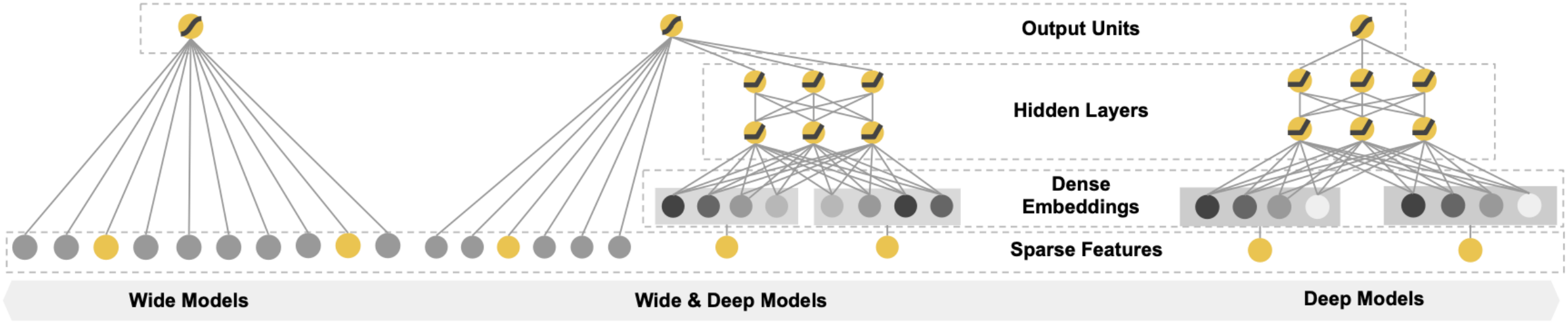
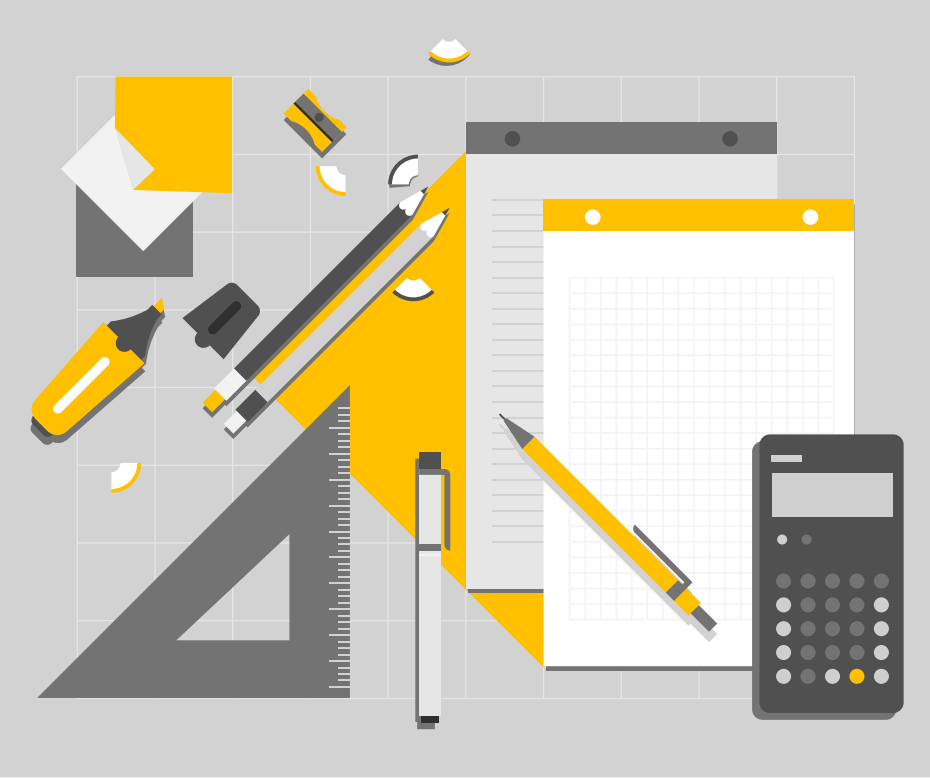
Wide & Deep Learning for Recommender Systems



