a horse is a horse, of course

image classification using convolutional neural networks

presented by Katie Zaback

a mind-bending conundrum

oh dang, what a way to start!

a mind-bending conundrum

oh dang, what a way to start!

a mind-bending conundrum

oh dang, what a way to start!

horse or cactus?

a mind-bending conundrum

oh dang, what a way to start!

a mind-bending conundrum

oh dang, what a way to start!

horse or cactus?

a mind-bending conundrum

oh dang, what a way to start!

horse

cactus

did you get both of those wrong?

did you get both of those wrong?

you might be a computer.

mutually exclusive confusion

	human	computer
multiplying thousands of numbers together extremely quickly	uhhhh…you'll have to give me a second…	**i'm a genius**
horse or cactus?	**i'm a genius**	what's a horse?

computers will always be made of astounding electric efficiency

&

humans will always be made of miraculous image-classifying stardust

facing a harsh reality

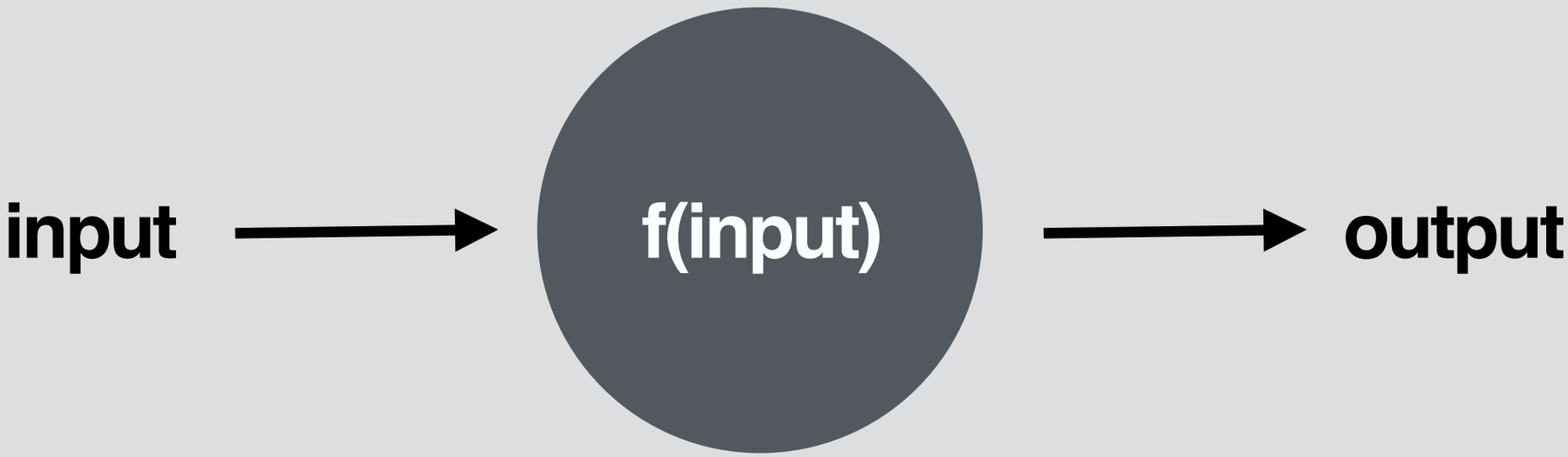
houston, we need an **algorithm.**

defining a problem space

????

that’s a horse

defining a problem space



defining a problem space

that’s a horse

f()

f()

f()

a horse has…

four legs and a long tail

two ears & two eyes

long nose

f()

a cactus has…

zero or more arms

sometimes a funny thing on top of its head

prickles/needles all over

f()

a horse has…

four legs and a long tail

two ears & two eyes

long nose

f()

a cactus has…

zero or more arms

sometimes a funny thing on top of its head

prickles/needles all over

f()

a horse has…

four legs and a long tail

two ears & two eyes

long nose

f()

a cactus has…

zero or more arms

sometimes a funny thing on top of its head

prickles/needles all over

how are horses like hard-core porn?

how are horses like hard-core porn?

In 1964, the U.S. Supreme Court was asked to determine whether a certain piece of questionable material could be classified as “obscene” and therefore not protected speech.

As part of their ruling, Justice Potter Stewart wrote:
“I shall not attempt to define the kinds of material I understand to be hard-core pornography… **but I know it when I see it.**”

how are horses like hard-core porn?

In 1964, the U.S. Supreme Court was asked to determine whether a certain piece of questionable material could be classified as “obscene” and therefore not protected speech.

As part of their ruling, Justice Potter Stewart wrote:
“I shall not attempt to define the kinds of material I understand to be hard-core pornography… **but I know it when I see it.**”

horse? no.

horse? yes.

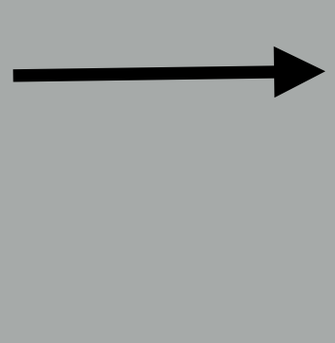
problem:

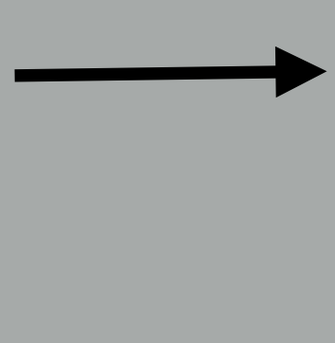
we don’t know what exactly the function should look like, but we know what the function should return given the input

that’s a horse

problem:

we don’t know what exactly the function should look like, but we know what the function should return given the input





that’s NOT

problem:

we don’t know what exactly the function should look like, but we know what the function should return given the input

that’s NOT

solution?

problem:

we don’t know what exactly the function should look like, but we know what the function should return given the input

that’s NOT

solution?

neural networks, y’all.

problem:

we don’t know what exactly the function should look like, but we know what the function should return given the input

that’s NOT

solution?

neural networks, y’all.

why?

problem:

we don’t know what exactly the function should look like, but we know what the function should return given the input

that’s NOT

solution?

neural networks, y’all.
why? they’re great at **function approximation** and are a **supervised learning algorithm.**

welcome to…

welcome to…

everything you need to know about neural networks

welcome to…

everything you need to know about neural networks

that i could fit into a forty-minute presentation

welcome to…

everything you need to know about neural networks

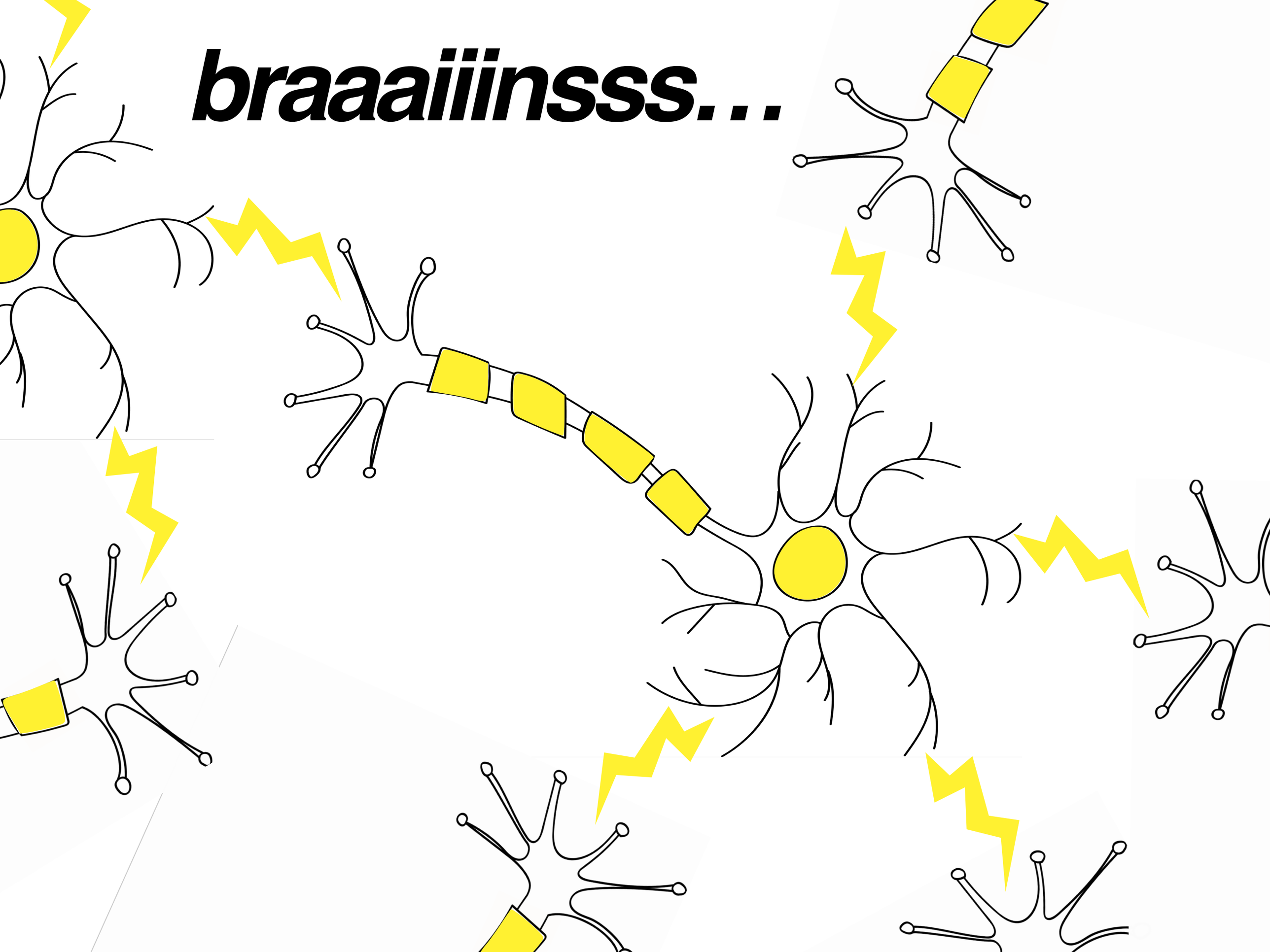
that i could fit into a forty-minute presentation

sorry there's gonna be math :(

a brief foray into some NN history

great scott!

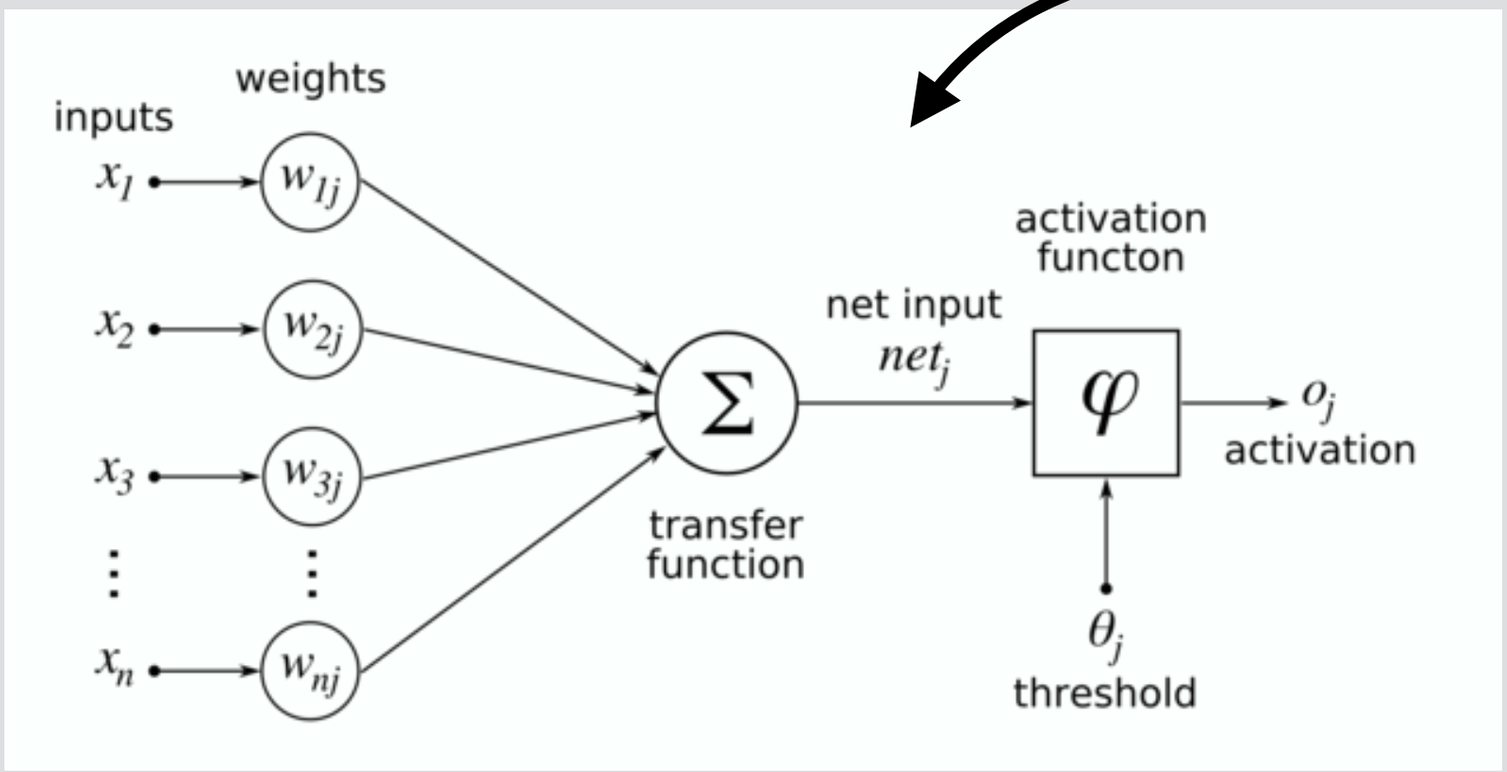
braaaiiinsss...



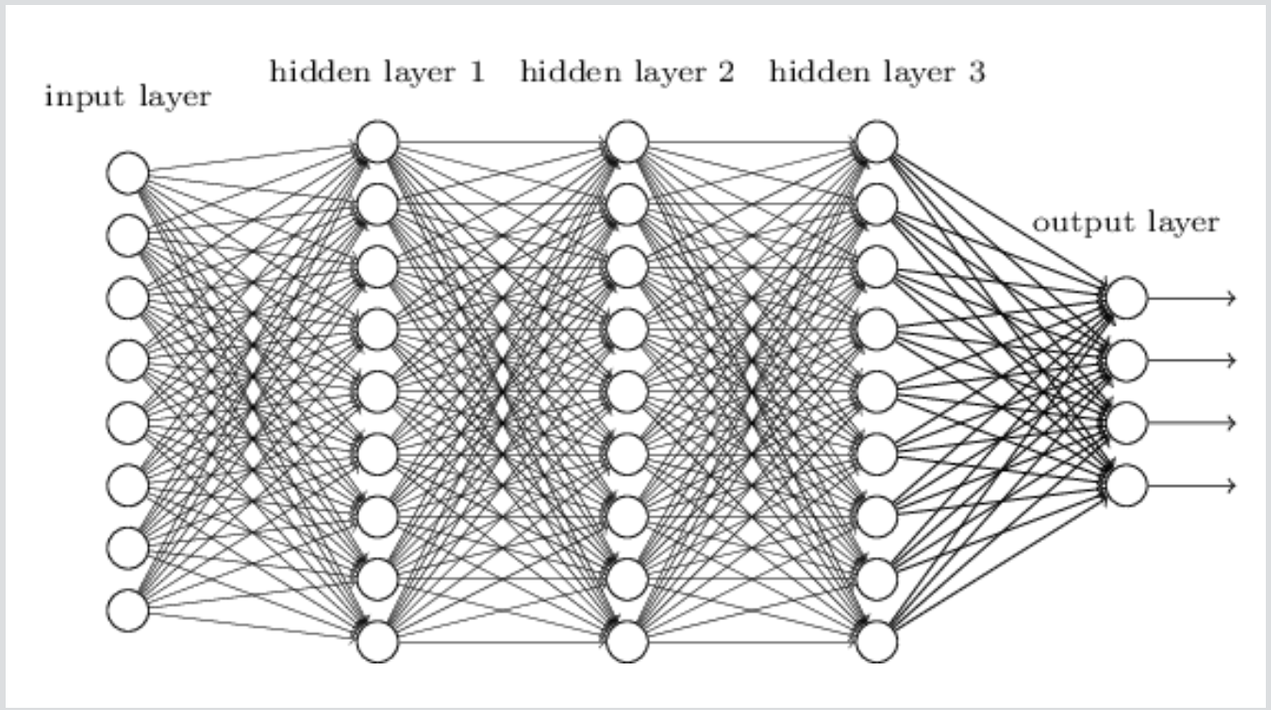
let's make a computer brain

frank rosenblatt (1957)

perceptron



let's make a computer brain



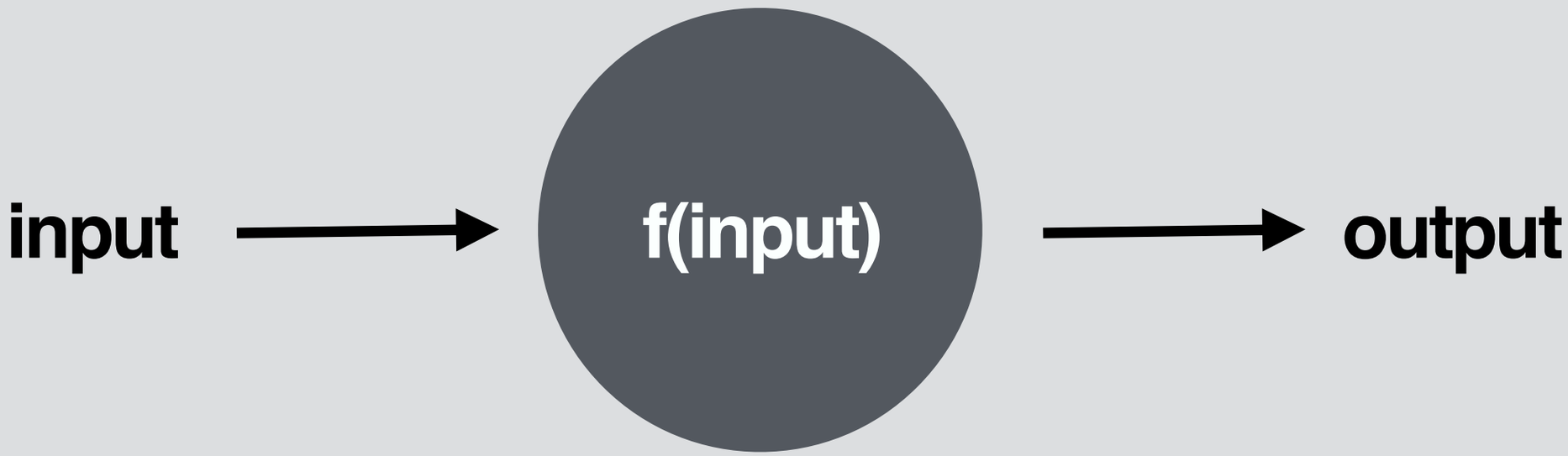
As computing power increased (and big data was a reality) NNs began being used to compute **increasingly complex functions**. They are essentially **chains and chains of perceptrons**, assembled to create deep nets that can used supervised learning methods to **approximate all kinds of functions** that are difficult to specify

There are many different *types* and *architectures* of neural networks.

