Machine Learning Worksheet—Perceptron, Neuron

Background

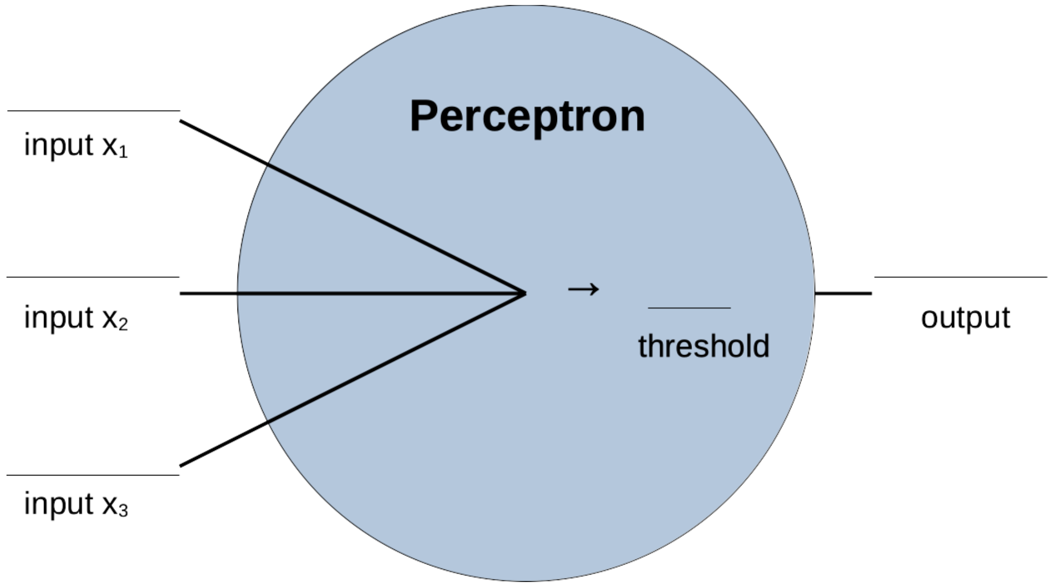
Machine learning typically occurs via a neural network, and the fundamental building block of a neural network is the *neuron*. A neuron is modeled very roughly on the biological neurons in a brain: input signals from dendrites potentially activate the neuron's soma, in which case a signal is sent as output via axons.

In this activity we'll be exploring the operating principles of a *perceptron* and a neuron. The perceptron, initially conceived of and developed in the 1950s and 60s, is a simpler model, while the neuron has some features that make it much more useful for building a neural network (NN).

Perceptron Example 1

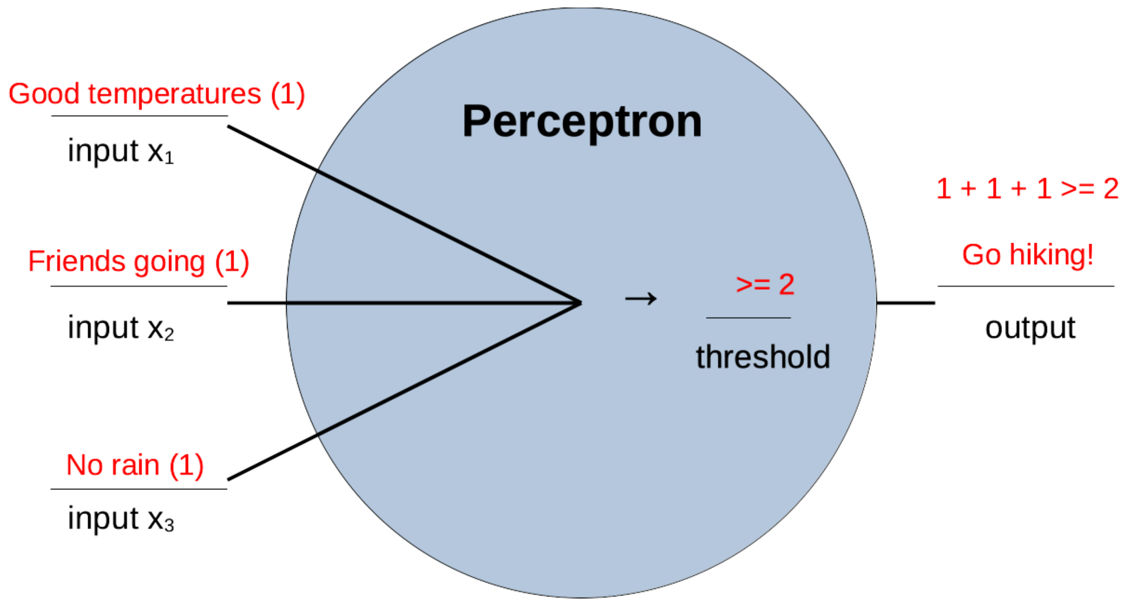
You've decided that you might want to go for a hike this weekend, but only under certain conditions. If the temperatures are going to be right (not too hot, not too cold), that would be good. If you've got some friends that want to join you, that would be even better. Also, you'd prefer to *not* go hiking in the rain. None of these circumstances would be an absolute "deal-breaker" on its own, but if it's raining and you're hiking alone, well... that doesn't sound like much fun. Let's say that you'll go hiking as long as two of these three conditions are satisfied.

