풀잎스쿨 13기

코드로 배우는 추천 시스템 2주차

이수진

Index

- Deep Neural Networks for Youtube Recommendations
- Wide & Deep Learning for Recommender Systems

Deep Neural Networks for Youtube Recommendations

01 Youtube RecSys paper
Introduction

- Youtube RecSys paper
  - https://static.googleusercontent.com/media/research.google.com/ko//pubs/archive/45530.pdf
  - 딥러닝 기반 추천 시스템 논문 중 가장 유명한 논문 중 하나
  - 많은 Insight를 도출 할 수 있는 논문

- 3가지 주요 관점이 존재
  - Scale
    - 데이터가 너무 많기 때문에 scalability가 중요함
  - Freshness
    - 새로운 영상이 추천에 빠르게 반영
  - Noise
    - 데이터의 sparsity
    - Implicit feedback
    - Metadata 엉망 등

01 Youtube RecSys paper
System overview

- 해당 논문의 모델은 아래와 같은 특징을 지님
  - 2단계 구조를 띄고 있음
    - Candidate Generation : 후보군 추출
    - Ranking : 랭킹 모델
  - Candidate Generation
    - 사용자의 활동 정보를 사용해 CF 구조로 후보 추출
      - 활동 정보 : 시청 기록, 성별, 연령 등등
    - High-precision으로 볼 것 같은 영상을 추출
      - 사용자들이 자기가 좋아하는 영상을 조금 빠뜨리는 것엔 관대
      - 하지만, 싫어하는 것을 추천하는 것 민감하게 반응
    - Ranking 단계에서 너무 많은 영상이 있으면 계산량 부하
      - Scalability 문제
  - Ranking
    - 더 다양한 정보를 추가하여 사용자가 볼 것 같은 것을 추천
  - 개발 단계에서는 다양한 Metric을 활용했고 A/B test를 계속 진행하였음

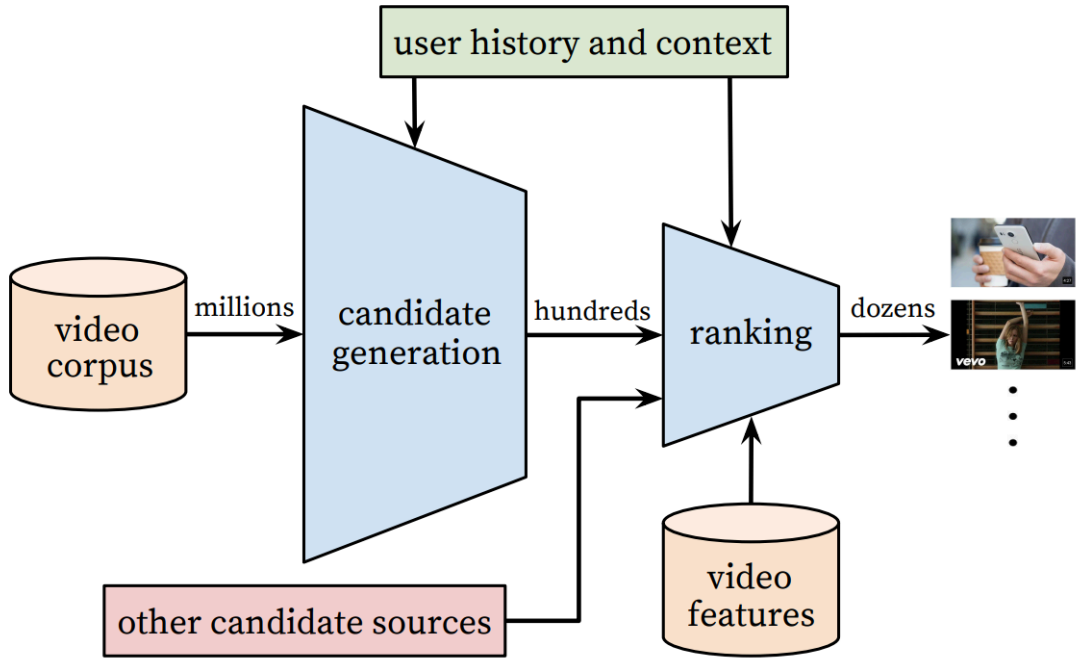


Figure 2: Recommendation system architecture demonstrating the "funnel" where candidate videos are retrieved and ranked before presenting only a few to the user.

01 Youtube RecSys paper
Candidate Generation

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

- Candidate Generation 단계
  - 추천 모델을 Multi classification 문제로 정의
    - 단, 엄청나게 클래스가 많은 Extreme multiclass classification
  - 사용자가 비디오를 끝까지 보았는가? 보지 않았는가?로 사용
    - Explicit feedback은 사용하지 않고 implicit feedback을 사용

- Extreme Multiclass Classification의 효율성
  - Negative sampling으로 학습 속도 개선
    - Softmax 분류의 단점인 class가 많아지면 가능한 모든 class에 내적을 수행해서 계산량이 증가해 느려짐
    - 마치 Word2vec에서 사용한 Negative sampling처럼 이용
  - 정확도는 조금 포기하더라도 속도를 개선
    - 수많은 영상 중 top N을 뽑아내야 하기 때문
    - Multiclass classification에서 top K개를 뽑아내는 문제는 output vector space상에서 KNN search를 하는 것과 동일
    - 즉, 적당히 좋은 top K개의 후보를 빠르게 뽑아내려면 approximate KNN search를 사용한다고 함
      - 여러 KNN search를 사용해봤는데 다 비슷하다고 함

01 Youtube RecSys paper
Candidate Generation

- Candidate Generation 모델 아키텍처

- Input
  - Video Embedding (시청 이력)
  - Search Embedding (검색 이력)
    - Average를 사용
    - 정보를 압축하는 의미라고 함
    - 예시)
      - 마지막 검색은 중요한 정보임
      - 그러나 마지막 검색에 의존하면 질 낮은 추천이 될 수 있음
      - 이러한 정보를 뭉개는 효과(압축)
  - Demographic (인구 통계) 정보
    - Cold start 문제 해결에 도움
  - Example age
    - Video freshness
    - 뒷장에서 더 설명

