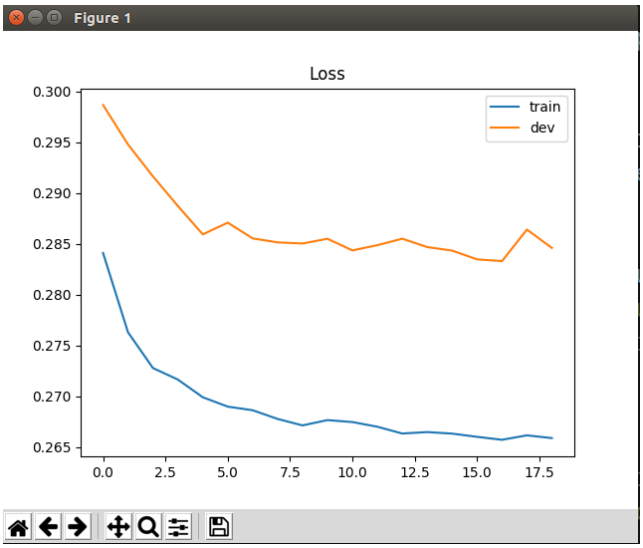
學號：B06901053 系級：電機三 姓名：謝承延

1. (2%) 請比較實作的 generative model 及 logistic regression 的準確率，何者較佳？請解釋為何有這種情況?

Logistic model:

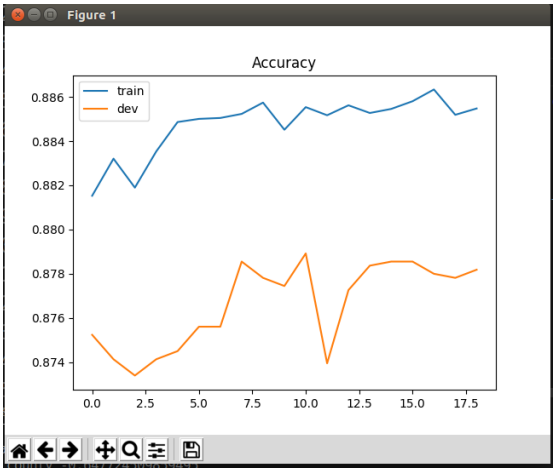
LOSS:

train loss:0.26589720233881575
validation set loss:0.28461453578221163



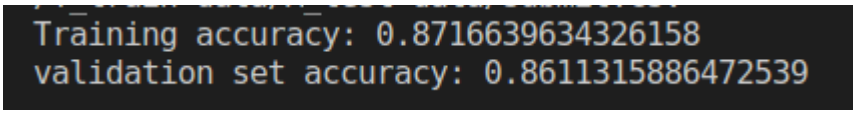
accuracy:

train accuracy: 0.8854802375588777
validation set accuracy:0.8781791374861777



Generative model:

accuracy:



從結果可看見，generative model 的表現較差,原因大概是因為 generative model 預先對這個模型做出了假設，

第一：它假設這個資料分佈是 gaussian distribution,

第二：它假設我們手上的現有 data,能夠完整表達整個 gaussian distribution 的狀態與參數，

然而這兩件事並非一定成立,很有可能這個 dataset 的資料分佈根本不是 gaussian distribution,也很可能這個 dataset 的所有資料剛好都是 gaussian distribution 模型中比較極端的例子,因此算出來的參數根本不能表達完整的 distribution 狀態,綜上所述 generative model 很可能因此有比較差的表現

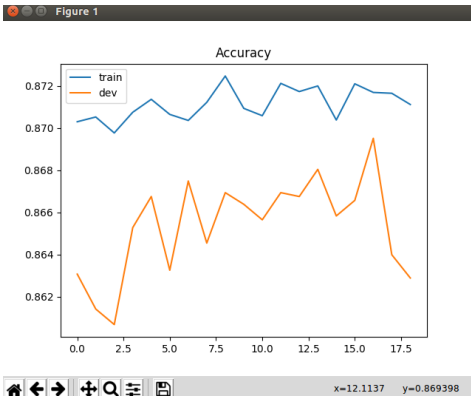
反之 logistic regression 並沒有預先做出如上的假設, 它只設我們可以利用 sigmoid function（z）去預測出結果, 並且利用 w 跟 b 去找出 z， 雖然這樣的想法來自於 gaussian distribution 化簡後的結果, 但是這種方式並沒有假設說 training data 可以代表整個 gaussian distribution 的分佈模式， 因此 可以利用 gradient descent 去找到更好的 z=wx+b 使得結果更精確

