**Name:** Mohit Mohit

**BU ID:** U33745431

CS 542 PS3

**Programming Report**

**Sparse Autoencoder**

The code implemented in python. File:- Sparse\_Autoencoder.py

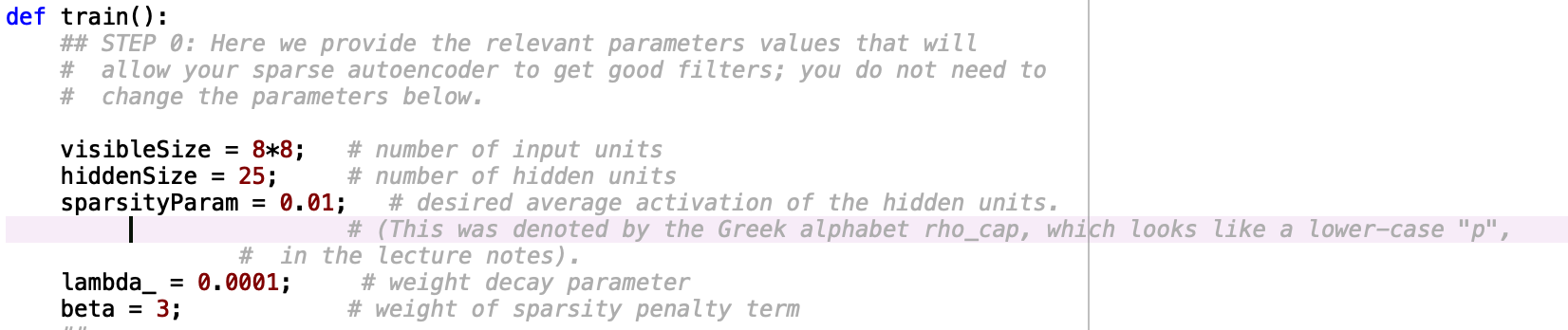
The python file includes implementation of all of the functions as provided in the .m files.

Below steps are implemented in the **Train()** function.

**Step 0**: **Parameter Initialization**

The given parameters in the file *train.m* are initialized. These parameters allows the sparse autoencoder to get good filters.

Initialize the parameters according to the .m file.

