Basic Instructions

1. Enter your Name and UID in the provided space.
2. Do the assignment in the notebook itself
3. you are free to use Google Colab

Name: **Arpit Aggarwal**
UID: **116747189**

In the first part, you will implement all the functions required to build a two layer neural network. In the next part, you will use these functions for image and text classification. Provide your code at the appropriate placeholders.

1. Packages

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
```

2. Layer Initialization

Exercise: Create and initialize the parameters of the 2-layer neural network. Use random initialization for the weight matrices and zero initialization for the biases.

In [2]:
```
def initialize_parameters(n_x, n_h, n_y):
    """
    Argument:
    n_x -- size of the input layer
    n_h -- size of the hidden layer
    n_y -- size of the output layer

    Returns:
    parameters -- python dictionary containing your parameters:
                    W1 -- weight matrix of shape (n_h, n_x)
                    b1 -- bias vector of shape (n_h, 1)
                    W2 -- weight matrix of shape (n_y, n_h)
                    b2 -- bias vector of shape (n_y, 1)
    """

    np.random.seed(1)

    ### START CODE HERE ### (≈ 4 lines of code)

    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros(shape=(n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros(shape=(n_y, 1))

    ### END CODE HERE ###

    assert(W1.shape == (n_h, n_x))
    assert(b1.shape == (n_h, 1))
    assert(W2.shape == (n_y, n_h))
    assert(b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

In [3]:
```
parameters = initialize_parameters(3,2,1)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
W1 = [[ 0.01624345 -0.00611756 -0.00528172]
 [-0.01072969  0.00865408 -0.02301539]]
b1 = [[0.]
 [0.]]
W2 = [[ 0.01744812 -0.00761207]]
b2 = [[0.]]
```

Expected output:

W1	[[0.01624345 -0.00611756 -0.00528172] [-0.01072969 0.00865408 -0.02301539]]
b1	[[0.] [0.]]
W2	[[0.01744812 -0.00761207]]
b2	[[0.]]

3. Forward Propagation

Now that you have initialized your parameters, you will do the forward propagation module. You will start by implementing some basic functions that you will use later when implementing the model. You will complete three functions in this order:

- LINEAR
- LINEAR -> ACTIVATION where ACTIVATION will be either ReLU or Sigmoid.

The linear module computes the following equation:

$$Z = WA + b \tag{4}$$

3.1 Exercise: Build the linear part of forward propagation.

In [4]:
```
def linear_forward(A, W, b):
    """
    Implement the linear part of a layer's forward propagation.

    Arguments:
    A -- activations from previous layer (or input data): (size of pr
evious layer, number of examples)
    W -- weights matrix: numpy array of shape (size of current layer,
size of previous layer)
    b -- bias vector, numpy array of shape (size of the current laye
r, 1)

    Returns:
    Z -- the input of the activation function, also called pre-activa
tion parameter
    cache -- a python dictionary containing "A", "W" and "b" ; stored
for computing the backward pass efficiently
    """

    ### START CODE HERE ### (≈ 1 line of code)

    Z = np.dot(W, A) + b

    ### END CODE HERE ###

    assert(Z.shape == (W.shape[0], A.shape[1]))
    cache = (A, W, b)

    return Z, cache
```

In [5]:
```
np.random.seed(1)

A = np.random.randn(3,2)
W = np.random.randn(1,3)
b = np.random.randn(1,1)

Z, linear_cache = linear_forward(A, W, b)
print("Z = " + str(Z))
```

```
Z = [[ 3.26295337 -1.23429987]]
```

Expected output:

Z	[[3.26295337 -1.23429987]]

3.2 - Linear-Activation Forward

In this notebook, you will use two activation functions:

- **Sigmoid**: $\sigma(Z) = \sigma(WA + b) = \frac{1}{1+e^{-(WA+b)}}$. Write the code for the `sigmoid` function. This function returns **two** items: the activation value " `a` " and a " `cache` " that contains " `Z` " (it's what we will feed in to the corresponding backward function). To use it you could just call:

```
A, activation_cache = sigmoid(Z)
```

- **ReLU**: The mathematical formula for ReLu is $A = RELU(Z) = max(0, Z)$. Write the code for the `relu` function. This function returns **two** items: the activation value " `A` " and a " `cache` " that contains " `Z` " (it's what we will feed in to the corresponding backward function). To use it you could just call: ``` python A, activation_cache = relu(Z)

Exercise:

- Implement the activation functions
- Build the linear activation part of forward propagation. Mathematical relation is: $A = g(Z) = g(WA_{prev} + b)$

In [6]:
```
def sigmoid(Z):
    """
    Implements the sigmoid activation in numpy

    Arguments:
    Z -- numpy array of any shape

    Returns:
    A -- output of sigmoid(z), same shape as Z
    cache -- returns Z, useful during backpropagation
    """
    ### START CODE HERE ### (≈ 2 line of code)

    A = 1.0 / (1.0 + np.exp(-Z))
    cache = Z

    ### END CODE HERE ###

    return A, cache

def relu(Z):
    """
    Implement the RELU function.

    Arguments:
    Z -- Output of the linear layer, of any shape

    Returns:
    A -- Post-activation parameter, of the same shape as Z
    cache --  returns Z, useful during backpropagation
    """

    ### START CODE HERE ### (≈ 2 line of code)

    A = np.maximum(0, Z)
    cache = Z

    ### END CODE HERE ###

    assert(A.shape == Z.shape)
    return A, cache
```

In [7]:
```
def linear_activation_forward(A_prev, W, b, activation):
    """
    Implement the forward propagation for the LINEAR->ACTIVATION laye
r

    Arguments:
    A_prev -- activations from previous layer (or input data): (size
 of previous layer, number of examples)
    W -- weights matrix: numpy array of shape (size of current layer,
size of previous layer)
    b -- bias vector, numpy array of shape (size of the current laye
r, 1)
    activation -- the activation to be used in this layer, stored as
 a text string: "sigmoid" or "relu"

    Returns:
    A -- the output of the activation function, also called the post-
activation value
    cache -- a python dictionary containing "linear_cache" and "activ
ation_cache";
             stored for computing the backward pass efficiently
    """

    if activation == "sigmoid":
        # Inputs: "A_prev, W, b". Outputs: "A, activation_cache".
        ### START CODE HERE ### (≈ 2 lines of code)

        Z, linear_cache = linear_forward(A_prev, W, b)
        A, activation_cache = sigmoid(Z)

        ### END CODE HERE ###

    elif activation == "relu":
        # Inputs: "A_prev, W, b". Outputs: "A, activation_cache".
        ### START CODE HERE ### (≈ 2 lines of code)

        Z, linear_cache = linear_forward(A_prev, W, b)
        A, activation_cache = relu(Z)

        ### END CODE HERE ###

    assert (A.shape == (W.shape[0], A_prev.shape[1]))
    cache = (linear_cache, activation_cache)

    return A, cache
```

In [8]:
```
np.random.seed(2)
A_prev = np.random.randn(3,2)
W = np.random.randn(1,3)
b = np.random.randn(1,1)

A, linear_activation_cache = linear_activation_forward(A_prev, W, b,
activation = "sigmoid")
print("With sigmoid: A = " + str(A))

A, linear_activation_cache = linear_activation_forward(A_prev, W, b,
activation = "relu")
print("With ReLU: A = " + str(A))
```

```
With sigmoid: A = [[0.96890023 0.11013289]]
With ReLU: A = [[3.43896131 0.        ]]
```

Expected output:

**With sigmoid: A **	[[0.96890023 0.11013289]]
**With ReLU: A **	[[3.43896131 0.]]

4 - Loss function

Now you will implement forward and backward propagation. You need to compute the loss, because you want to check if your model is actually learning.

Exercise: Compute the cross-entropy loss J, using the following formula:

$$-\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log\left(a^{(i)}\right) + (1-y^{(i)}) \log\left(1-a^{(i)}\right)) \tag{7}$$

In [9]:
```
# GRADED FUNCTION: compute_loss

def compute_loss(A, Y):
    """
    Implement the loss function defined by equation (7).

    Arguments:
    A -- probability vector corresponding to your label predictions,
 shape (1, number of examples)
    Y -- true "label" vector (for example: containing 0 if non-cat, 1
if cat), shape (1, number of examples)

    Returns:
    loss -- cross-entropy loss
    """

    m = Y.shape[1]

    # Compute loss from aL and y.
    ### START CODE HERE ### (≈ 1 lines of code)

    loss = (-1.0 / m) * np.sum((Y * np.log(A)) + ((1.0 - Y) * np.log(
1.0 - A)))

    ### END CODE HERE ###

    loss = np.squeeze(loss)      # To make sure your loss's shape is
 what we expect (e.g. this turns [[17]] into 17).
    assert(loss.shape == ())

    return loss
```

In [10]:
```
Y = np.asarray([[1, 1, 1]])
A = np.array([[.8,.9,0.4]])

print("loss = " + str(compute_loss(A, Y)))
```

```
loss = 0.41493159961539694
```

Expected Output:

loss	0.41493159961539694

5 - Backward propagation module

Just like with forward propagation, you will implement helper functions for backpropagation. Remember that back propagation is used to calculate the gradient of the loss function with respect to the parameters.

