Neural Collaborative Filtering

X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua
World Wide Web (WWW '17)

Kook Univ.
홍세원

Index

Introduction

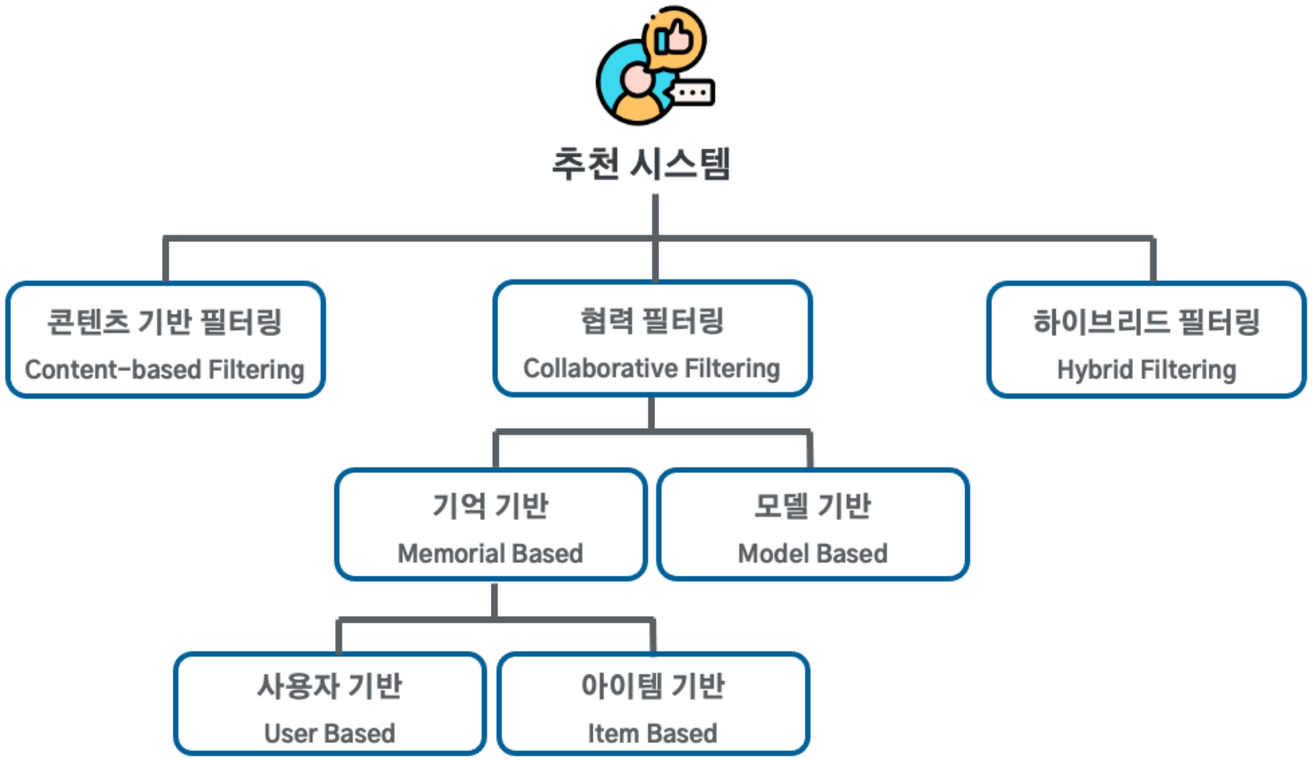
Introduction

추천시스템이란?

- 사용자에 대해 이해하도록 도와주는 정보 통하여
  사용자가 관심을 보일 다른 아이템을 추천해주는 것

Introduction

추천시스템 종류

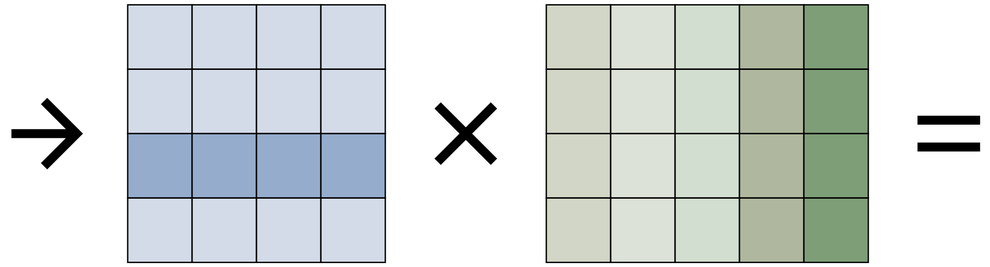


Introduction

기존 Matrix Fatorization

- 사용자와 아이템 사이에는 사용자의 행동과 평점에 영향을 끼치는 잠재적 특성이 있을 것이다’ 라는 가정
- 사용자와 아이템 사이의 계산을 내적연산으로 점수 예측

	A	B	C	D	E
사용자 1	1	1	1	?	1
사용자 2	1	1	1	?	1
사용자 3	1	?	1	1	?
사용자 4	1	?	?	1	1

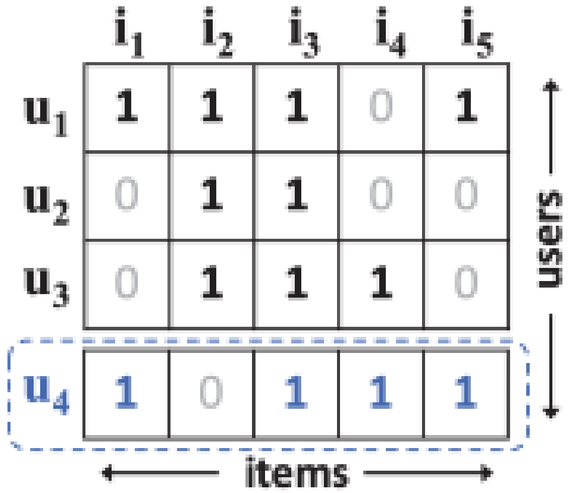


	A	B	C	D	E
사용자 1	1	1	1	0	1
사용자 2	1	1	1	1	1
사용자 3	1	1	1	1	0
사용자 4	1	0	1	1	1

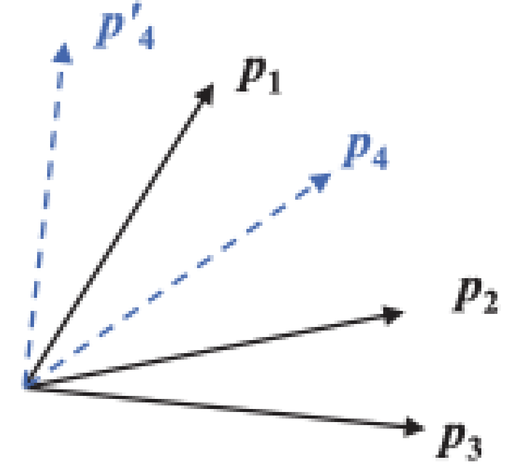
Introduction

기존 Matrix Fatorization 한계

- 내적과 같은 linear 모델은 사용자와 아이템 간의 복잡한 관계를 표현하는 데 한계가 있음



(a) user–item matrix



(b) user latent space

$$s_{23}(0.66) > s_{12}(0.5) > s_{13}(0.4)$$

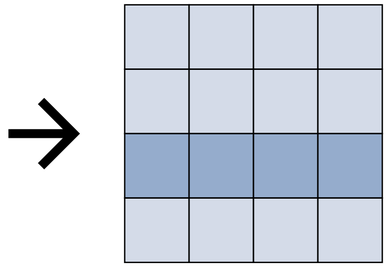
$$s_{41}(0.6) > s_{43}(0.4) > s_{42}(0.2)$$

