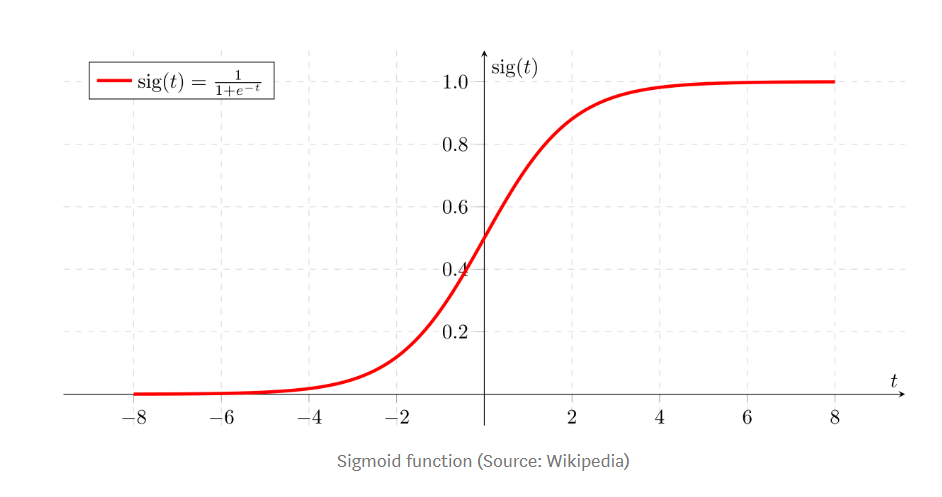
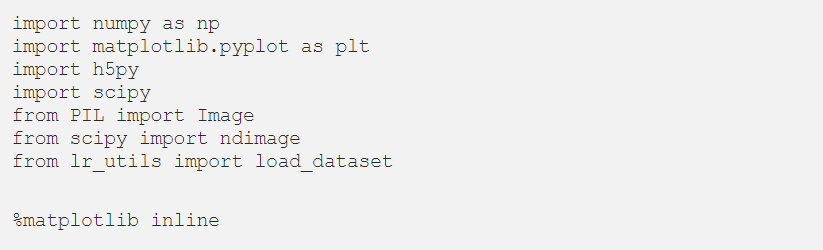
# **A step-by-step tutorial on coding Neural Network Logistic Regression model from scratch**

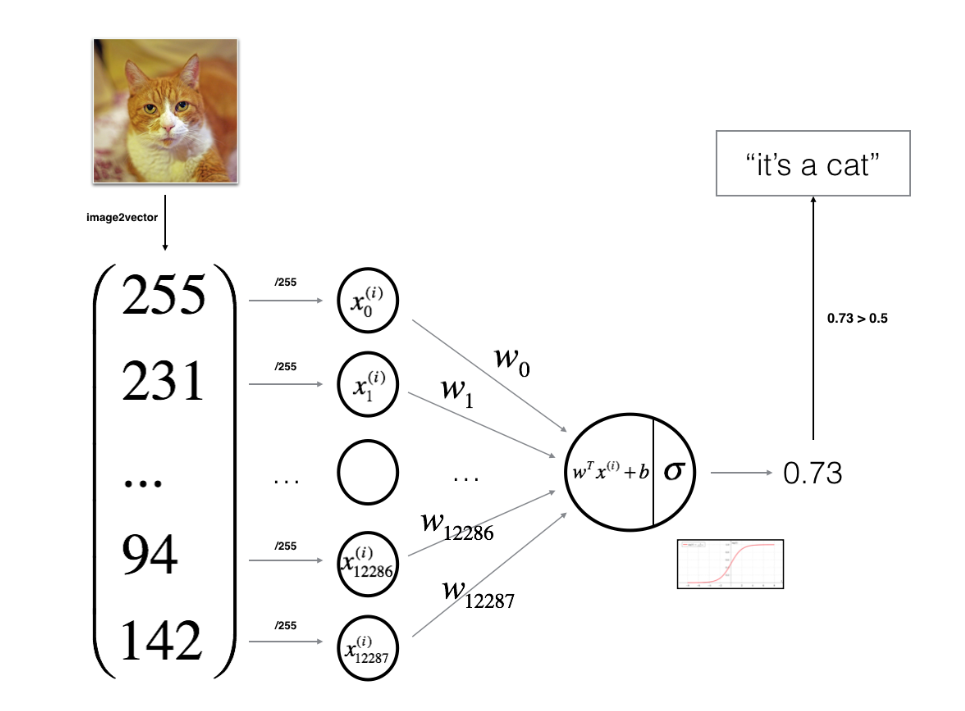
## **A brief introduction to Logistic Regression**

Logistic regression is a simple form of a neural network that classifies data categorically. For example, classifying emails as spam or non-spam is a classic use case of logistic regression. So how does it work? Simple. Logistic regression takes an input, passes it through a function called *sigmoid function* then returns an output of probability between 0 and 1. This sigmoid function is responsible for classifying the input.

