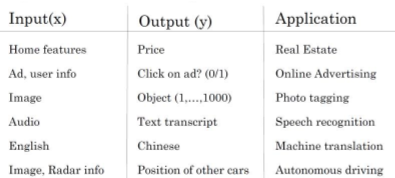
**Supervised learning for Neural Network**

Supervised learning is where we have a dataset & already know what our correct output should look like and we use an algorithm to learn the mapping function from the input to the output. i.e. Y = f(x).

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process of her students. The teacher already knows the correct answer, and only tells the students how to reach it by means of an algorithm. The student then iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results with a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results with a discrete output. In other words, we are trying to map input variables into discrete categories.

Here are some examples of supervised learning along with their applications:

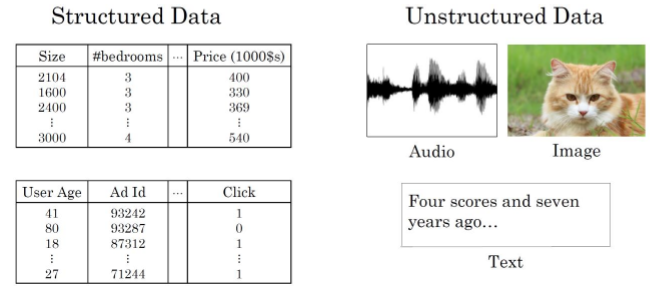


 There are different types of neural networks, for example

* Convolution Neural Network (CNN) used often for image application
* Recurrent Neural Network (RNN) used for one-dimensional sequence data such as translating English to Chinses or a temporal component such as text transcript.
* As for the autonomous driving, it is a Hybrid Neural Network architecture.

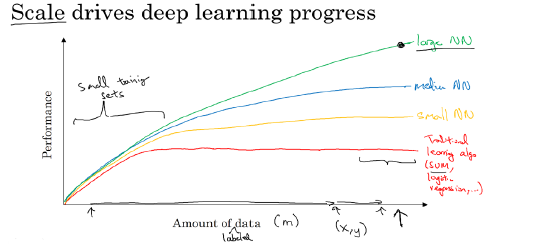
**Structured vs unstructured data**

Structured data refers to things that has a defined meaning such as price, age whereas unstructured data refers to thing like pixel, raw audio, text.



**Why is deep learning becoming so popular?**

 Deep learning is taking off due to a large amount of data available through the digitization of the society, faster computation and innovation in the development of neural network algorithm.

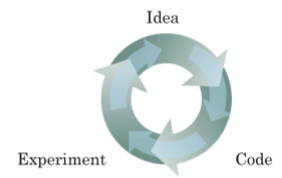


 Two things have to be considered to get to the high level of performance:

1. Being able to train a big enough neural network

2. Huge amount of labelled data

The process of training a neural network is iterative:



-We first have an idea that we would like to implement

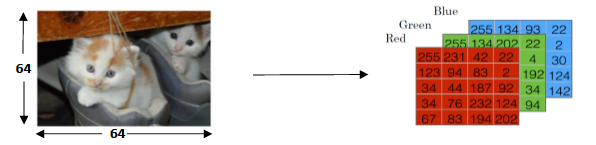
- We try it out using code.

- The experiment tells us how well our neural network does and based on how it performed, we revisit the idea and try to improve it.

This process can take a lot of time and affect our productivity. Thus, faster computing helps to iterate (repeat the process) & improve our new algorithm.

Our project revolves around a binary classification example where the result is a discrete value output (whether the image is a ‘cat’ or ‘non-cat’).

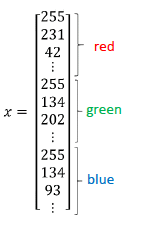
 The goal is to train a classifier that the input is an image represented by a feature vector, 𝑥, and predicts whether the corresponding label 𝑦 is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).