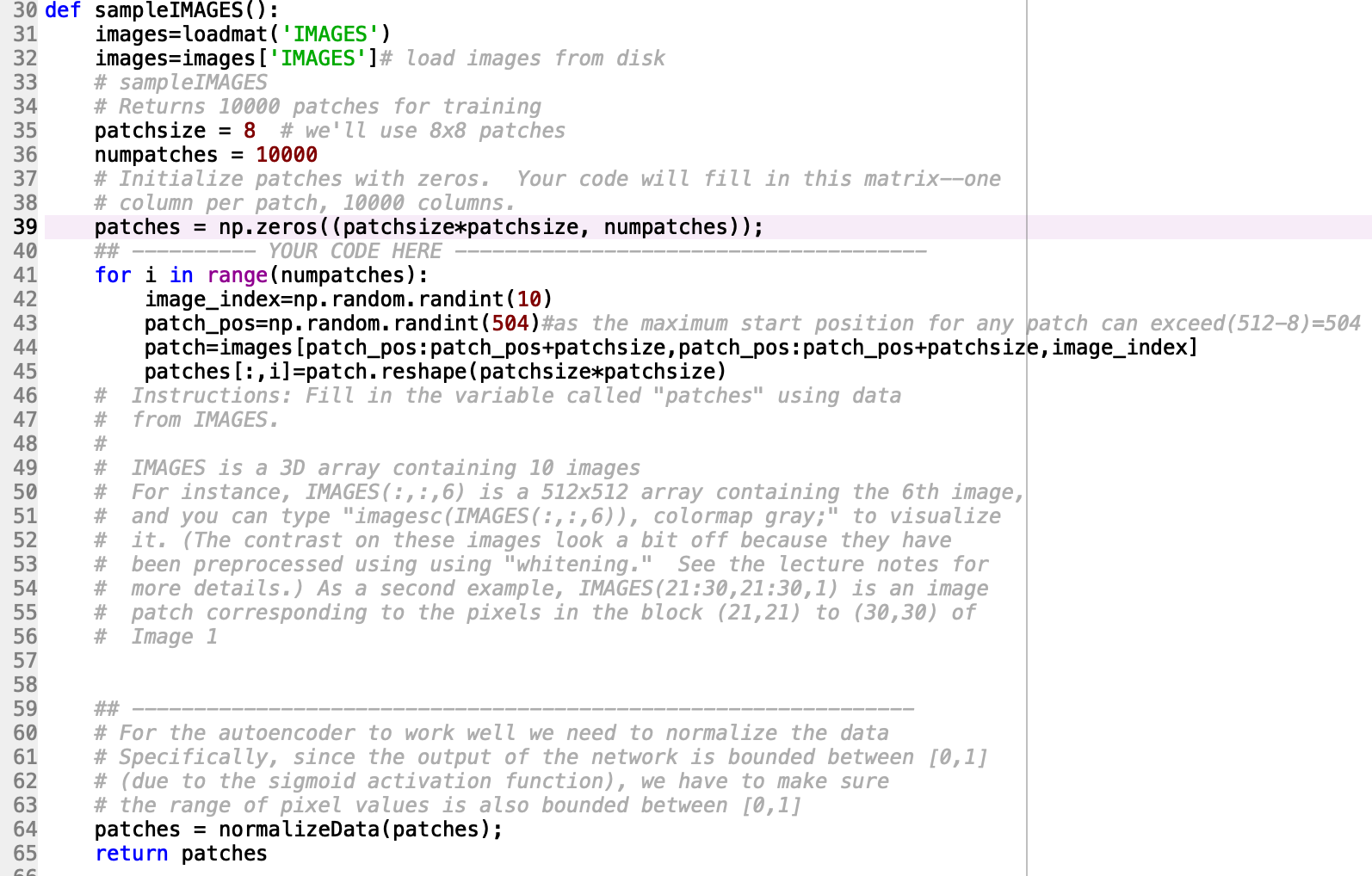


**Step 1**: **Generate training set**

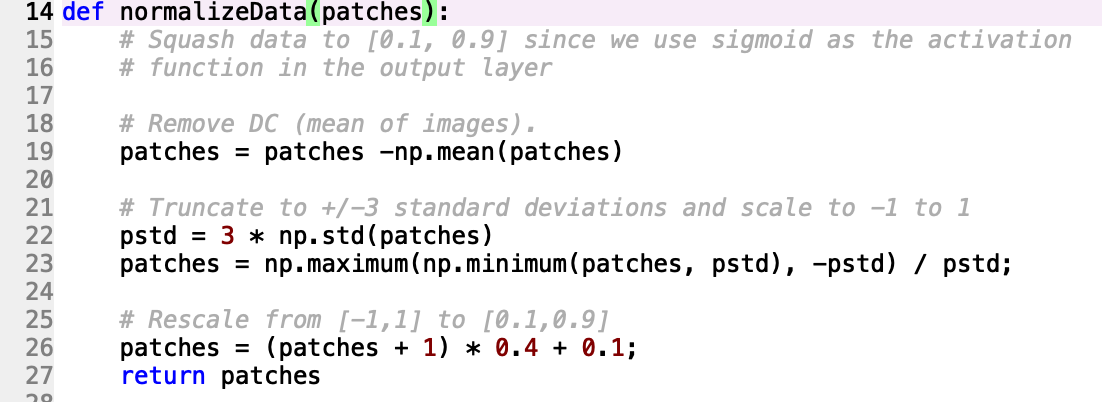
Need to implement **﻿sampleIMAGES()** method.

Will use to load *scipy.io.loadmat* ‘IMAGES.mat’ file.

