1. Introduction

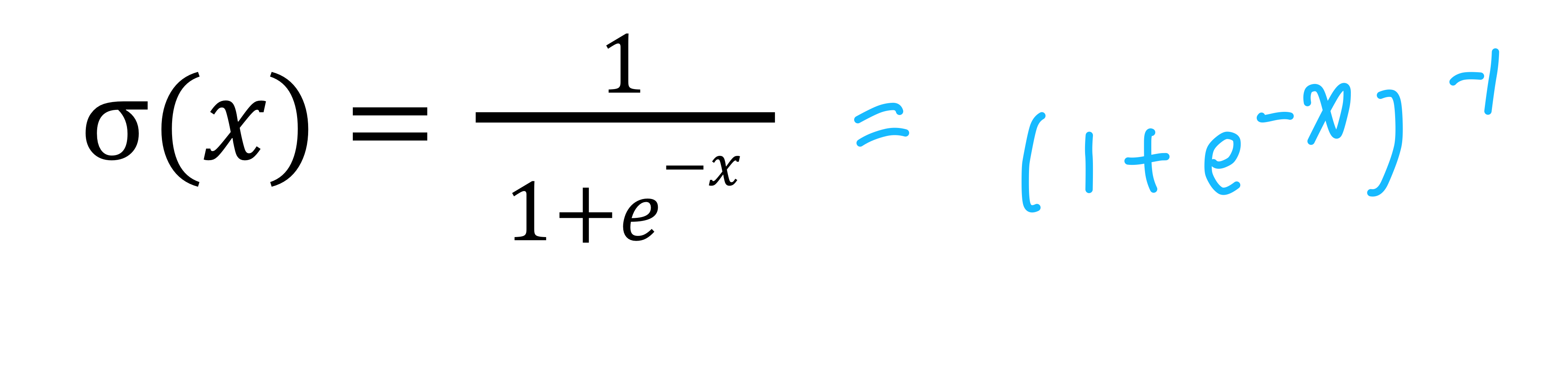
在這次的lab中，我實作了一個有兩層hidden layer的簡單的neural network。將input餵進網路中後，可以利用forward propagation算出此input預測的分類值(正確來說是算出一個數字，此數字大於0.5的話就將此input預測為label 1，反之則將此input預測為label 0)。為了使neural network中的參數們(weight)能夠足夠合適(能使loss function有最小值的weight們)，將input預測為正確的分類，就必須在訓練過程中一步步調整參數的值，直到訓練完成。神經網路預測出的數值和真實答案的差距(loss)經過back propagation的向後計算之後，就可以得到這些參數更新的方向，使得預測結果可以越來越接近真實答案。

除了實作出神經網路之外，在此lab中也會透過調整各種不同的hyper parameter(如網路的hidden width、learning rate等等)，來觀察它們所造成的各種影響。

1. Experiment setups
2. Sigmoid functions

在神經網路中，前一層input和weight做內積之後，會再通過一層非線性的activation function，才正式變成那一層的output。在這裡sigmoid function就是做為這個非線性的activation function。

下圖為sigmoid function的定義



下圖則為計算出的sigmoid function的微分

一張含有 圖表 的圖片

自動產生的描述

因此參考了助教的hint之後，code中的derivative\_sigmoid的input值應該為要微分位置的sigmoid output值，而不是要微分的位置。

一張含有 文字 的圖片

自動產生的描述

1. Neural network

參考spec中所附的網路架構圖，我的網路中儲存了各個weight以及各個hidden layer的output值，才能在forward propagation以及back propagation時都將他們儲存起來。

一張含有 圖表 的圖片

自動產生的描述

這是當所有input data一起和第一層layer weight做內積的矩陣乘法。在這次時做的神經網路中，每次只會有一筆data被送進來，因此我的各個神經網路輸入輸出都是以列向量的方式儲存，weight則是以矩陣方式儲存，wij代表的是要和前一層第i個neuron值做相乘，成為此層第j個neuron值一部份的weight。

