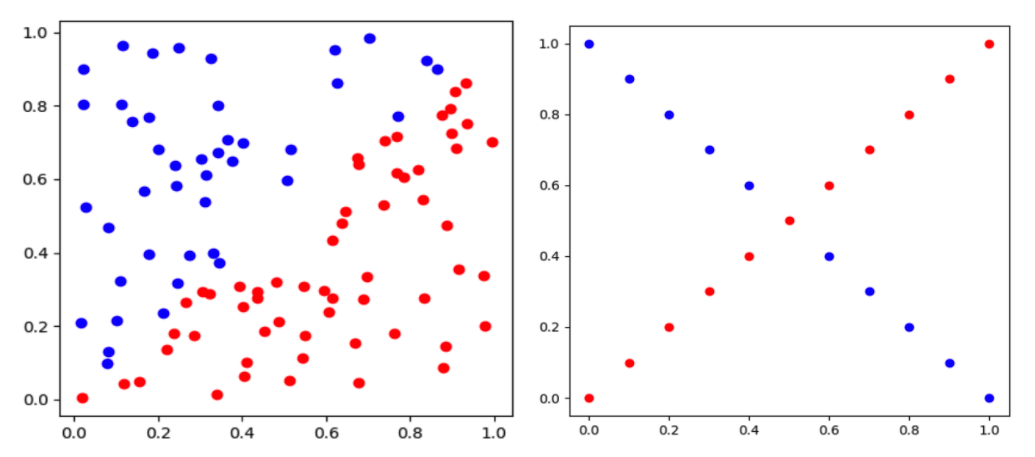
Introduction:

當我們知道gradient descent 可以調整model中weight對output的影響，就可以藉由調整參數來使模型有更好的輸出、表現。但因巨大的模型（例如：電腦視覺、語音辨識）可能有數百萬個參數需要調整，因此訓練的過程變得繁瑣且無效率。

所以現在最大的困難就是，如何把數百萬維的weight給計算出來Backpropagation的出現就是要處理這種問題，BP本身就是GD，但將每一個階段的weight對整個model輸出的影響，利用微積分的連鎖率清楚的將關係表現出來，且正因為有此關係，weight所需的改變可以利用backwardpass輕鬆算出，且計算量如同forwardpass。因此，有了BP就可以大幅提高訓練的效率。

在作業一中，分別給兩個不同的訓練資料，一為linear 的資料、二為XOR的資料，如圖一。且network architecture綁定為一層2dim input layer、兩層 hidden layer一層 1 dim outputlayer，如圖二。圖１。