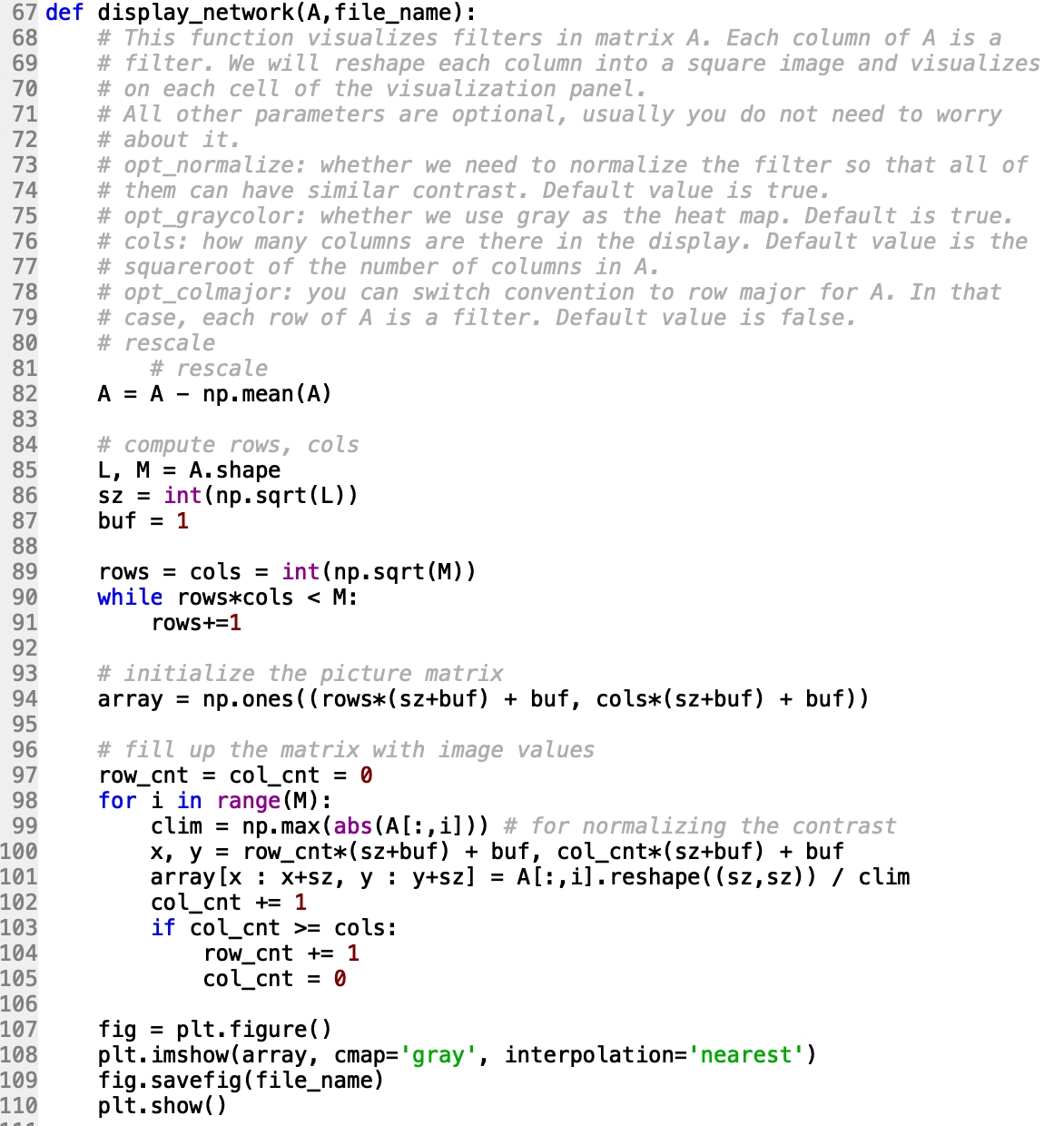
Then we will take the ‘IMAGE’ attribute which hold the 10 images. The training set will be obtained by randomly picking one of the 10 images and then randomly selecting an 8×8 image patch from the selected image, and by converting the image patch into a 64-dimensional vector. We will take 10000 samples and concatenated into a 64×10000 matrix.



For the autoencoder to work well we need to normalize the data. Specifically, since the output of the network is bounded between [0,1] (due to the sigmoid activation function), we have to make sure the range of pixel values is also bounded between [0,1].



