謝侑哲

**DLP**

**LAB 1**

112550069

Github:

<https://github.com/youzhe0305/NYCU-DLP>

**1. INTRODUCTION**

本次lab為，在不使用深度學習套件的情況下，使用NumPy跟基本的library時做出一個有2層hidden layer的神經網路，示意圖如下。

一張含有 圖表, 行, 螢幕擷取畫面, 圓形 的圖片

自動產生的描述

Lab中我們實現了神經網路的architecture, forward, backpropagation以及其他需要使用到的函數的導數計算等等。

這個lab中，我們用來進行訓練與測試的數據為，在二維平面上被分成2個類別的點，也就是說我們的神經網路實現的是二元分類問題，採用Logistic Regression來實現。訓練的資料示意圖如下:

一張含有 螢幕擷取畫面, 圖表, 鮮豔 的圖片

自動產生的描述一張含有 螢幕擷取畫面, 圖表, 文字, 行 的圖片

自動產生的描述

**2. Experiment setups**

**A. Sigmoid functions**

Lab中，我們首先實現了sigmoid function，這是一個可以把輸出限制在0~1之間的activation function，算式為，其中sigmoid給出的輸出，也可以做為二元分類的機率，為y=1的class的機率，相對的也可以算出y=0的class的機率。用線性層搭配上sigmoid就可以變成該lab的實作對象Logistic Regression。

以下為我的sigmoid實現，輸入x會是numpy的array，並且是個二維的陣列，儲存了一個batch中的每個sample的資料。一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

另外，我也實現了sigmoid的導數，為，如下:

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

**B. Neural network**

如同在Introduction提到的，我實作了2層hidden layer的神經網路，並給出了示意圖，這裡我將描繪更加詳細的圖片，把架構用圖的方式描寫得更詳細。

一張含有 文字, 圖畫, 寫生, 紙張 的圖片

自動產生的描述

圖中可以看到，層數的輸入依序是X->Z1->a1->Z2->a2->Z3->a3，Z1,Z1為hidden layer的部分Z3為output layer，其中每個layer後面都會加上activation function讓模型能有更non-linear的能力。

另外，我設計的神經網路架構，可以一次吃好幾筆資料(一個batch)，並把整個batch透過矩陣運算的方式一起計算。

實作上整體的架構如下:

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

Layer的建構，我是採用直接在class用NumPy array設各個layer的matrix，用Z1, a1, W1等等方式做命名區分，其中W權重的初始化為隨機在0~1之間的數值。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

Forward的部分，因為神經元之間的連線、計算，可以整合成矩陣的運算，因此實作上我直接使用NumPy的矩陣乘法跟上面建構好的Sigmoid function來做。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Loss Function因為輸出是二元分類的機率，所以使用Binary Cross Entropy來做評估，另外因為一次吃了整個batch，因此loss我是採用batch裡面每一筆資料的平均(總和值用矩陣乘法計算)。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

訓練的部分，最基本的Model我是使用mini-batch Gradient Descent實作。一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

**C. Backpropagation**

Backpropagation的核心是透過微分的連鎖律，可以透過上一項的導數乘上下一項的導數，做到計算隔好幾層layer的導數傳遞。可以進而做到梯度下降讓loss值變得更小。Backpropagation的原理如下圖:

一張含有 文字, 白板, 筆跡 的圖片

自動產生的描述

因此為了做Backpropagation，我首先實現了神經網路中會使用到的計算的導數，利用了上課提到的Matrix微分。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

(以上Y=XW+b中，Loss對於W, X, b的梯度計算，並把後一層的梯度作為參數)

得到各算式的導數之後，就可以開始做backpropagation的流程，流程如下:從Cross-Entropy的導數開始，一步一步往下傳，套入上面的梯度計算的函數，再

把W, b等參數減去梯度\*學習率，進行更新。第二個hidden layer到output的程式碼如下圖，其他層同理。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

**3. Results of testing**

**A. Screenshot and comparison figure**

在訓練上，我皆採用10000個epoch來做訓練，0.1的learning rate，(10,10)的hidden units，並且每500個epoch做一次輸出，供我們查看loss值的變化。底下分別呈現2個data function產生的資料。

1. def generate\_liner(n=100)

loss狀況:

一張含有 文字, 螢幕擷取畫面, 功能表, 字型 的圖片

自動產生的描述

(training process)

一張含有 文字, 螢幕擷取畫面, Rectangle, 行 的圖片

自動產生的描述

(epoch-loss curve)

可以看到在經過10000個epoch之後，loss達到了幾近於0，並且準確率達到了100%，也就是說model能給出準確率高且信心也很高的預測。

分類結果比對:

一張含有 圖表, 螢幕擷取畫面 的圖片

自動產生的描述

1. def generate\_XOR\_easy(n=10)

loss狀況:

一張含有 文字, 螢幕擷取畫面, 功能表, 字型 的圖片

自動產生的描述

(training process)

一張含有 文字, 行, 螢幕擷取畫面, 繪圖 的圖片

自動產生的描述

(epoch-loss curve)

可以看到在經過10000個epoch之後，loss達到了幾近於0，並且準確率達到了100%，也就是說model能給出準確率高且信心也很高的預測。跟liner的資料主要的差別是一開始loss下降的比較慢，很可能是因為XOR的資料相對比較複雜，更偏離線性可分，所以到了1500個epoch才進入收斂。

分類結果比對:

一張含有 螢幕擷取畫面, 行, 圖表, 繪圖 的圖片

自動產生的描述

**4. Discussion**

**A. Try different learning rates**

因為XOR data的數據跑起來更明顯，所以我這裡呈現用不同的learning rate做出來的結果。同樣是每500 epoch為1個點，共訓練100000次。

Learning rate: 0.01 (1/10倍)

