ML2017 HW5 Report

B04705003 資工三 林子雋

P1: 請比較有無 normalize(在 rating 上)的差別。並說明如何 normalize. (1%) 請說明你如何 normalize，以及附上 normalize 前後的準確率比較。

模型設定：

- 無加入 Bias
- Latent dimension 為 60
- Batch size 為 1024

Normalize 實作：

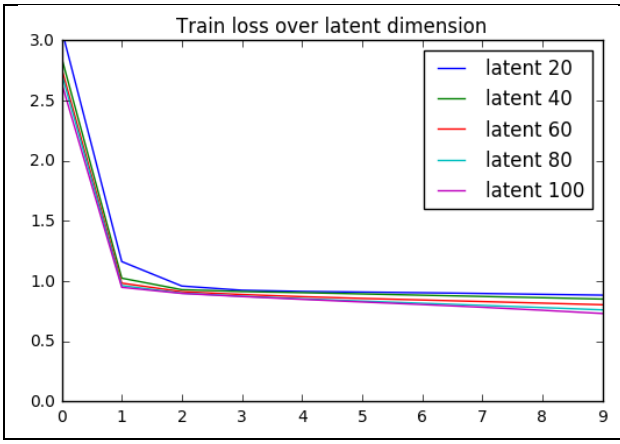
Training 的時候讓模型 fit 上 normalize 過的 label，測試 validation 和 testing set 的時候，將模型預測出來的結果回復至[1,5]之間(有使用 np.clip 將模型預測的結過限制在 1 到 5 之間)

Normalize 方法	RMSE
1.不 normalize	0.857769632829673
2.使用 (xi – min) / (max – min) normalize	0.8553505211916588
3.使用 (xi – mean) / (std) normalize	0.8575428294297883
結論： 使用 Normalize 都會使結果好一些些	

P2: 比較不同的 latent dimension 的結果(1%) 請附上不同 latent dimension 的實驗數據。

模型設定：

- 無加入 Bias
- Batch size 為 1024



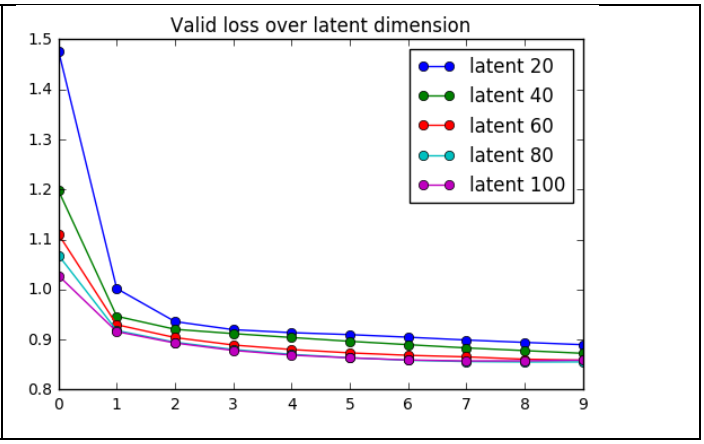


Figure 1. Training set 訓練過程

Figure 2. Validation set 訓練過程

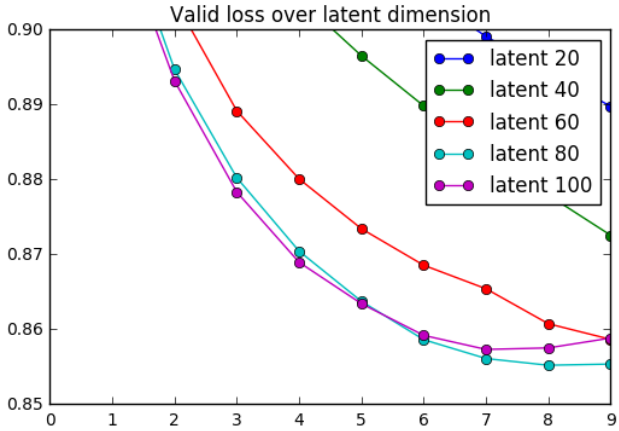


Figure 3. 聚焦 y = [0.85, 0.90] 之間

結論：

結果可以看出，latent dimension 在 80 可以獲得最好的 validation loss，其他的小於 80 的 dimension，validation loss 的值收斂之後比較高，而 latent dimension 為 100 的時候，就不會再比 80 的 valdiation loss 好了。

P3: 比較有無 bias 的結果。(1%) 請附上有無 bias 的實驗數據。

模型設定：

- Latent dimension 為 80
- Batch size 為 1024

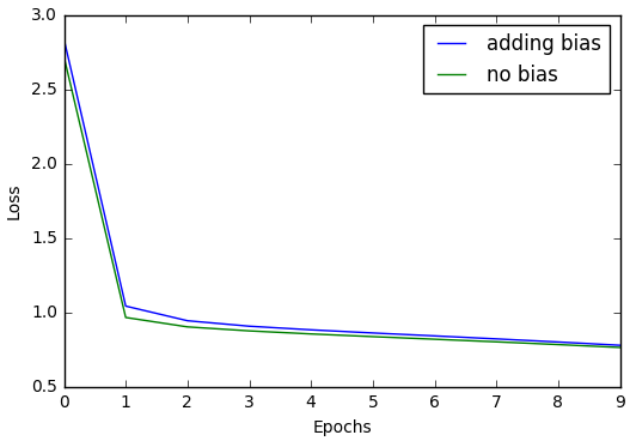


Figure 4. Training set 訓練過程(使用 normal distribution 初始化 bias)