Now after creating the patches we will call the **display\_network(A,file\_name)** function to display 200 samples. The function will generate a PDF file with name passed as “file\_name”.

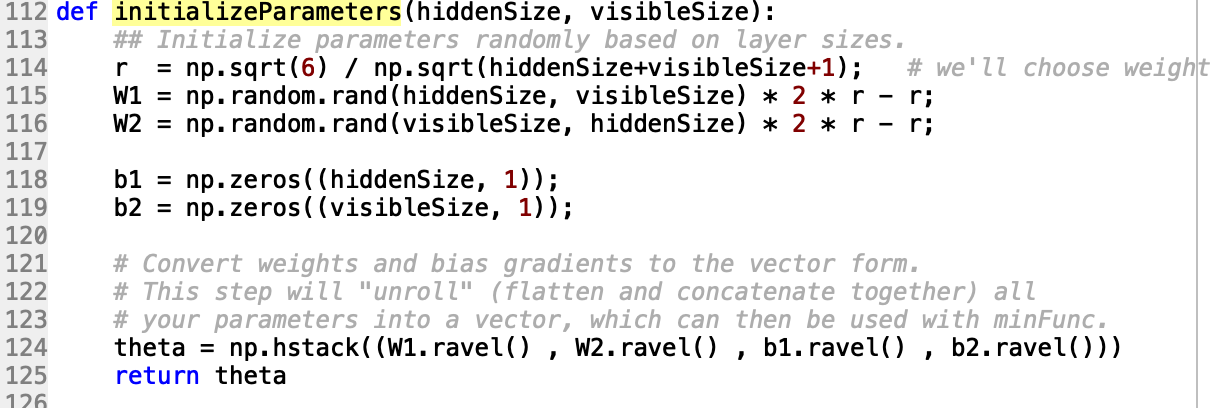


Genrated pdf:-



To obtains random parameter theta: -

Will call the **initializeParameters(hiddenSize,visibleSize)** method with the number of nurones in hidden layer and input layer.



**STEP 2: Implement sparseAutoencoderCost**

Implement **sparseAutoencodercost()** method

This method will take following parameters:-

**visibleSize**: number of input units

**hiddenSize**: number of hidden units

**lamda**\_: weight decay parameters

**sparsityParam**: rho(p)

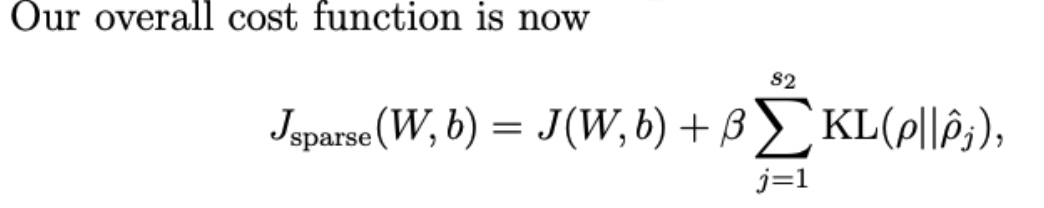
**beta**: weight of sparsity penalty term

**theta**: Is a vector

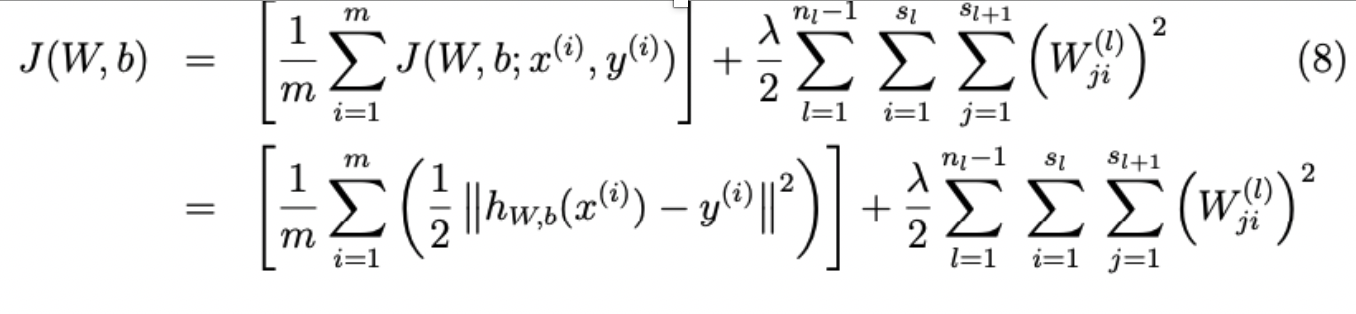
First will take the W1, W2, B1 and B2 from theta .