Introduction

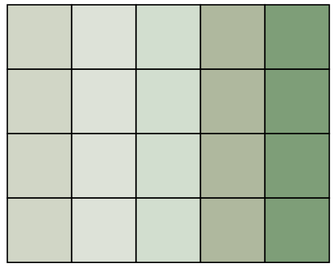
Neural Collaborative Filtering

- 내적 연산을 데이터로부터 임의의 함수를 학습할 수 있는 뉴럴 네트워크 구조로 대체

	A	B	C	D	E
사용자 1	1	1	1	?	1
사용자 2	1	1	1	?	1
사용자 3	1	?	1	1	?
사용자 4	1	?	?	1	1



Neural



=

	A	B	C	D	E
사용자 1	1	1	1	0	1
사용자 2	1	1	1	1	1
사용자 3	1	1	1	1	0
사용자 4	1	0	1	1	1

Method

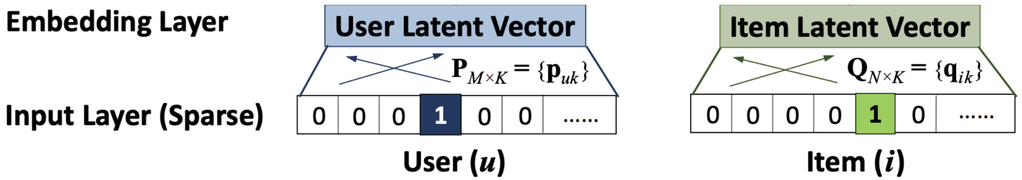
Method - MLP

과정

- 사용자, 아이템 One-Hot 임베딩을 Dense vector 로 변환

UserEmbeddingMatrix : $P_{M \times K}$
ItemEmbeddingMatrix : $Q_{N \times K}$

UserEmbedding : p_{uk} (1 x k)
ItemEmbedding : q_{ik} (1 x k)



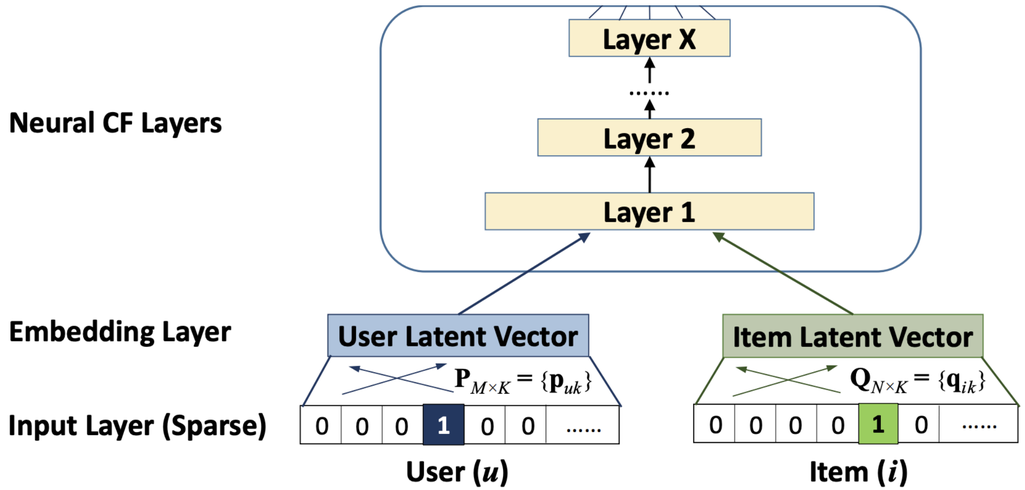
Method - MLP

과정

- 사용자, 아이템 Dense vector를 Concat 후 MLP forward

UserEmbedding : p_{uk} (1 x k)
ItemEmbedding : q_{ik} (1 x k)

ConcatEmbedding : $p_{uk} + q_{ik}$



Method - MLP

과정

- 예측값과 실제값의 차이를 통해 파라미터 학습

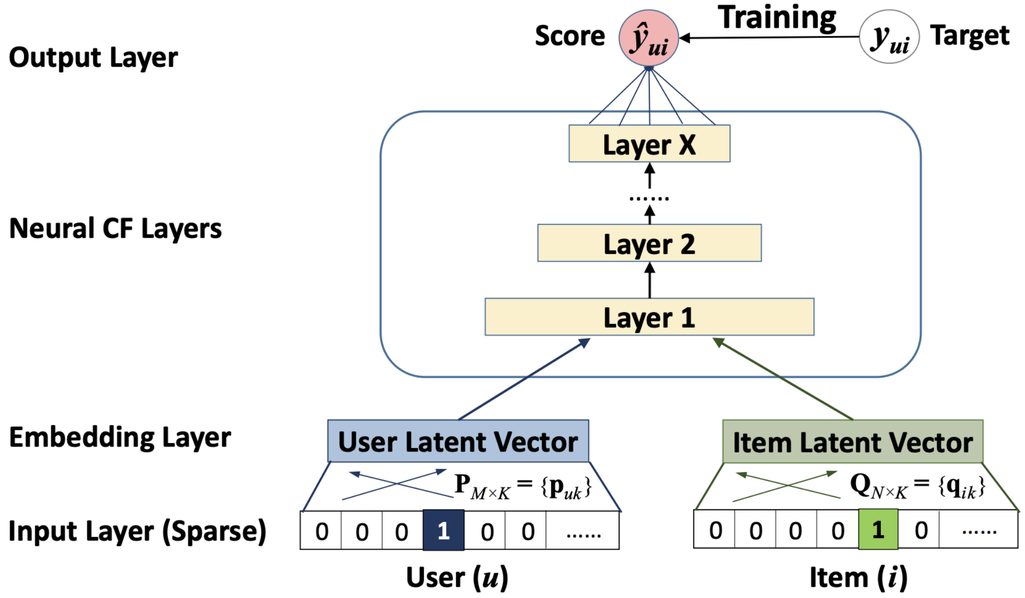
$$L = -\sum_{(u,i)\in Y\cup Y^-} y_{ui}\log\hat{y}_{ui} + (1-y_{ui})\log(1-\hat{y}_{ui})$$

