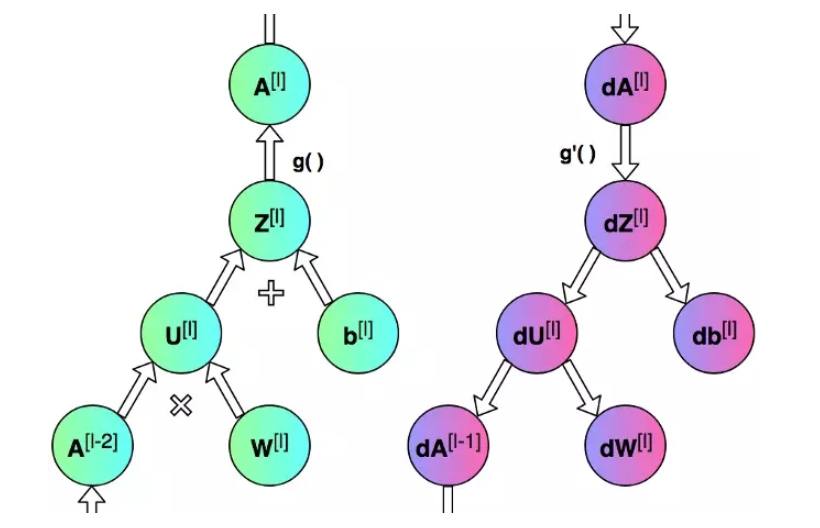
DEEP LEARNING

Homework 1

309511045 黃麟凱

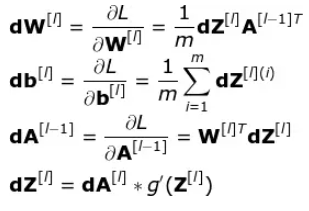
1. 神經網路：
   1. Forward & Backward propagation

深度學習的計算，本身是透過對每個網路層裡的節點做forward & backward propagation，計算得出來的結果，而每個節點的計算，又可以把它想像成一種矩陣計算，如下圖所示：



forward propagation計算上可以想像成這樣的表示式：

其中A表示Z經過activation(g function)之後的結果；而backward propagation在計算上需要使用微分，原因是我們在做backward propagation時，希望的是讓節點輸出的值能越接近我們想要的值，因此藉由微分算出其向量之後，在對每個節點做update，因此back propagation算式會如下圖所示：



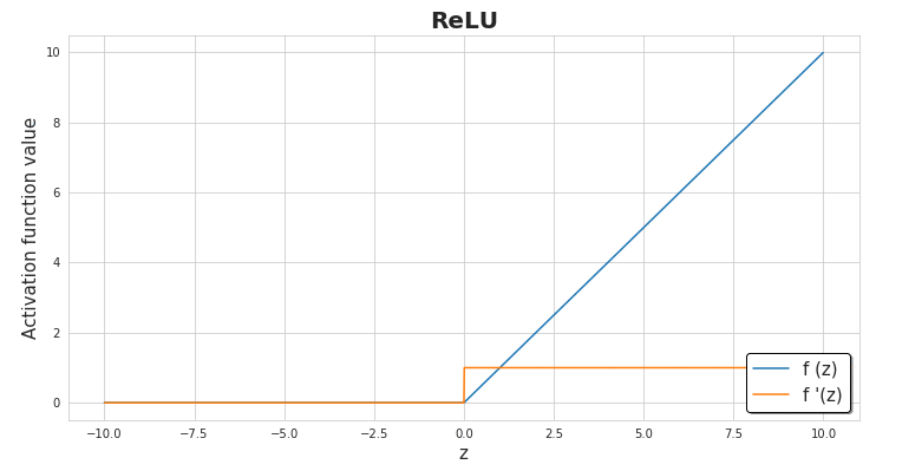
有了以上的算式後，就可以簡單的創出自己的神經網路模型

* 1. Activation Function

基本上在使用Activation Function的目的在於讓神經網路輸出不再是一個linear function，如上述提到的，神經網路內部的傳播就只是矩陣的運算而已，因此算出來的結果仍然是線性的，因此需要藉由Activation Function來讓結果非線性化，本次我有使用的有ReLu & Sigmoid。

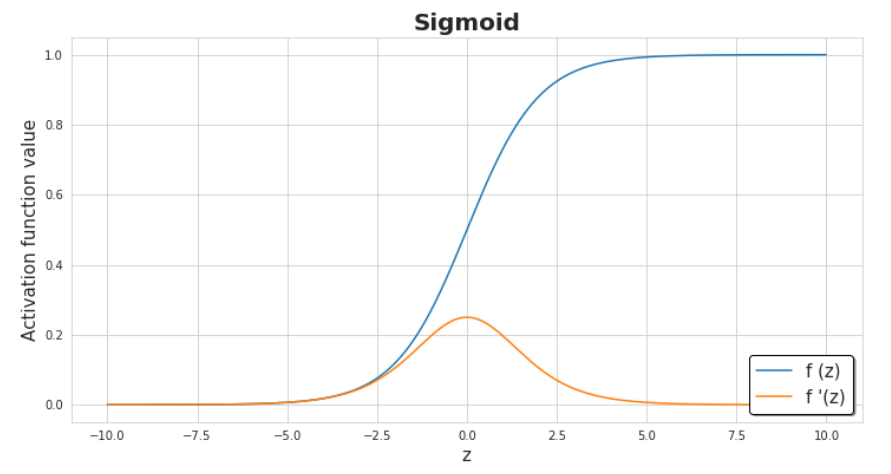
* + 1. ReLu

ReLu的算法很簡單，就是讓輸出結果大於0時不變，而小於0時讓其值為0，其算式以及微分結果如下圖所示：



* + 1. Sigmoid

Sigmoid是希望讓輸出結果讓介於0~1之間，常用於在分類的網路上，如這次的作業的第二部分，其算式以及微分結果如下圖所示



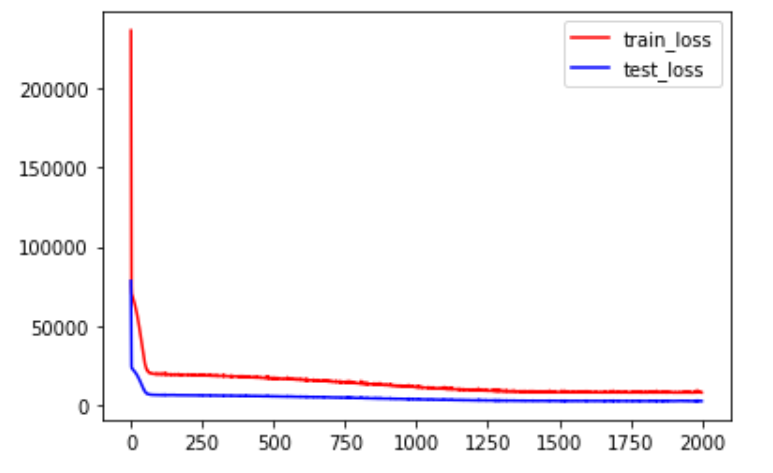
1. Regression(作業的第一部分)

(a)(b)

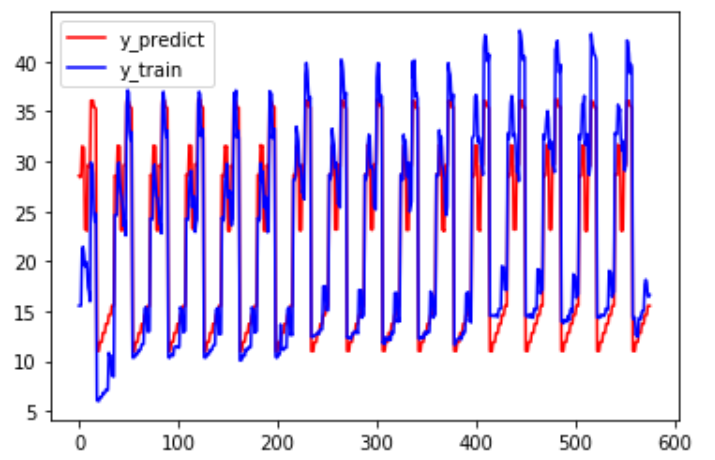
1. network architecture & training/testing error rate

|  |  |
| --- | --- |
| Network architecture | 16 – 32 – 16 – 8 – 1 |
| Selected feature | [0,…,7](全都用) |
| Training | 3.775739 |
| Test | 3.758734 |

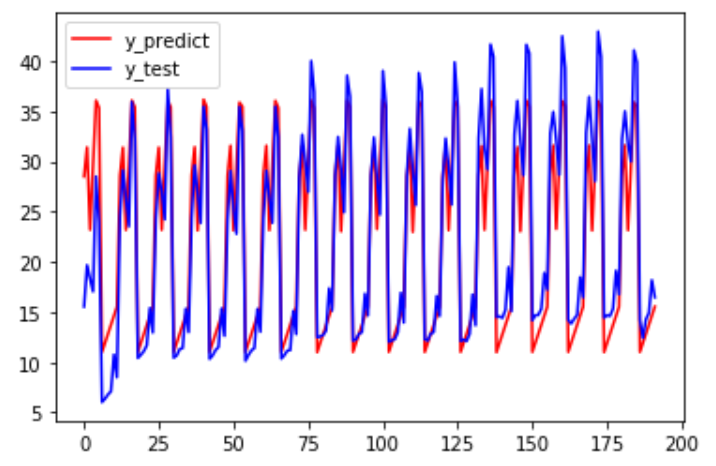
2. learning curve：



3. Regression result with training labels

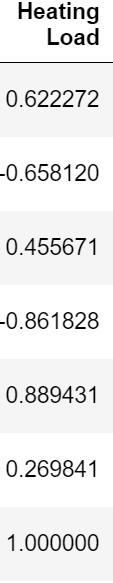
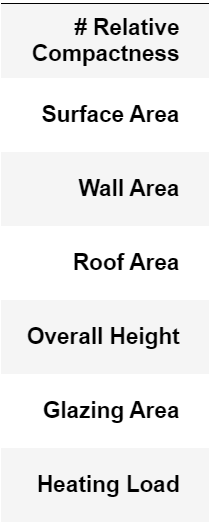


4. Regression result with testing labels



(c)

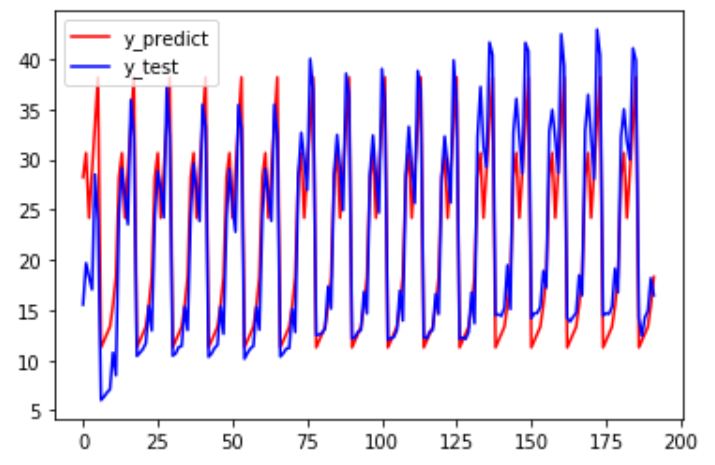
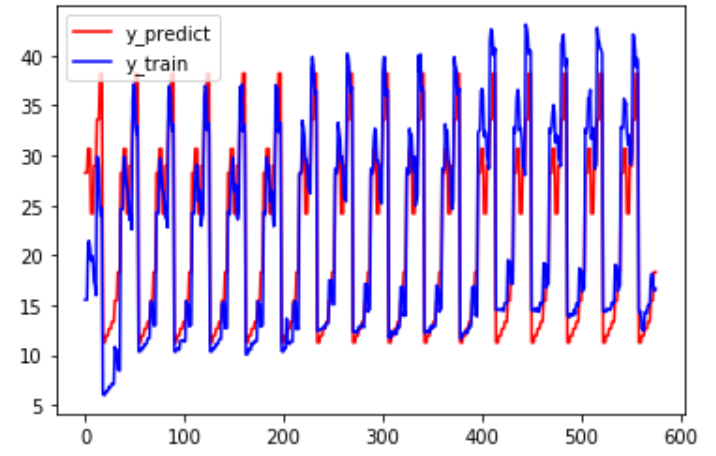
我計算各個feature以及output之間的相關係數，其中有排除掉那些需要做one-hot encoding資料，因為算不出相關係數，結果如下圖所示：



我在挑選的時候只選相關係數高的，因此選了”Roof Area”、”Overall Height” and “ Surface Area”，由於feature變少，因此網路的架構也要稍微改一下：

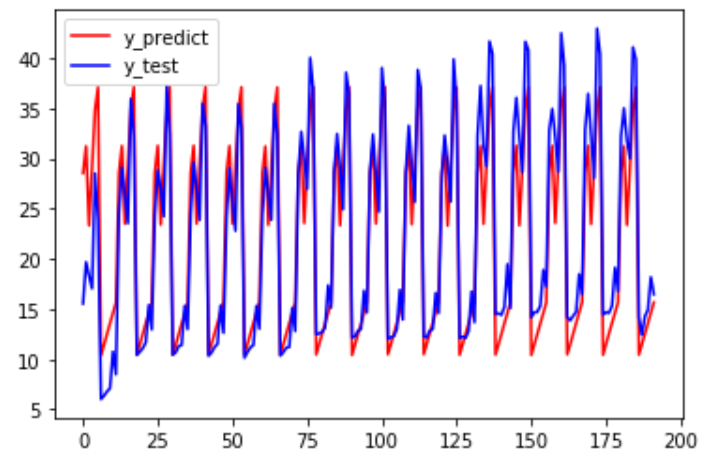
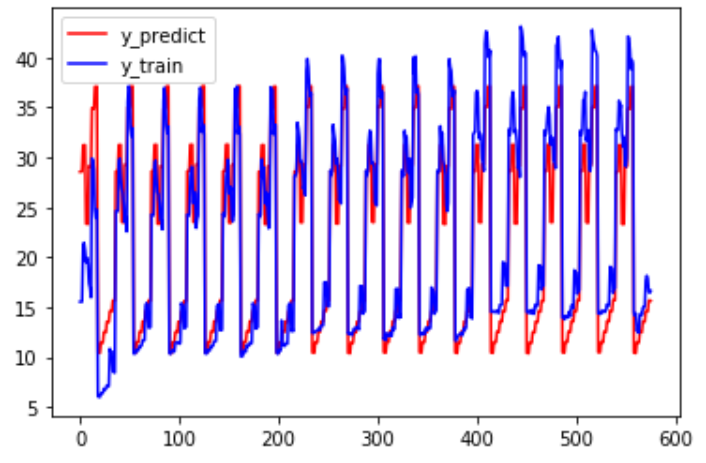
|  |  |
| --- | --- |
| Network architecture | 3 – 32 – 16 – 8 – 1 |
| Selected feature | [1, 3, 4] |
| Training | 4.006456 |
| Test | 4.047719 |

而其Prediction的結果如下：



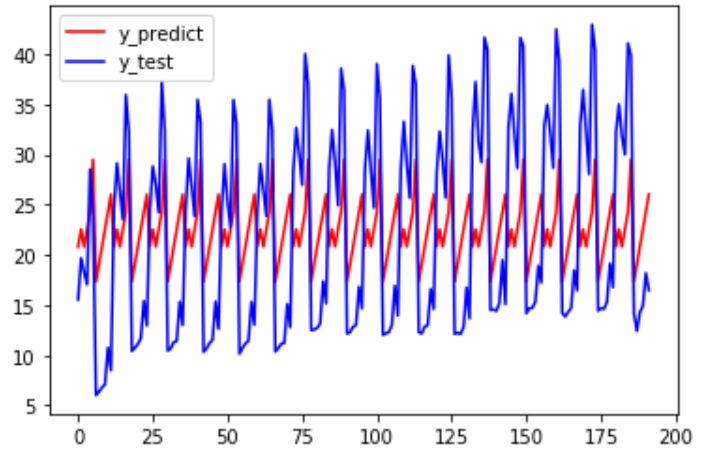
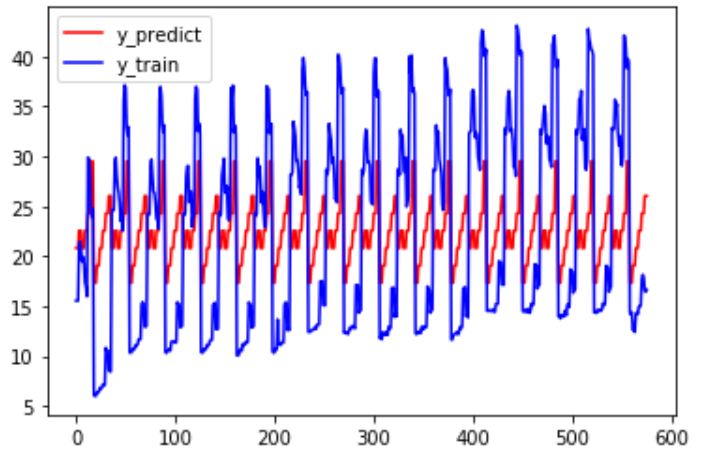
就結果來說，比全部feature都丟進去差了一點，可能的原因在於少了categorical features，所以將categorical features加回來，結果如下：

|  |  |
| --- | --- |
| Network architecture | 13 – 32 – 16 – 8 – 1 |
| Selected feature | [1, 3, 4] |
| Training | 3.820058 |
| Test | 3.859445 |



得到RMS的結果已經很接近全部feature都丟進去訓練的結果，會有一些差別可能來自於訓練的不足之類的。最後我又嘗試了一下被我只用被我剔除的那些feature來訓練會怎樣，結果如下：

|  |  |
| --- | --- |
| Network architecture | 3 – 32 – 16 – 8 – 1 |
| Selected feature | [0, 2, 6] |
| Training | 9.098864 |
| Test | 9.110089 |



