學號：B03902042　　系級：資工三　　姓名：宋子維

1. (1%) 請比較有無 normalize (rating) 的差別。並說明如何 normalize。

   模型的設定如下：

   - 有加入 user 和 movie 的 bias。
   - latent dimension 為 15。
   - Batch size 為 1024。
   - Validation set 為 training data suffle 後的最後 10%。

   Normalize 方面，將剩下的 training set 的所有 rating 算出平均 μ 和標準差 σ 後，對 training 和 valid set 做標準化，亦即對於 rating r，將其依下列公式進行轉換：

   $$r^\star = \frac{r - \mu}{\sigma}$$

   在訓練時，讓模型 fit 每筆資料的 $r^\star$ 即可。而測試時，對於 prediction $\hat{p}$，必須將其反轉換回去，才能取得預測的 rating $\hat{r}$：

   $$\hat{r} = \sigma\hat{p} + \mu$$

   從 Figure 1可以看出，在 validation set 上，有 normalize 的結果比較好 (RMSE 約低 0.005。)。至於在 public score 上，從 Table 1可以得知，normalize 後也能取得較好的成績。

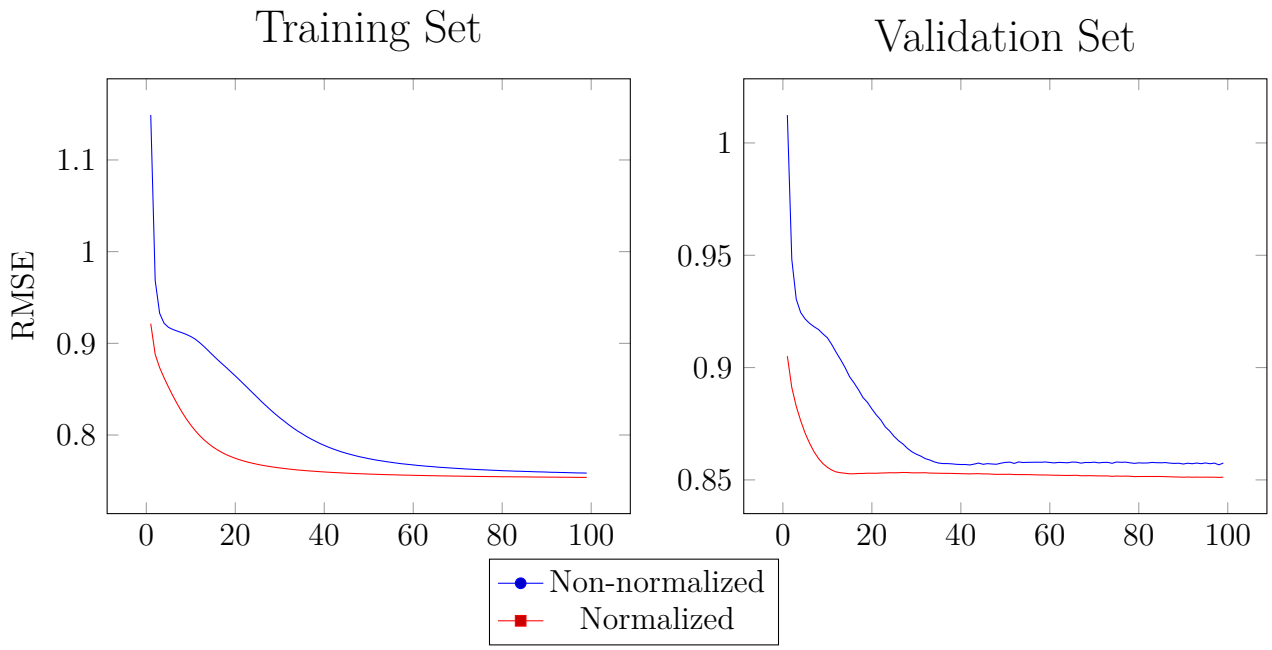

Figure 1: 有無 normalize 的訓練過程。

	Public Score
Non-normalized	0.85074
Normalized	0.84787

Table 1: 有無 normalize 在 public score 上的比較。

2. (1%) 比較不同的 latent dimension 的結果。

模型的設定如下：

- 有加入 user 和 movie 的 bias。
- 沒有 normalize。
- Batch size 為 1024。
- Validation set 為 training data suffle 後的最後 10%。

