DL Lab4: forward pass & back-propagation

Lab Objective:

In this assignment, you need to build a simple neural network (NN) with 3 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. Once the T.A. finds plagiarism, they will receive a score of **0** on the assignment.

(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) If the assignment format and files are not in accordance with the regulations, the assignment score × 0.9.

(4) If the assignment is missing or incomplete training for any item, the assignment score will be deducted proportionally to the incompleteness.

(5) If you submit your assignment late, your score will be multiplied by 0.9 for each day of delay.

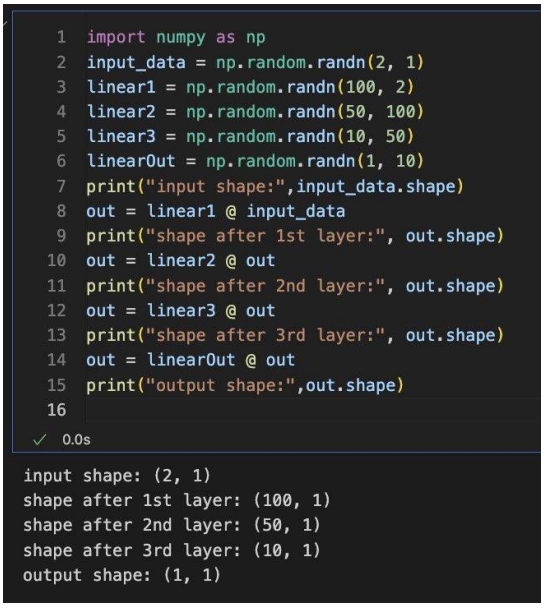
Submission:

(1) Please write your code on Jupyter notebook and the filename should be **A4_studentID_studentName.ipynb**.

(2) A Report in PDF format and filename should be **A4_studentID_studentName.pdf** . The report needs to include the any parts implemented by yourself, and comment your implementation for easy understanding, finally detail your procedures and discussions in the end of report, but it is strictly forbidden to post the entire code.

Requirements:

(1) Implement a simple neural network with three hidden layers **(with 100 nodes in hidden layer 1 , 50 nodes in hidden layer 2 and 10 nodes in hidden layer 3).**



```python
import numpy as np
input_data = np.random.randn(2, 1)
linear1 = np.random.randn(100, 2)
linear2 = np.random.randn(50, 100)
linear3 = np.random.randn(10, 50)
linearOut = np.random.randn(1, 10)
print("input shape:",input_data.shape)
out = linear1 @ input_data
print("shape after 1st layer:", out.shape)
out = linear2 @ out
print("shape after 2nd layer:", out.shape)
out = linear3 @ out
print("shape after 3rd layer:", out.shape)
out = linearOut @ out
print("output shape:",out.shape)

```

✓ 0.0s

```
input shape: (2, 1)
shape after 1st layer: (100, 1)
shape after 2nd layer: (50, 1)
shape after 3rd layer: (10, 1)
output shape: (1, 1)
```

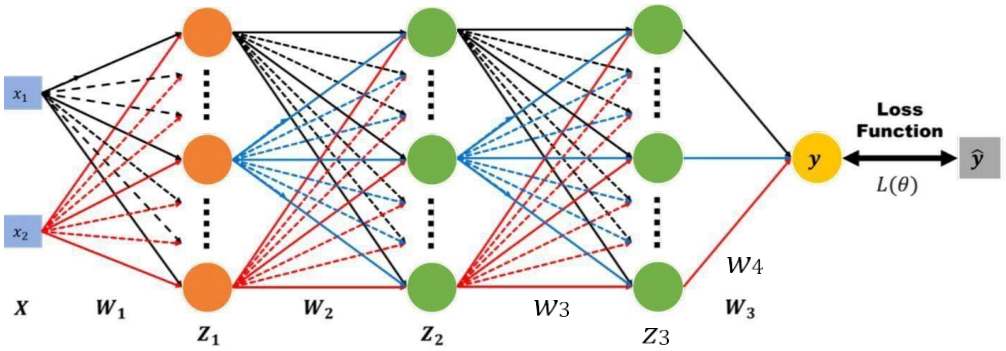
(2) **The number of data points generated should > 100, and > 1000 for chessboard.**

(3) You must use the back-propagation algorithm in this NN and build it from scratch. **Only Numpy and other Python standard libraries are allowed**.