Now, we know that there is a high chance of a wrong classification by the sigmoid function, which is bad for the algorithm. This “mistake” is also known as ***weight (w)*** or loss. The goal of a good logistic regression algorithm is to reduce loss or weight by improving the correctness of the output and this is achieved by a function called ***Gradient Descent***. A good way to evaluate the performance of the logistic regression algorithm is by achieving a minimal ***cost function***. Cost function quantifies the error between the predicted value and the expected values. Therefore, a logistic regression model must contain all these functions and we will code out these functions in python.

First, let’s import all the packages that we will be needing:

## **Logistic Regression with a Neural Network Mindset**

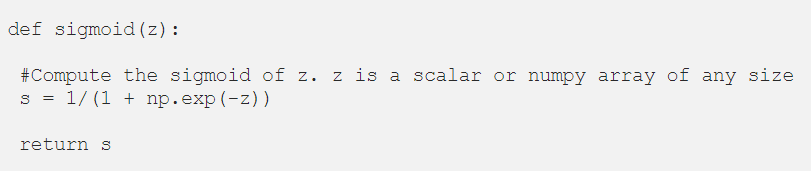
Logistic regression with a neural network mindset simply means that we will be doing a forward and backward propagation mode to code the algorithm as is usually the case with neural network algorithms. For logistic regression, the forward propagation is used to calculate the cost function and the output, *y*, while the backward propagation is used to calculate the gradient descent. This algorithm can be used to classify images as opposed to the ML form of logistic regression and that is what makes it stand out. The main steps for building the logistic regression neural network are:

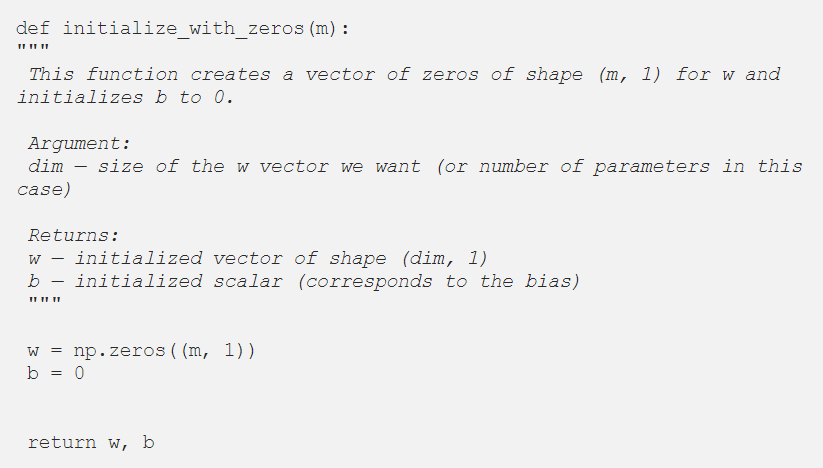
1. Define the model structure (such as number of input features)
2. Initialize the model’s parameters
3. Loop:

* Calculate current loss (forward propagation)
* Calculate current gradient (backward propagation)
* Update parameters (gradient descent)

Now, let’s code. First, we will begin by coding the sigmoid function by computing sigmoid( z) = 1/1+exp(-z), Where *z = wx+b (*Don’t worry about the formula if it does not make sense now, you will understand in the code below):

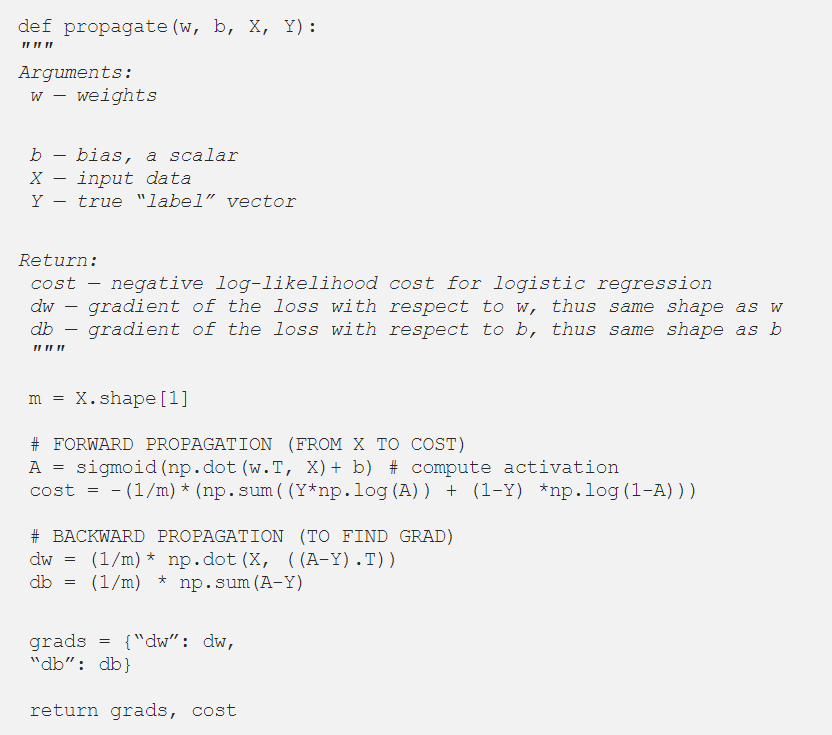
**Step 1: Implement the sigmoid function**

Now, we will continue by initializing the model parameters. The model parameters are the weights (*w*) and bias (*b*) with *x* as the input feature.

