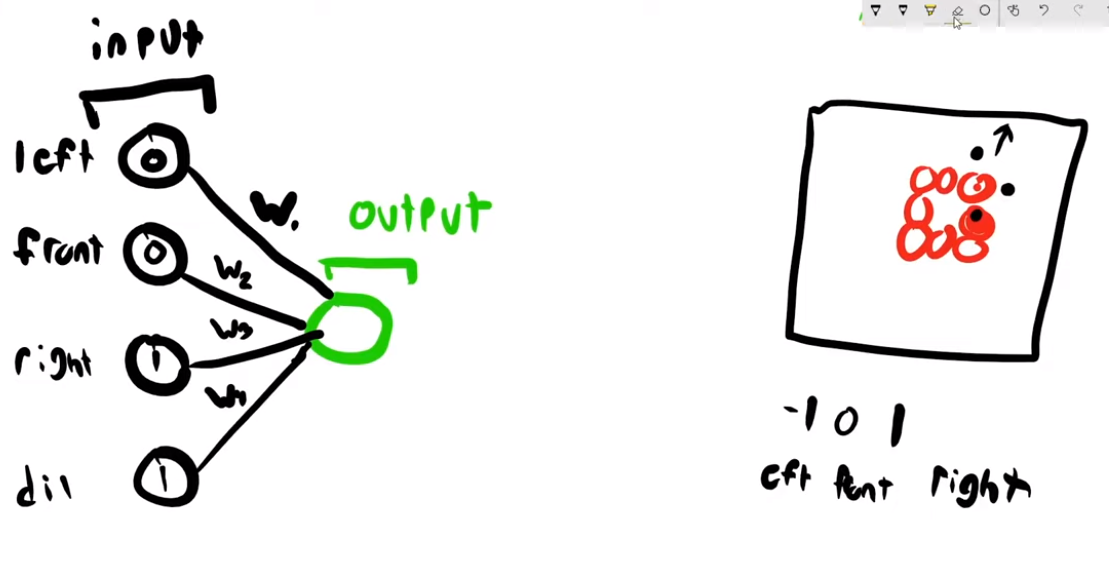
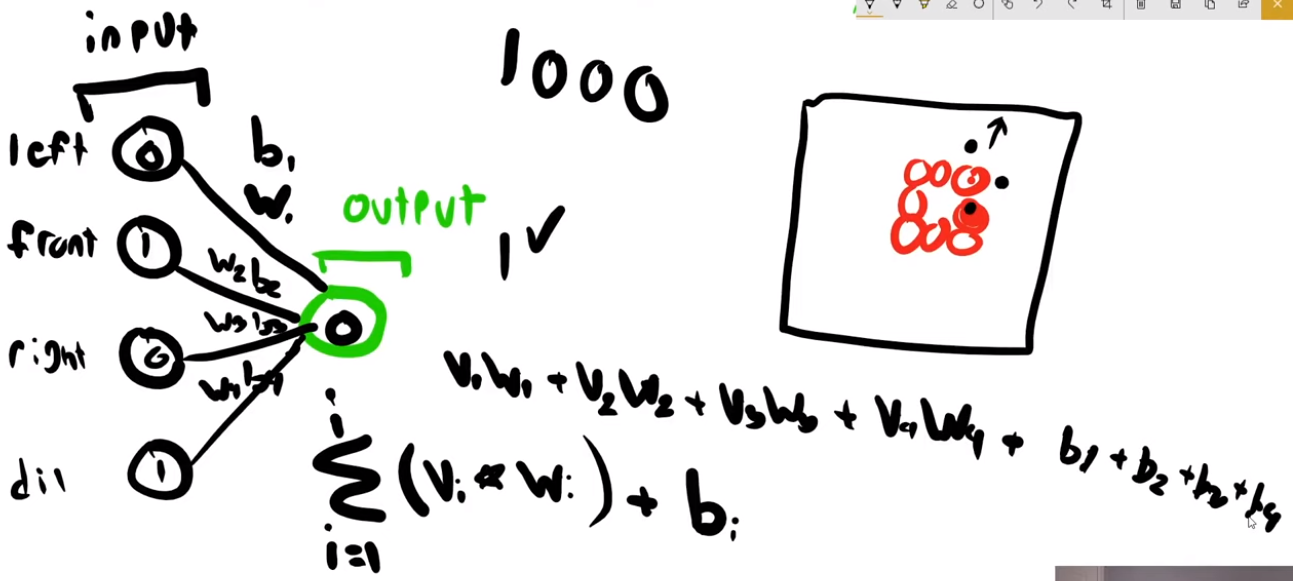
Tensor Flow

