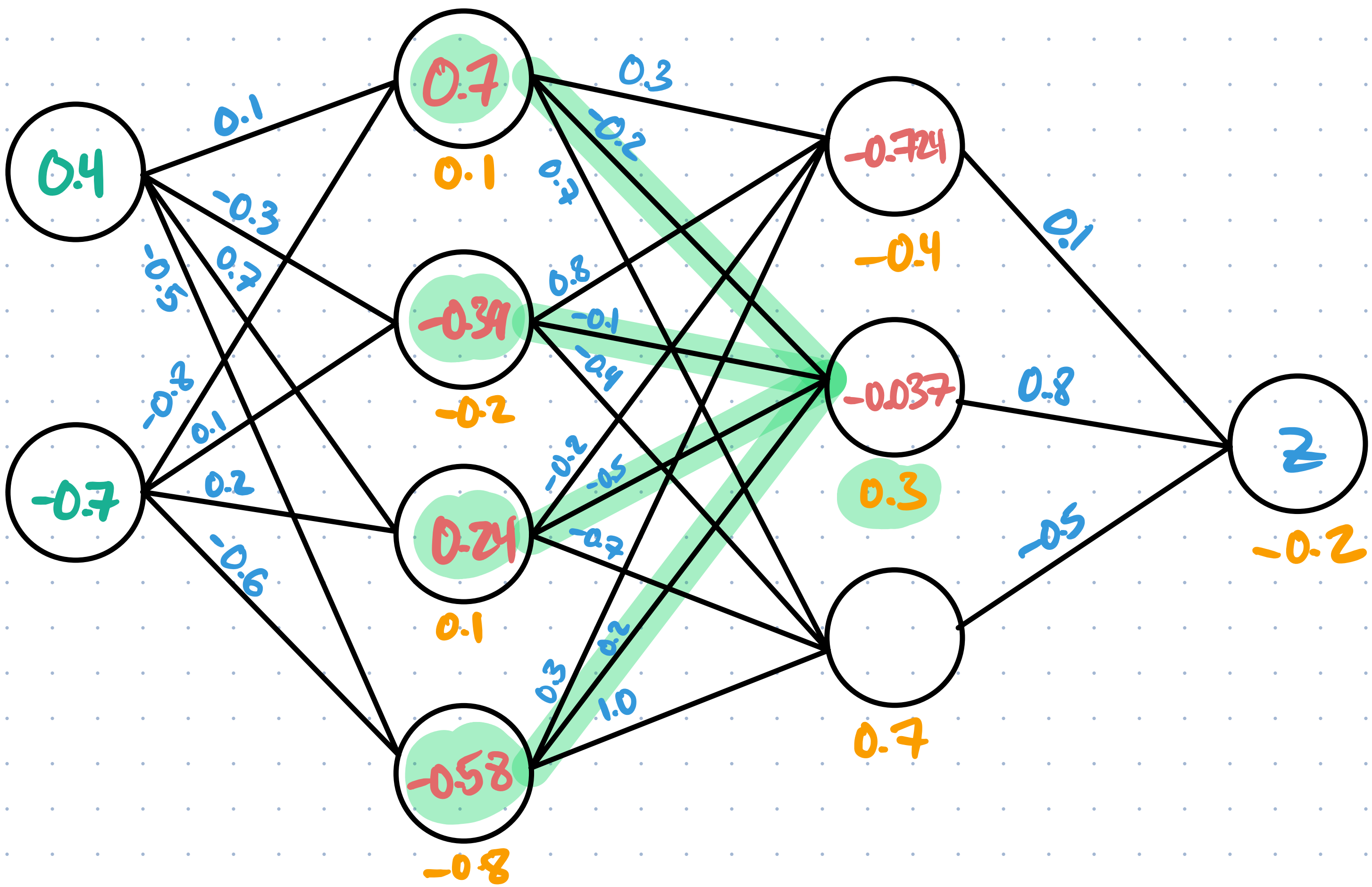
$f(u,v) = z$



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

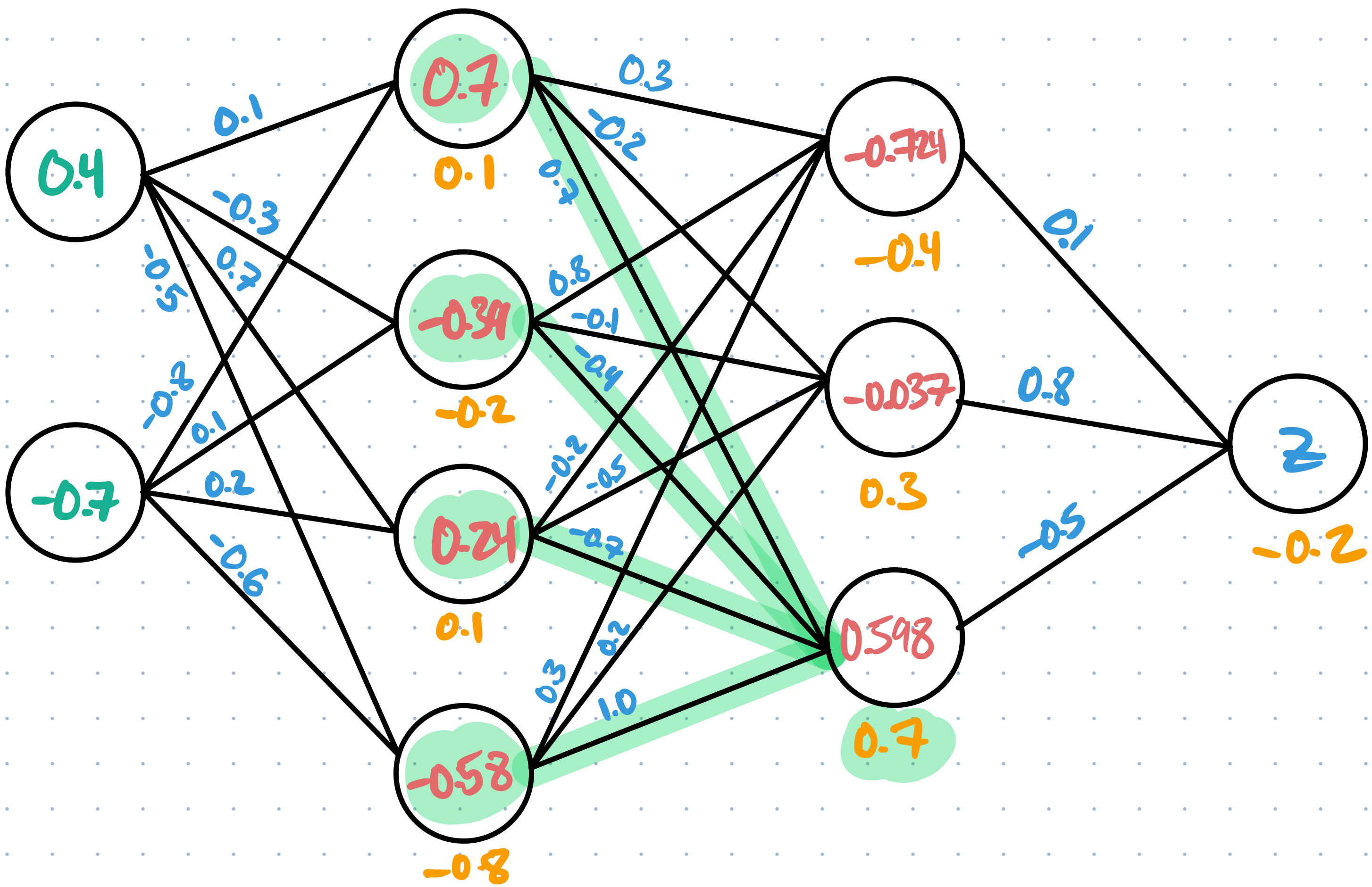
