NYCU-DL-Lab1

資科工碩一廖洺玄

July 2025

1 Introduction

在這個 Lab 我們要手刻出一個二層 Layer 的神經網路並且不能用任何 DL Framework，主要的重點在 Backpropagation 的計算和 Model weight 更新，然後根據提供訓練時的 Loss、epoch 截圖以及測試後的 Iter、Ground truth、Prediction、Accuracy 截圖、畫出訓練結果的 Loss-epoch 曲線。整體的實作流程爲：

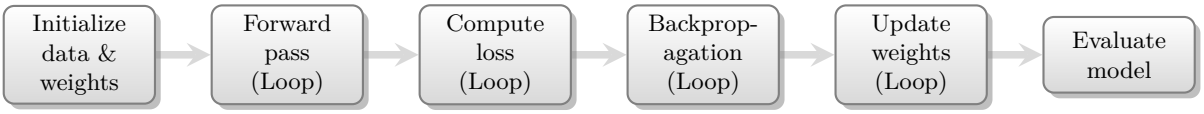

Figure 1: 神經網路實作流程圖

2 Implementation Details

2.1 Network Architecture

本次實驗用到的是一個二層 Layer 的神經網路模型，主要是由輸入層、第一隱藏層、第二隱藏層、輸出層組成，根據實驗要求輸入層的神經元數量爲 2，而輸出層的神經元數量爲 1，中間的第一和第二隱藏層我主要是設定爲 9 和 4，這部分在 Discussion 會調整來做出不同神經元數量差異來達到結報要求。而部分做圖的 code 是用 LLM 生成出來的其他皆爲自己參考公式後刻出來的。

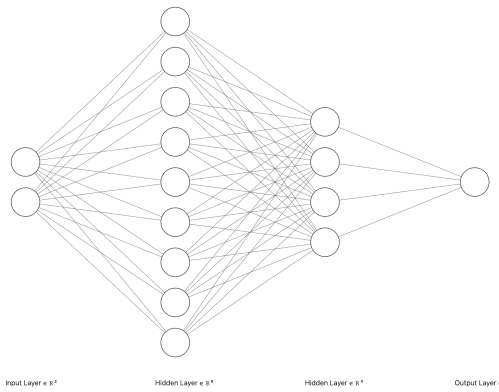

Figure 2: 神經網路架構和神經元數量

2.2 Activation Functions

目前主要用到的激活函數是 sigmoid 函數，他的優點是適合做 binary classification，因爲它能夠將輸出壓縮到 0～1 的範圍內。但它也有一些缺點，例如在輸入值非常大或非常小時，梯度會趨近於 0，造成梯度消失進而影響模型的學習效率。[1]

$$Sigmoid\ \ \sigma(x) = \frac{1}{1+e^{-x}}$$

2.3 Backpropagation

在這裡我們要用從 Forwardpass 那邊計算得到的預測值和原來的眞實值用 MSE 計算出他的損失值，接下來就可以一步步的用偏微分和連鎖率計算出 w1、w2、w3 的損失用做每個 epoch 的權重更新。在這邊比較重要的地方是要學會計算 MSE 公式、矩陣乘法、Sigmoid 公式的偏微分。[2]、[3]、[4]

$$W_1 \text{ 損失} = \frac{\partial \mathcal{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial A_1} \cdot \frac{\partial A_1}{\partial W_1}$$

$$W_1 \text{ 更新} = W_1 \text{ 原值} - \text{learning rate} \cdot W_1 \text{ 損失}$$

$$W_2 \text{ 損失} = \frac{\partial \mathcal{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_2} \cdot \frac{\partial A_2}{\partial W_2}$$

$$W_2 \text{ 更新} = W_2 \text{ 原值} - \text{learning rate} \cdot W_2 \text{ 損失}$$

$$W_3 \text{ 損失} = \frac{\partial \mathcal{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial A_3} \cdot \frac{\partial A_3}{\partial W_3}$$

$$W_3 \text{ 更新} = W_3 \text{ 原值} - \text{learning rate} \cdot W_3 \text{ 損失}$$

2.4 Extra Implementation

除了實驗提到的基本需求以外，我有把整個 NN 包成 class 和加上可以用 argv 來讓使用者直接調整 epoch、layer units、激活函數、optimizer、learning rate、data type。讓使用者在跑一次全部的神經網路 data generation、forwardpass、backwardpropagtion、model weight refresh、test model 可以更便利也提升 code 的可讀性。

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-lr", type=float, default=0.1, help="learning rate")
    parser.add_argument("-l1", type=int, default=9, help="number of units in layer 1")
    parser.add_argument("-l2", type=int, default=4, help="number of units in layer 2")
    parser.add_argument("-act", type=str, default="sigmoid", help="activation type(sigmoid, relu, tanh)")
    parser.add_argument("-opt", type=str, default="GD", help="optimizer type(GD, Monumnet)")
    parser.add_argument("-epoch", type=int, default=10000, help="number of epochs")
    parser.add_argument("-data", type=str, default="linear", help="data type(linear, xor)")
    args = parser.parse_args()
    if args.data == "linear":
        x, y = generate_linear(n=100)
```

Figure 3: Argument parser 讓使用者自行調整參數