 An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n-dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, 𝑥, the pixel intensity values will be “unroll” or “reshape” for each color. The dimension of the input feature vector 𝑥 is 𝑛𝑥 = 64 𝑥 64 𝑥 3 = 12 288.



**Logistic Regression**

 Logistic regression is a learning algorithm used in a supervised learning problem when the output 𝑦 are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.

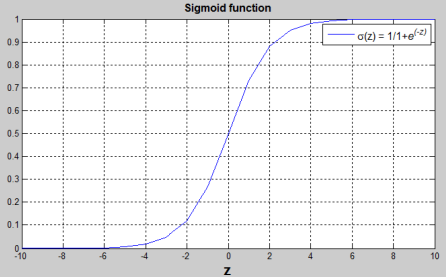
Example: Cat vs Non-Cat

Given an image represented by a feature vector 𝑥, the algorithm will evaluate the probability of a cat being in that image.

𝐺𝑖𝑣𝑒𝑛 ,𝑦̂=𝑃(𝑦=1|𝑥), where 0 ≤𝑦̂≤1

The parameters used in Logistic regression are:

* • The input features vector: 𝑥 ∈ℝ𝑛𝑥, where 𝑛𝑥 is the number of features
* • The training label: 𝑦∈0,1
* • The weights: 𝑤 ∈ℝ𝑛𝑥, where 𝑛𝑥 is the number of features
* • The threshold: 𝑏 ∈ℝ
* • The output: 𝑦̂=𝜎(𝑤𝑇𝑥+𝑏)
* • Sigmoid function: s = 𝜎(𝑤𝑇𝑥+𝑏) = 𝜎(𝑧)= 11+ 𝑒−𝑧

****

 (𝑤𝑇𝑥+𝑏) is a linear function (𝑎𝑥+𝑏), but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

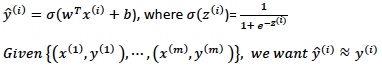
* • If 𝑧 is a large positive number, then 𝜎(𝑧) = 1
* • If 𝑧 is small or large negative number, then 𝜎(𝑧) = 0
* • If 𝑧=0, then 𝜎(𝑧) = 0.5

**The Cost Function**

 To train the parameters 𝑤 and 𝑏, we need to define a cost function.

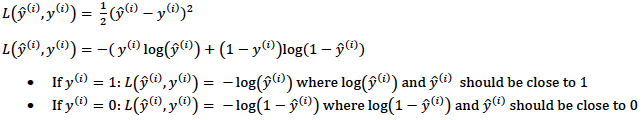
Recap:

../Screen%20Shot%202017-11-13%20at%2007.14.37.png



Loss (error) function:

The loss function measures the discrepancy between the prediction (𝑦̂(𝑖)) and the desired output (𝑦(𝑖)). In other words, the loss function computes the error for a single training example.

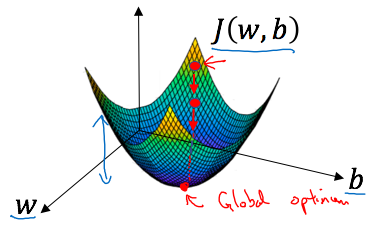


Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters 𝑤 𝑎𝑛𝑑 𝑏 that minimize the overall cost function.

../Screen%20Shot%202017-11-12%20at%2022.32.39.png

Now, in order to find the optimal value of (w,b) that minimises the cost function, we need to apply something called the Gradient Descent algorithm. Its workings are explained below:



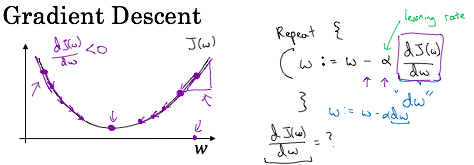
For gradient descent, we first initialise the parameter values of w, b to some initial value (denoted by the red dot for example).

Note: We usually initialise at zero.

Since the cost function is convex by definition, no matter where we initialise, we will end up at roughly the same optimal value.