NNs can be structured differently in order to optimize for different types of input or output data. It's easy to get lost in the weeds…

Remember:

the goal of this process is to come up with a function that, given some input, produces the correct (desired) output



forget horses for a second.

horses are hard.

let's start simple.

let's try & predict the future!

yearly income? $10,500
hates puppies? no

yearly income? $34,700
hates puppies? yes

yearly income? $60,400
hates puppies? no

yearly income? $2,300
hates puppies? yes

yearly income? $10,500
hates puppies? no

yearly income? $34,700
hates puppies? yes

will their next purchase be a

Mac or PC?

yearly income? $60,400
hates puppies? no

yearly income? $2,300
hates puppies? yes

yearly income? $10,500
hates puppies? no
next purchase: Mac

yearly income? $10,500
hates puppies? no
next purchase: Mac

yearly income? $10,50
hates puppies? no
next purchase: Mac

yearly income? $34,700
hates puppies? yes
next purchase: PC

yearly income? $34,700
hates puppies? yes
next purchase: PC

yearly income? $34,70
hates puppies? yes
next purchase: PC

yearly income? $60,400
hates puppies? no
next purchase: Mac

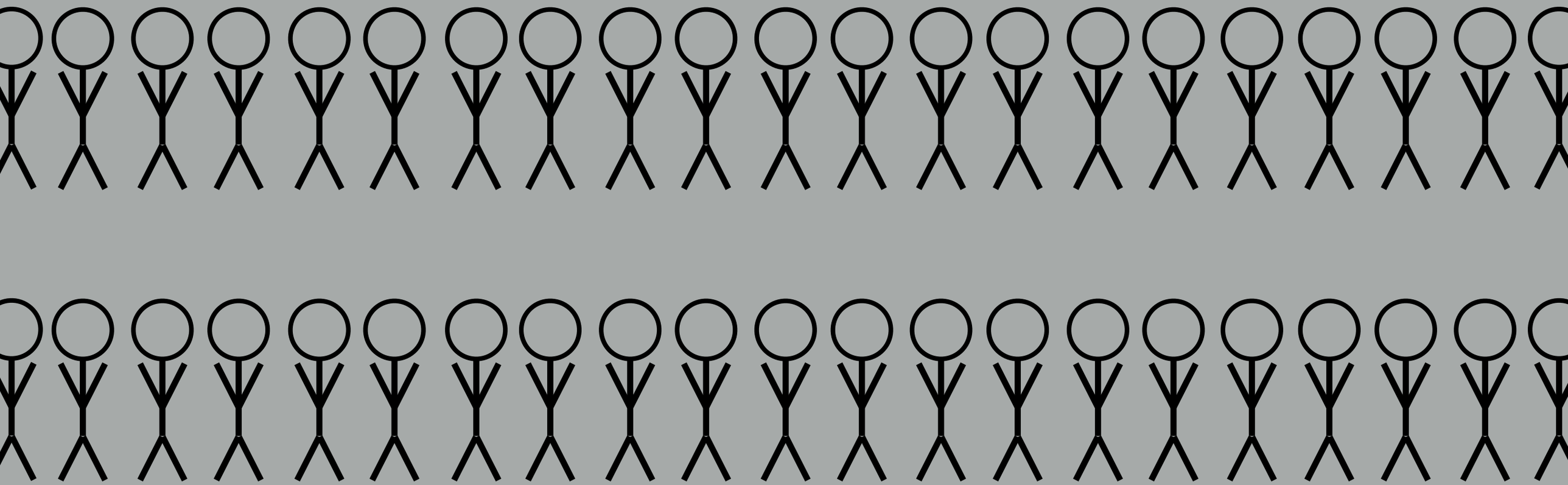
yearly income? $60,400
hates puppies? no
next purchase: Mac

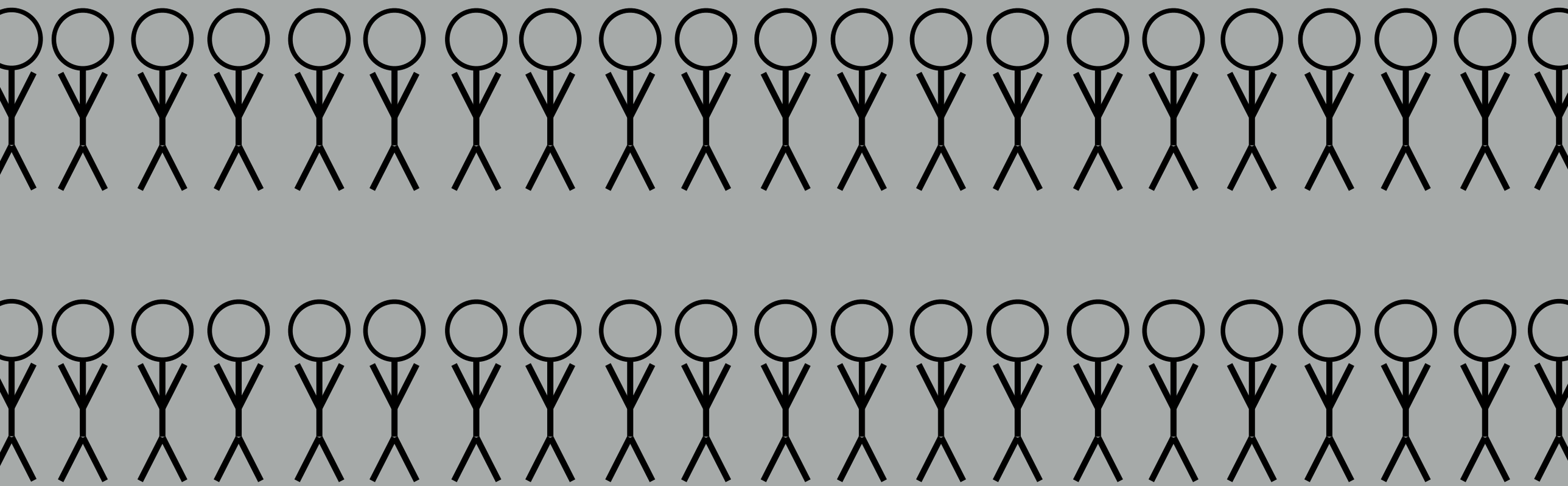
yearly income? $60,40
hates puppies? no
next purchase: Mac

yearly income? $2,300
hates puppies? yes
next purchase: PC

yearly income? $2,300
hates puppies? yes
next purchase: PC

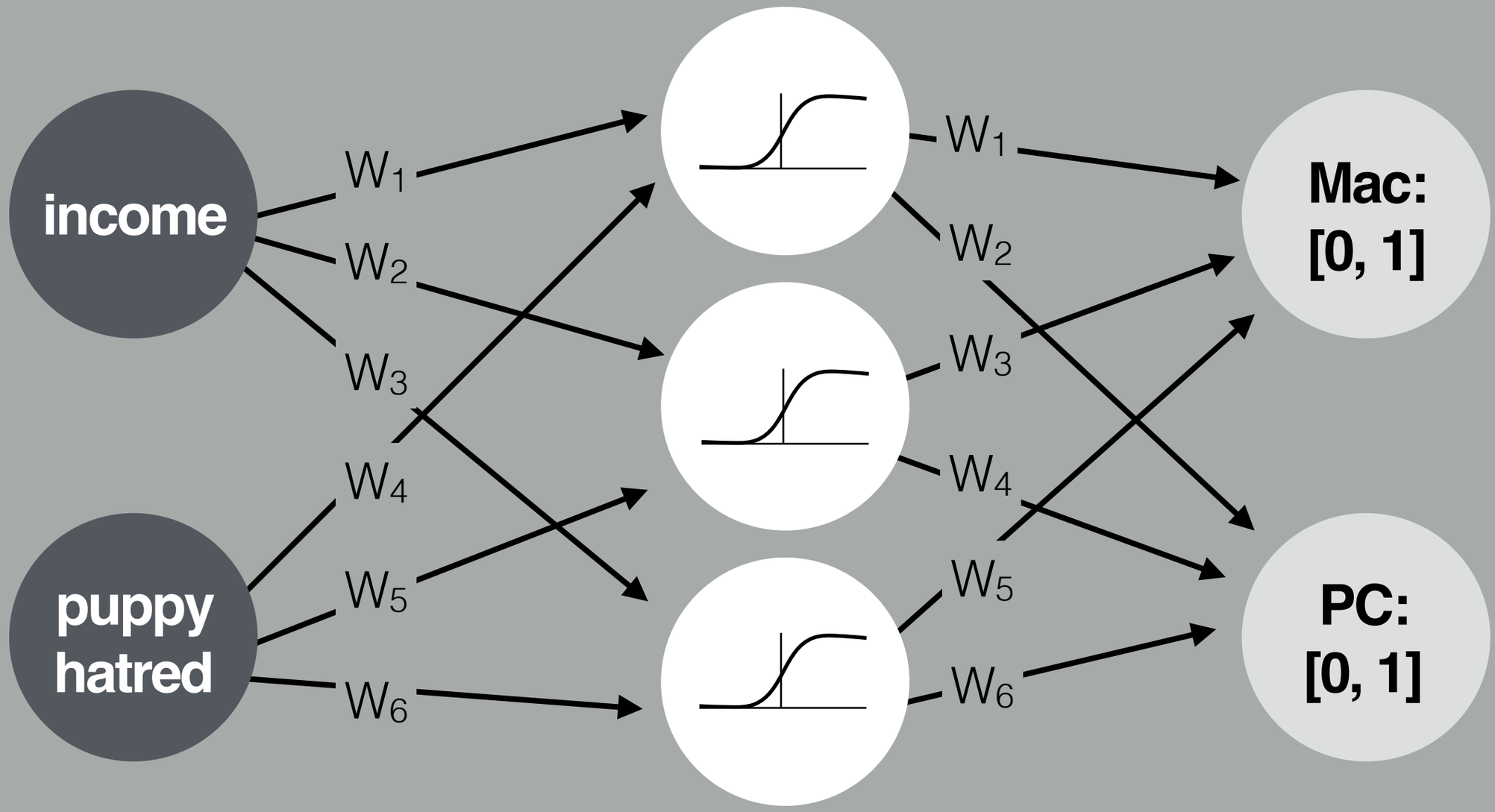
yearly income? $2,300
hates puppies? yes
next purchase: PC



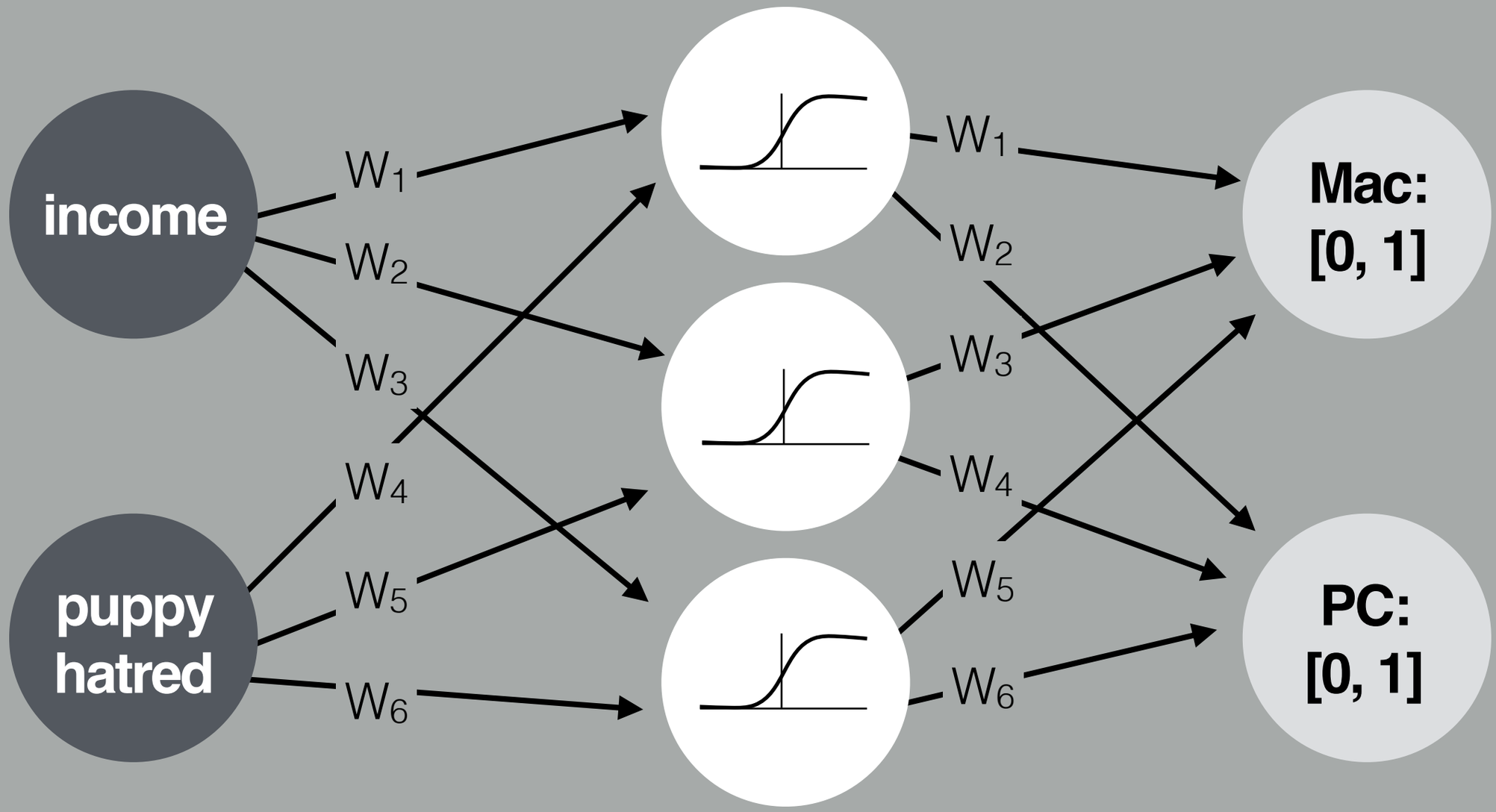


a simple NN for multi-class prediction

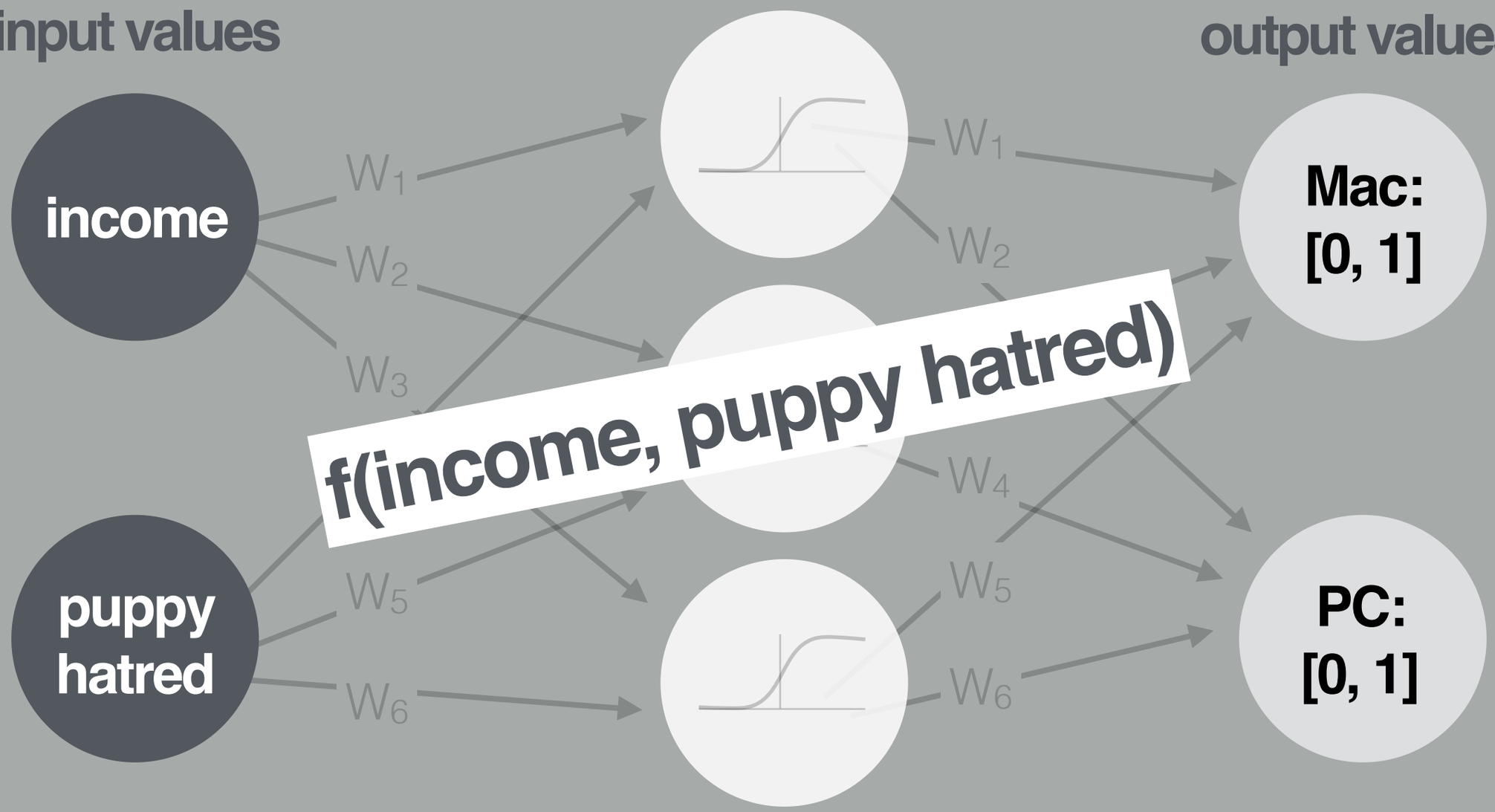
given income and puppy hatred, can we predict what computer a person will buy?



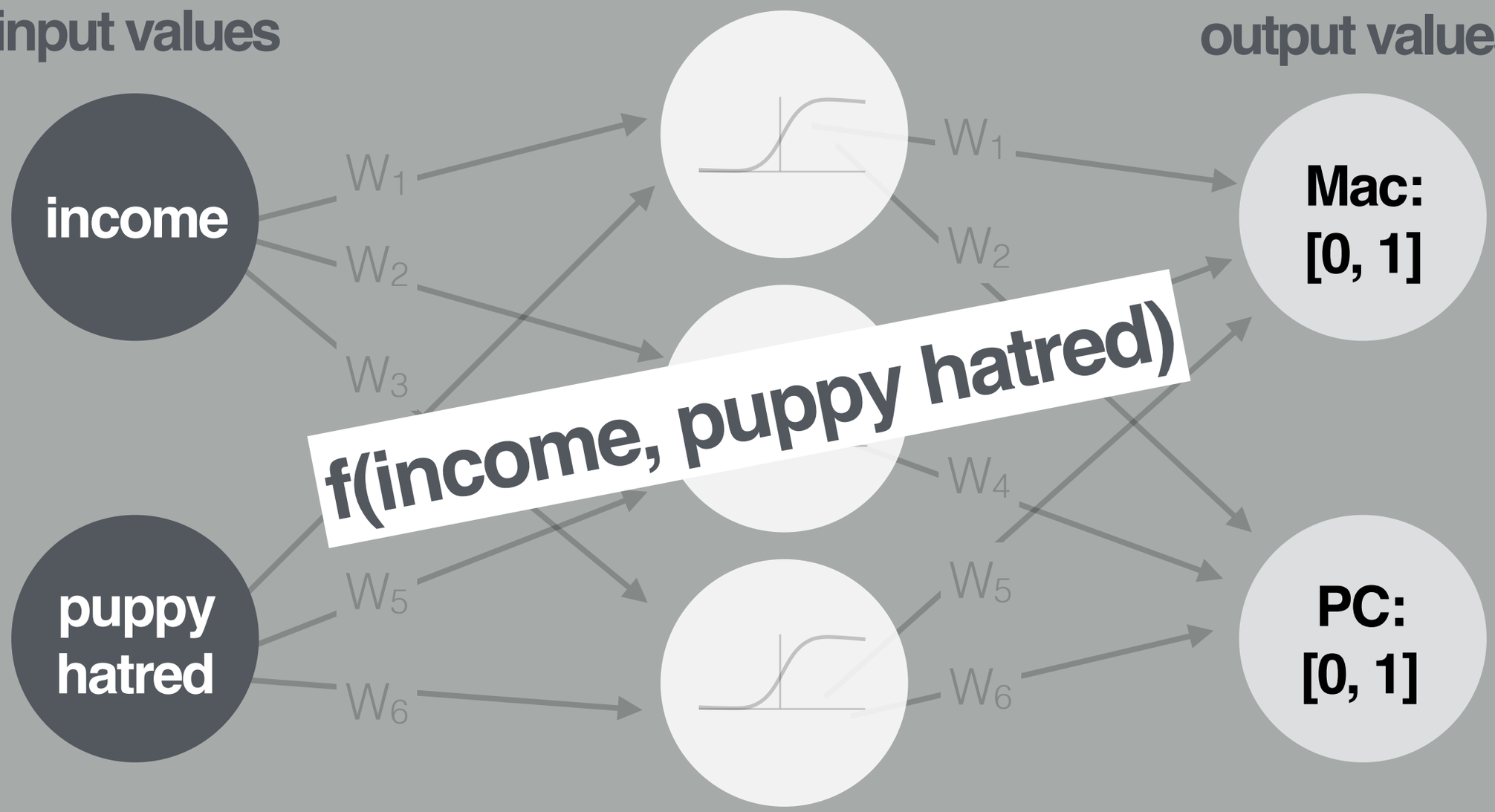
how can we teach this cute lil algorithm a thing?



how can we teach this cute lil algorithm a thing?