在這樣的設定之下，我將 latent dimension 分別設置為 5, 15, 30, 45, 60, 75, 90，並訓練 100 個 epoch，觀察其訓練過程。從 Figure 2可以看出，雖然在 training set 上，latent dimension 越大，RMSE 越低，但在 validation set 上則不然，甚至有 latent dimension 越小，RMSE 越低的趨勢。也因此，我最後選擇 15 作為 latent dimension。至於在 training set 上的收斂速度，大致上而言，latent dimension 越小，收斂的越快，我想可能是 latent dimension 較小，要訓練的參數就比較少，因此收斂的較快。

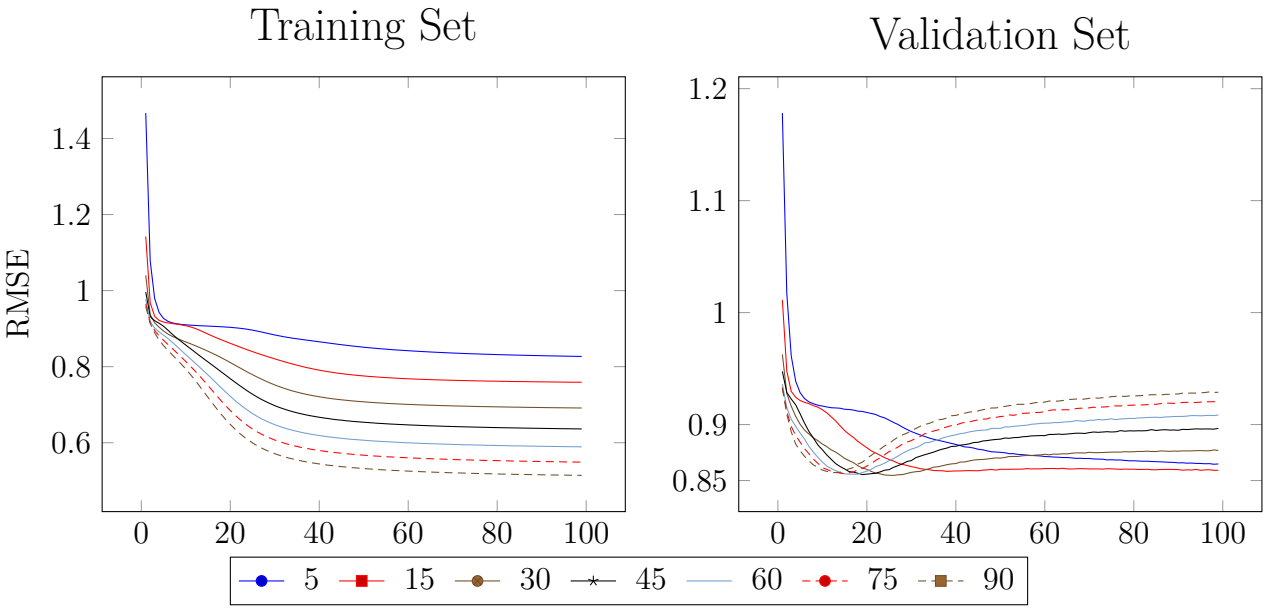

Figure 2: 不同 latent dimension 的訓練過程。

3. (1%) 比較有無 bias 的結果。

   模型的設定如下：

   - 沒有 normalize。
   - latent dimension 為 15。
   - Batch size 為 1024。
   - Validation set 為 training data suffle 後的最後 10%。

   而 Figure 3顯示了有無加入 bias 對 RMSE 的影響，可以看到無論是在 training set 抑或 validation set，有 bias 都略勝沒有 bias 的 MF。猜想可能在這個資料中，每個 user 和 movie 都有不同的 rating 傾向，因此加入 bias 能明顯提昇表現。