Now, similar to forward propagation, you are going to build the backward propagation in two steps:

- LINEAR backward
- LINEAR -> ACTIVATION backward where ACTIVATION computes the derivative of either the ReLU or sigmoid activation

5.1 - Linear backward

In [11]:
```
# GRADED FUNCTION: linear_backward

def linear_backward(dZ, cache):
    """
    Implement the linear portion of backward propagation for a single layer (layer l)

    Arguments:
    dZ -- Gradient of the loss with respect to the linear output (of current layer l)
    cache -- tuple of values (A_prev, W, b) coming from the forward propagation in the current layer

    Returns:
    dA_prev -- Gradient of the loss with respect to the activation (of the previous layer l-1), same shape as A_prev
    dW -- Gradient of the loss with respect to W (current layer l), same shape as W
    db -- Gradient of the loss with respect to b (current layer l), same shape as b
    """
    A_prev, W, b = cache
    m = A_prev.shape[1]

    ### START CODE HERE ### (≈ 3 lines of code)

    dA_prev = np.dot(W.T, dZ)
    dW = np.dot(dZ, A_prev.T)
    db = np.array([np.sum(dZ, axis = 1)]).T

    ### END CODE HERE ###

    assert (dA_prev.shape == A_prev.shape)
    assert (dW.shape == W.shape)
    assert (db.shape == b.shape)

    return dA_prev, dW, db
```

In [12]:
```
np.random.seed(1)
dZ = np.random.randn(1,2)
A = np.random.randn(3,2)
W = np.random.randn(1,3)
b = np.random.randn(1,1)
linear_cache = (A, W, b)

dA_prev, dW, db = linear_backward(dZ, linear_cache)
print ("dA_prev = "+ str(dA_prev))
print ("dW = " + str(dW))
print ("db = " + str(db))
```

```
dA_prev = [[ 0.51822968 -0.19517421]
 [-0.40506361  0.15255393]
 [ 2.37496825 -0.89445391]]
dW = [[-0.2015379   2.81370193  3.2998501 ]]
db = [[1.01258895]]
```

Expected Output:

dA_prev	[[0.51822968 -0.19517421] [-0.40506361 0.15255393] [2.37496825 -0.89445391]]
dW	[[-0.2015379 2.81370193 3.2998501]]
db	[[1.01258895]]

5.2 - Linear Activation backward

Next, you will create a function that merges the two helper functions: **`linear_backward`** and the backward step for the activation **`linear_activation_backward`**.

Before implementing `linear_activation_backward`, you need to implement two backward functions for each activations:

- **`sigmoid_backward`**: Implements the backward propagation for SIGMOID unit. You can call it as follows:

```
dZ = sigmoid_backward(dA, activation_cache)
```

- **`relu_backward`**: Implements the backward propagation for RELU unit. You can call it as follows:

```
dZ = relu_backward(dA, activation_cache)
```

If $g(.)$ is the activation function, `sigmoid_backward` and `relu_backward` compute

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]}) \tag{11}$$

.

Exercise:

- Implement the backward functions for the relu and sigmoid activation layer.
- Implement the backpropagation for the *LINEAR->ACTIVATION* layer.

In [13]:
```
def relu_backward(dA, cache):
    """
    Implement the backward propagation for a single RELU unit.

    Arguments:
    dA -- post-activation gradient, of any shape
    cache -- 'Z' where we store for computing backward propagation ef
ficiently

    Returns:
    dZ -- Gradient of the loss with respect to Z
    """

    Z = cache
    dZ = np.array(dA, copy=True) # just converting dz to a correct ob
ject.

    ### START CODE HERE ### (≈ 1 line of code)

    dZ = dA * np.where(Z <= 0, 0, 1)

    ### END CODE HERE ###

    assert (dZ.shape == Z.shape)

    return dZ

def sigmoid_backward(dA, cache):
    """
    Implement the backward propagation for a single SIGMOID unit.

    Arguments:
    dA -- post-activation gradient, of any shape
    cache -- 'Z' where we store for computing backward propagation ef
ficiently

    Returns:
    dZ -- Gradient of the loss with respect to Z
    """

    Z = cache

    ### START CODE HERE ### (≈ 2 line of code)

    sigmoid_derivative = sigmoid(Z)[0] * (1.0 - sigmoid(Z)[0])
    dZ = dA * sigmoid_derivative

    ### END CODE HERE ###

    assert (dZ.shape == Z.shape)

    return dZ
```

In [14]:
```
# GRADED FUNCTION: linear_activation_backward

def linear_activation_backward(dA, cache, activation):
    """
    Implement the backward propagation for the LINEAR->ACTIVATION lay
er.

    Arguments:
    dA -- post-activation gradient for current layer l
    cache -- tuple of values (linear_cache, activation_cache) we stor
e for computing backward propagation efficiently
    activation -- the activation to be used in this layer, stored as
 a text string: "sigmoid" or "relu"

    Returns:
    dA_prev -- Gradient of the loss with respect to the activation (o
f the previous layer l-1), same shape as A_prev
    dW -- Gradient of the loss with respect to W (current layer l), s
ame shape as W
    db -- Gradient of the loss with respect to b (current layer l), s
ame shape as b
    """
    linear_cache, activation_cache = cache

    if activation == "relu":
        ### START CODE HERE ### (≈ 2 lines of code)

        dZ = relu_backward(dA, activation_cache)
        dA_prev, dW, db = linear_backward(dZ, linear_cache)

        ### END CODE HERE ###

    elif activation == "sigmoid":
        ### START CODE HERE ### (≈ 2 lines of code)

        dZ = sigmoid_backward(dA, activation_cache)
        dA_prev, dW, db = linear_backward(dZ, linear_cache)

        ### END CODE HERE ###

    return dA_prev, dW, db
```

In [15]:
```
np.random.seed(2)
dA = np.random.randn(1,2)
A = np.random.randn(3,2)
W = np.random.randn(1,3)
b = np.random.randn(1,1)
Z = np.random.randn(1,2)
linear_cache = (A, W, b)
activation_cache = Z
linear_activation_cache = (linear_cache, activation_cache)

dA_prev, dW, db = linear_activation_backward(dA, linear_activation_ca
che, activation = "sigmoid")
print ("sigmoid:")
print ("dA_prev = "+ str(dA_prev))
print ("dW = " + str(dW))
print ("db = " + str(db) + "\n")

dA_prev, dW, db = linear_activation_backward(dA, linear_activation_ca
che, activation = "relu")
print ("relu:")
print ("dA_prev = "+ str(dA_prev))
print ("dW = " + str(dW))
print ("db = " + str(db))
```

```
sigmoid:
dA_prev = [[ 0.11017994  0.01105339]
 [ 0.09466817  0.00949723]
 [-0.05743092 -0.00576154]]
dW = [[ 0.20533573  0.19557101 -0.03936168]]
db = [[-0.11459244]]

relu:
dA_prev = [[ 0.44090989  0.        ]
 [ 0.37883606  0.        ]
 [-0.2298228   0.        ]]
dW = [[ 0.89027649  0.74742835 -0.20957978]]
db = [[-0.41675785]]
```

Expected output with sigmoid:

dA_prev	[[0.11017994 0.01105339] [0.09466817 0.00949723] [-0.05743092 -0.00576154]]
dW	[[0.20533573 0.19557101 -0.03936168]]
db	[[-0.11459244]]

Expected output with relu:

dA_prev	[[0.44090989 0.] [0.37883606 0.] [-0.2298228 0.]]
dW	[[0.89027649 0.74742835 -0.20957978]]
db	[[-0.41675785]]

6 - Update Parameters

In this section you will update the parameters of the model, using gradient descent:

$$W^{[1]} = W^{[1]} - \alpha \; dW^{[1]} \tag{16}$$
$$b^{[1]} = b^{[1]} - \alpha \; db^{[1]} \tag{17}$$
$$W^{[2]} = W^{[2]} - \alpha \; dW^{[2} \tag{16}$$
$$b^{[2]} = b^{[2]} - \alpha \; db^{[2]} \tag{17}$$

where α is the learning rate. After computing the updated parameters, store them in the parameters dictionary.

Exercise: Implement `update_parameters()` to update your parameters using gradient descent.