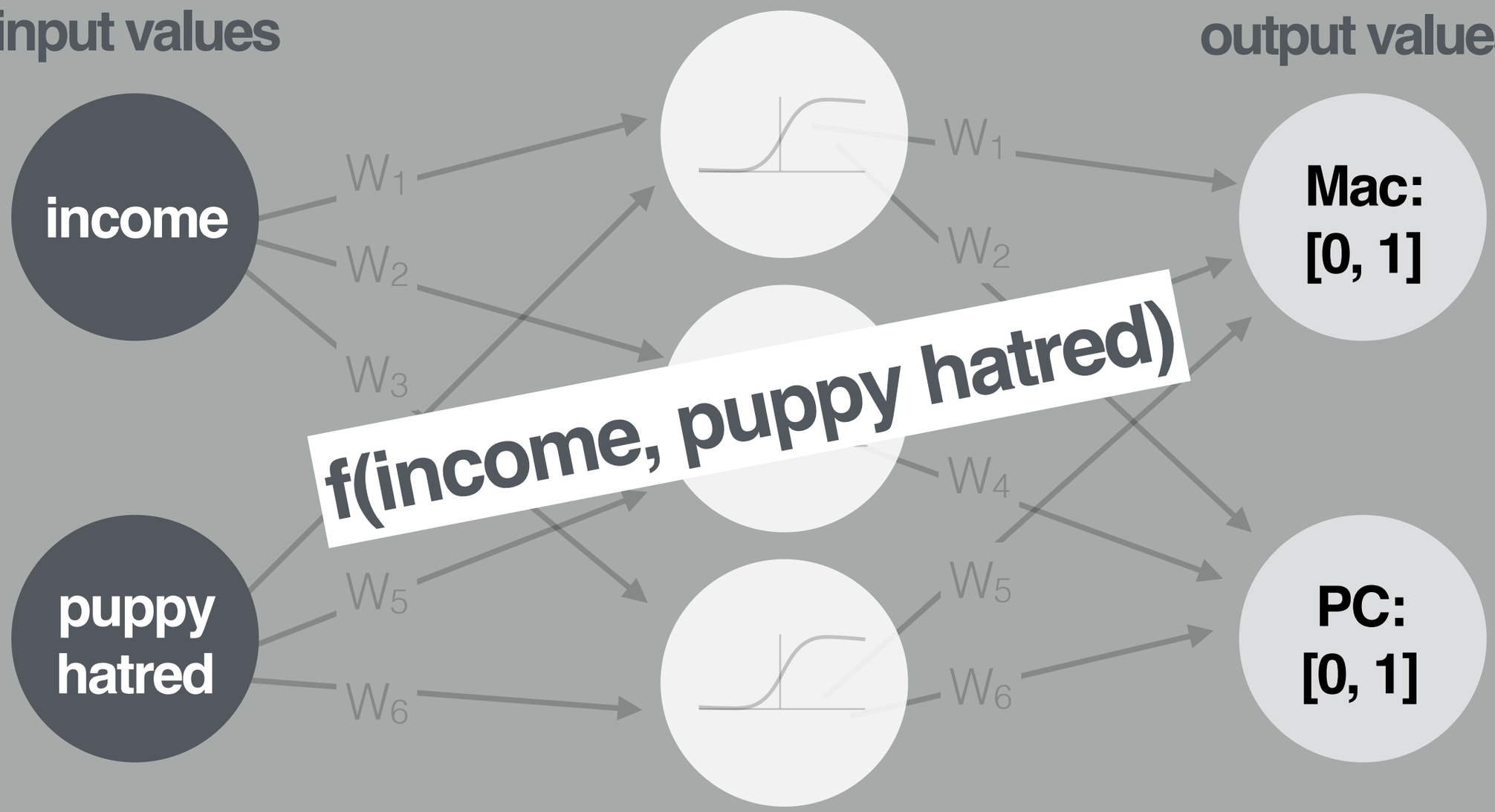
how can we teach this cute lil algorithm a thing?



supervised learning

inferring a function from labeled training data

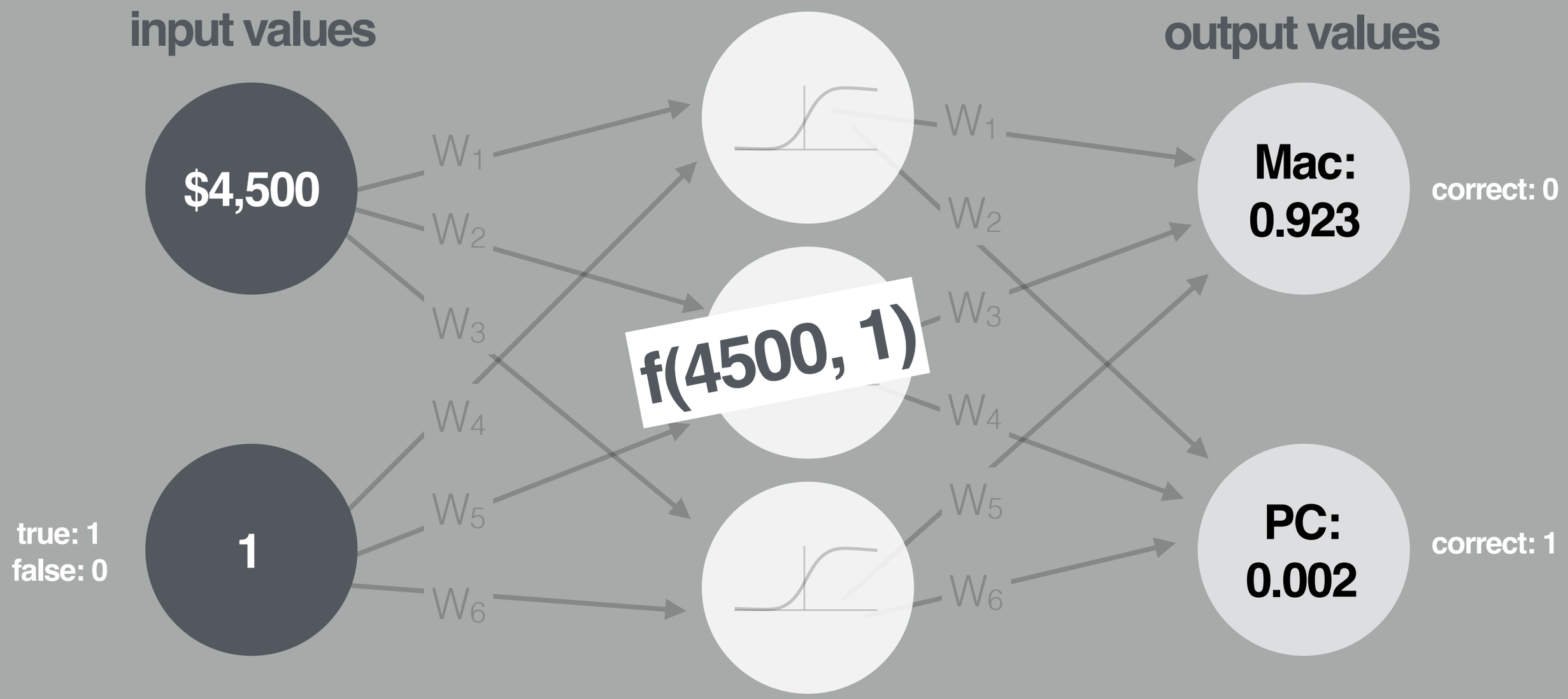
how can we teach this cute lil algorithm a thing?



supervised learning

let the algorithm compute a predicted output for a given input, then use math to correct its error

how can we teach this cute lil algorithm a thing?

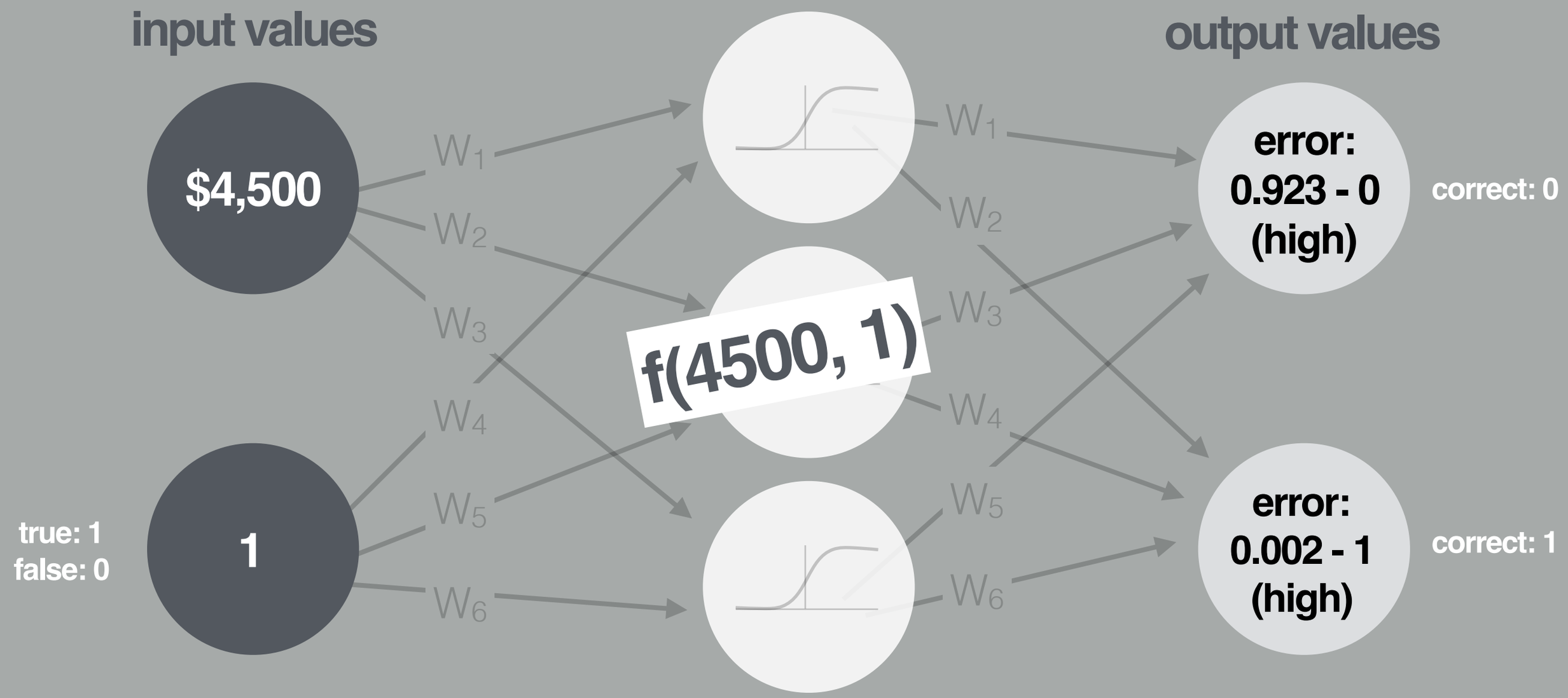


supervised learning

we can define error as a function — typically, it is some variation of

(predicted value - actual/correct value)

how can we teach this cute lil algorithm a thing?

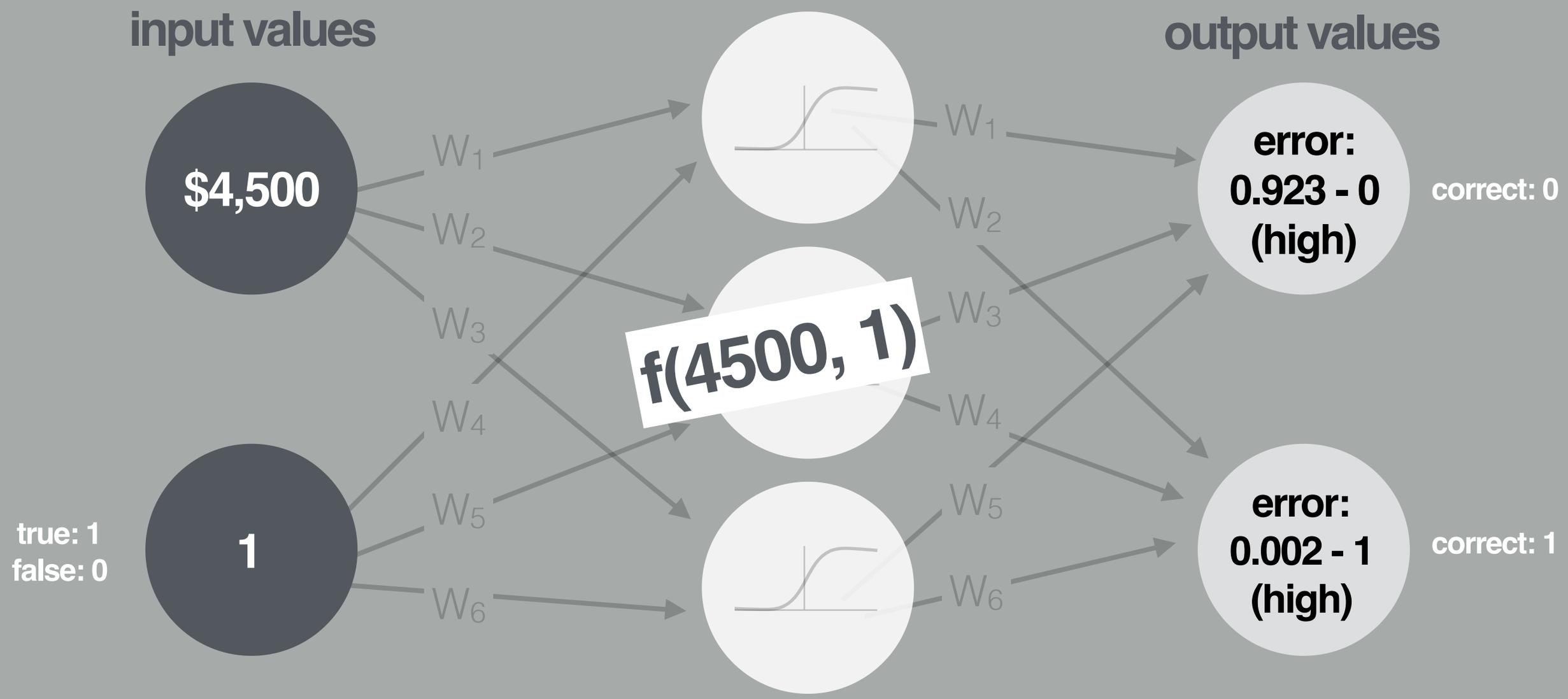


supervised learning

we can define error as a function — typically, it is some variation of

(predicted value - actual/correct value)

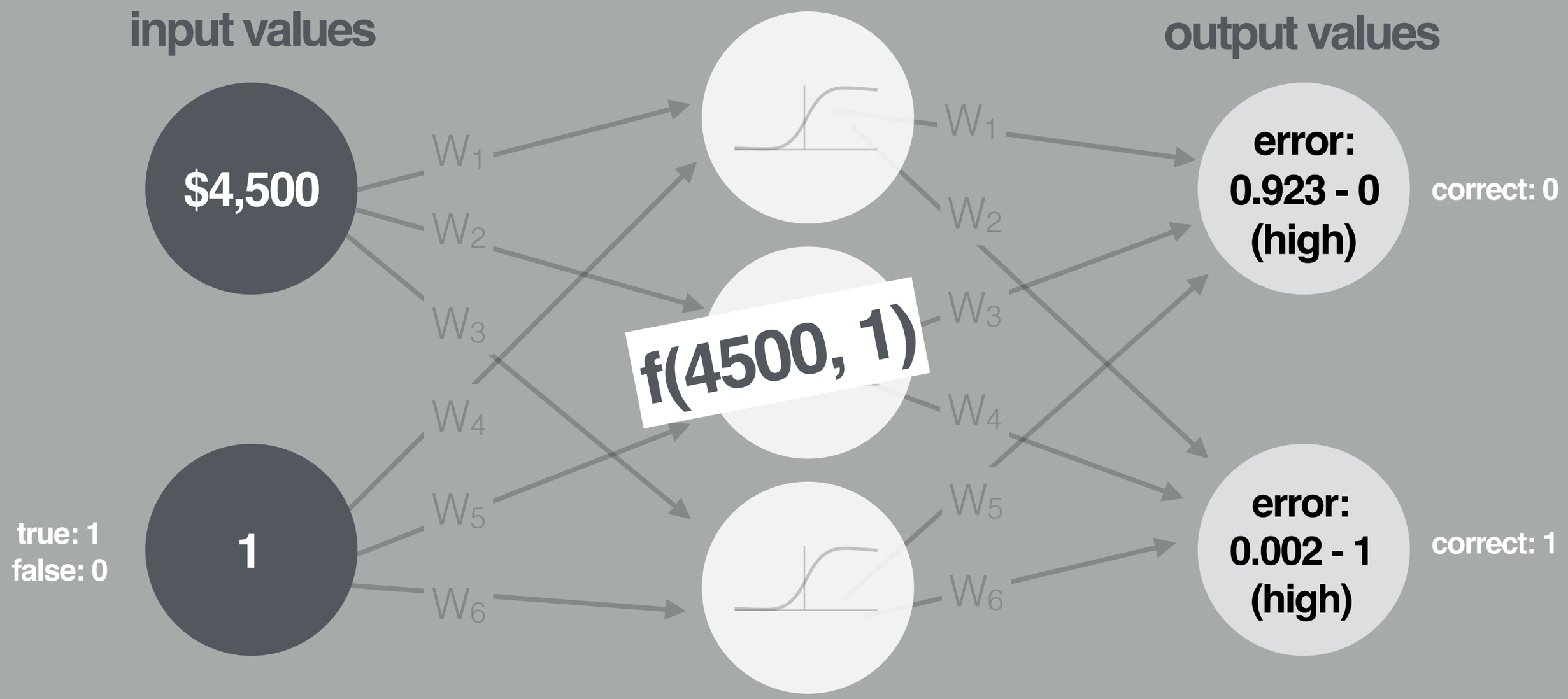
how can we teach this cute lil algorithm a thing?



supervised learning

this function is typically called the “cost function” or “loss function.” It returns a high value when the error is large — **our goal is to minimize this error.**

how can we teach this cute lil algorithm a thing?

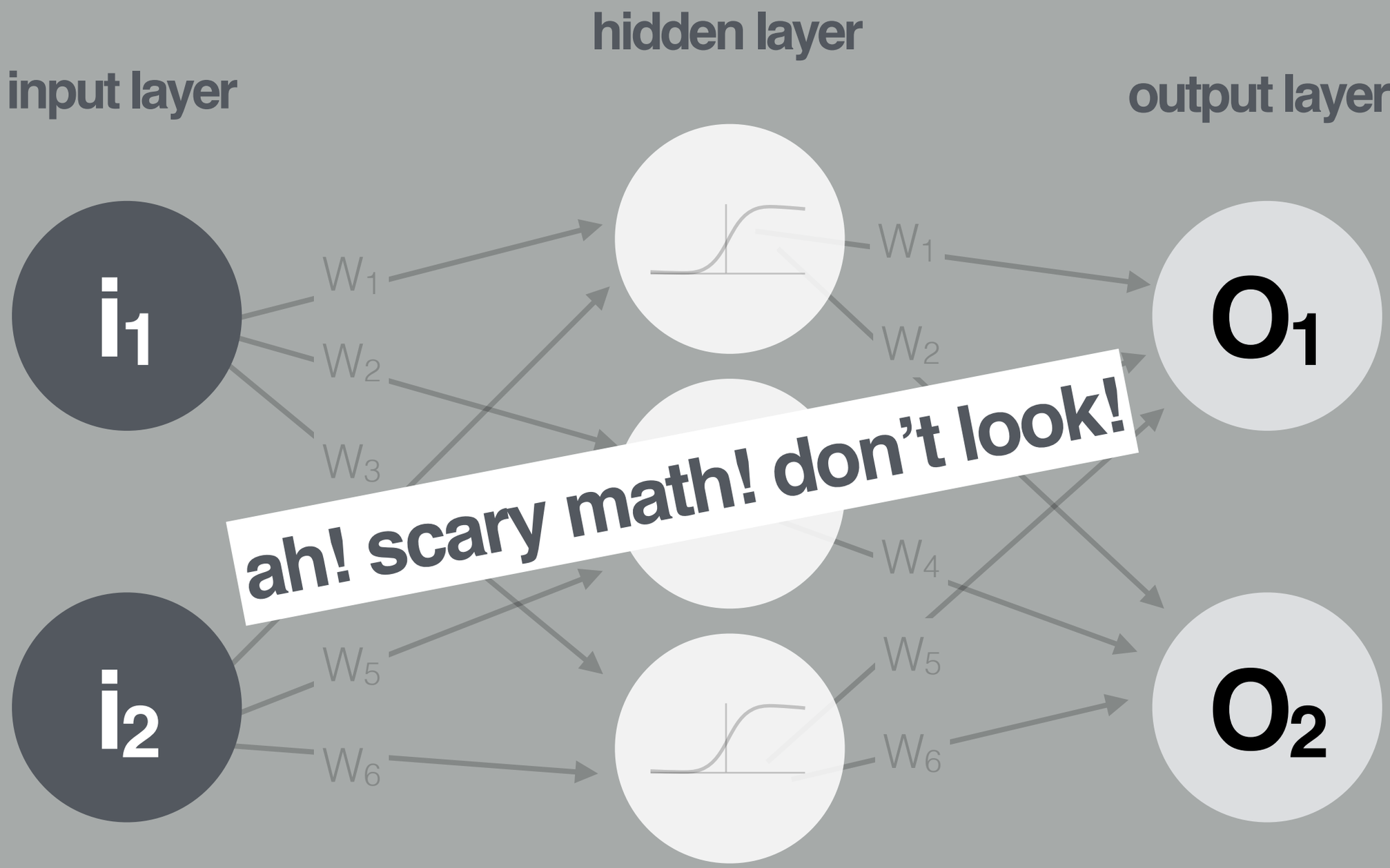


supervised learning

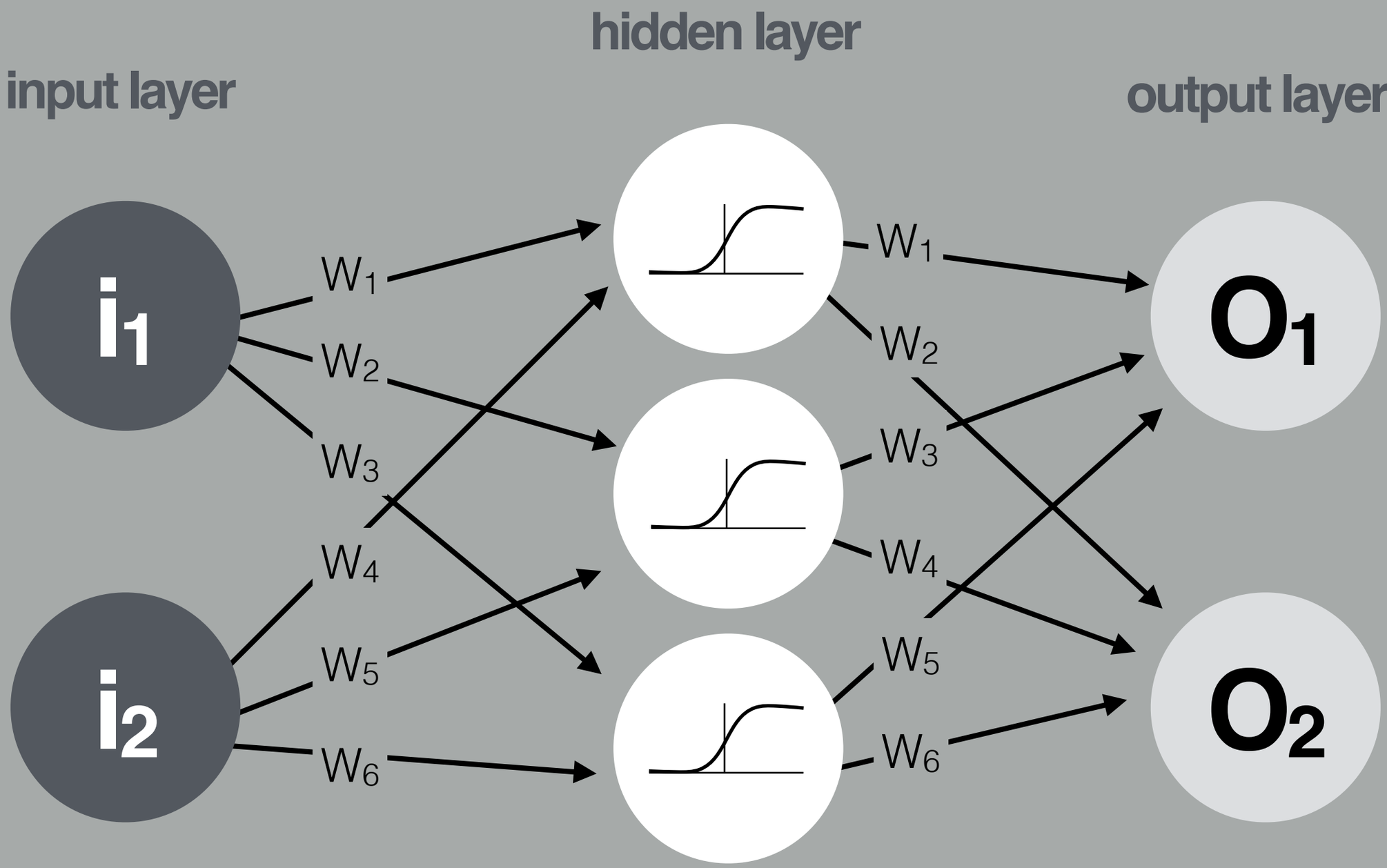
we will do this by using **backpropagation**, which will modify the function in order to minimize the error it exhibits