$= -0.724$

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

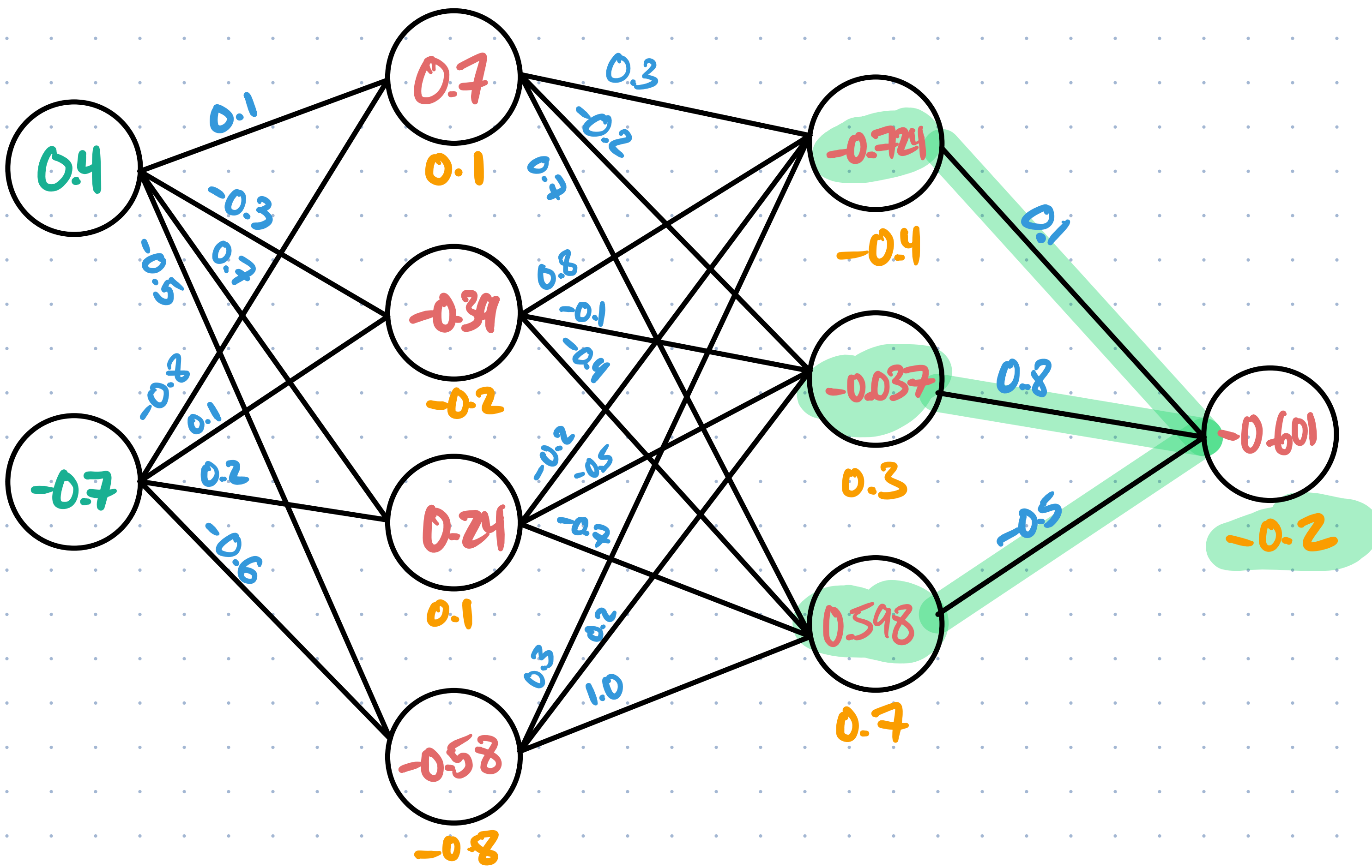


Example: A NN that takes 2 inputs and has 1 output

