Deep Learning Lab #1 (Fall 2020)

Lab Objective:

In this assignment, you build a simple neural network (NN) with two hidden layers. This NN needs to have both forward pass and back-propagation functionality.

Rules:

1) This assignment should be done individually. Plagiarism is strictly prohibited.
2) You can **only** use Numpy and other Python standard library. Any deep-learning related frameworks (TensorFlow, PyTorch, etc.) are **not allowed** in this lab.
3) You should add comments throughout your implementation for easy understanding.
4) Write a report to detail your procedures and discussions, and convert your report into **.pdf** format.
5) You may copy sample code to your colab. Before submission, please make sure your colab file can be viewed by others (anyone with the link).
6) You are not required to train your model on colab, but you should make sure your code can run on colab without any error.

Submission:

Format: Please write your code on colab, and submit the followings:

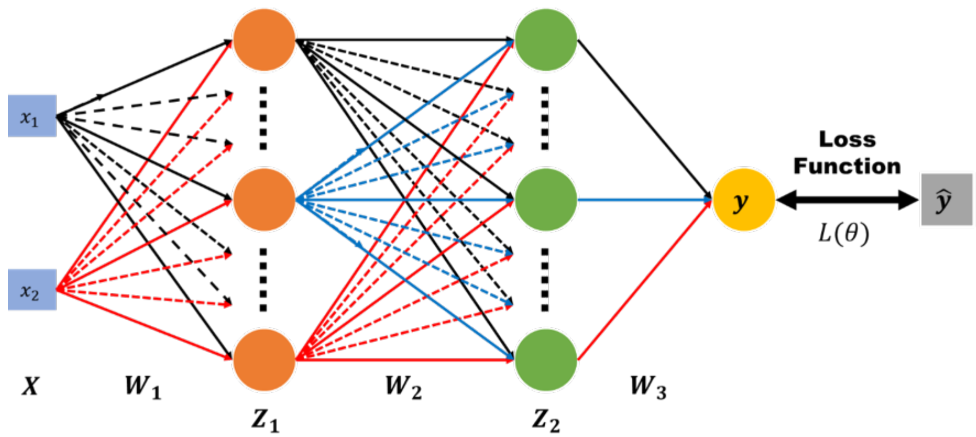
1) Submit the link of the file (please double check the accessibility: **anyone with the link** can **view** the file).
2) Download **.ipynb** from colab and submit the file.
   Please name your file as "Lab1_YourStudentID.ipynb" (ex: Lab1_309832000.ipynb)
3) Report

Deadline: 2020/11/09 (Mon.) 23:55
Late submission: 2020/11/12 (Thu.) 23:55

Requirements:

1) Implement a simple neural network with **two hidden layers (with 100 nodes in hidden layer 1 and 10 nodes in hidden layer 2)**. Use the 0.01 for learning rate.
2) You must use the back-propagation algorithm in this NN and build it from scratch. Only Numpy and other Python standard libraries are allowed.
3) Plot your comparison between ground truth and the predicted result.
4) The number of epochs is not restricted, but your model performance will be evaluated

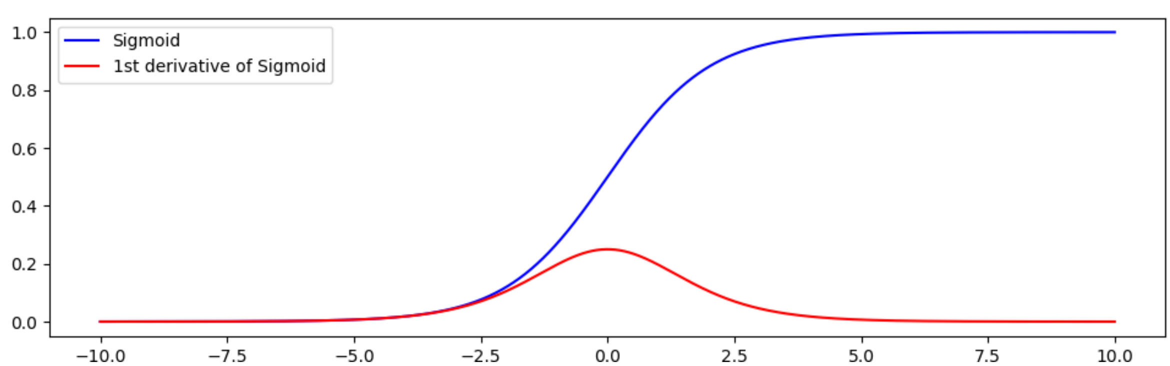
Descriptions:



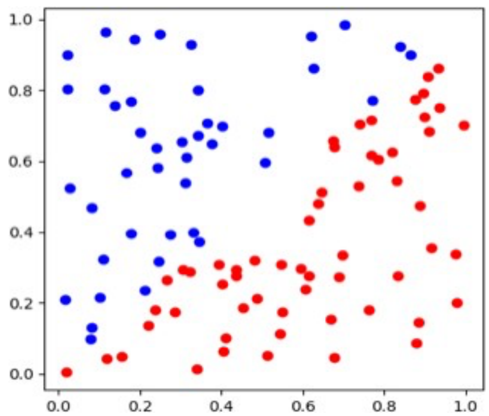
1) Notations:
    a. x_1, x_2: neural network inputs
    b. X : $[x_1, x_2]$
    c. y: neural network outputs
    d. $\hat{y}$: ground truth
    e. $L(\theta)$: loss
    f. w_1, w_2, w_3: weight matrix of each network layers

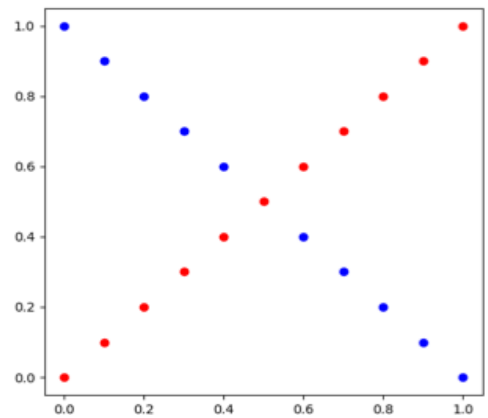
2) $Z_1 = \sigma(XW_1), Z_2 = \sigma(Z_1W_2), y = \sigma(Z_2W_3)$
   σ is a sigmoid function that refers to the special case of the logistic function and defined by the formula: $\sigma(x) = \frac{1}{1+e^{-x}}$

3) Sigmoid function:
   A sigmoid function is a mathematical function having a characteristic "S"-shaped curve. It is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point. In general, a sigmoid function is monotonic and has a first derivative that is bell-shaped.



4) Input data:
You should train your model on the following two types of data separately and also show their testing performance on the same data. You can also change the training data or test on data that is different from the training data to observe the behavior.





5) Back-propagation:
Back-propagation is an algorithm that is commonly used in neural networks to calculate gradients that are needed in the network weight update. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. The back-propagation algorithm can be divided into two parts; propagation and weight update.
   a. Part 1: Propagation, each propagation stage involve the following steps:
      i. Feed data into and propagate through the network to generate the output of each layer.
      ii. Compute the cost $L(\theta)$ (error term).
      iii. Propagate the output activations back through the network using the training target to generate Δ (the difference between the targeted and actual output values) of all hidden neurons and output layer.
   b. Part 2: Weight update, each weight update involve the following steps:
      i. Multiply its output Δ and input activation to get the gradient of the weight.
      ii. Subtract a percentage of the gradient from the weight.
      iii. This percentage influences the speed and quality of learning; it is called **learning rate (LR)**. The greater the LR, the faster the neuron trains; the lower the LR, the more accurate the training is. The sign of the gradient of weight indicates where the error is increasing, this is why the weight must be updated in the **opposite** direction.
   c. Training: Repeat part 1 & 2 until the performance of the network is satisfactory.

6) Pseudocode:

```
initialize network weights (often small random values)
do
   forEach training example named ex
      prediction = neural-net-output(network, ex)  // forward pass
      actual = teacher-output(ex)
      compute error (prediction - actual) at the output units
      compute Δw_h for all weights from hidden layer to output layer  // backward pass
      compute Δw_i for all weights from input layer to hidden layer   // backward pass continued
      update network weights // input layer not modified by error estimate
until all examples classified correctly or another stopping criterion satisfied
return the network
```

Report Specification:

1) Introduction (10%)
2) Experimental Setup (40%)
   a. Sigmoid functions
   b. Neural network (model structure and parameters settings)
   c. Back-propagation (explain the mathemetics implemented in your code)
3) Experimental Result (30%)
   a. Specify your training and results (how many epochs did you run? What is the final loss?)
   b. Figures:
      i. Training loss plots (epoch-loss)
      ii. Screenshots of predictions
      iii. Comparison figures (Ground Truth and Predicted)
   c. Anything else you would like to share
4) Discussion and extra experiments (20%)

Assignment Evaluation:

1) Code & model performances (60%)
2) Report (40%)

References:

http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html
http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML17_2.html

Please contact TA if you have any questions.