time for some backpropagation, y'all

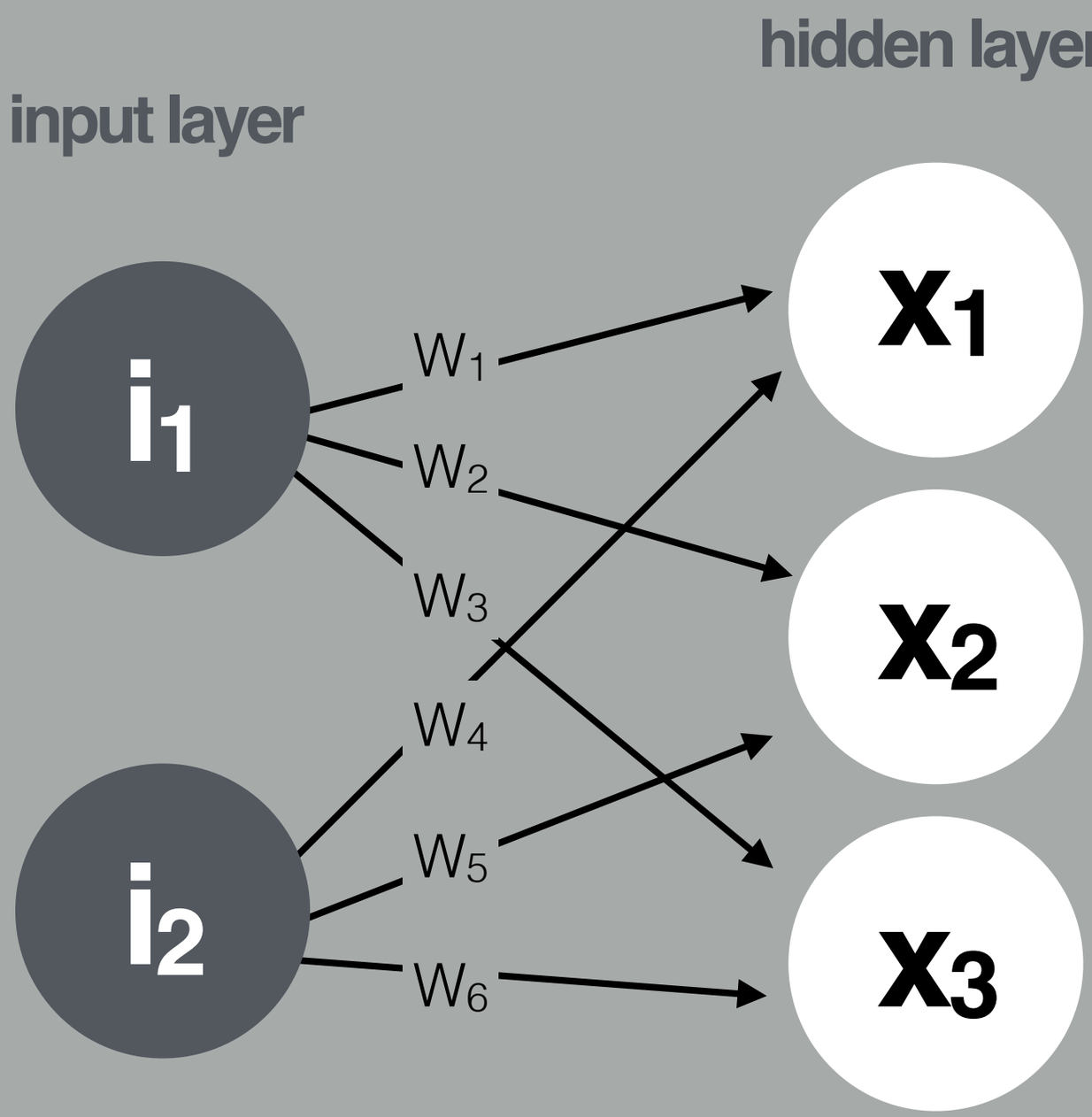
oh wow! sounds unbelievably exciting!



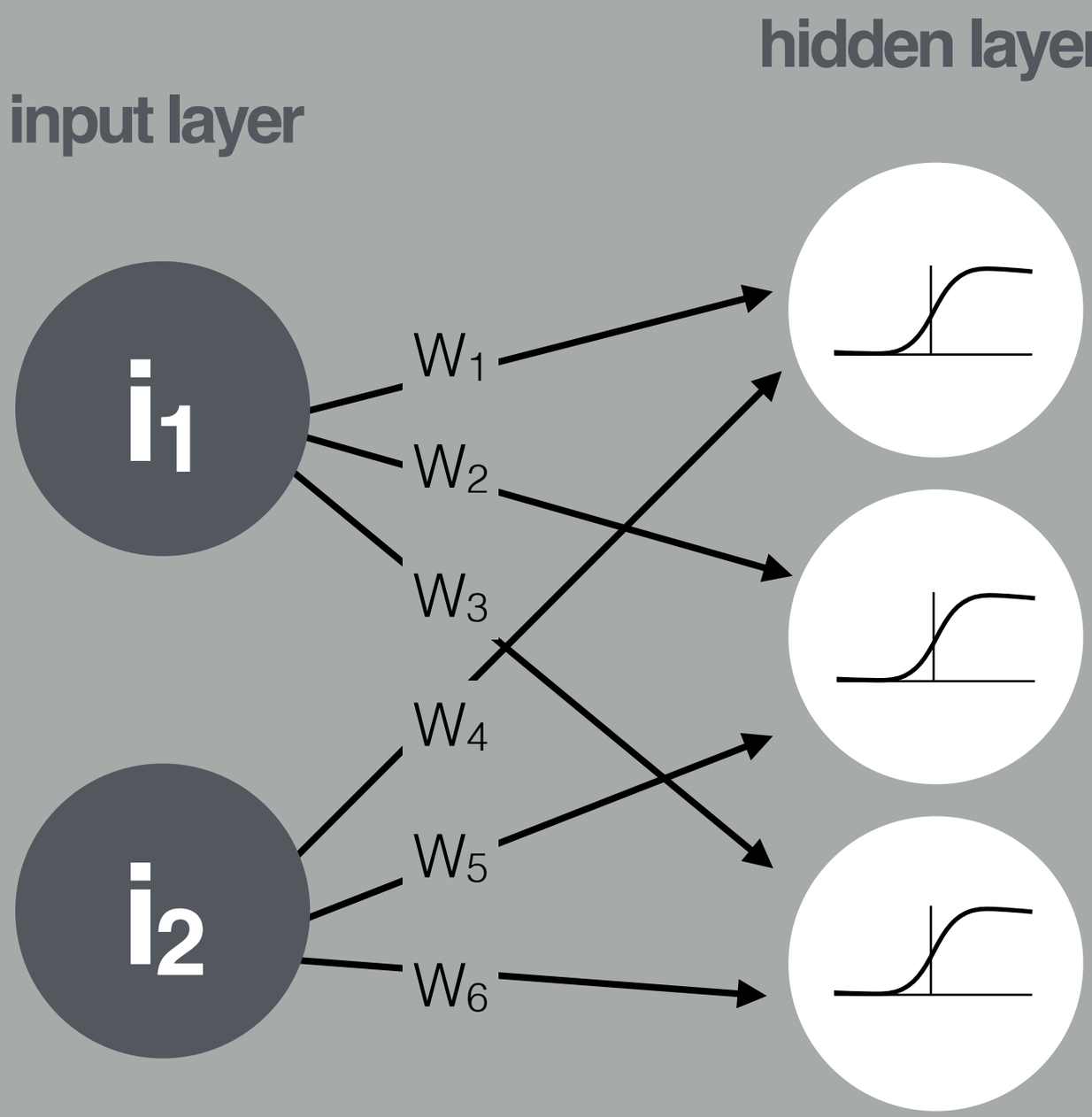
In order to determine how to minimize the cost function (via backprogagation), we need to understand how a neural network arrives at its prediction. This is called **forward propagation.**



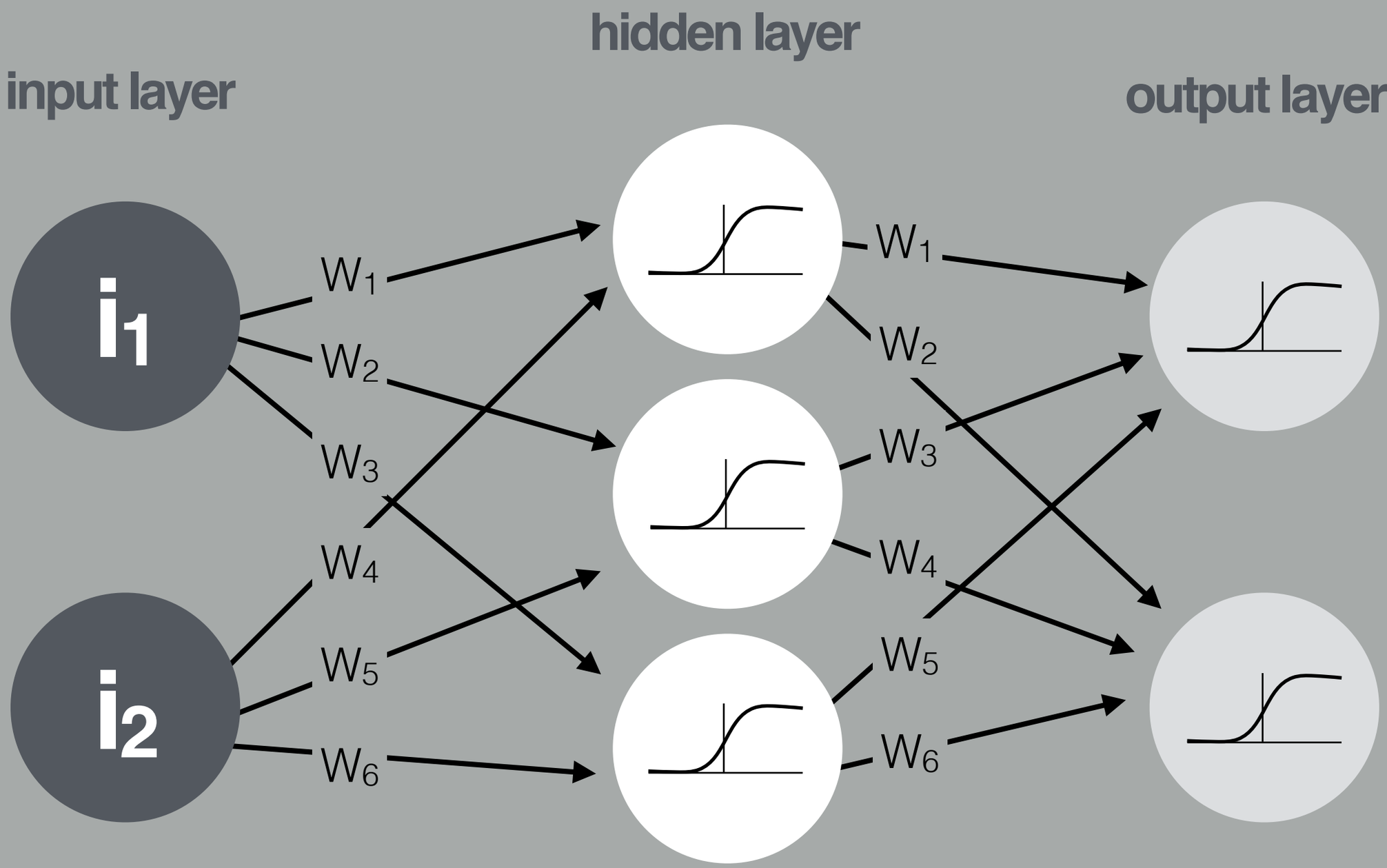
Let's look under the hood!



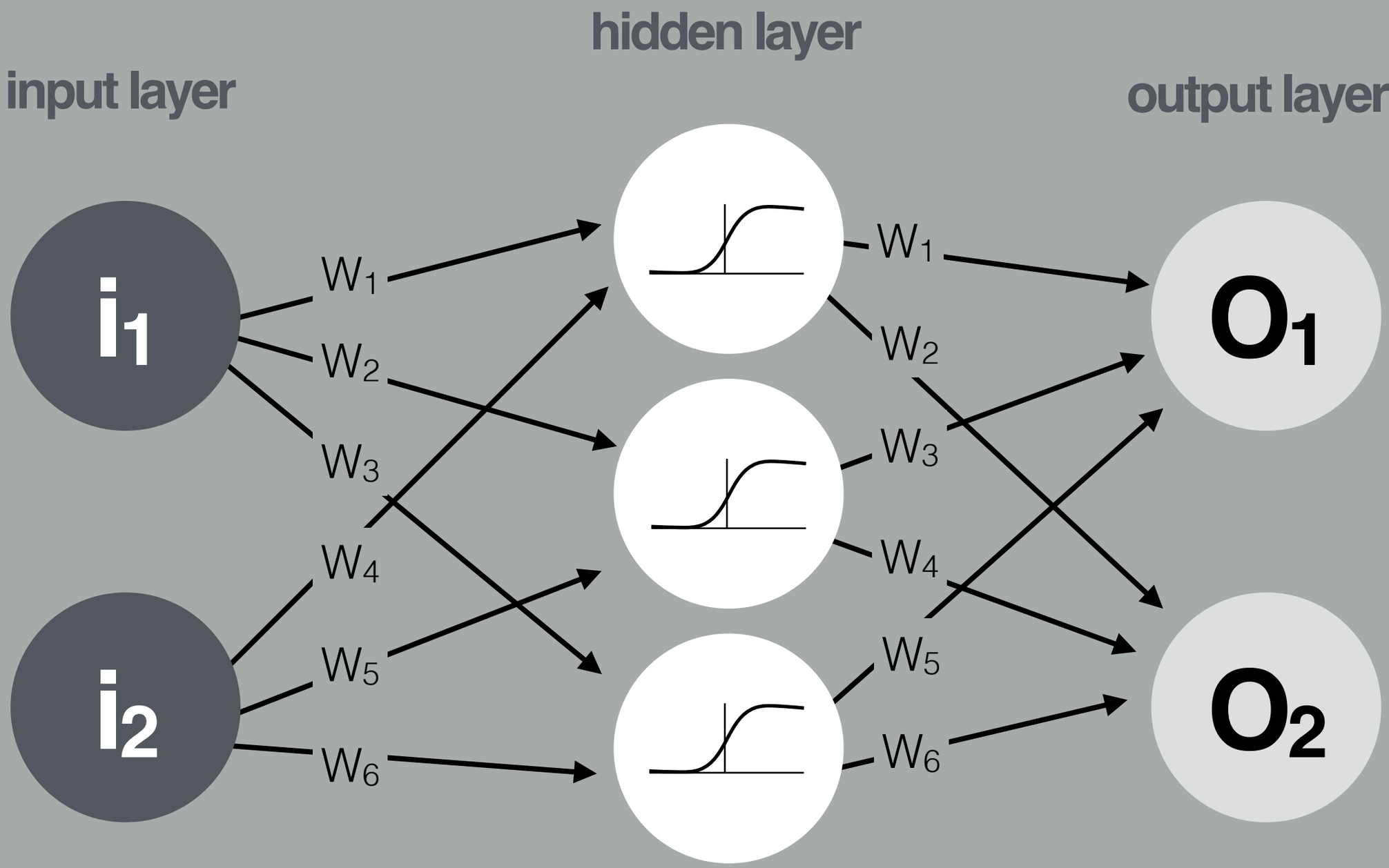
Each input node is multiplied by each of the weights that correspond to a hidden layer node. (This value is denoted as x_i)



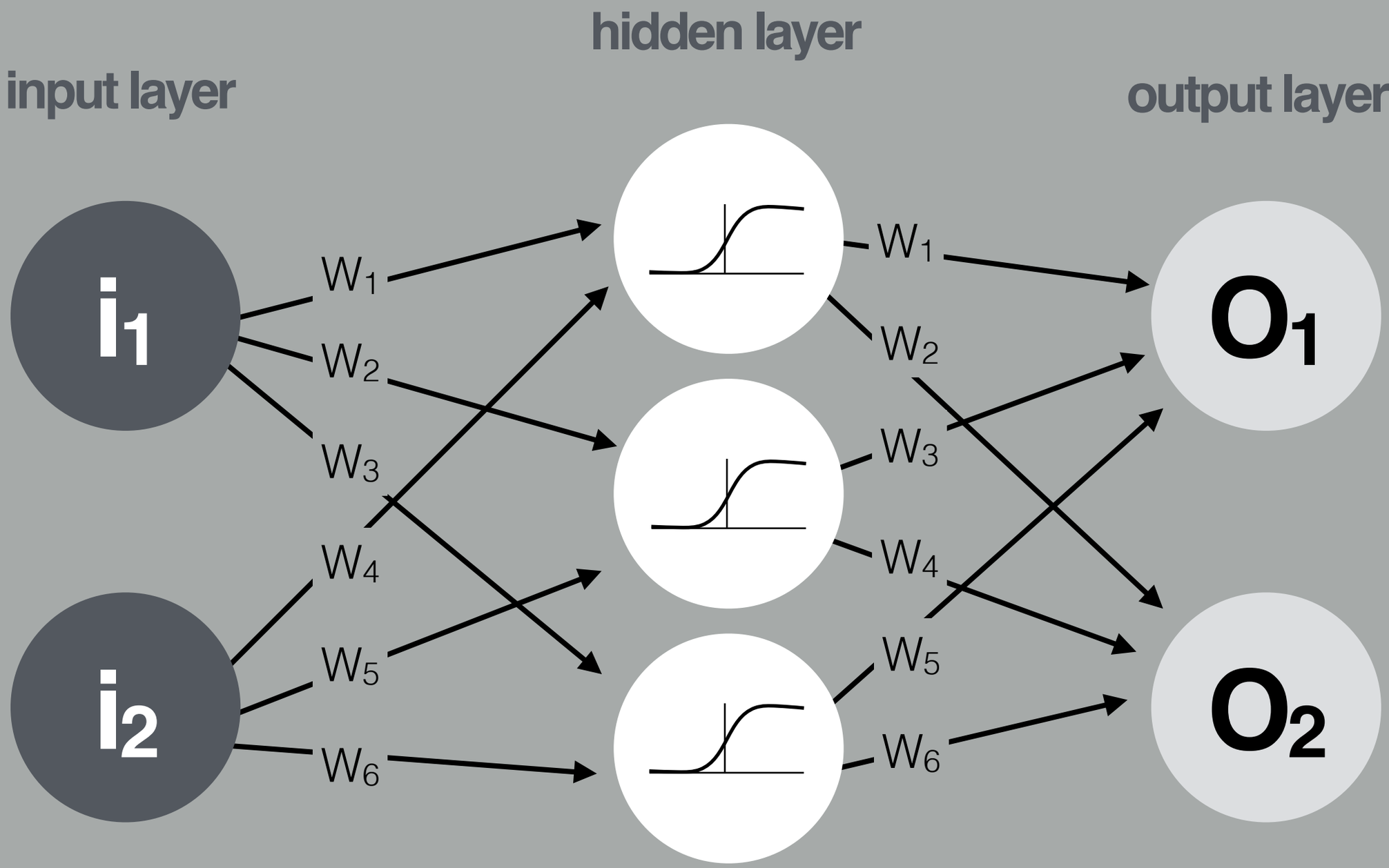
Apply some *activation function* to each of the x values. This function must be *differentiable*. We'll be using the *sigmoid function*, which will map x_i to a value between 0 and 1.