AI빅데이터 프로젝트: 캡스톤 디자인

Wide & Deep

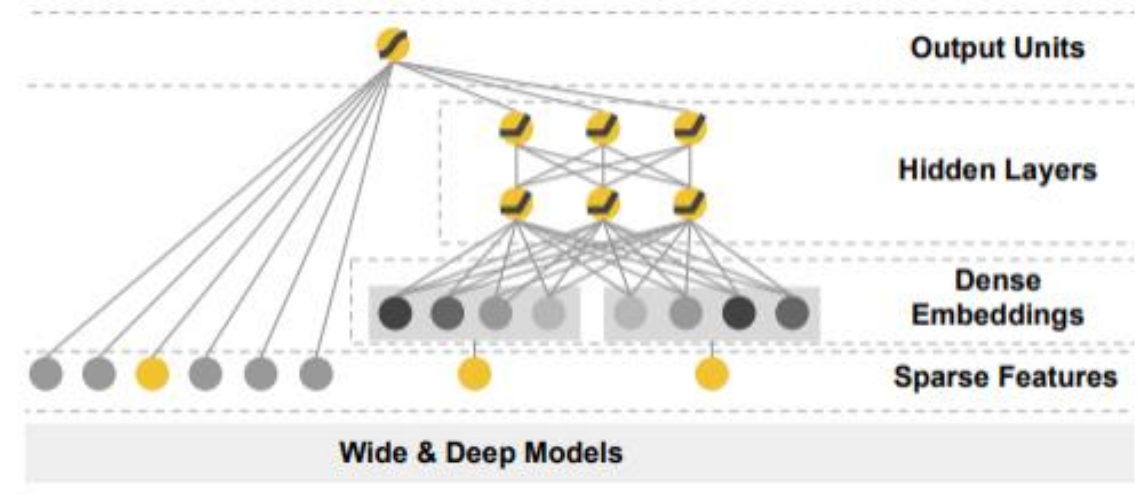
지윤혁, 장영수, 조기흠, 백찬진



CONTENTS

WIDE & DEEP

OVERVIEW



2016년 구글이 발표한 추천랭킹 알고리즘으로,

현재 구글 플레이 앱 추천에 사용됨

Memorization과 Generalization을 동시에 잘 수행할 수 있는 추천시스템 모델

WIDE
(Memorization)

+

WIDE & DEEP

+

DEEP
(Generalization)

기존 모델 대비

유의미한 성능 향상

Memorization

‘펭귄은 날 수 없다’

‘독수리는 날 수 있다’

상관관계를 바탕으로 동시 출현의 빈도를 파악

장점 : 세부적인 부분까지 기억하기 편함

단점 : 발견하지 못한 부분에 대해서는 무지

Generalization

‘날개 달린 새는 모두 날 수 있다’

과거에 거의 발생하지 않았던 새로운 피처 조합 탐색

장점 : 발견하지 못한 부분의 일반화, 추천의 다양성

단점 : 세부적으로는 기억하지 못함

Memorization

‘펭귄은 날 수 없다’ ‘독수리는 날 수 있다’

상관관계를 바탕으로 동시 출현의 빈도를 파악

장점 : 세부적인 부분까지 기억하기 편함

단점 : 발견하지 못한 부분에 대해서는 무지

Generalization

‘날개 달린 새는 모두 날 수 있다’

과거에 거의 발생하지 않았던 새로운 피처 조합 탐색

장점 : 발견하지 못한 부분의 일반화, 추천의 다양성

단점 : 세부적으로는 기억하지 못함

Memorization

Generalization

‘날개를 가진 모든 새들은 날 수 있지만 펭귄, 닭 등 예외적인 새들도 있다’