```python
class SimpleTwoLayerModel:
    def __init__(self, l1=9, l2=4, learning_rate=0.1, activitation_type="sigmoid", optimizer_type   ="GD", epoch=10000):
        super(SimpleTwoLayerModel, self).__init__()
        self.w1 = np.random.rand(2, l1)
        self.w2 = np.random.rand(l1, l2)
        self.w3 = np.random.rand(l2, 1)
        self.learning_rate = learning_rate
        self.loss_list = []
        self.activitation_type = activitation_type
        self.optimizer_type = optimizer_type
        self.epoch = epoch
        self.y_pred_binary = None

    def train(self, x, y):
        for i in range(self.epoch):
            # forward pass
            # layer 1
            a1 = x@self.w1
            z1 = activitate(self.activitation_type, a1)

            # layer 2
            a2 = z1@self.w2
            z2 = activitate(self.activitation_type, a2)

            # output
            a3 = z2@self.w3
            y_train_pred = sigmoid(a3)
```

Figure 4: 整個流程封裝成 class 提升 code 可讀性

3 Experimental Results

3.1 Screenshot and comparison figure

這兩組圖片 show 出兩種不同資料在 training 後都有很好的結果。兩個模型最後都達到 **100% 的準確率**，這表示它們能夠完美地區別出所有的是非題的答案。兩者的最終損失值也都非常低，分別爲 **0.00072** 和 **0.00068**，這顯示模型對其判斷幾乎沒有偏差。而且從兩

個不同的資料結果都可以看到 grund truth 和 prediction 都 divide 出非常相似的兩種不同 class。

```
Epoch: 0, Loss: 0.3746922644298039
Epoch: 500, Loss: 0.02298729887447683
Epoch: 1000, Loss: 0.011856867223713218
Epoch: 1500, Loss: 0.008972476128705621
Epoch: 2000, Loss: 0.0077399943017921104
Epoch: 2500, Loss: 0.007311245849416853
Epoch: 3000, Loss: 0.007052202596605563
Epoch: 3500, Loss: 0.006639234766365286
Epoch: 4000, Loss: 0.0057347529412260685
Epoch: 4500, Loss: 0.005468520023331854
Epoch: 5000, Loss: 0.005111388835584442
Epoch: 5500, Loss: 0.004693849144671763
Epoch: 6000, Loss: 0.004219876917834214
Epoch: 6500, Loss: 0.00335955768698582
Epoch: 7000, Loss: 0.0008286485745936495
Epoch: 7500, Loss: 0.0007106730456676391
Epoch: 8000, Loss: 0.0006185299405214265
Epoch: 8500, Loss: 0.0005440979505134883
Epoch: 9000, Loss: 0.00048288296806643546
Epoch: 9500, Loss: 0.0004318728582871376
```

Figure 5: 線性模型訓練過程

```
Iter0    | Ground truth: 1.0 | prediction: 0.99999
Iter1    | Ground truth: 1.0 | prediction: 0.99998
Iter2    | Ground truth: 0.0 | prediction: 0.00001
Iter3    | Ground truth: 1.0 | prediction: 0.99999
Iter4    | Ground truth: 0.0 | prediction: 0.00000
Iter5    | Ground truth: 1.0 | prediction: 0.99999
Iter6    | Ground truth: 0.0 | prediction: 0.00452
Iter7    | Ground truth: 0.0 | prediction: 0.00000
Iter8    | Ground truth: 1.0 | prediction: 0.99999
Iter9    | Ground truth: 0.0 | prediction: 0.00000
Iter10   | Ground truth: 1.0 | prediction: 0.99999
Iter11   | Ground truth: 0.0 | prediction: 0.00000
Iter12   | Ground truth: 1.0 | prediction: 0.99999
Iter13   | Ground truth: 1.0 | prediction: 0.99999
Iter14   | Ground truth: 0.0 | prediction: 0.00000
Iter15   | Ground truth: 0.0 | prediction: 0.00000
Iter16   | Ground truth: 0.0 | prediction: 0.00000
Iter17   | Ground truth: 1.0 | prediction: 0.99997
Iter18   | Ground truth: 1.0 | prediction: 0.99999
Iter19   | Ground truth: 1.0 | prediction: 0.99999
Iter20   | Ground truth: 0.0 | prediction: 0.00000
Iter21   | Ground truth: 1.0 | prediction: 0.99998
Iter22   | Ground truth: 1.0 | prediction: 0.99999
Iter23   | Ground truth: 0.0 | prediction: 0.00000
Iter24   | Ground truth: 0.0 | prediction: 0.00000
Iter25   | Ground truth: 1.0 | prediction: 0.99999
Iter26   | Ground truth: 1.0 | prediction: 0.99996
Iter27   | Ground truth: 1.0 | prediction: 0.99999
Iter28   | Ground truth: 0.0 | prediction: 0.00000
Iter29   | Ground truth: 1.0 | prediction: 0.99999
Iter30   | Ground truth: 1.0 | prediction: 0.99997
Iter31   | Ground truth: 0.0 | prediction: 0.00000
Iter32   | Ground truth: 0.0 | prediction: 0.00002
Iter33   | Ground truth: 0.0 | prediction: 0.00000
Iter34   | Ground truth: 0.0 | prediction: 0.00606
Iter35   | Ground truth: 0.0 | prediction: 0.00001
Iter36   | Ground truth: 1.0 | prediction: 0.99999
Iter37   | Ground truth: 1.0 | prediction: 0.99999
Iter38   | Ground truth: 0.0 | prediction: 0.00000
```

Figure 6: 線性模型測試結果圖 1