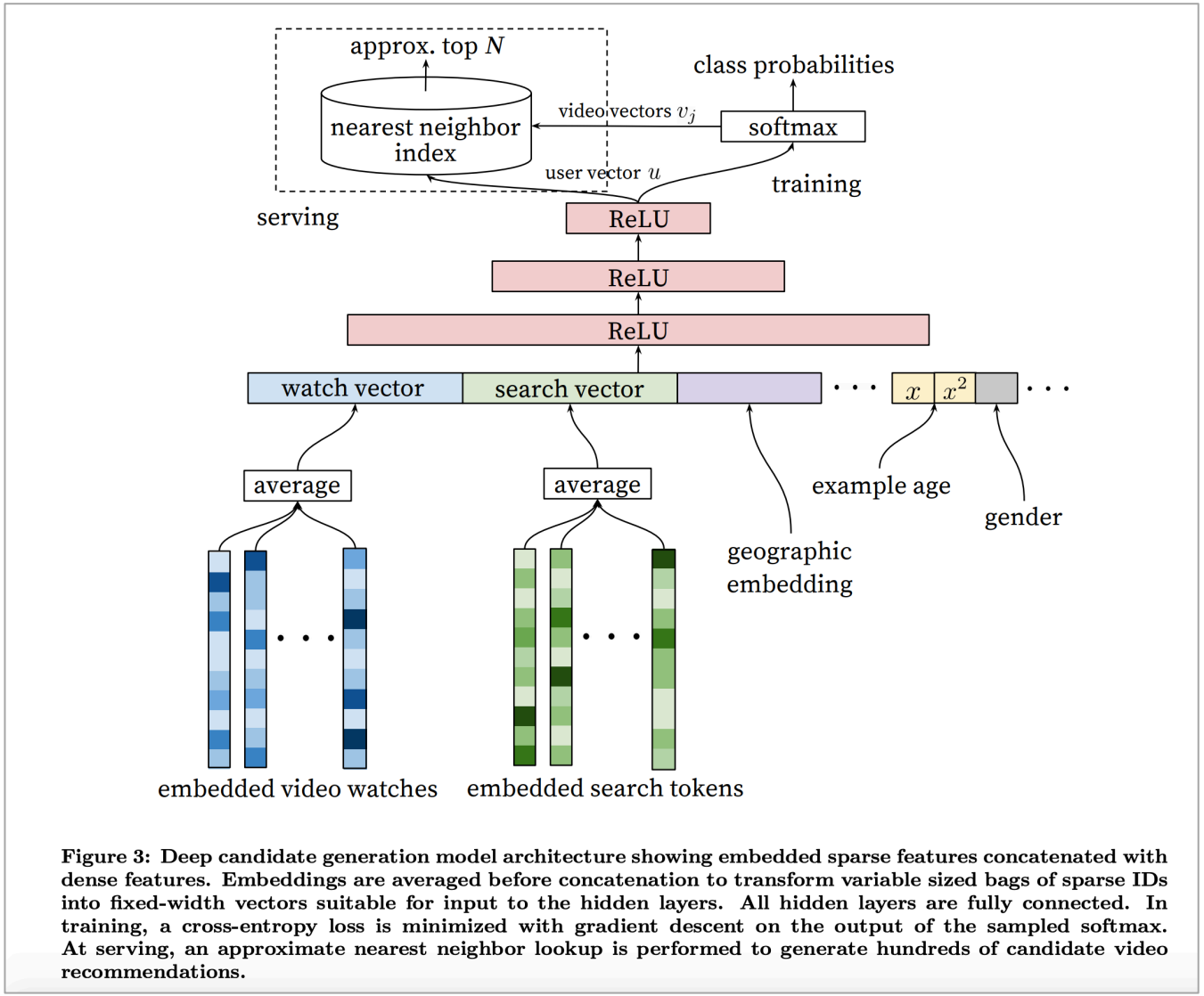
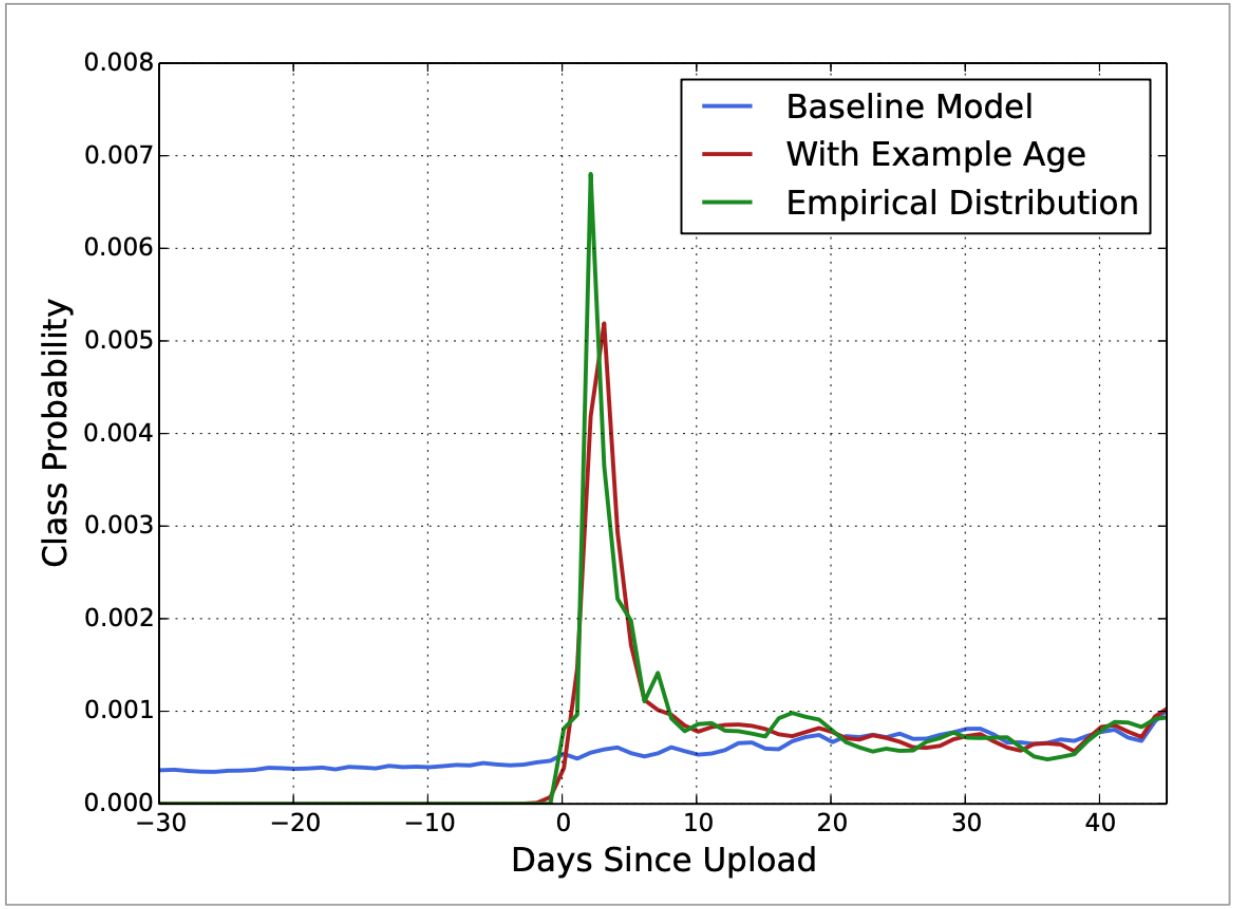