Multiply the value obtained by the hidden layer node by the corresponding weight value. Then, apply the sigmoid activation function again.



This value (O_i) is your predicted output. Now, we can calculate the error for this prediction.



We will be using the mean-squared error function. This ensures that the calculated error will always be greater than zero.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

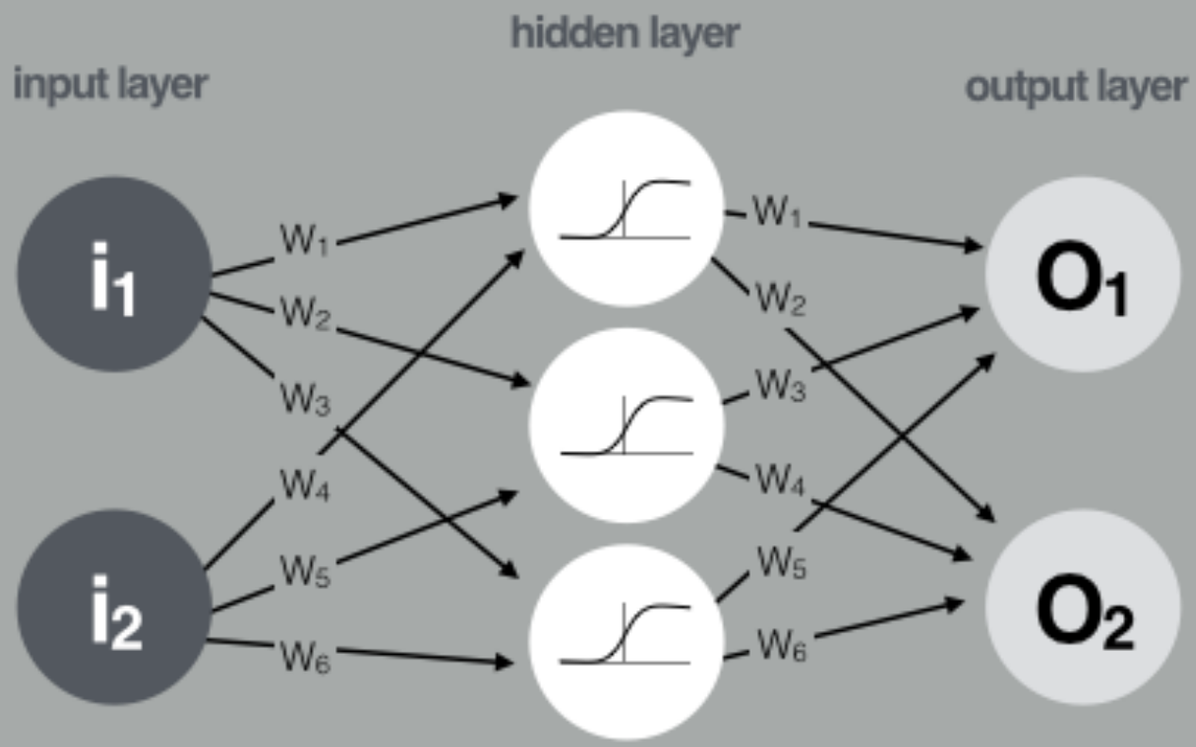
how can we teach this cute lil algorithm a thing?

intuition

we need to figure out how to change the weights of the NN to minimize the cost function (MSE)

math speak

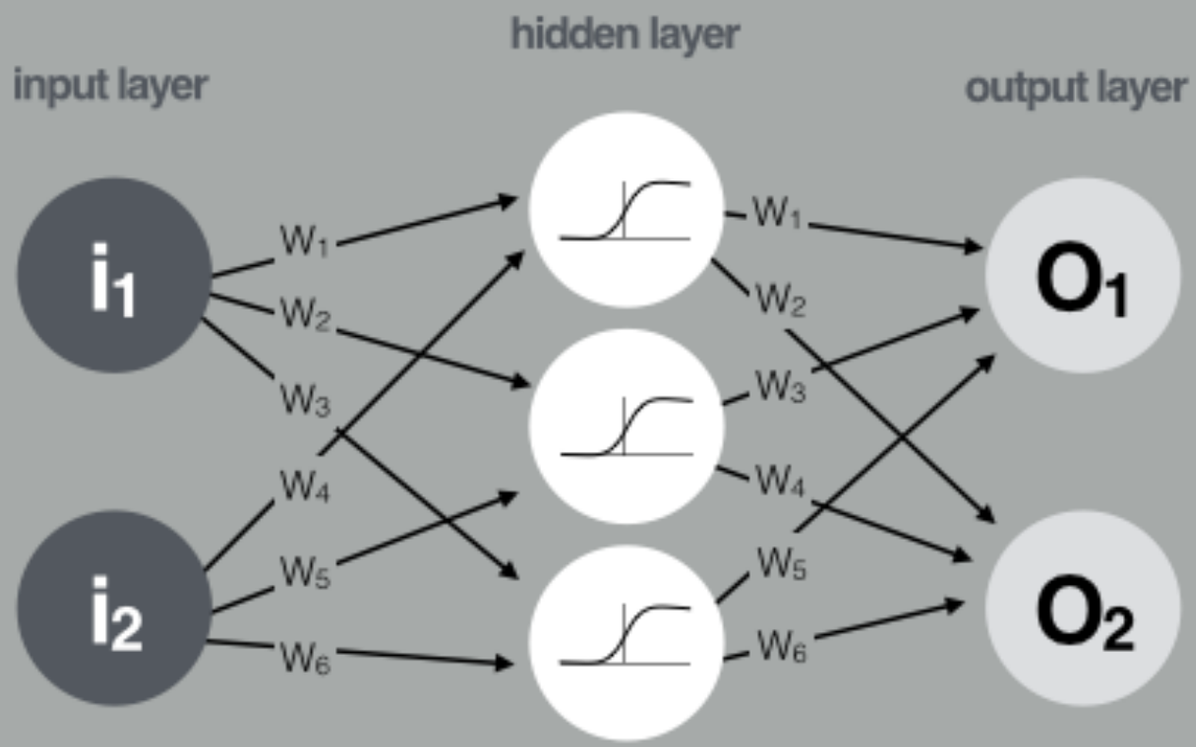
we need to calculate the partial derivative of each weight value with respect to the cost function this will tell us the rate of change of the MSE w.r.t the weight values



how can we teach this cute lil algorithm a thing?

mo' intuition

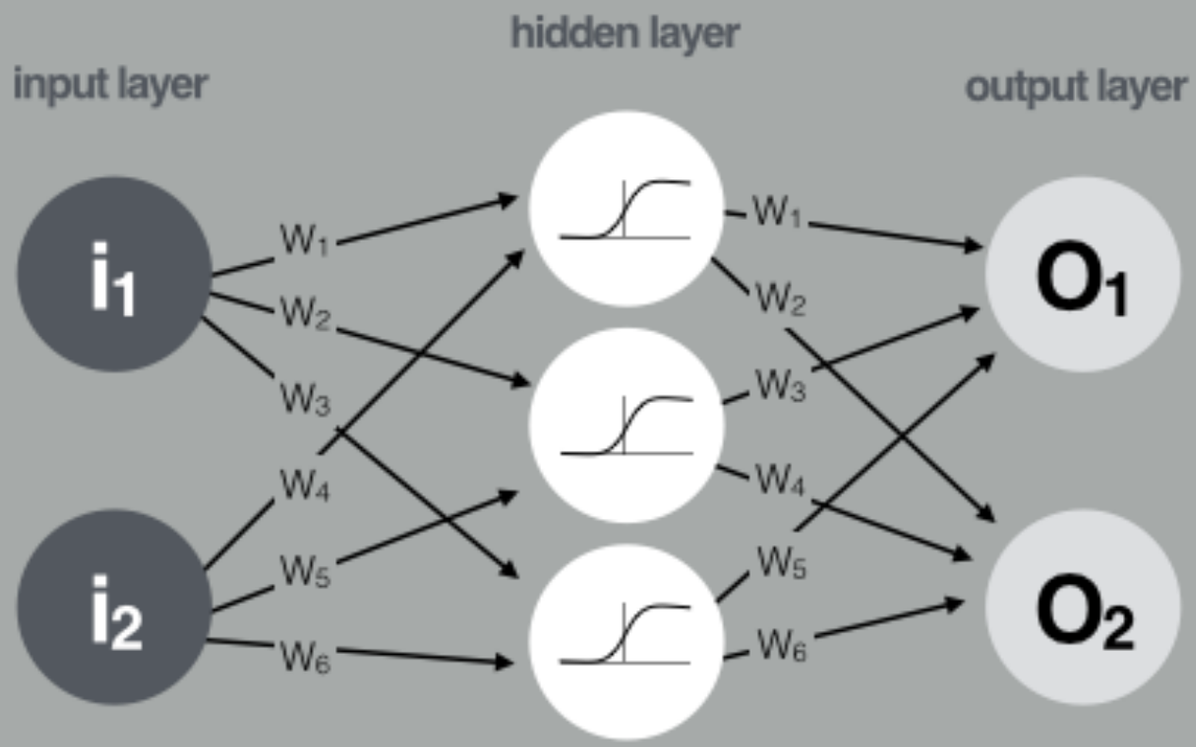
it makes sense that the weights will need to be adjusted in a way that corresponds to the significance (or effect) each input value has on the final value we want to obtain



how can we teach this cute lil algorithm a thing?

mo' intuition

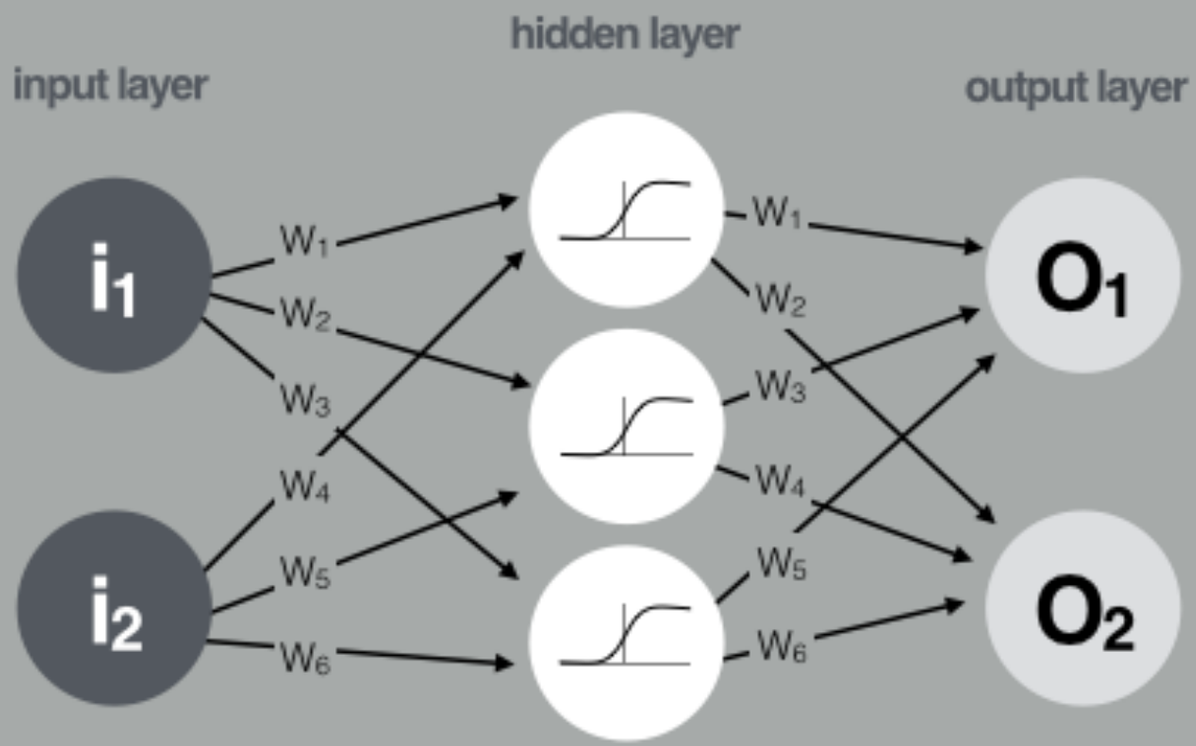
for example: it's clear that those with higher incomes may be more able to afford a Mac, and therefore might buy it over a cheaper PC. Similarly, puppy-haters would obviously be more likely to buy a PC because they hate fun.



how can we teach this cute lil algorithm a thing?

mo' intuition

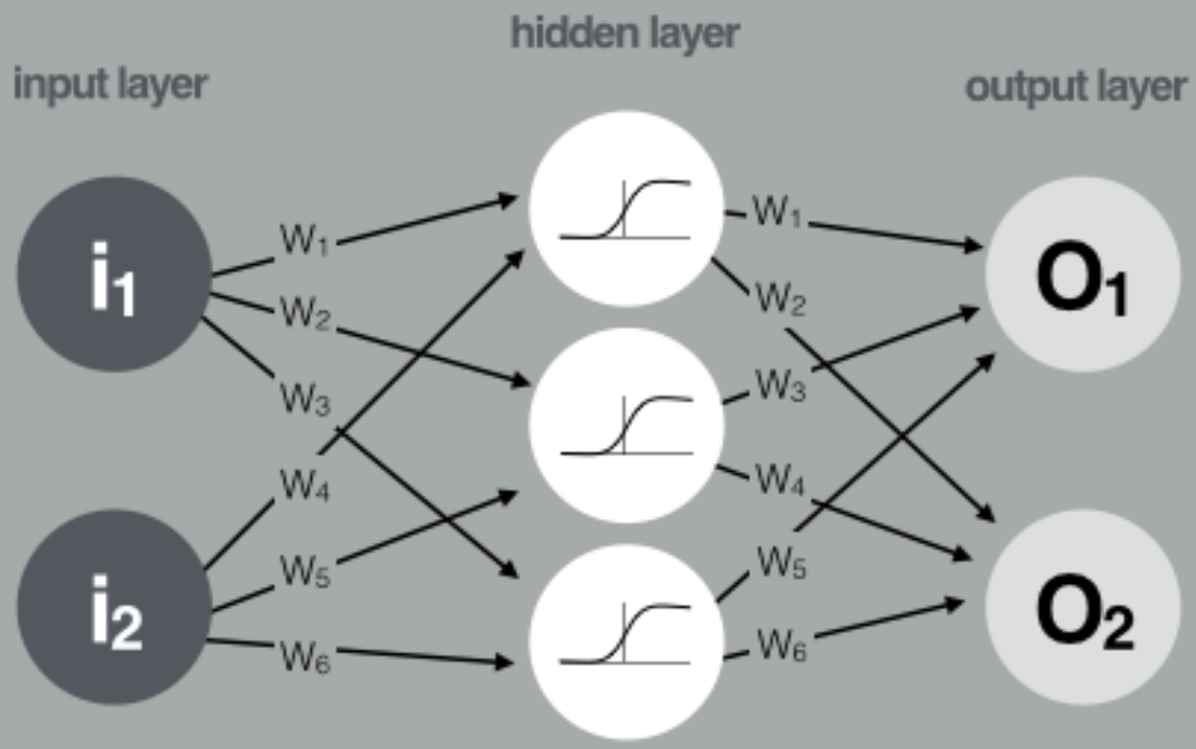
the derivatives tell us how each input value should be weighted in order to achieve the correct (desired) result



how can we teach this cute lil algorithm a thing?

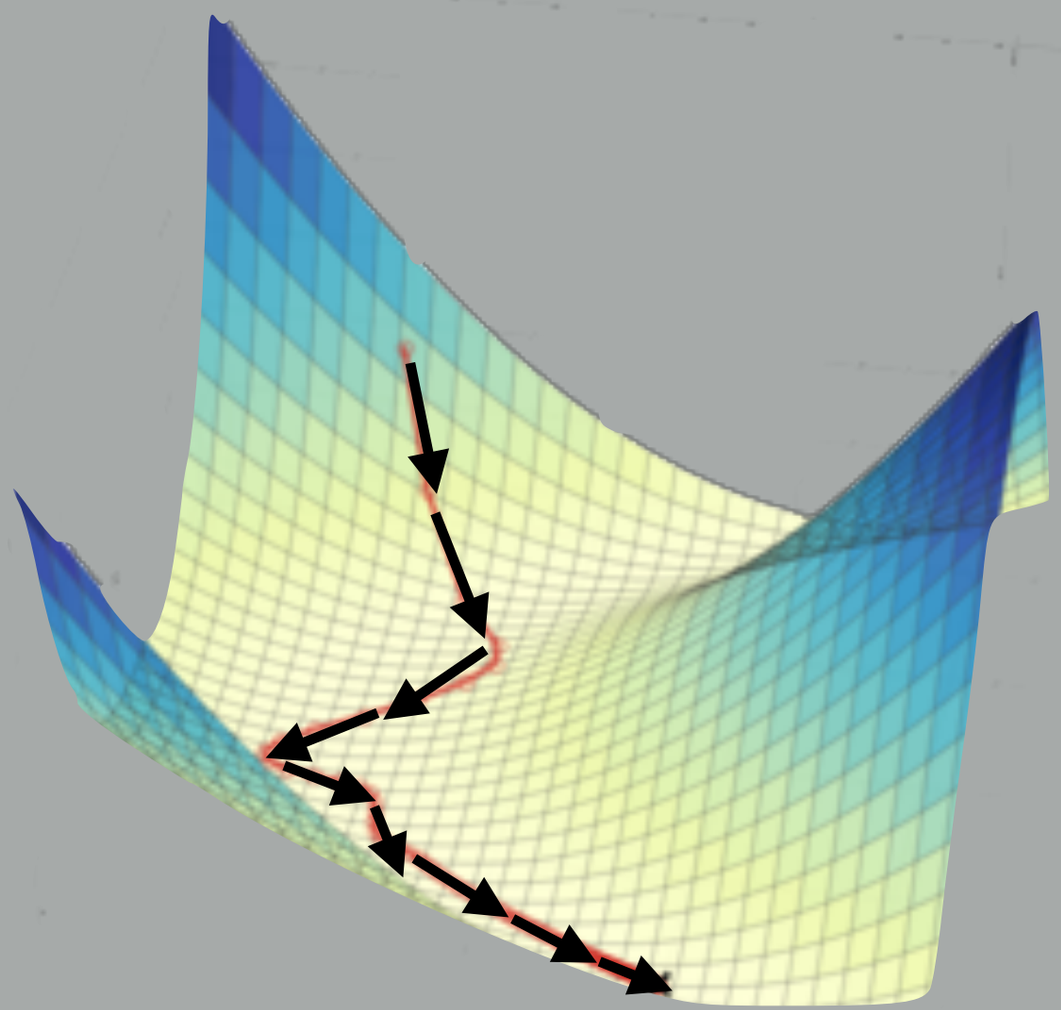
how does that help?