상관관계를 바탕으로 동시 출현의 빈도를 파악

장점 : 세부적인 부분까지 기억하기 편함

단점 : 발견하지 못한 부분에 대해서는 무지

과거에 거의 발생하지 않았던 새로운 피처 조합 탐색

장점 : 발견하지 못한 부분의 일반화, 추천의 다양성

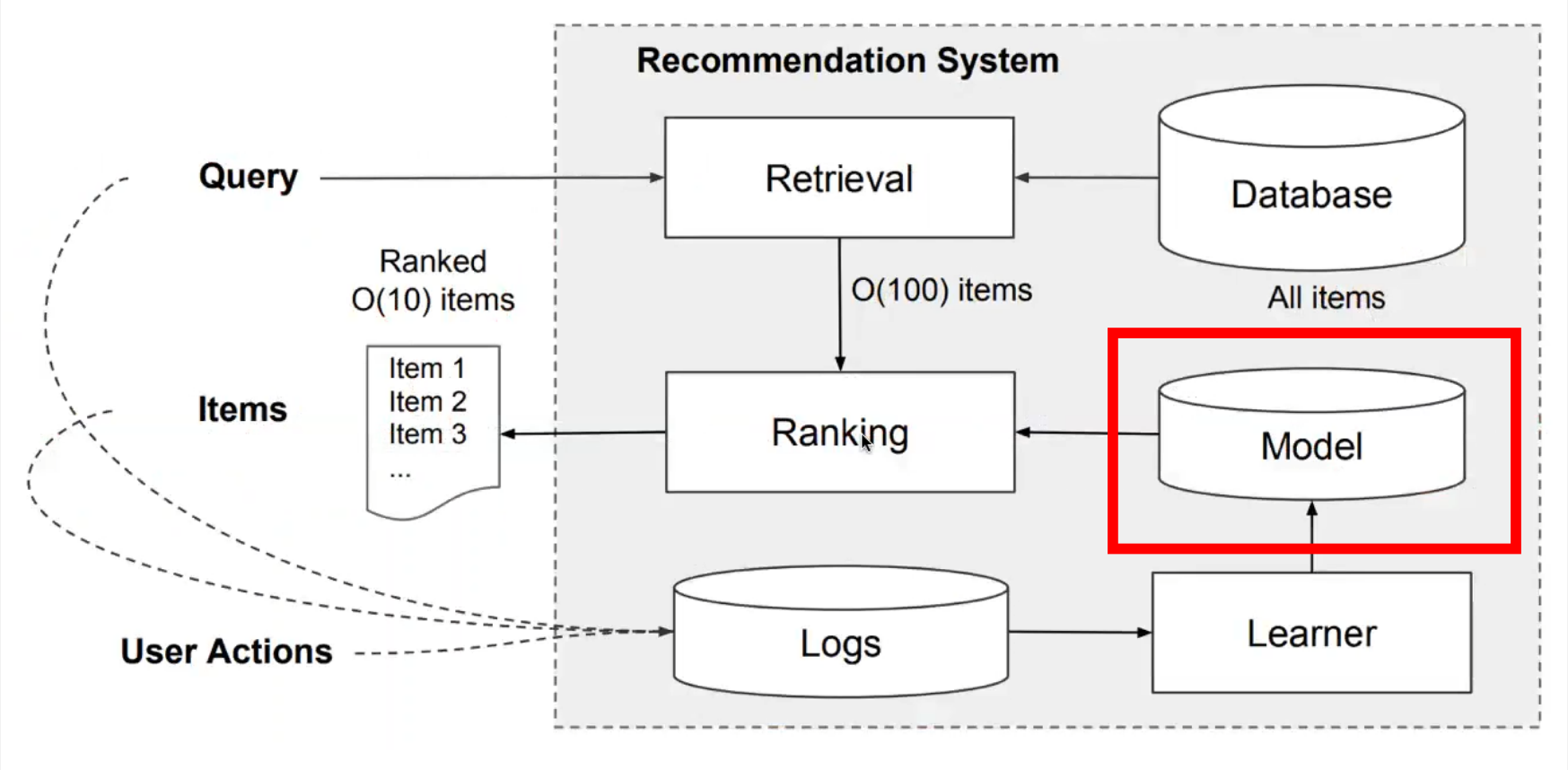
단점 : 세부적으로는 기억하지 못함

Memorization에 특화된 Wide모델과,

Generalization에 특화된 Deep 모델을

결합한 **Wide and deep모델을 제안**

APP Recommender system



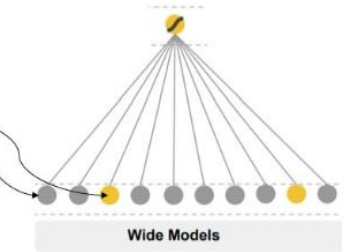
03-1 The wide component

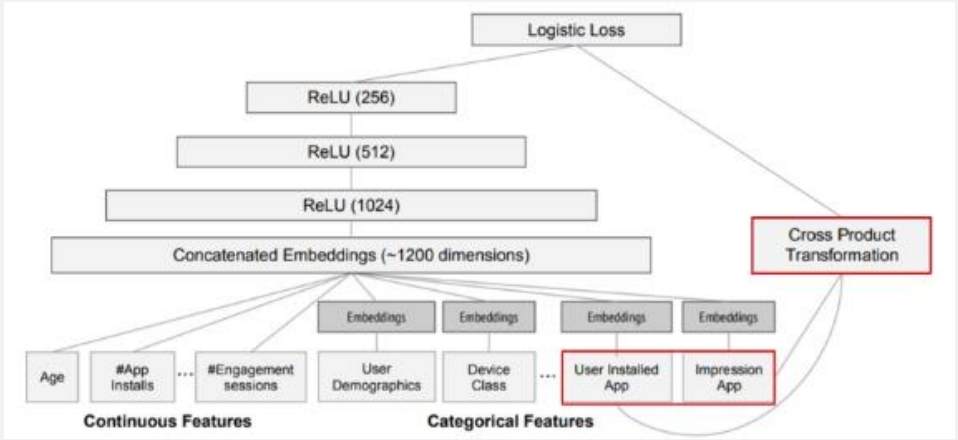
OVERVIEW

SCHEME

$$\text{user\_install\_app} = [A, B]$$
$$\text{user\_impression\_app} = [A, C]$$

Install	Impression	(Install, Impression)	Install x Impression