Figure 3: Deep candidate generation model architecture showing embedded sparse features concatenated with dense features. Embeddings are averaged before concatenation to transform variable sized bags of sparse IDs into fixed-width vectors suitable for input to the hidden layers. All hidden layers are fully connected. In training, a cross-entropy loss is minimized with gradient descent on the output of the sampled softmax. At serving, an approximate nearest neighbor lookup is performed to generate hundreds of candidate video recommendations.

01 Youtube RecSys paper
Candidate Generation

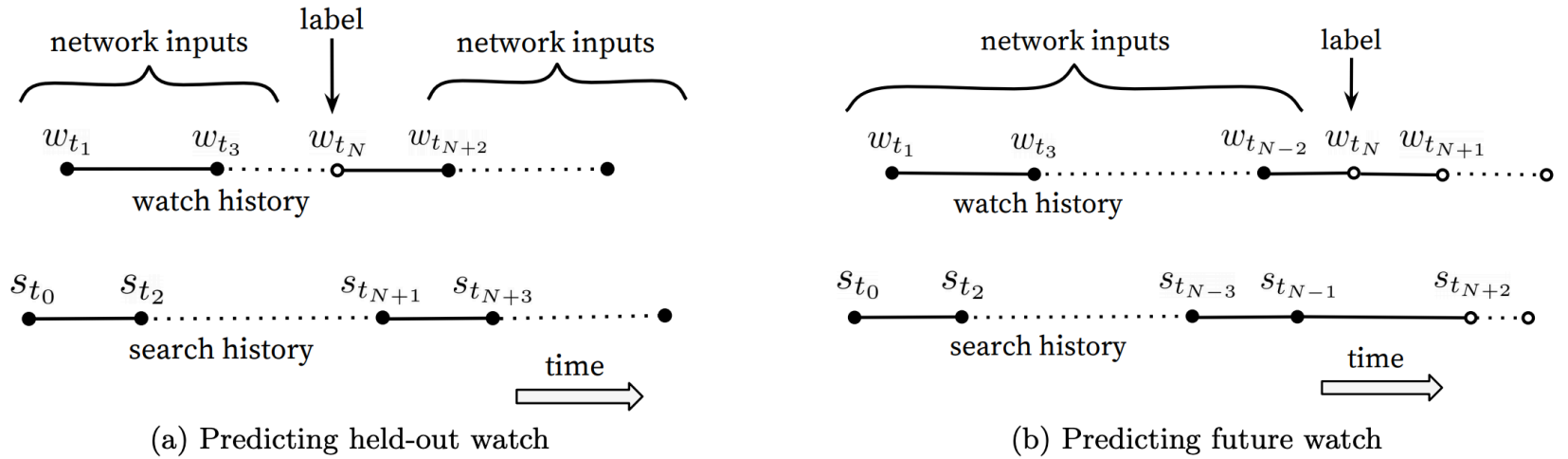
- Model
  - Fully-connected layer + Relu 구조 사용

- Example Age
  - 최근에 업로드 된 비디오를 추천하는 것이 중요
  - 사용자가 얼마나 새로운 비디오를 선호하는지 중요
  - Training data 특성 상 오래된 아이템들이 더 추천 받음
  - 이를 해결하기 위해 Video Age를 넣어줌



01 Youtube RecSys paper
Candidate Generation

- Label and Context selection
  - 시청 이력은 Youtube 내부 뿐 아니라 외부에서 본 것(공유 된 것 등)도 포함해야 bias가 없음
    - 자체 추천 결과만으로 진행하게 되면 heavy user의 bias와 추천의 추천
  - Next prediction을 하는 것이 더 효과적
    - MF 모델 등은 이러한 소비패턴을 따라가지 못함
    - 따라서 특정 시점의 과거 데이터를 가지고 next를 예측하는 방식



(a) Predicting held-out watch

(b) Predicting future watch

Figure 5: Choosing labels and input context to the model is challenging to evaluate offline but has a large impact on live performance. Here, solid events • are input features to the network while hollow events ∘ are excluded. We found predicting a future watch (5b) performed better in A/B testing. In (5b), the example age is expressed as $t_{\max} - t_N$ where $t_{\max}$ is the maximum observed time in the training data.