A neural network is essentially a connected layer of neurons.

In a fully connected neural network, each neuron is connected to another neuron exactly one time. The first and last layer of the neural network is called the input and output layer respectively.



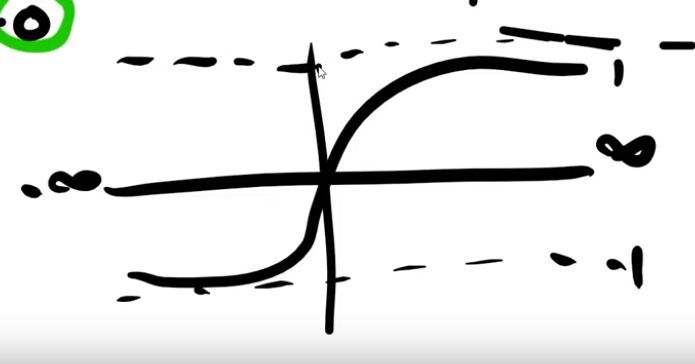
In the above example, we have a game in which our only purpose is to keep the snake alive by not letting it crash into its tail. The inputs in this case could be 3 directions (left, right and forward) and a value suggesting if there is something in that direction (1->yes and 0-> no). We could also have another output called the **recommended direction** with can contain 3 values (-1->left 0->forward 1->right). Our snake will move in the recommended direction. The value of the output will be 1 id the recommended direction causes it to die else 0 if it remains alive.



Every neuron is associated with a **weight** and **bias**. The output is basically the weighted sum of the inputs. The bias and weights are adjusted to get the output by trying every possible combination.

The above formula is similar to that of a slope. In general, we’re creating a fairly complex linear function. That is not what we want because it limits the degree of complexity that our model could have. That’s why we have an activation function. An activation functions enables to have a greater degree of complexity that will allow us to have a function that could be curvy of some sort. An example of this could be a sigmoid function.

A sigmoid function will map any value you give it between -1 and 1.



An activation function also allows us to normalize the data. This helps us save LOT of space.

Another widely used activation function is called the Rectify Linear Unit. It basically takes all the negative function and makes them 0 and takes the all the positive function and makes them more positive.

