Report of Homework 3

姓名: 吳耿暉

學號: 0556171

所有討論基於使用每個class各800筆資料當作training data, 剩下的200筆則為validation。在weights和biases的初始化上, weights使用standard normal * sqrt(2/n) (arXiv:1502.01852), 而biases則初始化為0。

1. Principal Component Analysis(PCA)

在此簡單的使用scikit-learn裡面的PCA來進行降維, 速度比自己寫的快非常多, 並且在上一個作業的實驗中, 一開始對於資料做z-normalization對後面的效果會較好, 因此所有資料在進行PCA投影以及做PCA的fit的前都先經過z-normalization才做相關的preprocessing。

2. Stochastic gradient descent

feedforward pass:

$$z^1 = W^1 x^r + b^1$$

$$z^l = W^l a^{l-1} + b^l, \;\; a^l = \sigma(z^l)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

output-layer轉為softmax輸出。

backward pass:

$$\frac{C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l}\frac{\partial C^r}{\partial z_i^l} = a_j^{l-1}\delta_i^l$$

1. output layer

$\delta^L = \sigma(z^L) \cdot \nabla C^r(y^r)$

2. other layers

$\delta^l = \sigma'(z^l) \cdot (W^{l+1})^T \delta^{l+1}$

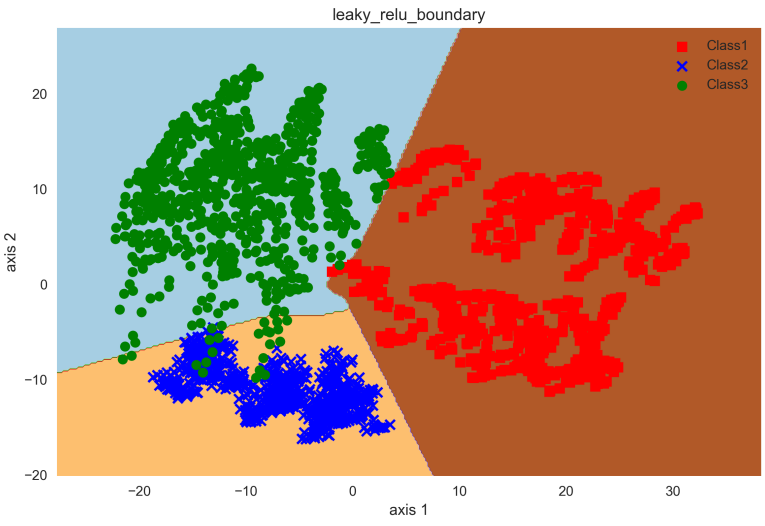
(1) 2-layer(1-hidden layer) network using sigmoid function

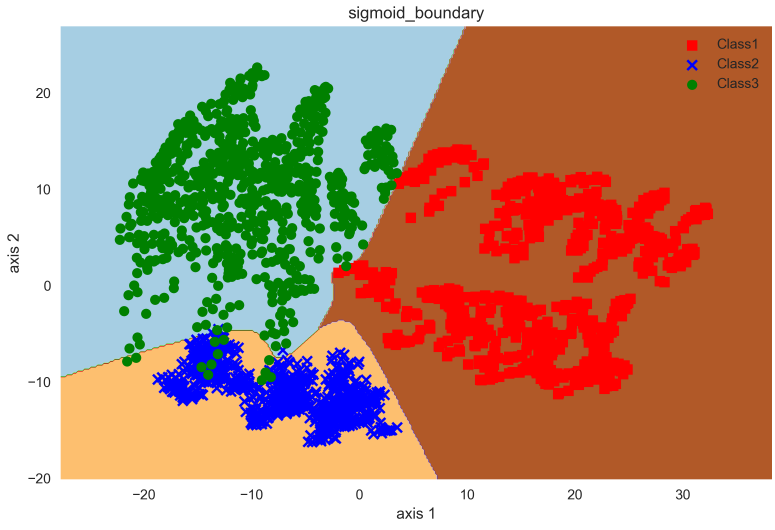
參數設定如下: 300個neuron, # epoch=100, learning rate=0.1, 每經過一個epoch, learning rate降為當前learning rate的95%。training/validation accuracy為99.37%及99.16%, 這數字會隨初始化的不同而有些微變化。而在實驗中, 如果neuron數增加越多, 對於performance並沒有顯著的改善, 反而有時還會有降低的情形, 這有可能是因為有些時候我們其實只需要微調其中幾個neuron就夠了, 全部都調整反而會造成原本調得不錯的neuron被改變, 而neuron數目越多這個情形就會比較嚴重。

