Programming Assignment 2 - Personal Recommendation System

Name : 林宥廷

Student ID: R13945035

Project Overview

```
R13945035/
├── main.py # entry point
├── model.py
├── train.py
├── preprocessing.py
├── config.py
├── requirements.txt
├── run.sh
├── README.md
├── config/
        │   ├── data.yaml
        │   ├── train_bpr.yaml
        │   ├── train_bce.yaml
```

Config 檔案與參數詳解

存放資料前處理與模型訓練之參數檔

必要檔案包含：

- `data.yaml`：資料前處理參數檔，內涵參數
    - `split_mode`：

        - `"fixed"` ：每個 User 按照固定數量 `val_n` 隨機選取作為 validation
          ❗數量不足
          `val_n` 的 User 會全數做完 training data ，在 validation 時會跳過
        - `"ratio"`：每個 User 按照固定比例 `val_ratio` 隨機選取作為 validation
    - `neg_mode`：
        - `"fixed"`：固定為每個 User 產生 `neg_sample_num` 個負樣本：
        - `"even"`： 按每個 User 的正樣本數量 1 : 1 採樣
    - `val_ratio`： 搭配 `split_mode` "ratio" 使用
    - `val_n` ：搭配 `split_mode` "val_n" 使用
    - `neg_sample_num`：搭配 `neg_mode` "fixed" 使用
- `train_bpr.yaml`：BPR Loss 訓練參數
    - `device`：決定是否使用 CPU 或 GPU 訓練
    - `latent_factors`：MF Model 的 hidden factors 數量
    - `epochs`, `lr`, `batch_size`, `weight_decay`：Model 訓練參數
    - `top_k`：計算 `mAP@` 的數量
    - `margin`：BPR Loss 參數
    - `num_candidate_neg_samples`：Hard Negative Sampling 參數，初步抽樣的候選負樣本數 (N)
    - `num_hard_neg_samples` ：Hard Negative Sampling 參數，從候選中篩選出的 top-k 困難負樣本數 (M)
    - `num_random_neg_samples` :Hard Negative Sampling 參數， 額外加入的隨機負樣本數(K)
- `train_bce.yaml`：BCE Loss 訓練參數
    - `device`：決定是否使用 CPU 或 GPU 訓練
    - `latent_factors`：MF Model 的 hidden factors 數量
    - `epochs`, `lr`, `batch_size`, `weight_decay`：Model 訓練參數
    - `top_k`：計算 `mAP@` 的數量

Quick Start

```
pip install -r requirements.txt
chmod +x ./run.sh
./run.sh -o ./
```

支援參數說明

- `-o`：`required`, 輸出 submission.csv 的儲存路徑
- `-bce`：`store ture`，改啟用 BCE Loss 訓練模型，若未指定參數則預設使用 BPR Loss

Model Factorization

```
class LFM(nn.Module):
    def __init__(self, num_users:int, num_items:int, num_factors:int):
        super(LFM, self).__init__()
        self.user_factors = nn.Embedding(num_users, num_factors)
        self.item_factors = nn.Embedding(num_items, num_factors)

        """Update bias"""
        self.user_biases = nn.Embedding(num_users, 1)
        self.item_biases = nn.Embedding(num_items, 1)
        self.global_bias = nn.Parameter(torch.zeros(1))


        nn.init.normal_(self.user_factors.weight, std=0.01)
        nn.init.normal_(self.item_factors.weight, std=0.01)

        """Update: Init bias"""
        nn.init.zeros_(self.user_biases.weight)
        nn.init.zeros_(self.item_biases.weight)

    def forward(self, user_indices, item_indices):
        user_vecs = self.user_factors(user_indices)
        item_vecs = self.item_factors(item_indices)
```

```
        user_b = self.user_biases(user_indices).squeeze()
        item_b = self.item_biases(item_indices).squeeze()

        dot_product = (user_vecs * item_vecs).sum(dim=1)

        return self.global_bias + user_b + item_b + dot_product
```

- `self.user_factors` : 學習每個 user 的 `num_factors` dim vector
- `self.item_factors` : 學習每個 item 的 `num_factors` dim vector
- `self.user_biases` : 學習每個 user 的潛在偏差
- `self.item_biases` : 學習每個 item 的潛在偏差
- `self.global_bias` : 學習整體平均偏差
- Predict Score : $\hat{score}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^\top \mathbf{q}_i$
  - μ : `global_bias`
  - b_u : `user_biases`
  - b_i : `item_biases`
  - $\mathbf{p}_u^\top \mathbf{q}_i$: user vector 跟 item vector 內積