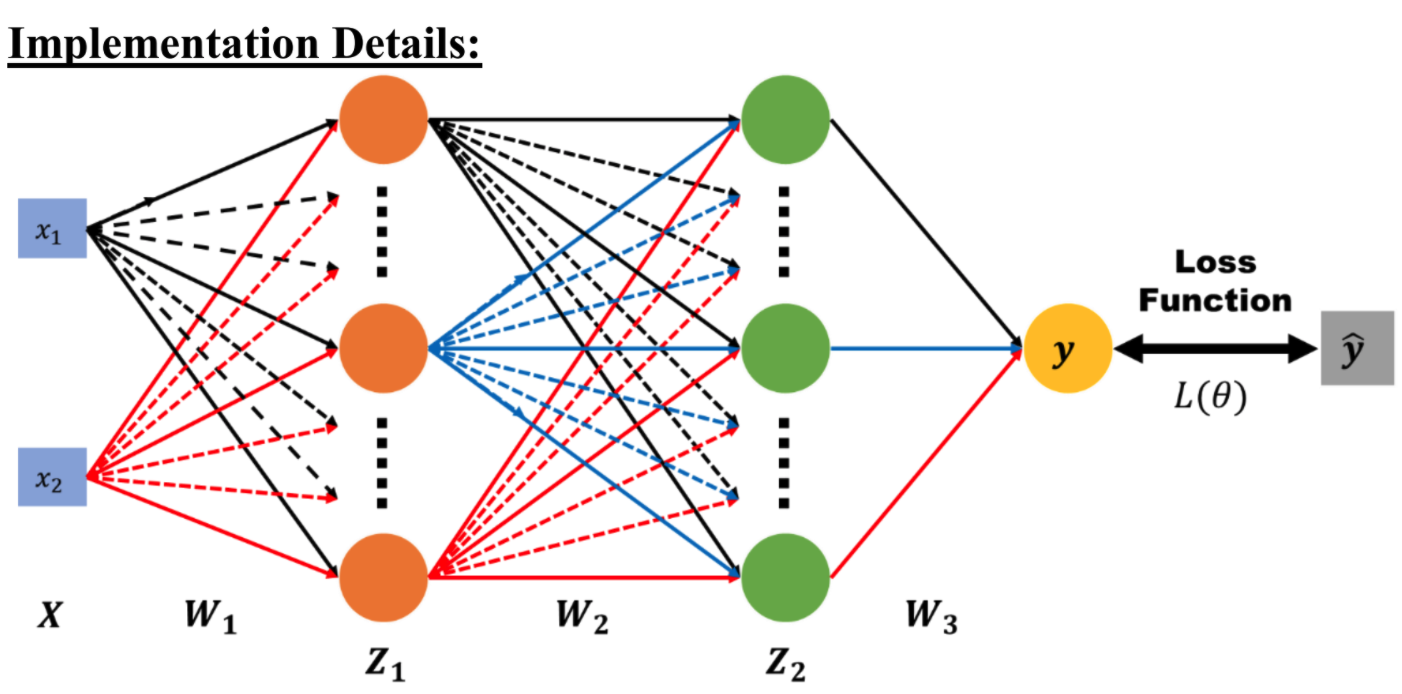
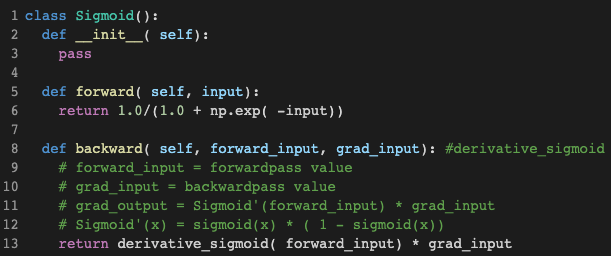


圖２。

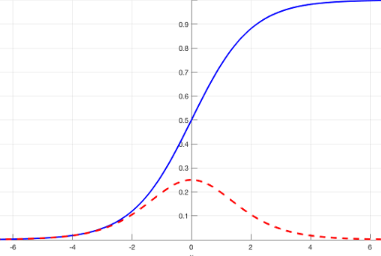
Experiment setups:

Sigmoid functions:

Sigmoid functions 為其中之一的activation function，其主要目的為，將neural的輸出轉換為非線性的，更直白的說，他能令neural對不同的數值做出不同的反應，如同決定是否發火一樣，就如同電子元件一樣超過多少電壓，才視為有輸出。而在我的實作中，我令activation function為network中的一層layer，方便管理，如圖三。