It would be possible to write a series of conditional statements to help analyze the situation, but let's write this using a *perceptron* instead. The perceptron will have three binary *inputs*, one for each of the factors you're evaluating, and a minimum *threshold* value that we have to meet if we're going to go hiking. The binary *result* will indicate that we should either go hiking, or not go hiking.

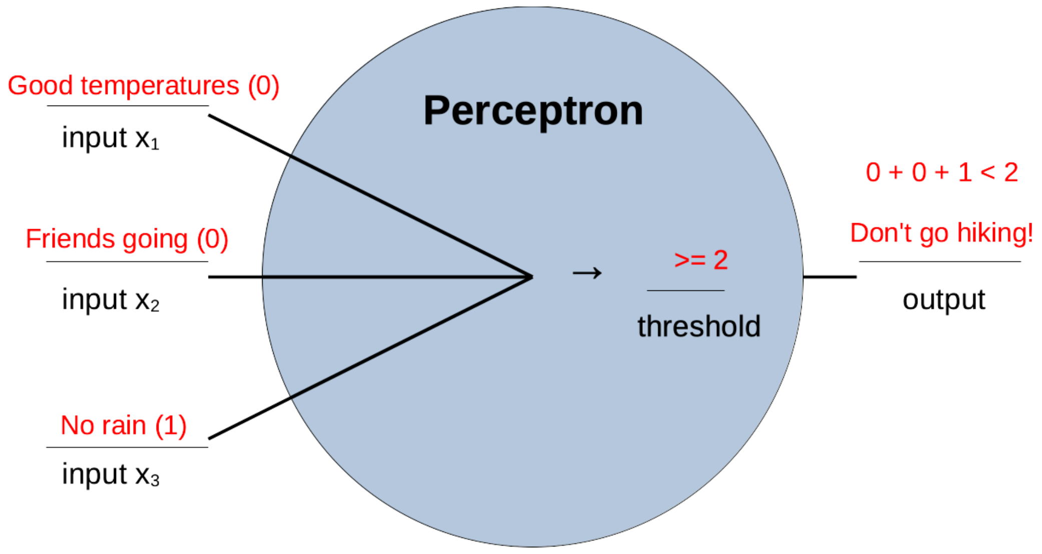


What does this look like for our hiking scenario? We can label each of the input conditions on the left side, and for a given scenario, assign them a value of 0 or 1 (false or true), depending on the specifics of that scenario. The threshold condition is on the right. To decide whether or not we go hiking, you sum the values of the inputs, see if they meet the threshold requirement, and then identify the result.

