# Neural Collaborative Filtering

2021 Spring
Special Lectures on Databases (Recsys)

2021-05-12
한민희

# 기존 MF의 한계와 이 연구의 기여

- ~~(딥러닝에 익숙한 우리들에게는 당연히 딥러닝이 더 잘될 것 같지만)~~
- 논문의 발표 시점(2017)까지도 MF는 latent factor model-based 추천 시스템은 가장 널리 쓰이는 방식이었으나
- MF(의 inner product)는 latent feature를 **linear하게 조합하기 때문에, 실제로 non-linear한 관계를 포착할 수 없음**
- 이 연구는 복잡한 관계를 나타낼 수 있는 **DNN을 도입**, 일반적으로 적용 가능한 Neural Collaborative Filtering의 모델을 제시

# 사용하는 데이터: Implicit Feedback

- 상품 별점을 주었다 -> explicit (선호가 명확히 드러남)
- 상품 설명을 열어보았다 -> implicit (이런 걸 좋아하는 건가?)
- Implicit Feedback은 자동으로 추적되고 양이 더 많지만, 활용하기는 어려움

m: user 개수

n: item 개수

Y: user-item 행렬

$$Y_{m,n} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,n} \end{pmatrix}$$

$$where, \quad y_{u,i} = \begin{cases} 1, & \text{if interaction(user } u\text{, item } i\text{) is observed} \\ 0, & \text{otherwise} \end{cases}$$

- 결국 user u와 item i에 대해,
- u가 i와 상호작용을 할 확률을 예측하는 문제로 귀결
- (아래 식에서 f와 Φ를 구하는 문제)

$$\hat{y}_{u,i} = f(u, i|\Theta),$$

$$where\ f\ \text{is interaction function}, \Theta\ \text{is model parameters}$$

# 기존 Matrix Factorization의 방법

- Y를 더 낮은 차원의 행렬 2개 P, Q로 분리
- 그럼 여기서 y의 예측치를 어떻게 구하는가?

$$
\underset{}{Y(user-item)} \quad \underset{}{P(user)} \quad \underset{}{Q(item)}
$$

$$
\begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \ddots & \vdots \\ y_{m,1} & \cdots & y_{m,n} \end{bmatrix} = \begin{bmatrix} p_{11} & \cdots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mk} \end{bmatrix} \begin{bmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{k1} & \cdots & q_{kn} \end{bmatrix}
$$

$$
m \times n \qquad m \times k \qquad k \times n
$$

$$
where \ \mathbf{p}_u = [p_{u1}, \ldots, p_{uk}], \quad \mathbf{q}_i = [q_{1i}, \ldots, q_{ki}]
$$

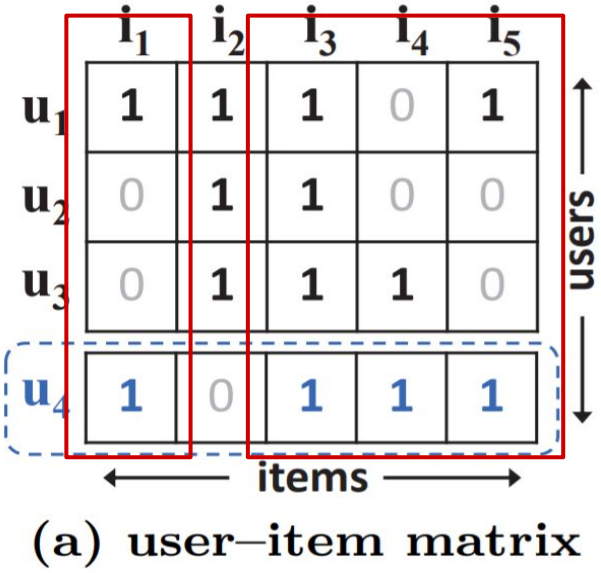
# 기존 Matrix Factorization의 방법

- MF에선 두 vector의 inner product로 y의 예측값을 계산
- 그런데 이것이 잘 작동하지 않는 경우가 있음

$$\hat{y}_{ui} = f(u, i|\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ik}$$

