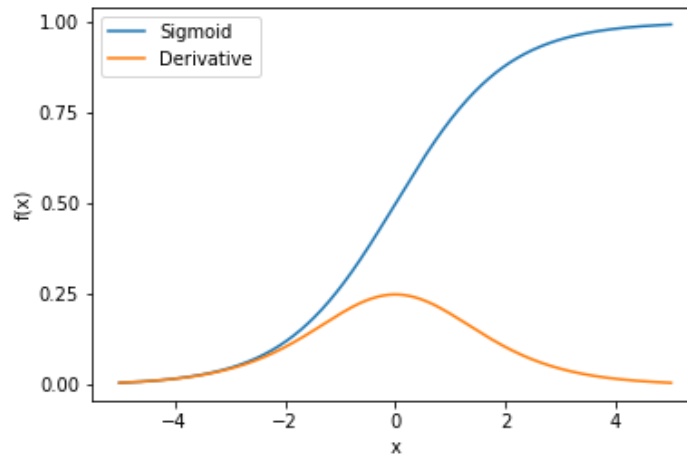


1. Introduction

本實驗分別使用 `generate_linear` 與 `generate_XOR_easy` 產生訓練資料和測試資料。兩者並分別丟入神經網絡中進行訓練，可求得以下結果。

2. Experiment setups

A. Sigmoid functions



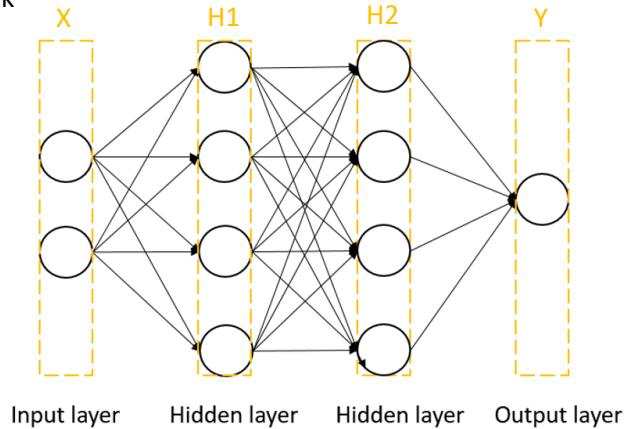
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid 為上圖中的藍線，主要目的是將輸入 x 值壓縮到 $0, 1$ 之間，可用來做二分類。

$$\text{Derivation}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Derivation 為上圖中的橘線，由於計算 **Backpropagation** 中的 **Chain Rule** 時，認為激發函數必須連續可微分。

B. Neural Network



主要分為三部分

- Input Layer：輸入資料集
- Hidden Layer：可以有 N 層 Hidden Layer，為一層層抽取特徵的過程。
- Output Layer：預測資料集最好的輸出。以利之後與 ground truth 做比較。

C. Backpropagation

用來續練 Neural Network 進行梯度下降，也就是使用 Chain Rule 計算每個 weight 對於 loss function 的影響，最後再更新 weight 使 loss function 最小化。

Step1. 用 forward 最後求得值，與 ground truth 計算誤差

Step2. 將誤差值對每個神經元偏微，計算神經元對於誤差的影響(backward)

Step3. 用誤差影響更新權重

Step4. 重複多次，即可求得最佳權重

Forward：

$$\begin{aligned}H_1 &= \sigma(XW_1) \\ H_2 &= \sigma(H_1W_2) \\ Y &= \sigma(H_2W_3)\end{aligned}$$