**Step 2: Initialize the model parameters**

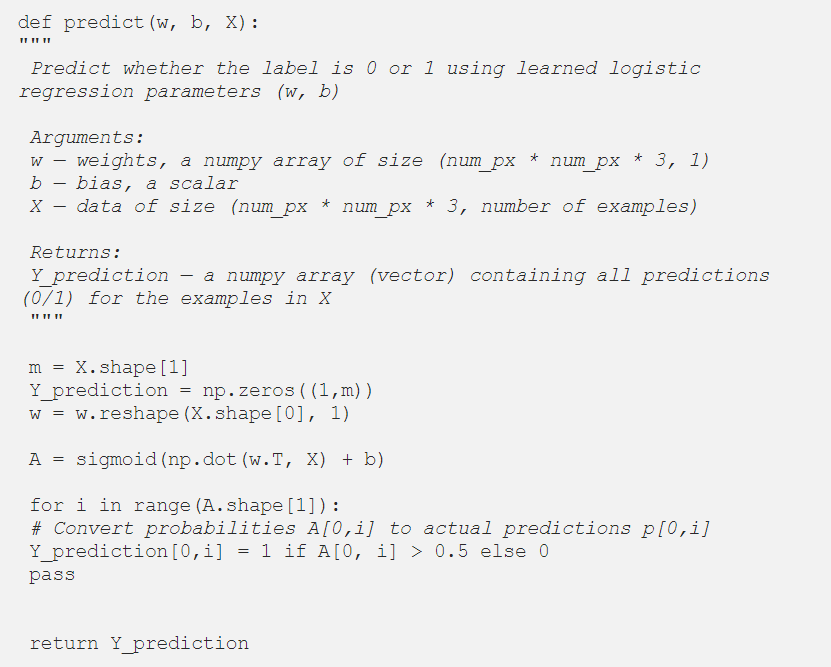
**Step 3: Implement forward and backward propagation for learning the parameters**

The next step is to implement the function called propagate() that learns the parameters *w, b,* and *y* from *x* by computing the cost function (forward) and its gradient (backward).



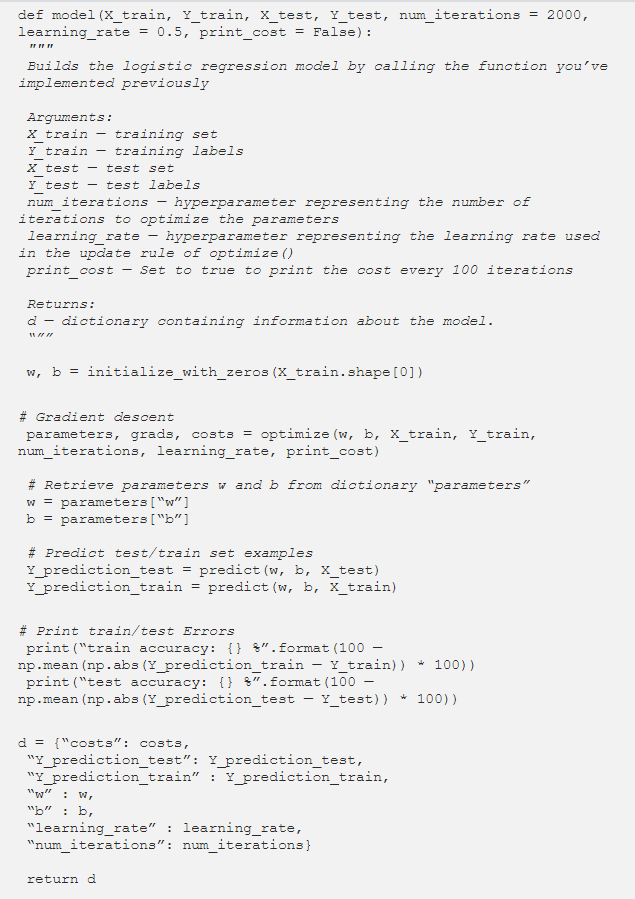
**Step 4: Update parameters with gradient descent**

Having established the output and the cost function, we will need to optimize our algorithm by updating our parameters with gradient descent. This will reduce cost function and minimize loss.



**Step 5: Putting it all together to form a model**

Now that we have our sigmoid function, cost function, and gradient descent, we will then combine everything into one single model and use this model to predict whether an image is a cat or non-cat



And we have successfully implemented a neural network logistic regression model from scratch with Python.