# 기존 MF가 잘 작동하지 못하는 예시

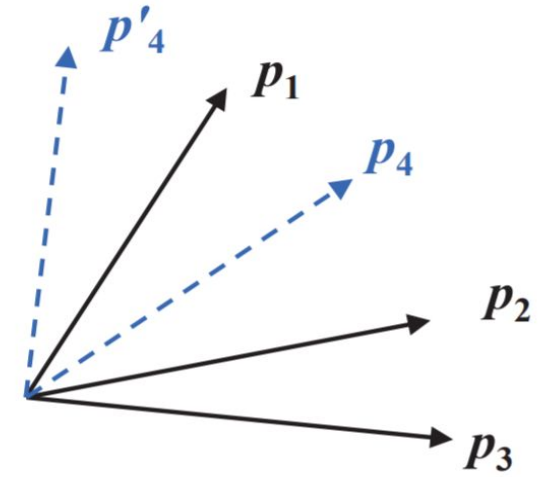
- 두 user의 유사성 또한 inner product로 구해질 수 있음 (user와 item이 같은 latent space)
- u4와 나머지 user들의 유사성을 계산할 경우
- Jaccard similarity(겹치는 항목 수 / 전체 항목 수)를 기준으로 u1 > u3 > u2 순으로 유사
- **“그럼 u4 벡터를 u1 벡터에 가장 가깝게 위치시키면 되겠지?”**