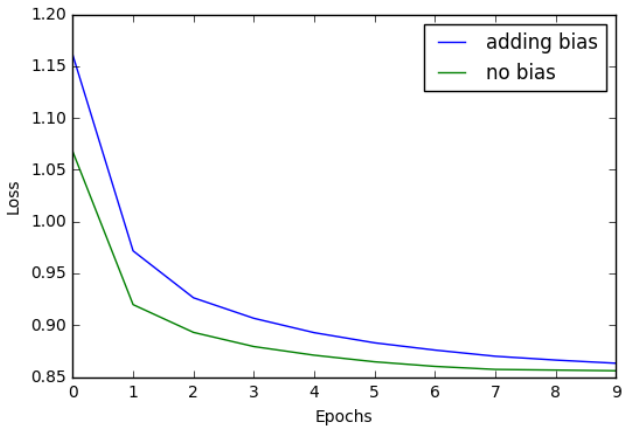


Figure 5. Validation 訓練過程(使用 normal distribution 初始化 bias)

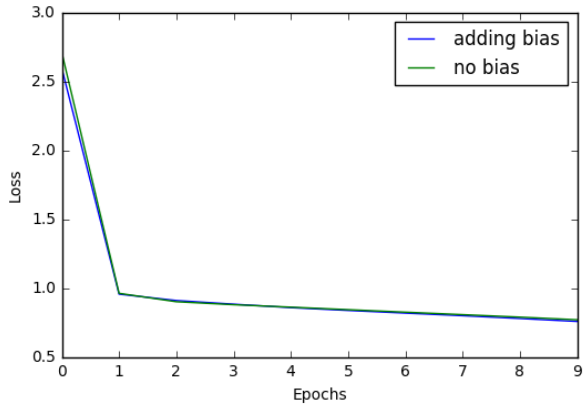


Figure 6. Training set 訓練過程(使用 0 初始化 bias)

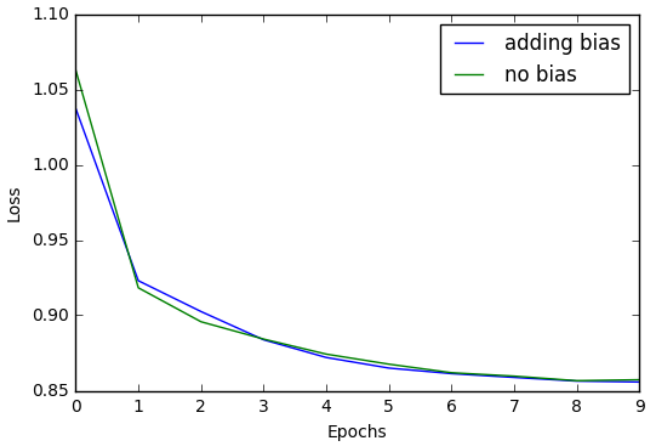


Figure 7. Validation 訓練過程(使用 0 初始化 bias)

結論：

一開始在做這個實驗時，我使用 pytorch nn.Embedding 實作 bias，但做出的結果發現有加 bias 還變比較差是怎麼一回事，後來發現 embedding 的初始化是使用 normal distribution，我就把 bias 改成最常用在 bias 的初始化－使用 0 初始來試試看，結果發現加上 bias loss 是比不加 bias 還要低一些的(如 Figure 7.)，這也間接證明了初始化的重要性。

P4: 請試著用 DNN(投影片 p.28)來解決這個問題，並且說明實做的方法(方法不限)。並比較 MF 和 NN 的結果，討論結果的差異。(1%) 說明實做 DNN 的方法,並且附上不同參數以及架構的實驗數據,並簡單分析原因。

	Matrix Factorization	Deep Neural Nets
模型架構	1. 無 Normalize 2. 無 bias 3. Latent dimension = 80	1. 無 Normalize 2. 有 bias(pytorch nn.Linear 內建的 bias) 3. Latent dimension = 80 4. DNN 架構：

<table>
<tr><td></td><td></td><td>

```
if DNN:
    self.linear = nn.Sequential(
            nn.BatchNorm1d(2*dimension),
            nn.Linear(2*dimension, 100),
            nn.SELU(inplace=True),
            nn.BatchNorm1d(100),
            nn.Linear(100, 50),
            nn.SELU(inplace=True),
            nn.BatchNorm1d(50),
            nn.Linear(50, 25),
            nn.SELU(inplace=True),
            nn.BatchNorm1d(25),
            nn.Linear(25, 1)
        )
    # initial with uniform
    for m in self.linear.modules():
        if isinstance(m, nn.Linear):
            m.weight.data.uniform_(-0.1, 0.1)
users_moives_matrix = torch.cat([users_matrix, movies_matrix], dim=1)
y_ = self.linear(users_moives_matrix)
# (B, 1) -> (B, )
y_ = y_.squeeze(1)
return y_
```