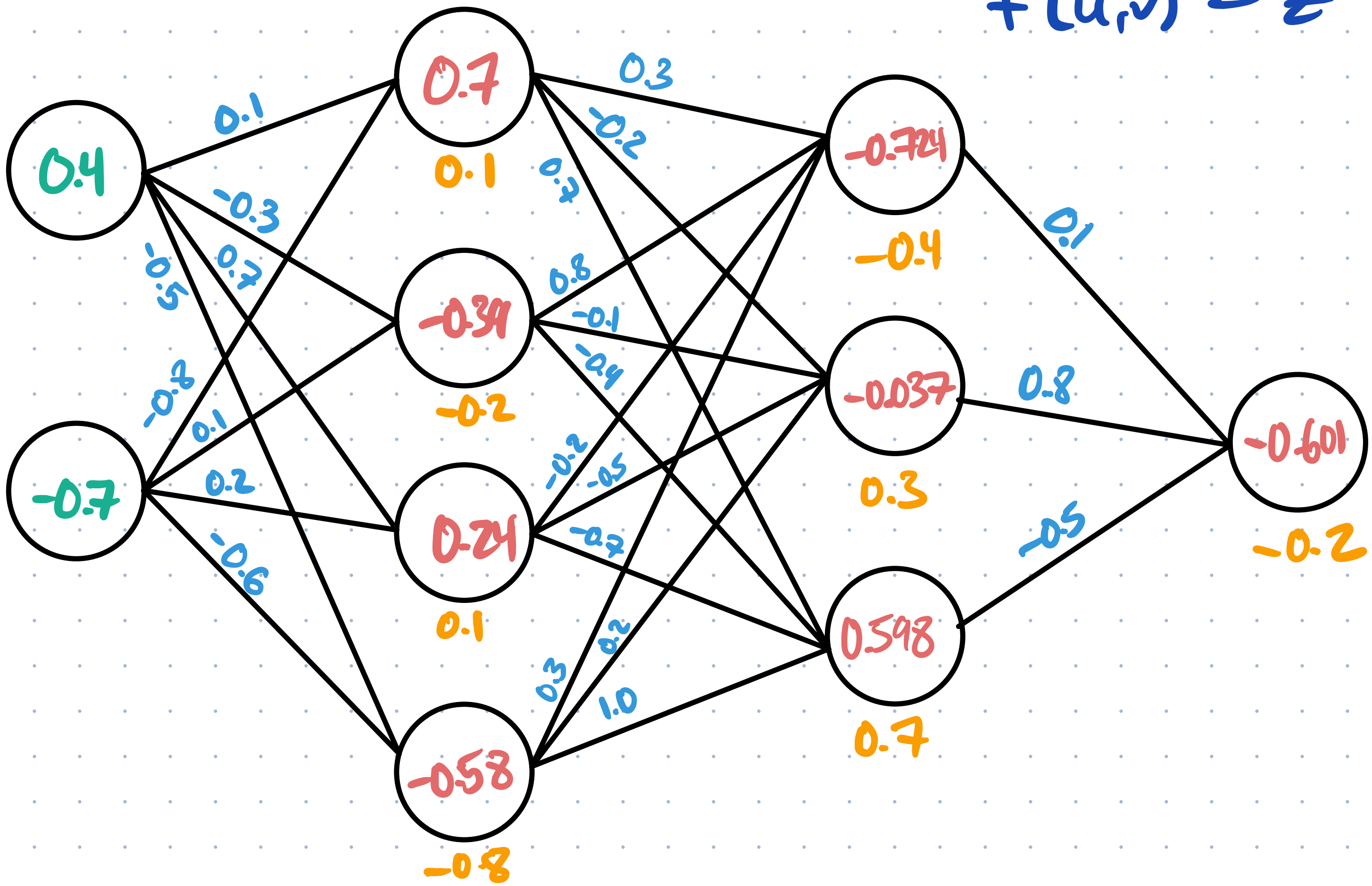
$f(u,v) = z$



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



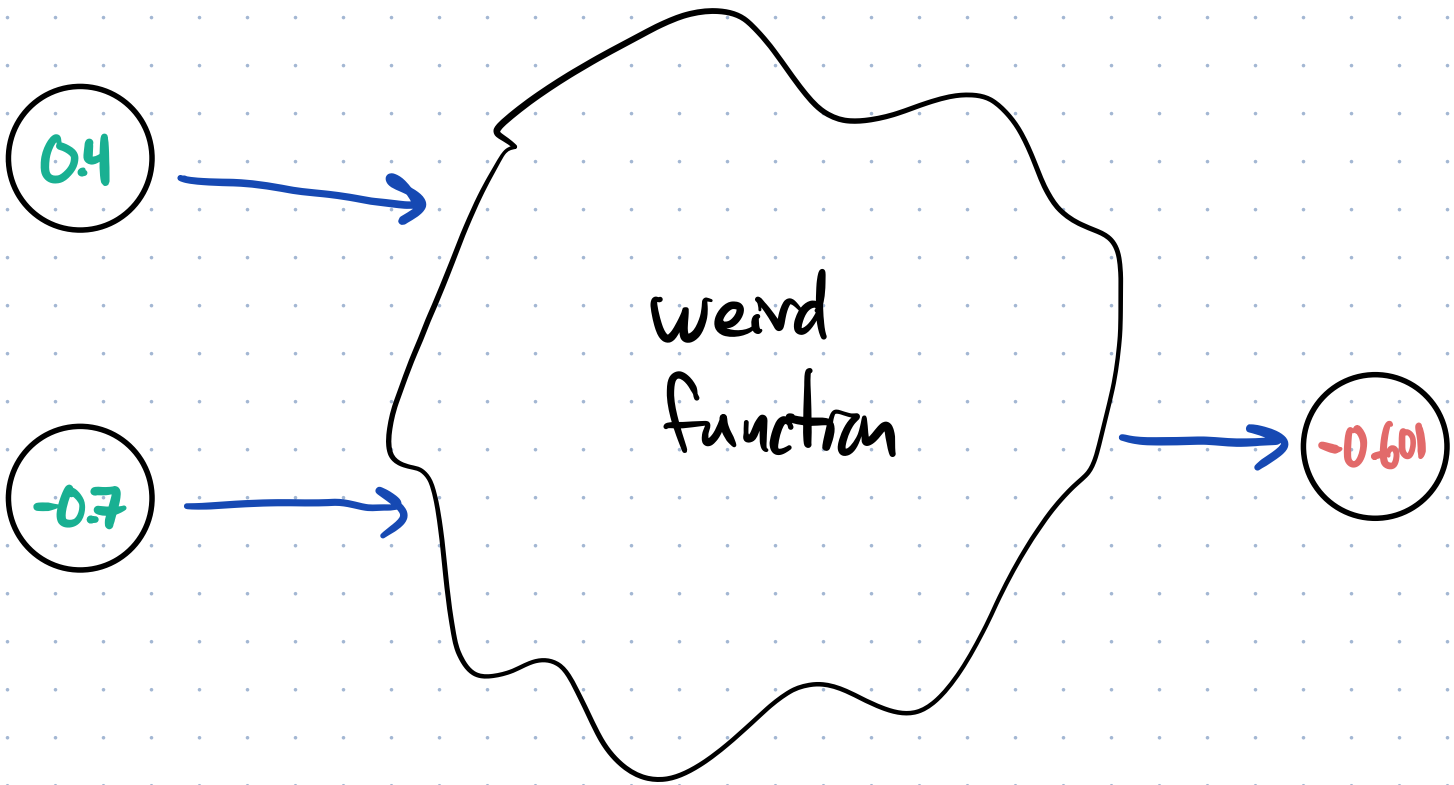
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



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