(a) user–item matrix

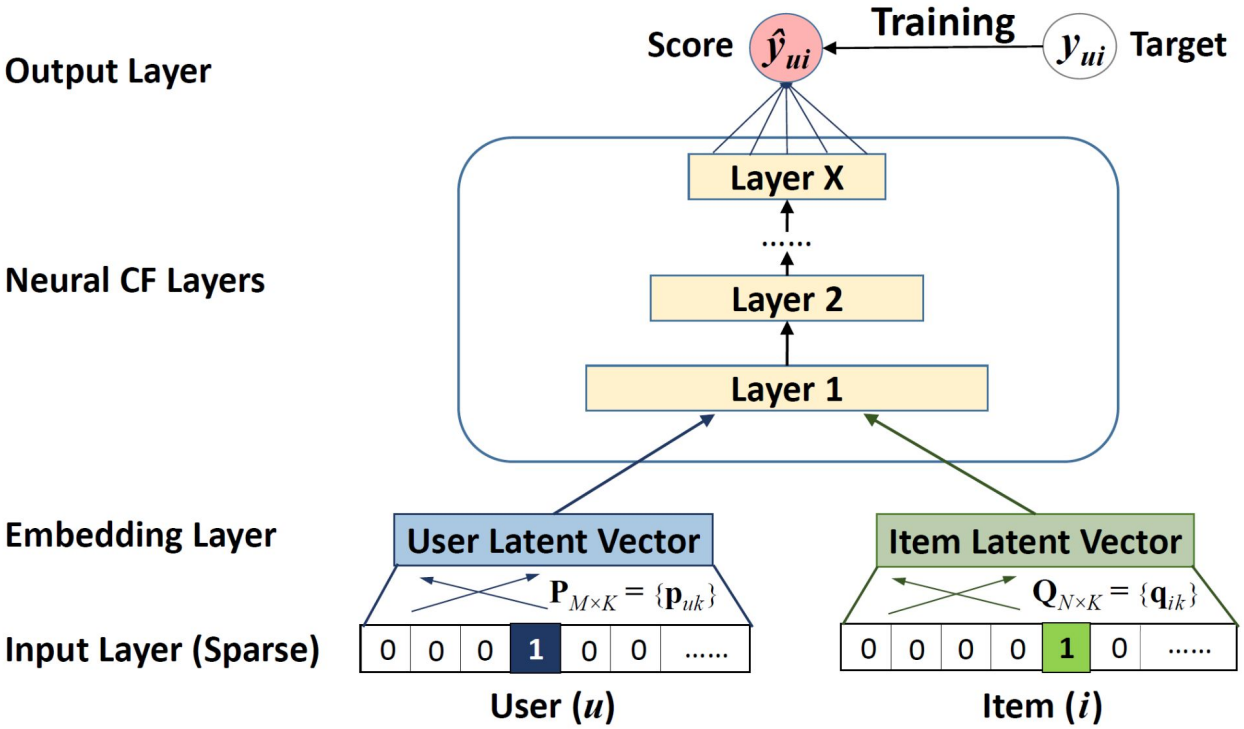
# 기존 MF가 잘 작동하지 못하는 예시

- 기존의 p1, p2, p3가 위치한 latent space에
- u4에 대응하는 벡터인 p4를 위치시키면
- p1에는 가장 가깝지만, p3보다 p2에 더 가깝게 나타나 앞서 계산한 유사성 순위가 다른 결과가 나타남
- 이것은 linear 모델이기 때문