01 Youtube RecSys paper
Candidate Generation

- Experiments with Features and Depth
  - Depth가 깊어질 수록 성능 좋아짐
  - All Features가 좋은 성능을 보임

- Depth 0: A linear layer simply transforms the concatenation layer to match the softmax dimension of 256
- Depth 1: 256 ReLU
- Depth 2: 512 ReLU → 256 ReLU
- Depth 3: 1024 ReLU → 512 ReLU → 256 ReLU
- Depth 4: 2048 ReLU → 1024 ReLU → 512 ReLU → 256 ReLU

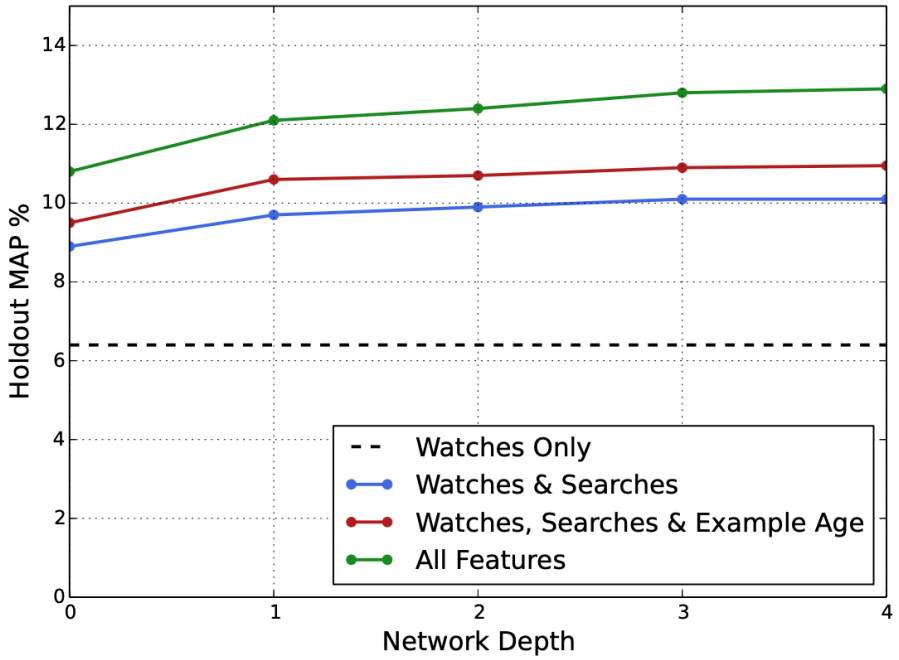


Figure 6: Features beyond video embeddings improve holdout Mean Average Precision (MAP) and layers of depth add expressiveness so that the model can effectively use these additional features by modeling their interaction.

01 Youtube RecSys paper
Ranking

- 앞선 후보군에서 랭킹으로 영상 추천
  - 더 다양한 feature를 추가적으로 삽입
    - Embedding vector
    - Categorical feature
    - Continuous feature
  - 수백 개의 feature 사용
  - Categorical feature
    - 상위 N개를 사용(나머진 OOV)
  - Continuous feature
    - [0, 1] Normalize 등
    - 너가 뭘 좋아하는지 몰라서 다 준비했어 ^^
  - 전통적인 ML 방법(정보)들이 중요한 feature로 많이 보였음
    - Impression video
    - 유저와 video의 interaction 등
  - Watch time으로 가중치를 준 weighted logistic (Weighted cross-entropy loss)
    - 어뷰징 행위를 걸러내기 위함 (낚시성 비디오나)

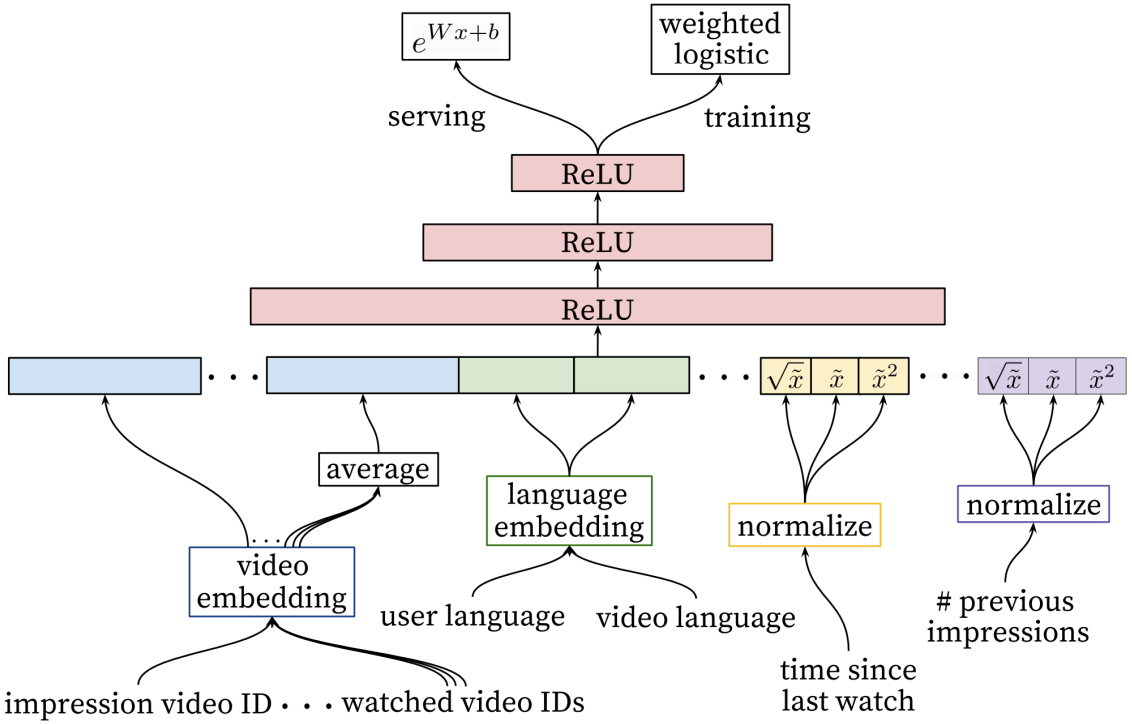


Figure 7: Deep ranking network architecture depicting embedded categorical features (both univalent and multivalent) with shared embeddings and powers of normalized continuous features. All layers are fully connected. In practice, hundreds of features are fed into the network.

01 Youtube RecSys paper
Conclusion

- 추천 논문이지만 뭔가 찜찜
  - 막상 읽으면 모델 보단 엔지니어링
  - Feature Engineering이 매우 중요함
  - 모델은 뭐 ㅇㅇㅇ 그렇다고 함
  - 삽질 열심히 해라.. 라는 교훈

Hidden layers	weighted, per-user loss
None	41.6%
256 ReLU	36.9%
512 ReLU	36.7%
1024 ReLU	35.8%
512 ReLU → 256 ReLU	35.2%
1024 ReLU → 512 ReLU	34.7%
1024 ReLU → 512 ReLU → 256 ReLU	34.6%

Table 1: Effects of wider and deeper hidden ReLU layers on watch time-weighted pairwise loss computed on next-day holdout data.