What one step of gradient descent does is it starts at that initial point and takes a step in the steepest downhill direction. This means that we can even end up at the “global optimum” point (the point which gives optimal w,b) even after just one step.

In case you are a little overwhelmed by how this algorithm works, don’t worry. Let’s explain it in a 2D setting so it will make things easier to understand:



Suppose we are trying to minimise the cost function J(w); thus we want to find the optimal w.

Note the formula for w: **w= w – alpha\*dw** (where dw is the gradient of the cost function with respect to w).

* If we are initialising from the **right** of the minimum point , we get a **positive gradient** and thus the new value of w will be smaller . Thus, the system will converge to the global optimum from the right.
* If we are initialising from the **left** of the minimum point , we get a **negative gradient** and thus the new value of w will also be smaller . Thus, the system will converge to the global optimum from the left.

I’m pretty confident that you can now imagine what will happen in 3 Dimensions, when we add the parameter b.

In summary, the computations of a Neural Network are organised as a forward pass/ forward propagation step where we compute the output of the neural network along with the value of the cost function for one training example, followed by a backward propagation step where we compute the gradients/ derivatives. We do this m times (where m represents the number of training examples we are using) and finally, after the last iteration, we will end up with the optimal values of w, b which we use to compute the output of the images i.e. whether they are cats (1) or non-cats (0).

**Example**

Let us now build an image recognition algorithm using logistic regression to correctly recognise the pictures of cats. We will do this step by step and walk you through the code used:

We first build the general architecture of a learning algorithm, including:

- Initializing parameters

- Calculating the cost function and its gradient

- Using an optimization algorithm (gradient descent)

- Gather all three functions above into a main model function, in the right order.

**Step 1: Import Packages**

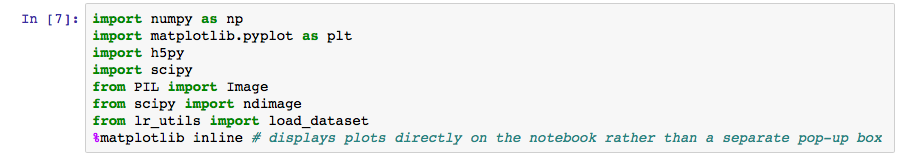
First, let's run the cell below to import all the packages that we will need during this assignment.

- [**numpy**](www.numpy.org) is the fundamental package for scientific computing with Python.

- [**h5py**](http://www.h5py.org) is a common package to interact with a dataset that is stored on an H5 file.

- [**matplotlib**](http://matplotlib.org) is a famous library to plot graphs in Python.

- [**PIL**](http://www.pythonware.com/products/pil/) and [**scipy**](https://www.scipy.org/) are used here to test your model with your own picture at the end.



**Step 2: Load Data**

We have a dataset ("data.h5") containing:

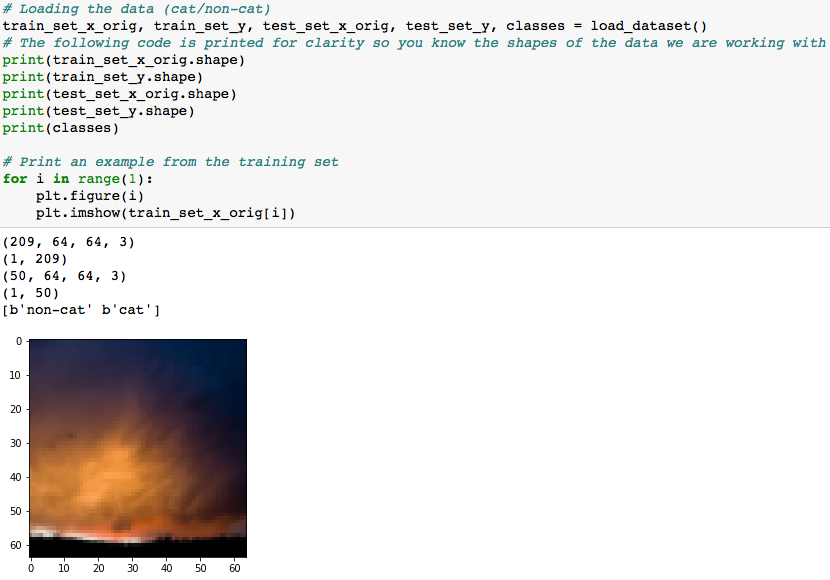
- a training set of m\_train = 209 images labeled as cat (y=1) or non-cat (y=0)