Let's consider this example. In the scenario shown here, all three of the inputs have a value of 1: the *good temperatures* input has a value of 1, meaning that we have good temps. *Friends can join* has a value of 1, meaning we'll have company. And *no rain* has a value of 1, meaning "yes, we'll have no rain." The sum of our inputs is 3, which is greater than or equal to 2, so we should go hiking.

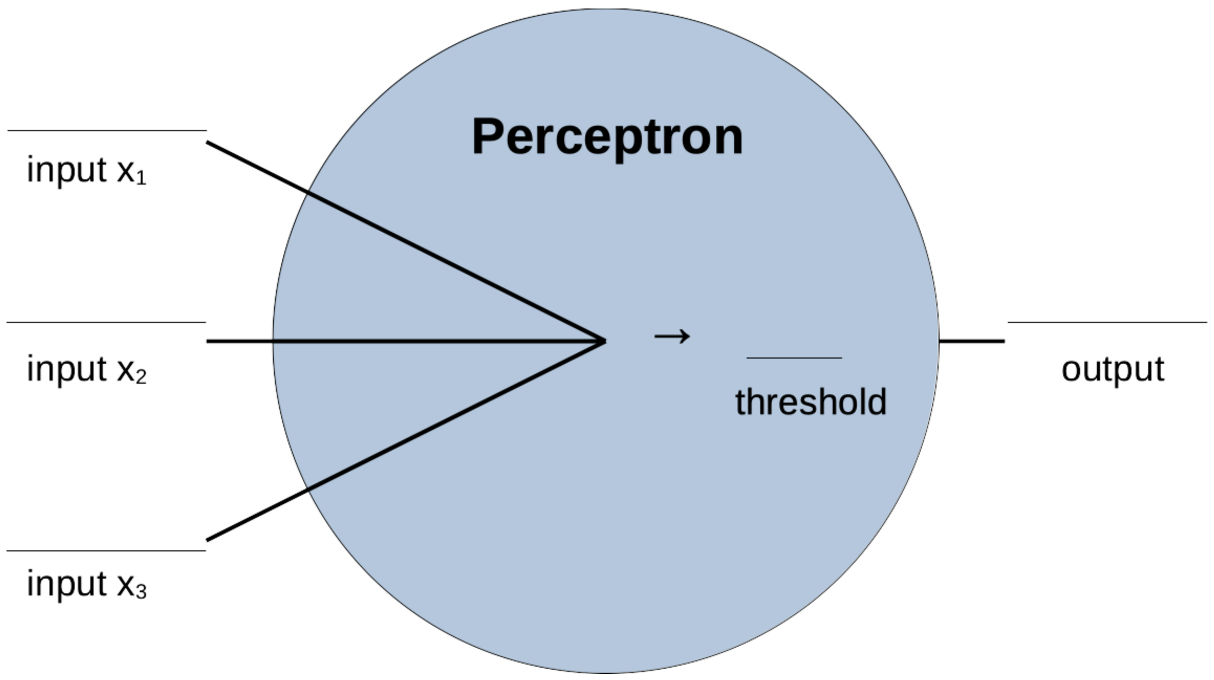


If our good temperatures condition was not met, and friends couldn't join, the diagram would look like this:



Activity 1

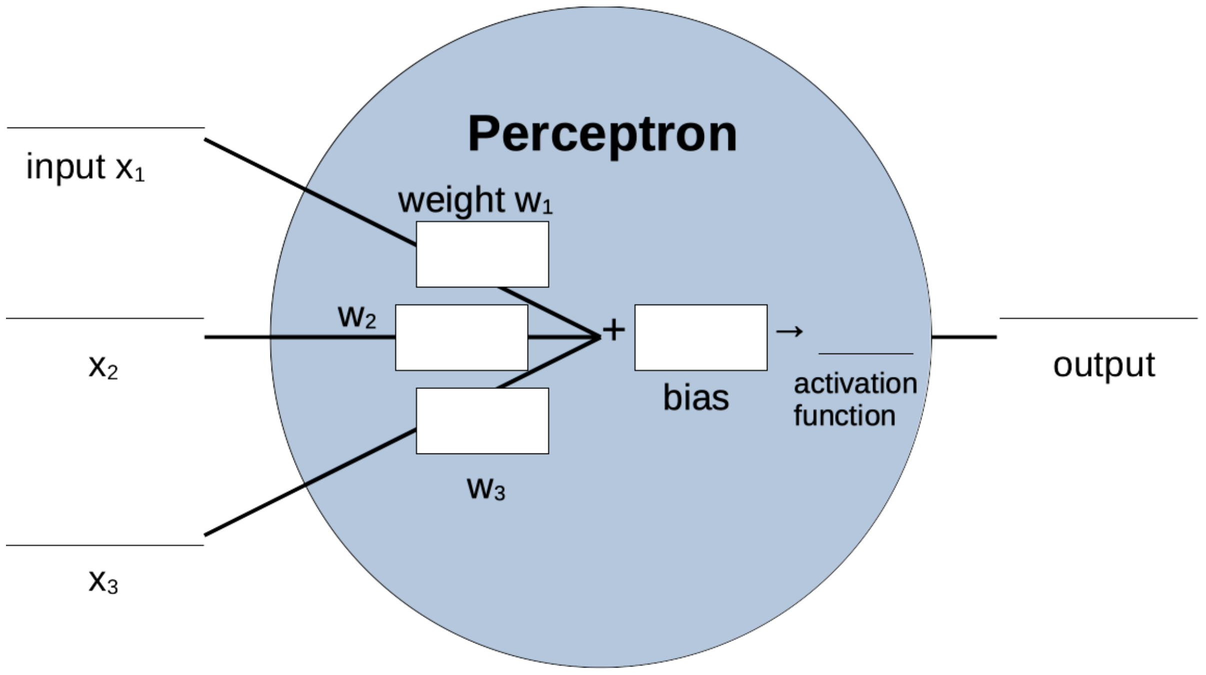
Create your own Perceptron based on a simple set of 3-4 binary input conditions on the left, a threshold value on the right, and a binary outcome.



Perceptron Example 2

The binary (0 or 1) inputs that we've used to this point are convenient, but they lack the ability to "fine-tune" our decision making. If "hiking with friends" is a little more important to us than the "no rain" requirement, we'd like to be able to *weight* that aspect of the decision more heavily.

We can modify our perceptron to do just that by introducing a *weight* factor for each of the inputs:



$x_1w_1 + x_2w_2 + x_3w_3$ + bias → activation function → output

When we perform the calculation, we'll take each input *x* and multiply it by its corresponding weight *w* before adding all the *xw* terms together.

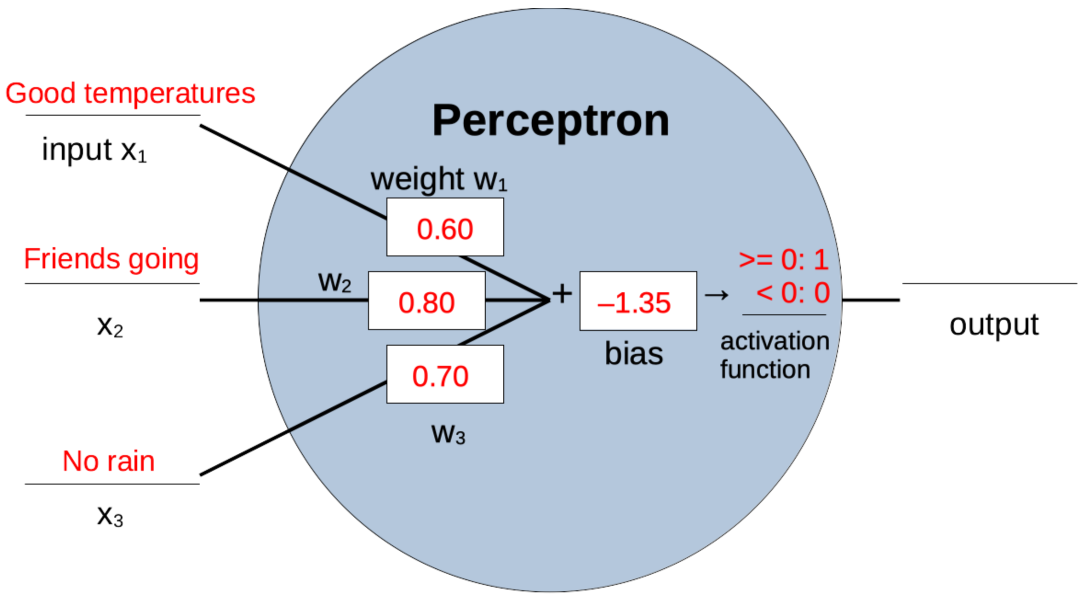
What should we choose for our weights? As I think about hiking, the most important of the three factors is whether or not friends are going, so I'll give that a slightly higher weight of **0.80**, a value that I selected somewhat randomly. I'm also not a big fan of the rain, so I'll give that a pretty good weight as well: **0.70**. I like good temperatures, but that's the least important factor, so I'll give it the lowest weight, **0.60**. See the diagram below for how these weights were placed into the perceptron.