```
Iter39   | Ground truth: 1.0 | prediction: 0.99999
Iter40   | Ground truth: 0.0 | prediction: 0.00000
Iter41   | Ground truth: 1.0 | prediction: 0.99999
Iter42   | Ground truth: 0.0 | prediction: 0.00000
Iter43   | Ground truth: 1.0 | prediction: 0.99724
Iter44   | Ground truth: 1.0 | prediction: 0.99999
Iter45   | Ground truth: 0.0 | prediction: 0.00000
Iter46   | Ground truth: 0.0 | prediction: 0.00001
Iter47   | Ground truth: 1.0 | prediction: 0.99999
Iter48   | Ground truth: 0.0 | prediction: 0.00000
Iter49   | Ground truth: 0.0 | prediction: 0.00000
Iter50   | Ground truth: 0.0 | prediction: 0.00408
Iter51   | Ground truth: 1.0 | prediction: 0.99999
Iter52   | Ground truth: 1.0 | prediction: 0.99995
Iter53   | Ground truth: 0.0 | prediction: 0.00000
Iter54   | Ground truth: 1.0 | prediction: 0.99999
Iter55   | Ground truth: 1.0 | prediction: 0.99999
Iter56   | Ground truth: 1.0 | prediction: 0.99999
Iter57   | Ground truth: 0.0 | prediction: 0.00000
Iter58   | Ground truth: 0.0 | prediction: 0.00000
Iter59   | Ground truth: 1.0 | prediction: 0.99999
Iter60   | Ground truth: 1.0 | prediction: 0.99999
Iter61   | Ground truth: 1.0 | prediction: 0.99999
Iter62   | Ground truth: 1.0 | prediction: 0.99998
Iter63   | Ground truth: 1.0 | prediction: 0.99999
Iter64   | Ground truth: 1.0 | prediction: 0.99999
Iter65   | Ground truth: 1.0 | prediction: 0.99999
Iter66   | Ground truth: 0.0 | prediction: 0.00000
Iter67   | Ground truth: 1.0 | prediction: 0.99999
Iter68   | Ground truth: 1.0 | prediction: 0.99999
Iter69   | Ground truth: 0.0 | prediction: 0.00000
Iter70   | Ground truth: 1.0 | prediction: 0.99999
Iter71   | Ground truth: 0.0 | prediction: 0.00000
Iter72   | Ground truth: 1.0 | prediction: 0.99976
Iter73   | Ground truth: 1.0 | prediction: 0.99999
Iter74   | Ground truth: 1.0 | prediction: 0.99999
Iter75   | Ground truth: 1.0 | prediction: 0.99999
Iter76   | Ground truth: 0.0 | prediction: 0.26788
Iter77   | Ground truth: 0.0 | prediction: 0.00000
```

Figure 7: 線性模型測試結果圖 2

```
Iter68   | Ground truth: 1.0 | prediction: 0.99999
Iter69   | Ground truth: 0.0 | prediction: 0.00000
Iter70   | Ground truth: 1.0 | prediction: 0.99999
Iter71   | Ground truth: 0.0 | prediction: 0.00000
Iter72   | Ground truth: 1.0 | prediction: 0.99976
Iter73   | Ground truth: 1.0 | prediction: 0.99999
Iter74   | Ground truth: 1.0 | prediction: 0.99999
Iter75   | Ground truth: 1.0 | prediction: 0.99999
Iter76   | Ground truth: 0.0 | prediction: 0.26788
Iter77   | Ground truth: 0.0 | prediction: 0.00000
Iter78   | Ground truth: 0.0 | prediction: 0.00000
Iter79   | Ground truth: 0.0 | prediction: 0.00000
Iter80   | Ground truth: 1.0 | prediction: 0.99996
Iter81   | Ground truth: 1.0 | prediction: 0.99999
Iter82   | Ground truth: 1.0 | prediction: 0.99998
Iter83   | Ground truth: 1.0 | prediction: 0.99999
Iter84   | Ground truth: 1.0 | prediction: 0.99999
Iter85   | Ground truth: 1.0 | prediction: 0.99999
Iter86   | Ground truth: 0.0 | prediction: 0.00000
Iter87   | Ground truth: 1.0 | prediction: 0.99999
Iter88   | Ground truth: 1.0 | prediction: 0.99998
Iter89   | Ground truth: 1.0 | prediction: 0.99999
Iter90   | Ground truth: 0.0 | prediction: 0.00000
Iter91   | Ground truth: 1.0 | prediction: 0.99993
Iter92   | Ground truth: 0.0 | prediction: 0.00000
Iter93   | Ground truth: 1.0 | prediction: 0.99999
Iter94   | Ground truth: 1.0 | prediction: 0.99999
Iter95   | Ground truth: 1.0 | prediction: 0.99999
Iter96   | Ground truth: 0.0 | prediction: 0.00000
Iter97   | Ground truth: 0.0 | prediction: 0.00000
Iter98   | Ground truth: 0.0 | prediction: 0.00000
Iter99   | Ground truth: 1.0 | prediction: 0.99999
-------------------------------------------------
loss=0.00072 accuracy=100.00%
-------------------------------------------------
```

Figure 8: 線性模型測試結果圖 3 和精準度