Wide & Deep Learning for Recommender Systems

02 Wide & Deep Learning paper
Introduction

- 딥러닝 기반 추천 시스템
  - Youtube와 더불어 유명한 논문 중 하나
  - Wide한 영역의 장점과 Deep한 영역의 장점을 살려서 추천 모델을 만드는 구조
    - Memorization(Wide) : cross-product feature. 효과적이지만 엔지니어링 노력이 들어감
    - Generalization(Deep) : 차원 Embedding으로 feature를 결합. 지나친 일반화가 될 수 있음

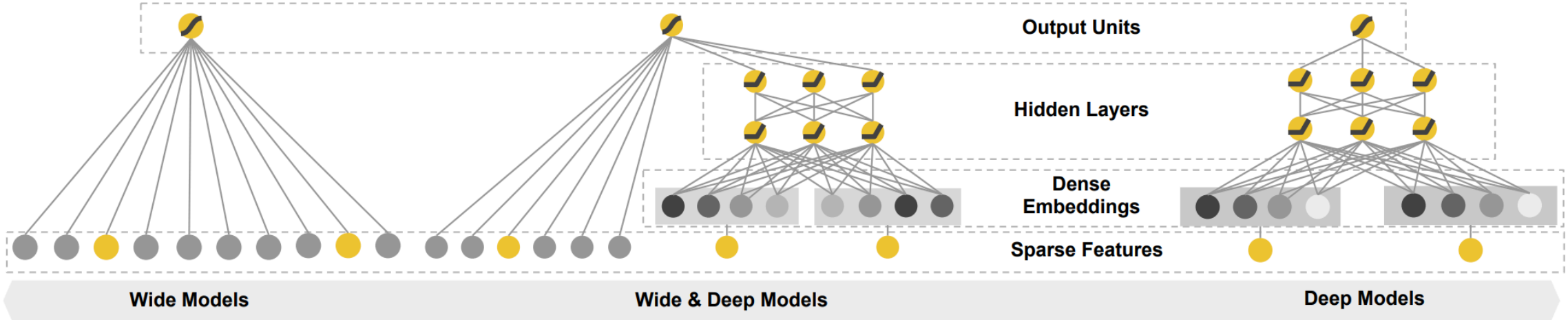
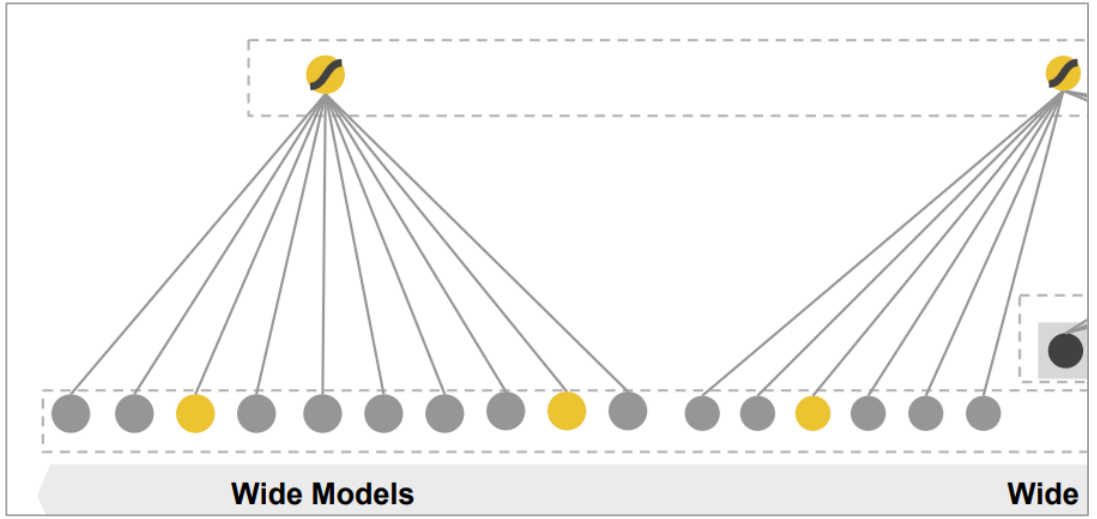


Figure 1: The spectrum of Wide & Deep models.

02 Wide & Deep Learning paper
Introduction

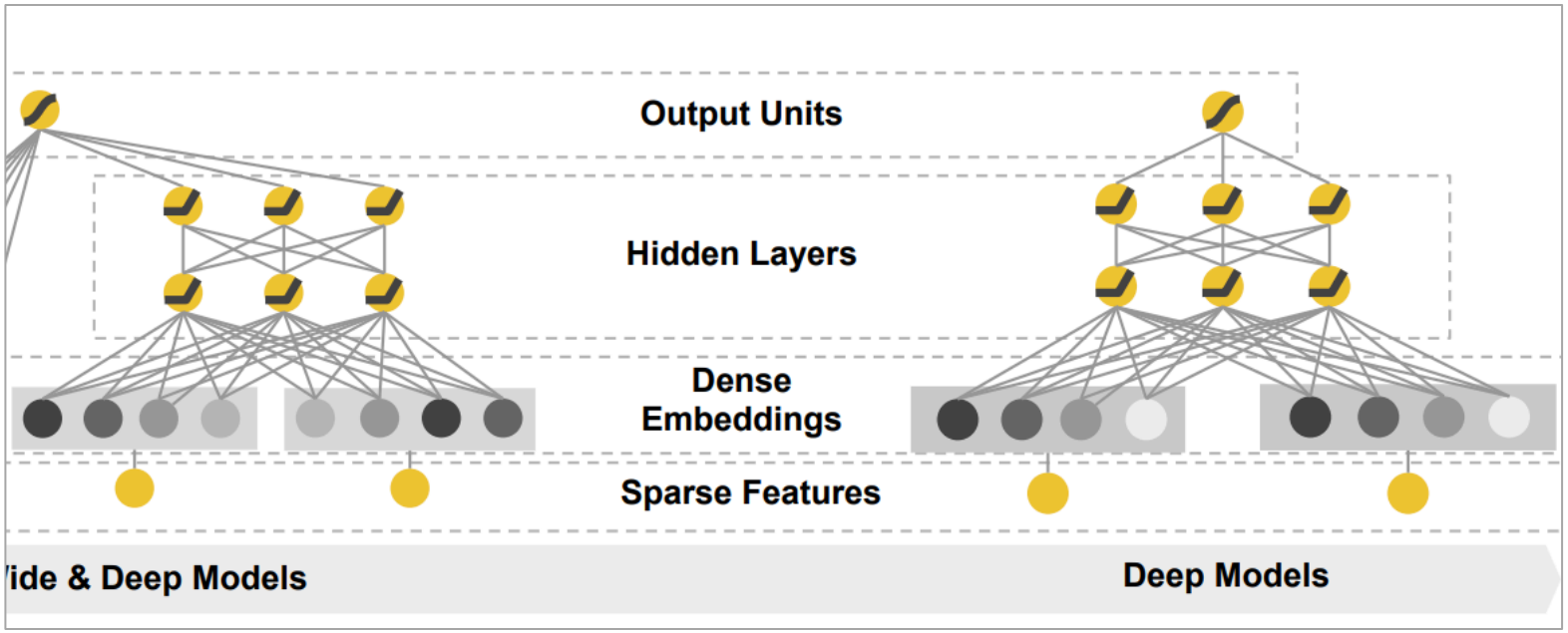


- Memorization
  - Wide 영역에 해당 됨
  - 아이템 상관관계 학습
  - User_install_app, User_impression_app의 목록이 있음
    - 이 둘의 interaction을 이용해 cross-product를 이용함
    - 예시)
      - 유저 A가 있고 '카카오톡', '유튜브'설치 // '카카오톡' 앱을 열람했고 '인스타그램'앱을 열람함
      - 카카오톡 같은 경우 열람했고 설치함 -> (1, 1) -> 1
      - (카톡 설치, 유튜브 미열람) -> (1, 0) -> 0
      - (카톡 설치, 인스타그램 열람) -> (1, 1) -> 1
      - (인스타 미설치, 인스타그램 열람) -> (0, 1) -> 0
  - 위와 같은 방식의 장점
    - Memorization에 강함
    - User Interaction에 강함
  - 단점
    - 그 외 Pair 정보는 학습하지 못함(일반화를 못함)

ple, scalable and interpretable. The models are often trained on binarized sparse features with one-hot encoding. E.g., the binary feature "`user_installed_app=netflix`" has value 1 if the user installed Netflix. Memorization can be achieved effectively using cross-product transformations over sparse features, such as `AND(user_installed_app=netflix, impression_app=pandora`"), whose value is 1 if the user installed Netflix and then is later shown Pandora. This explains how the co-occurrence of a feature pair correlates with the target label. Generalization can be added by using features that are less granular, such as `AND(user_installed_category=video, impression_category=music)`, but manual feature engineering is often required. One limitation of cross-product transformations is that they do not generalize to query-item feature pairs that have not appeared in the training data.

02 Wide & Deep Learning paper
Introduction

- Generalization
  - Deep한 영역
  - 고정된 low dim에 Embedding
  - Interaction이 적은 것(없는 것)들의 feature 결합 제공
  - 일반화에 강함
  - 전혀 관련 없는 데이터가 나올 수도 있음



02 Wide & Deep Learning paper
RecSys Overview

- 추천 시스템의 전체 구조
  - Youtube와 마찬가지로 2-step 구조를 가지고 있음
  - 본 논문에서는 Ranking쪽에 focus를 맞춤
  - 즉, Wide & Deep model은 Ranking 모델이라고 말할 수 있음
  - 검색 시스템쪽은 various signals를 이용
    - ML model과 Human defined rule에 의해 동작
    - 가장 잘 맞는 Item 후보 리스트 return

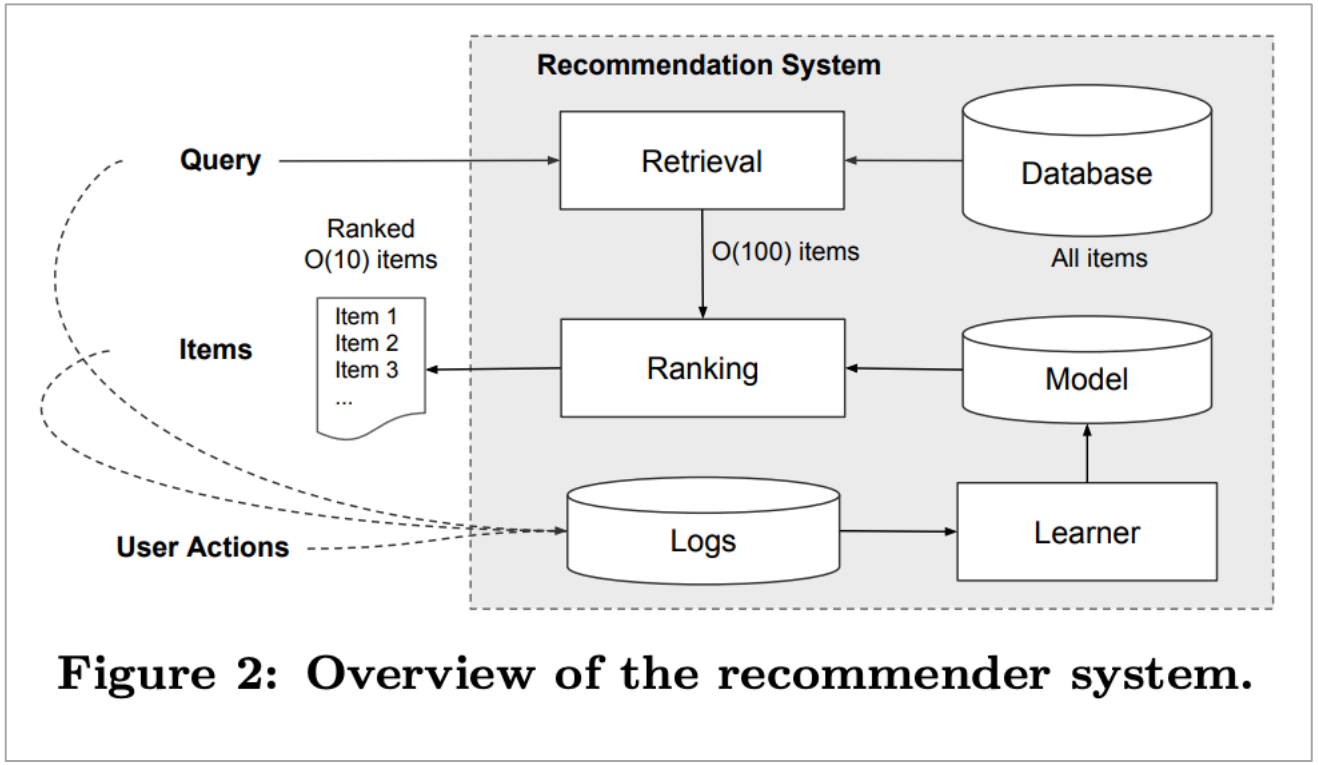
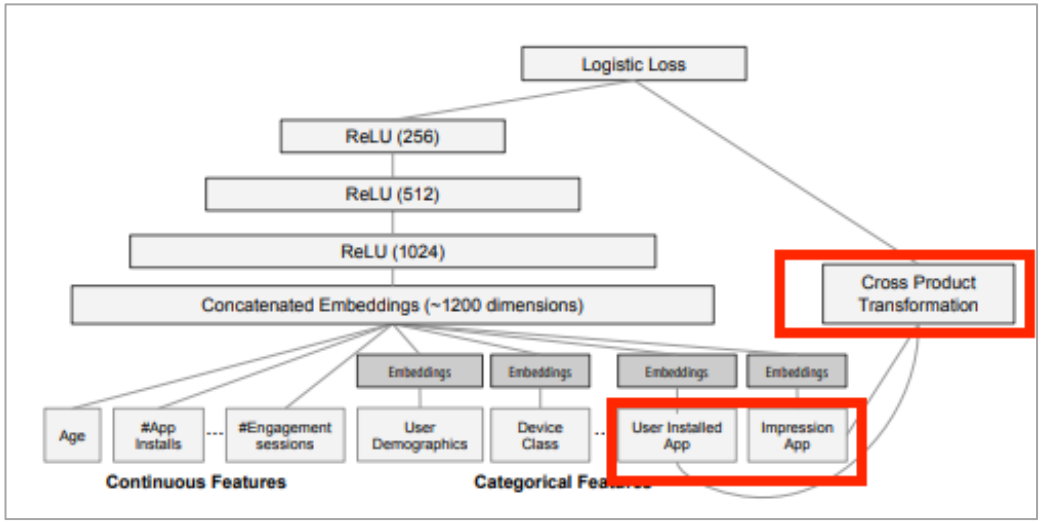


Figure 2: Overview of the recommender system.

02 Wide & Deep Learning paper
Wide Component

- Generalized linear model
  - Y = wTx + b
  - X는 user install app, impression app의 feature를 cross-product한 결과

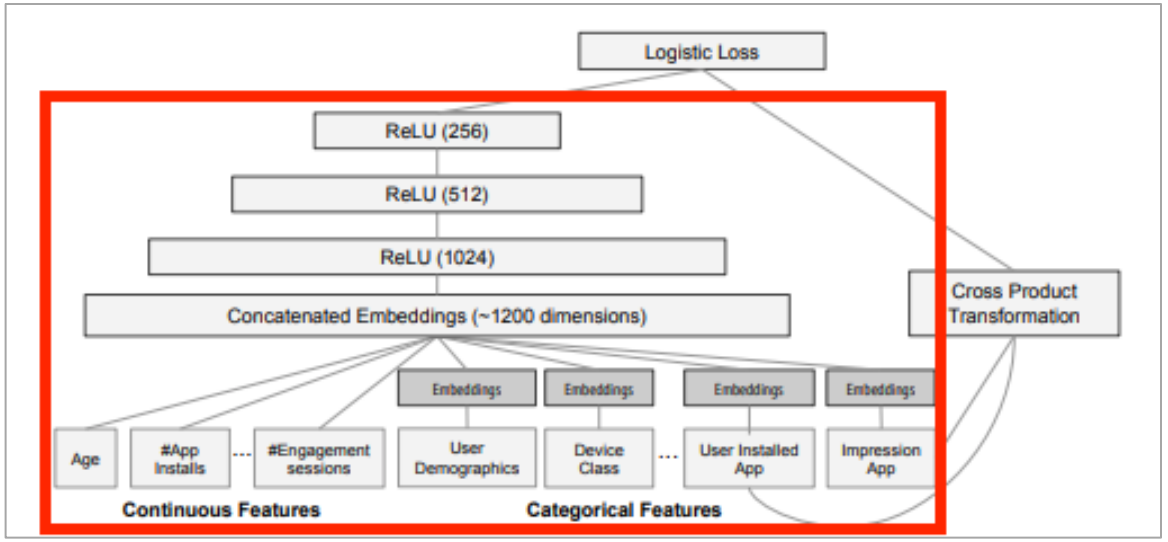
$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$