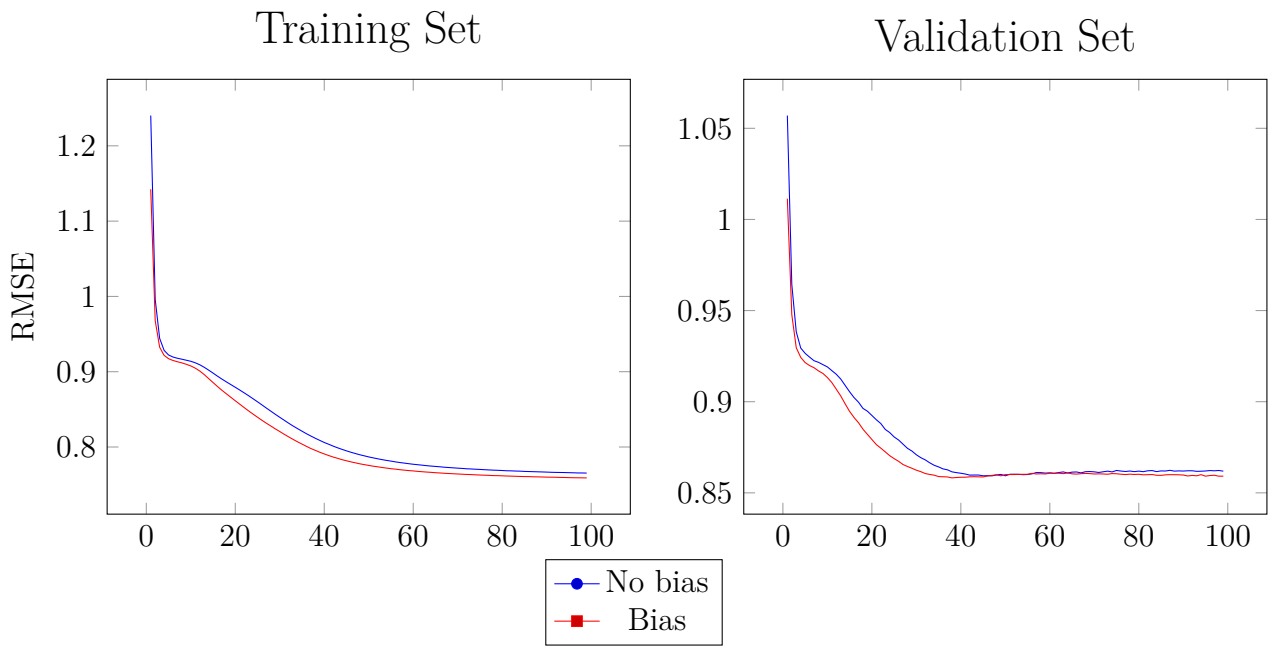

Figure 3: 有無 bias 的訓練過程。

4. (1%) 請試著用 DNN 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。

DNN 的模型也是有一個 user 的 embedding 和 movie 的 embedding，但和 MF 不同的是，DNN 是將兩者 concatenate 在一起，之後經過數層 dense layer 後，輸出單一數值，也就是我將他當作 regression 的問題來解決。而 MF 的模型，我採用的是上一題中有 bias 的版本。詳細的 DNN 模型設定如下：

- 沒有 normalize。
- latent dim 為 256。
- Dense layer 的 output 大小依序為 512、256、128、1。
- Batch size 為 1024。
- Validation set 為 training set suffle 後的最後 10%。

從 Figure 4可以看到，DNN 的模型很容易就在 training set 上 overfit 了，因為 validation set 的 RMSE 約莫在 5 個 epoch 後就不斷高升，而和 MF 相比之下，DNN 最低的 RMSE 也比較差。而依據 Table 2，DNN 在 public score 也較 MF 來得差。