Y : 관찰된 상호작용 집합

Y^- : 관찰되지 않은 상호작용 집합

y_{ui} : 사용자와 아이템간의 실제 상호작용 여부

$\hat{y}_{ui}$: 모델이 예측한, 사용자가 아이템과 상호작용할 확률



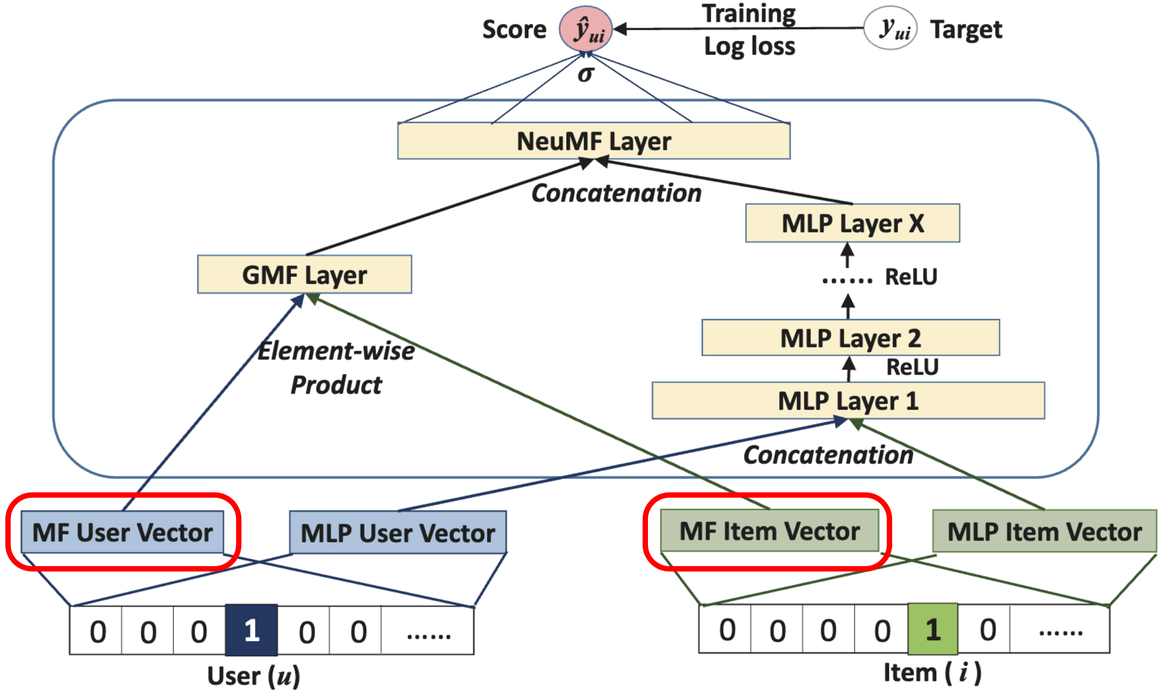
Method - GMF

과정

- 사용자, 아이템 One-Hot 임베딩을 Dense vector 로 변환

UserEmbeddingMatrix : $P_{M \times K}$
ItemEmbeddingMatrix : $Q_{N \times K}$

UserEmbedding : p_{uk} (1 x k)
ItemEmbedding : q_{ik} (1 x k)



Method - GMF

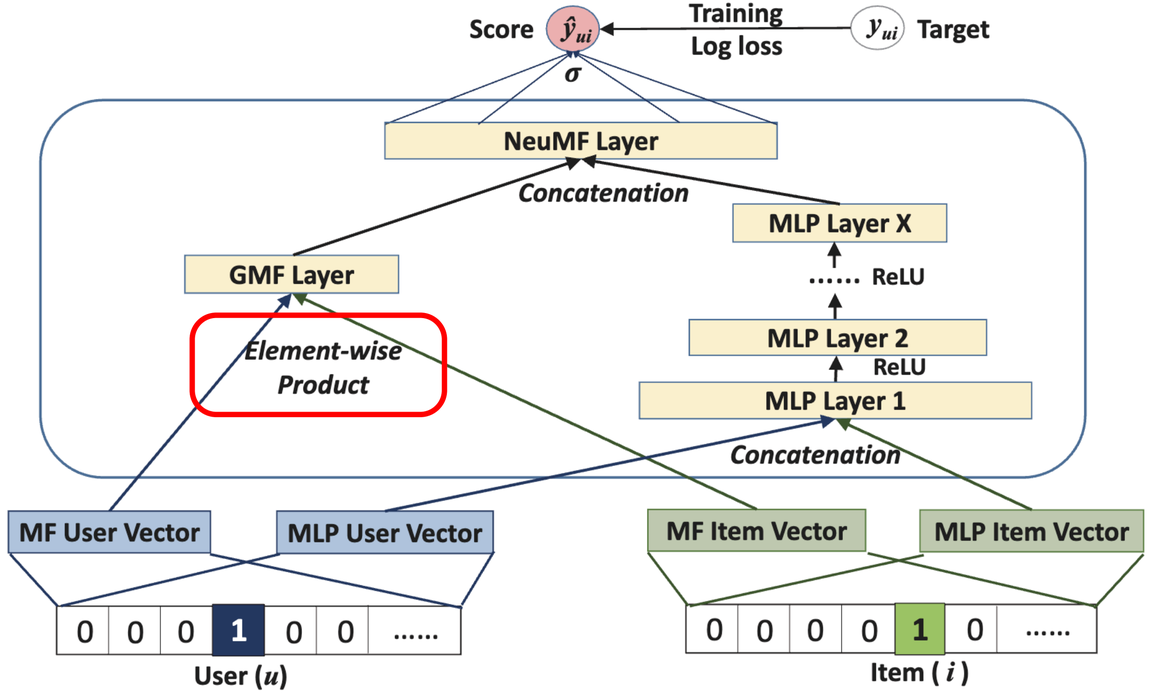
과정

- 변환된 Dense vector를 Element-wise Product

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i))$$

a_{out} : 관찰된 상호작용 집합

$\mathbf{h}^T$: 관찰되지 않은 상호작용 집합



Method - Fusion of GMF and MLP

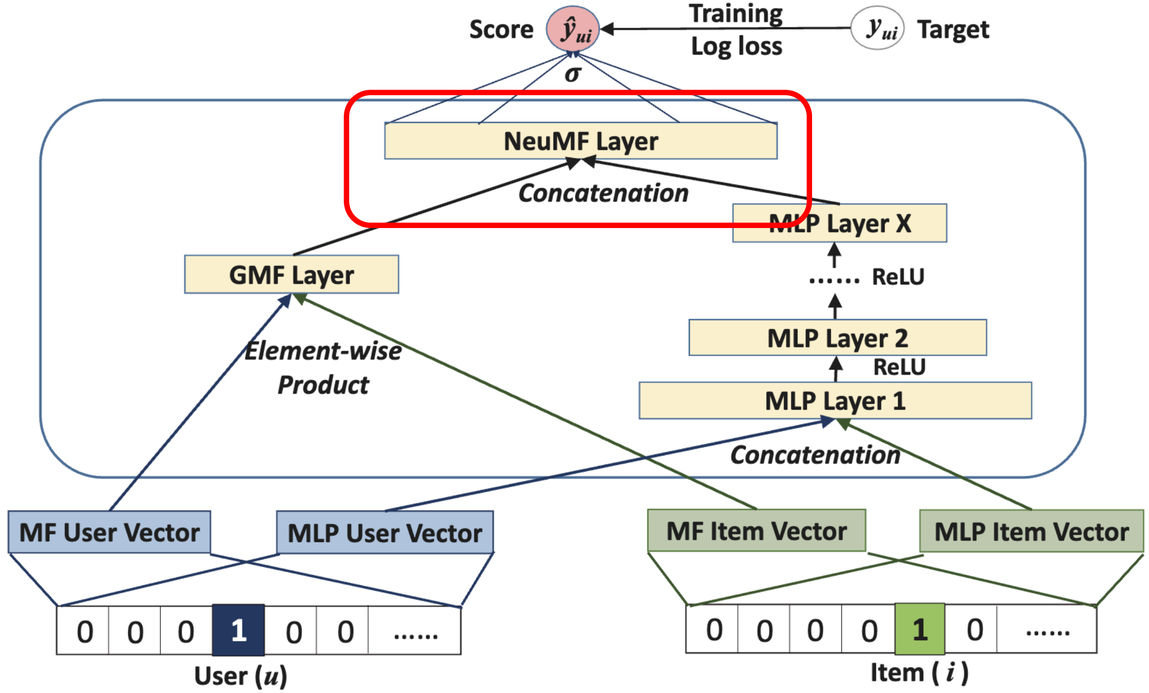
과정

- GMF 모델 결과값과 MLP 모델 결과값 Concat 후 NeuMF층에 입력

$$\phi^{GMF} = \mathbf{p}_u^G \odot \mathbf{q}_i^G$$

$$\phi^{MLP} = a_L(\mathbf{W}_L^T(a_{L-1}(\dots a_2(\mathbf{W}_2^T \begin{bmatrix} \mathbf{p}_u^M \\ \mathbf{q}_i^M \end{bmatrix} + \mathbf{b}_2)\dots))) + \mathbf{b}_L$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix})$$