2. (2%) 請實作 logistic regression 的正規化 (regularization)，並討論其對於你的模型準確率的影響。接著嘗試對正規項使用不同的權重 (lambda)，並討論其影響。(有官 regularization 請參考 https://goo.gl/SSWGhf p.35)

   關於此題的實踐 我沒有改變 cross entropy loss 的計算方法 而是只對 gradient 的作法做更動,也就是只對 training 階段做更動

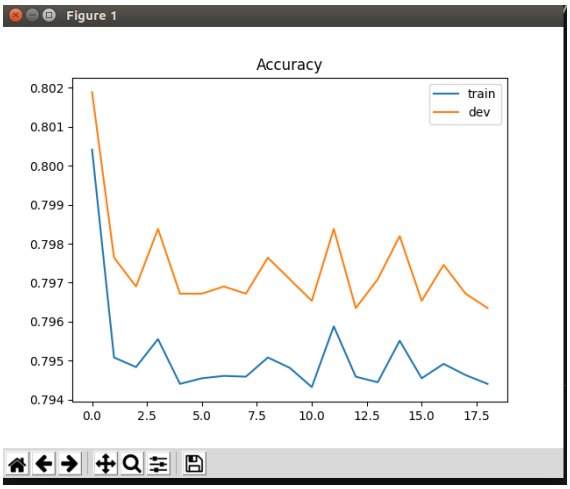
```python
def _gradient(X, Y_label, w, b):
    # This function computes the gradient of cross ent
    lam = 1
    y_pred = _f(X, w, b)
    pred_error = Y_label - y_pred
    w_grad = -np.sum(pred_error * X.T, 1) + 2*lam*w
    b_grad = -np.sum(pred_error)
    return w_grad, b_grad
```

   lambda = 0.5:




```
Training accuracy: 0.871124308826541
Development accuracy: 0.8628824179874678
```

lambda = 10:




```
Training accuracy: 0.794409174687692
Development accuracy: 0.7963509030593439
```

權重使用越大的時候,準確率反而越低,我認為可能的原因是 regularization 會讓 b+wx 這條線變得更加圓滑, 丟到 sigmoid function 之後會讓 sigmoid function 的 output 無法去靈活的對應預測結果（因為 z 的變化幅度變小）, 所以從 train accuracy 看到明顯的下跌

3. (1%) 請說明你實作的 best model，其訓練方式和準確率為何?

我使用 logistic regression model 並加入新的 feature:

1.二次項

2.三次項

3.四次項

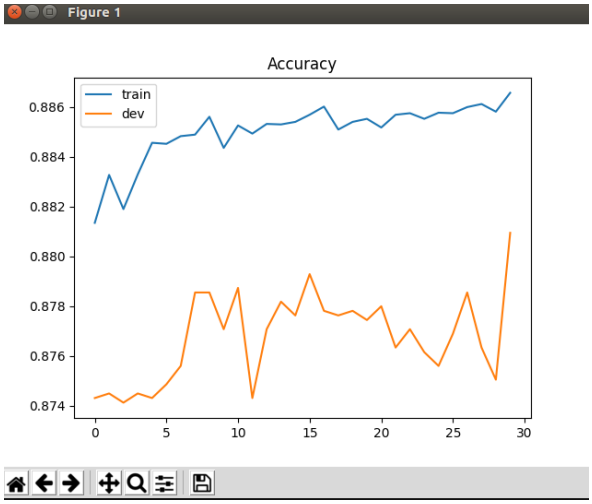
4.一些 feature 互相相乘

5.train validation set 比例為 9：1

6. batch size：8　learning rate:0.1　iteration time:30

```
Training accuracy: 0.8865656358795823
Development accuracy: 0.8809436048654626
```