圖三。

而sigmoid 及其導數如圖四。



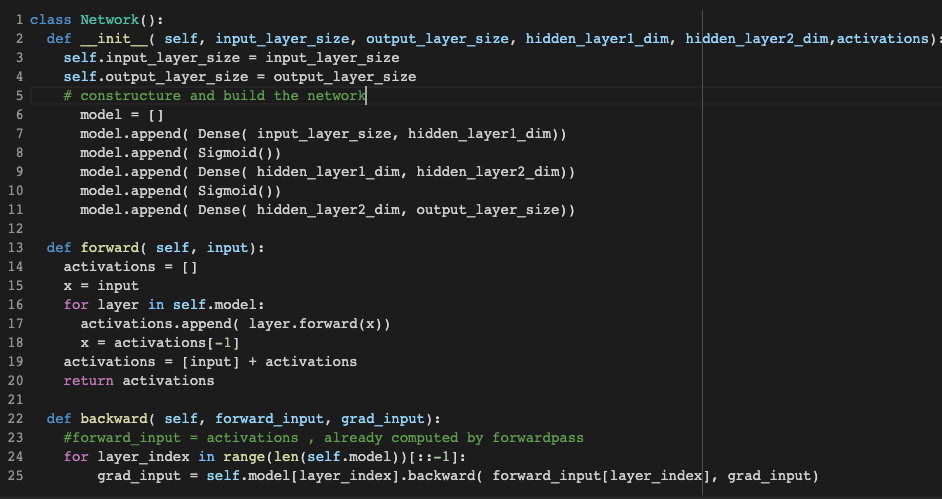
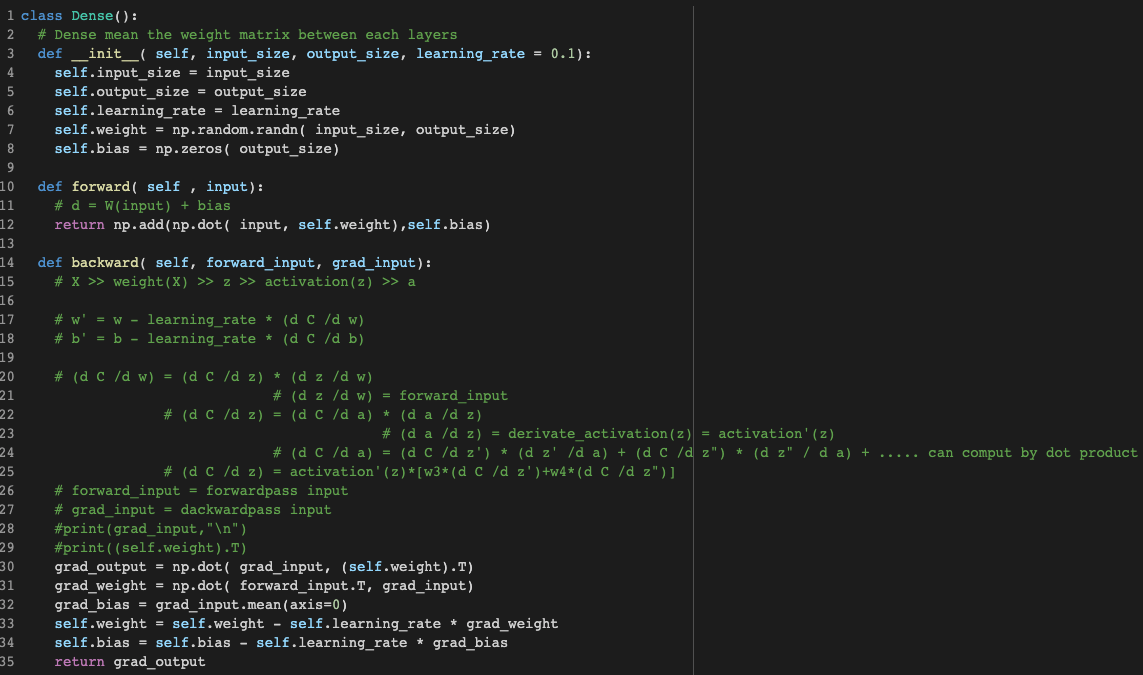
圖四，藍色為sigmoid、紅色為其導數

Network:

在整個model中最重要的部分就是層與層之間的weight參數，下一層的輸出即為上一層的輸出和參數weight的線性組合，再加上activation function。

如前述，我將整個Network利用OO method架構而成，最上層Class network 為主要骨幹，管理整個network到forward和backward，如圖五。而model中的weight則用Class Dense來管理，Dense為附屬於network，主要負責參數傳送，forward、backward、還有gradient weight的update，如圖六。

這麼做的好處就是，network不是被寫死的，模型很好修改城想要的樣子，責任分工明確。另外，宣告一list作為保存每一層結果的輸出（包含activation function），以便forwardpass、backwardpass使用。