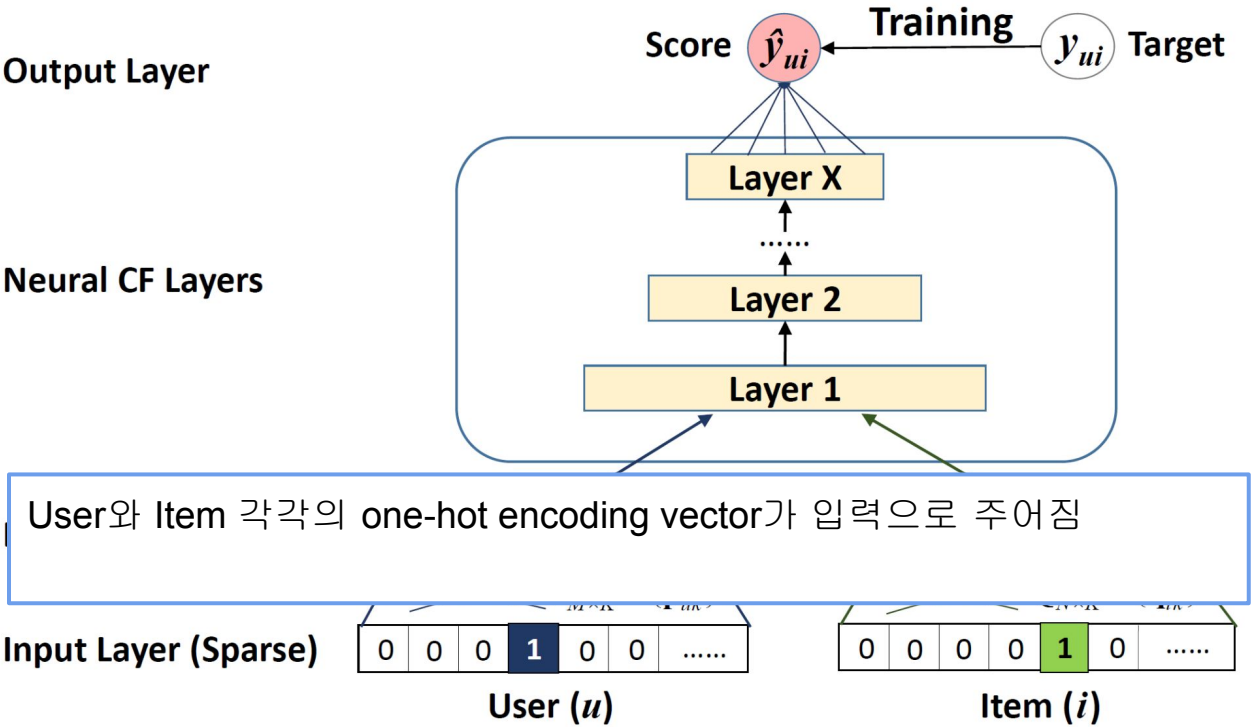



(b) user latent space

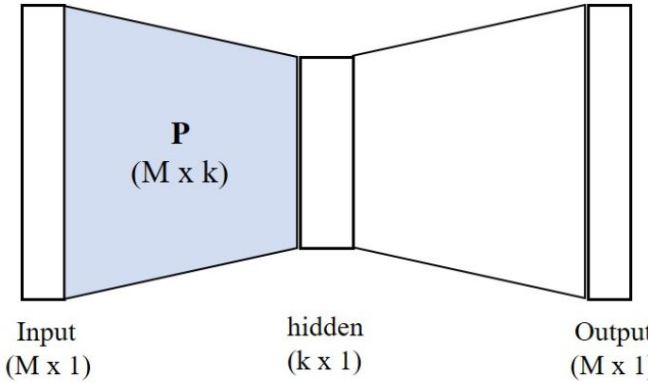
# NCF Framework



# NCF Framework



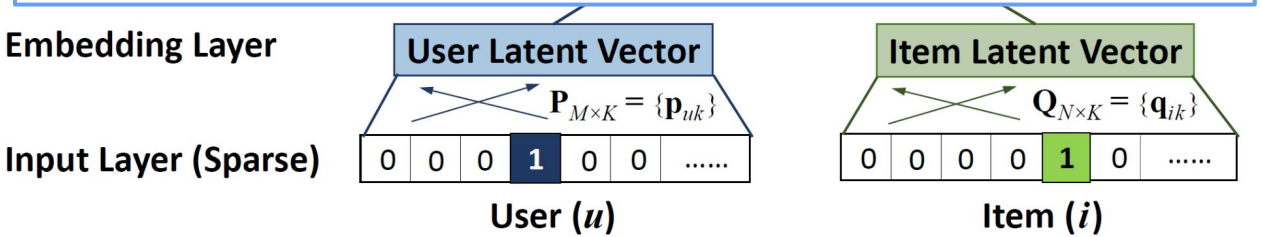
# NCF Framework



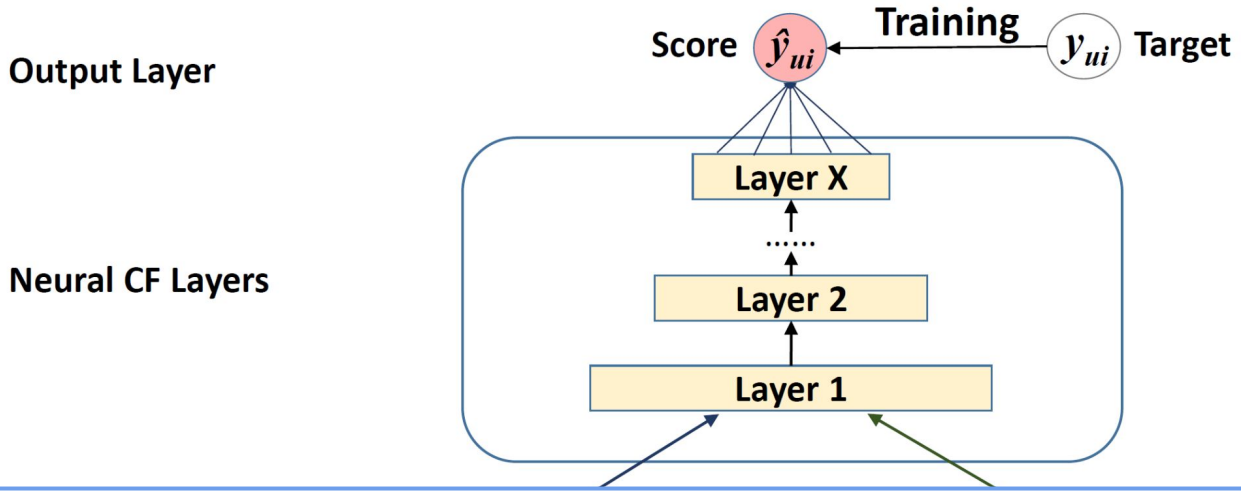

M 차원의 벡터를
M보다 작은 k 차원의 공간에 project

이때 P의 각 row는,
각 user/item을 표현하는
latent vector인 셈

Embedding Layer에서는,
Input Layer에 입력으로 주어진 one-hot vector를 dense vector로 매핑

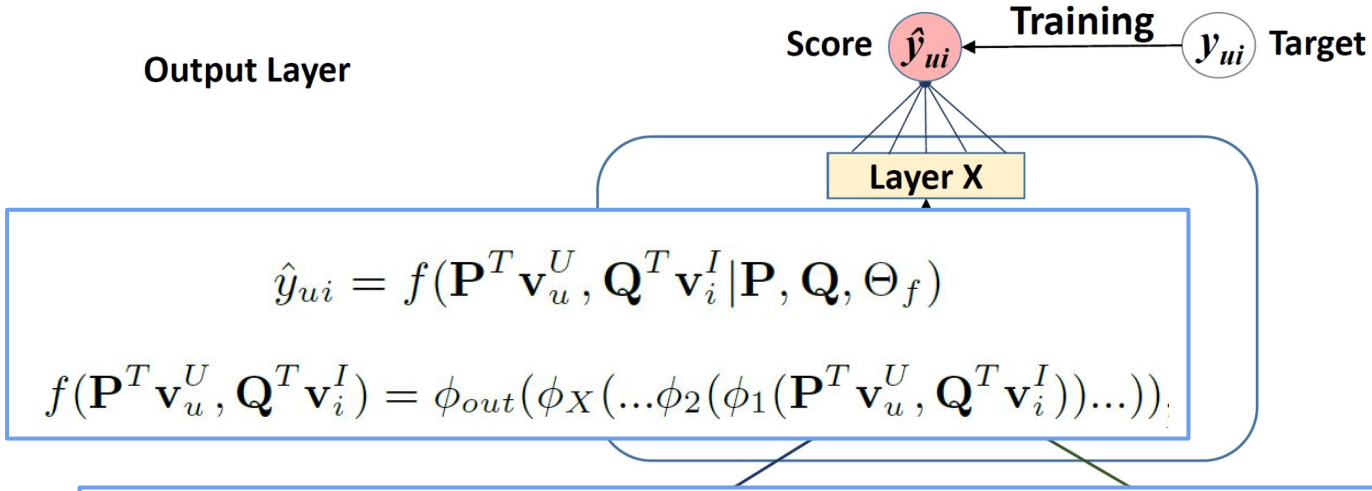


# NCF Framework




이제 user, item 각각의 입력이 embedding으로 표현된 vector를 concatenate하여 Neural CF layer에 입력으로 주어짐

# NCF Framework




그렇다면, y를 구하는 함수 f는

model parameter Θ와 각 layer Φ에 대해,

위와 같이 정식화될 수 있음.

Φout에선 logistic이나 probit 함수를 사용

# Learning NCF

- 학습하기 위해선 loss function을 일단 정의해야 하는데,
- *Y*가, y 값이 1인 set, *Y*-는 y 값이 0인 set을 나타낸다면
- (각각 interaction이 관측된 것들과 그렇지 않은 것들)
- likelihood function은 다음과 같이 표현됨

$$p(\mathcal{Y}, \mathcal{Y}^{-}|P, Q, \Theta_f) = \prod_{(u,i)\in\mathcal{Y}} \hat{y}_{u,i}^{\,y_{u,i}} \prod_{(u,j)\in\mathcal{Y}^{-}} (1-\hat{y}_{u,j})^{1-y_{u,i}}$$