(Again, these weights were arrived at somewhat randomly. If you're wondering how you would go about selecting your own weights, we'll answer that question in a moment.)

We're also going to modify the threshold slightly: rather than calculating a result and comparing it to the threshold, we'll use a *bias* value to modify our result. For this example I chose (again, somewhat randomly) a bias of **–1.35**. When that negative value is added to our calculation, the result will be closer to 0, either positive or negative. If my inputs and weights, added to the bias, give me positive result, I'll go hiking. If I get a negative result, I won't go hiking.

This interpretation of the results can be formalized in an *activation function*, which tells us how to determine if the output is **1** (going hiking) or **0** (not going hiking). Here, that function is simply: *if result >= 0: 1 (go hiking), else (result < 0): 0 (don't go hiking).*

Let's take a look at our new perceptron.



$x_1w_1 + x_2w_2 + x_3w_3$ + bias → activation function → output

Take a look at this perceptron. If friends can join (1) and there's no rain (1) but the temperature isn't good (0), what is the numeric result? Will our perceptron recommend that we go hiking?

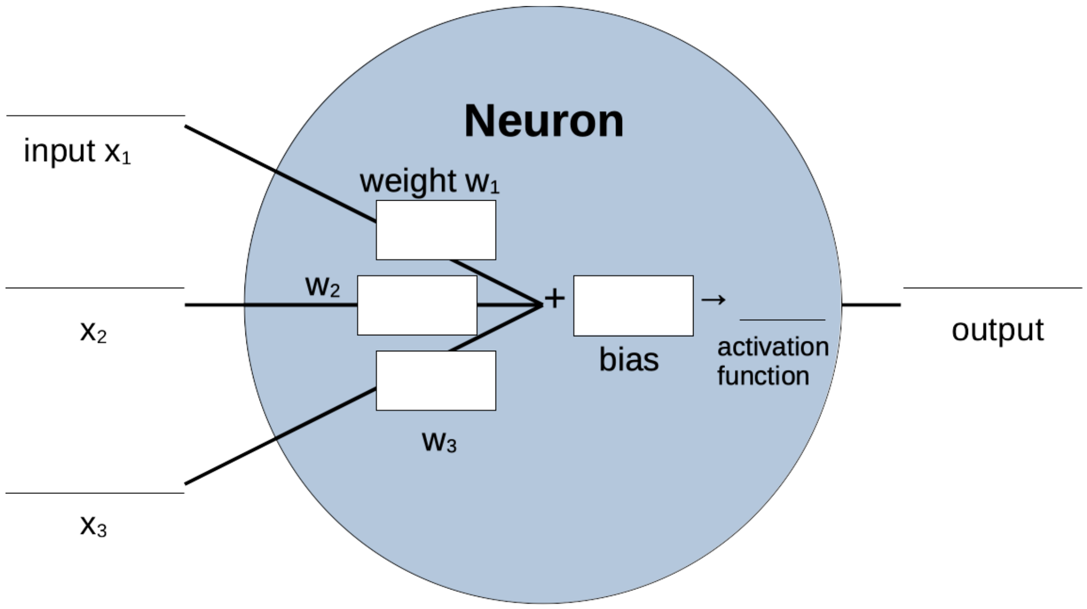
What if there are good temps (1) and no rain (1), but friends can't go (0)? What is the numeric result? Will the perceptron recommend that we hike?