```
epoch: 0, loss: 0.4694334061437931
epoch: 500, loss: 0.24933650171006758
epoch: 1000, loss: 0.24925089931080086
epoch: 1500, loss: 0.24911319511059146
epoch: 2000, loss: 0.2488630767837527
epoch: 2500, loss: 0.2483283011326979
epoch: 3000, loss: 0.24680743681710446
epoch: 3500, loss: 0.23941511553589656
epoch: 4000, loss: 0.1957215338265419
epoch: 4500, loss: 0.16919493732987062
epoch: 5000, loss: 0.048313063694003816
epoch: 5500, loss: 0.026962865950848854
epoch: 6000, loss: 0.01223361893527022
epoch: 6500, loss: 0.00626990618934876
epoch: 7000, loss: 0.0036470497590428466
epoch: 7500, loss: 0.002379132173721466
epoch: 8000, loss: 0.0016886413273063854
epoch: 8500, loss: 0.0012733987400041602
epoch: 9000, loss: 0.0010037785527524025
epoch: 9500, loss: 0.000818056620769
```

Figure 9: Xor 模型訓練過程

```
Iter0    | Ground truth: 0.0 | prediction: 0.00004
Iter1    | Ground truth: 1.0 | prediction: 0.98638
Iter2    | Ground truth: 0.0 | prediction: 0.00025
Iter3    | Ground truth: 1.0 | prediction: 0.99752
Iter4    | Ground truth: 0.0 | prediction: 0.00271
Iter5    | Ground truth: 1.0 | prediction: 0.99961
Iter6    | Ground truth: 0.0 | prediction: 0.01868
Iter7    | Ground truth: 1.0 | prediction: 0.99981
Iter8    | Ground truth: 0.0 | prediction: 0.04858
Iter9    | Ground truth: 1.0 | prediction: 0.94909
Iter10   | Ground truth: 0.0 | prediction: 0.05404
Iter11   | Ground truth: 0.0 | prediction: 0.03491
Iter12   | Ground truth: 1.0 | prediction: 0.93384
Iter13   | Ground truth: 0.0 | prediction: 0.01686
Iter14   | Ground truth: 1.0 | prediction: 0.99991
Iter15   | Ground truth: 0.0 | prediction: 0.00731
Iter16   | Ground truth: 1.0 | prediction: 1.00000
Iter17   | Ground truth: 0.0 | prediction: 0.00319
Iter18   | Ground truth: 1.0 | prediction: 1.00000
Iter19   | Ground truth: 0.0 | prediction: 0.00148
Iter20   | Ground truth: 1.0 | prediction: 1.00000
--------------------------------------------------
loss=0.00068 accuracy=100.00%
--------------------------------------------------