02 Wide & Deep Learning paper
Deep Component

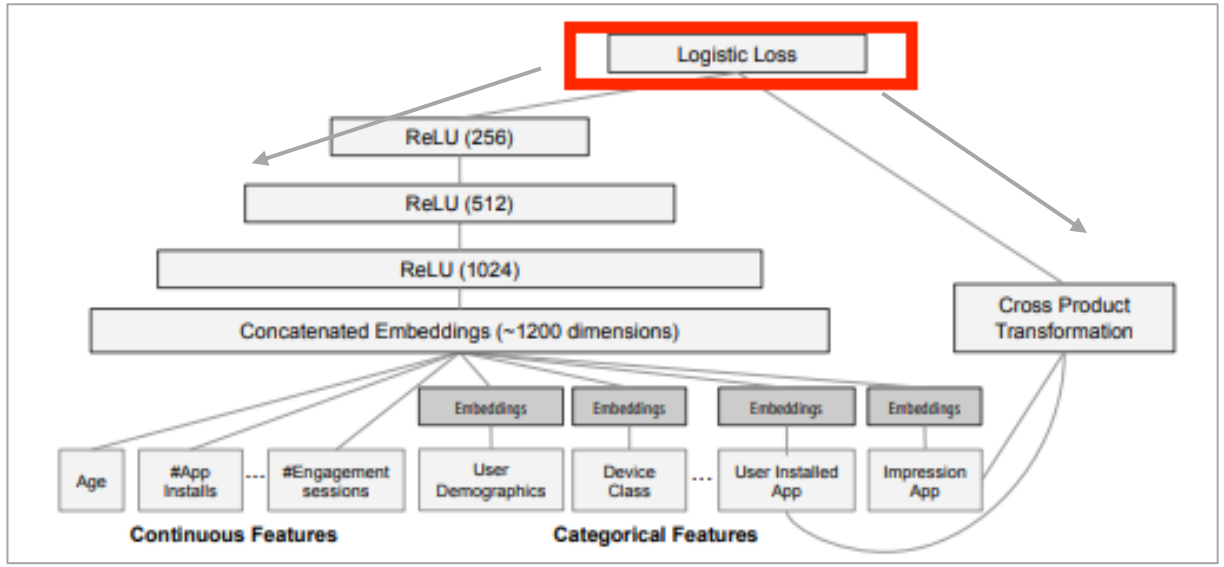
- Dense 모델 (Feed-forward neural network)
  - Continuous feature
    - 그대로 넣음
  - Categorical features
    - Sparse & high-dimensional
    - Low dim으로 변환 (Embedding)

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



02 Wide & Deep Learning paper
Deep Component

- Joint training



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

  - Ensemble
    - 개별 모형을 훈련 (서로 알지 못함)
    - Prediction할 때 결합
  - Joint training
    - Wide와 Deep의 결합으로 나온 output의 gradient를 wide, deep에 뿌려 최적화

02 Wide & Deep Learning paper
System implementation

- Data Generation
  - 특정 기간에 사용자와 app 노출 데이터를 사용
  - App impressed & install 하면 label은 1, 아니면 0
  - Continuous는 [0, 1]로 normalized

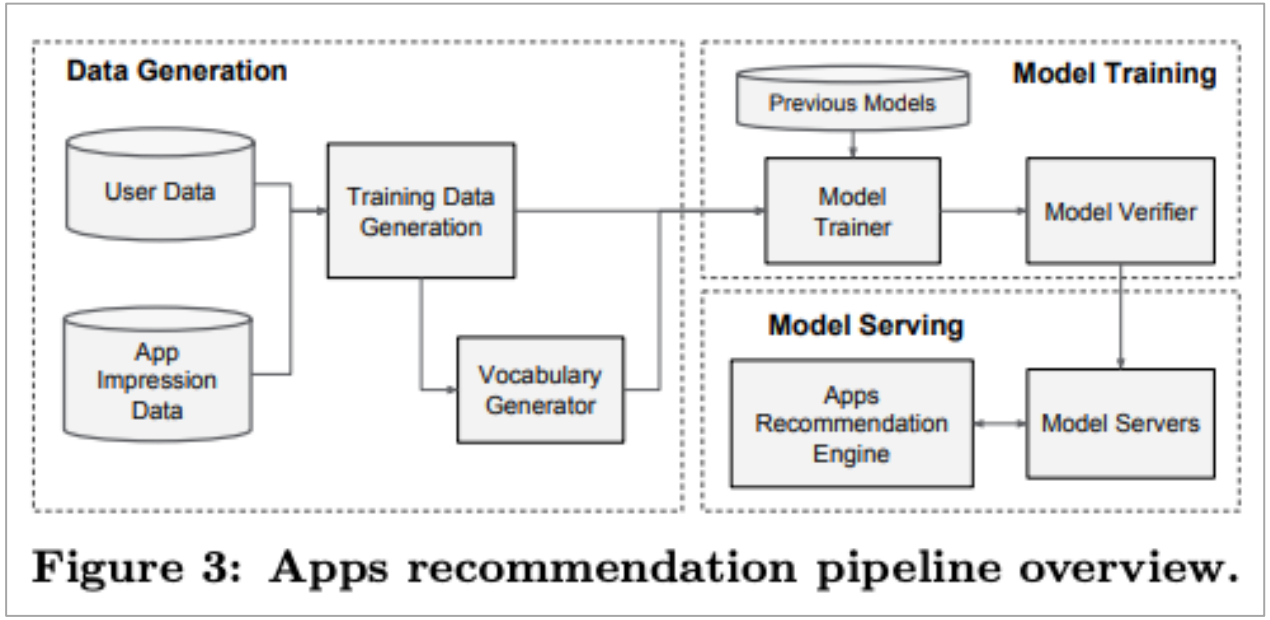


Figure 3: Apps recommendation pipeline overview.

- Model Training
  - Wide & Deep 영역으로 훈련
  - 500 billion (5천억?)개의 sample로 훈련
  - New set of training data가 오면 재훈련
    - 처음부터 훈련하는 것은 비효율적
    - Warm-starting system으로 구현
    - Initializes a new model with the embeddings and the linear model weights from the previous model
- Model Serving
  - Model이 trained되고 verified되면 model server에 load
  - Highest scores to the lowest로 order를 한 뒤 사용자에게 보여줌

02 Wide & Deep Learning paper
Experiment results

- 실험 평가
  - 두 가지 측면에서 평가
    - 앱 가입(App Acquisitions)
    - Serving Performance
- 3주간 A/B test 진행
  - 대조군 : 1% 랜덤 샘플. 이전 버전의 랭킹 모델이 추천한 것 적용
    - Highly-optimized wide only logistic regression model
  - 실험군 : 1% 사용자에게 Wide & Deep 노출

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%

- AUC로 offline에서 평가도 진행
  - Offline에서도 효과가 좋지만 전반적으로 online이 더 좋음
  - Offline은 dataset 등이 fixed되어 있지만 online은 사용자의 새로운 반응 등을 적용할 수 있으므로

02 Wide & Deep Learning paper
Experiment results

- Serving Performance
  - Serving with high throughput and low latency는 어려운 문제
  - 여기서는 트래픽이 가장 높을 때 초당 1천만 개 이상의 앱에 점수를 매긴다고 함
  - Single threading으로는 31ms가 걸림
  - 따라서 multithreading을 구현하고 batch를 더 작게해서 지연 시간을 14ms로 줄임

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

Thanks!