# Learning NCF

- 이에 대한 loss function은 다음과 같아지는데, 이는 binary corss entropy loss와 동일
- NCF는 이 L을 최소화하는 parameters를 찾아야 함
- 예측값이 다르다면 L은 무한이 되고, 같다면 L은 0이 된다

$$L = -log\, p(\mathcal{Y}, \mathcal{Y}^- | P, Q, \Theta_f)$$

$$= -\sum_{(u,i)\in\mathcal{Y}} y_{u,i}\, log\, \hat{y}_{u,i} - (\sum_{(u,j)\ in\mathcal{Y}^-} (1 - y_{u,i})\, log\, (1 - \hat{y}_{u,j}))$$

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

# MF는 NCF의 한 사례

- 그런데, MF는 NCF의 한 케이스가 된다

- NCF를 다음과 같이 설정하면 된다
- 하나의 layer만 갖도록(Φ가 하나)
- output activation(a)은 identity
- 각 output weight(h)는 모두 1
- Φ는 두 벡터의 내적이 되도록 설정

$$\hat{y}_{u,i} = a_{out}(h^T \phi_1(P^T v_u^U, Q^T v_i^I))$$
$$where \;\; a_{out} = \text{identity function}$$
$$h^T = [1, \ldots, 1]_{1 \text{ x k}}$$
$$\phi_1 = P^T v_u^U \odot Q^T v_i^I$$

# Generalized Matrix Factorization

- 아까 설정한 ϕ와 ϕ의 수, h, a를
- 모델마다 다르게 설정하면
- 여러 가지 경우에 적용 가능
- 이것을 GMF라 명명

$$\hat{y}_{u,i} = a_{out}(h^T \phi_1(P^T v_u^U, Q^T v_i^I))$$

$$where \;\; a_{out} = \text{identity function}$$

$$h^T = [1, \ldots, 1]_{1 \text{ x k}}$$

$$\phi_1 = P^T v_u^U \odot Q^T v_i^I$$

# Multi-Layer Perceptron

- GMF는 linear하고 fixed되어 있어서 표현력에 한계가 존재
- MLP는 non-linear하고 flexible
- 다음과 같이 parameter들을 설정 가능
- Φ1은 user / item vector를 concatenate
- 다른 Φ들은 Neural Net 함수로, W는 weight, b는 bias
- GMF와 달리, non-linear한 Φ들이 존재

$$\mathbf{z}_1 = \phi_1(\mathbf{p}_u, \mathbf{q}_i) = \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix},$$