Instructions: Update parameters using gradient descent.

In [16]:
```
# GRADED FUNCTION: update_parameters

def update_parameters(parameters, grads, learning_rate):
    """
    Update parameters using gradient descent

    Arguments:
    parameters -- python dictionary containing your parameters
    grads -- python dictionary containing your gradients, output of L
_model_backward

    Returns:
    parameters -- python dictionary containing your updated parameter
s
                  parameters["W" + str(l)] = ...
                  parameters["b" + str(l)] = ...
    """
    # Update rule for each parameter. Use a for loop.
    ### START CODE HERE ### (≈ 4 lines of code)

    for key in parameters:
        parameters[key] = parameters[key] - (learning_rate * grads[
"d" + str(key)])

    ### END CODE HERE ###
    return parameters
```

In [17]:
```
np.random.seed(2)
W1 = np.random.randn(3,4)
b1 = np.random.randn(3,1)
W2 = np.random.randn(1,3)
b2 = np.random.randn(1,1)
parameters = {"W1": W1,
              "b1": b1,
              "W2": W2,
              "b2": b2}
np.random.seed(3)
dW1 = np.random.randn(3,4)
db1 = np.random.randn(3,1)
dW2 = np.random.randn(1,3)
db2 = np.random.randn(1,1)
grads = {"dW1": dW1,
         "db1": db1,
         "dW2": dW2,
         "db2": db2}
parameters = update_parameters(parameters, grads, 0.1)

print ("W1 = "+ str(parameters["W1"]))
print ("b1 = "+ str(parameters["b1"]))
print ("W2 = "+ str(parameters["W2"]))
print ("b2 = "+ str(parameters["b2"]))
```

```
W1 = [[-0.59562069 -0.09991781 -2.14584584  1.82662008]
 [-1.76569676 -0.80627147  0.51115557 -1.18258802]
 [-1.0535704  -0.86128581  0.68284052  2.20374577]]
b1 = [[-0.04659241]
 [-1.28888275]
 [ 0.53405496]]
W2 = [[-0.55569196  0.0354055   1.32964895]]
b2 = [[-0.84610769]]
```

Expected Output:

W1	[[-0.59562069 -0.09991781 -2.14584584 1.82662008] [-1.76569676 -0.80627147 0.51115557 -1.18258802] [-1.0535704 -0.86128581 0.68284052 2.20374577]]
b1	[[-0.04659241] [-1.28888275] [0.53405496]]
W2	[[-0.55569196 0.0354055 1.32964895]]
b2	[[-0.84610769]]

7 - Conclusion

Congrats on implementing all the functions required for building a deep neural network!

We know it was a long assignment but going forward it will only get better. The next part of the assignment is easier.

Part 2:

In the next part you will put all these together to build a two-layer neural networks for image classification.

In [18]:
```
%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plo
ts
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2

np.random.seed(1)
```

Dataset

Problem Statement: You are given a dataset ("data/train_catvnoncat.h5", "data/test_catvnoncat.h5") containing:

```
- a training set of m_train images labelled as cat (1) or non-cat (0)
- a test set of m_test images labelled as cat and non-cat
- each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels
 (RGB).
```

Let's get more familiar with the dataset. Load the data by completing the function and run the cell below.

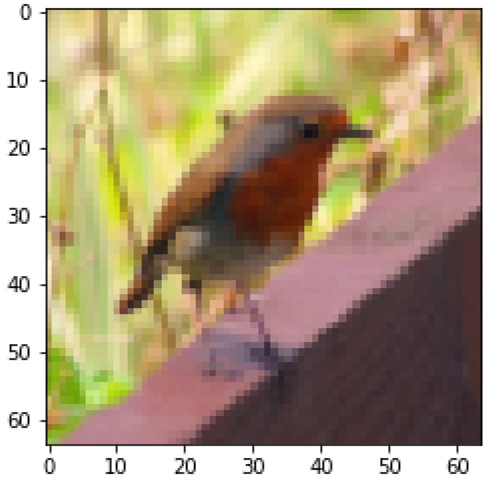
In [19]:
```
def load_data(train_file, test_file):
    # Load the training data
    train_dataset = h5py.File(train_file, 'r')

    # Separate features(x) and labels(y) for training set
    train_set_x_orig = np.array(train_dataset['train_set_x'])
    train_set_y_orig = np.array(train_dataset['train_set_y'])

    # Load the test data
    test_dataset = h5py.File(test_file, 'r')

    # Separate features(x) and labels(y) for training set
    test_set_x_orig = np.array(test_dataset['test_set_x'])
    test_set_y_orig = np.array(test_dataset['test_set_y'])

    classes = np.array(test_dataset["list_classes"][:]) # the list of
classes

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.
shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.sha
pe[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_
set_y_orig, classes
```

In [20]:
```
train_file="data/train_catvnoncat.h5"
test_file="data/test_catvnoncat.h5"
train_x_orig, train_y, test_x_orig, test_y, classes = load_data(train
_file, test_file)
```

The following code will show you an image in the dataset. Feel free to change the index and re-run the cell multiple times to see other images.

In [21]:
```
# Example of a picture
index = 10
plt.imshow(train_x_orig[index])
print ("y = " + str(train_y[0,index]) + ". It's a " + classes[train_y
[0,index]].decode("utf-8") +  " picture.")
```

```
y = 0. It's a non-cat picture.
```



In [22]:
```
# Explore your dataset
m_train = train_x_orig.shape[0]
num_px = train_x_orig.shape[1]
m_test = test_x_orig.shape[0]

print ("Number of training examples: " + str(m_train))
print ("Number of testing examples: " + str(m_test))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px)
+ ", 3)")
print ("train_x_orig shape: " + str(train_x_orig.shape))
print ("train_y shape: " + str(train_y.shape))
print ("test_x_orig shape: " + str(test_x_orig.shape))
print ("test_y shape: " + str(test_y.shape))
```

```
Number of training examples: 209
Number of testing examples: 50
Each image is of size: (64, 64, 3)
train_x_orig shape: (209, 64, 64, 3)
train_y shape: (1, 209)
test_x_orig shape: (50, 64, 64, 3)
test_y shape: (1, 50)
```

As usual, you reshape and standardize the images before feeding them to the network.

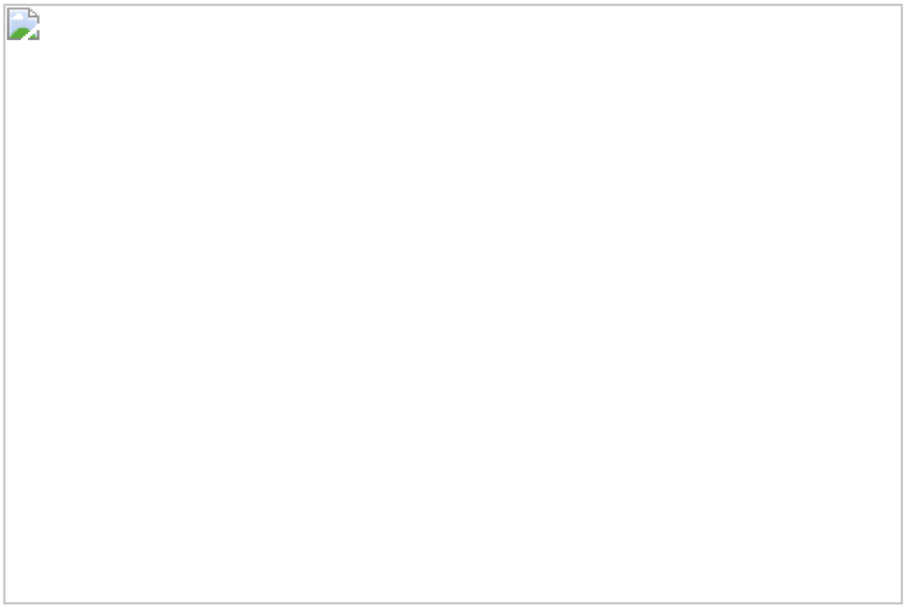


Figure 1: Image to vector conversion.

In [23]:
```
# Reshape the training and test examples
train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T
# The "-1" makes reshape flatten the remaining dimensions
test_x_flatten = test_x_orig.reshape(test_x_orig.shape[0], -1).T

# Standardize data to have feature values between 0 and 1.
train_x = train_x_flatten/255.
test_x = test_x_flatten/255.

print ("train_x's shape: " + str(train_x.shape))
print ("test_x's shape: " + str(test_x.shape))
```

```
train_x's shape: (12288, 209)
test_x's shape: (12288, 50)
```

3 - Architecture of your model

Now that you are familiar with the dataset, it is time to build a deep neural network to distinguish cat images from non-cat images.