Train and Validation Splitting

```
def splitting(train_mapping:dict, val_n : int = 5, seed:int = 42 , split_mode:str = "fi
    """
    Random Pick n item for validation
    Paratmeter:
    - train_mapping : `defaultdict(set)` : 型態轉換完成，做完 item id mapping 並且移
    - val_n : 採樣數量
    - split_mode:
        - fixed:
        - ratio:
```

```
    Return:
    - train_pos_dict : defaultdict(list) without dulplicated elements
    - val_pos_dict : defaultdict(set)
    Hint: 所有切分完的 data 都是經過 mapping 的 idx
    """
    print(f"Start Spliting Train/Validation with mode :{split_mode}")
    random.seed(seed)

    train_pos_dict = defaultdict(list)
    val_pos_dict = defaultdict(set)

    for user_idx, item_idx_set in train_mapping.items():
        if split_mode == "fixed":
            if len(item_idx_set) <= val_n:

                train_pos_dict[user_idx] = list(item_idx_set)
            else:
                current_user_items_idx = list(item_idx_set)
                random.shuffle(current_user_items_idx)

                val_items = current_user_items_idx[-val_n:]
                train_items = current_user_items_idx[:-val_n]
                train_pos_dict[user_idx] = train_items

                val_pos_dict[user_idx].update(set(val_items))
        elif split_mode == "ratio":
            val_n = int(len(item_idx_set) * val_ratio)

            if val_n<=0:
                train_pos_dict[user_idx] = list(item_idx_set)
            else:
                current_user_items_idx = list(item_idx_set)
                random.shuffle(current_user_items_idx)

                val_items = current_user_items_idx[-val_n:]
                train_items = current_user_items_idx[:-val_n]
```

```
            train_pos_dict[user_idx] = train_items

            val_pos_dict[user_idx].update(set(val_items))
        else:
            raise ValueError(f"Spliting mode {split_mode} not found, shoud be 'fixed'

    return train_pos_dict, val_pos_dict
```

主要區分為兩種模式

- `split_mode`:
    - `"fixed"` ：每個 User 按照固定數量 `val_n` 隨機選取作為 validation
      ❗數量不足
      `val_n` 的 User 會全數做完 training data，在 validation 時會跳過
    - `"ratio"`：每個 User 按照固定比例 `val_ratio` 隨機選取作為 validation

MF with BCE

Binary Cross Entropy Loss

```
class BCELoss(nn.Module):
    def __init__(self, weight=None):
        super(BCELoss, self).__init__()

    def forward(self, prediction, Label):
        log_inputs = torch.clamp(torch.log(torch.sigmoid(prediction)), min=-100.0)
        log_one_minus_inputs = torch.clamp(torch.log(1 - torch.sigmoid(prediction))


        single_loss = - (Label * log_inputs + (1 - Label) * log_one_minus_inputs)
        final_loss = single_loss.mean()
        return final_loss
```

- Binary Cross Entropy Loss:

$$\text{BCELoss} = -\left[\sum_{(u,i)\in\mathcal{D}^+} \log\sigma\left(u^T i\right) \; + \; \sum_{(u,j)\in\mathcal{D}^-} \log\left(1-\sigma(u^T j)\right)\right]$$

- `prediction` : model 預測的 raw score 即 $u^T i$ ，經 Sigmoid 函數轉換成機率
- `Label` : Sample 的分類， 1 為 positive sample ; 0 為 negative sample
- 對於一個 mini-batch，將所有樣本 loss 取平均：

$$\mathcal{L} = -\frac{1}{N}[\sum_{(u,i)\in\mathcal{D}^+} \log\sigma\left(u^T i\right) \; + \; \sum_{(u,j)\in\mathcal{D}^-} \log\left(1-\sigma(u^T j)\right)$$