Method - Fusion of GMF and MLP

과정

- GMF 모델 결과값과 MLP 모델 결과값 Concat 후 NeuMF층에 입력

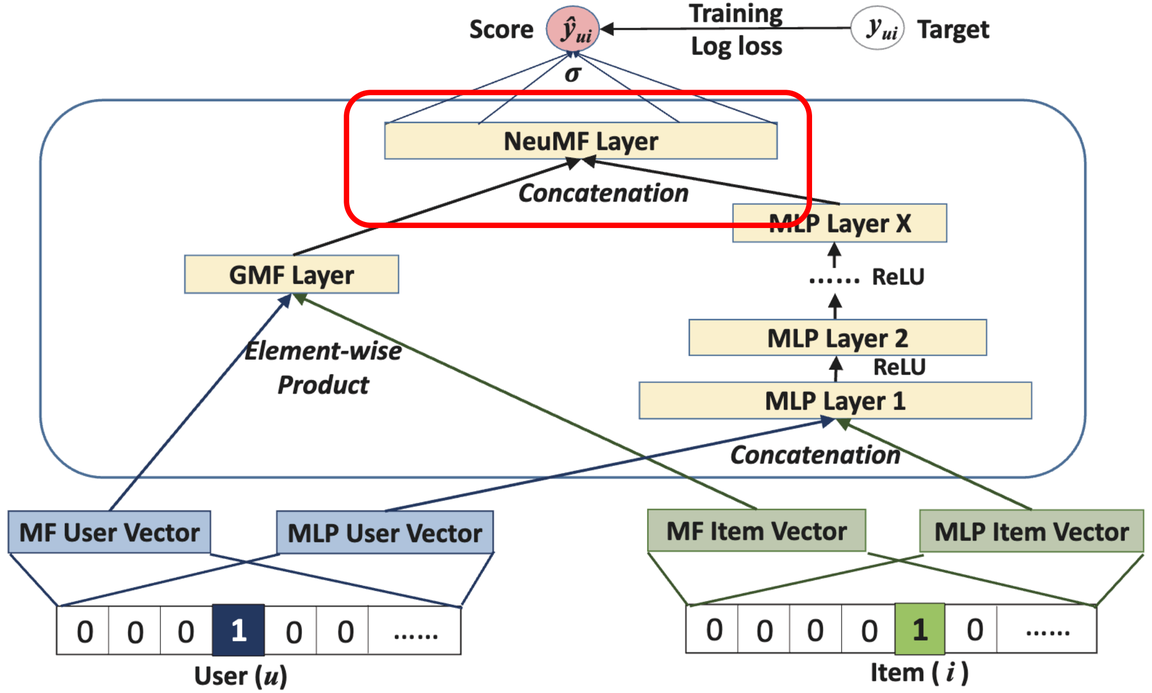
$$L = -\sum_{(u,i)\in Y\cup Y^-} y_{ui}\log\hat{y}_{ui} + (1-y_{ui})\log(1-\hat{y}_{ui})$$

Y : 관찰된 상호작용 집합

Y^- : 관찰되지 않은 상호작용 집합

y_{ui} : 사용자와 아이템간의 실제 상호작용 여부

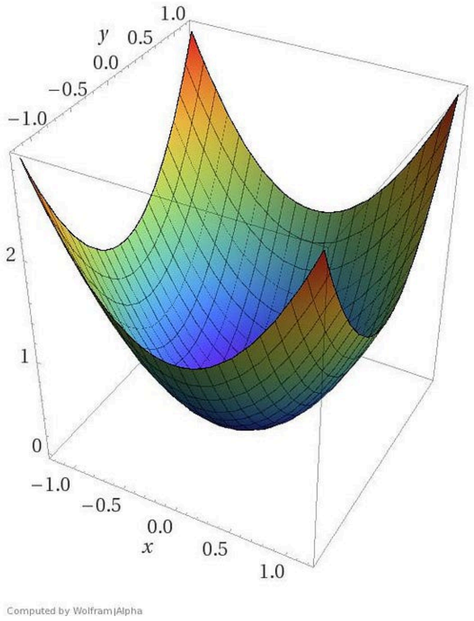
$\hat{y}_{ui}$: 모델이 예측한, 사용자가 아이템과 상호작용할 확률

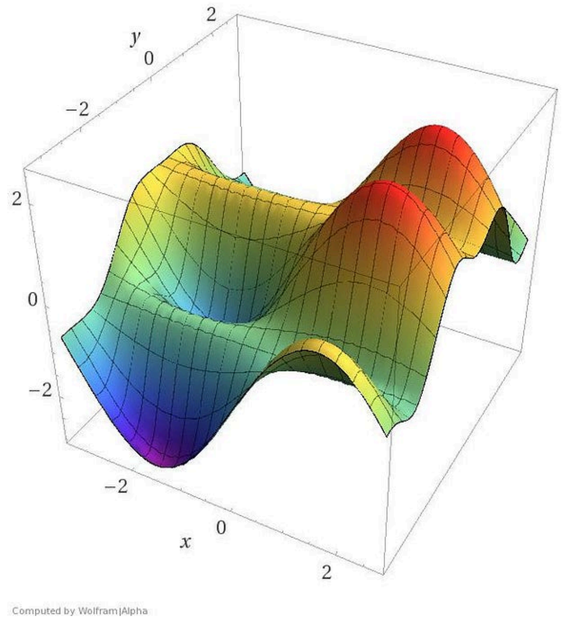


Method - Pretraining

필요성

- 비선형 딥러닝 구조 + 비볼록 함수 → local optimum에 빠질 수 있음
- 최적화가 어려움





Method - Pretraining

과정

- GMF, MLP를 사전 학습 (Adam)
- GMF, MLP 파라미터 가져와서 NeuMF 재학습(VanillaSGD)
- GMF와 MLP의 적용 비율을 α를 통해 비율 설정

$$\mathbf{h} \leftarrow \begin{bmatrix} \alpha \mathbf{h}^{GMF} \\ (1-\alpha)\mathbf{h}^{MLP} \end{bmatrix}$$

Experiments

Experiments

평가 지표 - HR@k

- 사용자가 선호한 아이템 중 하나를 제외하고 학습
- 그 후 사용자별로 k개의 아이템을 추천하고, 앞서 제외한 아이템이 있으면 Hit
- 전체 사용자 수 대비 Hit한 사용자 수 비율이 HR

$$Hit\ Rate = \frac{\#\ of\ Hit\ User}{\#\ of\ User}$$

Experiments

평가 지표 - nDCG@k

- DCG : 분자는 관련 여부, 분모는 log(등수+1)로 관련성에 대해 평가한 지표
- IDCG : 가장 이상적인 DCG
- nDCG : DCG/IDCG