(4) Plot your comparison between ground truth and the predicted result.

(5) The training epochs is not restricted, but model performance will be evaluated.

Descriptions:



(1) Notations

I. x_1, x_2: neural network inputs

II. X: $[x_1, x_2]$

III. y: neural network outputs

IV. $\hat{y}$: ground truth

V. $L(\theta)$: loss

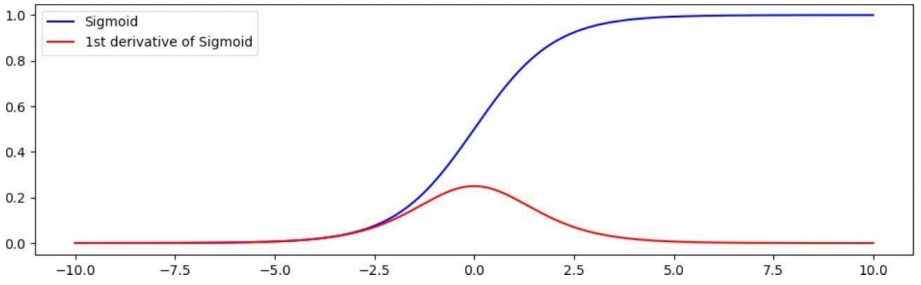
VI. w_1, w_2, w_3, w_4: weight matrix of each network layers

(2) $Z_1 = \sigma(XW_1), Z_2 = \sigma(Z_1W_2), Z_3 = \sigma(Z_2W_3), y = \sigma(Z_3W_4)$

σ is a sigmoid function that refers to the special case of the logistic function and defined by the formula: $\frac{1}{1+e^{-x}}$

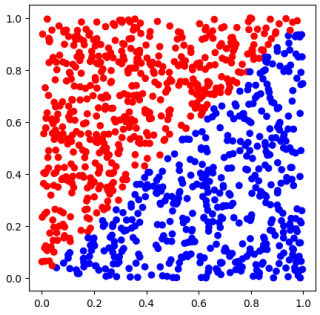
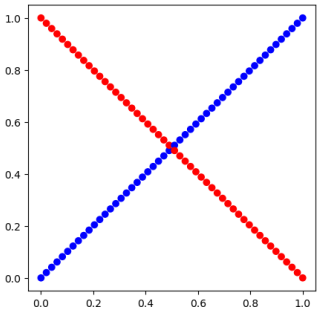
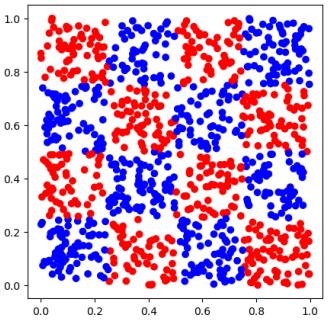
(3) Sigmoid function

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 3 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.

I. Part 1: Propagation, each propagation stage involved 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.

II. Part 2: Weight update, each weight update involved 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.

III. 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
```

Assignment Evaluation:

(1) Finished all requirements (80%)

(2) Report, Performance, Bonus, Experiments (20%+5%)

(3) Your score will be deducted if you break the rules.

Hints for higher performance (Bonus):

(1) Learning Rate Schedules and Adaptive Learning Rate Methods.

(2) More complex back propagation strategies.

Reference:

https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

https://reurl.cc/adA7XY

Deadline: 10/21 12:00 PM

---------- TA Contact information ----------

sk774325.ai12@nycu.edu.tw

andy90123.ai12@nycu.edu.tw