A	A	(1,1)	1
A	B	(1,0)	0
A	C	(1,1)	1
B	A	(1,1)	1
B	B	(1,0)	0
B	C	(1,1)	1
C	A	(0,1)	0
C	B	(0,0)	0
C	C	(0,1)	0





INPUT DATA

Install APP
[A, B]

Cross product = 1

Impression
APP
[A, B]

PROS AND CONS

PROS

- Memorization
- 특이취향을 반영한 학습

CONS

00이 되는 pair 학습불가능

The wide component

INPUT DATA

Install APP
= [A, B]

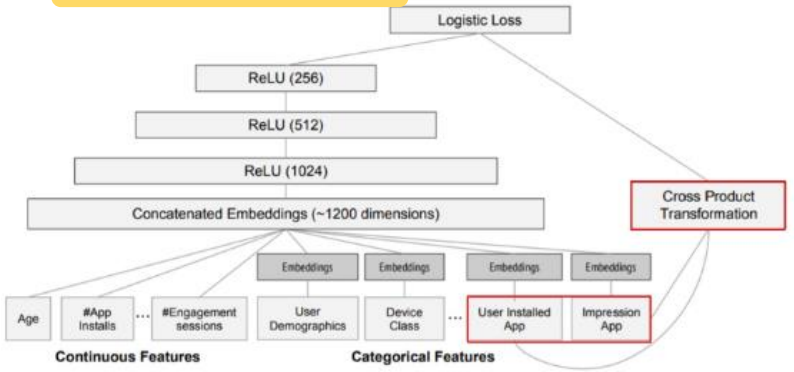
Impression APP
= [A, C]

Install	Impression	(Install, Impression)	Install x Impression
A	A	(1,1)	1
A	B	(1,0)	0
A	C	(1,1)	1
B	A	(1,1)	1
B	B	(1,0)	0
B	C	(1,1)	1
C	A	(0,1)	0
C	B	(0,0)	0
C	C	(0,1)	0