- a test set of m\_test = 50 images labeled as cat or non-cat

- each image is of shape (num\_px=64, num\_px=64, 3) where 3 is for the 3 channels (RGB). Thus, each image is square (height = 64) and (width = 64).

Note: We use 64 by 64 pixels just to demonstrate this example, but the dimensions of an image can take any size between 0-255(the maximum value of a pixel channel).

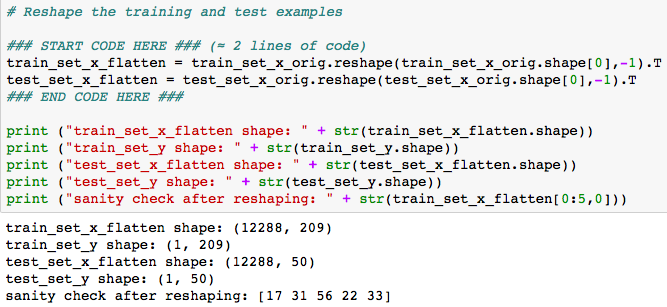
We first load the data by running the following code.



**Image(Data) Pre-Processing**

Notice that we added the extension “\_orig” to the train set x and test set labels. This is because we first need to “**pre-process**” the data. This just means that we need to convert it into the right shape so we can work with it i.e. the shape of the feature vector X:( 64\*64\*3, 209) & (64\*64\*3, 50) respectively. Notice that the train and test set y doesn’t need pre-processing as it already is in the correct shape : (1, 209) and (1,50) respectively for train and test sets.

The following code will reshape the datasets:



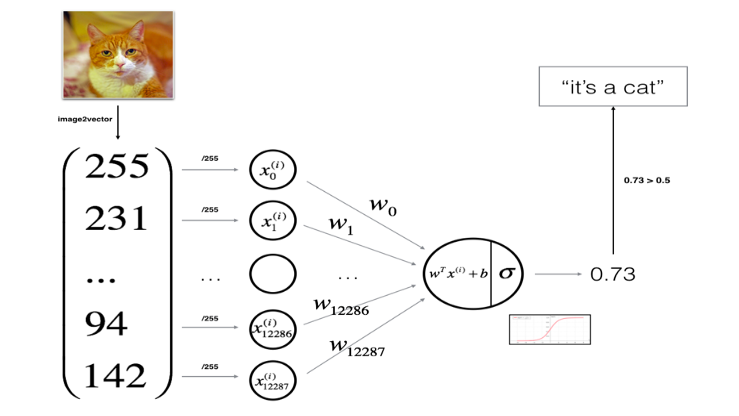
To represent color images, the red, green and blue channels (RGB) must be specified for each pixel, and so the pixel value is actually a vector of three numbers ranging from 0 to 255.

Note : 12288 = pixel\_dimension\_of\_image \* 3(red, green, blue colour channels) = 64\*64\*3.

One common preprocessing step in machine learning is to center and standardize your dataset, meaning that you substract the mean of the whole numpy array from each example, and then divide each example by the standard deviation of the whole numpy array. But for picture datasets, it is simpler and more convenient and works almost as well to just divide every row of the dataset by 255 (the maximum value of a pixel channel).