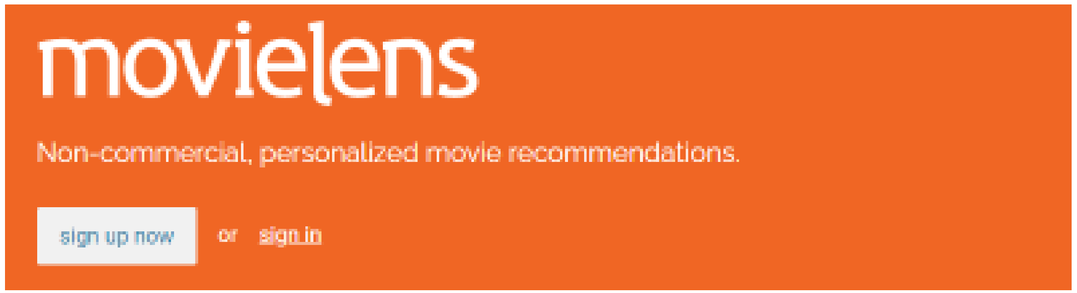
$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

Experiments - MovieLens

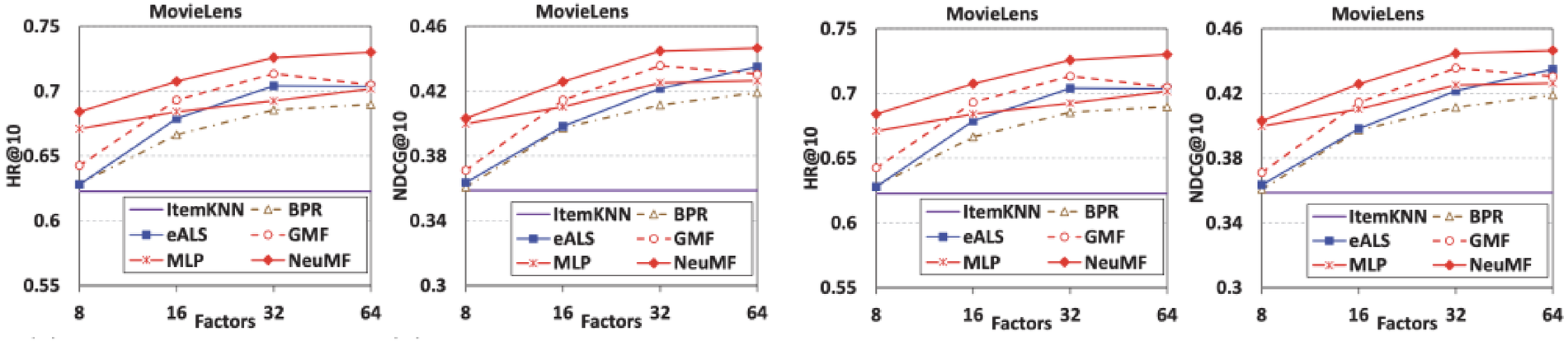
데이터 셋

- MovieLens
- 기존의 MovieLens는 Explicit data이지만, 이것을 Implicit Data라고 가정하고 실험



Experiments - MovieLens

실험결과



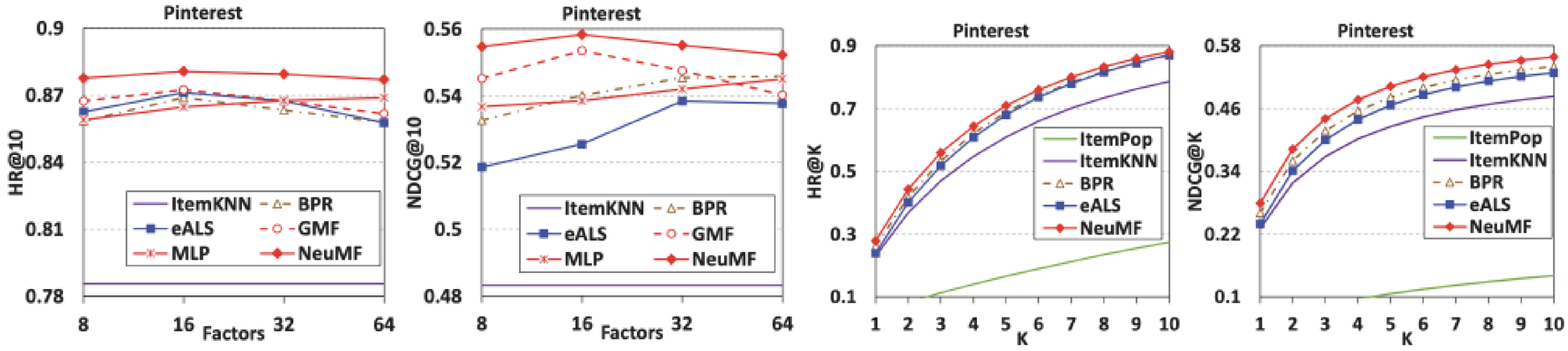
Experiments-Pinterest

데이터 셋

- Pinterest
- 전체 사용자 중 20% 이상이 단 하나의 핀만 가지고 있음
- 따라서 핀이 20개 이상인 데이터로만 평가

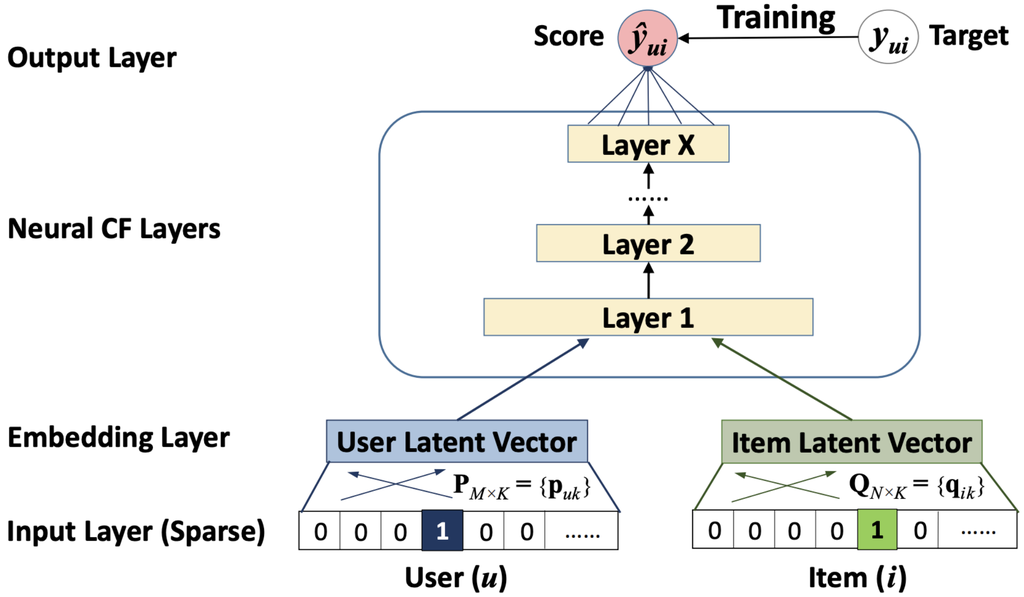
Experiments-Pinterest

실험결과



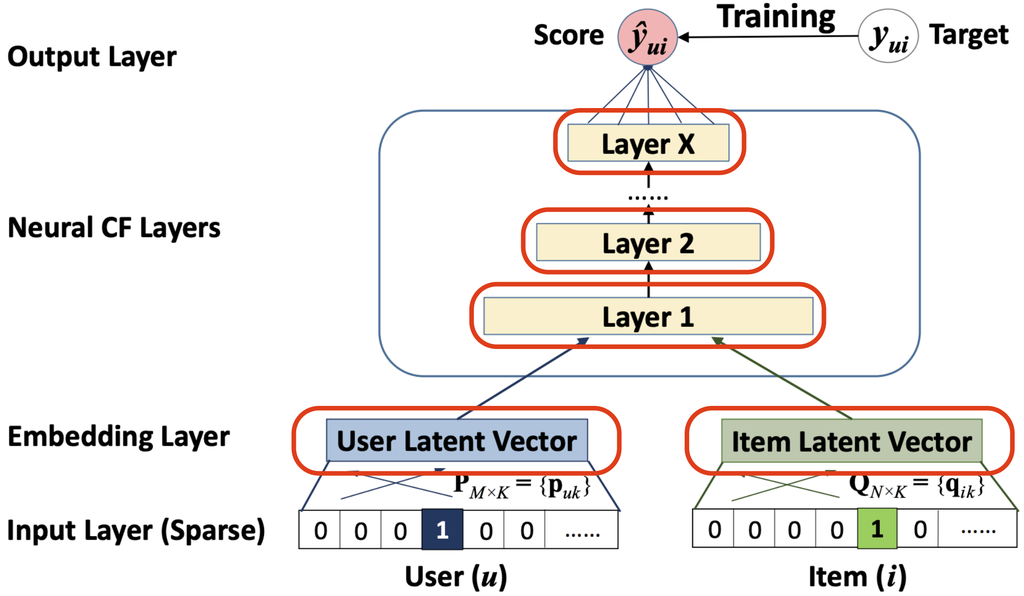
Code

Code - MLP



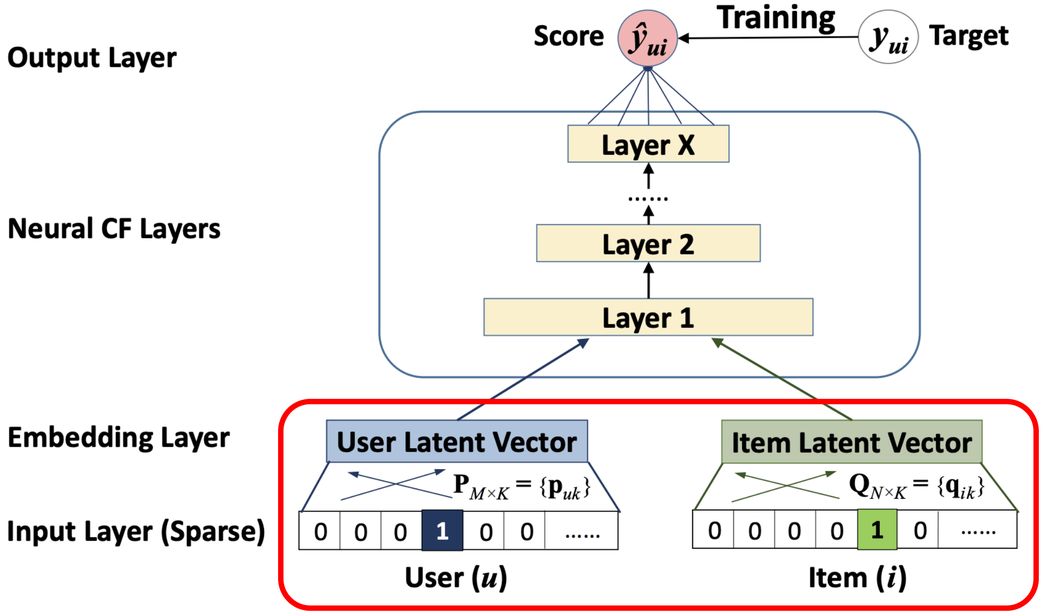

```python
class DL_MF(nn.Module):
    def __init__(self, user_len, movie_len,rank):
        super(DL_MF,self).__init__()
        self.user_len = user_len
        self.movie_len = movie_len
        self.rank = rank

        self.users_embs = nn.Embedding(user_len+1, rank)
        self.items_embs = nn.Embedding(movie_len+1, rank)

        self.fc1 = nn.Linear(rank*2, rank)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(rank, 5)
        self.fc3 = nn.Linear(5, 1)

    def forward(self, user_idx, movie_idx):
        user_emb = self.users_embs(user_idx)
        item_emb = self.items_embs(movie_idx)
        user_item_concat_emb = torch.concat([user_emb, item_emb], dim=1)
        x = self.fc1(user_item_concat_emb)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)