- 採用 `torch.clamp` 限制 log 值範圍，避免 $\log(0)$ 產生 `NaN`

Negative Sample Method

Pseudo Code

Random Negative Sampling Module

```
Function generate_bce_samples(train_mapping, train_pos_dict, num_items, neg_

    Initialize item_pool ← {0, 1, ..., num_items - 1}
    Initialize bce_train_data ← empty list
    Initialize RNG with given seed

    For each user_id, pos_items in train_pos_dict:

        For each item in pos_items:
            Add (user_id, item, label=1) to bce_train_data

        If neg_mode == "fixed":
            neg_n ← neg_sample_num
        Else if neg_mode == "even":
            neg_n ← length of pos_items

        interacted_items ← train_mapping[user_id]
```

```
        neg_candidates ← item_pool - interacted_items

        If length of neg_candidates ≤ neg_n:
            sampled_neg_items ← all in neg_candidates
        Else:
            sampled_neg_items ← randomly sample neg_n items from neg_candidate

        For each item in sampled_neg_items:
            Add (user_id, item, label=0) to bce_train_data

    Return bce_train_data
```

Config : `data.yaml`

```
split_mode: "ratio"
val_ratio: 0.1
val_n : 1
neg_mode: "fixed"
neg_sample_num: 3
```

主要區分為兩種模式

- `neg_mode`:
  - `"fixed"`：固定為每個 User 產生 `neg_sample_num` 個負樣本：
    - Example : The User with 15 interaction records and `neg_sample_num = 3`
    - 資料集 : 該User 共18個樣本，其中 15 個為正樣本與 3 個為負樣本，
  - `"even"`： 按每個 User 的正樣本數量 1 : 1 採樣
  - 樣本採集步驟：
    - 移除該 User 互動過的 Item (包含 train/validation)
    - 隨機採樣組合成負樣本資料集 `( user, negative item, 0 )`
- `neg_sample_num`：使用於 `"fixed"` 模式，決定為每個正樣本採樣多少個負樣本

Parameters

`data.yaml`：資料前處理的參數

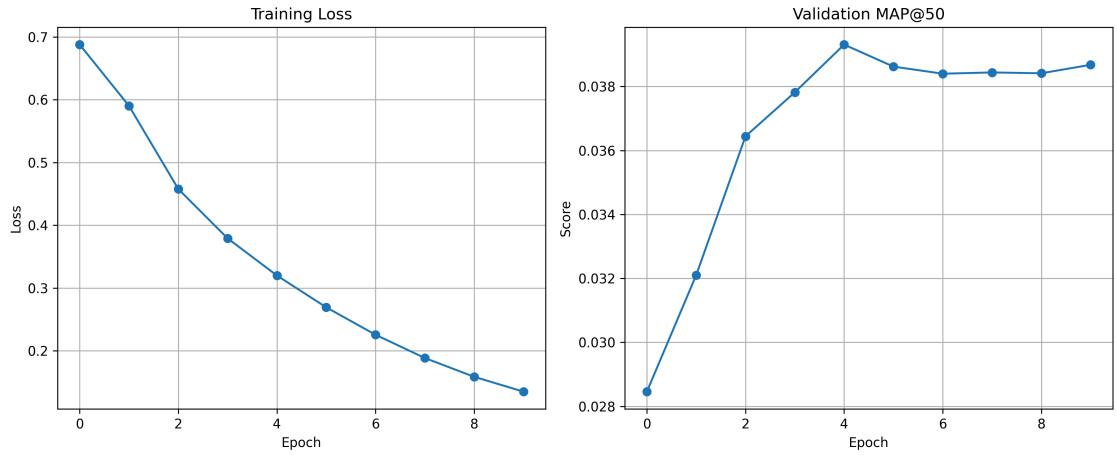
```
split_mode: "ratio"
val_ratio: 0.1
val_n : 1
neg_mode: "even"
neg_sample_num: 600
```

`train_bce.yaml`：模型訓練參數

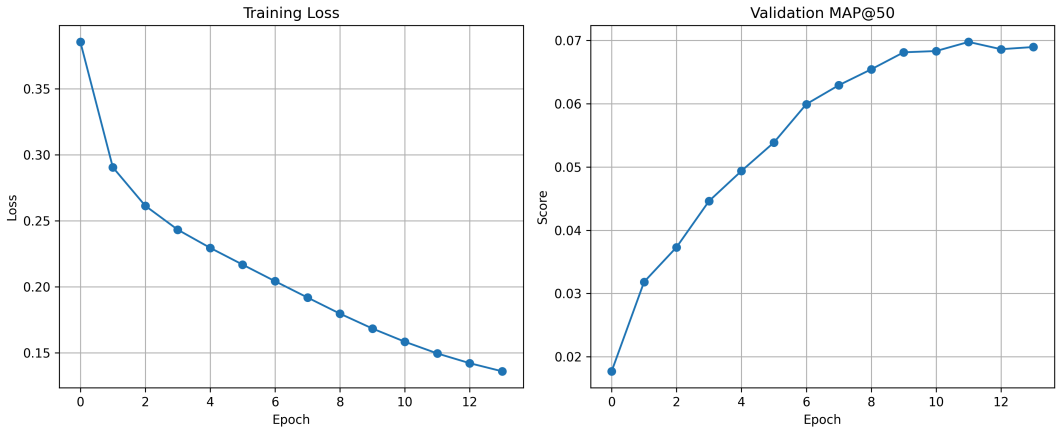
```
device : "cpu"
latent_factors : 400
epochs: 20
lr : 0.001
batch_size : 512
weight_decay: 0.00001
top_k : 50
```