一張含有 圖表 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

因為每次只會有一筆data丟進神經網路，forward的寫法就是照順序把各層都串起來。

一張含有 文字 的圖片

自動產生的描述

1. Backpropagation

在backpropagation的一開始，我先計算min square error來作為loss function。

接著再將下圖寫出來的微分式子由下往上一步步算出來，前面算出來的結果可以給後面要算微分的式子用的。比較要注意的是計算微分時的矩陣維度，以及相乘所用的符號。’\*’是element-wise的乘法，’@’則是一般的矩陣乘法。像是sigmoid的微分時，因為原本sigmoid function的input和output維度就相同，所以在微分的chain rule相乘時，我就會使用element-wise的’\*’。像是計算weight的微分時，用的就會是矩陣乘法的’@’，並且也要注意何時要轉置矩陣。

一張含有 文字, 白板 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

算出loss對weight的微分之後，因為此gradient所指的方向是指向loss升高最多的地方，而我們所想要的是loss最低的地方，所以會用加負號來更新weight，同時也會在乘上learning rate後再更新參數。

一張含有 文字 的圖片

自動產生的描述

1. Results of your testing
2. Screenshot and comparison figure

兩種不同的input資料train到最後都能在test時獲得100%準確率。

|  |  |
| --- | --- |
| Linear | XOR |
| 6000 epochs | 60000 epochs |

一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述

1. Show the accuracy of your prediction

|  |  |
| --- | --- |
| Linear | XOR |

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

1. Learning curve (loss, epoch curve)

|  |  |
| --- | --- |
| Linear | XOR |

一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述

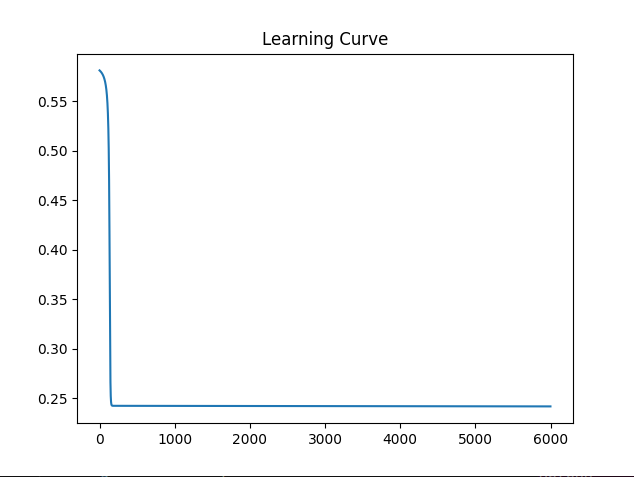
1. anything you want to present

比起linear的資料，XOR的資料相對來說是比較困難的問題，不僅training到收斂所需的epoch數比linear多(大概是10倍的關係)，從learning curve中也可以看出，雖然兩者一開始的loss都會快速下降，可是在到完全收斂時，都會有一段loss有點停滯下不去的階段，這個停滯的階段，XOR的也比linear的長很多，接著再到最後終於開始下降時，雖然相比停滯的時候已經是快速下降了，但XOR到最後收斂時大約也花了10000個epoch。另一個造成XOR的training epoch數比linear多的可能的原因則是資料筆數，linear的資料筆數有100筆，但XOR的資料筆數只有20筆。

1. Discussion
2. Try different learning rates
3. Linear

先固定其他參數，hidden width = (10, 10) (兩層hidden layer都各10個neuron)、training epochs = 6000

|  |  |  |  |
| --- | --- | --- | --- |
| Lr = 0.001 | Lr = 0.01 | Lr = 0.1 | Lr = 1 |

