First we import all the basic modules and libraries. H5py module is used to read dataset from .h5 file used to store large amount of multidimensional array data

Then we load the data from the h5 file into a training set and testing set with x and y variables.

This here is an example of how an image of a dataset looks like.

The Original train and test ‘x’ data consist of 4-dimensional matrices with shape (pixel, pixel, 3, m) where 3 represents that image has RGB channels and ‘m’ is the number of training examples. The value of pixels ranges from 0 to 255. So we need to do data pre-processing on this data. First we flatten the data into a 2-dimensional shape (pixel\*pixel\*3,m) where one column is an image and m columns represent m image data. Then we normalize the pixel value by dividing by 255.

This is an activation function which returns sigmoid function value

Next we initialize the weights and the bias value as zero. Weights is a column vector with dimension of pixel\*pixel\*3 that is the flattened size of one image of test or train x data. Assert function is used to compare the shape or type of two matrices, vectors or scalar. Here it will check whether the weight vector created is of right shape or not.

Propagate function performs both forward and backward propagation. In this model we do not have a hidden layer and only an input and output layer. In forward propagation, we compute our predicted value A. I performed both sum of weighted inputs and activation here in one line only. Here A or X or Y all are matrices and I perform all these operations on complete matrices instead of individual images. This is a very useful property of python called vectorization. It helps in saving us from writing loops. Then I perform back propagation and calculate dw and db which from the chain rule we know is X\*(A-Y) where A-Y is nothing but error. We divide by m to calculate the average value.

Next is weight upgradation function. Here we calculate the earlier defined propagate function according to the number of iterations defined. Next we update weight and bias using w = w + alpha\*dw

Next is the predict function to finally predict the class after weight adjustment is over. We create a column vector Y for predicting class. So we predict in the same way we perform forward propagation but with finalized weights. As logistic regression tells us the probabilistic values we convert into actual classes by assigning 0 if A value is less than 0.5 and 1 otherwise.

So now here is a model function where we combine all the functions defined earlier and perform stepwise logistic regression. First we initialize the weights. Then we call the optimize function which in itself iteratively calls propagate function . Then using the final weight we call the predict function and finally we calculate the final test and train error.

Here we call the model function for the above data set where I set iteration = 2000 and learning rate alpha 0.005. We can tune hyperparameters here as we wish. We can see for above dataset and these values of hyperparameters the train accuracy is 99% and test accuracy 70%.