2-layer neural network

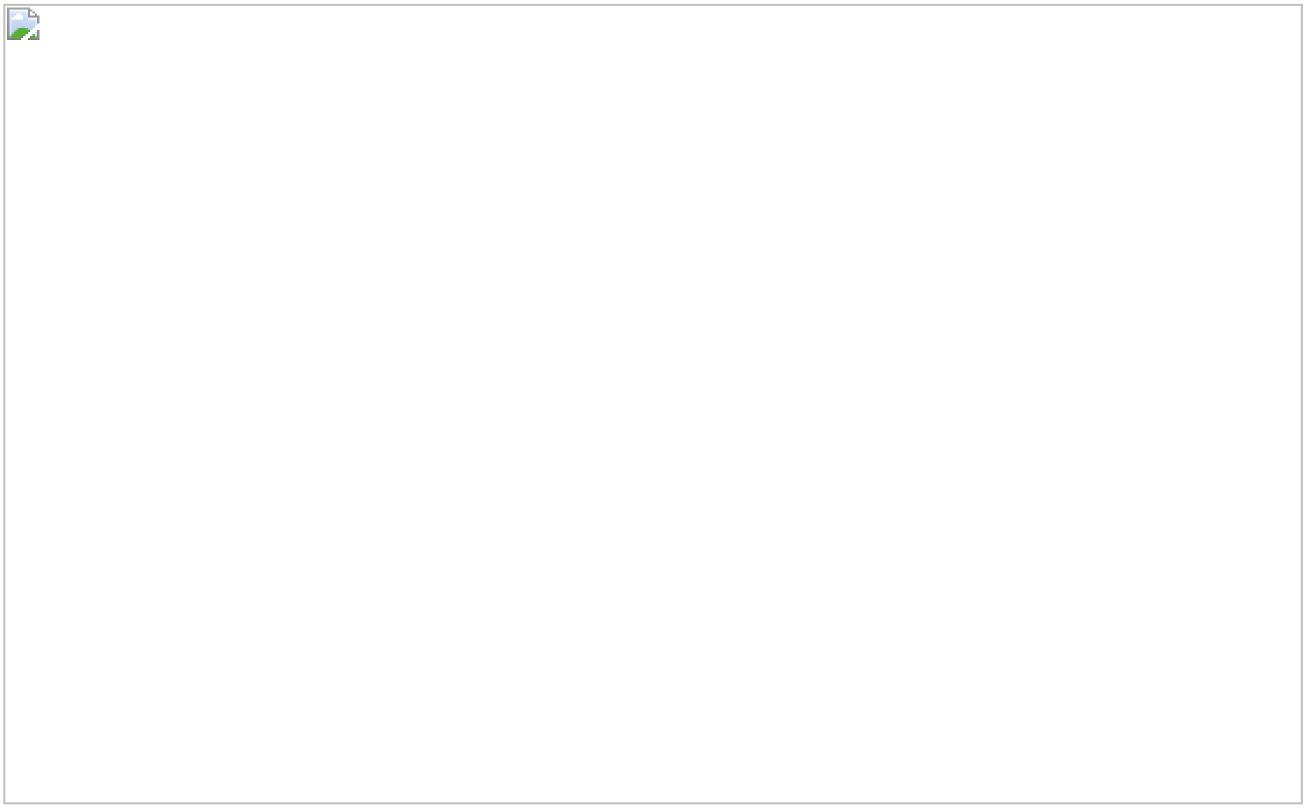


Figure 2: 2-layer neural network.
The model can be summarized as: ***INPUT -> LINEAR -> RELU -> LINEAR -> SIGMOID -> OUTPUT***.

Detailed Architecture of figure 2:

- The input is a (64,64,3) image which is flattened to a vector of size $(12288, 1)$.
- The corresponding vector: $[x_0, x_1, \ldots, x_{12287}]^T$ is then multiplied by the weight matrix $W^{[1]}$ of size $(n^{[1]}, 12288)$.
- You then add a bias term and take its relu to get the following vector: $[a_0^{[1]}, a_1^{[1]}, \ldots, a_{n^{[1]}-1}^{[1]}]^T$.
- You multiply the resulting vector by $W^{[2]}$ and add your intercept (bias).
- Finally, you take the sigmoid of the result. If it is greater than 0.5, you classify it to be a cat.

General methodology

As usual you will follow the Deep Learning methodology to build the model:

```
1. Initialize parameters / Define hyperparameters
2. Loop for num_iterations:
    a. Forward propagation
    b. Compute loss function
    c. Backward propagation
    d. Update parameters (using parameters, and grads from backprop)
```

Question: Use the helper functions you have implemented in the previous assignment to build a 2-layer neural network with the following structure: *LINEAR -> RELU -> LINEAR -> SIGMOID*. The functions you may need and their inputs are:

```python
def initialize_parameters(n_x, n_h, n_y):
    ...
    return parameters
def linear_activation_forward(A_prev, W, b, activation):
    ...
    return A, cache
def compute_loss(AL, Y):
    ...
    return loss
def linear_activation_backward(dA, cache, activation):
    ...
    return dA_prev, dW, db
def update_parameters(parameters, grads, learning_rate):
    ...
    return parameters
```

In [113]:
```python
### CONSTANTS DEFINING THE MODEL ####
n_x = 12288     # num_px * num_px * 3
n_h = 15
n_y = 1
layers_dims = (n_x, n_h, n_y)
```

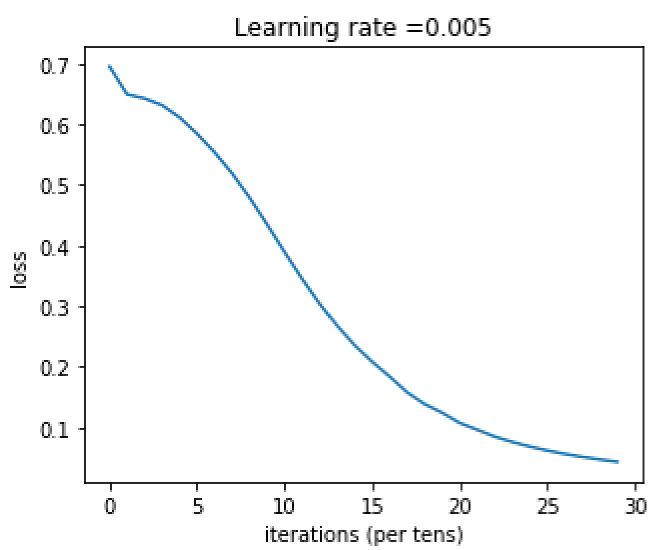
In [114]:
```
def two_layer_model(X, Y, layers_dims, learning_rate = 0.0075, num_it
erations = 3000, print_loss=False):
    """
    Implements a two-layer neural network: LINEAR->RELU->LINEAR->SIGM
OID.

    Arguments:
    X -- input data, of shape (n_x, number of examples)
    Y -- true "label" vector (containing 0 if cat, 1 if non-cat), of
 shape (1, number of examples)
    layers_dims -- dimensions of the layers (n_x, n_h, n_y)
    num_iterations -- number of iterations of the optimization loop
    learning_rate -- learning rate of the gradient descent update rul
e
    print_loss -- If set to True, this will print the loss every 100
 iterations

    Returns:
    parameters -- a dictionary containing W1, W2, b1, and b2
    """

    np.random.seed(1)
    grads = {}
    losses = []                              # to keep track of the l
oss
    m = X.shape[1]                           # number of examples
    (n_x, n_h, n_y) = layers_dims

    # Initialize parameters dictionary, by calling one of the functio
ns you'd previously implemented
    ### START CODE HERE ### (≈ 1 line of code)

    parameters = initialize_parameters(n_x, n_h, n_y)

    ### END CODE HERE ###

    # Get W1, b1, W2 and b2 from the dictionary parameters.
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    # Loop (gradient descent)

    for i in range(0, num_iterations):

        # Forward propagation: LINEAR -> RELU -> LINEAR -> SIGMOID. I
nputs: "X, W1, b1, W2, b2". Output: "A1, cache1, A2, cache2".
        ### START CODE HERE ### (≈ 2 lines of code)

        A1, cache1 = linear_activation_forward(X, W1, b1, "relu")
        A2, cache2 = linear_activation_forward(A1, W2, b2, "sigmoid")

        ### END CODE HERE ###

        # Compute loss
```

```
        ### START CODE HERE ### (≈ 1 line of code)

        loss = compute_loss(A2, Y)

        ### END CODE HERE ###

        # Initializing backward propagation
        dA2 = - (np.divide(Y, A2) - np.divide(1 - Y, 1 - A2))/m

        # Backward propagation. Inputs: "dA2, cache2, cache1". Output
s: "dA1, dW2, db2; also dA0 (not used), dW1, db1".
        ### START CODE HERE ### (≈ 2 lines of code)

        dA1, dW2, db2 = linear_activation_backward(dA2, cache2, "sigm
oid")
        dA0, dW1, db1 = linear_activation_backward(dA1, cache1, "rel
u")

        ### END CODE HERE ###

        # Set grads['dWl'] to dW1, grads['db1'] to db1, grads['dW2']
 to dW2, grads['db2'] to db2
        ### START CODE HERE ### (≈ 4 lines of code)

        grads['dW1'] = dW1
        grads['db1'] = db1
        grads['dW2'] = dW2
        grads['db2'] = db2

        ### END CODE HERE ###


        # Update parameters.
        ### START CODE HERE ### (approx. 1 line of code)

        parameters = update_parameters(parameters, grads, learning_ra
te)

        ### END CODE HERE ###

        # Retrieve W1, b1, W2, b2 from parameters
        W1 = parameters["W1"]
        b1 = parameters["b1"]
        W2 = parameters["W2"]
        b2 = parameters["b2"]

        # Print the loss every 100 training example
        if print_loss and i % 100 == 0:
            print("Loss after iteration {}: {}".format(i, np.squeeze(
loss)))
        if print_loss and i % 100 == 0:
            losses.append(loss)

    # plot the loss

    plt.plot(np.squeeze(losses))
    plt.ylabel('loss')
```

```
    plt.xlabel('iterations (per tens)')
    plt.title("Learning rate =" + str(learning_rate))
    plt.show()

    return parameters
```