Lets Standardise our dataset:

../Screen%20Shot%202017-11-13%20at%2007.06.26.png

The General Architecture of the learning algorithm is as follows: 

Notice that we will be using the sigmoid function as the activation function for this example. Thus we will now need to define it. Recall its definition:

../Screen%20Shot%202017-11-13%20at%2007.16.07.png

Applying it in code in Python:

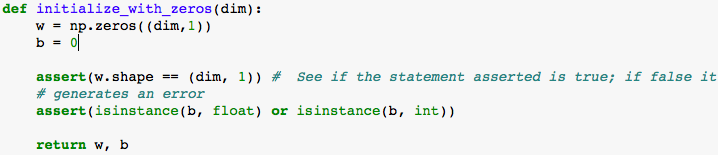
../Screen%20Shot%202017-11-13%20at%2007.34.20.png

For example:

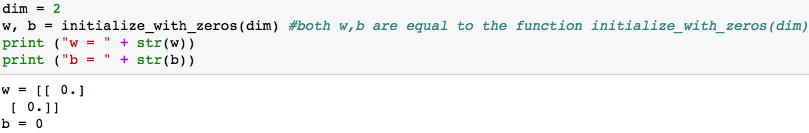
../Screen%20Shot%202017-11-13%20at%2007.35.59.png

**Step 3: Initialise Parameters(w,b)**

We now initialise w,b to be a vector of zeros using the np.zeros() function :

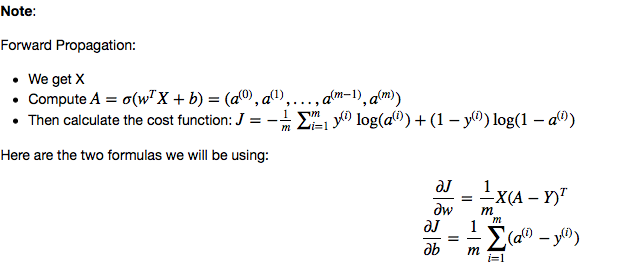


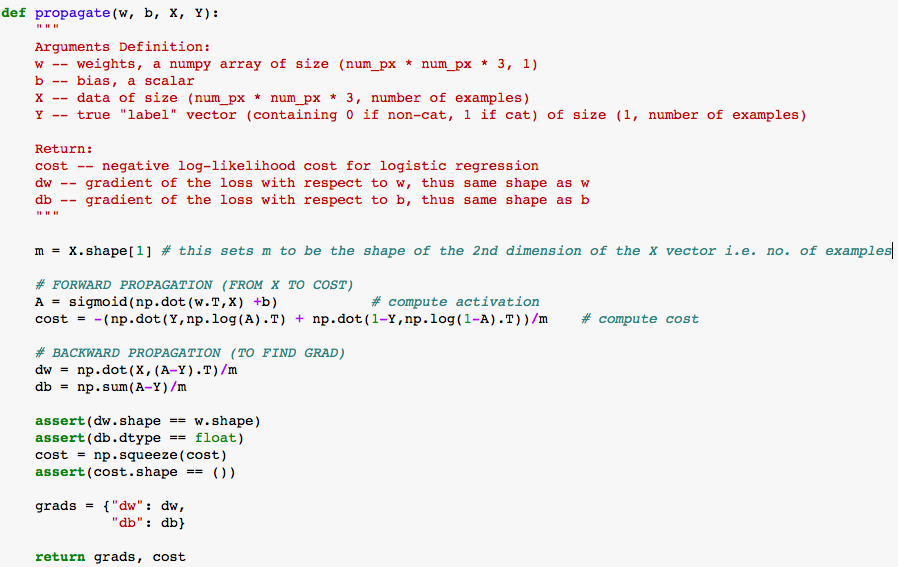
For example:



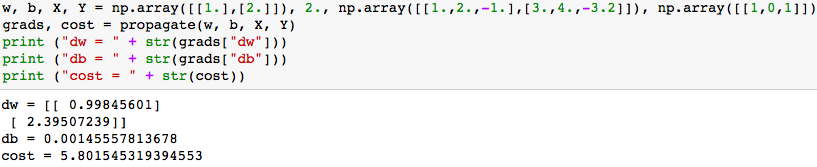
**Step 4: Forward & Backward Propagation**

Now that our parameters are initialized, we can do the "forward" and "backward" propagation steps for learning the parameters.