我想 DNN 的結果不如預期，比 MF 差有幾個原因，一是我並沒有仔細調整 DNN 的參數，可能 dense layer 給太大、太多是造成不好結果的原因，二是 latent dimension 的大小，當時想說即使給 DNN 模型較大的 latent dimension，在足夠大小的 dense layer 之下，應該也能萃取出有用的資訊，也因此我使用一個偏大的數字作為 latent dimension，而沒有好好的挑選過。

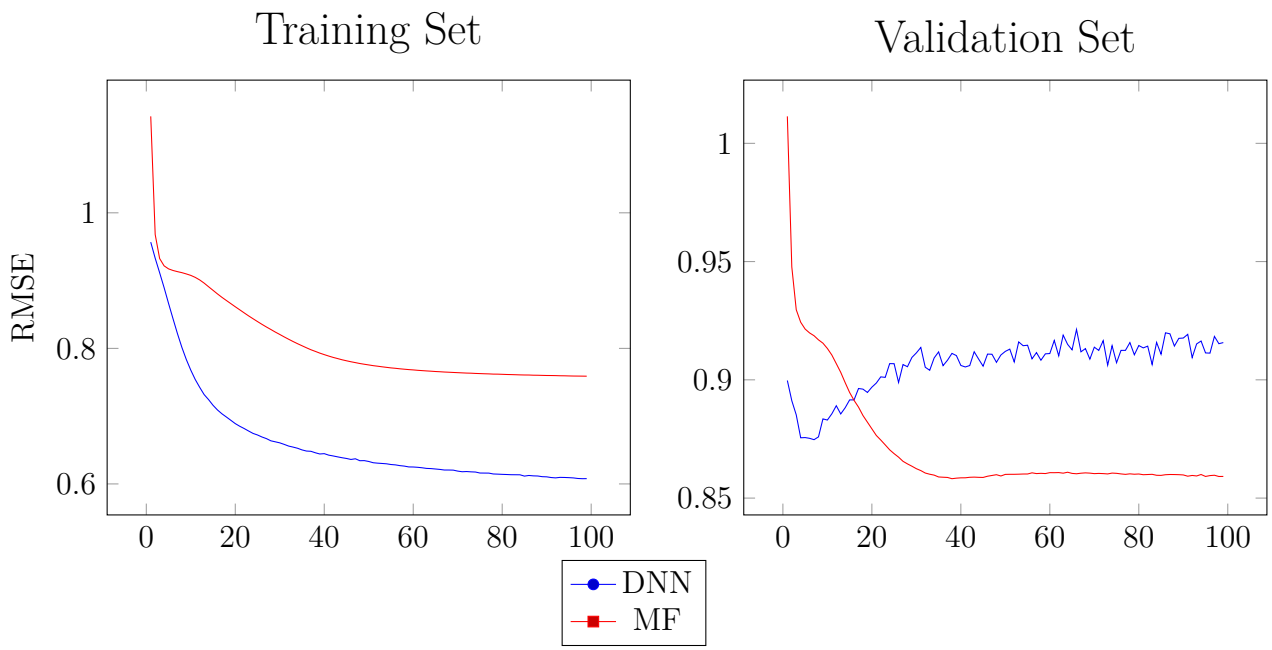

Figure 4: DNN 和 MF 的訓練過程。

	Public Score
MF	0.85074
DNN	0.86427

Table 2: DNN 和 MF 在 public score 上的比較。

5. (1%) 請試著將 movie 的 embedding 用 tsne 降維後，將 movie category 當作 label 來作圖。

   總共有 18 種 movie category，我先透過 `t-SNE` 將 movie 的 embedding 降維，並畫出 18 張 “是 A - 不是 A” 二分類的圖片，藉此觀察 label 的分佈情況。如 Figure 5所示，明顯可以看出 “Children's” 這個類別都聚集在右上角的區塊，於是我以此為基準，並合併數個相似的類別，得到如 Figure 6的圖片，可以發現橘色的 “Animation, Children's, Fantasy” 這些小孩子比較喜歡看的電影類別都聚集在右上方；藍色的 “Action, Sci-Fi, Horror, Thriller, Crime” 這種比較刺激的電影也有明顯的群聚情況；而綠色的 “Drama, Musical” 偏文藝性的電影，則因為數量較多，從右上角一路綿延至左下角，為一個大的群聚。