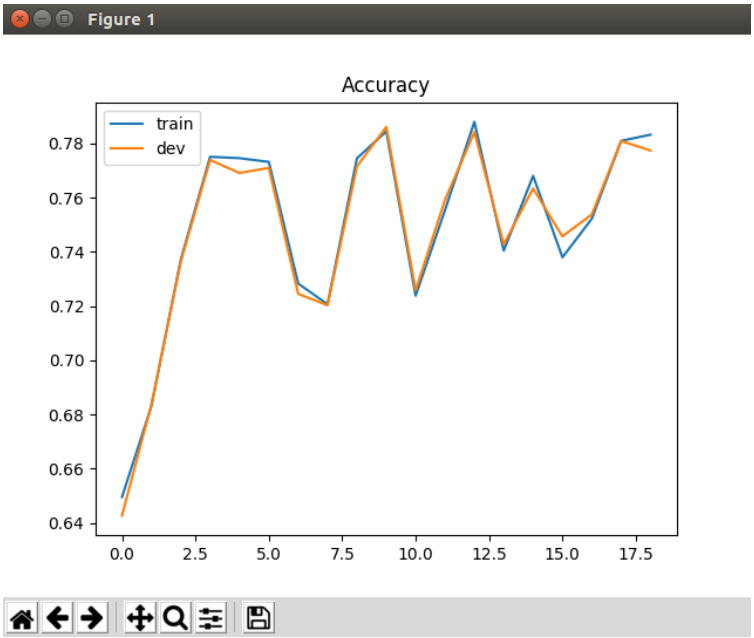
我有試著繼續加入其他交互相乘的 feature，雖然 train 跟 development accuracy 都有上升 但是在 kaggle 上的預測卻沒有變好，有可能是這個 model 過度擬何了 validation set 的 data



在 kaggle public set 最好結果為 0.89016 ,沒過 strong baseline,相信有符合 kaggle HW2 competition 的比賽要求： 別當 xx 人 大家一起爛

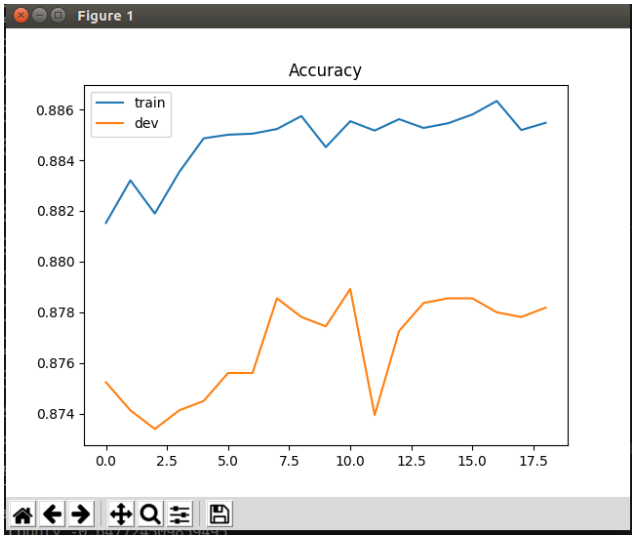
4. (1%) 請實作輸入特徵標準化 (feature normalization)，並比較是否應用此技巧，會對於你的模型有何影響。

Without normalization:




```
Training accuracy: 0.7832070448494778
Development accuracy: 0.7773682270549207
```

with normalization:




```
Training accuracy: 0.8854802375588777
Development accuracy: 0.8781791374861777
```

使用完全相同的 model 的時候 當我進行標準化時,

train accuracy 和 test accuracy 大約會落在 0.87～0.89 之間

若沒有標準化 則掉到了 0.78 左右

我認為原因是來自於這個 training dataset 中, 並不是所有的 feature 都是 discrete value 1 跟 0, 有一些 continuous value 存在 ,也就是說,很有可能因為這些 continuous value 的分佈範圍很大, 或是他們本身的值很大, 因此對預測結果產生過大的影響, 讓其他 feature 無法 發揮作用, 而這些 continuous 的 value 又不足以找到最符合結果的參數, 因此導致了 accuracy 的下滑