        return x.squeeze()
```

Code - MLP




```
class DL_MF(nn.Module):
    self.user_len = user_len
    self.movie_len = movie_len
    self.rank = rank

    self.users_embs = nn.Embedding(user_len+1, rank)
    self.items_embs = nn.Embedding(movie_len+1, rank)

    self.fc1 = nn.Linear(rank*2, rank)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(rank, 5)
    self.fc3 = nn.Linear(5, 1)
        user_emb = self.users_embs(user_idx)
        item_emb = self.items_embs(movie_idx)
        user_item_concat_emb = torch.concat([user_emb, item_emb], dim=1)
        x = self.fc1(user_item_concat_emb)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)

        return x.squeeze()
```

Code - MLP

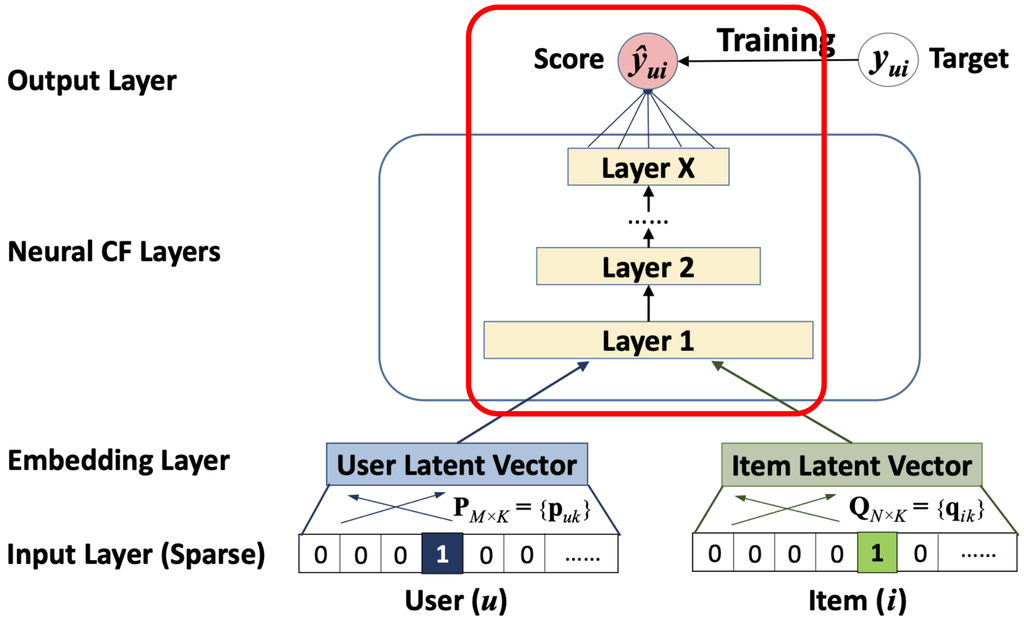



```python
class DL_MF(nn.Module):
    def __init__(self, user_len, movie_len,rank):
        super(DL_MF,self).__init__()
        self.user_len = user_len
        self.movie_len = movie_len
        self.rank = rank

        self.users_embs = nn.Embedding(user_len+1, rank)
        self.items_embs = nn.Embedding(movie_len+1, rank)

        self.fc1 = nn.Linear(rank*2, rank)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(rank, 5)
```

```python
user_emb = self.users_embs(user_idx)
item_emb = self.items_embs(movie_idx)
user_item_concat_emb = torch.concat([user_emb, item_emb], dim=1)
```