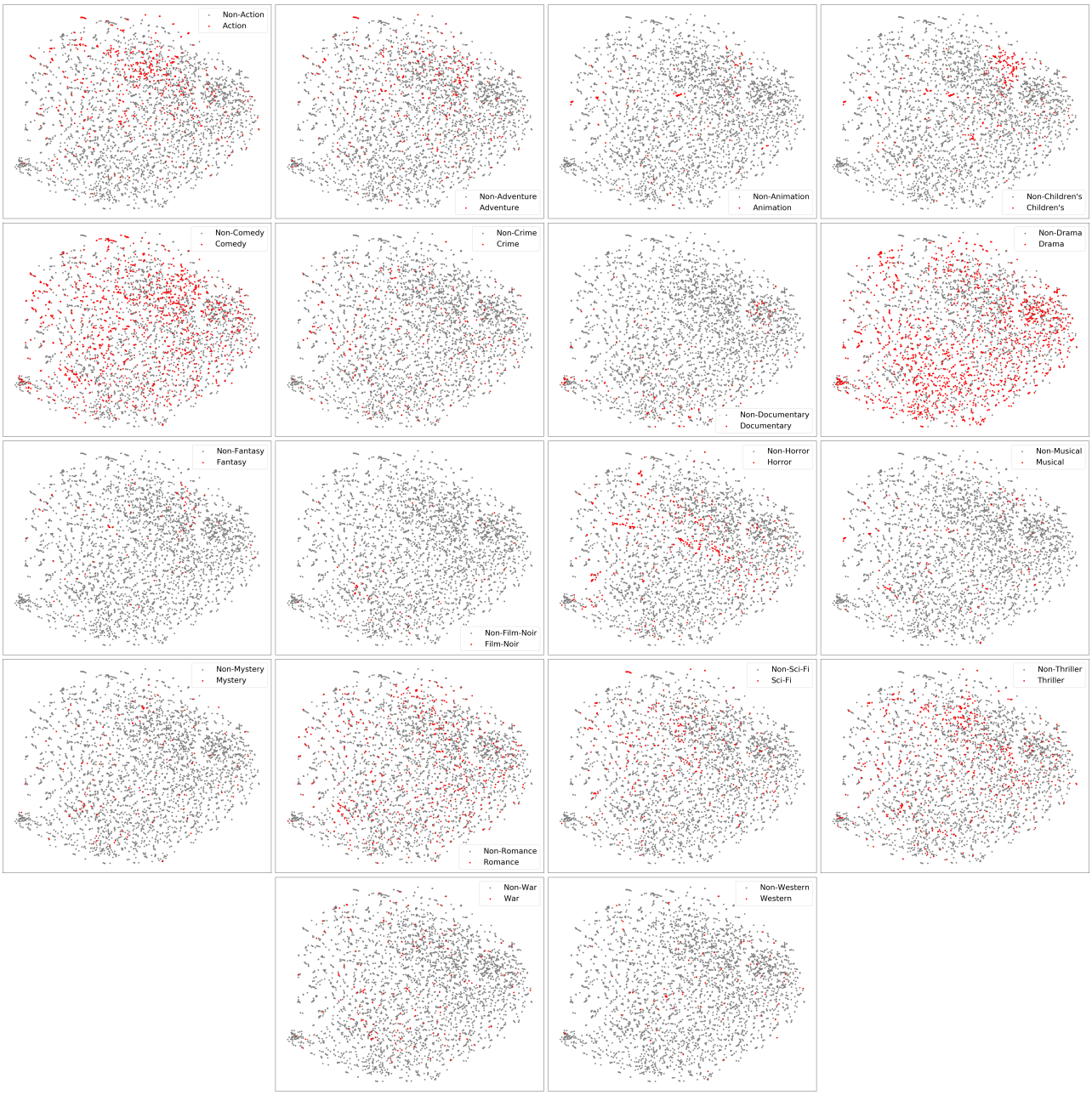

Figure 5: Binary relevance scatter。

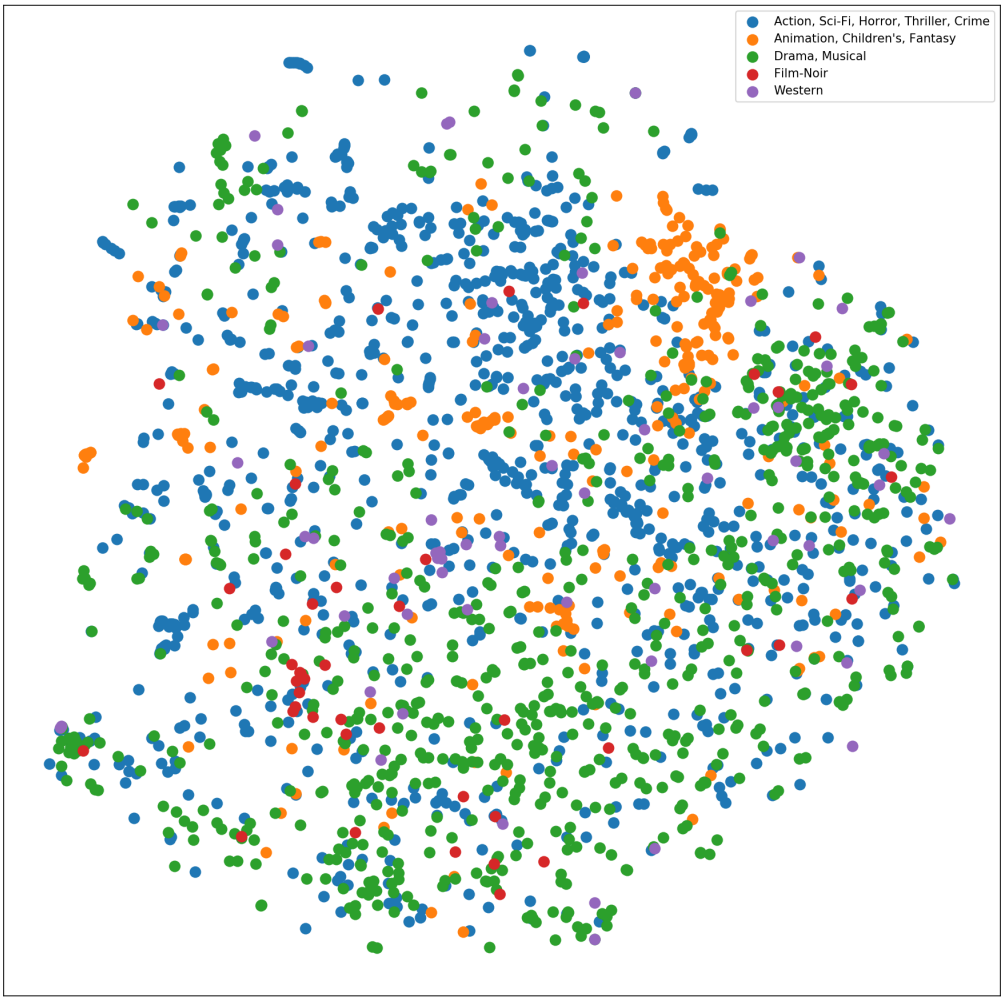

Figure 6: 透過 t-SNE 降維後得到的 movie 散布圖。

6. (BONUS) (1%) 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。

我使用的基本概念一樣是 MF，而額外的 feature 為 “user 曾經評價過的電影”，也就是 SVD++ 算法中的 implicit feedback。

原先所採用的 BiasedMF 算法，也就是 user u 對於 movie i 的 rating r_{ui} 可用下列式子描述：

$$r_{ui} = b_u^{\text{USER}} + b_i^{\text{MOVIE}} + p_u q_i$$

而 SVD++ 算法將其修正為：

$$r_{ui} = \mu + b_u^{\text{USER}} + b_i^{\text{MOVIE}} + (p_u + \frac{1}{N(u)} \sum_{j \in N(u)} y_i) q_i$$