Experiment Result:

- `split_mode`: "ratio" (按照比例 `val_ratio` 切分)
- `val_ratio`: 0.1
- 訓練參數：如上 `train_bce.yaml`
- `neg_mode`: "even" (按每個 User 的正樣本數量 1 : 1 採樣)
  - Result :
    - Best mAP@50 in local validation dataset: 0.0393



- `neg_mode`: "Fixed" (固定為每個 User 產生 `neg_sample_num = 600` 個負樣本)
  - Result :
    - Best mAP@50 in local validation dataset: 0.0698



MF with BPR

Bayesian Personalized Ranking Loss

```
class BPRLoss(nn.Module):
    """
    Parameters:
    - margin: float, 拉高 pos item score
```

```
    """
    def __init__(self, margin = 1.0):
        super(BPRLoss, self).__init__()
        self.margin = margin

    def forward(self, pos_score, neg_score):
        """
        Parameters:
        - pos_score: tensor [batch_size]
        - neg_score: tensor [batch_size]
        """
        diff = pos_score - neg_score - self.margin
        log_likelihood = F.logsigmoid(diff)


        final_loss = -log_likelihood.mean()

        return final_loss
```

- Bayesian Personalized Ranking Loss:

  $$\text{BPRLoss} = \sum_{(u,i,j)\in\mathcal{D}} \ln \sigma(u^T i - u^T j) + \lambda\|\Theta\|^2$$

- `margin`: 是一個可選的參數，負責拉開正負樣本距離：$u^T i - u^T j - \gamma$，若 margin 設為 0，則退化為傳統 BPR。
- `F.logsigmoid`：等價於 $\log(\sigma(x))$
- **負號 (-)**：將最大化 log likelihood 轉換為最小化損失
- Weight Decay 項 : $\lambda\|\Theta\|^2$ 則由 Pytorch optimizer 實現

Negative Sample Method

除了與 MF with BCE 相同的 Random Negative Sampling Module 外，還加入了 Hard Negative Sampling 的額外方法如下：

Hard Negative Sampling

- Introduction : 所謂 Hard Negative Sample 指的是 Model 認為高分，但實際上是負樣本的樣本特別挑出做訓練，加強模型對這些困難樣本的識別能力。並搭配 Random Negative Sampling 對廣泛的樣本池做挑選綜合使用
- Method : 每個 epoch 針對當前 Model 狀態動態抽樣 hardest negatives
  - 針對每個 User 篩選出潛在 negative sample pools (移除該User 的正樣本)
  - 隨機挑選 N 個進行評分後做降冪排列
  - 針對分數做降冪排列，挑選前 M 個高分的負樣本
  - 然後再從 negative sample pools 排除 M 個高分負樣本後隨機挑選 K 個隨機負樣本
  - 將 M+K 個困難/隨機負樣本 與正樣本組成訓練資料 `(user, positive item, negative item)`
  - 每個 batch 中的每個 user 的正樣本都經過此動態抽取後進行訓練
- Pseudo Code :

```
Function hard_negative_sampling(users, train_mapping, num_items, model,
                                num_candidate_neg_samples, num_hard_neg_samples, n

    item_pool ← {0, 1, ..., num_items - 1}
    batch_size ← number of users
    hard_neg_items_batch ← empty list
    random_neg_items_batch ← empty list

    For each user in users:
        neg_candidates ← item_pool - train_mapping[user]
        candidate_neg_samples ← randomly sample M items from neg_candidat
        scores ← model.predict(user, candidate_neg_samples)
        hard_negatives ← select top-K items from candidate_neg_samples base
        add hard_negatives to hard_neg_items_batch
        if num_random_neg_samples > 0:
            remaining_candidates ← neg_candidates - set(hard_negatives)
            random_negatives ← randomly sample R items from remaining_candic
            add random_negatives to random_neg_items_batch
```

```
    if num_random_neg_samples > 0:
        final_neg_items_batch ← concatenate hard_neg_items_batch and rando
    else:
        final_neg_items_batch ← hard_neg_items_batch

    Return final_neg_items_batch
```