```

Figure 10: XOR 模型測試結果圖和精準度

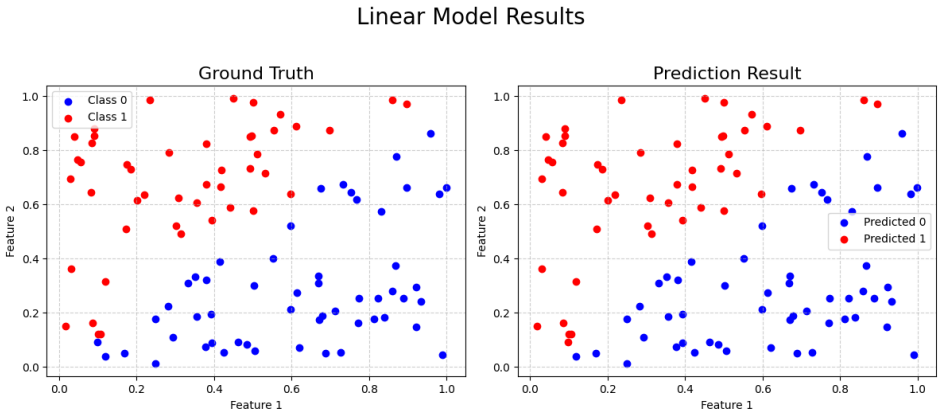

Figure 11: Xor 模型訓練過程

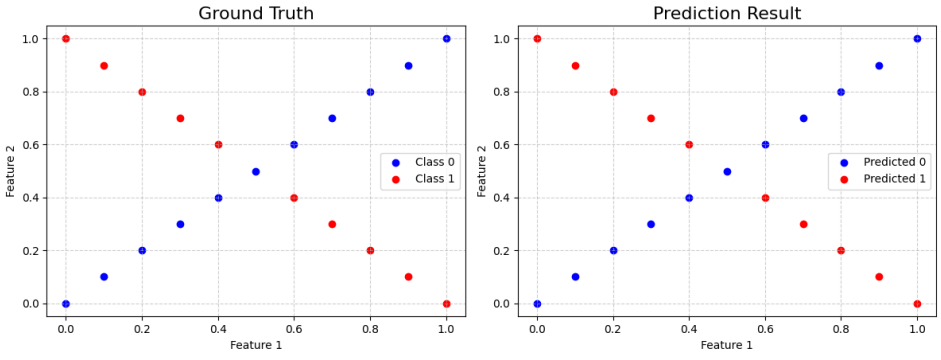

Figure 12: Xor 眞實資料及模型預測散步圖

3.2 Show the accuracy of your prediction

在模型精確度這部分，我訓練完的線性模型（Figure 8）與 XOR 模型（Figure 10）皆達到 100% 的準確率。

3.3 Learning curve (loss-epoch curve)

這邊我們激活函數用 Sigmoid 函數 train 出來的 model 在線性和 XOR 情況下可以看到，線性模型的曲線下降的趨勢在 1000 以前下降的非常快，而在 1000 之後他就開始下降的非常緩慢並且斜率也變得很平坦，在 2000 之後基本上已經趨近於 0。但是在 XOR 模型的情況可以看到 loss 在 epoch 4000 之前先直線下降並待在 0.25 左右，之後經過一個小震盪才在 epoch 4500 左右開始往下降到趨近於 0。

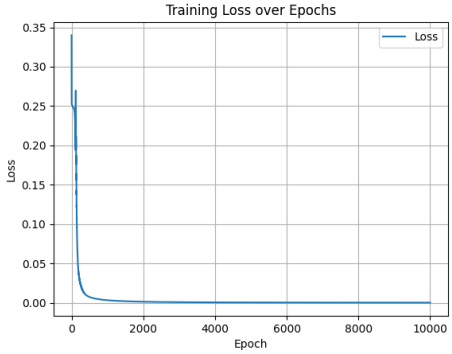

Figure 13: 線性模型損失-epoch 曲線

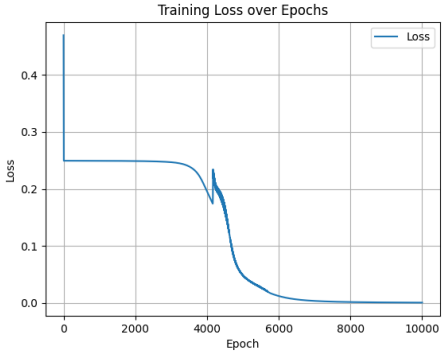

Figure 14: XOR 模型損失-epoch 曲線

3.4 Anything you want to present

除了實驗提到的基本需求以外

4 Discussions

4.1 Try different learning rates

learning rate（以下簡稱 LR 值），從這兩個模型可以看到兩者的情況非常相似。在我從 0.1 調整到 0.3 時，這兩個模型的損失-epoch 曲線的梯度變化都變平滑了。這讓我原本以爲 LR 值越高越好，但我直接調到一個較大的值 0.9 之後，發現損失-epoch 曲線的梯度變化在中間段開始出現很多的大幅震盪。所以 LR 值的調整對模型收斂速度與穩定性有明顯的影響，但越大不代表越好反而會造成模型訓練的不穩定。[5]

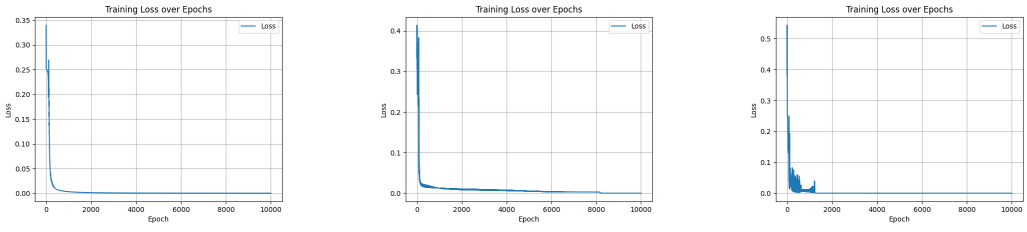

Figure 15: 線性模型損失-epoch 曲線從左至右 LR 值分別爲：0.1(預設值)、0.3、0.9

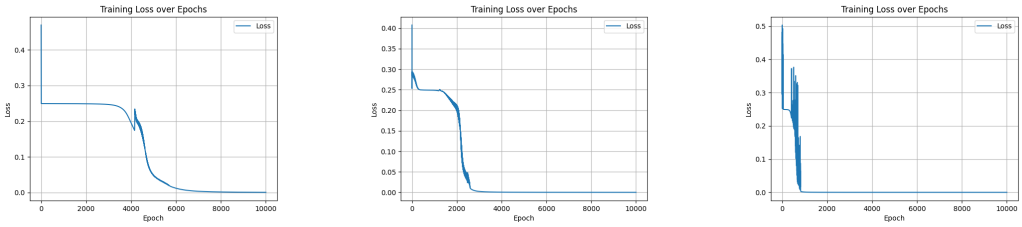

Figure 16: XOR 模型損失-epoch 曲線從左至右 LR 值分別爲：0.1(預設值)、0.3、0.9

4.2 Try different numbers of hidden units

Numbers of hidden units(以下簡稱 HU 值)，我們可以看到這兩個模型的變化在相同的 HU 值也有類似的情況。兩模型在 HU 值爲 2x1 的時候的損失-epoch 曲線的梯度表現最平滑且沒有任何震盪，但當 HU 值上升了一點到報告預設值 9x4 的時候就開始在中間段有一些小幅震盪，值到上升到 20x14 的時候在訓練過程中都出現的非常大幅度的變化。這顯示了 HU 值也不是越大越好，我觀察到因爲我們的生成出的資料因爲輸入特徵點只有兩個，所以在 HU 值 2x1 最接近他判斷的特徵數量才有較好的表現。而太多的神經元在訓練模型的時候就會容易造成老師上課說到的 overfitting。

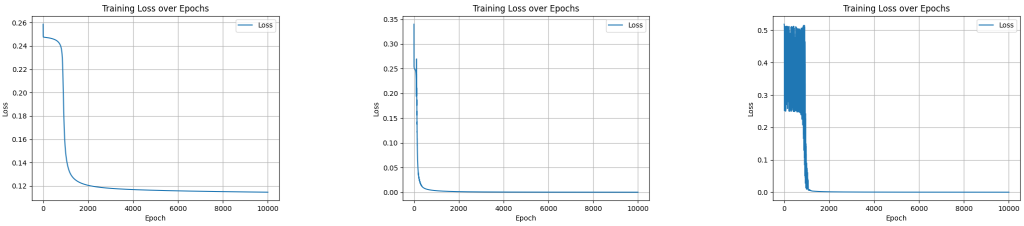

Figure 17: 線性模型損失-epoch 曲線從左至右 HU 值分別爲：2x1、9x4(預設值)、20x14

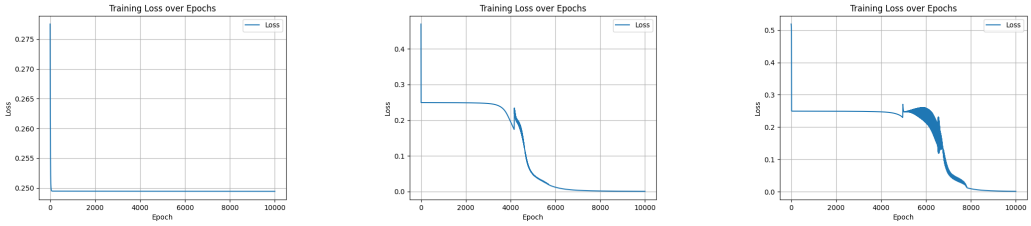

Figure 18: XOR 模型損失-epoch 曲線從左至右 HU 值分別爲：2x1、9x4(預設值)、20x14

4.3 Try without activation functions

在移除所有 activation function 後，我觀察到在多層神經網路中，forward pass 產生的數值在多次矩陣乘法後快速放大，導致指數計算時出現 overflow，最終 loss 變爲 NaN，模型失效 (如圖4.3所示)。從這裡我發現 activation function 除了提供非線性能力外，也在數值穩定性上扮演關鍵角色。沒有它模型容易陷入數值爆炸完全無法訓練，二者最終 accuracy 僅有 50