Then will perform forward propagation and compute the error. We are using sigmoid as the activation function.

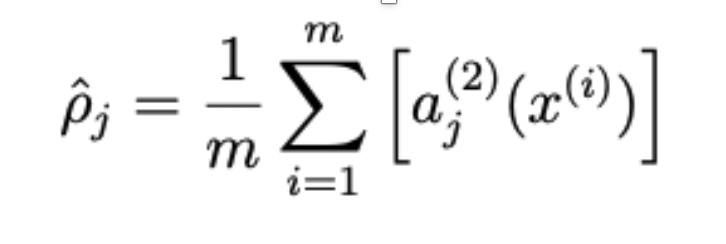
Now by using the following formulas, we will calculate the cost.

****

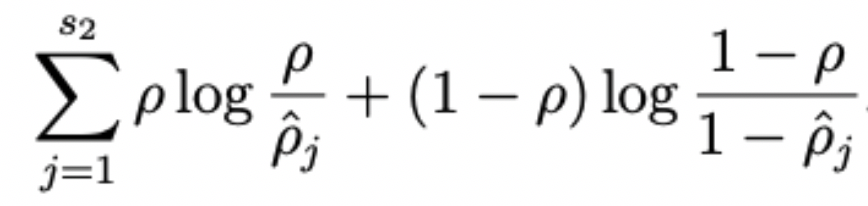
Where J(W,b) is



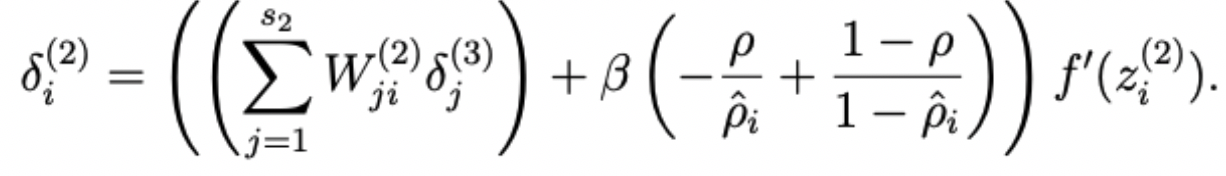
Rho(p) is the sparsity Parameter and for rho^



KL Divergence is given by:-



Next, we need to find the gradients of the weight and bias parameters (Backward propagation).



After the computation of delta values the gradients are updated.