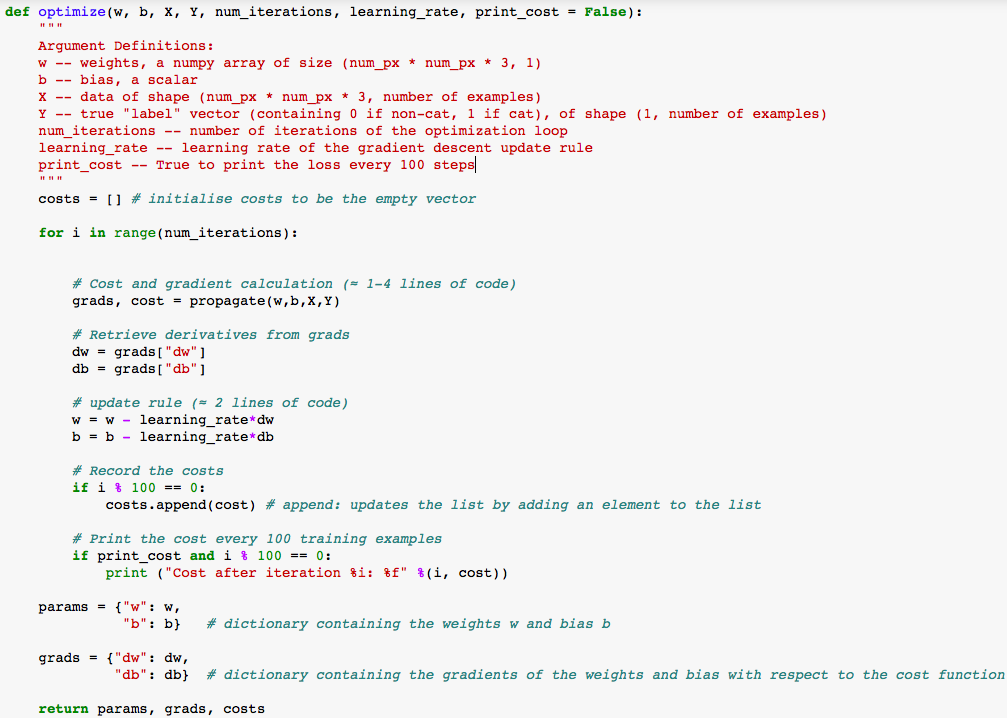


Example run:

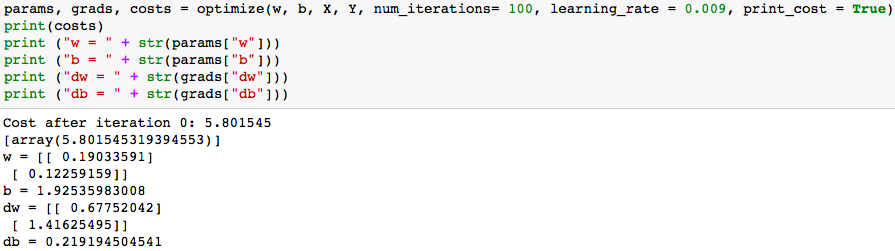


**Step 5: Optimize**

Now, we need to update w,b (to minimise the cost function J) using the gradient descent algorithm explained previously. For a parameter θ, the update rule is θ=θ−α dθ, where α is the learning rate.