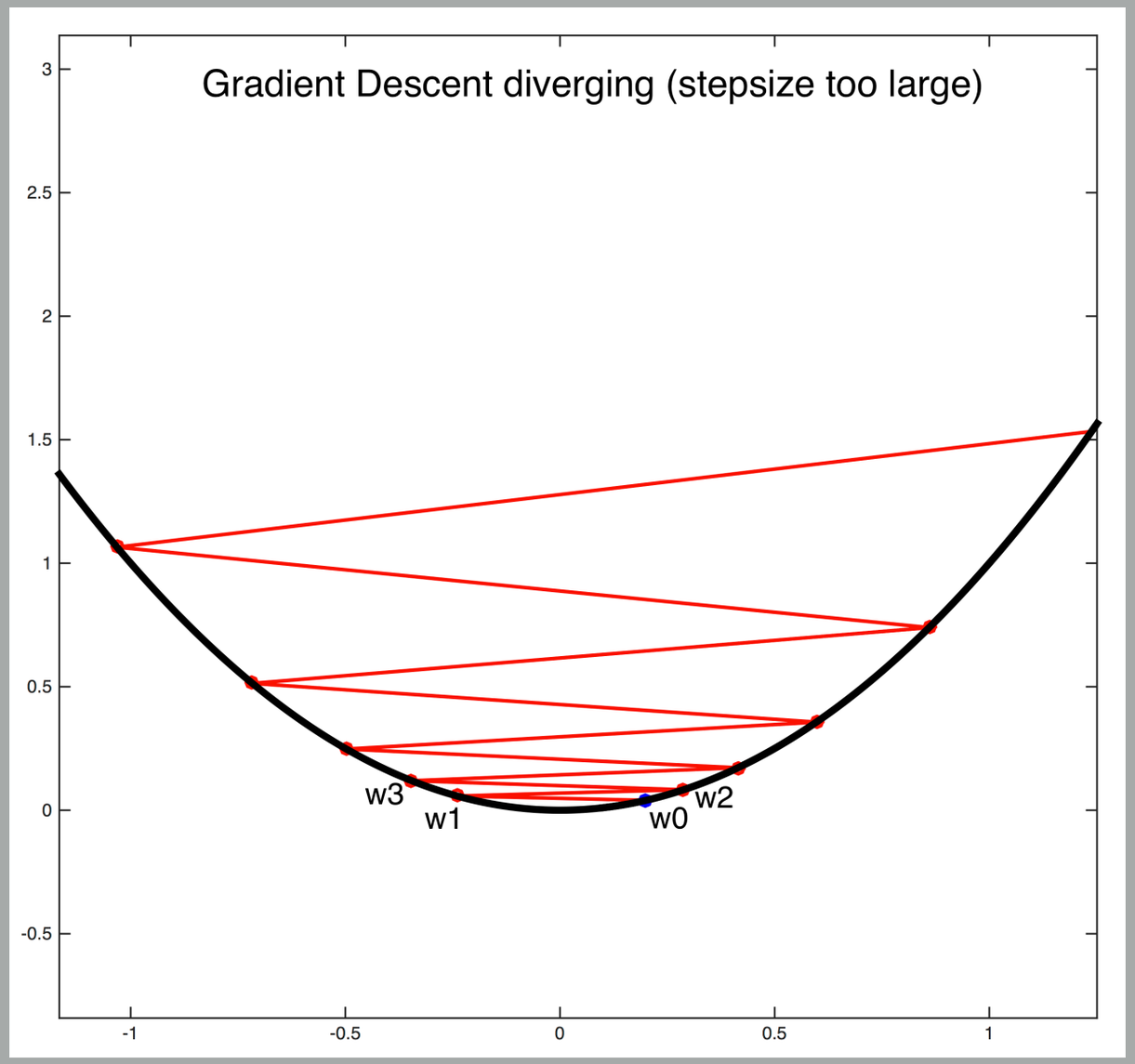
once we have the derivative for each weight value w.r.t the MSE, we can perform gradient descent to update the weights to minimize the MSE



gradient descent? ? ??

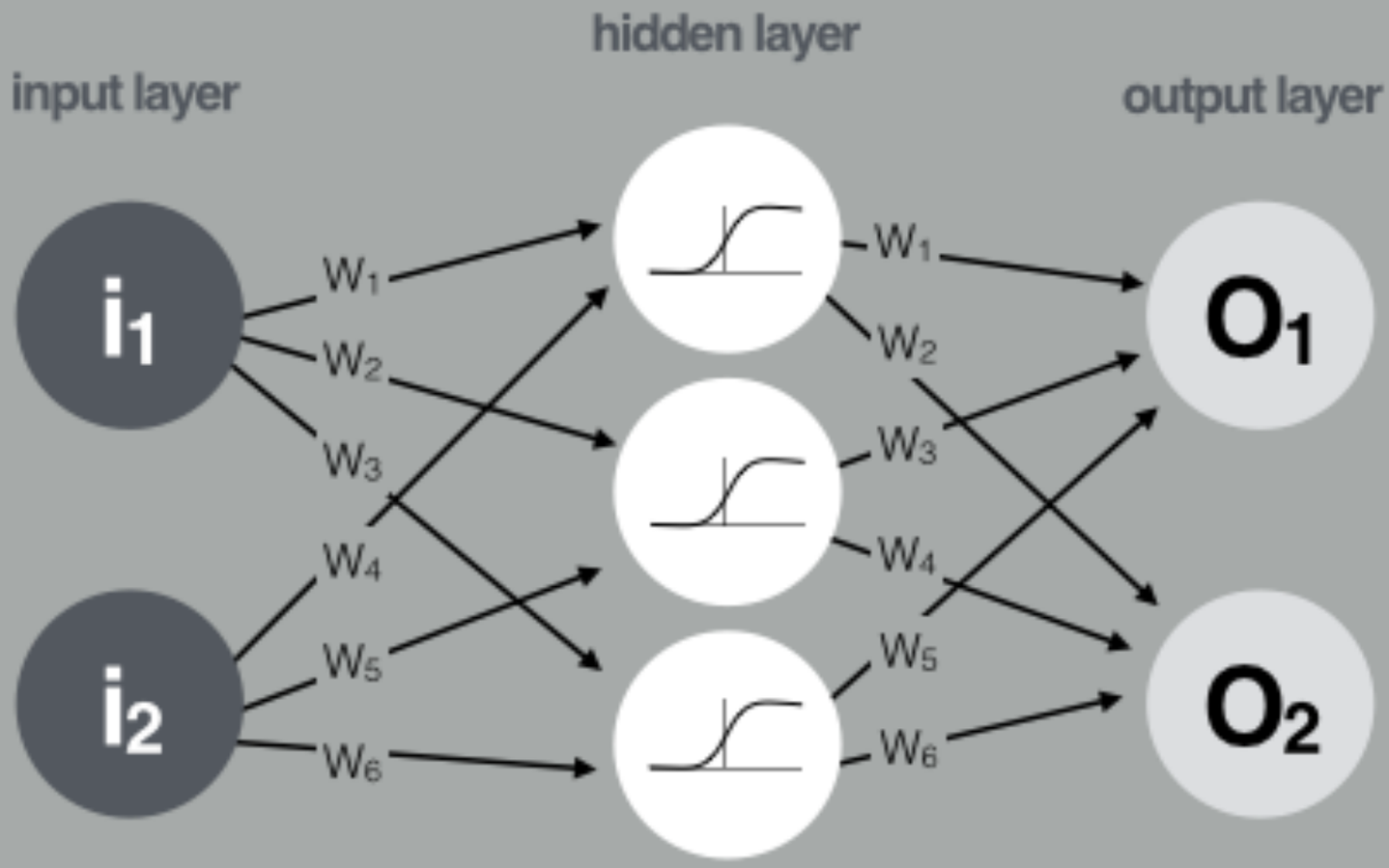
now we have to do math :(



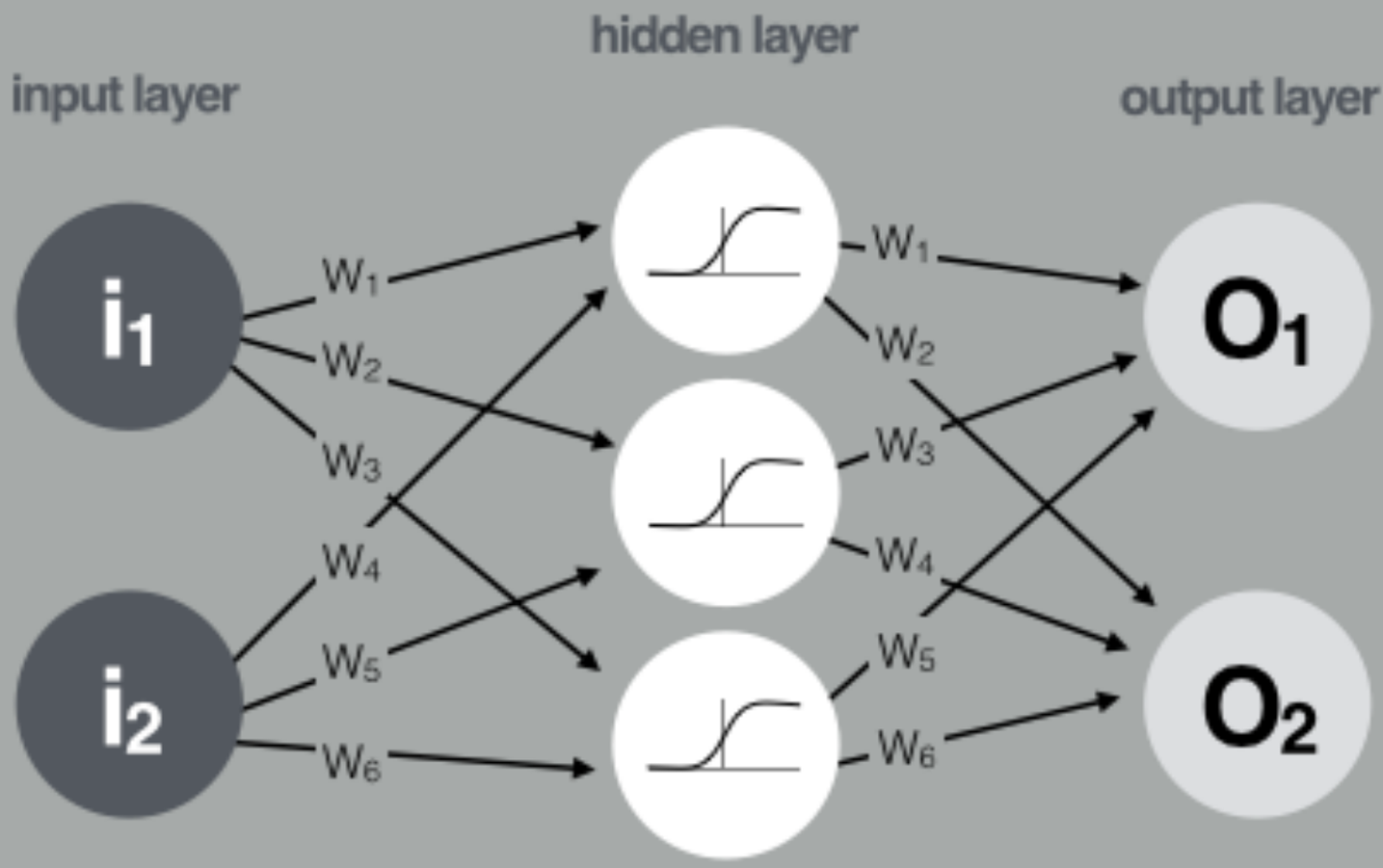


The derivatives tell us which way is downhill (towards the minimum of the function), but we need to take iterative steps to get to the minima.

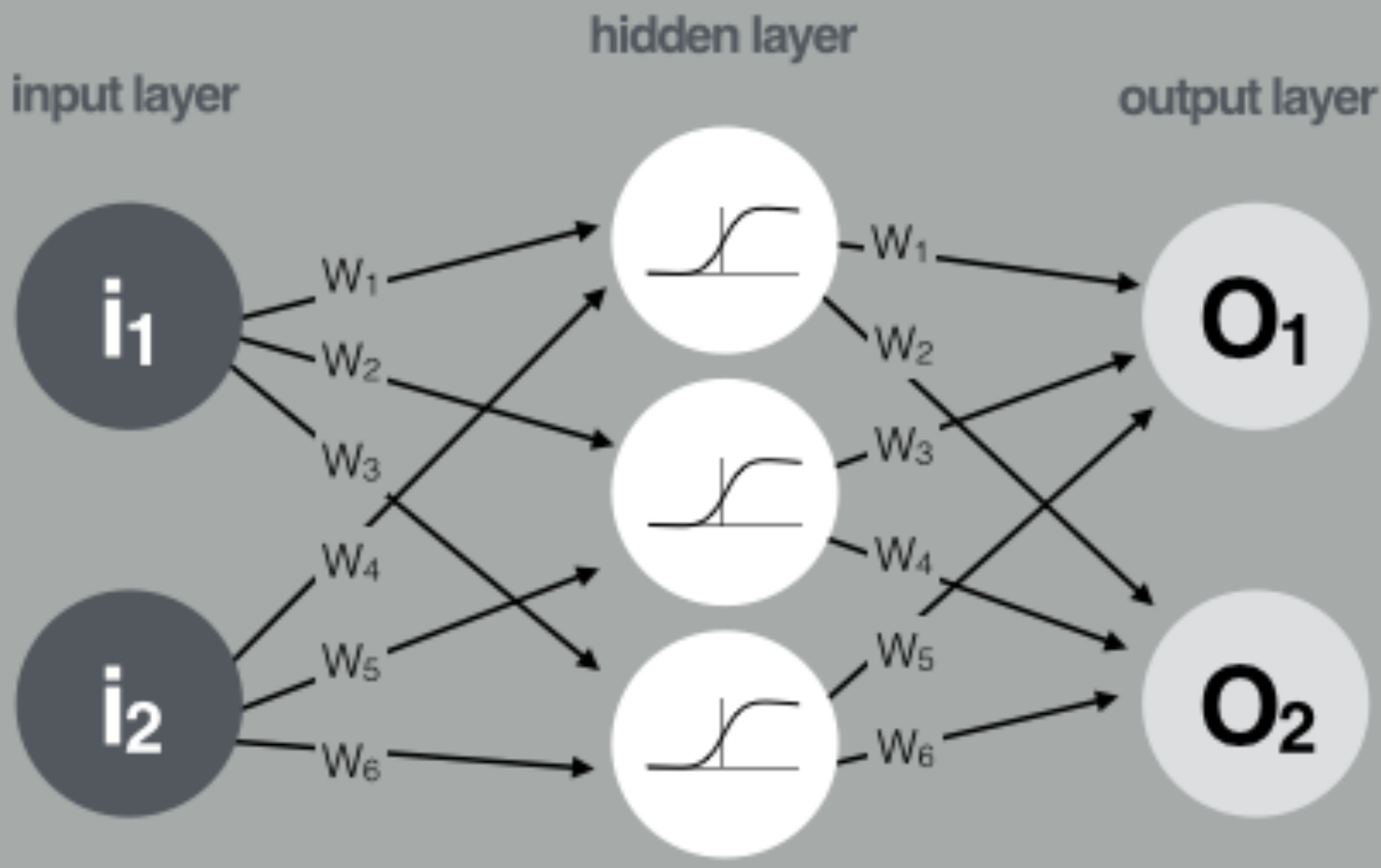
We use an alpha value (or learning rate) that will dictate how big this step size should be. If it's too small, the NN will take a long time to converge. If it's too large, the NN may overshoot the minima.



We continue forward-propagation with different examples (called *training samples)* and then backpropagate the error. This process continues with many, many samples. Eventually, the NN should obtain a low rate of error.



Essentially, we are feed this lil guy a bunch of examples. He makes a guess, then we tell him how off he is. He adjusts his weight values a bit, then we give him another one. We keep going until he's learned what we want him to.



Now we have an algorithm that can correctly predict (within some margin of error) whether a person will buy a Mac or a PC, if we tell them their income and whether or not they hate puppies.

WE DID IT

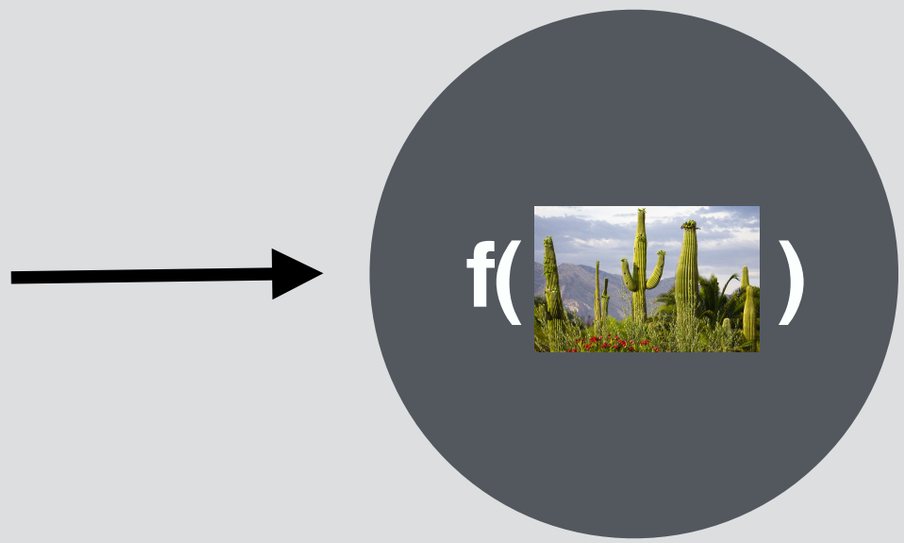
except we started off talking about horses.

crap.

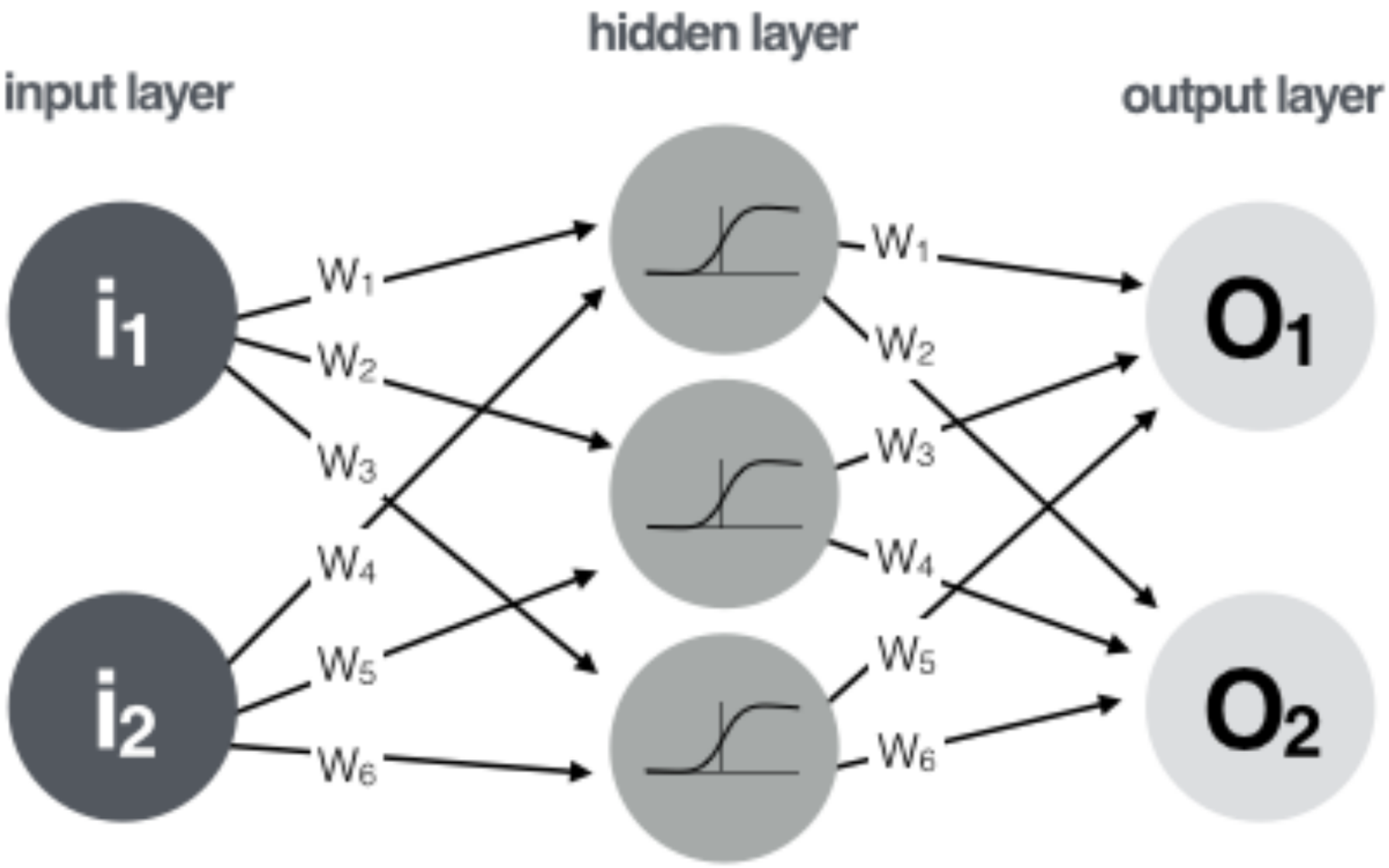
we originally wanted our computer to identify whether something was a horse or a cactus

that’s a horse

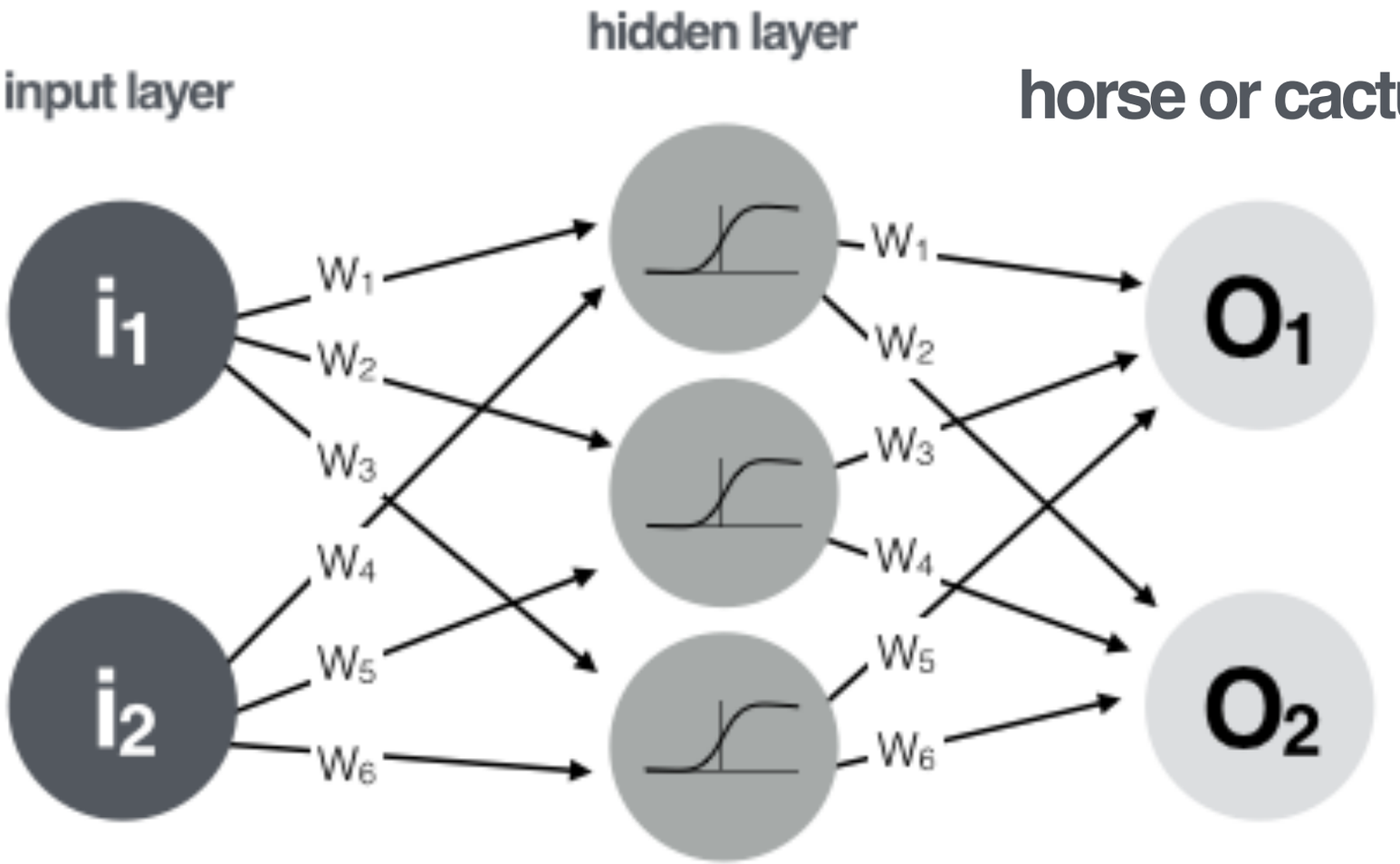
we originally wanted our computer to identify whether something was a horse or a cactus