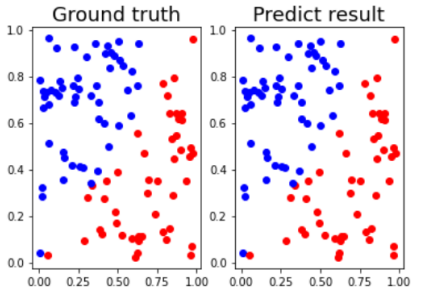
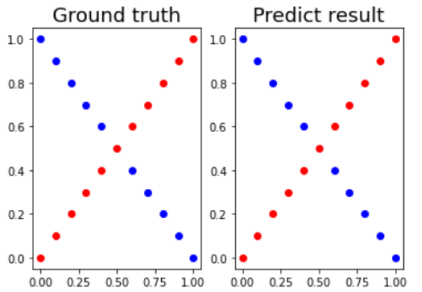
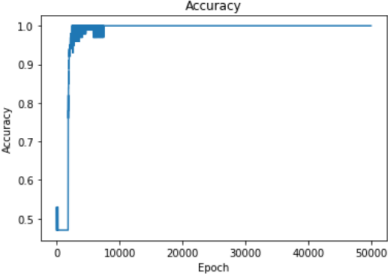
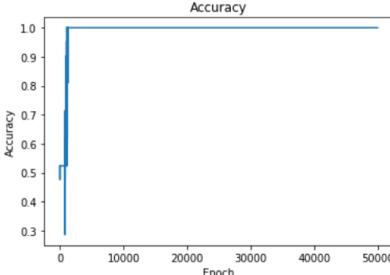
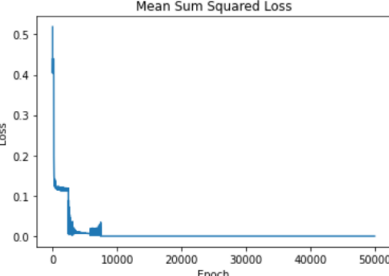
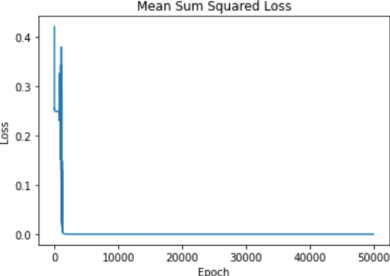
Backward：

$$\begin{aligned}\frac{\partial L(\theta)}{\partial W_1} &= \frac{\partial H_1}{\partial W_1} \frac{\partial H_2}{\partial H_1} \frac{\partial Y}{\partial H_2} \frac{\partial L(\theta)}{\partial Y} \\ \frac{\partial L(\theta)}{\partial W_2} &= \frac{\partial H_2}{\partial W_2} \frac{\partial Y}{\partial H_2} \frac{\partial L(\theta)}{\partial Y} \\ \frac{\partial L(\theta)}{\partial W_3} &= \frac{\partial Y}{\partial W_3} \frac{\partial L(\theta)}{\partial Y}\end{aligned}$$

New weight：

$$\theta^1 = \theta^0 - \rho \nabla L(\theta^0)$$

3. Result of my testing

Linear	XOR
Screenshot and comparison figure	
	
Show the accuracy of my prediction	
 <p data-bbox="462 1081 631 1115">Accuracy : 1.0</p>	 <p data-bbox="961 1094 1130 1127">Accuracy : 1.0</p>
Learning curve	
 <p data-bbox="454 1493 639 1526">Loss : 0.000000</p>	 <p data-bbox="953 1493 1138 1526">Loss : 0.000002</p>