其中的 μ 為所有 rating 的平均 (這樣做其實跟在輸入時對 rating 做標準化有相似之處)，而 $N(u)$ 為 “user u 曾經評價過的電影”，亦即 “user u 的 implicit feedback”，這也是我所使用的額外 feature，至於 y_j 則為一組待訓練的向量。

而訓練方式和 MF 相同，都是透過 Gradient Descent，逐步更新參數。很明顯的，SVD++ 需要更新更多的參數，也需要額外的空間儲存 implicit feedback，因此，需要耗費較大的記憶體空間以及時間才能完成訓練。

從 Figure 7能看出，SVD++ 算法在 validation set 上能取得較低的 RMSE，甚至能低於 0.85。在 public score 上，SVD++ 與 BiasedMF 相比，也能有顯著的進步。

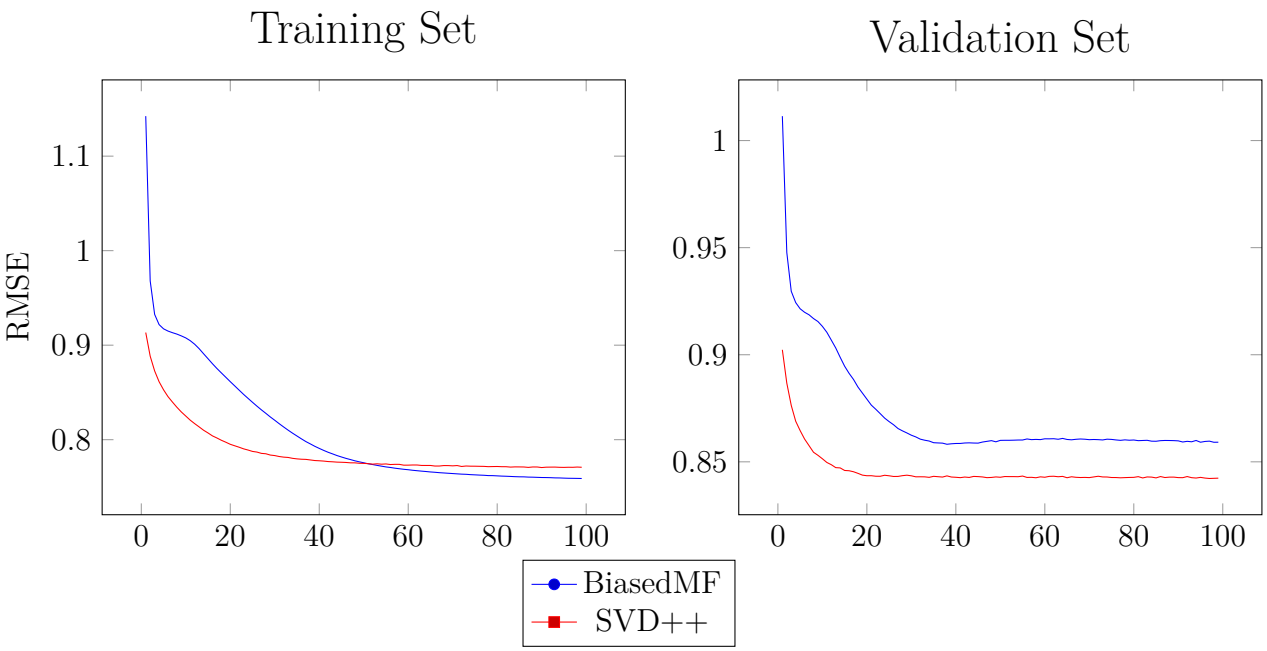

Figure 7: SVD++ 和 BiasedMF 的訓練過程。

	Public Score
BiasedMF	0.85074
SVD++	0.84178

Table 3: SVD++ 和 BiasedMF 在 public score 上的比較。