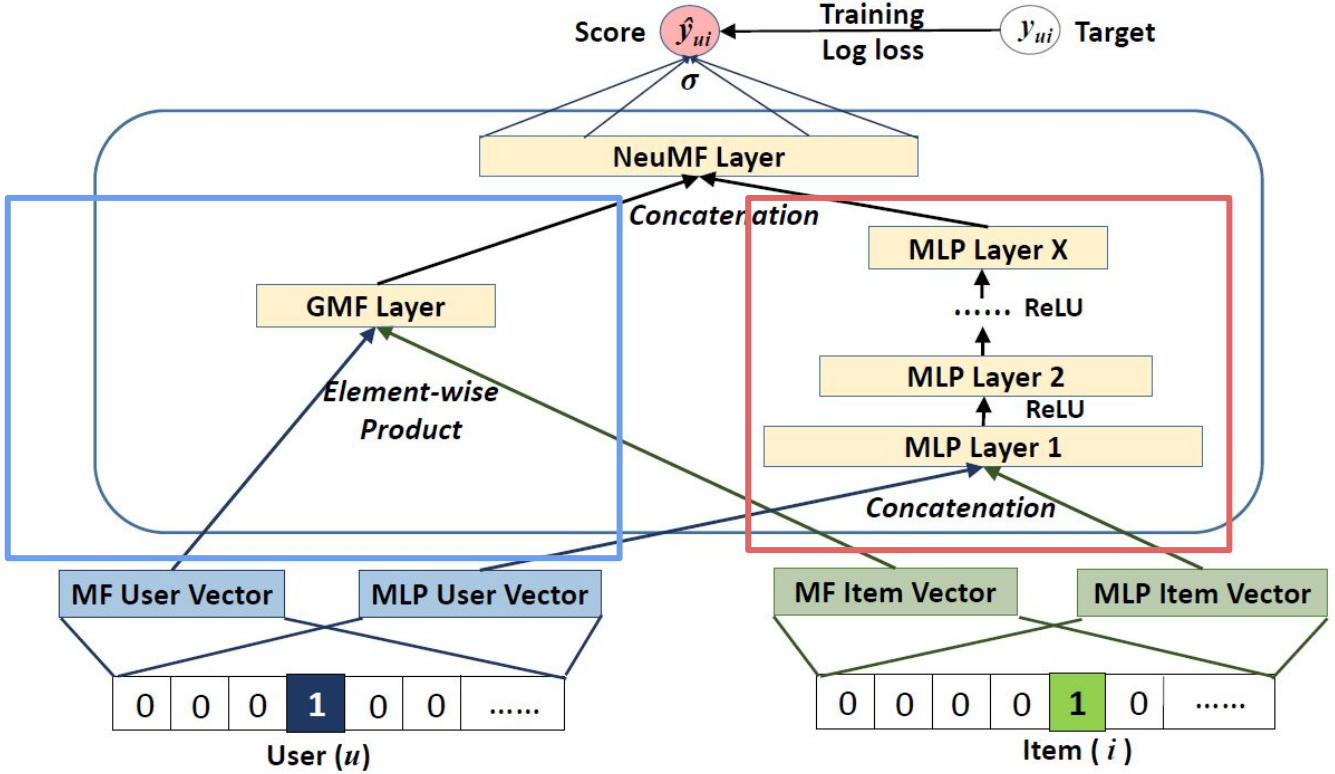
$$\phi_2(\mathbf{z}_1) = a_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2),$$

$$\ldots\ldots$$

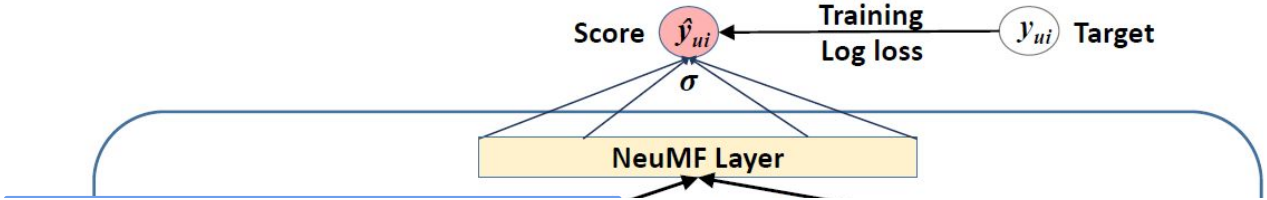
$$\phi_L(\mathbf{z}_{L-1}) = a_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L),$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \phi_L(\mathbf{z}_{L-1})),$$