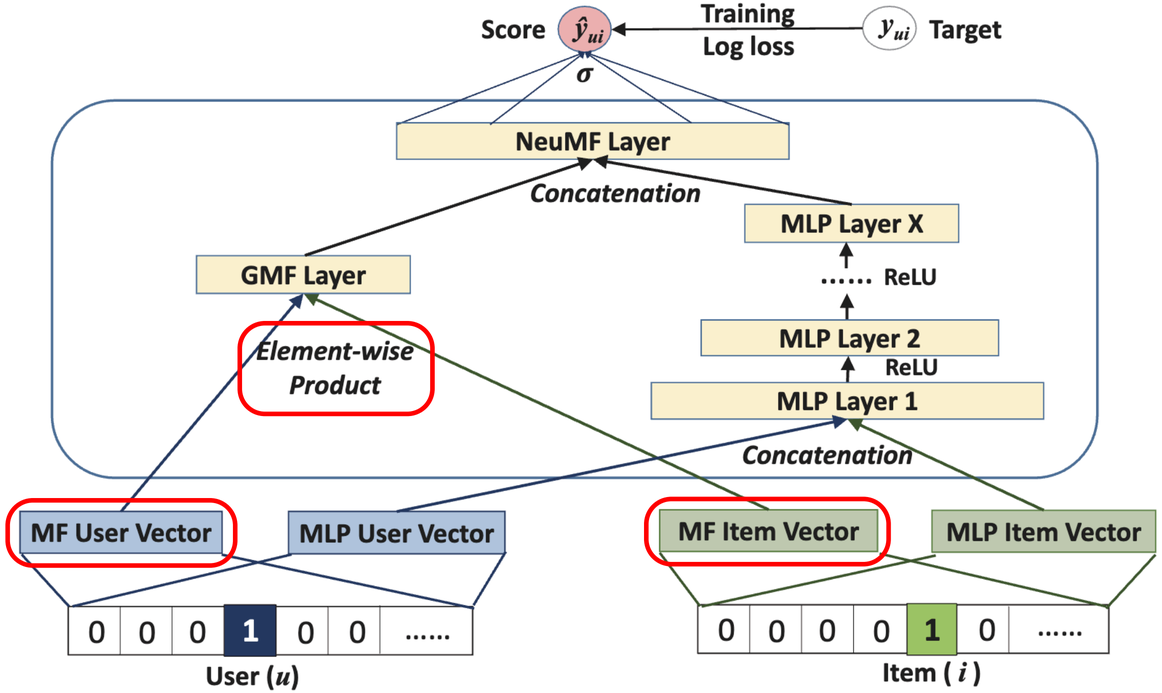
```python
        item_emb = self.items_embs(movie_idx)
        user_item_concat_emb = torch.concat([user_emb, item_emb], dim=1)
        x = self.fc1(user_item_concat_emb)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)

        return x.squeeze()
```

Code - MLP



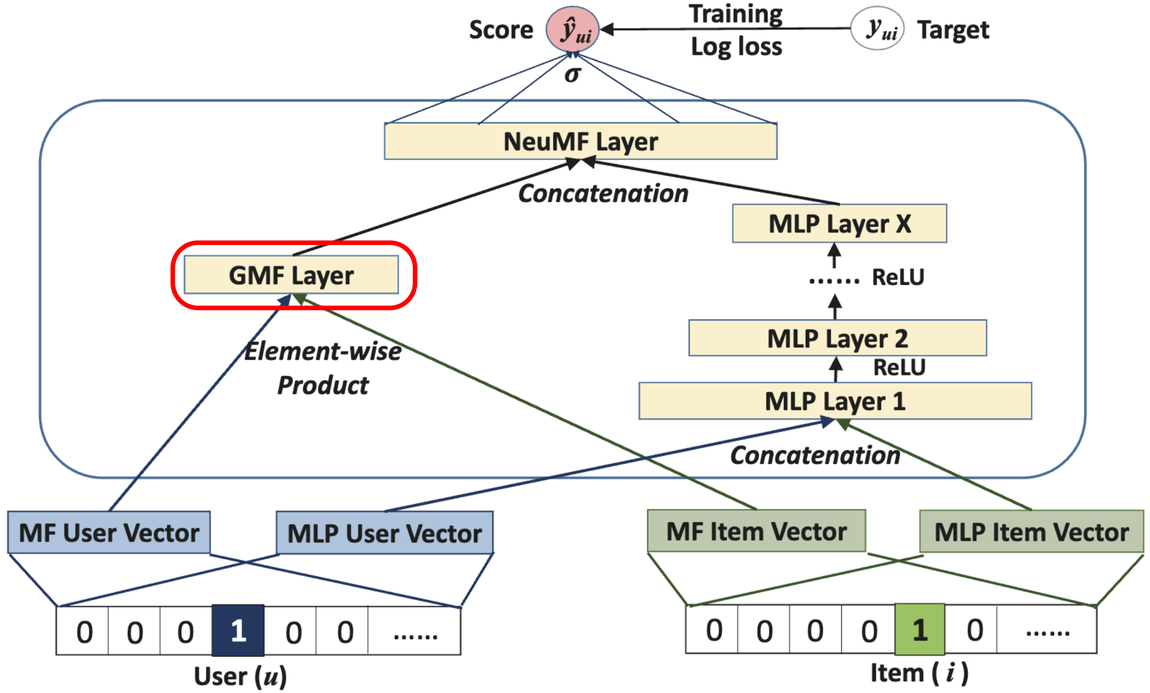

```
class DL_MF(nn.Module):
    def __init__(self, user_len, movie_len,rank):
        super(DL_MF,self).__init__()
        self.user_len = user_len
        self.movie_len = movie_len
        self.rank = rank

        self.users_embs = nn.Embedding(user_len+1, rank)
        self.items_embs = nn.Embedding(movie_len+1, rank)

        self.fc1 = nn.Linear(rank*2, rank)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(rank, 5)
        self.fc3 = nn.Linear(5, 1)

    def forward(self, user_idx, movie_idx):
        user_emb = self.users_embs(user_idx)
        item_emb = self.items_embs(movie_idx)
        x = self.fc1(user_item_concat_emb)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x.squeeze()
```