圖五。

圖六。

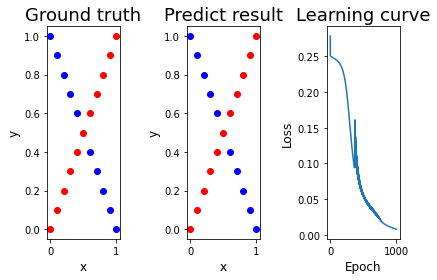
Backpropagation:

作為Deep learning中最基本也最重要的元素，BP其實也沒有什麼高深的數學，單純就只是微積分中的連鎖率。如果想計算weight對其最後cost function的影響，則可利用連鎖率拆成如下：

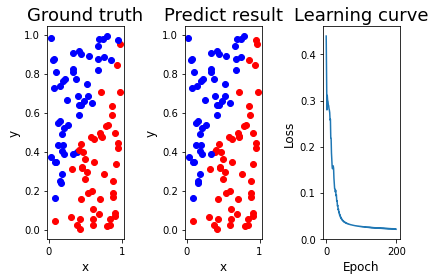
其中，

因此整個模型就可以一樣利用線性組合，反向傳播計算各weight對其輸出的影響，進而調整整個模型。大致上就是BP的精神，詳情非常推薦去看台大李宏毅的BP教學，Slides跟講解影片都非常清楚，我大概反覆看了五次才搞明白，但看完就忘了。

Results of your testing: Screenshot and comparison fugure: Show the accuracy of your prediction: Learning curve (loss, epoch curve): anything you want to present:



XOR problem , epochs = 1000 , hidden\_layer1\_dim = 10 , hidden\_layer2\_dim = 5 , act\_fun = sigmoid , n = default lreaning\_rate = 0.1 accuracy is 1.0



Linear problem , epochs = 200 , hidden\_layer1\_dim = 10 , hidden\_layer2\_dim = 5 , act\_fun = sigmoid , n = 100 lreaning\_rate = 0.1 accuracy is 0.97

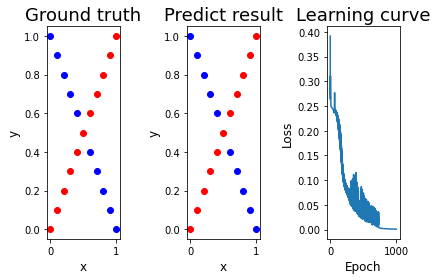
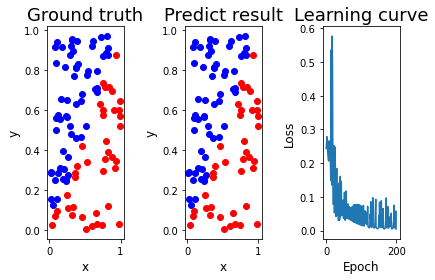
Discussion (30%)