從數據可以看出，其結果跟前面比起來差了很多，但有一點點擬合的感覺，可能的原因在於我仿造前面的模式，都訓練一樣的epochs，若訓練長一點說不定就可以達到跟其他狀況差不多的結果，但有此可知，計算相關係數是一個能挑出有用feature的方法。

1. Classification(作業的第二部分)

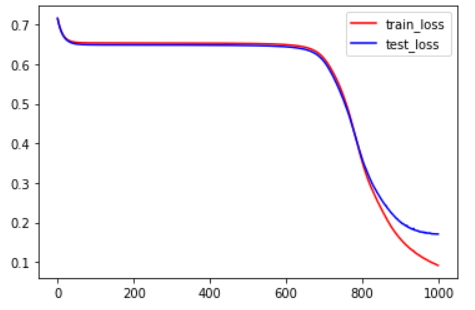
(a)(b)

1. network architecture & training/testing error rate

|  |  |
| --- | --- |
| Network architecture | 34 – 17 – 8 – 2 – 1 |
| Selected feature | [0,…,33](全都用) |
| Training | 0.01071% |
| Test | 0.0563% |

2. learning curve：

實驗過程發現，有時很快就會收斂，但有時會像下圖那樣，過了好幾個epochs才收斂，可能的原因在於，由於SGD在作計算時，容易陷入local min的狀況，導致無法收斂。

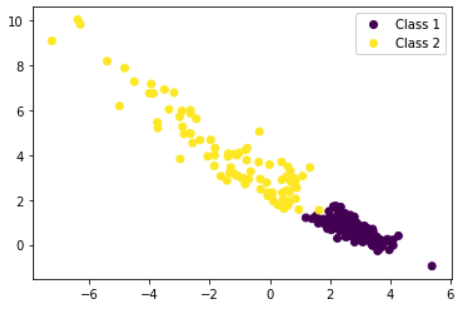
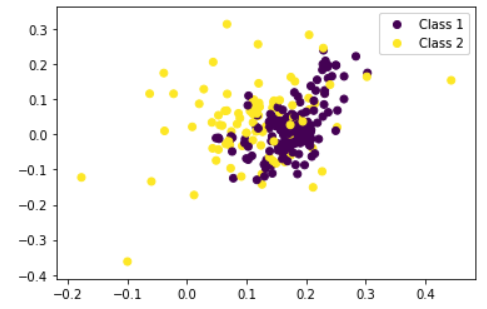


(c)

1. if the latent feature we choose 2 dims：

|  |  |
| --- | --- |
| Network architecture | 34 – 17 – 2 – 1 |
| Selected feature | [0,…,33](全都用) |
| Training | 0.0036% |
| Test | 0.0704% |

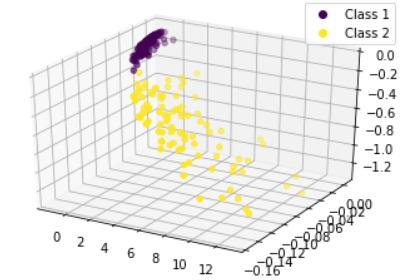
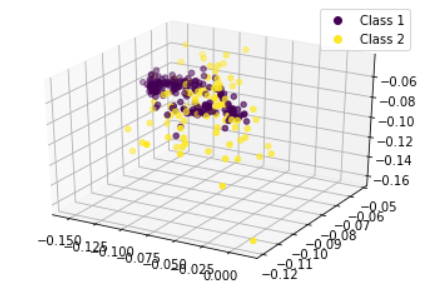
2D feature 10 epochs 2D feature 1000 epochs



2. if the latent feature we choose 3 dims：

|  |  |
| --- | --- |
| Network architecture | 34 – 17 – 3 – 1 |
| Selected feature | [0,…,33](全都用) |
| Training | 0.0036% |
| Test | 0.0704% |

3D feature 10 epochs 3D feature 1000 epochs



1. 心得

這次的作業寫起來非常有難度，因為很少遇到有課程會讓學生手刻function，一開始真的有點不知該如何下手，而且剛開始在做的時候老師還沒教到back propagation的部分，但所幸網路上的資源很多，很多人都願意分享相關知識，老師最後也有上到相關的部分，從中學習到簡單Deep learning的就只是矩陣間的運算而已，並沒有一開始想的那麼複雜。

Reference：

[1] <https://www.itread01.com/feffx.html>

[2] <https://quantdare.com/create-your-own-deep-learning-framework-using-numpy/>

[3] <https://towardsdatascience.com/lets-code-a-neural-network-in-plain-numpy-ae7e74410795>