Random Negative Sampling

Parameters

`data.yaml` : 資料前處理的參數

```
split_mode: "ratio"
val_ratio: 0.1
val_n : 1
neg_mode: "even"
neg_sample_num: 3
```

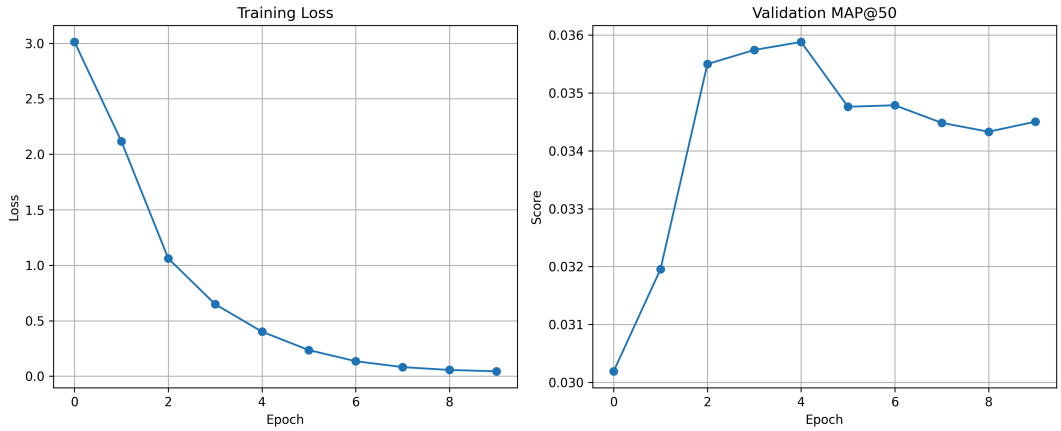
`train_bpr.yaml` : 模型訓練參數

```
device : "cpu"
latent_factors : 400
epochs: 20
lr : 0.002
batch_size : 512
weight_decay: 0.00001
top_k : 50
margin: 3
```

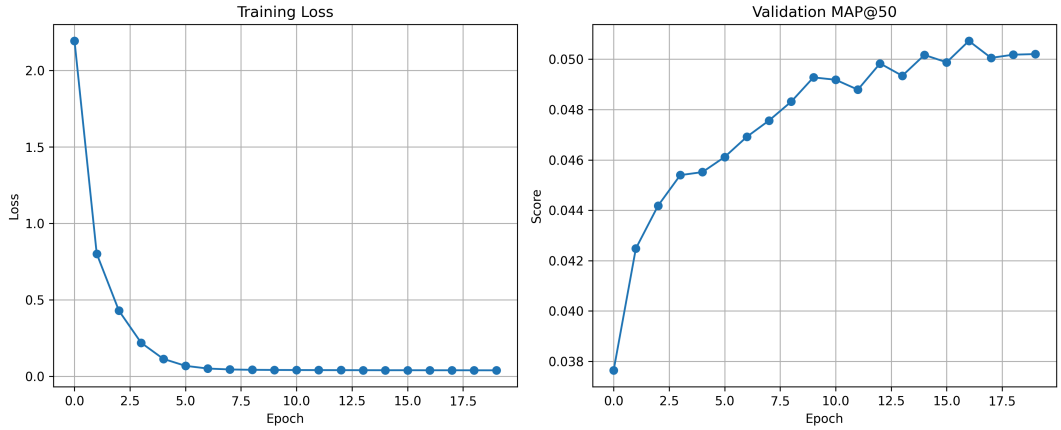
Experiment Result:

- `split_mode` : "ratio" (按照比例 `val_ratio` 切分)
- `val_ratio` : 0.1
- 訓練參數：如上 `train.yaml`

- `neg_mode`: "even" (按每個 User 的正樣本數量 1 : 1 採樣)
  - Result :
    - Best mAP@50 in local validation dataset: 0.0359