In [130]:
```
parameters = two_layer_model(train_x, train_y, layers_dims = (n_x, n_
h, n_y), learning_rate=0.005, num_iterations = 3000, print_loss=True)
```

```
Loss after iteration 0: 0.695340595645
Loss after iteration 100: 0.649548208986
Loss after iteration 200: 0.642676975158
Loss after iteration 300: 0.631660284394
Loss after iteration 400: 0.611600199222
Loss after iteration 500: 0.584694561332
Loss after iteration 600: 0.554108895641
Loss after iteration 700: 0.519408709446
Loss after iteration 800: 0.479346954263
Loss after iteration 900: 0.435289848899
Loss after iteration 1000: 0.389759684786
Loss after iteration 1100: 0.345536220696
Loss after iteration 1200: 0.304332345327
Loss after iteration 1300: 0.26739268936
Loss after iteration 1400: 0.235050695399
Loss after iteration 1500: 0.207926871865
Loss after iteration 1600: 0.183558187172
Loss after iteration 1700: 0.157476388129
Loss after iteration 1800: 0.13816776328
Loss after iteration 1900: 0.123984741236
Loss after iteration 2000: 0.10718025021
Loss after iteration 2100: 0.096021444701
Loss after iteration 2200: 0.0846817819257
Loss after iteration 2300: 0.0759370533833
Loss after iteration 2400: 0.0683579103607
Loss after iteration 2500: 0.0618350235786
Loss after iteration 2600: 0.0562150908573
Loss after iteration 2700: 0.0513348887243
Loss after iteration 2800: 0.0470535200906
Loss after iteration 2900: 0.0432182189221
```



Expected Output:

Loss after iteration 0	0.6930497356599888
Loss after iteration 100	0.6464320953428849
...	...
Loss after iteration 2400	0.048554785628770206

Good thing you built a vectorized implementation! Otherwise it might have taken 10 times longer to train this.

Now, you can use the trained parameters to classify images from the dataset.

Exercise:

- Implement the forward function
- Implement the predict function below to make prediction on test_images

In [131]:
```
def two_layer_forward(X, parameters):
    """
    Implement forward propagation for the [LINEAR->RELU]*(L-1)->LINEA
R->SIGMOID computation

    Arguments:
    X -- data, numpy array of shape (input size, number of examples)
    parameters -- output of initialize_parameters_deep()

    Returns:
    AL -- last post-activation value
    caches -- list of caches containing:
                every cache of linear_relu_forward() (there are L-1 o
f them, indexed from 0 to L-2)
                the cache of linear_sigmoid_forward() (there is one,
 indexed L-1)
    """

    caches = []
    A = X

    # Implement LINEAR -> RELU. Add "cache" to the "caches" list.
    ### START CODE HERE ### (approx. 3 line of code)

    W1, b1 = parameters["W1"], parameters["b1"]
    A1, cache1 = linear_activation_forward(A, W1, b1, "relu")
    caches.append(cache1)

    ### END CODE HERE ###

    # Implement LINEAR -> SIGMOID. Add "cache" to the "caches" list.
    ### START CODE HERE ### (approx. 3 line of code)

    W2, b2 = parameters["W2"], parameters["b2"]
    A2, cache2 = linear_activation_forward(A1, W2, b2, "sigmoid")
    caches.append(cache2)

    ### END CODE HERE ###

    assert(A2.shape == (1,X.shape[1]))

    return A2, caches
```

In [132]:
```
def predict(X, y, parameters):
    """
    This function is used to predict the results of a  L-layer neural 
network.

    Arguments:
    X -- data set of examples you would like to label
    parameters -- parameters of the trained model

    Returns:
    p -- predictions for the given dataset X
    """

    m = X.shape[1]
    n = len(parameters) // 2 # number of layers in the neural network
    p = np.zeros((1,m))

    # Forward propagation
    ### START CODE HERE ### (≈ 1 lines of code)

    probas, caches = two_layer_forward(X, parameters)

    ### END CODE HERE ###

    # convert probas to 0/1 predictions
    for i in range(0, probas.shape[1]):
        ### START CODE HERE ### (≈ 4 lines of code)

        if(probas[0][i] > 0.5):
            p[0][i] = 1
        else:
            p[0][i] = 0

        ### END CODE HERE ###

    print("Accuracy: "  + str(float(np.sum((p == y)/float(m)))))
    return p
```

In [133]:
```
predictions_train = predict(train_x, train_y, parameters)
```

```
Accuracy: 1.0
```

In [134]:
```
predictions_test = predict(test_x, test_y, parameters)
```

```
Accuracy: 0.72
```

Exercise: Identify the hyperparameters in the model and For each hyperparameter

- Briefly explain its role
- Explore a range of values and describe their impact on (a) training loss and (b) test accuracy
- Report the best hyperparameter value found.

Note: Provide your results and explanations in the report for this question.

Hyperparameters The hyperparameters are:

1. Learning rate - It is used for updating the parameters of the neural network that is the weights and biases of the neural network. It controls the amount of update that needs to take place so that we are able to reach the minima of the loss function.
2. Epochs - It represents the number of times the network sees the data and adjusts its parameters for optimal learning.
3. Number of hidden neurons in the hidden layer - The number of neurons in the hidden layer where each neuron is learning some properties of the input data and able to establish a relationship between input and output.

Values of Hyperparameters tried:

1. Learning rate = 0.05, Epochs = 3000, Hidden neurons = 50, Training loss = 0.05, Testing accuracy: 74%
2. Learning rate = 0.01, Epochs = 3000, Hidden neurons = 50, Training loss = 0.05, Testing accuracy: 74%
3. Learning rate = 0.01, Epochs = 4000, Hidden neurons = 50, Training loss = 0.05, Testing accuracy: 74%
4. Learning rate = 0.001, Epochs = 3000, Hidden neurons = 50, Training loss = 0.52, Testing accuracy: 58%
5. Learning rate = 0.005, Epochs = 3000, Hidden neurons = 50, Training loss = 0.03, Testing accuracy: 70%
6. Learning rate = 0.05, Epochs = 3000, Hidden neurons = 40, Training loss = 0.001, Testing accuracy: 76%
7. Learning rate = 0.01, Epochs = 3000, Hidden neurons = 40, Training loss = 0.01, Testing accuracy: 72%
8. Learning rate = 0.01, Epochs = 4000, Hidden neurons = 40, Training loss = 0.02, Testing accuracy: 72%
9. Learning rate = 0.001, Epochs = 3000, Hidden neurons = 40, Training loss = 0.52, Testing accuracy: 54%
10. Learning rate = 0.005, Epochs = 3000, Hidden neurons = 40, Training loss = 0.04, Testing accuracy: 74%
11. Learning rate = 0.05, Epochs = 3000, Hidden neurons = 30, Training loss = 0.005, Testing accuracy: 74%
12. Learning rate = 0.01, Epochs = 3000, Hidden neurons = 30, Training loss = 0.0011, Testing accuracy: 76%
13. Learning rate = 0.01, Epochs = 4000, Hidden neurons = 30, Training loss = 0.001, Testing accuracy: 76%
14. Learning rate = 0.001, Epochs = 3000, Hidden neurons = 30, Training loss = 0.53, Testing accuracy: 50%
15. Learning rate = 0.005, Epochs = 3000, Hidden neurons = 30, Training loss = 0.04, Testing accuracy: 72%
16. Learning rate = 0.05, Epochs = 3000, Hidden neurons = 20, Training loss = 0.0009, Testing accuracy: 78%
17. Learning rate = 0.01, Epochs = 3000, Hidden neurons = 20, Training loss = 0.011, Testing accuracy: 70%
18. Learning rate = 0.01, Epochs = 4000, Hidden neurons = 20, Training loss = 0.02, Testing accuracy: 70%
19. Learning rate = 0.001, Epochs = 3000, Hidden neurons = 20, Training loss = 0.54, Testing accuracy: 46%
20. Learning rate = 0.005, Epochs = 3000, Hidden neurons = 20, Training loss = 0.04, Testing accuracy: 72%
21. Learning rate = 0.05, Epochs = 3000, Hidden neurons = 15, Training loss = 0.005, Testing accuracy: 70%
22. Learning rate = 0.01, Epochs = 3000, Hidden neurons = 15, Training loss = 0.011, Testing accuracy: 74%
23. Learning rate = 0.01, Epochs = 4000, Hidden neurons = 15, Training loss = 0.02, Testing accuracy: 74%
24. Learning rate = 0.001, Epochs = 3000, Hidden neurons = 15, Training loss = 0.56, Testing accuracy: 36%
25. Learning rate = 0.005, Epochs = 3000, Hidden neurons = 15, Training loss = 0.043, Testing accuracy: 72%