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

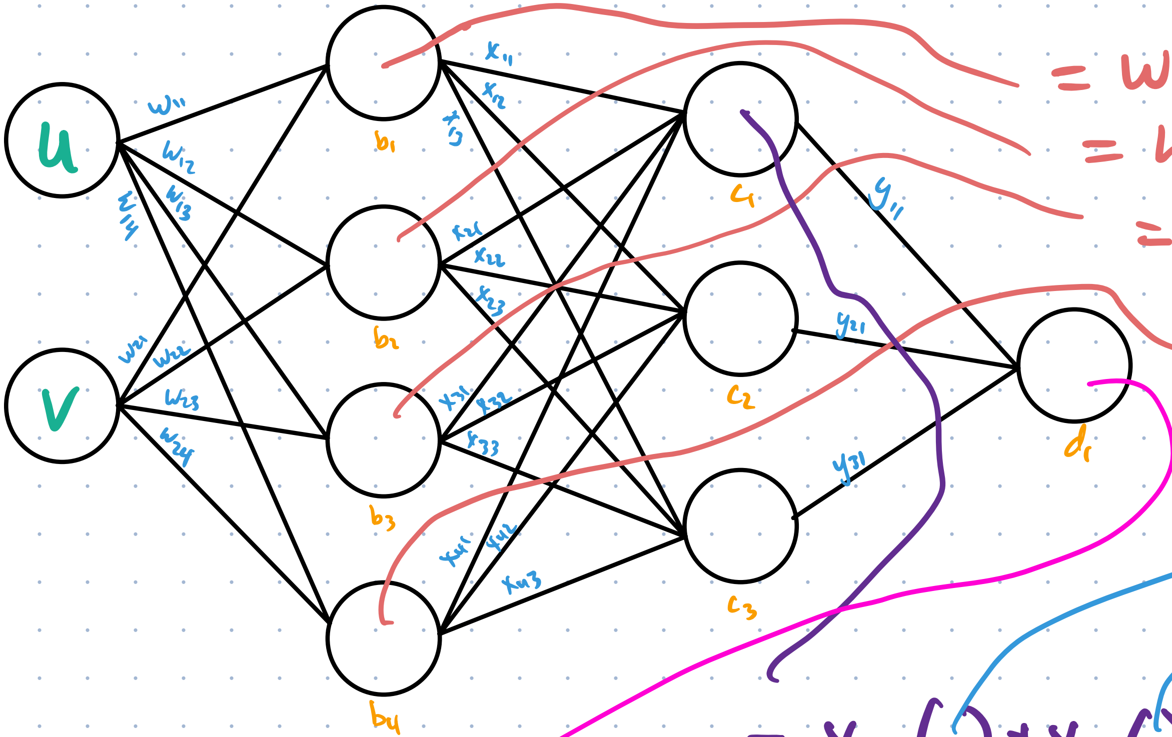
Example: A NN that takes 2 inputs and has 1 output
$f(u,v) = z$