一張含有 文字, 行, 繪圖, 螢幕擷取畫面 的圖片

自動產生的描述

可以明顯看見loss的下降速度變慢了，並且有很長一段的時間卡在差不多的數值上，很可能是在那段接近水平的線期間，model進入了local minimum。類似於掉入了一個小低谷，到了後面才成功爬出來。

這也展現了learning rate太小的問題，model很可能因為某個局部的極小值就停止更新。步長不夠讓他跨出那個局部極小值的低谷。

Learning rate: 1 (10倍)

一張含有 文字, 螢幕擷取畫面, 行, 陳列 的圖片

自動產生的描述

可以看到loss下降的速度比0.1時更快，但是他收斂的地方在2.8e-4，相較於學習率0.1時的收斂值3.3e-5是稍大一些的。反映出如果學習率的步長設的太長的話，如果global minimum的低谷比較淺，那model就會跨不進去裡面，或是一跨進去裡面就會直接跳出來，所以學習率設太高，雖然訓練得比較快，但是有可能達不到global minimum。

**B. Try different numbers of hidden units**

這裡使用linear data來做呈現，因為這樣的結果更明顯。我採用的training data與test data是同一批，所以探討hidden unit數量過多造成的overfitting是不實際的，因此這裡只探討hidden unit不足時的結果。

\

1. hidden units: (1,1)

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

可以明顯看見訓練的速度變慢了，甚至中間有一段loss上升，可見當參數不足時，是很難在少量的更新得出比較好的結果的，而後面收斂的那段，收斂的loss值為1e-3，明顯少於我設定的一般情況(10,10)的7.3e-10，差了10^7倍。所以當hidden unit不足時，他是很難給出非常好的結果的(雖然在這個case中準確率仍是100%)。

**C. Try without activation functions**

這裡把2個hidden layer的activation刪除並測試結果，使用的是XOR data，因為XOR更加的non-linear，而去掉activation就是讓model的non-linear能力下降。。

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

可以看見一下子收斂，但是收斂的loss值在0.71，相較於標準的3e-5是差了好幾個量級。準度也只剩下52%，從下述分布圖中可以看見，她幾乎就是在用亂猜的

一張含有 螢幕擷取畫面, 文字, 圖表, 繪圖 的圖片

自動產生的描述

從這個例子可以明確看出activation造成的non-linear效果有多大的影響，XOR的這種極端線性不可分的資料，就會在沒有activation的狀況下直接炸掉。

**5. Extra**

**A. Implement different optimizers.**

原本使用的optimizer是mini-batch Gradient Descent，而在此之上，我把它改成了Stochastic Gradient Descent (SGD)，做法是在每個epoch之前，都先把整個dataset的順序打亂，再套上mini-batch Gradient Descent。程式碼如下，多了把資料的index做shuffle的部分。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

Result:

Linear data:

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

XOR data:



Linear data epoch-loss curve:

一張含有 文字, 螢幕擷取畫面, 繪圖, 行 的圖片

自動產生的描述

XOR data epoch-loss curve

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

從數據及各圖中可以看出，SGD的效果在這個case會跟一般的mini-batch GD差不多，在XOR的線似乎有訓練的更快的跡象，但不是很明顯。不過如果換在更複雜的case，SGD應該會比mini-batch GD的效果更好，因為batch內數據的隨機排列組合讓generalize的能力更強。

**B. Implement activation functions.**

在此lab的基本神經網路架構中，採用sigmoid作為hidden layer的activation function，不過在大部分的深度學習中，大家還是比較傾向於採用ReLU系列的activation function，因為他的導數會是單純的0, 1，可以避免梯度消失或爆炸的問題，也可以讓負數的數值直接被砍掉，相當於讓bias成為了神經元被激發的門檻，會讓整體的non-linear有足夠的強度。

這次實做中，我分別採用了ReLU及其延伸Leaky ReLU:

(1) ReLU:

實現:

基本上就是對於整個matrix套max(0,x)，即對於每個神經元分別套ReLU

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

Result:

Linear Data:

一張含有 螢幕擷取畫面, 文字, 行, 繪圖 的圖片

自動產生的描述

XOR Data:

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

基本上得到的成效跟套sigmoid是差不多的，可能要在更深的神經網路上，才能比較看出sigmoid跟ReLU的差別，另外在實驗的過程中，我也有觀察到使用ReLU後壞掉的狀況，如圖:

一張含有 文字, 螢幕擷取畫面, 繪圖, 行 的圖片

自動產生的描述

中間有loss逆勢上升的狀況，讓優化變得很不穩定，我推測可能是ReLU的一個特性，因為出現負數後，梯度為0，導致神經元被關閉，可能會比較難再次開啟他，就會導致model的複雜度下降，讓她沒辦法給出很好的結果，甚至變得更糟。因為觀察到了ReLU可能出現的不穩定性，所以我後面嘗試了更soften的Leaky ReLU。

1. Leaky ReLU:

實現:

就是ReLU在負數時的輸出由0改成alpha\*x，alpha設為0.01

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

Result:

Linear Data

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

XOR Data:

一張含有 螢幕擷取畫面, 文字, 繪圖, 行 的圖片

自動產生的描述

得到的效果在這個case裡面仍是差不多，不過在我的多次實驗下，都沒有出現像是ReLU那樣不穩定的問題，Leaky ReLU讓神經網路的訓練能夠保持穩定的梯度傳遞，就不會出現花好久時間訓練，結果就因為一點不穩定性爆炸的狀況。