deeplizard.com

Activation Functions in a Neural Network explained

15-19 minutes

Activation functions in a neural network

In this post, we'll be discussing what an activation function is and how we use these functions in neural networks. We'll also look at a couple of different activation functions, and we'll see how to specify an activation function in code with Keras.

In a previous post, we covered the layers within a neural network, and we just briefly mentioned that an activation function typically follows a layer, so let's go a bit further with this idea.

What is an activation function?

Let's give a definition for an activation function:

In an artificial neural network, an activation function is a function that maps a node's inputs to its corresponding output.

This makes sense given the illustration we saw in the previous post on layers. We took the weighted sum of each

incoming connection for each node in the layer, and passed that weighted sum to an activation function.

node output = activation(weighted sum of inputs)

The activation function does some type of operation to transform the sum to a number that is often times between some lower limit and some upper limit. This transformation is often a non-linear transformation. Keep this in mind because this will come up again by the time this post reaches its conclusion.

What do activation functions do?

What's up with this activation function transformation? What's the intuition? To explain this, let's first look at some example activation functions.

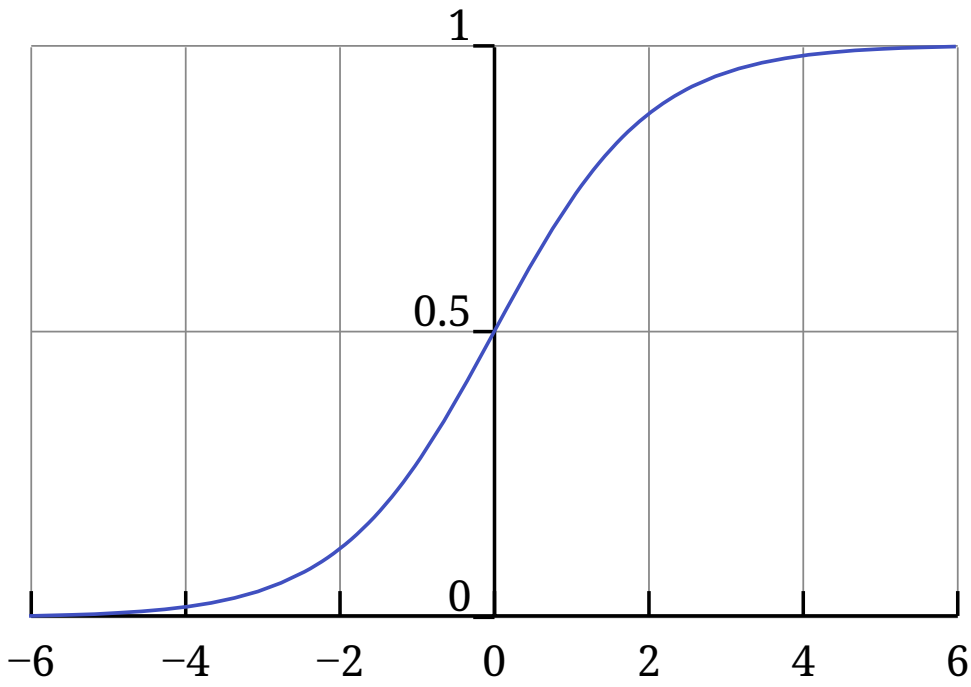
Sigmoid activation function

Sigmoid takes in an input and does the following:

- For most negative inputs, sigmoid will transform the input to a number very close to 0.
- For most positive inputs, sigmoid will transform the input into a number very close to 1.
- For inputs relatively close to 0, sigmoid will transform the input into some number between 0 and 1.

Mathematically, we write

$$sigmoid(x) = \frac{e^x}{e^x + 1}$$



So, for sigmoid, 0 is the lower limit, and 1 is the upper limit.

Alright, we now understand mathematically what one of these activation functions does, but what's the intuition?

Activation function intuition

Well, an activation function is biologically inspired by activity in our brains where different neurons fire (or are activated) by different stimuli.

For example, if you smell something pleasant, like freshly baked cookies, certain neurons in your brain will fire and

become activated. If you smell something unpleasant, like spoiled milk, this will cause other neurons in your brain to fire.

Deep within the folds of our brains, certain neurons are either firing or they're not. This can be represented by a 0 for not firing or a 1 for firing.

```
// pseudocode
if (smell.isPleasant()) {
    neuron.fire();
}
```

With the Sigmoid activation function in an artificial neural network, we have seen that the neuron can be between 0 and 1, and the closer to 1, the more activated that neuron is while the closer to 0 the less activated that neuron is.

Relu activation function

Now, it's not always the case that our activation function is going to do a transformation on an input to be between 0 and 1.

In fact, one of the most widely used activation functions today called *ReLU* doesn't do this. ReLU, which is short for *rectified linear unit*, transforms the input to the maximum of either 0 or the input itself.

$$relu(x) = max(0, x)$$

So if the input is less than or equal to 0, then relu will output 0. If the input is greater than 0, relu will then just output the given input.

```
// pseudocode
function relu(x) {
    if (x <= 0) {
        return 0;
    } else {
        return x;
    }
}
```

The idea here is, the more positive the neuron is, the more activated it is. Now, we've only talked about two activation functions here, sigmoid and relu, but there are other types of activation functions that do different types of transformations to their inputs.

Why do we use activation functions?

To understand why we use activation functions, we need to first understand linear functions.

Suppose that f is a function on a set X.
Suppose that a and b are in X.
Suppose that x is a real number.

The function f is said to be a linear function if and only if:

$$f(a + b) = f(a) + f(b)$$

and

$$f(xa) = xf(a)$$

An important feature of linear functions is that the composition of two linear functions is also a linear function. This means that, even in very deep neural networks, if we

only had linear transformations of our data values during a forward pass, the learned mapping in our network from input to output would also be linear.

Typically, the types of mappings that we are aiming to learn with our deep neural networks are more complex than simple linear mappings.

This is where activation functions come in. Most activation functions are non-linear, and they are chosen in this way on purpose. Having non-linear activation functions allows our neural networks to compute arbitrarily complex functions.

Proof that Relu is non-linear

Let's prove that the relu activation function is non-linear. To do this, we will show that relu fails to be linear.

For every real number x, we define a function f to be

$$f(x) = relu(x)$$

Suppose that a is a real number and that $a < 0$.

Using the fact that $a < 0$, we can see that

$$f(-1a) = max(0, -1a) > 0$$

and that

$$(-1)f(a) = (-1)max(0, a) = 0$$

This allows us to conclude that

$$f(-1a) \neq (-1)f(a)$$

Therefore, we have shown that the function f fails to be linear.

Activation functions in code with Keras

Let’s take a look at how to specify an activation function in a Keras Sequential model.

There are two basic ways to achieve this. First, we’ll import our classes.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

Now, the first way to specify an activation function is in the constructor of the layer like so:

```
model = Sequential([
    Dense(units=5, input_shape=(3,),
activation='relu')
])
```

In this case, we have a `Dense` layer and we are specifying relu as our activation function `activation='relu'`.

The second way is to add the layers and activation functions to our model after the model has been instantiated like so:

```
model = Sequential()
model.add(Dense(units=5, input_shape=(3,)))
model.add(Activation('relu'))
```

Remember that:

node output = activation(weighted sum of inputs)

For our example, this means that each output from the nodes in our `Dense` layer will be equal to the relu result of

the weighted sums like

node output = relu(weighted sum of inputs)

Hopefully now you have a general understanding about how activation functions fit into the neural network structure and what they are doing, and also how to use them in code with Keras. I'll see ya in the next one!