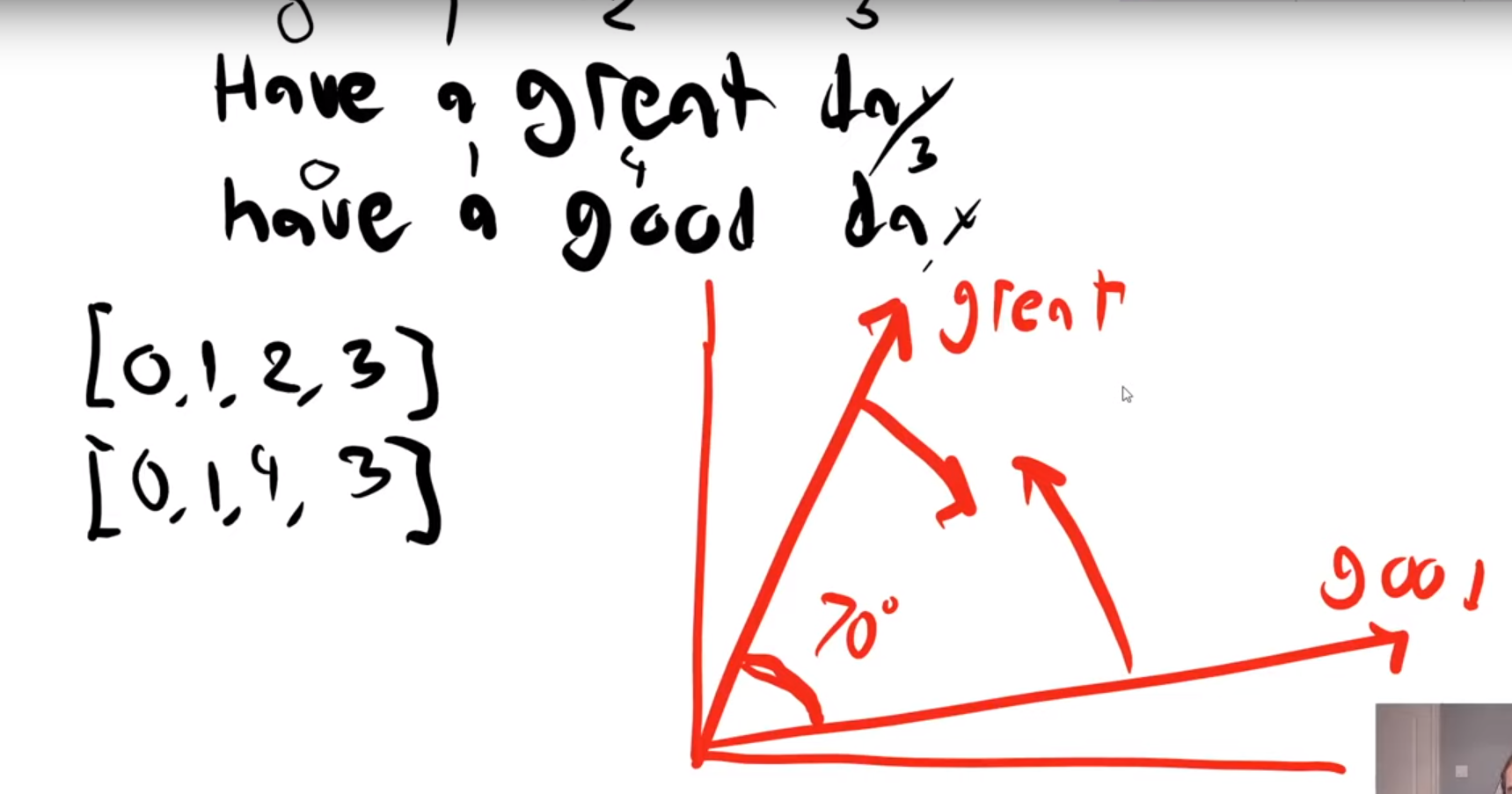
A Loss function will tell you how wrong the answer is.

**Understanding Example1.py:**

1. Embedding Layer:

The embedding layer will try to group words in a similar way (good,great). The embedding layer generates word vectors to pass to the future layers. A word vector can be in any kind of dimensional space.

For example:



In the above example, let us create a 16 dimensional vector. We plot a word vector for the word **have.**

When we initially, start we create a Embedded layer with 10,000 word vectors, each word vector representing a word. At first, the word vectors are drawn at random. The embedding layer uses these word vectors and uses that as the data for the next layer.

Initially, the words **great** and **good** are grouped far apart when in reality we want them to be close. We do this by measuring the angle between the vectors and adjusting them. They are adjusted on the basis of the context they’re used in and not the content.

*16 dimensions represents the number of coefficients in the vector.*

We want to scale these 16 dimensions down to a lower dimension using **Global Average Pooling**.

It’s important to save the model we train because the larger once will take days or some even years to train. It is even possible to save the model half way through the training process.