```
❯ python3 main.py
Epoch: 0, Loss: 30.34835008493112
/Users/minghsuan/Documents/NYCU-Deep-Learning/Lab1/main.py:49: RuntimeWarning: overflow encountered in square
  return np.mean((y-y_hat)**2)
/Users/minghsuan/Documents/NYCU-Deep-Learning/Lab1/main.py:69: RuntimeWarning: overflow encountered in matmul
  pre_sigmoid_layer_two = z1@w2_linear
/Users/minghsuan/Documents/NYCU-Deep-Learning/Lab1/main.py:112: RuntimeWarning: invalid value encountered in subtract
  w3_linear = w3_linear - learning_rate*backpro_output_l2
/Users/minghsuan/Documents/NYCU-Deep-Learning/Lab1/main.py:113: RuntimeWarning: invalid value encountered in subtract
  w2_linear = w2_linear - learning_rate*backpro_output_l1
/Users/minghsuan/Documents/NYCU-Deep-Learning/Lab1/main.py:114: RuntimeWarning: invalid value encountered in subtract
  w1_linear = w1_linear - learning_rate*backpro_output_input
Epoch: 500, Loss: nan
Epoch: 1000, Loss: nan
Epoch: 1500, Loss: nan
Epoch: 2000, Loss: nan
Epoch: 2500, Loss: nan
```

Figure 19: 無啓用函數情況下的損失-epoch 曲線：由於數值不穩定 loss 很快發散爲 NaN

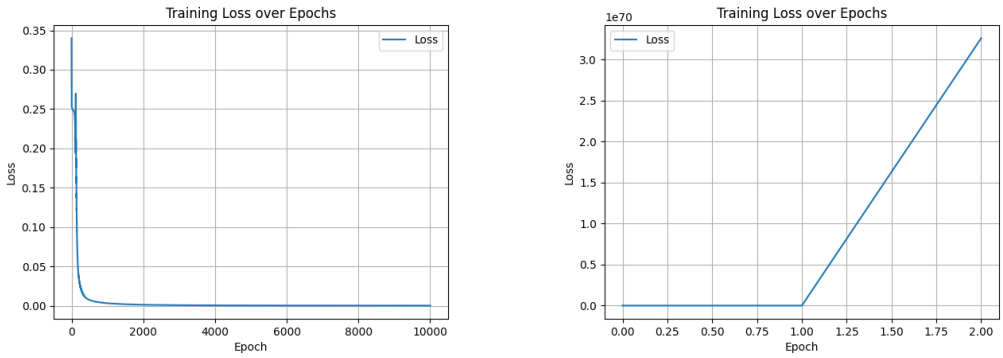

Figure 20: 線性模型損失-epoch 曲線從左至右分別爲：有激勵函數（預設值）、無激勵函數

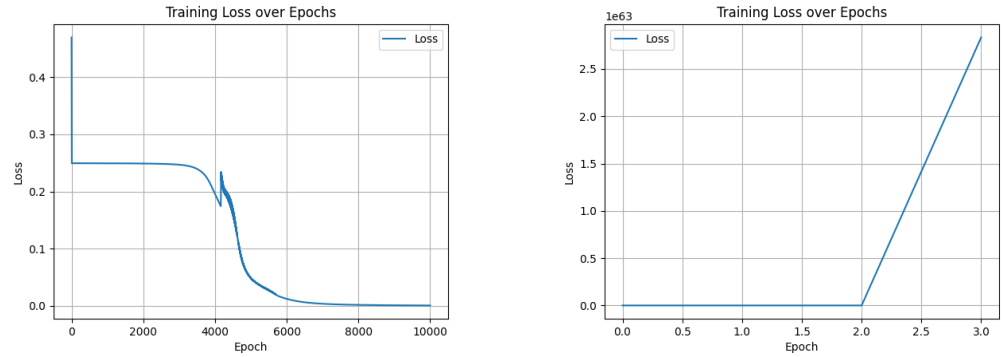

Figure 21: XOR 模型損失-epoch 曲線從左至右分別爲：有激勵函數（預設值）、無激勵函數

4.4 Extra Implementation Discussions

5 Questions

5.1 What are the purposes of activation functions?

激活函數的目的在於把模型套上非線性，使 NN 能夠學習和表示複雜的非線性關係。如果沒有激活函數，無論加多少層，整個網路都只是線性的組合 (因爲都只有線性的矩陣乘法)，無法處理像圖像辨識或語音識別這種需要非線性特徵的問題。