**Code:-**

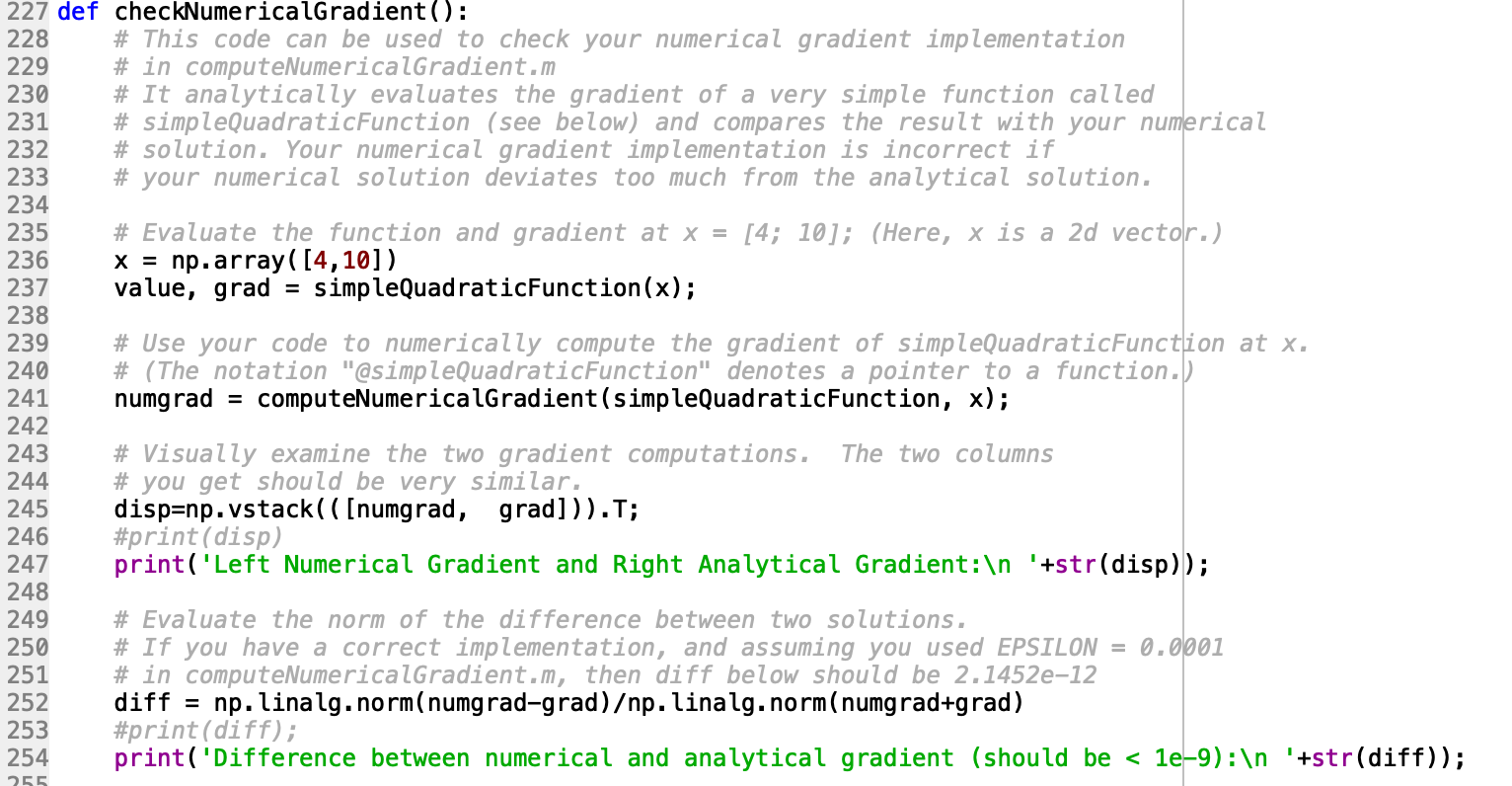


**STEP 3: Gradient Checking**

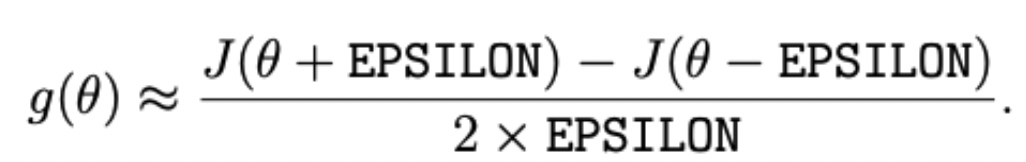
Need to implement **checkNumericalGradient()** function

This function evaluates the gradient of of very simplefunction called simpleQuadraticFunction and compares the result with numerical solution.

If the difference is too much in that case we can say the numerical computation is incorrect.

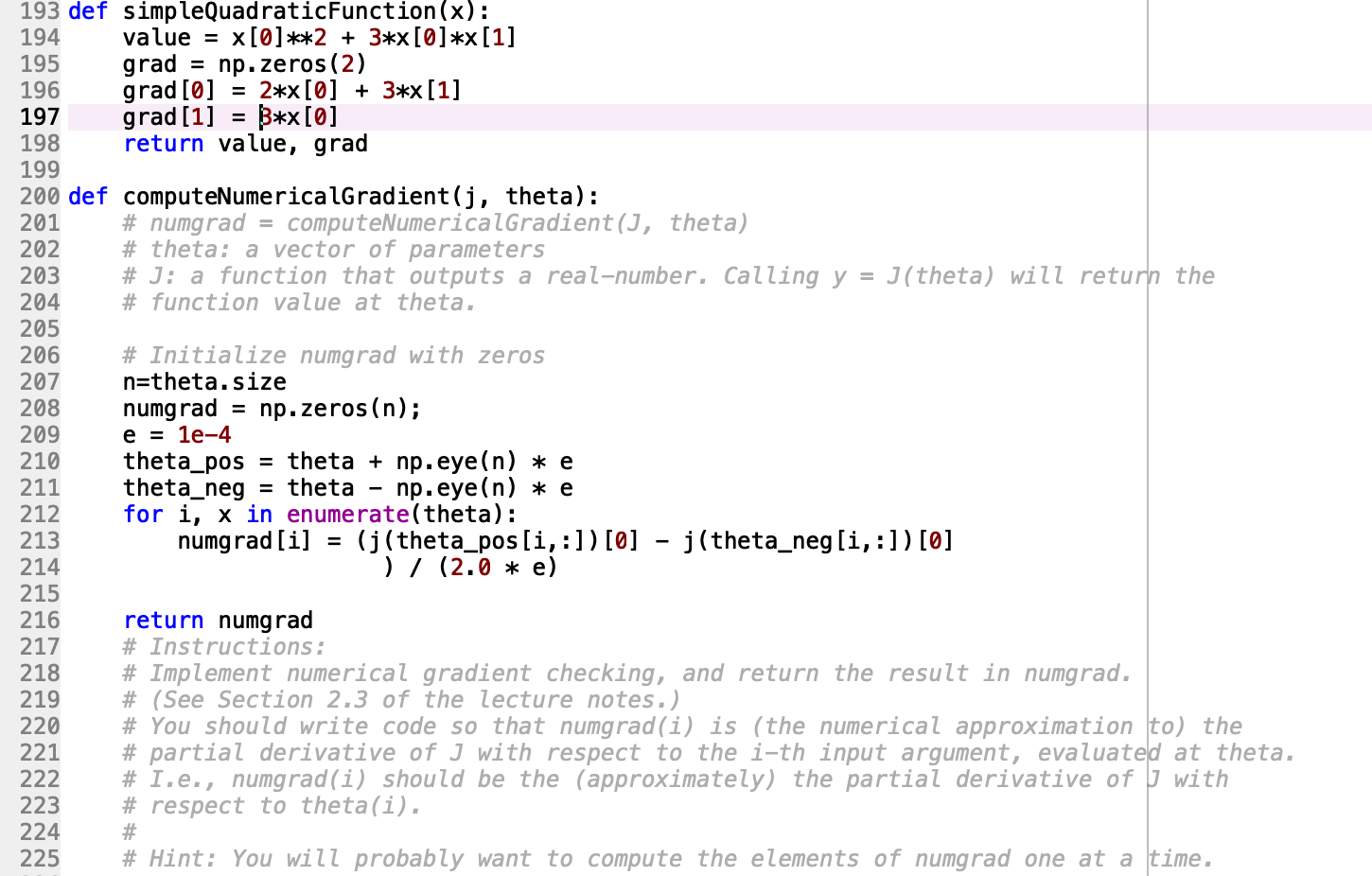


In **computeNumericalGradient(j,theta)** we will compute the gradient using the below formula.

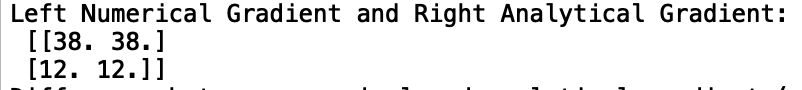


The function returns numgrad that is approximately the partial derivative of J with respect to theta.