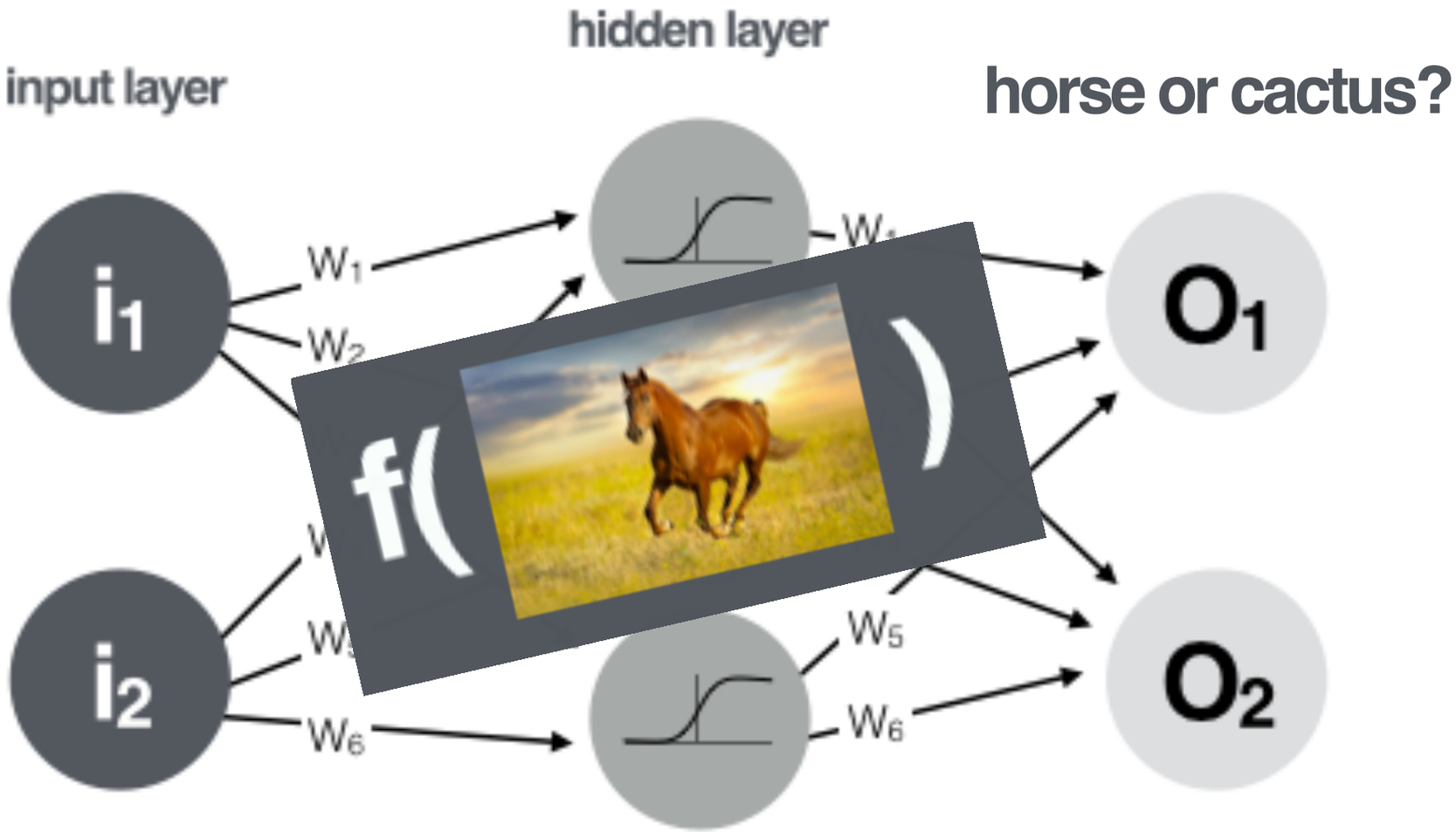
that’s a cactus



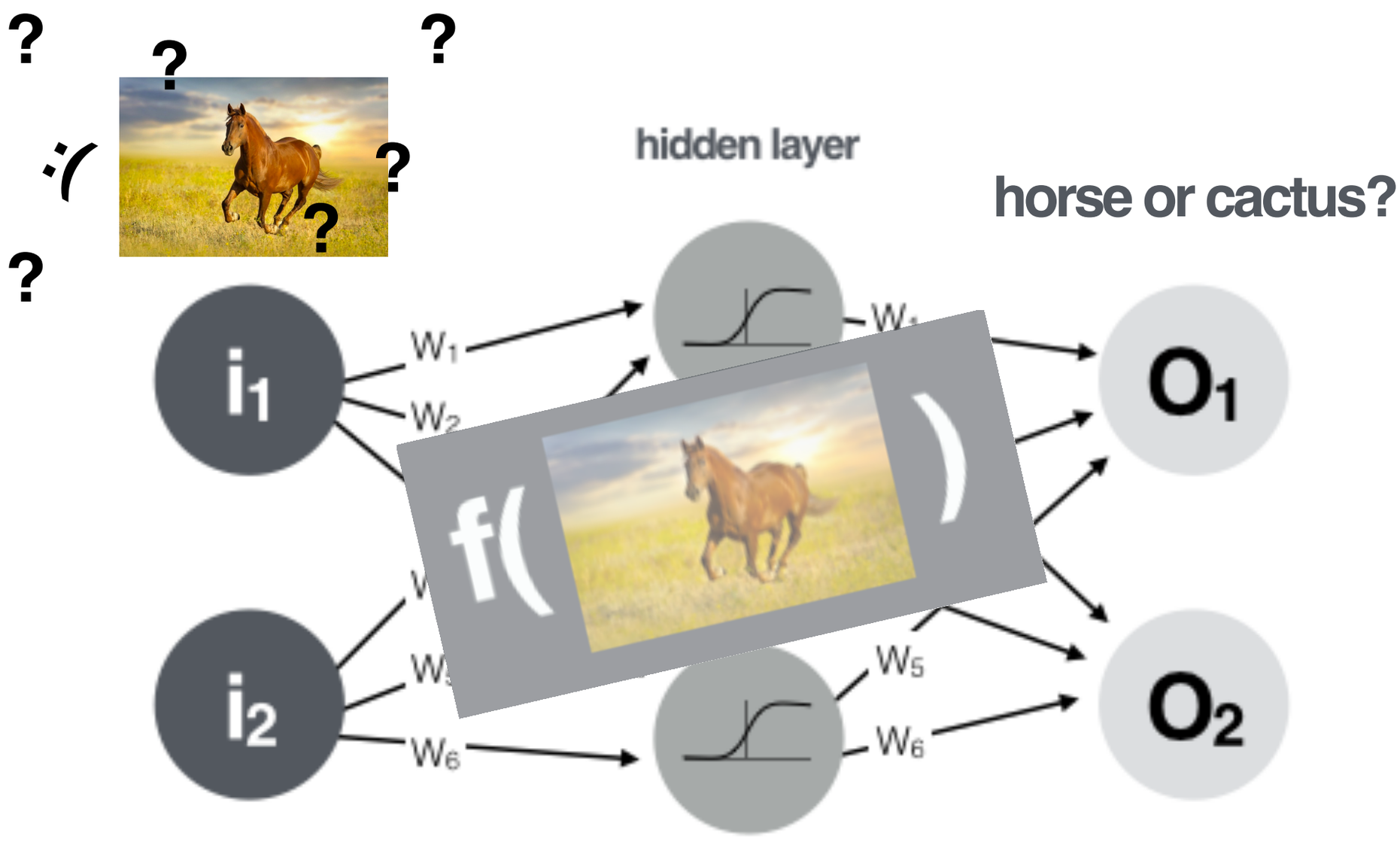
We’re actually pretty close. We have a NN that can learn a function given some input data.



We know we want the output to be similar to our Mac/PC model. (i.e. $O_1 = 1$ for horse, $O_2 = 1$ for cactus)



The function we want to approximate is, given an image, does it contain a horse or a cactus?



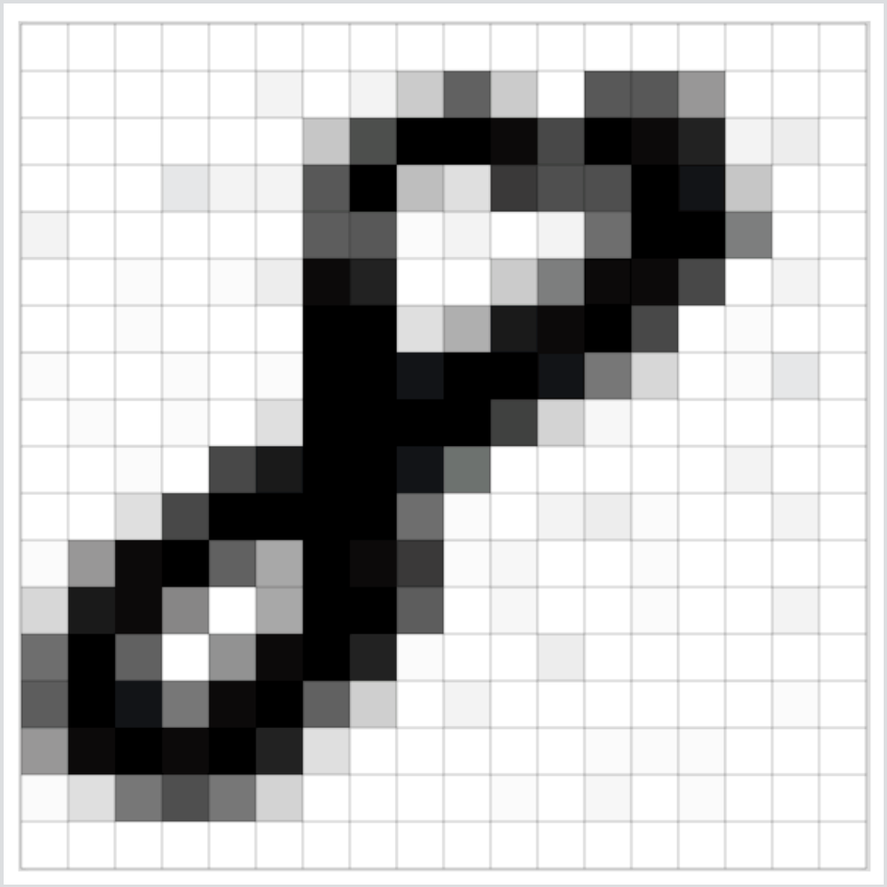
Our only problem now is how to taken an *image* as input.
Our simple NN can only take numeric values.

we need to take one more side-tour.

let's talk MNIST.

MNIST dataset

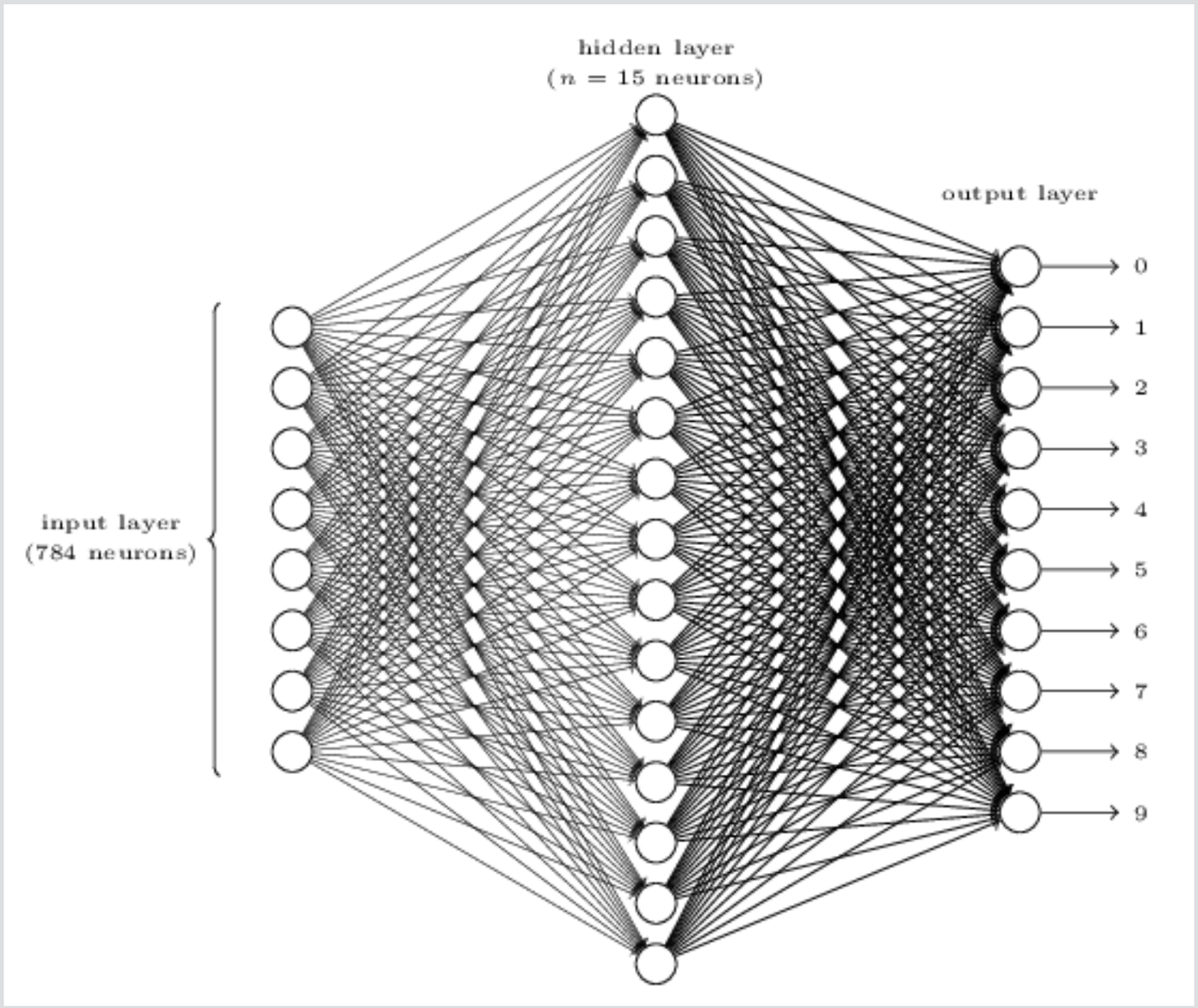
Each sample in the MNIST dataset is a 28x28 pixel image of a handwritten digit. The image is greyscale, meaning each pixel is represented in a computer as either a 0 to 255 (representing how dark the pixel is). The dataset contains 60,000 samples for training, and 10,000 samples for testing.