Try different learning rates

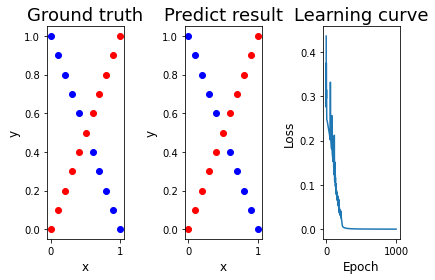
Try different numbers of hidden units

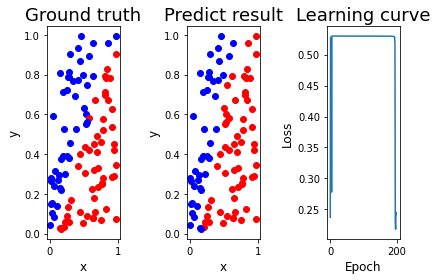
Trywithoutactivationfunctions

Anything you want to share

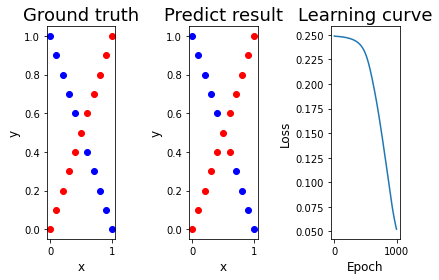
 

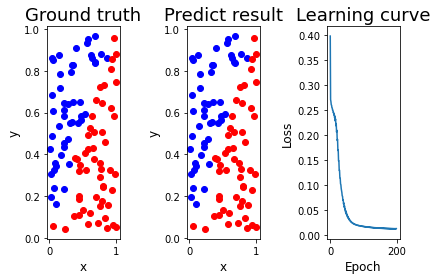
相同的配置但learning\_rate = 0.5 可以發現在這個case中，不但沒有更快降低loss而且訓練過程中震盪幅度大



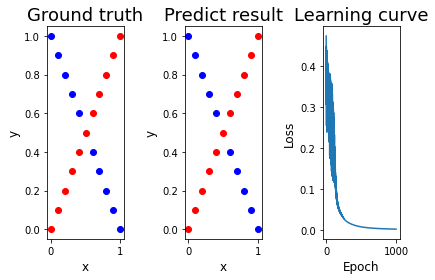


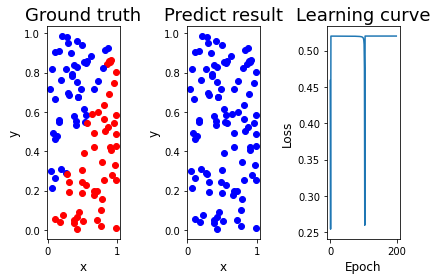
learning\_rate = 1 在learning case中甚至可以看到很不合常理的圖。





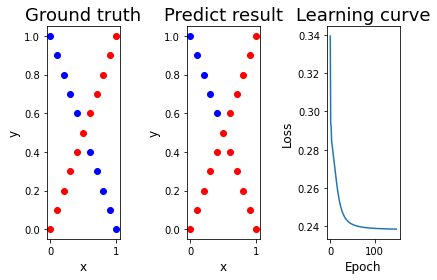
Hidden\_layer1 , Hidden\_layer2 = 5, 3

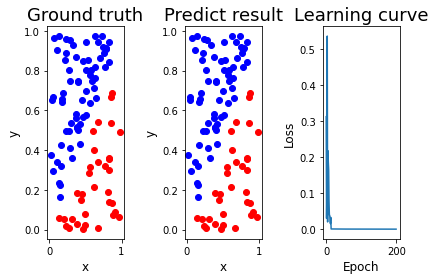




Hidden\_layer1 , Hidden\_layer2 = 50, 30

在linear problem中 accuracy is 0.48

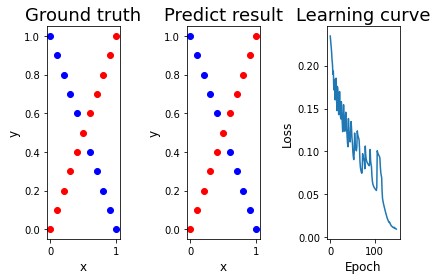
****



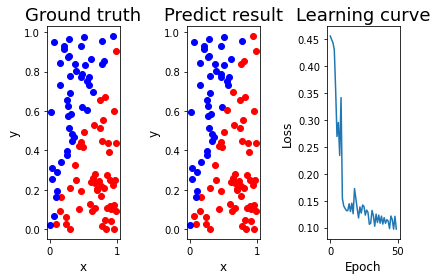
Hidden\_layer without activation function case:

XOR problem accuracy is 0.7619047619047619

Linear problem accuracy is 1.0



在Xor problem，如果將activation function改為Relu在相同架構下sigmoid需要近1000次才可以完成收斂但Relu只需要150



在linear problem中，在相同架構下，如果將activation function改成relu，訓練只要50次就可以有不錯的結果。

看似Relu 在兩個問題上似乎都比sigmoid強

結論 Relu >>>>> Sigmoid