$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

- Input은 User가 설치한 앱 Feature와, 열람한 Feature간 interaction
- 동시출현 빈도를 표현하는 Cross-product로 새로운 특성을 만들어 학습
- 피처 세트는 개별 원본 데이터의 특성과 Cross-product 변환으로 생성된 새로운 특성도 포함하여 고려
- One-hot 벡터를 사용하여 학습을 진행(해당 앱을 설치: 1, X: 0)
- 위의 결과를 바탕으로 결과를 단순히 곱하여 input으로 사용
- Install X Impression의 값이 1인 결과만 학습

Process



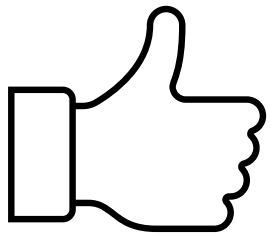
$$y = \mathbf{w}^T\mathbf{x} + b$$

$$where, \quad \mathbf{x} = [x_1, x_2, \ldots, x_d]$$
$$\mathbf{w} = [w_1, w_2, \ldots, w_d]$$
$$b = \text{bias}$$

- input X에 가중치 W를 곱하여 bias를 더한 형태
- W와 b를 Backpropagation을 통하여 학습, 갱신

The wide component

PROS & CONS

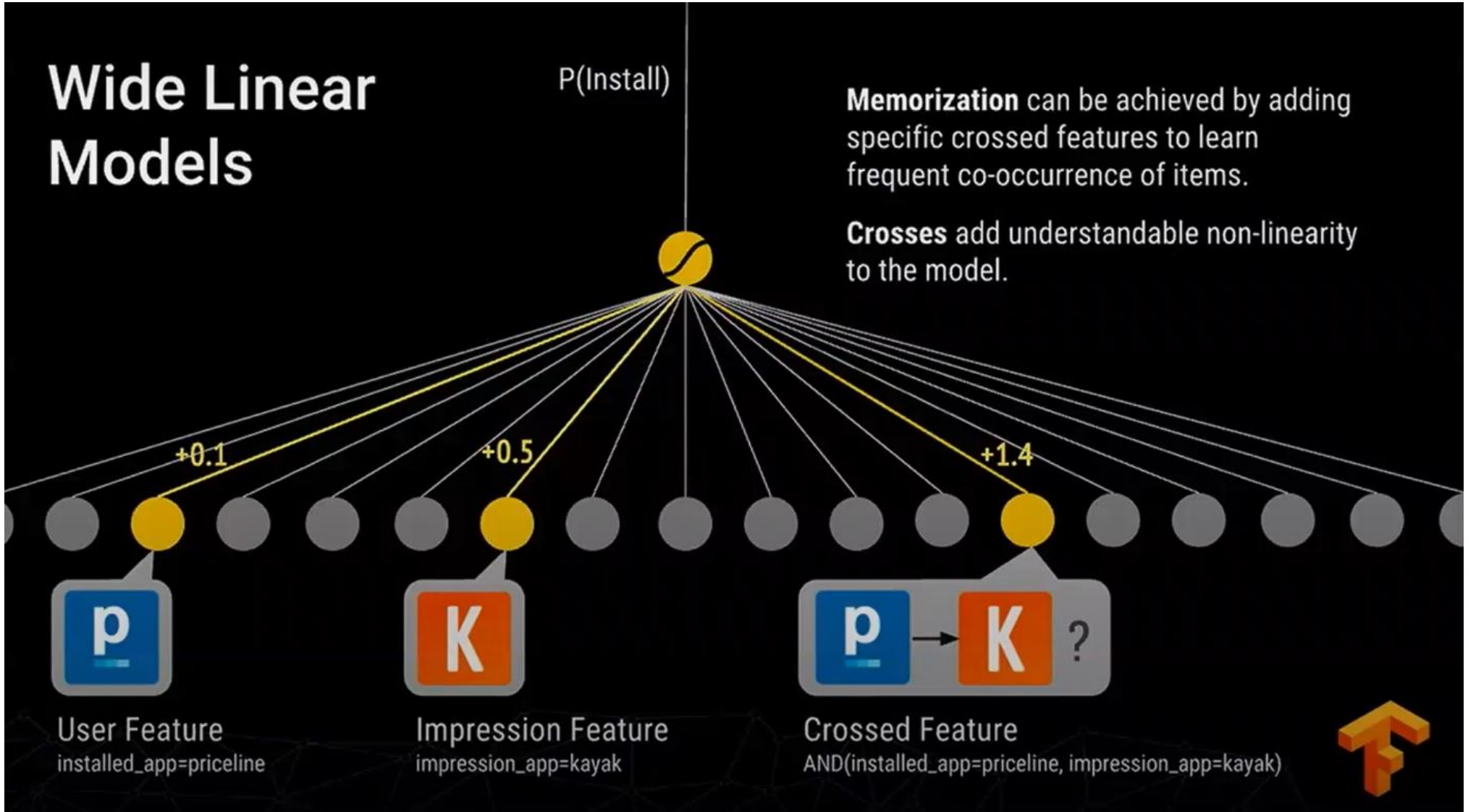


- 해당 방식은 1이 되는 모든 경우의 수를 학습하기 때문에 Memorization에 강함
- User의 특이 취향이 반영된 Niche Combination을 학습하기에 탁월

ex) 여행과 관련된 Airbnb앱을 깔았고, 야놀자, 여기어때 등의 앱을 본 기록이 있다면 해당 사용자는 여행에 관심이 있다는 것을 쉽게 알 수 있음