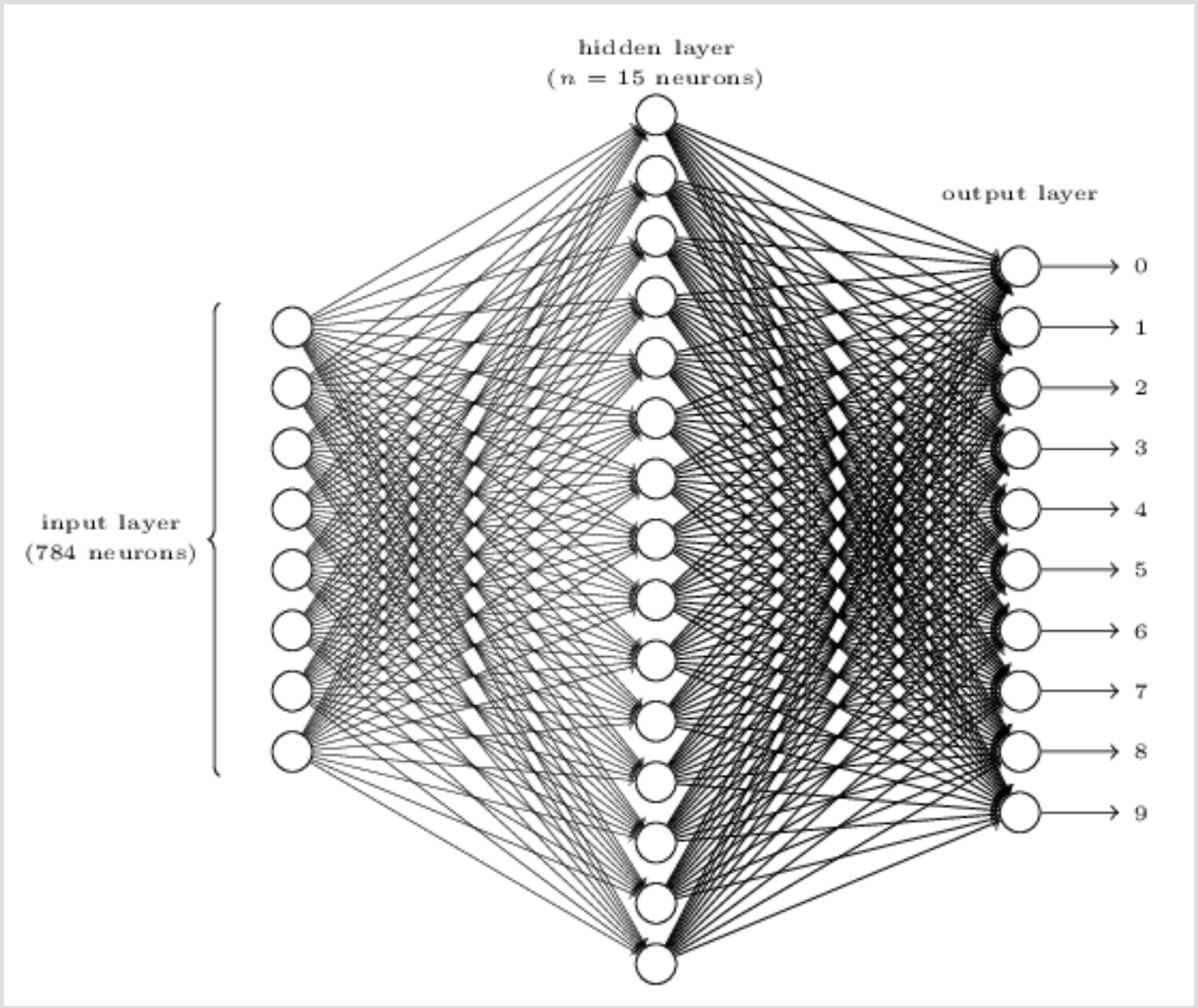
```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   1  12   0  11  39 137  37   0 152 147  84   0   0   0
  0   0   1   0   0   0  41 160 250 255 235 162 255 238 206  11  13   0
  0   0   0  16   9   9 150 251  45  21 184 159 154 255 233  40   0   0
 10   0   0   0   0   0 145 146   3  10   0  11 124 253 255 107   0   0
  0   0   3   0   4  15 236 216   0   0  38 109 247 240 169   0  11   0
  1   0   2   0   0   0 253 253  23  62 224 241 255 164   0   5   0   0
  6   0   0   4   0   3 252 250 228 255 255 234 112  28   0   2  17   0
  0   2   1   4   0  21 255 253 251 255 172  31   8   0   1   0   0   0
  0   0   4   0 163 225 251 255 229 120   0   0   0   0   0  11   0   0
  0   0  21 162 255 255 254 255 126   6   0  10  14   6   0   0   9   0
  3  79 242 255 141  66 255 245 189   7   8   0   0   5   0   0   0   0
 26 221 237  98   0  67 251 255 144   0   8   0   0   7   0   0  11   0
125 255 141   0  87 244 255 208   3   0   0  13   0   1   0   1   0   0
145 248 228 116 235 255 141  34   0  11   0   1   0   0   0   1   3   0
 85 237 253 246 255 210  21   1   0   1   0   0   6   2   4   0   0   0
  6  23 112 157 114  32   0   0   0   0   2   0   8   0   7   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

MNIST dataset

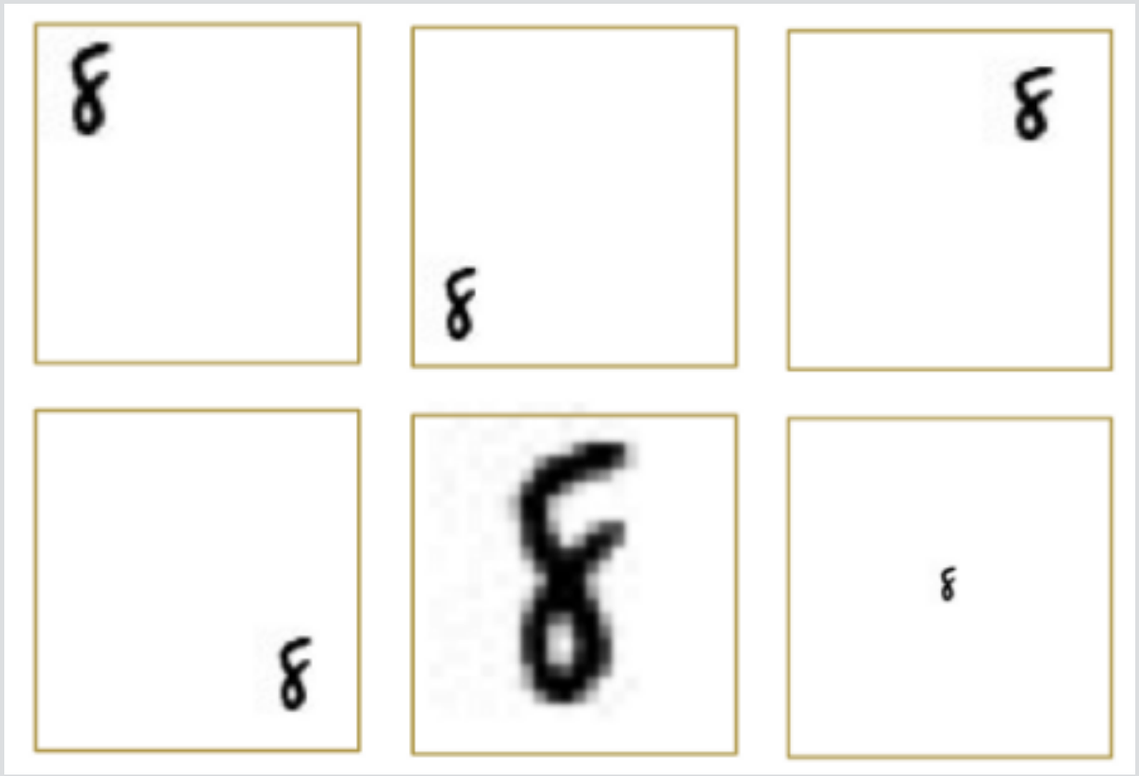
Each sample in the MNIST dataset is a 28x28 pixel image of a handwritten digit. The image is greyscale, meaning each pixel is represented in a computer as either a 0 to 255 (representing how dark the pixel is). The dataset contains 60,000 samples for training, and 10,000 samples for testing.



We can make the input nodes correspond to the pixels of the image. One node for each pixel (which will hold either a zero or a one). Each MNIST image is 28x28 pixels, meaning we will have 784 input nodes.



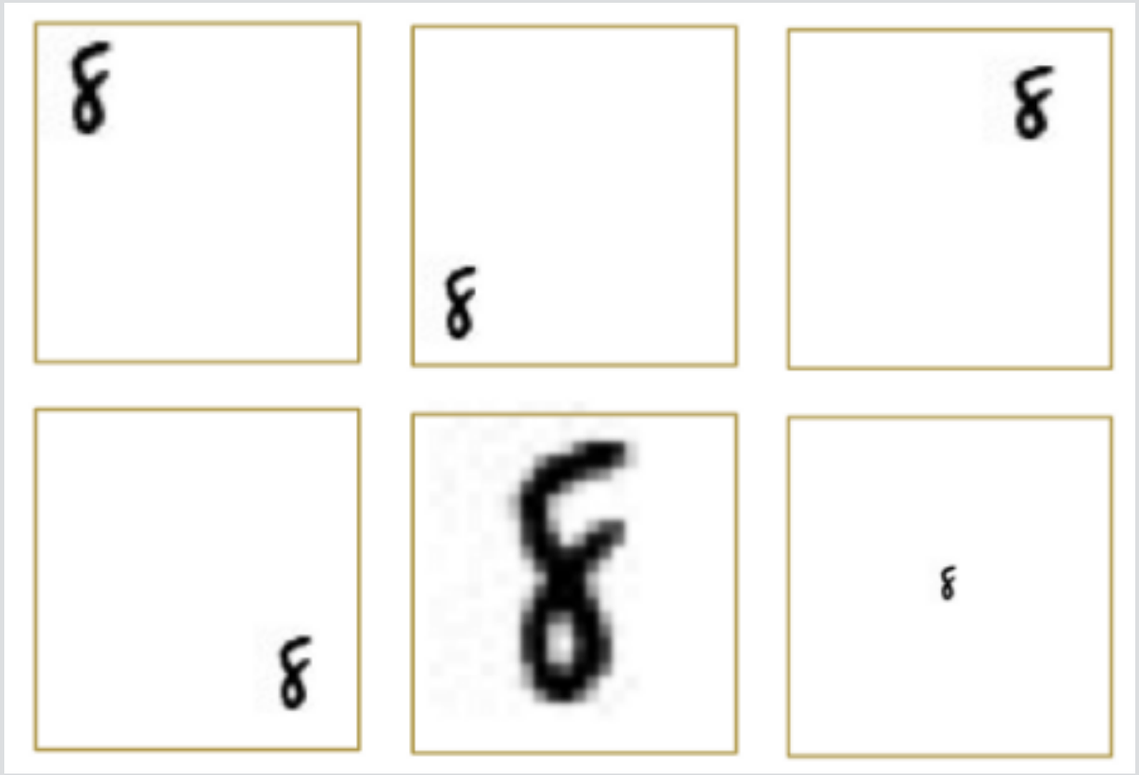
The output in this case would be similar to previous examples, with one node for each class (digit [0-9]). This setup will work great for the MNIST dataset. If we feed it all the inputs and use backprop, we will get great results on the test samples.



Our net would fail miserably on the image, because they aren't perfectly centered, and vary in size & shape

Tunnel Vision

Ok, so we can training our net on the MNIST data set using the simple back propagation method we looked at earlier. But our NN is kind of a derp now, because it can only identify a digit when it's perfectly centered in the image. That sucks.



We want to achieve what is called **translation invariance.** (i.e. an eight is an eight no matter what)

Some bad ideas

We could just provide the net with all kinds of training samples that show the different digits in all sorts of varied sizes and locations. But intuitively, we know that doesn't make any sense — we want the net to treat the eight like Stewart treated pornography: I know it when I see it. Giving it different types of locations and sizes makes it treat different eights differently, and it won't be able to find eights in unfamiliar locations or sizes.

Also: a fully-connected NN that had a node for each pixel would be terribly, terribly large.

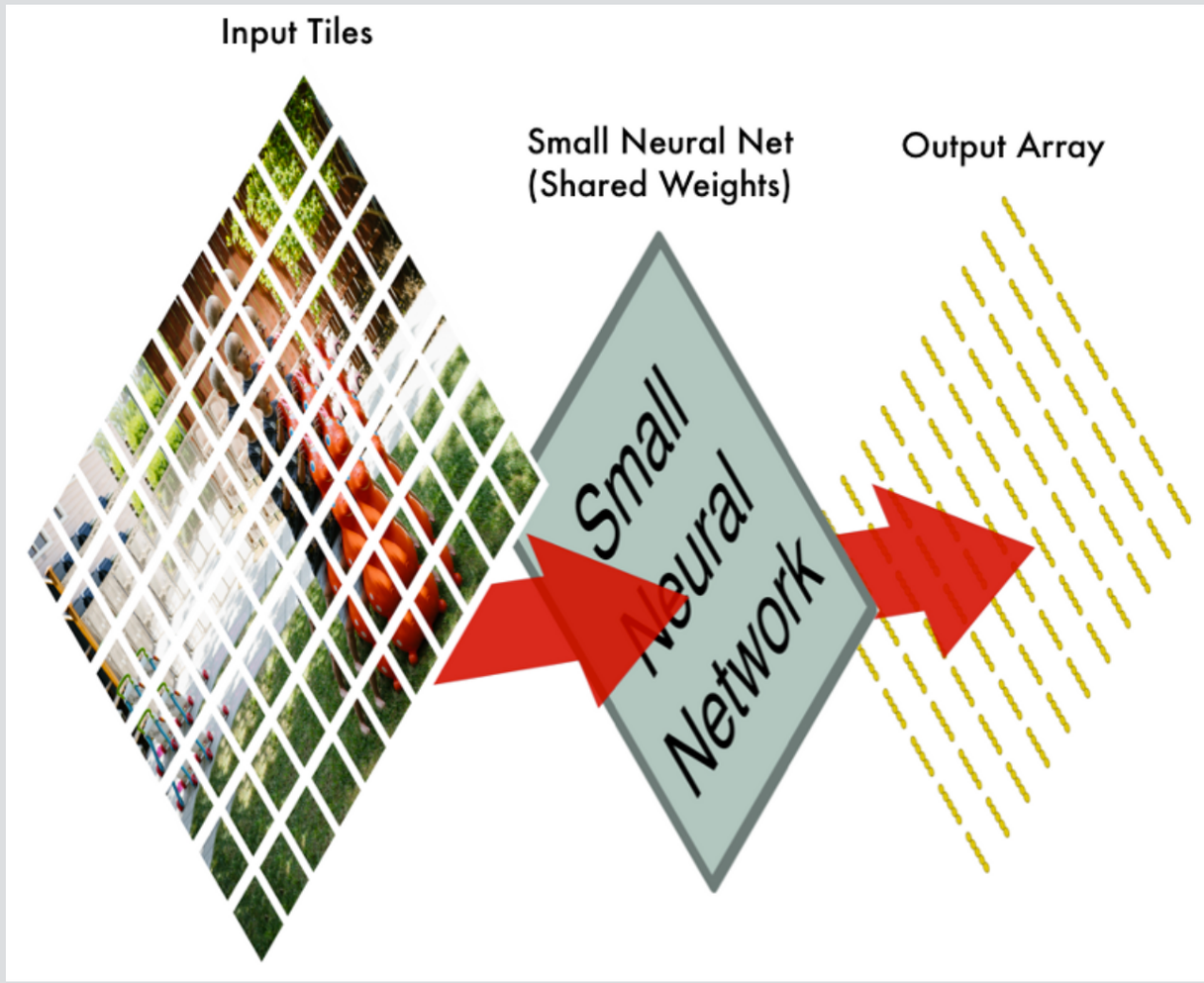
Computers are fast, but not that fast :(

Intuition:

Images have a hierarchy. This means that the pixels that are closer to each other are more closely related than the pixels further away. Currently, our net treats all pixels as equally related.
If we “scan” the image from left to right, we will be able to look at smaller parts of the image and look for the things we are most interested in.

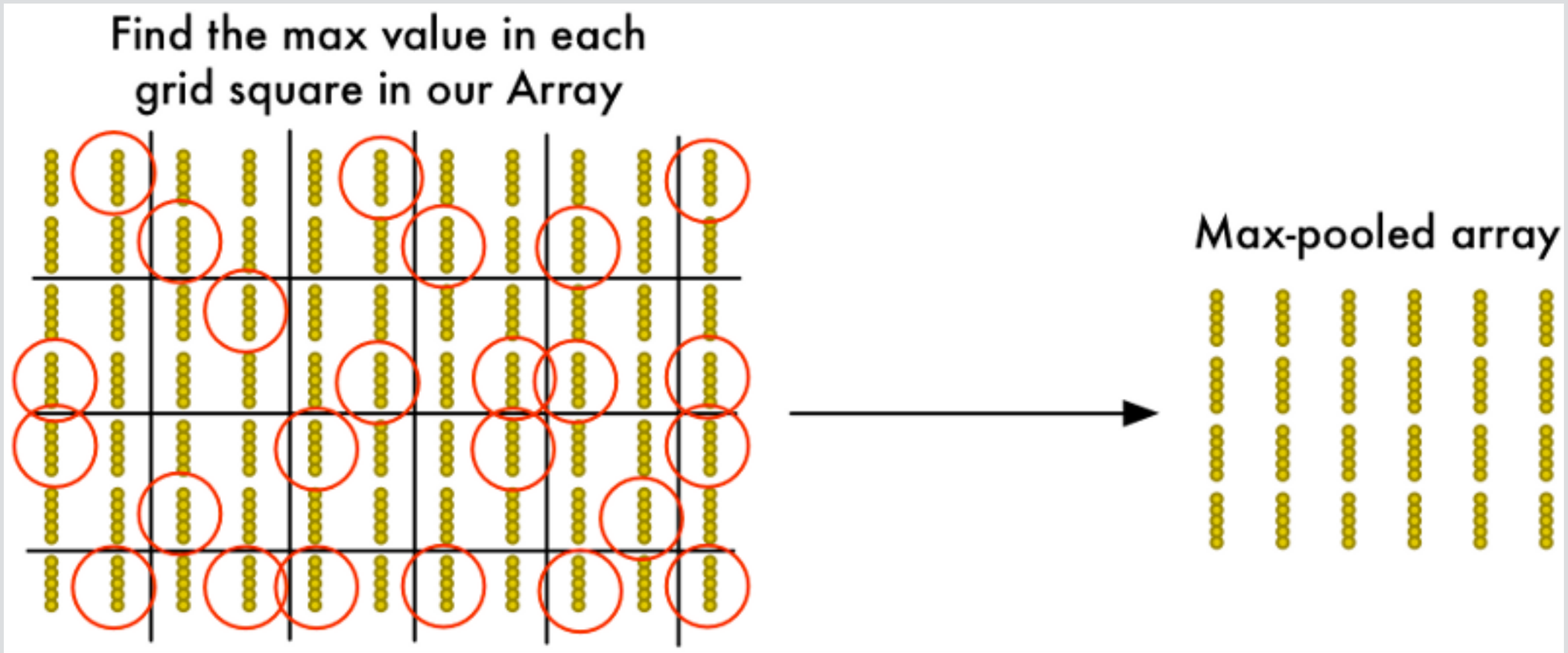
Convolution

This eliminates the problem of “where is it” on the image, because the NN is essentially scanning the image to look for the important features of the image. We will be feeding each tile into a small neural network. The weights will be updated for each tile, meaning that the net learn to look for important features in the image. This is known as convolution.



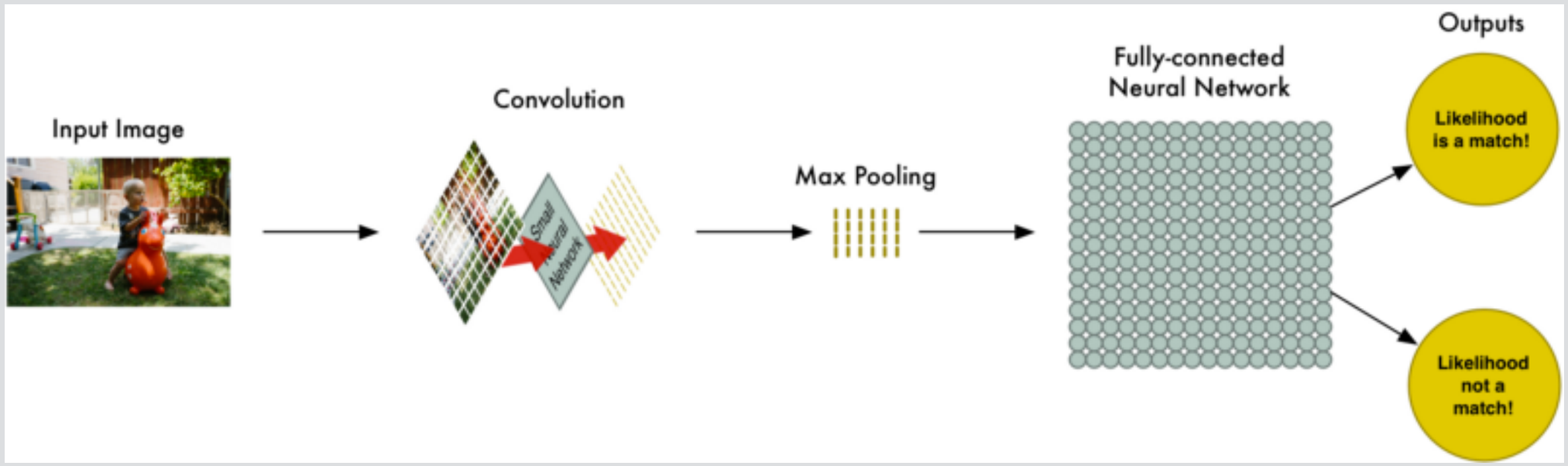
Convolution

We want to keep track of the order of all tiles (as to not jumble the image) so we will store the output of each tile in an array. This array now contains the important features of each tile. Now we just need to **downsample**.



Max Pooling

In max-pooling, we essentially take the most interesting parts of each chunk of our output array. This makes a smaller array that we can use to make our image classification predictions.



Putting it together

So, the input image is first fed through a convolution layer that “scans” the image via small tiles. These tiles are all fed through a neural net that will be trained to pick out the important parts of the tiles. The results of each tile’s run through the net will be stored in an output array, that we then apply the max-pooling algorithm in order to condense the array for the most interesting bits.

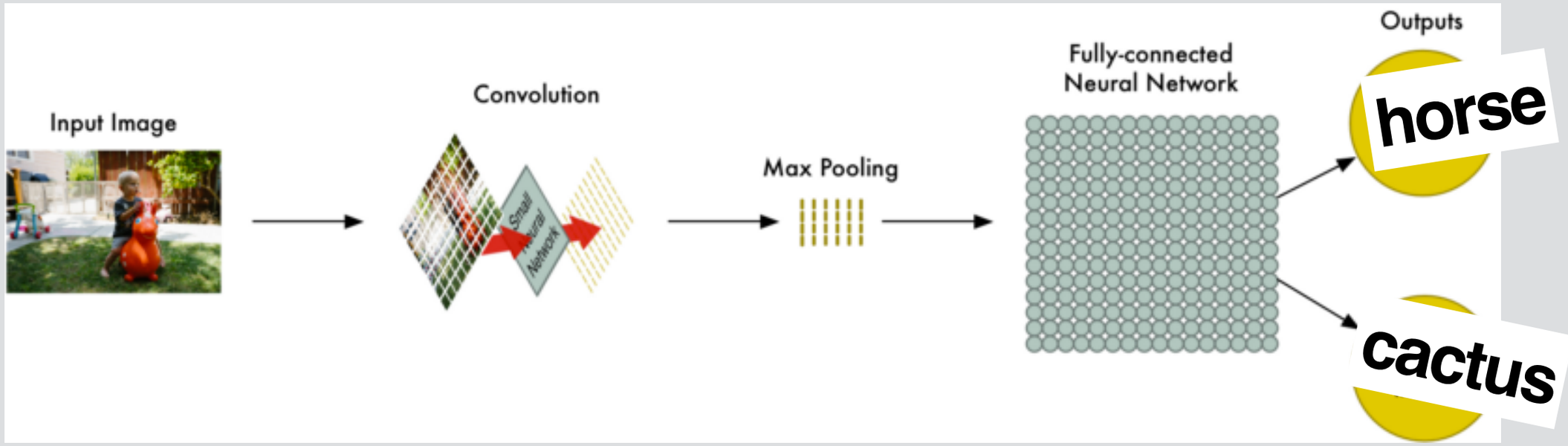
That max-pooled array will be an input to a fully-connected net that will be used to make our classification prediction.

wow! so simple!

i'm not overwhelmed at all!

but wait! we're not done!

don't forget our horses!



Ok, our CNN set up will actually work great with our horse images. The only thing left to do is deal with color. Originally, we had greyscale images (so each input node would represent one pixel, holding a value between 0 and 255). All we have to do is have three input nodes for each pixel, one for each of the three RGB values. That's it.

so that's the super basics. if you'd like to learn more, i'll put some resources to look over in my deliverable :)

questions?