(2) 3-layer(2-hidden layer) network using ReLu function

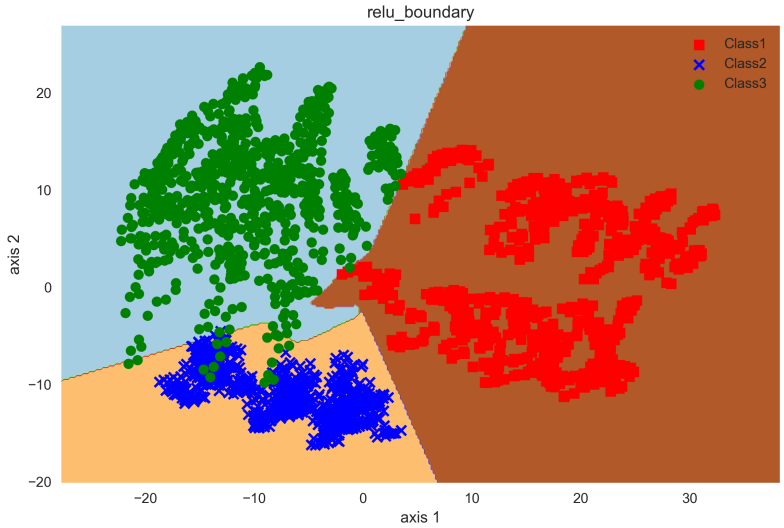
此處使用leaky-relu並設定小於0的斜率為0.01, 參數設定如下: 兩層各25個neuron (2*25+25+25*10+10個參數), # epoch=200, learning rate=0.05, 每經過一

個epoch, learning rate降為當前learning rate的95%, ReLu設定較小的initial learning rate的原因是由於relu在forward pass時, 對於值是沒有上限的, 因此傳到output layer之時可能有overflow的問題, 尤其在每一層的neuron數目越多時越容易產生這個問題, 但也因為如此其在相同的learning rate下, 收斂速度應該是比sigmoid還要快速, 因此在這兩個因素之下, 選擇較小的初始learning rate應是適當的選擇, 而在實驗上的確大的learning rate之收斂速度很緩慢且易發生問題。 training/validation accuracy為99%和99.16%, 同樣地會隨著初始化的不同而變化。在這個問題中, 我嘗試去加深network看看能否得到更好的performance, 但結果是否定的, 以5-layer, 每層25個neuron的model最後得到98.54%和98.33%的結果, 有可能是因為我們的input已經簡化成為2-dimension, 而且2-layer已經有不錯的效果, 導致加深network並不會有太大的改善空間, sigmoid部分加多neuron數而無法改善也可能是這方面的原因, 不過我並沒有花費太多的時間在加深network的部分, 也有可能是hyper-parameter設定不佳所造成。以下是兩個模型的decision boundary。





補充：純粹的relu, 參數設定如下: 兩層各25個neuron, # epoch=200, learning rate=0.01, 每經過一個epoch, learning rate降為當前learning rate的95%。 training/validation accuracy為99.12%和99.16%。



(3) 2-layer(1-hidden layer) network using sigmoid function with mini-batch

mini-batch size為32, 得到train/val = 98.66%和99.12%的結果, 對於training accuracy而言反而下降, 而收斂速度也差不多, 在這個問題上並沒有得到顯著的好處。