# GMF와 MLP를 섞으면 더 잘 되겠지?



# GMF와 MLP를 섞으면 더 잘 되겠지?




$$\phi^{GMF} = p_u^G \odot q_i^G$$

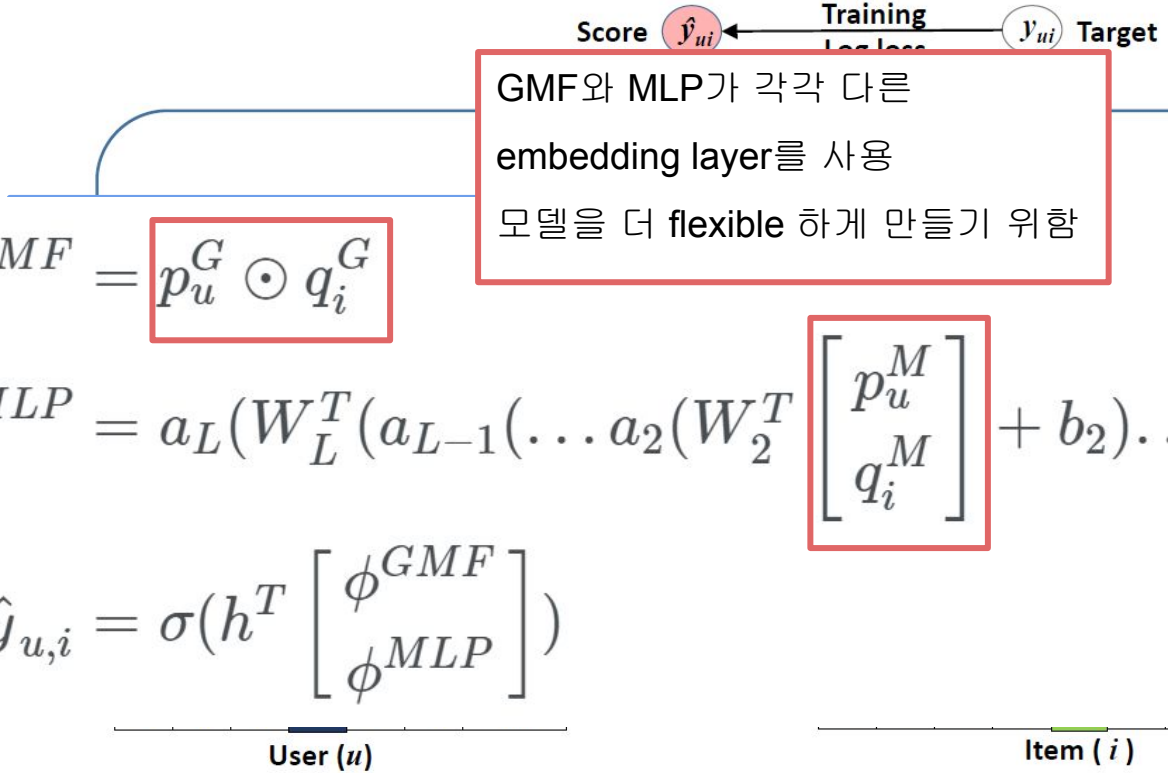
$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(\dots a_2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2)\dots)) + b_L)$$

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

User ($u$)

Item ( $i$ )

# GMF와 MLP를 섞으면 더 잘 되겠지?



# Pre-training

- 모델 전체를 바로 training시키는 것이 아니라

- 먼저, GMF와 MLP를 random initialization 후에
- 이 둘을 Adam optimizer로 train한 다음

- 그 train 결과로 NeuMF의 parameters를 initialize 후
- SGD optimizer로 train

# 실험 결과

- 실험을 통해 다음 세 질문의 결과를 얻음

RQ1. 기존의 implicit filtering method보다 정확한가? -> 그렇다

RQ2. 연구자들이 제시한 optimization framework (log loss with negative sampling)이 recommendation task에서 어떻게 동작하는가? -> iteration이 지날수록 loss가 감소했다. NeuMF가 MLP, GMF만 쓰는 것보다 loss가 낮았다. negative instance에 대해 더 flexible한 sampling ratio를 설정, 최적의 값을 찾을 수 있었다.

RQ3. 더 deep한 layer가 user-item interaction data를 학습하는 데 도움이 되는가? -> 대체로 좋아지다가, 너무 deep하면 오히려 성능이 안 좋아졌다.