Neuron Example

Our perceptron example to this point has a simple activation function that is based on that threshold (bias) value, but this is a limitation in our model. A *neuron* is similar to a perceptron but with the following important differences: the activation function is non-linear, and the output might be more than just a binary result (although we still use binary outputs sometimes). Those distinctions might not make much sense at this point, but you'll understand a bit more after we look at this next example.

Machine Learning **Worksheet—Perceptron, Neuron**

The basic graphical format of our neuron diagram remains the same as that of the perceptron:



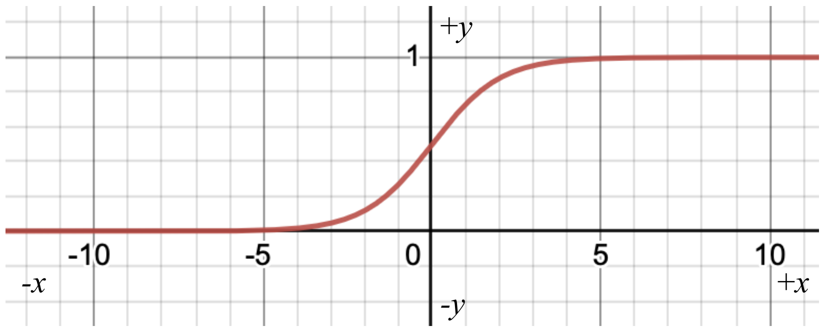
$x_1w_1 + x_2w_2 + x_3w_3$ + bias → activation function → output

The difference is in our activation function and outputs. One activation function that is commonly used when learning about neural networks is the *sigmoid function* (sometimes indicated with a lowercase sigma, σ).

$$y = f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

Here's what the function says: take any positive *x* value, and you'll get a *y* that approaches a maximum value of 1 as *x* approaches +∞; take any negative *x* value, and you'll get a *y* that approaches 0 as *x* approaches -∞.

In this graph of the sigmoid function, you can see that positive values of *x* as they move to the right approach a *y* result of 1, while negative values of *x* approach a *y* result of 0. The results of the sigmoid function will be a value such as 0.21, or 0.923, some decimal value between 0 and 1.



Machine Learning — Worksheet—Perceptron, Neuron

Let's calculate a few sample sigmoid function outputs using different values.