Code - GMF



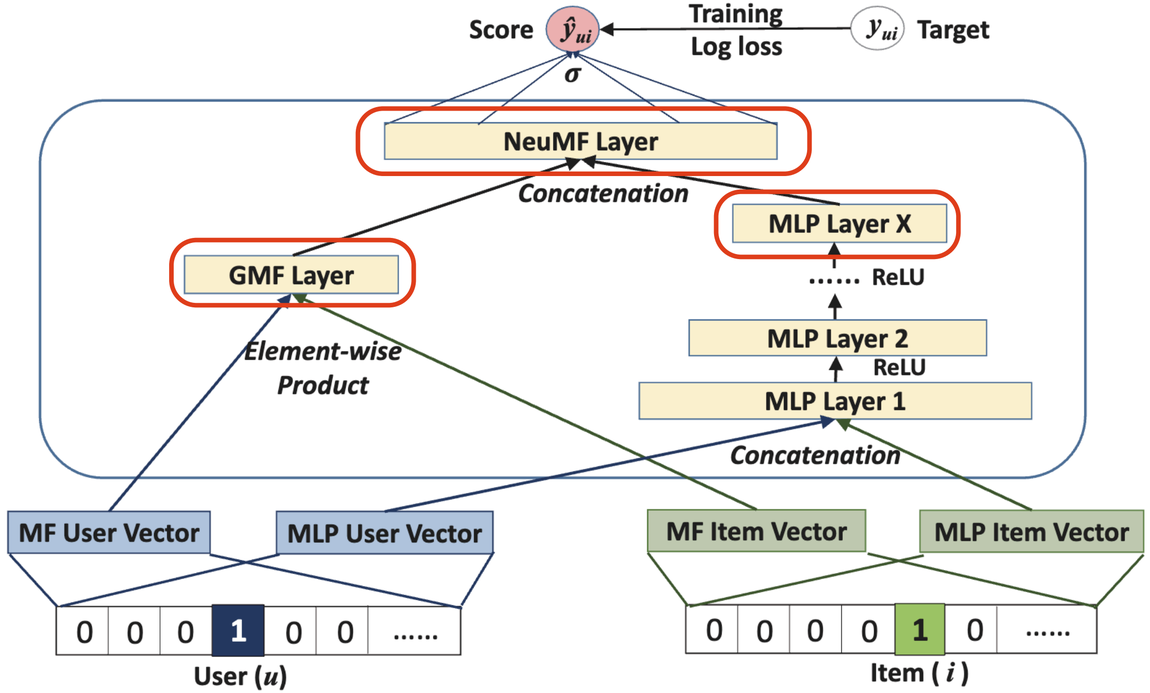

```
user_latent = self.user_embedding(user_indices)
item_latent = self.item_embedding(item_indices)
elementwise_product = user_latent * item_latent
```

Code - GMF




```
class GMF(nn.Module):
    def __init__(self, num_users, num_items, latent_features=8, regs=0.0):
        super(GMF, self).__init__()
        self.user_embedding = nn.Embedding(num_users, latent_features)
        self.item_embedding = nn.Embedding(num_items, latent_features)
        self.output_layer = nn.Linear(latent_features, 1)

    def forward(self, user_indices, item_indices):
        user_latent = self.user_embedding(user_indices)
        item_latent = self.item_embedding(item_indices)
        output = self.output_layer(elementwise_product)
        return output.squeeze()
```

Code - NeuMF

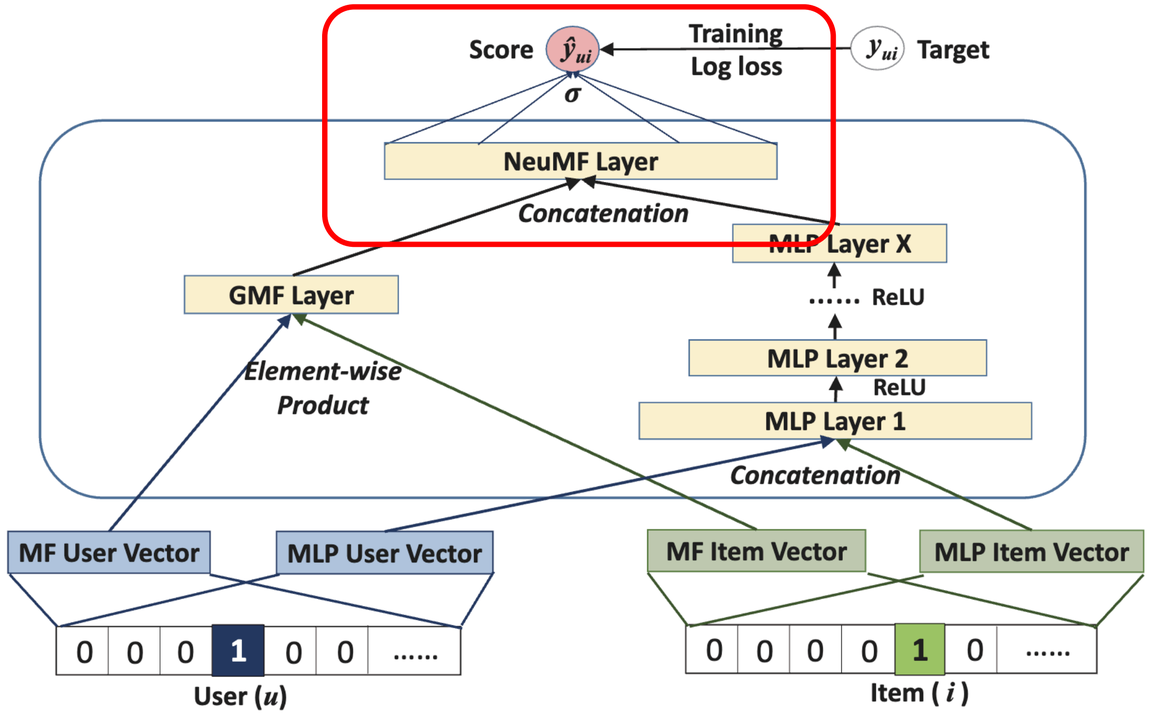



```
self.predict_layer = nn.Linear(latent_features + layers[-1], 1)
```

```
gmf_output = user_latent_gmf * item_latent_gmf
```

```
mlp_output = self.mlp_layers(mlp_input)
```

Code - NeuMF




```
# Concatenate GMF and MLP parts
final_input = torch.cat([gmf_output, mlp_output], dim=-1)
prediction = self.predict_layer(final_input).squeeze()
```

Conclusion

Conclusion

결론

- **기존 행렬 분해(Matrix Factorization)의 한계 극복**
  - 전통적인 행렬 분해 방식은 사용자-아이템 간 선형 내적만을 사용해 복잡한 상호작용을 충분히 표현하지 못함.
  - NCF는 신경망 구조를 도입해 비선형적이고 복잡한 사용자-아이템 상호작용을 효과적으로 모델링
- **일반화된 행렬 분해(GMF)와 다층 퍼셉트론(MLP)의 결합**
  - GMF와 MLP를 각각 활용해 선형적/비선형적 상호작용을 포착
  - 이를 결합한 NeuMF 구조를 제안

Conclusion

팀원 기여

- 20203157 하철환
  - 논문 분석
  - ppt 제작, 대본 작성, 영상 제작
  - 코드 분석

- 20203165 홍세원
  - 논문 분석
  - ppt 제작, 대본작성, 오프라인 발표
  - 코드 구현