5.2 What if the learning rate is too large or too small?

Learning rate 太小或太大對模型的學習都不好。如果 RL 值太大的話，模型在學習的時候容易跳過某些關鍵的特徵，進而讓預測能力沒那麼好，會間接讓模型有 underfitting 的可能，這就是爲什麼我在上面的 loss-epoch curve 結果圖片 (如圖4.1) 會出現大幅震盪。但 RL 值太小也不行，因爲會造成模型學習效率低下，學得太慢也會導致 Underfitting。

5.3 What are the purposes of weights and biases in a neural network?

從老師上課的內容中學到 weight 決定輸入特徵對結果的影響程度，而 bias 則用來調整 estimator，使模型有更好的擬合能力。model weight 控制資料如何傳遞和放大，bias 則幫助模型即使在輸入爲零時也能產生有效輸出。[6]

6 Bonus

6.1 Optimizers

我在本次實驗新添加了一個 Optimizer（Momentum），從實驗結果可以看到它讓模型在訓練時減少震盪，因爲它會保留前一次的更新方向，像滾球一樣「有慣性」地朝著正確方向前進。這樣可以避免在陡峭的方向反覆跳動，導致損失-Epoch 曲線上下震盪，從而讓損失-Epoch 看起來更平滑。可以看到在線性模型的情況下非常明顯，損失-Epoch 曲線的變化變得相當平緩。而 XOR 也是，他把震盪完全抵銷掉了。因爲我原本的實驗結果有一些震盪，所以我就找了一個可以消除震盪的 Optimizer。

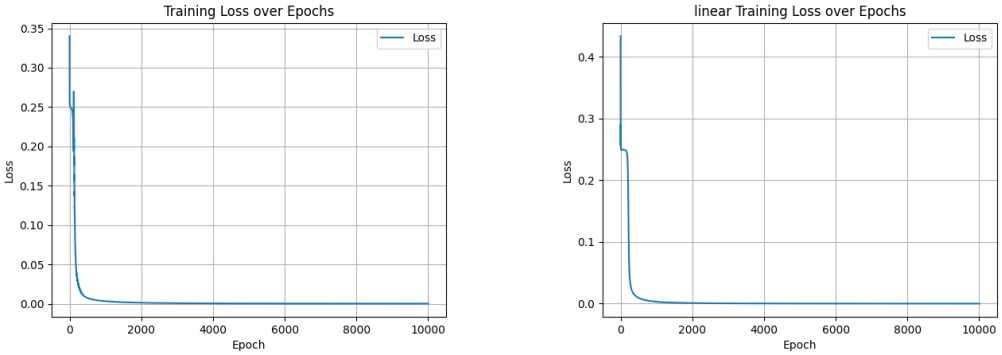

Figure 22: 線性模型損失-epoch 曲線從左至右 Optimizer 分別爲：GD(預設值)、Momentum

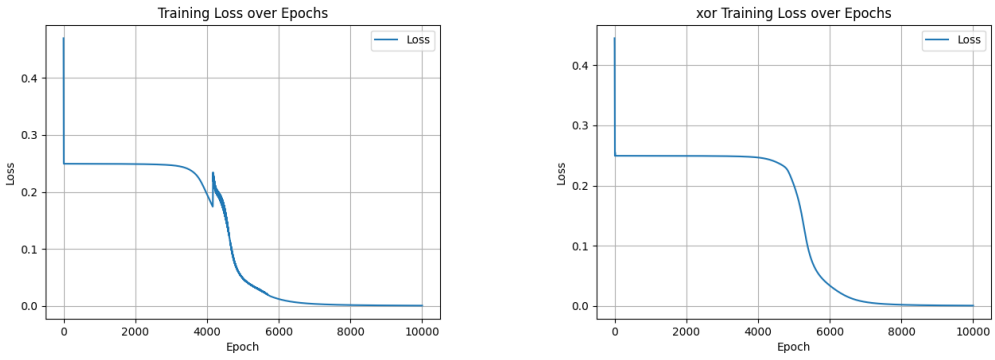

Figure 23: XOR 模型損失-epoch 曲線從左至右 Optimizer 分別爲：GD(預設值)、Momentum

6.2 Activation functions

我在 Bonus 分別作了其他兩種不同的激活函數，從一下兩個圖可以到線性模型在 sigmoid 和 tanh 種激活函數下都能快速收斂到接近零的損失值，其中 Sigmoid，他的變化更平滑，因爲教授上課有說他適合在處理這種簡單二元分類的問題。而 XOR 模型在 Sigmoid 激活函數雖然剛開始有一些奇怪的震盪但最終 loss 可以達到接近 0，relu 和 tanh 在 XOR 問題上表現較差，因爲 relu 會將負值變成零限制了模型的預測能力。我會做這三個的原因是因爲助教給的 slide 裡都有提到。

$$ReLU\text{函數：}x^{+} = max(0, x)$$

$$tanh\text{函數：}\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

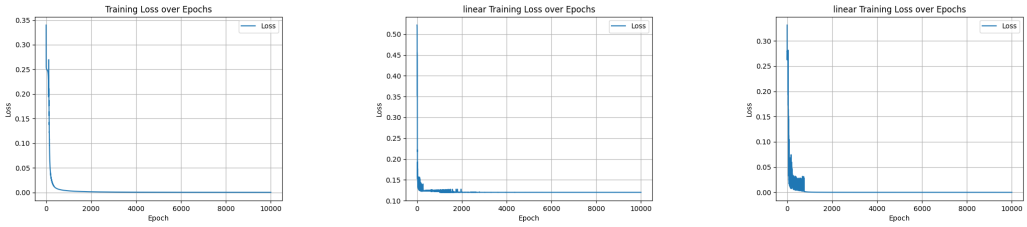

Figure 24: 線性模型損失-epoch 曲線從左至右激活函數分別爲：sigmoid(預設值)、relu、tanh

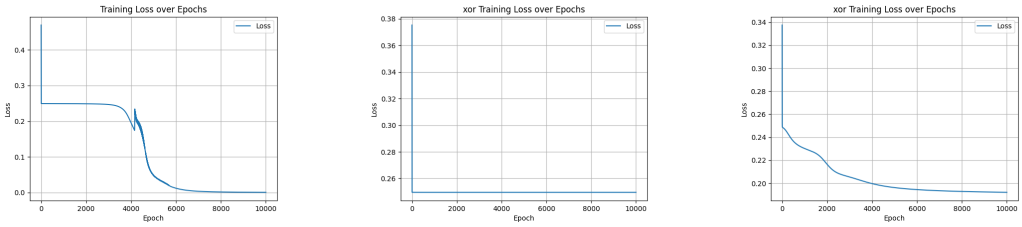

Figure 25: XOR 模型損失-epoch 曲線從左至右激活函數分別爲：sigmoid(預設值)、relu、tanh