- Feature Engineering을 일일이 해줘야함
- 0이 되는 pair는 학습을 하지 않기 때문에 추천사항이 제한적이며 관찰하지 못한 것에 대해 무지함
- Sparse Interaction의 문제가 있음 기억하는 값들만 잘 찾기 때문에 Overfitting의 문제가 있음

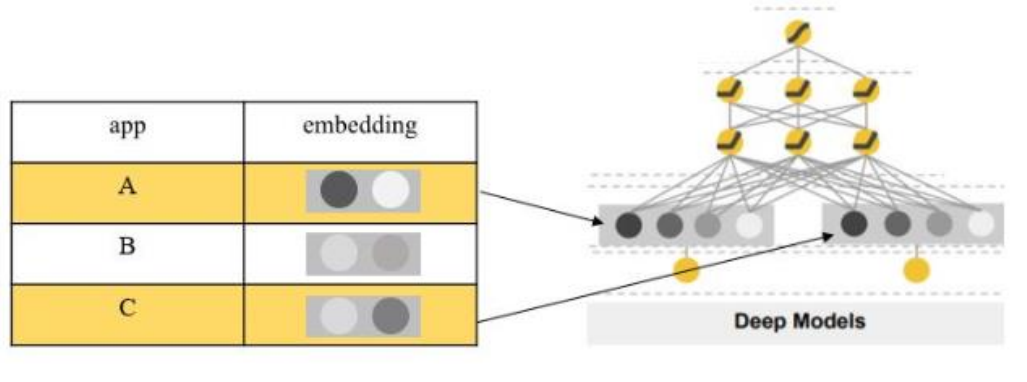
The wide component

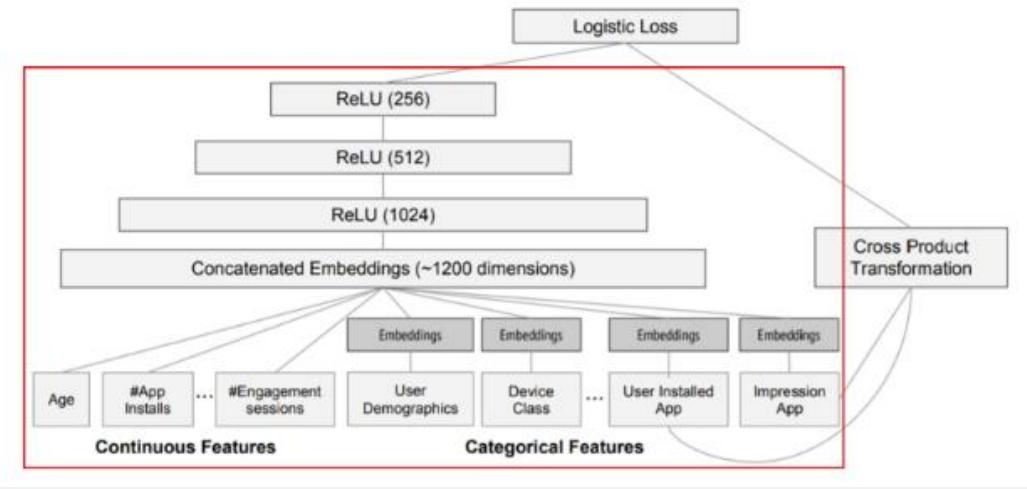


03-2 The deep component

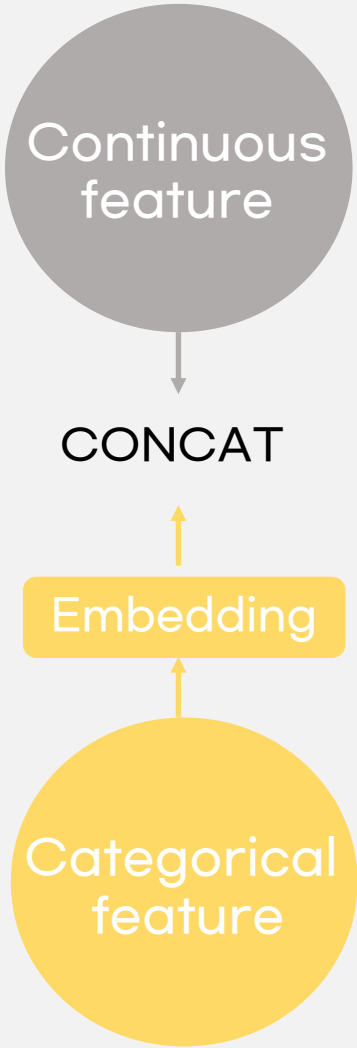
OVERVIEW

SCHEME





INPUT DATA



PROS AND CONS

PROS

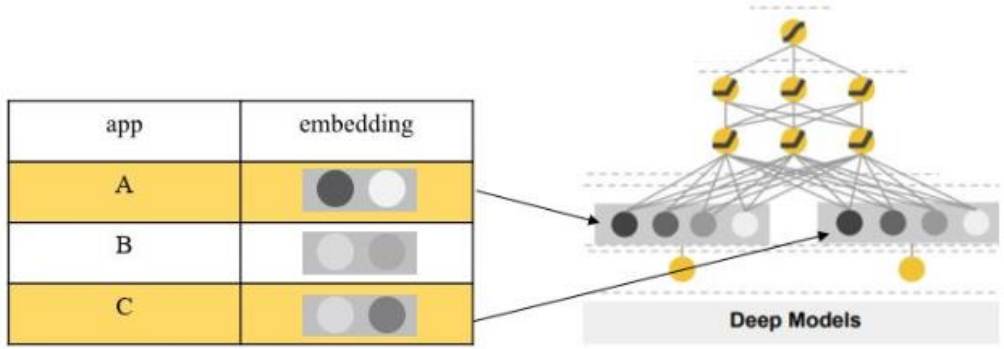
- Generalization
- pair가 없는 관계 학습가능

CONS

전혀 관계없는 아이템들이 추천될 수도 있음

The Deep component

INPUT DATA



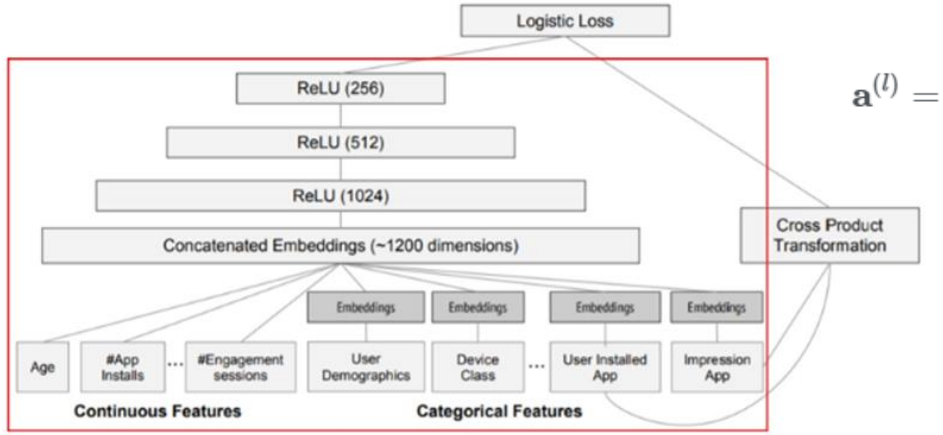
Deep 모델은 A, B, C 앱을 동일한 임베딩 공간에 표현.

-> 단순히 해당 값을 임베딩하여 input으로 넣음

continuous feature와,

임베딩 된 categorical feature를 concat하여 Input data로 사용

Process



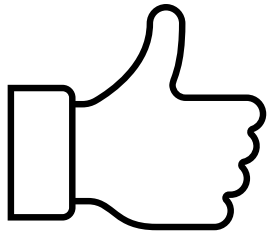
$$\mathbf{a}^{(l)} = f(W^{(l-1)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l-1)})$$