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.



$$= w_{11}u + w_{21}v + b_1$$
$$= w_{12}u + w_{22}v + b_2$$
$$= w_{13}u + w_{23}v + b_3$$
$$= w_{14}u + w_{24}v + b_4$$

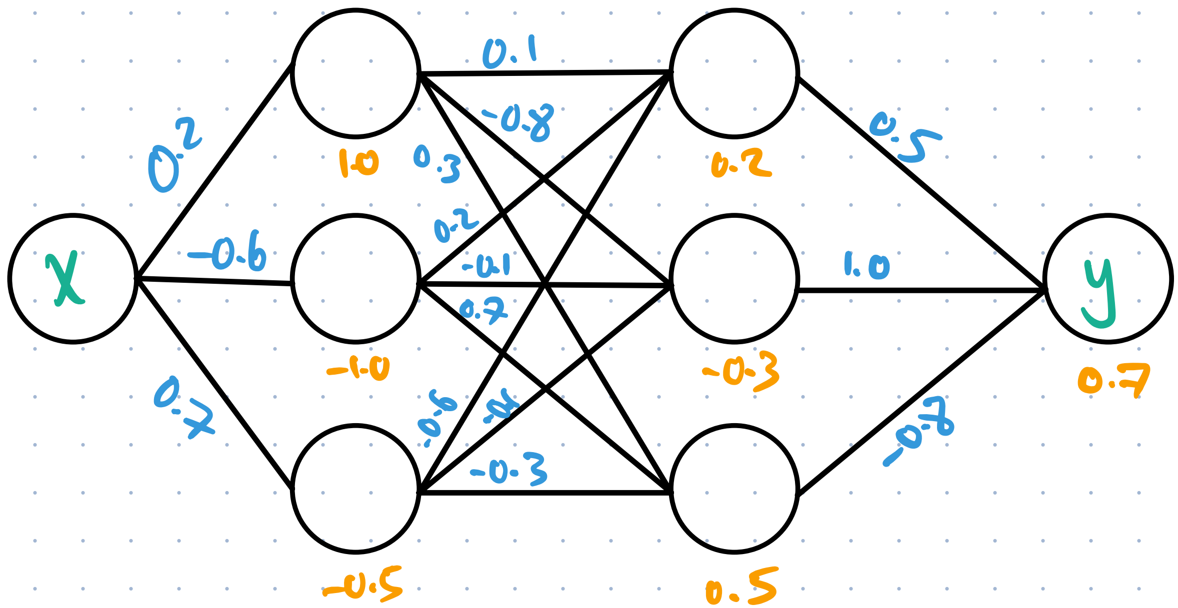
$$= x_{11}(\ ) + x_{21}(\ ) + x_{31}(\ ) + x_{41}(\ ) + c_1$$
$$= (x_{11}w_{11} + x_{21}w_{12} + x_{31}w_{13} + x_{41}w_{14}) \cdot u$$
$$+ (x_{11}w_{21} + x_{21}w_{22} + x_{31}w_{23} + x_{41}w_{24}) \cdot v$$
$$+ (b_1 + b_2 + b_3 + b_4)$$