Optimal hyperparameters found

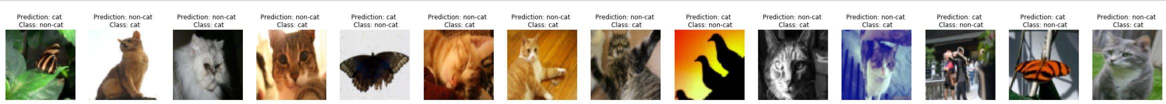
1. Learning rate = 0.05
2. Epochs = 3000
3. Number of hidden neurons = 20

Results Analysis

First, let's take a look at some images the 2-layer model labeled incorrectly. This will show a few mislabeled images.

In [135]:
```
def print_mislabeled_images(classes, X, y, p):
    """
    Plots images where predictions and truth were different.
    X -- dataset
    y -- true labels
    p -- predictions
    """
    a = p + y
    mislabeled_indices = np.asarray(np.where(a == 1))
    plt.rcParams['figure.figsize'] = (40.0, 40.0) # set default size
 of plots
    num_images = len(mislabeled_indices[0])
    for i in range(num_images):
        index = mislabeled_indices[1][i]

        plt.subplot(2, num_images, i + 1)
        plt.imshow(X[:,index].reshape(64,64,3), interpolation='neares
t')
        plt.axis('off')
        plt.title("Prediction: " + classes[int(p[0,index])].decode("u
tf-8") + " \n Class: " + classes[y[0,index]].decode("utf-8"))
```

In [136]:
```
print_mislabeled_images(classes, test_x, test_y, predictions_test)
```



Exercise: Identify a few types of images that tends to perform poorly on the model

Answer The model performs poorly when the cat is at certain angle or rotated at some angle, which makes it classify it as a non-cat class.

Now, lets use the same architecture to predict sentiment of movie reviews. In this section, most of the implementation is already provided. The exercises are mainly to understand what the workflow is when handling the text data.

In [137]:
```
import re
```

Dataset

Problem Statement: You are given a dataset ("train_imdb.txt", "test_imdb.txt") containing:

```
- a training set of m_train reviews
- a test set of m_test reviews
- the labels for the training examples are such that the first 50% belong to
class 1 (positive) and the rest 50% of the data belong to class 0(negative)
```

Let's get more familiar with the dataset. Load the data by completing the function and run the cell below.

In [138]:
```
def load_data(train_file, test_file):
    train_dataset = []
    test_dataset = []

    # Read the training dataset file line by line
    for line in open(train_file, 'r'):
        train_dataset.append(line.strip())

    for line in open(test_file, 'r'):
        test_dataset.append(line.strip())
    return train_dataset, test_dataset
```

In [139]:
```
train_file = "data/train_imdb.txt"
test_file = "data/test_imdb.txt"
train_dataset, test_dataset = load_data(train_file, test_file)
```

In [140]:
```
# This is just how the data is organized. The first 50% data is posit
ive and the rest 50% is negative for both train and test splits.
y = [1 if i < len(train_dataset)*0.5 else 0 for i in range(len(train_
dataset))]
```

As usual, lets check our dataset

In [141]:
```
# Example of a review
index = 10
print(train_dataset[index])
print ("y = " + str(y[index]))
```

```
I liked the film. Some of the action scenes were very interesting, te
nse and well done. I especially liked the opening scene which had a s
emi truck in it. A very tense action scene that seemed well done.<br
/><br />Some of the transitional scenes were filmed in interesting wa
ys such as time lapse photography, unusual colors, or interesting ang
les. Also the film is funny is several parts. I also liked how the ev
il guy was portrayed too. I'd give the film an 8 out of 10.
y = 1
```

In [142]:
```
# Explore your dataset
m_train = len(train_dataset)
m_test = len(test_dataset)

print ("Number of training examples: " + str(m_train))
print ("Number of testing examples: " + str(m_test))
```

```
Number of training examples: 1001
Number of testing examples: 201
```

Pre-Processing

From the example review, you can see that the raw data is really noisy! This is generally the case with the text data. Hence, Preprocessing the raw input and cleaning the text is essential. Please run the code snippet provided below.

Exercise: Explain what pattern the model is trying to capture using re.compile in your report.

Answer

1. re.compile() removes special characters like ', . " etc and makes all characters in lowercase. It is learning properties from words.

In [143]:
```
REPLACE_NO_SPACE = re.compile("(\.)|(\;)|(\:)|(\!)|(\')|(\?)|(\,)|(\"
)|(\()|(\))|(\[)|(\])|(\d+)")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
NO_SPACE = ""
SPACE = " "

def preprocess_reviews(reviews):

    reviews = [REPLACE_NO_SPACE.sub(NO_SPACE, line.lower()) for line
in reviews]
    reviews = [REPLACE_WITH_SPACE.sub(SPACE, line) for line in review
s]

    return reviews

train_dataset_clean = preprocess_reviews(train_dataset)
test_dataset_clean = preprocess_reviews(test_dataset)
```

In [144]:
```
# Example of a clean review
index = 10
print(train_dataset_clean[index])
print ("y = " + str(y[index]))
```

```
i liked the film some of the action scenes were very interesting tens
e and well done i especially liked the opening scene which had a semi
truck in it a very tense action scene that seemed well done some of t
he transitional scenes were filmed in interesting ways such as time l
apse photography unusual colors or interesting angles also the film i
s funny is several parts i also liked how the evil guy was portrayed
too id give the film an  out of
y = 1
```

Vectorization

Now lets create a feature vector for our reviews based on a simple bag of words model. So, given an input text, we need to create a numerical vector which is simply the vector of word counts for each word of the vocabulary. Run the code below to get the feature representation.

In [145]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(binary=True, stop_words="english", max_features=
2000)
cv.fit(train_dataset_clean)
X = cv.transform(train_dataset_clean)
X_test = cv.transform(test_dataset_clean)
```

CountVectorizer provides a sparse feature representation by default which is reasonable because only some words occur in individual example. However, for training neural network models, we generally use a dense representation vector.

In [146]:
```
X = np.array(X.todense()).astype(float)
X_test = np.array(X_test.todense()).astype(float)
y = np.array(y)
```

Model

In [147]:
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    X, y, train_size = 0.80
)
```

```
/home/arpitdec5/.local/lib/python2.7/site-packages/sklearn/model_sele
ction/_split.py:2178: FutureWarning: From version 0.21, test_size wil
l always complement train_size unless both are specified.
  FutureWarning)
```