- Input에 가중치 W곱하고 bias더한 것을 활성화 함수에 넣은 전형적인 MLP구조

- 위 전체 도식화 그림에 따르면 총 3개의 layer를 쌓았으며, 활성화 함수로 ReLU를 사용

The Deep component

PROS & CONS

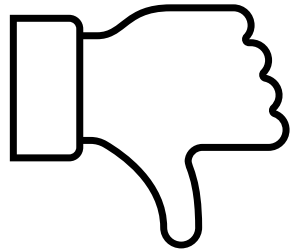


- A, B, C 를 각각 임베딩하여 input으로 넣기 때문에 wide에서의 단점인 pair가 없는 관계 학습 가능

  즉, Generalization에 강점을 가지고 있음

- 다양성 측면에서 개선이 가능하고, 적은 횟수의 Engineering을 이용,

  저차원의 Dense한 임베딩을 통해 일반화를 할 수 있다는 장점

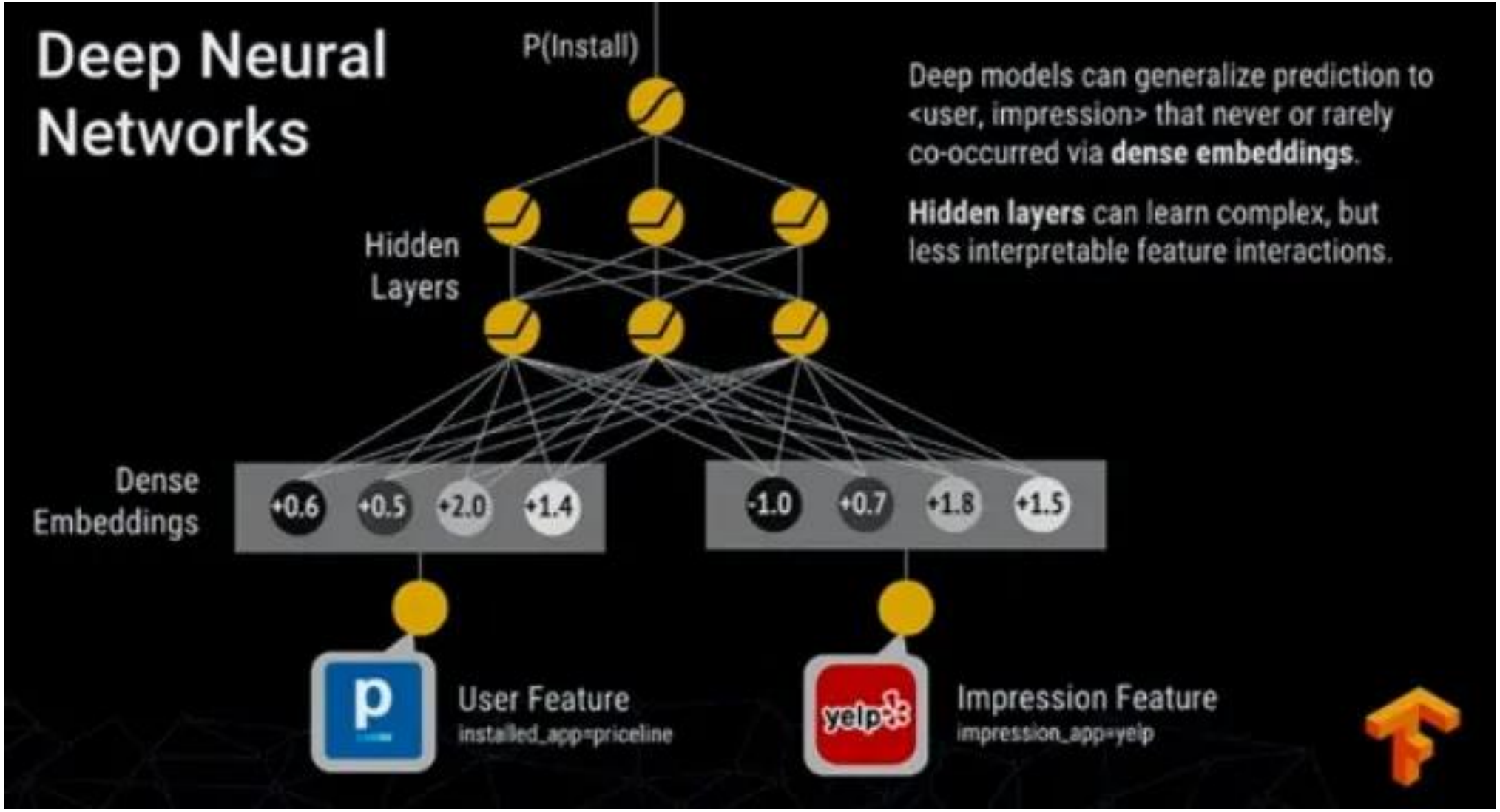


- Sparse하고 High-Rank의 경우에는 성능이 떨어짐

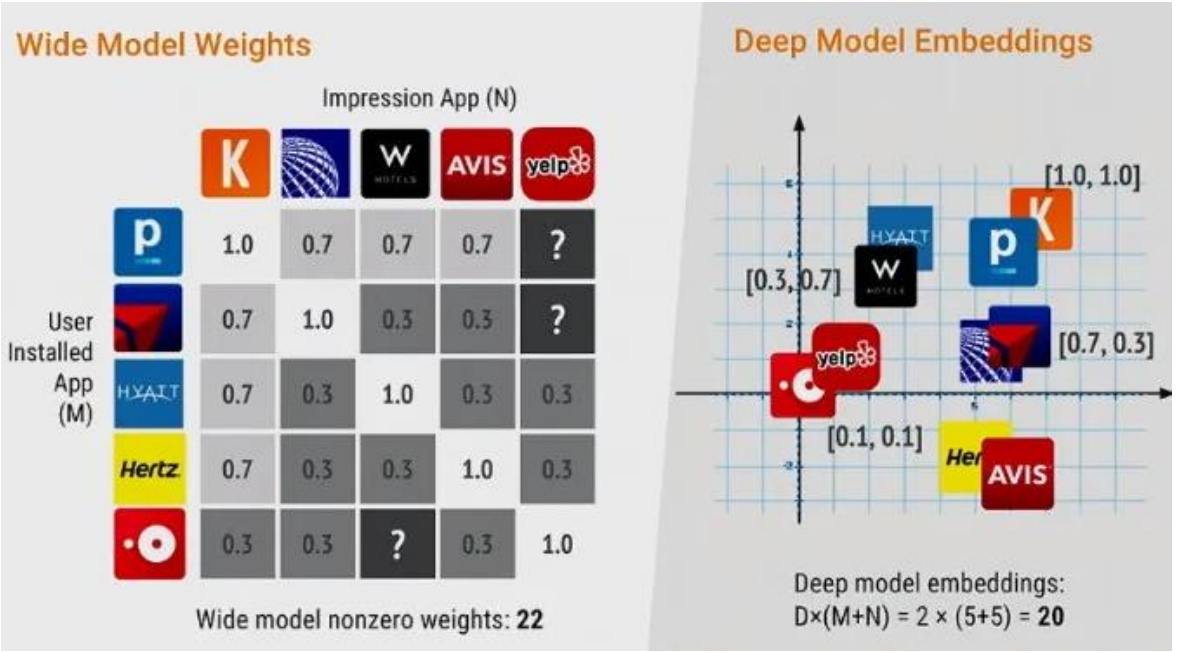
- pair가 없었던, 앱들은 다른 앱과의 관계를 제대로 표현하지 못한 임베딩 벡터를 가질 가능성이 큼

- 희소한 앱들은 학습이 잘 안되기 때문에 전혀 관계없는 아이템들이 추천될 수도 있음 (Underfitting)