=

```
(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
```

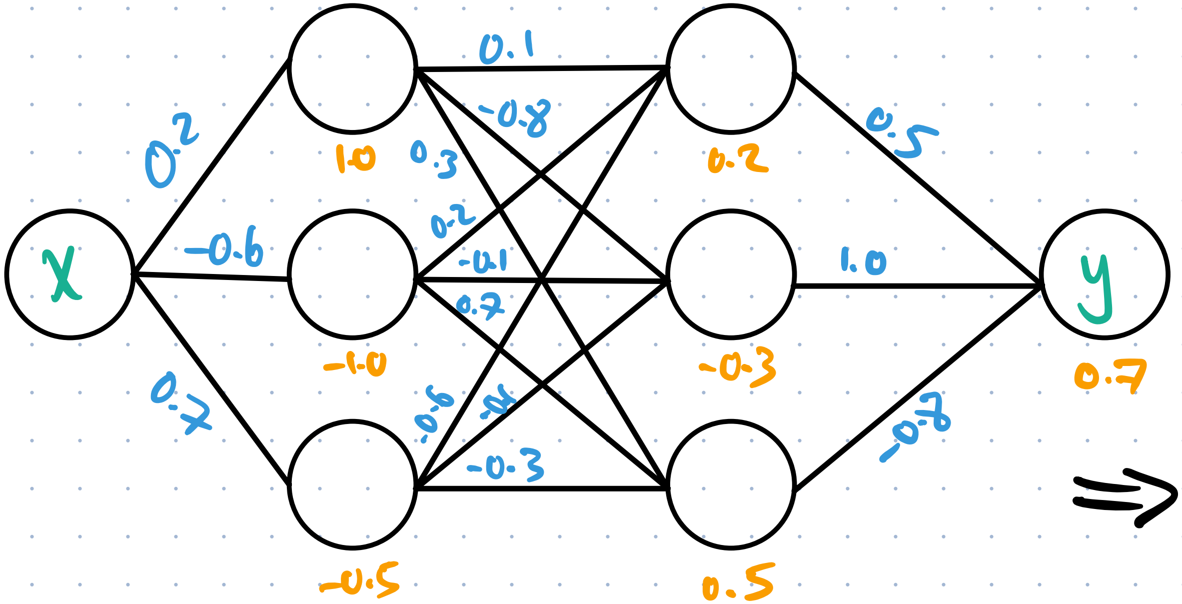
As we've built them NNs are still just (big) linear functions, so this is no better than linear regression.

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



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

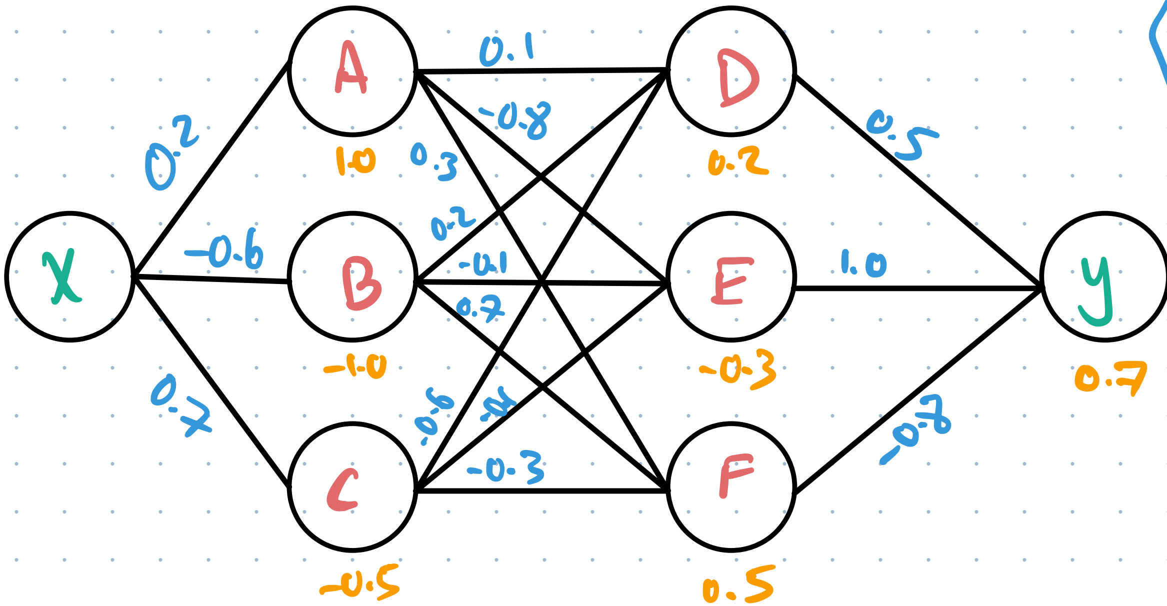
Just a line defined in a complicated way!



$\Rightarrow y = 0.026x - 0.25$

$(0.026)(0.3) - 0.25 =$
-0.2422 ✓

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



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

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

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

$D = 0.244$

$E = -1.001$

$F = 0.079$

$y = -0.2422$