(4) 3-layer(2-hidden layer) network using ReLu function with mini-batch

mini-batch size為32, leaky_relu得到99%和99.16%的結果,些微改善一點performance。,而relu則為98.66%和99%, 沒有改善, 但是這兩個在這部分的訓練速度上面都會比SGD還要快, 因為ReLu求微分只要簡單的運算, 比起每次微分後都做一次backpropagation還要省運算成本。

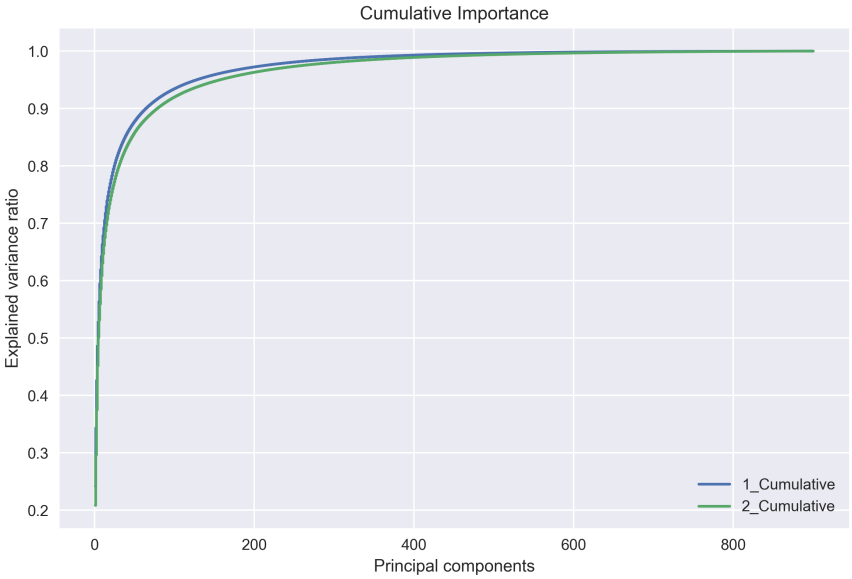
3. Compare

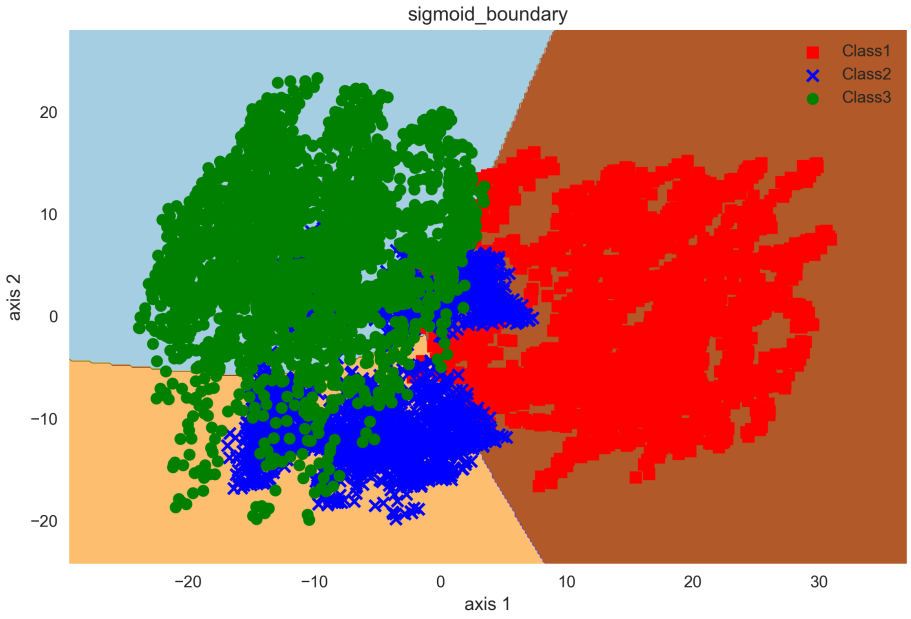
在上次的作業中, probabilistic generative model之training/validation accuracy為95%和97%, 而probabilistic discriminative model之training/validation accuracy則為98.3%及97.3%。雖然結果上來看, 無論是使用relu或是sigmoid在validation的表現上都優於這兩種model, 但是在調整hyper-parameter上比這兩種還要花費時間, 而且訓練時間也較長。利用mini-batch的relu在validation set的表現是最優的, 但是並沒有說明顯打敗其他的模型, 不過mini-batch混合了batch和SGD的好處, 能夠有SGD的跳離saddle point之效果, 也有batch的穩定更新的特性, zig-zag不會那麼嚴重, 及可平行加速的效果, 而且在一些activation function的選取上還能夠進一步的加速, 如上面的ReLu。