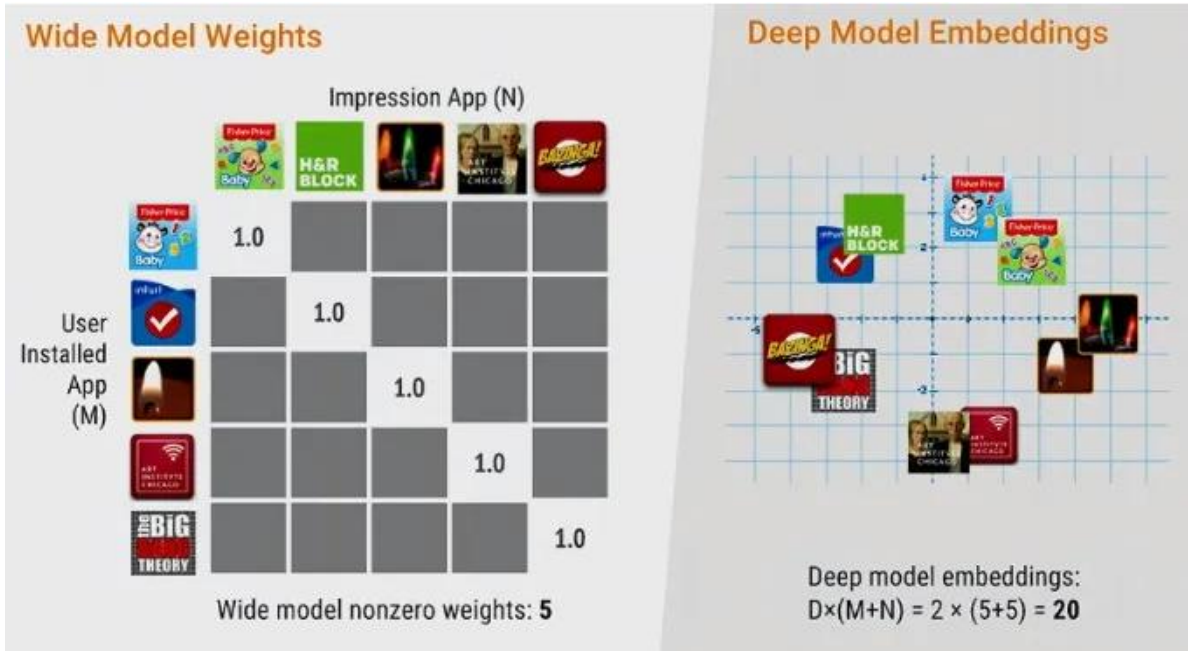
The Deep component



WIDE vs DEEP



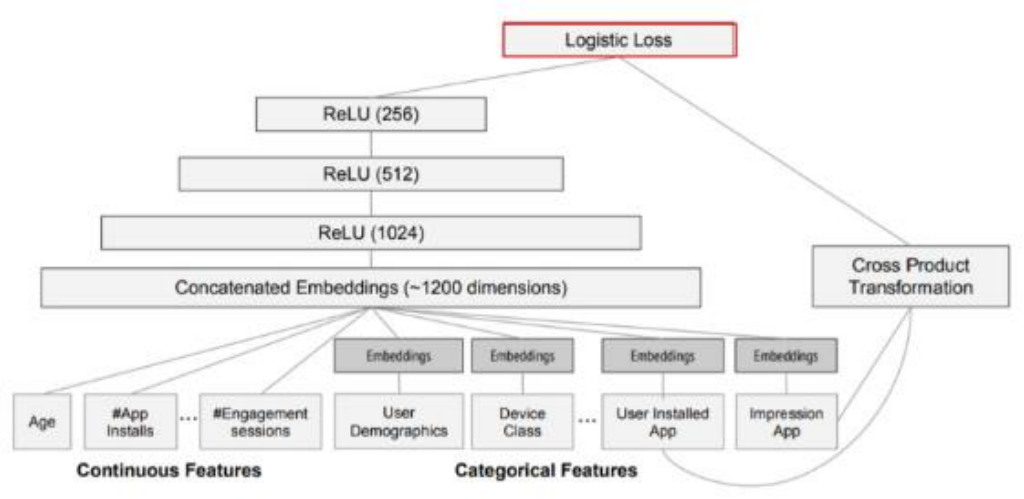
이와 같은 경우, Deep의 방법이 더 효율적



이와 같은 경우, Wide의 방법이 더 효율적

Joint training of wide & deep model

SCHEME



$$p(Y=1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

INPUT DATA

- joint training은 여러 개의 모델을 결합하는 앙상블과 달리. Output의 gradient를 wide와 deep모델에 동시에 backpropagation하여 학습

- 논문에 따르면, wide 모델에서는, optimizer로 online learing 방식인 Follow-the-regularized-leader(FTRL) 알고리즘을, deep 모델에서는 Adagrad를 사용

결과 도출

- W는 wide와 deep동시에 역전파로 학습, 각각을 각자의 output과 곱한 후 더하여 sigmoid에 넣은 것이 최종 결과값
- 결과값은 해당 앱을 추천에 포함할 확률

03 Wide & Deep Learning

Joint training of wide & deep model

SCHEME

$$p(Y=1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

INPUT DATA

- joint training은 여러 개여 모델을 결합하는 앙상블과 달리. Output의 gradient를 wide와 deep모델에 동시에 backpropagation하여 학습

Joint Training : Ensemble와는 다르게 동시에 학습하여 서로를 보완

- 논문에 따르면, wide 모델에서는, optimizer로 online learing 방식인 Follow-the-regularized-leader(FTRL) 알고리즘을, deep 모델에서는 Adagrad를 사용

결과 도출

- W는 wide와 deep동시에 역전파로 학습, 각각을 각자의 output과 곱한 후 더하여 sigmoid에 넣은 것이 최종 결과값
- 결과값은 해당 앱을 추천에 포함할 확률

Joint training of wide & deep model