 一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

由圖中可以看到，當lr很小時(0.001)，train到6000個epoch的時候他其實還沒收斂，準確率只有近六成左右。而當lr變大，lr=0.01~1左右時，到6000個epoch時雖然都有收斂，不過可以觀察出這三個在收斂過程中的分別。Lr=0.01時，其實lr還是偏小，因此在loss curve中有一段明顯loss停滯不往下的階段，這可能就是因為lr太小，所以對參數的改變不大造成的。Lr=0.1則相對是一個不錯的lr。到了Lr=1時，雖然有順利收斂，不過因為lr偏大的關係，在收斂到最低點時會有震盪的現象。Lr=7則過大，震盪更明顯且無法收斂至loss=0。

|  |
| --- |
| Lr = 7 |

一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述

1. Xor

一樣先固定其他參數，hidden width = (10, 10) (兩層hidden layer都各10個neuron)、training epochs = 60000

|  |  |  |  |
| --- | --- | --- | --- |
| Lr = 0.001 | Lr = 0.01 | Lr = 0.1 | Lr = 0.7 |

一張含有 正方形 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

XOR的結果和linear的也類似。不過在lr=0.7時，可能剛好那時參數的所在位置在陡峭的地方，因此更新太多時，讓他的loss變得非常大，雖然最後順利收斂了。

|  |
| --- |
| Lr = 10 |

一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述

1. Try different numbers of hidden units
2. Linear

固定其他參數，lr=0.05、training epochs = 6000

|  |  |  |
| --- | --- | --- |
| Hidden\_width = (2, 2) | Hidden\_width = (6, 6) | Hidden\_width = (10, 10) |

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

我覺得他們的結果主要差別可能是loss的初始值，因為hidden unit多的網路可以模擬比較複雜的函數，但可能這個linear data的複雜度並沒有很高，所以在一開始train的初始loss時，由左到右hidden unit多的網路反而loss最多。

1. XOR

固定其他參數，lr=0.1、training epochs = 60000

|  |  |  |
| --- | --- | --- |
| Hidden\_width = (2, 2) | Hidden\_width = (6, 6) | Hidden\_width = (10, 10) |

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

使用XOR作為資料，比較能看出hidden unit數所造成的影響。在hidden unit數只有2\*2時，因為model能力較弱的關係，到了epoch 60000時，都無法收斂至loss為0。當hidden unit數變多時，其他兩個結果都能順利收斂至loss約為0的地方。而且hidden unit數10\*10的收現情形，稍微比6\*6的再平滑一點點，可見在資料分布較難的情況下，hidden unit數變多，有助於學習。

1. Try without activation functions
2. linear

固定其他參數，lr=0.1、training epochs = 6000、hidden units = 6\*6

|  |  |
| --- | --- |
| 有activation function | 沒有activation function |

一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

沒有加activation function的情況，因為model少了非線性元素在，能力變弱了，變得比較難train。雖然理論上線性的input應該也能在少了activation fuction的model上train到100%準確率，不過我調整了幾個參數後，都沒能達到100%準確率，我在猜會不會是因為我實做的model並沒有加上bias有關。

1. XOR

固定其他參數，lr=0.1、training epochs = 60000、hidden units = 6\*6

|  |  |
| --- | --- |
| 有activation function | 沒有activation function |

一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述一張含有 圖表 的圖片

自動產生的描述 一張含有 圖表 的圖片

自動產生的描述

因為xor是非線性的關係，所以在沒有加activation function的線性model中，應該是沒有辦法train起來的，而且到最後的情況應該會趨近於亂猜(50%準確率)。跑出來的結果也和我的猜測很符合。

1. Anything you want to share

我在寫作業的時候，原本有碰到下面這種維度的問題，本來搞不清楚覺得下面例子中的a和b[0]明明都是row vector為什麼shape卻一個是(1, 4)一個是(4,)，結果在矩陣乘法那裡花了一點時間。不過現在我搞懂了，雖然表面上他們感覺都是row vector，不過a的寫法實際上是一個”二維”的一列四行的矩陣。

一張含有 文字 的圖片

自動產生的描述