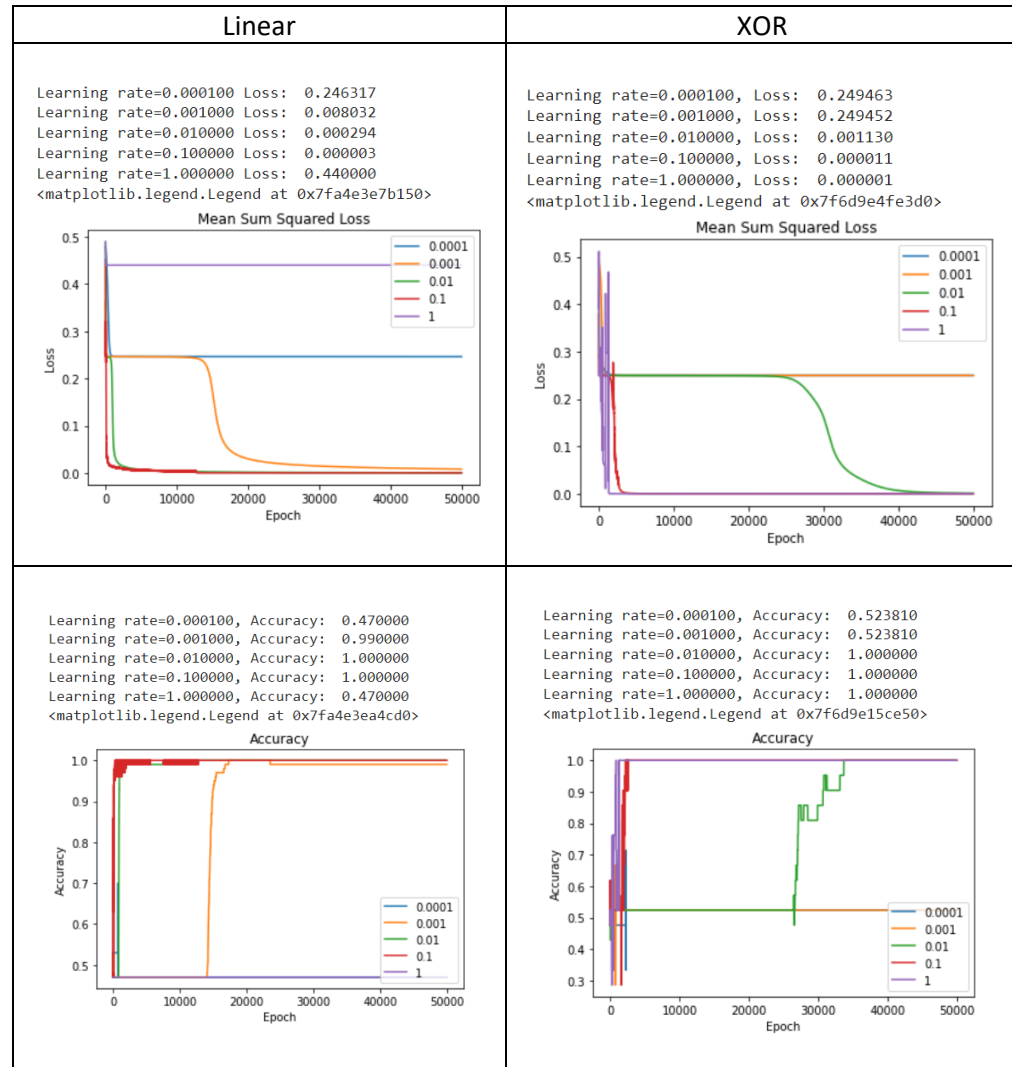
4. Discussion

A. Try different learning rates

由此結果可以出，不同的 **learning rate** 會影響到收斂速度及速度。

當 **learning rate** 越大，收斂速度越快，但同時也可能因為太快，導致最後 **Accuracy** 的結果不好。

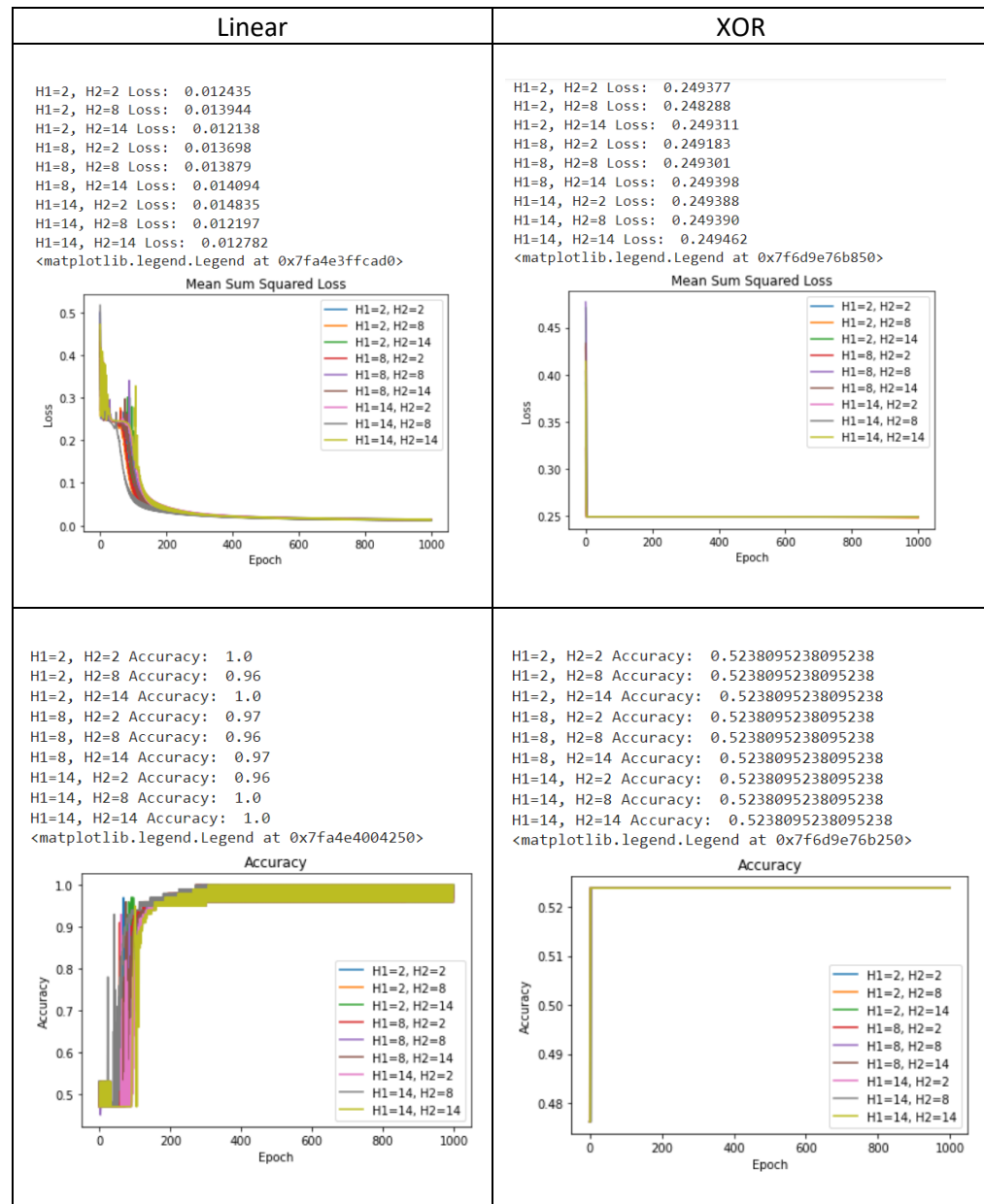
當 **learning rate** 越小，可以使收斂更加滑順，但同時訓練時花費的時間越多。



B. Try different numbers of hidden units

由結果可以看出，改變 hidden unit 的數量並沒有使 Accuracy 產生明顯的改變。

推測因為資料集小且不複雜，才使結果都幾乎等於一，但依 loss 的圖可以發現，unit 數量少時，loss 下降的速度似乎比較快。另外，進一步探討不同 hidden unit 的數量對於運算時間的影響，不過最後結果依然沒有看見特定的趨勢，可能已倍數形成長才會有明顯的趨勢。



H1=2, H2=2, Running time: 0.571002 Seconds	H1=2, H2=2, Running time: 0.226700 Seconds
H1=2, H2=8, Running time: 0.601708 Seconds	H1=2, H2=8, Running time: 0.227587 Seconds
H1=2, H2=14, Running time: 0.563897 Seconds	H1=2, H2=14, Running time: 0.203622 Seconds
H1=8, H2=2, Running time: 0.573330 Seconds	H1=8, H2=2, Running time: 0.200806 Seconds
H1=8, H2=8, Running time: 0.568916 Seconds	H1=8, H2=8, Running time: 0.200481 Seconds
H1=8, H2=14, Running time: 0.555665 Seconds	H1=8, H2=14, Running time: 0.207416 Seconds
H1=14, H2=2, Running time: 0.549923 Seconds	H1=14, H2=2, Running time: 0.221471 Seconds
H1=14, H2=8, Running time: 0.563543 Seconds	H1=14, H2=8, Running time: 0.192026 Seconds
H1=14, H2=14, Running time: 0.578566 Seconds	H1=14, H2=14, Running time: 0.215298 Seconds

C. Try without activation functions

將激活函數移除，會使 **loss** 不斷上升。

推測可能是因為在類神經網路中皆是以上層輸入的線性組合作為這一層的輸出（也就是矩陣相乘），所以才導致最後的 **loss** 被放大。