References

[1] E. Waite, "Why Do We Use the Sigmoid Function for Binary Classification?," YouTube video, [上傳日期, e.g., Oct. 20, 2023]. Available: https://www.youtube.com/watch?v=WsFasV46KgQ. Accessed: [實際訪問日期, e.g., Jul. 7, 2024].

[2] 台大資訊人工智慧導論, "FAI 4.2: Backpropagation 參數這麼多到底怎麼有效率算出來的呢?," YouTube video, [上傳日期, e.g., Feb. 15, 2022]. Available: https://www.youtube.com/watch?v=PdGgbsM9vrc. Accessed: [實際訪問日期, e.g., Jul. 6, 2024].

[3] K. Huang, " 反向傳播算法（backpropagation algorithm ）," Medium blog post, [發布日期, e.g., Sep. 23, 2019]. Available: https://medium.com/%E4%BA%BA%E5%B7%A5%E6%99%BA%E6%85%A7-%E5%80%92%E5%BA%95%E6%9C%89%E5%A4%9A%E6%99%BA%E6%85%A7/%E5%8F%8D%E5%90%91%E5%82%B3%E6%92%AD%E7%AE%97%E6%B395-backpropagation-algorithm-71a1845100cf. Accessed: [實際訪問日期, e.g., Jul. 6, 2024].

[4] W. Kang, "Back-propagation," Medium blog post, [發布日期, e.g., Mar. 20, 2020]. Available: https://medium.com/ai-academy-taiwan/back-propagation-3946e8ed8c55. Accessed: [實際訪問日期, e.g., Jul. 6, 2024].

[5] 22 12, " 模型壓縮及優化 — learning rate," Medium blog post, [發布日期, e.g., Nov. 22, 2023]. Available: https://gino6178.medium.com/%E6%A8%A1%E5%9E%8B%E5%A3%93%E7%B8%B2%E5%8F%8A%E5%84%AA%E5%8C%96-learning-rate-c340a0b940a4. Accessed: [實際訪問日期, e.g., Jul. 7, 2024].

[6] 22 12, "【機器學習 2021】預測本頻道觀看人數 (下) - 深度學習基本概念簡介" Medium blog post, [發布日期, e.g., Jan. 26, 2021]. Available: https://www.youtube.com/watch?v=bHcJCp2Fyxs&t=2699s. Accessed: [實際訪問日期, e.g., Jul. 7, 2024].