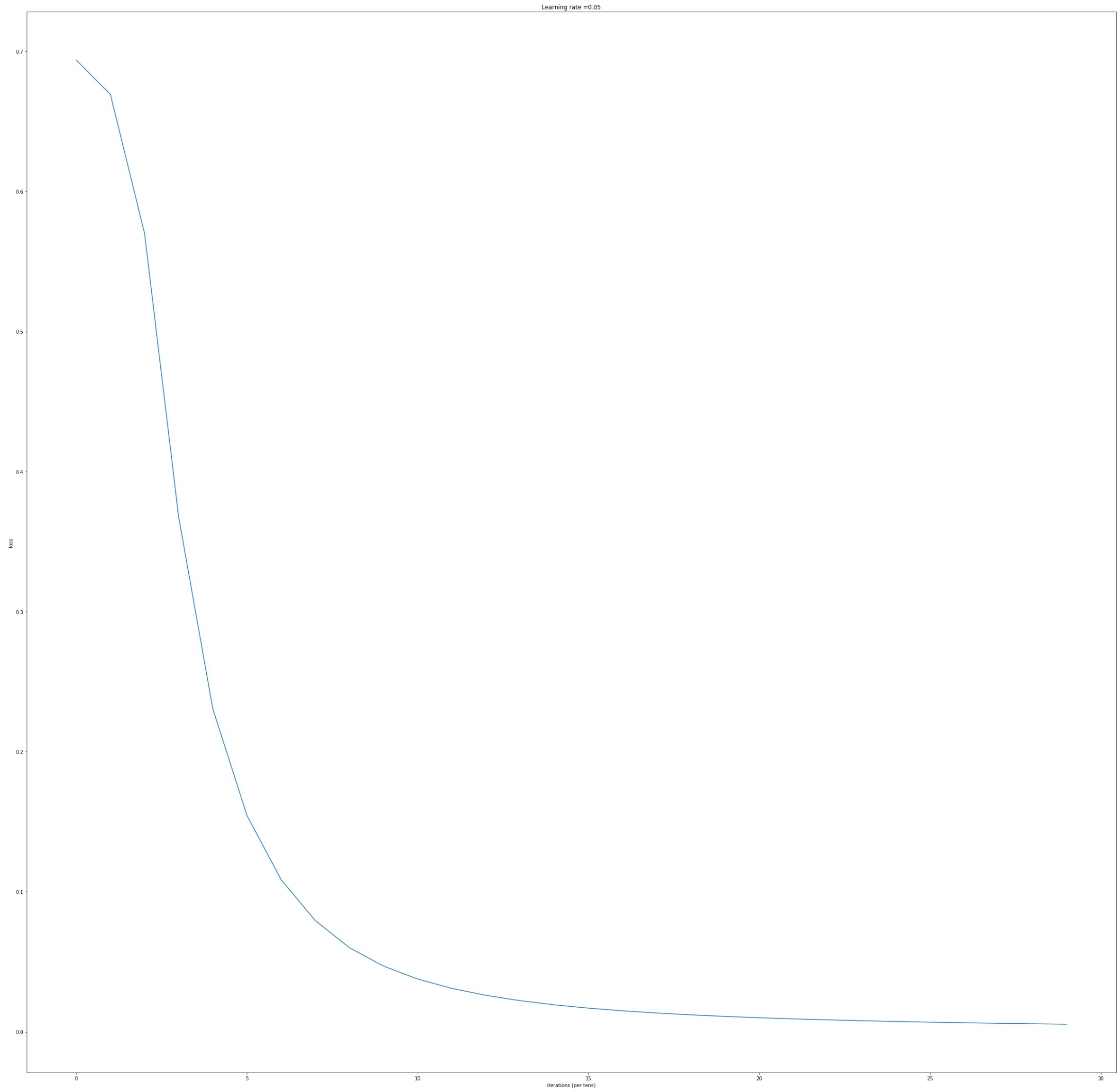
In [148]:
```
# This is just to correct the shape of the arrays as required by the
 two_layer_model
X_train = X_train.T
X_val = X_val.T
y_train = y_train.reshape(1,-1)
y_val = y_val.reshape(1,-1)
```

In [165]:
```
### CONSTANTS DEFINING THE MODEL ####
n_x = X_train.shape[0]
n_h = 800
n_y = 1
layers_dims = (n_x, n_h, n_y)
```

We will use the same two layer model that you completed in the previous section for training.

In [166]:
```
parameters = two_layer_model(X_train, y_train, layers_dims = (n_x, n_h, n_y), learning_rate=0.05, num_iterations = 3000, print_loss=True)
```

```
Loss after iteration 0: 0.693741186555
Loss after iteration 100: 0.669243147014
Loss after iteration 200: 0.570034533211
Loss after iteration 300: 0.367406293499
Loss after iteration 400: 0.230506374002
Loss after iteration 500: 0.154438318239
Loss after iteration 600: 0.108656728844
Loss after iteration 700: 0.0794306058322
Loss after iteration 800: 0.0600857685661
Loss after iteration 900: 0.0469245239498
Loss after iteration 1000: 0.0376980944249
Loss after iteration 1100: 0.0310273960424
Loss after iteration 1200: 0.0260620102052
Loss after iteration 1300: 0.0222701491873
Loss after iteration 1400: 0.019308336846
Loss after iteration 1500: 0.016948431047
Loss after iteration 1600: 0.0150356278223
Loss after iteration 1700: 0.0134616437642
Loss after iteration 1800: 0.0121489007512
Loss after iteration 1900: 0.0110410883932
Loss after iteration 2000: 0.0100963185528
Loss after iteration 2100: 0.00928302482988
Loss after iteration 2200: 0.00857702273427
Loss after iteration 2300: 0.00795952326132
Loss after iteration 2400: 0.00741573609318
Loss after iteration 2500: 0.00693385723277
Loss after iteration 2600: 0.00650442367197
Loss after iteration 2700: 0.00611972158743
Loss after iteration 2800: 0.00577344849862
Loss after iteration 2900: 0.0054603771549
```



Predict the review for our movies!

In [167]:
```
predictions_train = predict(X_train, y_train, parameters)
```

```
Accuracy: 1.0
```

In [168]:
```
predictions_val = predict(X_val, y_val, parameters)
```

```
Accuracy: 0.8407960199
```

Results Analysis

Let's take a look at some examples the 2-layer model labeled incorrectly

In [169]:
```
def print_mislabeled_reviews(X, y, p):
    """
    Plots images where predictions and truth were different.
    X -- dataset
    y -- true labels
    p -- predictions
    """
    a = p + y
    mislabeled_indices = np.asarray(np.where(a == 1))
    plt.rcParams['figure.figsize'] = (40.0, 40.0) # set default size
 of plots
    num_reviews = len(mislabeled_indices[0])
    for i in range(num_reviews):
        index = mislabeled_indices[1][i]

        print((" ").join(cv.inverse_transform(X[index])[0]))
        print("Prediction: " + str(int(p[0,index])) + " \n Class: " +
str(y[0,index]))
```

