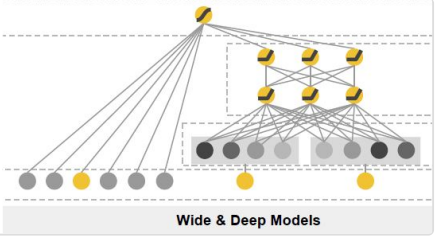
Wide_Deep Learning for Recommender System

[논문 리뷰] Wide & Deep Learning for Recommender Systems

Recommender System

https://leehyejin91.github.io/post-wide_n_deep/



Abstract

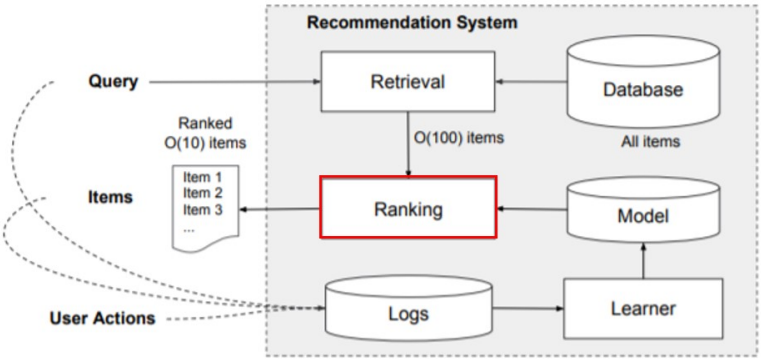
- regression이나 classification 문제에 대해서는 선형 모델을 사용한다.
- 이때 feature간의 cross-product는 데이터의 특징을 추출하고 저장하는데 적합하다.
  - 하지만 데이터의 일반화를 위해 전처리 및 후처리 과정을 거쳐야 한다.
  - 현 논문에서는 이 부분을 **Memorization** 에 특화된 **linear wide 모델**이라고 칭한다.
- 이와 다르게 nn 모델을 사용하는 embedding 방식에 있어서 이 과정들을 덜 거쳐도 된다.
  - 모든 feature간의 데이터를 학습하는데 있어 좋은 접근방법이다.
  - 하지만, 일반화에 초점을 맞추다보니 제대로 데이터를 활용하지 못할 수 있다.
  - 현 논문에서는 **generalization** 에 특화된 **non-linear Deep 모델** 이라고 칭한다.
- 현 논문에서는 위 2가지 모델을 결합한 wide and deep 모델을 제안한다.

Introduction

- 추천시스템은 search ranking system이라고 볼 수 있다.
- 이때, 유저와 contextual information가 input query, item의 rank list가 output이다.
  - query가 주어지면, db에서 제일 유사한 아이템들을 클릭 수 혹은 구매 여부와 같은 데이터를 기반으로 rank를 매겨서 뽑는다.
- 보통 search ranking system과 마찬가지로 `memorization` 과 `generalization`의 특징을 모두 가져야 하는 목표를 갖고 있다.
  - 여기서 memorization이란, 아이템들의 빈도수 혹은 과거 데이터를 통해 경우의 수를 학습하는 경향이다.
  - Generalization이란, 과거에 없었던 feature들을 학습할 수 있는 경향이다.

    - 하지만, low-dimensional 표현으로 query-item을 embedding한 데이터를 학습하는 것은 쉽지 않다. 만약 sparse하거나 raw 데이터의 차원이 크다면, overfitting이 된다.
    - 데이터의 unique case에 대해서도 nonlinear embedding 방식으로 인해 `nonzero prediction`을 예측하는 것이 큰 문제다.
    - 이 단점은 `linear model과 cross-product feature transformation` 방식으로 `memorize` 하는 방법이 해결할 수 있다.
- 이 논문에서 제안된 추천시스템은 구글 플레이의 앱 추천에 사용되며, **구체적으로 user의 검색 query를 바탕으로 생성된 candidate 앱을 정렬하는데 적용된다**

Recommender system overview



1. DB에 들어있는 데이터가 많기 때문에 `retrieval` 을 먼저 진행한다.
    - 선택된 데이터들은 제일 매칭이 잘되는 query들로 1차로 추출된다.
2. 그 다음 `ranking` system은 그 데이터들의 순위를 매긴다.

$$P(y|x)$$

- `유저 정보인 feature x들에 대해 아이템 정보 label y에 대한 확률`을 기반으로 rank가 정해진다.

WIDE & DEEP LEARNING

`wide 모델과 deep 모델의 특징에 대해 간단히 정리해보자.`

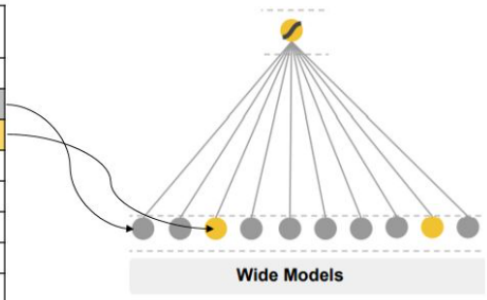
$$\text{user\_install\_app} = [A, B]$$
$$\text{user\_impression\_app} = [A, C]$$

- 위와 같이 데이터가 2개 주어진다고 가정해보자.
- user가 설치한 앱의 `feature와 열람한 앱의 feature간의` `interaction`을 input으로 사용한다고 하자.

- 여기서 앱은 A,B,C로 3개 존재한다.

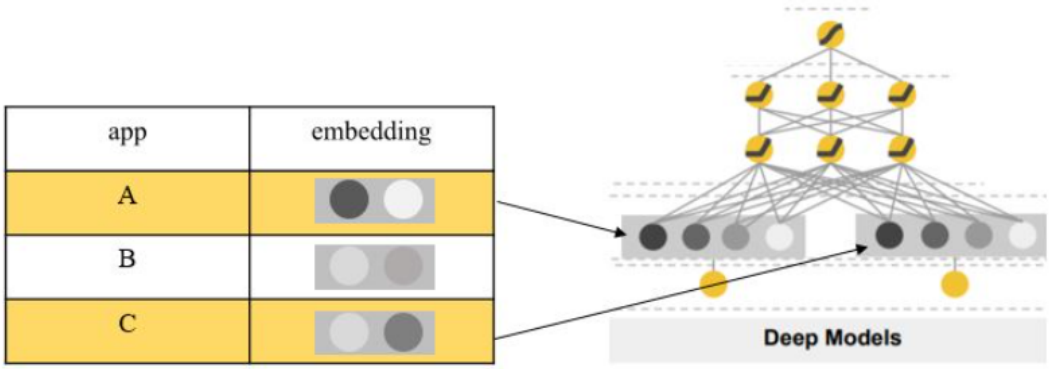
1. **Wide 모델**

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



- 설치한 앱과 열람한 앱 간의 cross-product를 통해 `interaction`을 표현한다.
- 예를 들어 user가 A를 Install 했고, C 앱을 본 상황은 (A,C)=(1,1) 로 표현되고 1*1값으로 1이 나온다.
  - 이런 식으로 3*3의 경우가 존재하고 위와 같이 경우의 수 중 1의 값은 총 4가지다.
  - Install 된 앱과 impression 된 앱이 모두 1이면 cross product값이 1이 되는 셈이다.
- 여기서 1이 나온 `모든 경우의 수를 학습하기 때문에` `memorization`에 강하다.
- user의 특이취향 (unique case, niche combination)도 학습이 가능하지만, 0이 되는 pair들은 학습에 사용되지 않다는 것이 단점이다.

2. **Deep 모델**

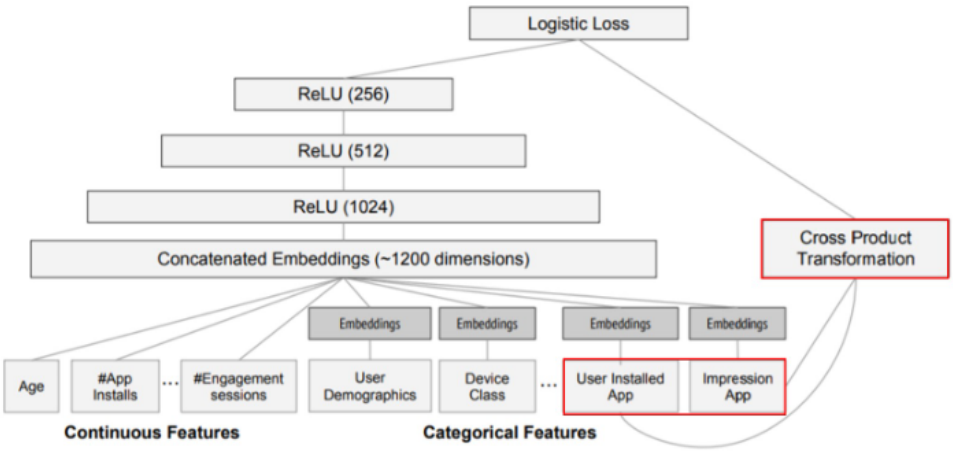


- A,B,C 앱을 동일한 embedding 공간 (해당 예시에서는 2차원)에 표현한다.

- (C,B) 처럼 pair가 없는 관계도 같은 embedding 공간에 표현이 되기 때문에 학습이 가능하다.
- 하지만 niche combination 경우와 같은 데이터를 학습하지 못하기 때문에 너무 일반화된 embedding vector를 갖게 된다.
- 즉 관계없는 아이템들이 추천될 수 있다.

`실제 논문에서 제안된 wide 모델과 deep 모델의 정리해보자.`

1. **wide component**



- user_installed_app, impression_app인 두 feature를 cross-product한 결과 x를 input으로 사용한다.
- 여기서 x는 interaction 정보라고 보면 된다. 위에서 언급된 표 중 `Install x impression column` 정보라고 생각하면 된다.
- 위에서 보이는 cross product transformation은 선형대수에서 보이는 cross-product과 과정이 다르다.
    - **<u>어떻게 다른가?</u>**
        - cross product transformation example
        - 만약 gender, education level, language 3개의 feature가 있다고 생각해보자.
            - 모두 binary feature일때 아래와 같이 case를 나눌 수 있다.

$$\text{gender} = [\text{male}, \text{female}] = [1, 0]$$
$$\text{education} = [\text{high}, \text{low}] = [1, 0]$$
$$\text{language} = [\text{eng}, \text{kor}] = [1, 0]$$

- user가 male이면서, education이 low에 해당하고 언어는 kor를 사용한다고 한다면, user A 벡터 x는 다음과 같이 표현 가능하다.

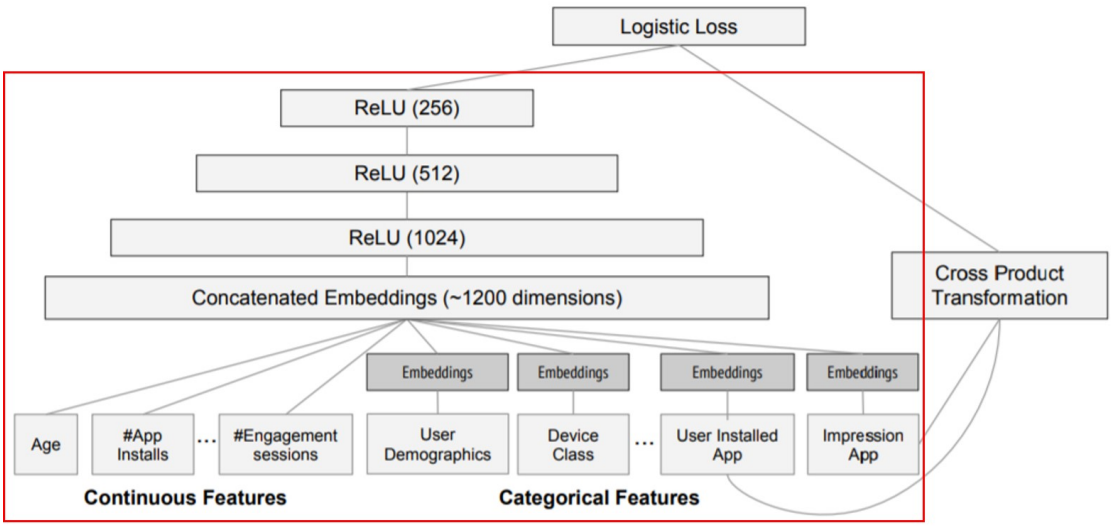
$$\mathbf{x} = [\text{gender}, \text{education}, \text{language}] = [\text{male}, \text{low}, \text{kor}] = [1, 0, 0]$$

- 만약 2가지 경우의 feature transformation ϕk(x)(k=1,2) 방법이 있다고 하자. 예를 들어 ϕ2(x)가 gender와 language의 조합을 나타내는 transformation이라고 한다면, ϕ2(x)는 다음과 같이 formulation 가능하다.

$$\phi_2(\mathbf{x}) = x_1^{c_{21}} x_2^{c_{22}} x_3^{c_{23}} = 1^1 0^0 0^1 = 0, \quad 0^0 \equiv 1$$

- 이때, c2i는 i(i=1,2,3)번째 feature를 ϕ2 transformation에 사용하는지 여부를 나타낸다. 예를 들어 ϕ2는 gender와 language를 사용하는 변환이므로, c2=[c21,c22,c23]=[1,0,1]가 된다.
- user A의 경우 male이면서, eng를 사용하지 않으므로 transformation 값은 0이 된다.

2. **deep component**

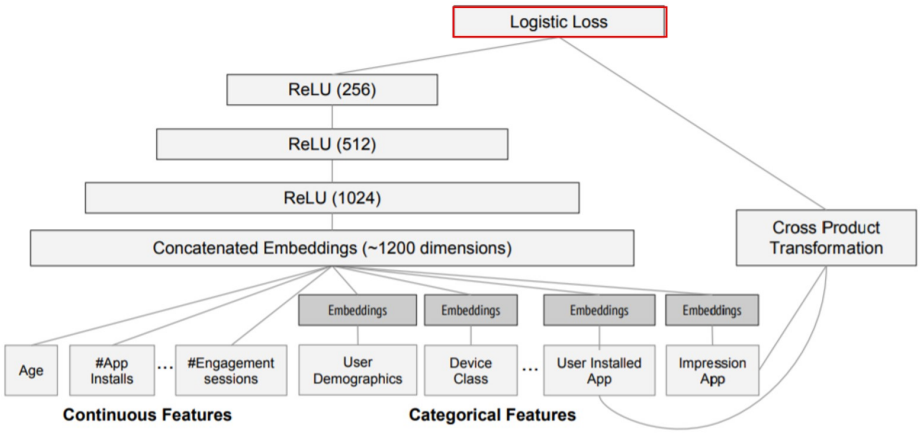


- deep 모델은 continuous feature와 embedding된 categorical feature를 concat한 값을 input(a)으로 사용한다.

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

- 위 보이는 수식처럼 그냥 nn layer를 거치는 것을 볼 수 있다.
- 논문에서는 3개의 layer를 사용했고, relu를 사용했다고 한다.

Joint training of Wide & Deep model



- 저자는 wide 모델과 deep 모델을 joint training 하는 방법을 사용했다고 한다.
  - joint training이란 여러 모델을 결합하는 앙상블과 달리, output의 gradient를 wide와 deep 모델에 동시에 backpropagation을 사용하는 방식이다.
  - optimizer 사용
    - wide 모델: Follow-the-regularized-leader(FTRL) 알고리즘을 사용했다.
    - deep 모델: Adagrad 을 사용했다.
- prediction layer

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

  - 특정 앱을 선택할 확률을 예측하는 것이다.
  - 위 식에서 알 수 있듯이 sigmoid를 사용하여 output을 도출한다.

System Implementation

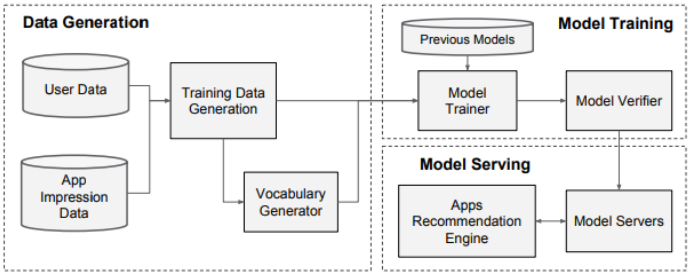


Figure 3: Apps recommendation pipeline overview.

Data Generation

- 이 단계에서는 일정 시간 동안 유저와 앱 데이터를 train data로 사용한다.
  - app data는 app을 사용자가 설치했는지에 여부에 따라 1 or 0
- 데이터 구성
  - Categorical feature
    - 논문에서 vocabularies라는 용어가 나왔는데 categorical feature의 mapping table이라고 보면 된다. 즉, feature 생성하는 단계라고 보면 된다.
  - Continuous feature
    - 정규화는 연속적인 실수 값 변수는 변숫값 x를 누적 분포 함수 n분위수로 나뉜 누적 분포함수 에 맵핑해 [0, 1]로 mapping한다.

Model Training

- input layer는 training data and vocabulary를 사용해 `sparse하고 dense한 feature`들을 생성할 수 있다.
  - wide component: 사용자가 설치한 앱과 노출된 앱의 cross prodcut transfromation으로 구성
  - deep component: categorical feature들을 32차원 embedding vector로 만들어진다.
- 모든 embedding vector들과 continuous feature (Dense)를 concat시켜서 1200 차원의 data를 생성한다.
- 이후 deep component는 relu layer를 3번 거치면서 logistic output을 내뱉낸다.

new data training

- 새로운 데이터가 들어올때마다 재학습을 진행해야 한다.

- 하지만 처음부터 다시 학습하는 것은 굉장히 시간이 걸린다.
- 이 문제를 해결하고자 warm-start으로 이전에 학습된 모델을 불러와서 재학습을 시킨다.

Model serving 하기 전

- 모델 서빙하는 경우에 학습된 모델을 사용하기 전에 dry-run을 진행한다.

> `Dry run 이란`

  - dry run은 모델 학습 혹은 추론 전에 다시 한 번 `디버깅하는 역할을 진행`하는 것
    - 코드 상 오류가 없는지 체크하는 과정이라고 보면 된다.
  - dry run이 진행되는 동안, 합성 데이터 혹은 subset 데이터에 대해서 모델을 검증한다.

Model serving

- 매번 request가 들어오면, 서버는 app candidate들을 app retrieval system으로부터 받는다.
- 그리고 user feature들도 받는데 각 app마다 점수를 평가하기 위해서다.
  - 유저들에게 app의 성적을 내림차순으로 반환한다.
- 여기서 저자는 multi threading을 통해 small batch로 parallel하게 inference를 진행했다.
  - 하나의 batch에서 모든 candidate에 대해서 score를 매기는 방법을 지양했다.

Experiment Result

- 실제 real-world 에서 성능 평가를 진행했다.

App Acquistion

Table 1: Offline & online metrics of different models. Online Acquisition Gain is relative to the control.

Model	Offline AUC	Online Acquisition Gain
Wide (control)	0.726	0%
Deep	0.722	+2.9%
Wide & Deep	0.728	+3.9%

- `3주 동안 online A/B 테스트를 진행했다고 한다.`

    - 1% 유저에게 wide방식만 사용한 이전 recommendation 진행했다고 한다.
    - 그리고 wide & deep model 방식을 1% 유저에게 적용해보았다.
        - wide & deep 방식이 3.9% 더 높은 성능을 보였다.
        - deep only 방식과도 비교했을 때 1% 성능이 더 높다는 것을 알 수 있다.
- `offline 측정방식으로 AUC score를 측정했다`.
    - offline AUC 성능에 큰 차이가 없음을 알 수 있다.
    - online이 더 성능이 크게 차이 나는 이유는 offline은 아예 `fixed` 된 상태이지만, online은 `새로운 데이터에 대한 추천`으로 memorization과 generalization을 적용할 수 있기 때문이라고 한다.

Serving performance

Table 2: Serving latency vs. batch size and threads.

Batch size	Number of Threads	Serving Latency (ms)
200	1	31
100	2	17
50	4	14

- 앱스토어에서 대용량 처리율과 빠른 속도로 추론하는 것이 어렵다.
    - 트래픽은 1000만을 넘긴 적도 있다고 한다.
- single thread로 31ms 추론 시간이 걸렸다고 한다.
    - 여기서 batch를 더 쪼개서 multi-threading으로 14ms로 시간을 줄였다고 한다.

Related work

1. Recommendation 계열
    - 이전 연구 중 linear 모델을 일반화하는 Factorization machine에 착안하여 이 논문을 저술했다고 한다.
        - non-linear의 일반화를 nn 네트워크를 사용해 가능하게 했기 때문이다.
        - linear 계열의 dot-product 방식의 일반화에 주목했던 것으로 판단한다.
    - 아래 2가지 경우와 다르게 CF based 접근법에서 벗어난 실험을 진행한 것에 의의를 두고 있다.
        - content information 와 collaborative filtering을 학습시키는 경우도 있다고 한다.

    - 그리고 CF based 앱 추천 실험도 있었다고 한다.

2. CV 계열
    - skip connection 방식을 사용하여 학습이 잘되는 경우가 있다.
        - 저자는 sparse한 feature와 output unit을 connect한 시도를 진행했다.
    - 또한 pose estimation 에서 joint training을 사용하여 실험하였다고 한다.
        - 저자는 linear 모델과 neural network의 joint training을 시도했다.
3. NLP 계열
    - rnn 계열 모델과 entropy n gram 모델과 joint training을 통해 모델 복잡도 줄였다고 한다.
        - input과 output unit에 대한 직접적인 가중치를 학습하기 때문에 줄었다고 한다.

Conclusion

- memorization과 generalization은 모두 추천시스템에서 중요한 요소다.
- wide linear model은 sparse feature에 대한 interaction을 memorize가 가능하고, nn 구조는 unseen data에 대해 일반화가 가능하다.
- 저자는 2가지 특징을 모두 가진 모델을 제안했으며, online experiment를 통해 좋은 성능을 평가할 수 있었다.