- `neg_mode`: "fixed" (每個 User 的正樣本採樣 `neg_sample_num = 3` 個負樣本)
  - Result :
    - Best mAP@50 in local validation dataset: 0.0507



Hard Negative Sampling

Parameters

`data.yaml`：資料前處理的參數

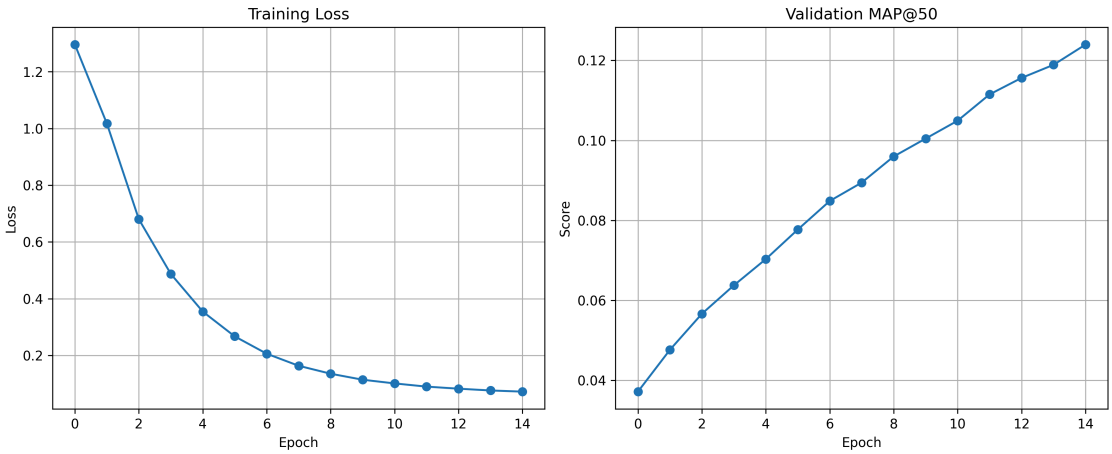
```
split_mode: "ratio"
val_ratio: 0.1
val_n : 1
neg_mode: "fixed"
neg_sample_num: 3
```

`train_bpr.yaml`: 模型訓練參數

```
device : "cpu"
latent_factors : 400
epochs: 20
lr : 0.002
batch_size : 512
weight_decay: 0.00001
top_k : 50
margin: 3
num_candidate_neg_samples : 10
num_hard_neg_samples  : 3
num_random_neg_samples : 3
```

Experiment Result:

- `split_mode`: "ratio" (按照比例 `val_ratio = 0.1` 切分)
- 訓練參數：如上 `train_bpr.yaml`
  - `num_candidate_neg_samples`：初步抽樣的候選負樣本數 (N)
  - `num_hard_neg_samples` ：從候選中篩選出的 top-k 困難負樣本數 (M)
  - `num_random_neg_samples` ：額外加入的隨機負樣本數(K)
- Result :
  - Best mAP@50 in local validation dataset: 0.1239



BCE vs BPR

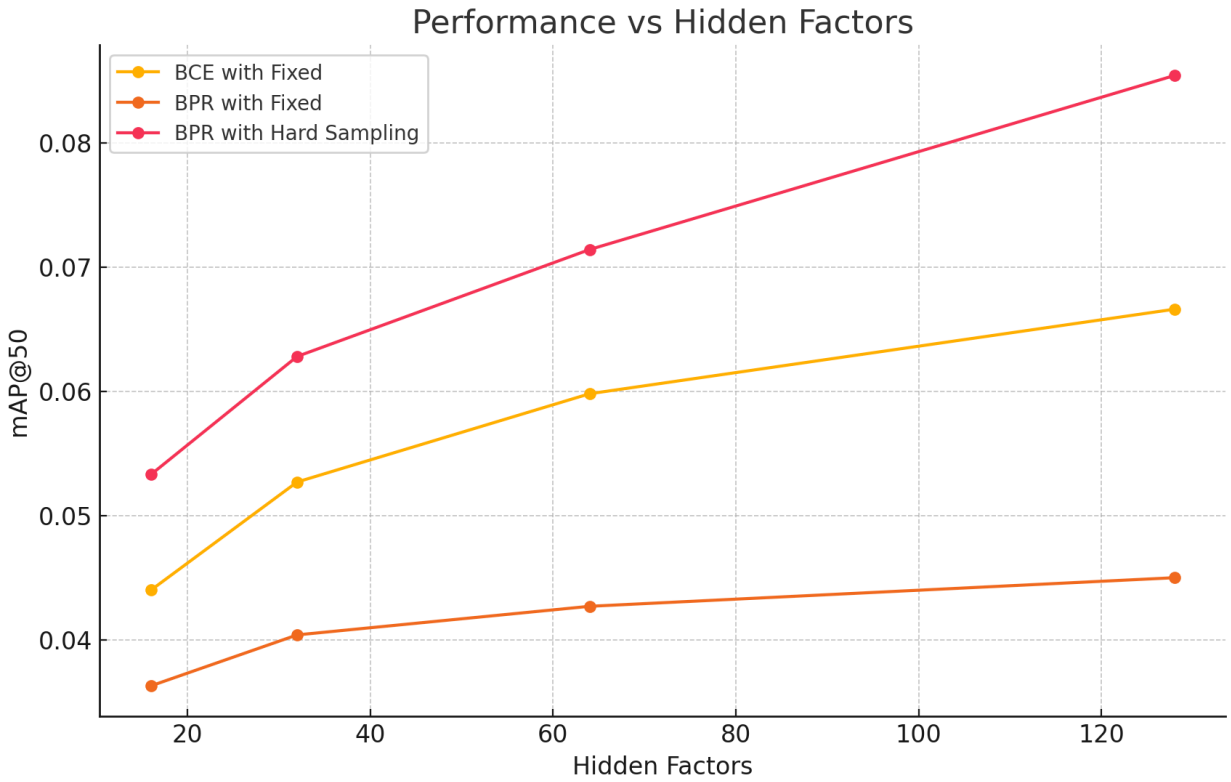
- Validation mAP@50 Performance
  - BCE Loss
    - 隨機一比一採樣 (even)：mAP@50 = **0.0393**
      - BCE training samples: 102738
    - 固定負樣本 600 個 (fixed) : mAP@50 =0.0698
      - BCE training samples: 465369
  - BPR Loss
    - 隨機一比一採樣 (even)：mAP@50 = **0.0359**
      - BPR training samples: 51369
    - 固定負樣本 3 個 (fixed)：mAP@50 = **0.0507**
      - BPR training samples: 154107
    - 困難負樣本 + 隨機負樣本採樣： mAP@50 = 0.1239
- Analysis:
  - 在最簡單的隨機一比一採樣 BPR 表現並未超過 BCE，盡管預期 BPR 表現應該要較 BCE 表現更佳，但結果不如預期，推測可能在於負樣本的採集品質和數量項更加敏感，在 training sample 數目和結果上也有相同趨勢：如 BPR Loss 在增加 training sample 數量 (fixed mode)後 mAP 顯著提升。進一步分析 BCE 在大

幅度增加 training sample 數量 (Fixed mode, training sample 46 萬筆) 後 mAP 也有顯著提升的狀況。所以在隨機採樣下 BPR 的優勢相較 BCE 不太明顯。

- 在對 BPR 負採樣方式採用更精細的困難負採樣 + 隨機負採樣策略後 mAP 有大幅的提升。顯示 BPR 在 pair-wise ranking 天然具備的優勢，可以學習 User 潛在的偏好，學習 user 間個性化的差異，直接地優化排序。
- 所以總結來說 BPR 在排序方面，以方法設計而言相較BCE還是更具備優勢，但是會受限於 negative sampling 的品質與數量。

MAP curve for different hidden factors

Hidden Factors	BCE with Fixed	BPR with Fixed	BPR with Hard Sampling
16	0.0440	0.0363	0.0533
32	0.0527	0.0404	0.0628
64	0.0598	0.0427	0.0714
128	0.0666	0.0450	0.0854



Analysis

- 隨著 Hidden Factors 增加，三種 model 的 mAP@50 都持續提升，表明提升 Model Dimension 能捕捉到更多潛在的 user-item 的資訊
- 不過當 Hidden Factors 增加到一定數量級時 mAP@50 提升開始放緩，推測當 Model Dimension 提升時，雖然能捕捉到更多資訊的同時也引入了更多的雜訊。
- 額外觀察結果是 BPR with Hard Sampling 在 Hidden Factors 增加時 mAP@50 提升放緩的速度較其他兩者更緩慢，顯示更精細的 Negative Sampling 策略的確為 Model 訓練帶來更多的資訊