In [170]:
```
print_mislabeled_reviews(X_val.T, y_val, predictions_val)
```

```
achieve acting actor actors adaptation animal annoying author away ba
ckground beautiful better book books camp care characters charismatic
chosen cinematography clear complaint complex country critics despite
different director doing drinking dull episode episodes especially ex
ist fact feels film films forget friends good guess hand having heavy
humour just know leads life like little lot main mainly make makes me
et men moment movie need novel obvious opinion opposite party people
person pleasant plot positive predictable presented presents psycholo
gical quiet quite rarely read reading real really recent relationship
scenes screen screenplay script somewhat spends story strong suffers
surprise surprised surprisingly taking talented thats thriller tim ti
me totally twice ultimately van woman works writers writing wrote yea
r
Prediction: 0
 Class: 1
actually apparent brother close connected cop drive extreme far figur
e films forget gang getting great hard harsh involved kid killer lawy
er look lot movie movies need ones place pretty scenes seen sex socie
ty sort start style takes themes tough trilogy true trying typical us
ual violent way ways worse worth youll youve
Prediction: 0
 Class: 1
animal ask aspect bad chair charlie check classic country desperate d
ont eventually fan film films final finale foreign forget friends get
s gore greatest guts heard hell heres highly horror hours japanese jo
b just kill kinds knowing like listen make movie people probably pure
rating read really recommend said seen series shes shock sick snuff s
ound spin star storyline stuff things think throw times torture under
ground use watch watching woman
Prediction: 0
 Class: 1
accident acting actually anti away bad beginning best better bit brin
g car character comes completely confusing create created decent devi
ce dialogue didnt die director effect elements episode expecting extr
eme face far feel felt film forth fresh future genres ghost gives goe
s going good gore grace great harder hero hes hope horrible horror id
ea im images issue john key later left let like little live look look
s main make man master nice opinion particularly plot prior protagoni
st read relationship remind revenge review reviewer revolves right sa
id saving say season second seen sense solid somewhat spectacular sta
te story strong suspense takes talking think time times turn twists v
iewer wants watch watching way wife wish wrong
Prediction: 0
 Class: 1
accurate actors actual actually amazing bad best better big bother br
eak budget came cameron capture care certainly cover critics day dece
nt dialogue dicaprio did didnt dont effects enjoyed face film films f
ine form general going good guess guys heard hearing hit huge includi
ng isnt ive james job just lets like love make maybe million mind mon
ey mouth movie needed obviously people personally public real really
romance say screen seeing seen ship sinking special star started stor
y success sucks sweet talk talking thank theaters thing think thought
time times titanic took touch twice type wasnt winslet word work wors
t worth wouldnt writers years
Prediction: 0
 Class: 1
angry attempts author band based bit case characters clever compariso
```

```
n didnt directors disappointing documentary dream excellent family fe
stival fiction film form genres good half just la lose lost main mean
t moving order original plus pretty previous quickly reminded rock ro
ll saw science sequences seriously songs story sure sympathetic tell
times tom way werent youll
Prediction: 1
 Class: 0
animal dont hours life love planet script series video wait
Prediction: 1
 Class: 0
actors art ask attempts audiences awful beautiful case character char
acters cinema cute date deliver despite development doesnt exactly ex
pectations expected expecting experience fails female film flaws fran
k good hes high humour interesting know lead leads learn like liked m
ain male matters men movie nice painful people plays potential pretty
questions reviews romance romantic school screenplay shes simply stor
y takes thats theyre tries try victim viewer women written youre
Prediction: 1
 Class: 0
actual actually better bore bored called course easily ghost good hum
an ive just like makes mindless movie original parts reporter reveale
d said slow st started story thats think thrown time town usually wat
ch watched wow yes
Prediction: 1
 Class: 0
acting actually added ask away basic biggest blame character characte
rs children come coming convincing couple decent development didnt di
sappointing does doesnt dont english european experience female fully
good great job king know later learns little lot lover main meant mov
ie musical ok opinion piece played portrayal possibly presence really
romance screenplay situation situations songs st stand story storylin
e things times understand viewer watched wife woman
Prediction: 1
 Class: 0
actually aspect bad bit certainly chinese decent english explained ex
tent family fighting film follow half hard head japanese just know la
nguage little make maybe middle plenty plus problem problems reading
ride sexual simply storyline tried twists unlikely violence watching
window
Prediction: 0
 Class: 1
ability able accident action actresses actually aspect away bad belie
ve better bit blood brothers cause cgi charlie crap crime cut days de
al death disturbing does doesnt dont effects especially eyes fact fak
e favorite film films footage forget funny happens hope horror im ins
tead leaving like look lot make makers making marry money movie movie
s overall people plot point porn probably pull rape rating real sayin
g says scene scenes seen series shocking snuff sound stand stars suck
ed sucks super supposed sure talent talking thing thinking time tried
visual visuals want wanted wasnt watch wouldnt
Prediction: 0
 Class: 1
book character charlotte excellent eyes eyre goes hurt ice jane laid
later like love mr novel passion read rochester saw story time totall
y version watching william wonderful years
Prediction: 1
 Class: 0
```

```
actors ago believe better bit characters contains crazy dvd ed felt g
erman good hard havent head high level like live look lot maybe missi
ng movie movies nasty really saw say scenes seen short sick story tha
ts thing uk version violence years
Prediction: 0
 Class: 1
actually ahead better box budget burning character charlie come damn
development did didnt dont enjoy eye fake far film flick forced genre
good guinea guts hand hear heard horror hours interested just know li
ke listen looks lot low making men minutes movie naturally offer pain
ful pretty really recommend say scene scenes second seen set sharp sh
ort simply snuff story think thought throwing told torture trying ult
imately unless various watching ways went woman worst
Prediction: 0
 Class: 1
apparent audience better book boy certain certainly clear complete da
rk doesnt effort feels felt film good holes inspired intriguing like
makes mystery nature portrayal premise probably production recommend
tension thing true truly williams woman written
Prediction: 0
 Class: 1
action age body brain building certainly computer crazy damme daughte
r dead entertaining especially fan fi fights folks genius goes going
goldberg good government guess hes humor just keeps king lame later l
atest like manages mean named new original particularly perfect power
pretty pro reason run sci sequel shoot site snake soldiers sort step
super takes thriller train usual van war white working wrong year yea
rs youre
Prediction: 1
 Class: 0
actors big deserves doesnt film goes great innocence movie real story
tell think time victor
Prediction: 1
 Class: 0
alive away basic brother camera characters cinema comes cool cop dead
director doesnt ending ends expect experience fairly fallen film gang
hard heart hollywood horrific humour incredibly isnt john just lawyer
list look making moments movie nasty new nice old pace painful quite
reveals said screen script seen sense showing stuff sudden takes taki
ng think thriller time todays truly twice unexpected unusual utterly
violence want way work working youre
Prediction: 0
 Class: 1
absolutely actor art bad bed boss brings called came captured charact
er chinese days definitely effort end english fact felt finish gem ge
ts girl going great harder immediately just kept kid king little live
love masks maybe movie nice night original pass perfectly really scot
t sees started thought totally touches trying turned turns watching
Prediction: 0
 Class: 1
actually ago bad better book church course does enjoyable familiar fi
lm forgotten forward good hadnt heard hour instantly job know laid le
t long minute minutes missed mr nearly overall quick read really scho
ol second sense short simply story tales thats thing time trilogy wat
ched worked write years
Prediction: 0
 Class: 1
```

```
acting actor actors audience authentic belongs better changes charact
ers considered daughter director directors drinking endless essential
feelings fights films foreign good half head india indian involving l
eading lets like live make male movie movies nana old ones patekar re
d return running sad said scene school sell song state storytelling t
ale tell thrown tight tough true village violence violent walk wants
watch wont word words
Prediction: 0
 Class: 1
acting actually appeal aspect audience bad ben big bit break brings c
aused completely consider damn decided did didnt died disbelief dolla
rs dont doubt dumb emotional emotions end enjoy epic exactly films fu
n girl gone good got greatest gross hate heavy idea im impact impress
ion isnt ive just kinda left let like little look make maybe means mi
llion mind mom movie movies order oscars possible power pretty quite
ready reason remember remembered ride saw score screen simply single
special sure theaters thoroughly thrill time times titanic total tv t
wice usually video visual wasnt whats whatsoever works worth years ye
s youll
Prediction: 0
 Class: 1
anybody away ball believe big bit blown boat bodies bother car cares
cause chases course dead death did does doesnt dull dumb ed exactly f
ake feel figure finds gets goes got great guy guys hell help hes hey
hospital hour house huge interested ive job just killer lady lame lat
er leaves like little lives media minute minutes mom oh original plai
n police president real really reporter right secret sharp shes sight
sister spoiler story street stuff suicide talking thats theres theyre
think time told took trailer true trying tv wait wasnt watching whats
white wife working yeah youve
Prediction: 1
 Class: 0
actor air attempts bloody bodies capture chance church continue cross
detective diamond die dont dying effective evil film films finish fli
ck forces gas gate gets griffith hand hands happen heart heavy help h
es host ill kill killer know like line major merely need normally par
tner patrick poor power problem pure really satan serial sister sort
soul stand stop supporting tried try type understanding villain weak
welcome women world
Prediction: 1
 Class: 0
adult atmosphere bad boy brother camera cast characters christopher c
lose despite didnt director drama experience family film given matter
memorable movie natural nearly necessary older people picture project
really robert sad shooting situation small story subject talent treat
ment unfortunately worth young
Prediction: 1
 Class: 0
film forgotten late little long makers money movie night present subt
le time todays tv
Prediction: 0
 Class: 1
ago based best big blood close copy ending enjoy fight funny glad got
guess hey hurt im jesus kill killer like looks lot man memory mention
movie scenes screen seen seven shes shown slasher spoil story surely
suspense thats theres times ups video wearing whos wont wood woods wo
rk years
```

```
Prediction: 1
 Class: 0
annoying beautiful easily escape films finds force george god goes go
od great hand high island isnt jane johnny mention people period play
er present priest robert singer song thing tries try used woman worst
worthy years
Prediction: 1
 Class: 0
action adventure adventures bad camp character characters check crew
decided doc elements familiar fan fans feel feeling film good hero he
roes im james jones just know long lot major minutes movie movies mus
ic number ones promise provided really resulting savage say seeing so
mewhat spirit star thats theres throw time trying unfortunate way
Prediction: 0
 Class: 1
black blood chair course days effects fans film finally gang guts hor
ror idea killing make makers making men murder nasty poor possible pr
esents pretty real seen series sick snuff special thing thrown tortur
e tried water woman
Prediction: 0
 Class: 1
acting animals best better die dont entire episode episodes funny goo
d horrible ice just killing know life like movie obviously plot pro p
roblem really remember right scene scenes season second series shocki
ng suspense think torture turns victims watch women wonderful worst
Prediction: 0
 Class: 1
```

Exercise: Provide explanation as to why these examples were misclassified below.

Type your answer here

The main aim of the model is to predict the sentiment. As each word is taken individually, the model is failing to learn the sentiment of the current word from previous words and is failing to learn from the sentence as a whole.

In [ ]: