PROGRAMMING RESIT ASSIGNMENT

Understanding Data

Submission Date : 07/09/2020
Due Date : 17/09/2020 23:59
TAs : Bahar Gezici-Necva Bölücü-Yunus Can Bilge

Accept your *Resit Assignment*.

Introduction:

In this experiment, you will implement a 2 Layer Neural Network to solve a real wold machine learning problem with a real world data. In this experiment, you will get familiar Python data science libraries such as; *Pandas*, *NumPy*, and *Matplotlib*. At the end, you will build a deep neural network that classifies *cat* vs. *non-cat* images.

Assignment:

Packages

- numpy is the fundamental package for scientific computing with Python.
- matplotlib is a library to plot graphs in Python.
- h5py is a common package to interact with a dataset that is stored on an H5 file.
- PIL and scipy are used here to test your model with your own picture at the end.
- **np.random.seed(1)** is used to keep all the random function calls consistent. It will help us grade your work.

In this assignment, you will build a 2 neural network to solve areal world problem.

Part 1 - Reading Data

You will use the same "Cat vs non-Cat" dataset. You are given a dataset ("data.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).

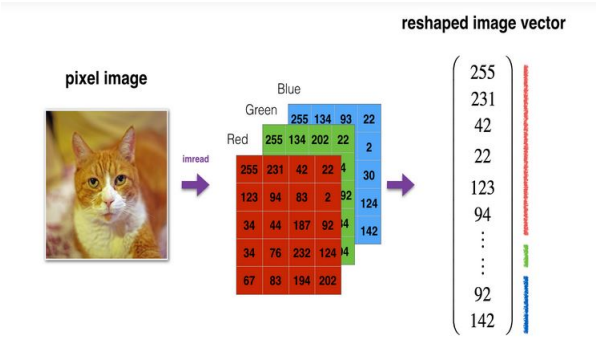

Figure 1: Image to vector conversion.

You will reshape and standardize the images before feeding them to the network.

Part 2 - Implementing a 2-layer Neural Network

Neural networks are simply modeled after the human neuron. Just as the brain is made up of numerous neurons, Neural Networks are made up of many activation units. In this assignment you will build a 2-layer neural network. The archştecture of the neural network is given in figure 2.

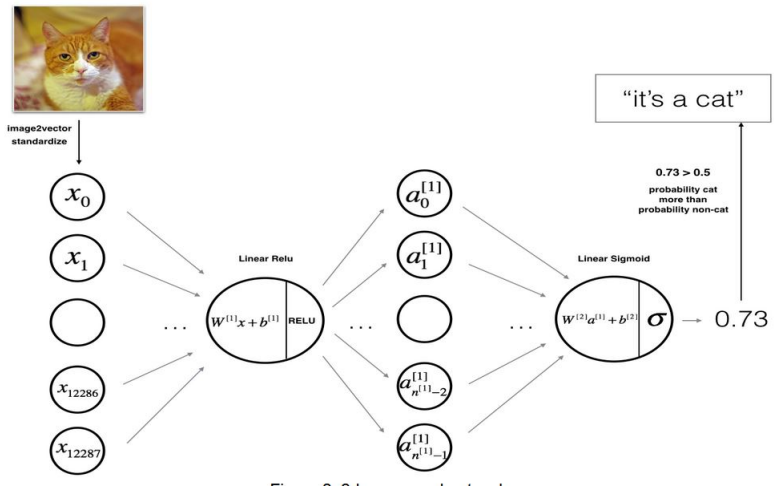

Figure 2: 2-layer neural network
The model can be summarized as: INPUT - LINEAR - RELU - LINEAR - SIGMOID - OUTPUT.

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: $[x_0, x_1, \cdots, x_{12287}]^T$ 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: $[a_0^{[1]}, a_1^{[1]}, \cdots, a_{n[1]-1}^{[1]}]^T$.

- You then repeat the same process.
- 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

To build the model you will follow the steps given below:

1. Initialize parameters / Define hyperparameters
2. Loop for num_iterations:
   (a) Forward propagation
   (b) Compute cost function
   (c) Backward propagation
   (d) Update parameters (using parameters, and grads from backprop)
3. Use trained parameters to predict labels

Forward propagation

After initialized your parameters, you will do the forward propagation module. You will apply LINEAR and ACTIVATION where ACTIVATION will be either ReLU or Sigmoid.

The forward propagation module computes the following equations:

$$Z^{[1]} = W^{[1]}X + b^{[1]} \quad (1)$$
$$A = RELU(Z^{[1]}) = max(0, Z^{[1]}) \quad (2)$$
$$Z^{[2]} = W^{[1]}A + b^{[2]} \quad (3)$$
$$AL = \sigma(Z^{[2]}) = \frac{1}{1 + e^{-Z^{[2]}}} \quad (4)$$

where X is the input, $Z^{[1]}$ is the output of the first LINEAR, A output of the Relu ACTIVATION, $Z^{[2]}$ is the output of the second LINEAR, and $\sigma(AL)$ is the output of the neural network.

The datils of the activation functions are given below::

Sigmoid: $\sigma(Z) = \sigma(WA + b) = \frac{1}{1+e^{-(WA+b)}}$.

ReLU: $A = RELU(Z) = max(0, Z)A = RELU(Z) = max(0, Z)$.

Cost function

You need to compute the cost function, because you want to check if your model is actually learning.

Compute the cross-entropy cost J, using the following formula:

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

, where m is the number of samples in your training set, y^i is the ith output and $AL^{(i)}$ is the output of the neural network for ith sample.

Backward propagation

You have already calculated the derivative $dZ^{[l]} = \frac{\partial \mathcal{L}}{\partial Z^{[l]}}$. You want to get $(dW^{[l]}, db^{[l]}, dA^{[l-1]})$.

The formulas you need: The three outputs $(dW^{[l]}, db^{[l]}, dA^{[l-1]})$ are computed using the input $dZ^{[l]}$.Here are the formulas you need:

$$dW^{[2]} = \frac{\partial \mathcal{J}}{\partial W^{[2]}} = \frac{1}{m} dZ^{[2]} A^T$$

$$db^{[2]} = \frac{\partial \mathcal{J}}{\partial b^{[2]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[2](i)}$$

$$dW^{[1]} = \frac{\partial \mathcal{J}}{\partial W^{[2]}} = \frac{1}{m} dZ^{[2]} X^T$$

$$db^{[1]} = \frac{\partial \mathcal{J}}{\partial b^{[1]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[1](i)}$$

Update Parameters

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

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

where α is the learning rate.

Plot Loss

In your training part, in each iteration you get cost value and append to costs array. You will plot loss figure with this array.

Calculate Accuracy

In test set you will use the linear_activation_forward function to calculate the accuracy. As mentioned above, if the output value is greater than 0.5 then your prediction is 1 and else is 0.

Accuracy will be calculated as; $accuracy = TruePositiveCount(TP)/TotalNumberofSamples$

TP = is an outcome that model predicts the true class correctly.

For instance; the predictions matrix and label matrixes are for 4 sample;

predictions matrix; $\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$ label matrix; $\begin{bmatrix}1\\0\\1\\0\end{bmatrix}$ TP count = 2 and Total Number of Samples = 4
Accuracy = 2/4 = 50%

You will fill the functions with the given parameters given in the below:

```
def initialize_parameters(n_x, n_h, n_y):
    ...
    return parameters

def linear_activation_forward(X, parameters):
    ...
    return AL

def compute_cost(AL, Y):
    ...
    return cost

def linear_activation_backward(X,cost):
    ...
    return grads

def update_parameters(parameters, grads, learning_rate):
    ...
    return parameters

def predict(test_x, test_y, parameters):
    ...
    return accuracy

def plot_loss(costs):
    ...

def two_layer_model(X, Y, layers_dims, learning_rate, num_iterations):
    ...
    return parameters,costs

def main():
    ...
```

Inputs to a unit is the features from training set. Weights $W^{[1]}$ and $W^{[2]}, b^{[1]}$ and $b^{[2]}$ are the parameters that you will learn. Activation functions (RELU and Sigmoid] modify the input data in order to learn complex relations between input and output data.

In Neural Networks, you forward propagate to predict output and compare it with the real value and calculate the loss. Loss value tells us how good our model make its predictions

which tells us how to change our prameters to make our model predict more accurate.

In order to minimize the loss, you propagate backwards with respect to each weight value by finding the derivative of the error. Thus, weights will be tuned. Neural networks are trained iteratively. After each iteration error is calculated via the difference between prediction and target.

You are expected to plot loss in each iteration.

Activation Functions

Sigmoid function:

$$h_\theta(x) = \frac{1}{1 + e^x} \tag{5}$$

The derivative of the Sigmoid function is;

$$\frac{\partial h_\theta(x)}{\partial x} = x * (1 - x) \tag{6}$$

Relu function:

$$h_\theta(x) = \begin{Bmatrix} x & x > 0 \\ 0 & x <= 0 \end{Bmatrix} \tag{7}$$

The derivative of the Relu function is;

$$\frac{\partial h_\theta(x)}{\partial x} == \begin{Bmatrix} 1 & x > 0 \\ 0 & x < 0 \end{Bmatrix} \tag{8}$$

Helper Links

- Transpose - `https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_14.html`
- Transpose - `https://docs.scipy.org/doc/numpy/reference/generated/numpy.transpose.html`
- Dot product - `https://www.mathsisfun.com/algebra/vectors-dot-product.html`
- Dot product - `https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html`

You will use the given code for the assignment. Please fill the functions given in the code.

Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

Important Notes

- Do not miss the submission deadline.
- Compile your code before submitting your work to make sure it compiles without any problems with *Python3.5*.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating. You can ask your questions via Piazza and you are supposed to be aware of everything discussed on Piazza. You cannot share algorithms or source code. All work must be individual! Assignments will be checked for similarity, and there will be serious consequences if plagiarism is detected.
- You must submit your work with the file hierarchy as stated below:

→ <makeup.ipynb>

Grading

Implementing a 2 layer Neural Network:
initialize_parameters (2 p)
linear_activation_forward (25 p)
compute_cost (15 p)
linear_activation_backward (25 p)
update_parameters (5 p)
predict (10 p)
plot_loss (5 p)
two_layer_model (10 p)
main (3 p)

References

1. `https://xkcd.com/1838/`
2. `https://towardsdatascience.com/train-test-split-and-cross-validation-in-py`
3. `https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975`
4. The Hundred-Page Machine Learning Book by Andriy Burkov, http://themlbook.com/wiki/doku.php
5. `https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html`