1. Example: Using $f(x)=\sigma(x)=\frac{1}{1+e^{-x}}$, calculate σ(1).

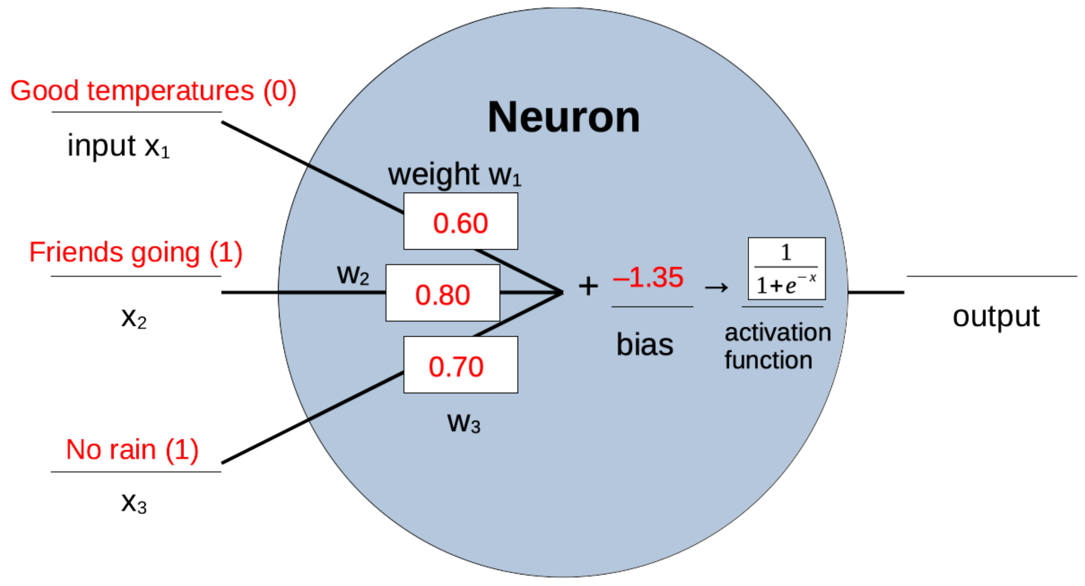
   Answer: Use a calculator to calculate $\frac{1}{1+e^{-x}}=\frac{1}{1+e^{-1}}=0.7311$
2. Calculate σ(5). ____________
3. Calculate σ(10). ____________
4. Calculate σ(–1). (Watch your signs on the exponent here!) ____________
5. Calculate σ(–5). ____________
6. Calculate σ(–10). ____________

Based on our results, you can see that a broad range of *x* values are being converted to a small range of results, all in the range 0-1. This is a very useful strategy for something like our hiking neuron, which is intended to identify one of two different situations: hiking or not hiking.

In machine learning, this kind of structure is called a *binary classifier.*

Let's go hiking...?

Let's look again at our hiking situation. Let's apply our neuron model to the conditions *temps not good, friends going, no rain* (input values of $x_1 = 0$, $x_2 = 1$, $x_3 = 1$).



$x_1w_1 + x_2w_2 + x_3w_3 + \text{bias} \rightarrow$ activation function $\rightarrow$ output

Use those 0, 1, 1, inputs with the weights of 0.60, 0.80, 0.70 and the bias of -1.35 first:

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$(0 \bullet 0.60) + (1 \bullet 0.80) + (1 \bullet 0.70) + -1.35 = 0.15$$

Now run that through the sigmoid function to get our output:

$$output = \sigma(0.15)$$

$$output = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-0.15}}$$

output = ______________

If you used a calculator, you'll find that the output of the activation function is **0.5374**. What does that result mean for us? If 0 is *don't go* hiking and 1 is *go* hiking, it appears as if this neuron is telling us that it's only slightly recommending that we go hiking—the output is slightly closer to 1 than it is to 0. This despite the fact that two of our concerns (going with friends and no rain in the forecast) have input values of 1, and those are the most heavily-weighted items.

If I had to guess, I'd say that this neuron isn't working quite as well as I thought it would.

How can we make it better?

Manually adjusting weights and bias

If the neuron isn't producing the kind of hiking advice that we think it should, we need to make some adjustments.

You might have already guessed at least part of the solution: if we were to tweak the values for our *weights* and *bias* we could almost certainly get the neuron to do a better job of identifying our hiking recommendation.

Take a moment to adjust one or more of the weights and bias in our neuron, and check to see you get a better output result. *Also*, try different input combinations—0, 1, 0, or 1, 1, 0—to see if the outputs work well for those situations, too.

Computers can do better

Ultimately, we want to automate this process. A computer can take a given set of inputs, weights, and biases, run the numbers, and see how well the output matches our expectations. Then, it adjusts the weights and bias in a systematic way, and checks to see if the output is any closer to matching our expectations. This process of identifying the weights-and-biases that turn inputs into correct outputs is one type of machine learning, called *supervised learning* because we're explicitly training the computer on what we'd like it to learn.