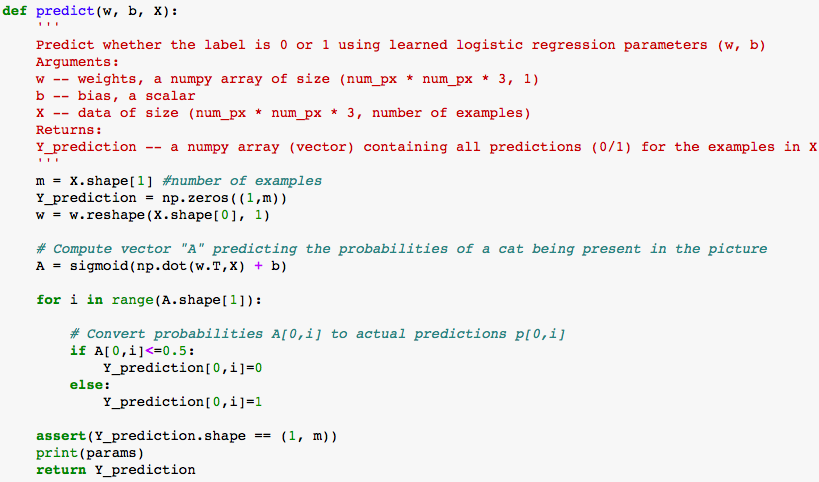
For example:



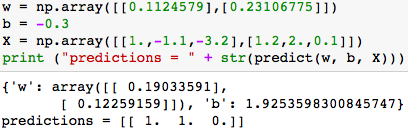
**Step 5: Predict Labels for Training Set X**

There are two steps to computing predictions:

1. Calculate ../Screen%20Shot%202017-11-13%20at%2008.12.27.png
2. Convert the entries of a into 0 (if activation <= 0.5) or 1 (if activation > 0.5), stores the predictions in a vector Y\_prediction.

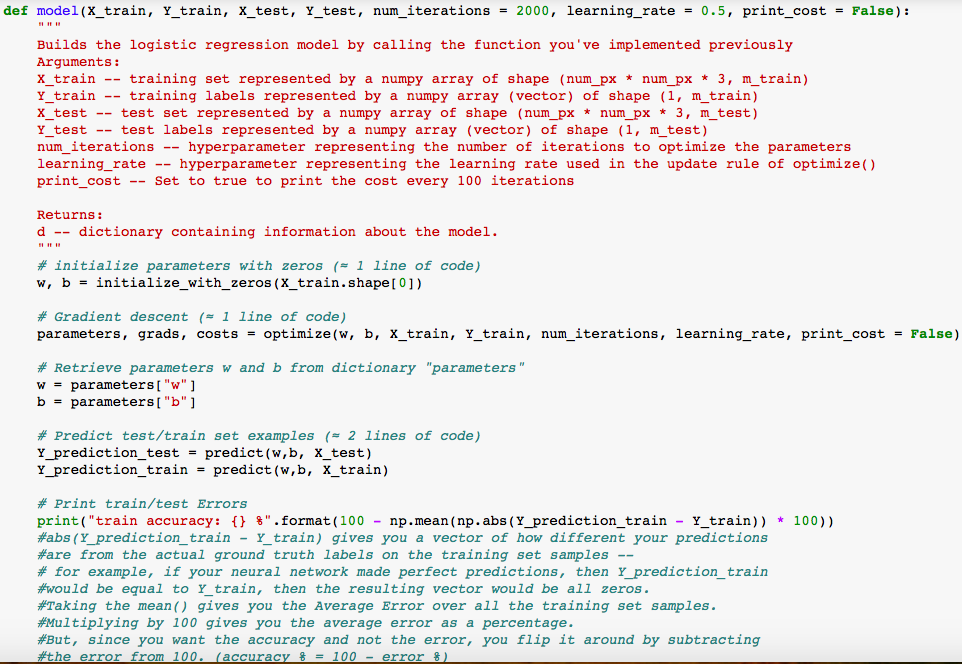


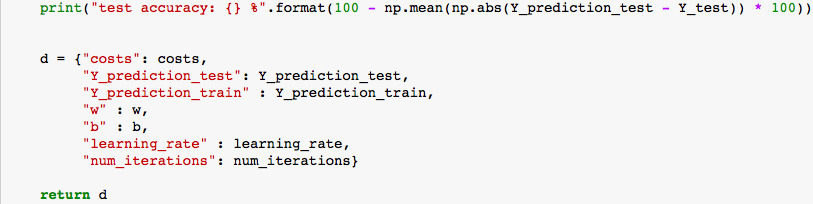
For example:



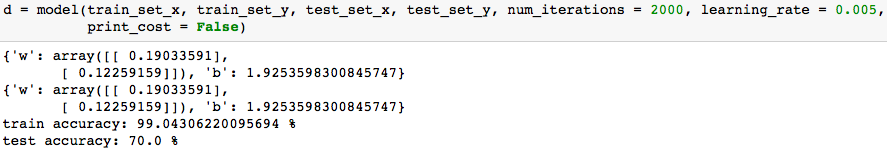
**Step 6: Merging all functions into a model**

We now merge all the functions we have previously built and combine it into a single function called “model”. We then use the model function to predict our test set labels:





We now run the following code to train our model:

****

**Comment**: Training accuracy is close to 100%. This is a good sanity check: our model is working and has high enough capacity to fit the training data. Test accuracy is around 70%. It is actually not bad for this simple model, given the small dataset we used and that logistic regression is a linear classifier (but we used a non-linear activation function).

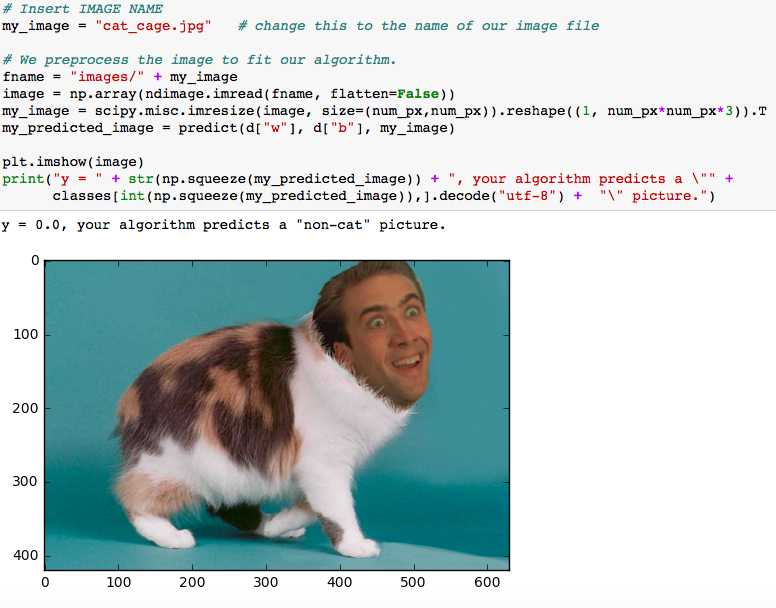
**Step 7: Test with our own image**

Finally, and perhaps the most exciting part of this simulation, is to test our model with any image of our choice!

Lets try out a few examples and see how well it does:



Awwww.. The algorithm seems to have no issue recognising a cat correctly. Lets try something a little more difficult:



Wow! Notice that even though Nicholas Cage weirdly seems to have the body of a cat, our algorithm wasn’t fooled by this. Pretty cool!

Now, as a final test, let’s actually confirm that our algorithm works pretty well by inserting a picture that doesn’t resemble a cat at all.



Yup...works ☺

**Note**: Even though these images were identified correctly, our model is only 70% accurate on the test set. This means that there are instances where it could go wrong. Some ways that it could be improved are by using a linear activation function, increasing the number of iterations, reducing overfitting (where our network adapted too much to the training data, to the point where it now performs poorly for data that is not already known) by regularisation etc. but we won’t go into all that here.