4. Other works

另外嘗試使用data augmentation來看看是否能夠進一步的提升對於validation set的performance,每張圖片會做flip一張, 以及一張隨機旋轉-10~-20或10~20度的圖, 因此資料量變為原本的3倍, 照理說資料量變多應該可以讓network訓練的比較好, 但從結果來看, performance是全面性的變差, 在其他條件不變之下, sigmoid變為93.29%及94%; leaky_relu為91.33%及94.5%, 而relu則為91.16%及95%, 一開始覺得很奇怪, 在不調整network結構下, training accuracy的下降是可預期的,

但是validation accuracy變差太多則是我沒有想到的情況, 不過在把圖畫出來之後就很明顯, 由於資料的變異程度變得更高, 因此利用PCA把資料降為二維的行為顯得過於簡化, 右圖為PCA中eigenvector(皆為unit length)的解釋程度(eigenvalue)做normalize後排序並做cumulation, 藍色為未經過data augmentation而綠色則經過data augmentation之處理, 可以看到data augmentation後的資料, 前幾個eigen vector的可解釋程度比較低, 以數據來看, 只用兩個components下, 原始資料的解釋程度有0.343, 而經過data augmentation後只剩下0.296, 這個差異性並不小, 表示有非常多的資料無法經由這兩個維度去完美的解釋, 因此沒辦法提升training/validation accuracy, 反而還會下降, 右圖為sigmoid經data augmentation後的訓練結果,





可以看出overlap的資料點多出很多。結論就是, 如果training data已經足以反映真實分佈, 那麼做data augmentation反而會讓network學到有偏誤的資料, 讓整體的效果變差, 另一個則是在使用降維的方法時, 必須要觀察到底要多少維度才足以解釋整個data set, 太少的解釋性將會造成network無法有效地學習。

5. Explain and compare the following nouns with words

(1) batch gradient descent

即最原始的gradient descent, 其在將所有資料的gradient加起來取平均之後才會做一次更新, 也就是一個epoch只更新一次, 其更新的路徑會很穩定, 但會有卡在saddle points的問題。

(2) mini-batch gradient descent

在SGD和batch間取平衡的結果, 如果都在sequential的寫法之下, SGD會是這三者中更新速度最快而batch則最慢, mini-batch在兩者之間, mini-batch可以像SGD一樣避免卡在saddle points而又能夠保有一定的相batch之穩定更新, 而若在平行化的寫法下, mini-batch中的每個資料都能夠同時計算, 並且比SGD有更少的backpropagation次數, 因此速度可以比SGD還要快, 而且又更為穩定, 所以這三者間, 在hyper-parameter設計良好下應是mini-batch會是比較好的選擇, 兼具速度(SGD), 穩定(batch), 以及較(batch)容易跑到global minimum。

(3) stochastic gradient descent

當mini-batch取1時就是SGD, 其在sequential寫法下會有最快的更新速度, 但是很不穩定, 更新時zig-zag的情形較為嚴重, 不過正因如此才能夠跳脫saddle points。

(4) online gradient descent

基本上跟SGD是一樣的概念, 只不過training data是一直會有新的資訊, 比如像是imdb上面, 當用戶對某部影片評分時就會產生一筆新的training data來訓練network, 也可以累積一定筆數之後再以mini-batch的方式進行訓練。