</td></tr>
<tr><td>Loss</td><td>0.8566052119323458</td><td>0.8718520035501256</td></tr>
<tr><td colspan="3">原因分析：
經過 fine tune 之後，DNN 還是略輸給 MF，我認為是其實在這個 task 上，用 MF 來做是比較合理的，因為 user embedding 和 movie embedding 乘在一起的動作，就好像是這兩個資訊在互動一樣，而 DNN 的方法我是把 user 和 movie 兩個 embedding 接在一起乘以多個矩陣，這樣在 user 和 movie 的互動上，明顯少了許多，因此 DNN performance 自然差了些。</td></tr>
</table>

P5: 請試著將 movie 的 embedding 用 tsne 降維後，將 movie category 當作 label 來作圖(如投影片 p.29)。(1%) 請畫出降維後的圖即可。

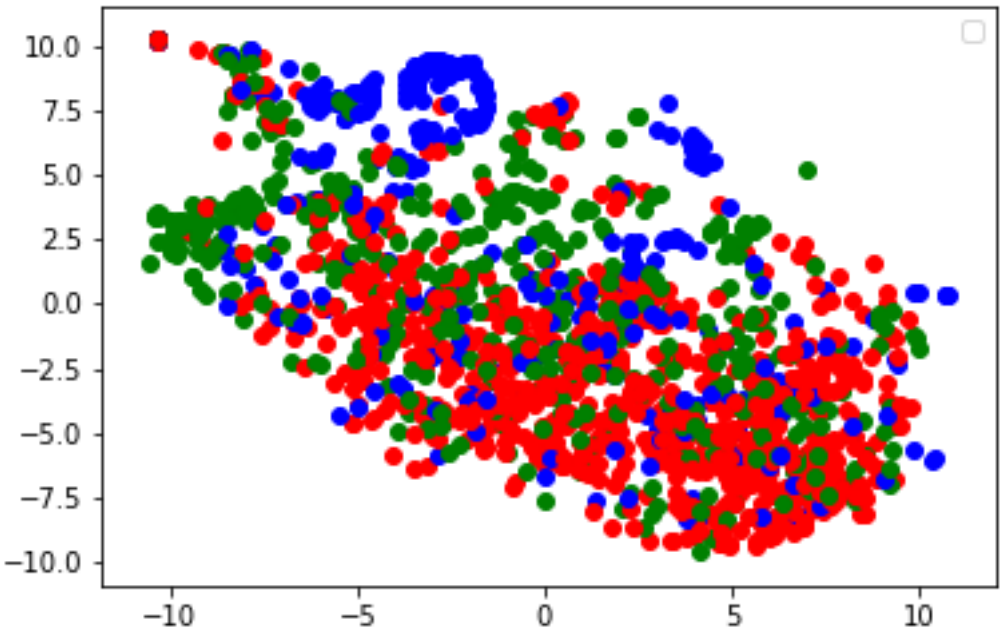


Figure 8. t-SNE 降維 embedding(紅色－["Romance", "Drama"]、綠色－["Comedy", "Children's"]、藍－["Crime", "Sci-Fi", "Thriller", "Horror"])

觀察：

我把電影分成這三類的原因是，我認為 Romance 和 Drama 比較屬於浪漫情境劇，而 Comedy 和 Children’ s 應屬於比較愉快輕鬆，Crime, Sci-Fi, Thriller, Horror 屬於比較陰森神秘類的。

圖中的確可以觀察到群聚的現象，譬如說藍色比較聚集在中上方、綠色是偏左方、紅色是偏右下方，顯示出電影的風格資訊是有多多少少被學進去的。

BONUS: 試著使用除了 rating 以外的 feature, 並說明你的作法和結果，結果好壞不會影響評分。(1%) 請說明你如何使用這些方法，以及其實驗數據

模型架構：	Loss 值
1. 將 movies.csv 的電影風格 one-hot encode 起來，當作一個 augment embedding，在做 matrix factorization 時，先將 one-hot augment embedding 接在 movie embedding 後面並乘上一個矩陣線性轉換到跟 user embedding 同一個維度之後，在用這個新的 embedding 跟 user 做 matrix factorization。(ex. movie embedding + augment embedding 是(Batch_size, latent dimension + 18(電影風格數目)) 線性轉換到	0.8594755062617508

(Batch_size, latent dimension)，用這個新的矩陣跟 user embedding 相乘)

```
movies_aug_matrix = self.movies_aug_embedding(movies_indices)
movies_matrix = torch.cat([movies_matrix, movies_aug_matrix], dim=1)
# bilinear (B, D+A) -> (B, D)
movies_matrix = self.bilinear_drop(self.bilinear(movies_matrix))
```