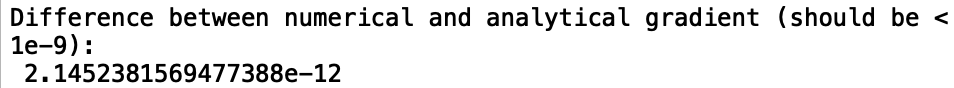
numgrad(i) is (the numerical approximation to) the partial derivative of J with respect to the i-th input argument, evaluated at theta.



To visualize the two gradient computation:-



The norm of the difference between two solutions is:

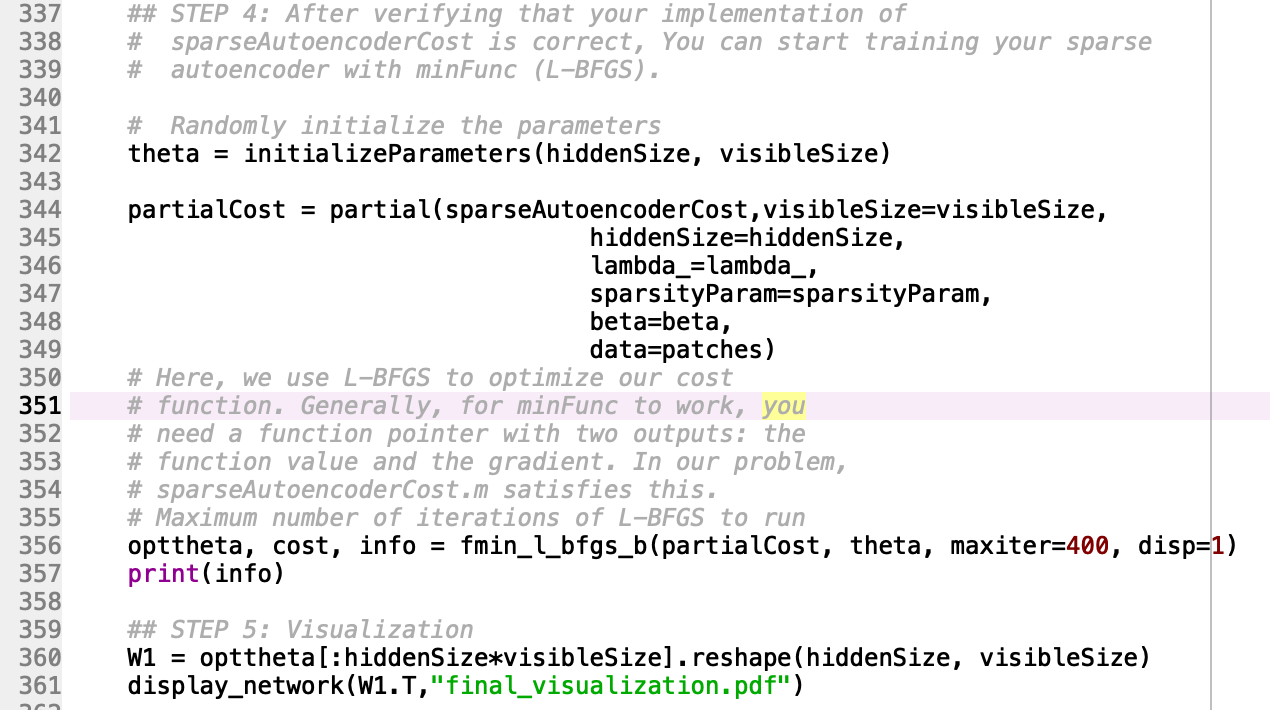




The argument j in computeNumericalGradient is a Partial function It allows us to fix a certain number of arguments of a function. Then, the  numerically computed gradients are compared with the updated theta.

**STEP 4: Training the sparse Autoencoder**

First will verify the implementation of *sparseAutoencoderCost* and then will train the sparse autoencoder with with L-BFGS algorithm. This algorithm minimizes the Jsparse(*W*,*b*)  w.r.t its parameters.



**STEP 5: Visualization**

After training the autoencoder, ***﻿display\_network(W1.T,"final\_visualization.pdf")*** is called to visualize the learned weights.

Generated plot:-

