Context-Aware Learning to Rank with Self-Attention

Abstract	Learning rank 는 이커머스 검색 엔진에서 중요한 요소다. learning to rank는 아이템 리스트에서 전체적인 순서를 최적화 하는 것에 관심이 있다. 인기있는 방법은 scoring function을 학습하는 것이다. Scoring function은 pointwise, pairwise or listwise를 최적화하여 아이템을 개별적으로 스코어를 낸다. 이 아이템 리스트를 높은 점수 순으로 정렬한다. 동일한 리스트에서의 아이템들간의 발생할 수 있는 상호관계는 학습단계 level에서 고려된다. 그러나 inference에서 아이템들이 개별적으로 점수화 되지만, 동일 리스트 아이템들간의 상호작용은 고려되지 않는다. 이 논문은 self-attention 방법을 적용한 아이템 스코어 방법을 학습하는 맥락 인지 네트워크(context-aware neural network) 모델을 제안 한다. 주어진 아이템의 관련성은 리스트에있는 모든 항목의 맥락에 따라 결정됩니다.(train 및 inference 모두에서)
ETC	https://www.slideshare.net/JunhoLee124/attention-is-all-you-need-113895030?from_action=save
Paper Keywords	learning to rank, self-attention, information retireval
Paper Link	https://paperswithcode.com/paper/context-aware-learning-to-rank-with-self
발제자	김성주
발표자	석우 강
한글 링크	

1. Introduction

Learning to rank (LTR). 보통의 머신 러닝 솔루션의 LTR 문제는 scoring 함수를 학습하는 것이다. 주어진 리스트의 각 아이템에 진짜 값 들을 할당하거나, 아이템 피처 데이터 셋이나 사람에 의해 수집 되거나 암시적으로 관련된 라벨을 할당해서. 성능 측정은 Mean Reciprocal Rank(MRR), Normalized Discounted Cumulative Gain(NDCG) 나 Mean Average Precision을 사용한다.

분류와 회귀 와 다르게 랭킹 알고리즘의 주요 목표는 아이템 들의 그룹에 대해 상대적인 preference를 결정하는 것이다. 목록의 특정 아이템에 대한 사용자의 preference는 같은 목록에 존재하는 다른 아이템에 따라 달라 진다. 그렇지 않으면 선호되는 아이템은 더 관련성이 높은 다른 아이템이 있는 경우 관련성이 떨어질 수도 있다. 아이템의 상대적 만족도는 결과 목록에 표시된 다른 아이템의 가격에 대한 가격의 관계에 따라 달라질 수 있다. 흔한 LTR 알고리즘은 아이템 간의 dependency를 loss level에서 모델링을 시도한다. 즉 목록 내 아이템 들은 각자 점수가 매겨 진다. 그러나, 평가 metric에 대한 interaction의 영향은 일반적으로 pairwise(RankNet, LambdaLoss)나 listwise(ListNet, ListMLE) 형태의 Loss 함수로 설명이 가능하다. 예를 들어, LambdaMART에서 pairwise loss의 gradient는 한 쌍의 아이템이 swap 된 경우 발생 가능한 NDCG의 변화로 재조정 된다.

본 논문에서는 scoring 함수에 기반한 learnable, context-aware, self-attention을 제시한다. 이는 loss level 뿐만이 아니라 아이템의 점수 계산에서 도 아이템 간의 dependency의 모델링을 허용한다. self-attention 연산은 permutation-equivalent(입력 순서에 관계 없이 동일한 방식으로 항목에 점수 매기기) 때문에, 우리는 랭킹에 적합한 permutation-equivariant scoring 함수를 얻었다. positional-encoding을 사용해서 모델을 추가로 정제하면 모델은 re-ranking setting에 적합하게 된다.

평가는 Multi-Layer Perceptron 베이스 라인에 대해 point-wise, pair-wise 그리고 list-wise 랭킹 loss로 테스트를 했다. Allegrro.pl 이라는 큰 E-Commerce 검색 엔진의 MSLR-WEB30K 라는 LTR 데이터 셋을 사용했다. 그리고 새로운 SOTA를 NDCG@5에서 얻었다.

Gitlab에 Torch 코드가 공개 되어 있다.

- section 2에서는 관련된 논문의 리뷰
- section 3에서는 본 논문에서 해결하는 문제를 공식으로 해석
- section 4에서는 우리의 self-attentive ranking 모델을 설명
- section 5에서는 실험 결과와 관련해서 논의할 점
- section 6에서는 우리 모델의 다양한 hyperparameter에 대한 ablathon study를 수행
- section 7에서 최종적인 요약

2. RELATED WORK

Learning to rank는 광범위하게 연구 되어 왔고 그래서 클래식한 pointwise, pairwise 그리고 listwise 의 접근에 관련 해서 많은 자료가 있다. Learning to Rank for

Information Retrieval ('09, Tie-Yan et al)의 논문을 보면 가장 유명한 메소드 들에 대해 참고할 수 있다.

대부분의 LTR 방식의 공통점은 scoring 함수가 아이템 별로 점수를 매긴다는 점이다. 아이템 간 의존성은 Loss 수준에서만 고려된다. 이전의 목록 내에서 다른 아이템 들을 context로 점수를 매기는 방식의 모델링은 포함한다 :

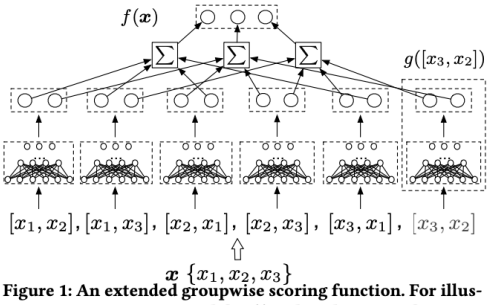
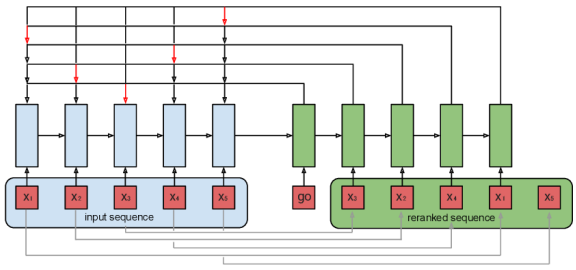


Figure 1: An extended groupwise scoring function. For illustrative purposes, we simplify $g(.)$ to be a bivariate function acting on permutations of size 2 formed from a list of 3 documents x. All 6 size-2 permutations from x are fed to $g(.)$ which itself outputs 2 scores per permutation. Intermediate scores computed by g are subsequently aggregated to compute the final vector of scores f.

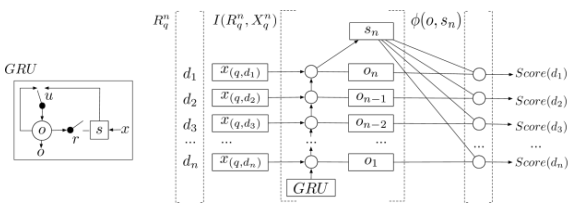
Learning Group-wise Multivariate Scoring Functions Using Deep Neural Networks('19, Qingyao et al)

- a pairwise scoring function 과 pairwise scoring function을 특수한 케이스로 합친 group-wise scoring(GSF)이 있다. 제안된 GSF 방법은 다중 아이템 들의 피처 벡터를 합치고 MLP로 보내는 것이었다. 합쳐진 아이템의 순서로 모델을 축소 시키기 위해서, Monte-Carlo Sampling이 사용하며, 이 Sampling은 확장이 불가능한 알고리즘을 내보낸다.



Seq2Slate : Re-Ranking and Slate Optimization with RNNs

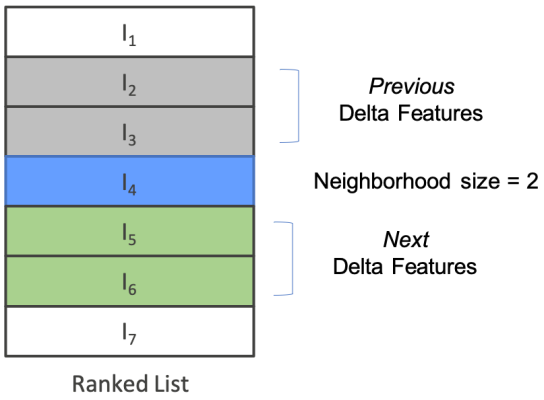
Seq2Slate 모델은 variant of Pointer Network와 결합된 RNN을 encoder-decoder 타입의 아키텍처를 사용해서, context-aware 방식으로 아이템 들을 인코딩하고, 아이템 들을 하나씩 선택해서 최적의 리스트를 생성한다. 저자는 이 방법을 클릭 데이터 만(실제, WEB30K)을 사용해서 평가했다.



- 비슷하게 더 간단한, Deep Listwise Context Model(DLCM)[RNN을 사용해서 아이템 셋을 re-ranking을 위해 encoding 한 방법 하고 하나의 decoding을 attention을 통해 적용

Deep List-wise Context Model (DLCM)

해서 re-ranking에 사용한다]도 있다.



Exploring the Effect of an Item's Neighborhood on its Sell-ability in eCommerce('19, Saratchandra et al)

- Exploring the Effect of an Item's Neighborhood on its Sell-ability in eCommerce에서는 delta 피처(주어진 목록에서 특정 아이템이 주변의 아이템 들과 어떻게 다른지를 표현하는)를 추가해서 아이템 간의 의존성을 고려하려고 했다. 이 방법은 단순한 형태의 local self-attention 메커니즘과 같다. 저자는 검색 로그 만을 사용해서 평가했다.

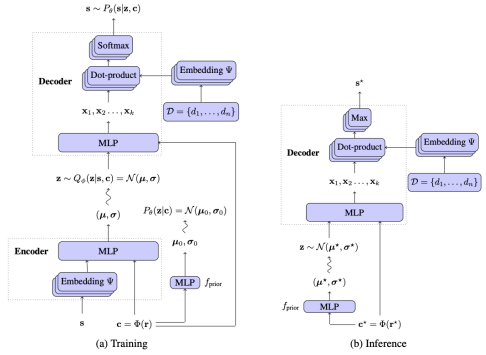


Figure 2: Structure of List-CVAE for both (a) training and (b) inference. $\mathbf{s} = (d_1, d_2, \ldots, d_k)$ is the input slate. $\mathbf{c} = \Phi(\mathbf{r})$ is the conditioning vector, where $\mathbf{r} = (r_1, r_2, \ldots, r_k)$ is the user responses on the slate $\mathbf{s}$. The concatenation of $\mathbf{s}$ and $\mathbf{c}$ makes the input vector to the encoder. The latent variable $\mathbf{z} \in \mathbb{R}^m$ has a learned prior distribution $\mathcal{N}(\mu_0, \sigma_0)$. The raw output from the decoder are k vectors $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_k$, each of which is mapped to a real document through taking the dot product with the matrix Φ containing all document embeddings. Thus produced k vectors of logits are then passed to the negatively downsampled k-head softmax operation. At inference time, $\mathbf{c}^*$ is the ideal condition whose concatenation with sampled $\mathbf{z}$ is the input to the decoder.

Beyond Greedy Ranking : Slate Optimization Via List-CVAE ('19, Ray et al)

- Beyond Greedy Ranking에서는 아이템 목록의 re-ranking 문제를 전체 아이템 생성 공식으로 만들었다. ListVAE를 소개한다. a variant of Conditional Variational Auto-Encoder는 사용자의 relevance 피드백에 따라 조정한 아이템 목록의 조합된 분포를 학습하고 랭킹 처리가 이미 된 아이템 목록을 생성한다. NDCG는 greedy ranking methods에 맞지 않다고 하고 평가에 사용하지는 않았다.

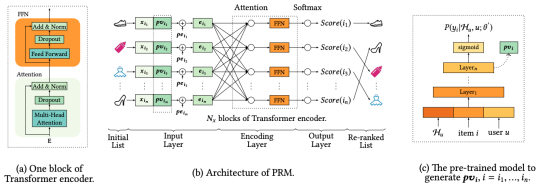


Figure 1: The detailed network structure of our PRM (Personalized Re-ranking Model) and its sub-modules.

- Personalized Re-Ranking for Recommendation에서는 이 논문의 접근법과 비슷하게, self-attention 메커니즘으로 아이템 간 의존성을 모델링한 경우도 있다. 그러나 표준 데이터 셋인 WEB30K로 평가하지는 않았

Personalized Re-ranking for Recommendation('19 Changhua et al)	고 ListNet 이라는 Loss 함수를 사용했다.

가장 처음으로 intra-attention을 LSTM에 적용했고 Transformer 구조가 도입 된 후에 더 많은 관심을 받았다. 본 논문의 모델은 Transformer의 encoder 파트의 특별한 경우로 보여진다.

차별화 된 loss function로 gradient 기반의 최적화 method를 사용해서 신경망 모델을 학습한다.

3. Problem Formulation

이 Section에서는 learning to rank 환경을 공식으로 만든다.

- X는 training set으로 (x, y)의 쌍이다.
- x는 d_f 차원의 실수 벡터다. x_i와 관련성인 label인 y_i (multi-level 혹은 binary) 와 함께 리스트 들로
  - x의 길이는 다 다양해질 수 있다
- scoring 함수 f를 찾는 것이 목표로 test set에서 검색의 지표인 NDCG를 최대화 하는 것이다.
- scoring 함수 f는 training data에 대해 surrogate loss의 평균을 감소하는 방향으로 학습한다.

$$\mathcal{L}(f) = \frac{1}{|X|} \sum_{(x,y)\in X} l((x, y), f),$$

overfitting 부분을 컨트롤 하기 위해 (network에 dropout 을 적용하거나 L1 이나 L2 페널티를 loss에 부여하는 등으로), learning to rank 알고리즘에서 중요한 두가지 선택지는 scoring function f와 loss function l 다. 보통, f 는 $x_i \in x$ 의 리스트를 대상으로 개별 적으로 점수를 매기기 $f(x_i)$로 하고, loss는 ground truth label y_i를 사용해서 측정한다. subsequent 섹션에서, context-aware scroing function f가 리스트 x 내의 아이템 들 x_i 들 간의 상호 작용을 모델링 할 수 있다는 것을 보여준다. 모델이 이미 일반적으로 표준화 된 point wise, pair wise, list wise loss에 충분히 잘 적용이 가능해서 실험에서 함께 사용한다.

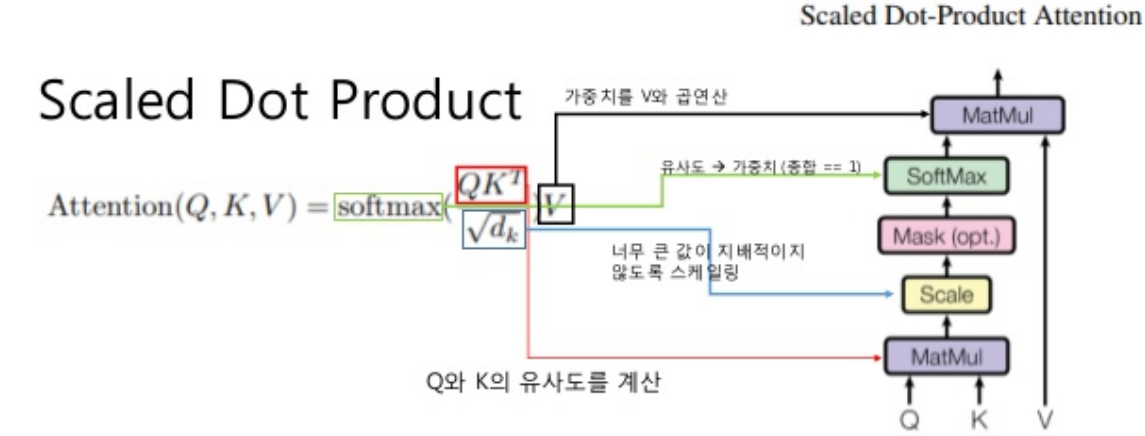
4. SELF-ATTENTIVE RANKER

이 section에서는 self-attention 기반의 ranking 모델의 아키텍처를 설명한다. transformer 아키텍처가 ranking setting에서 동작하도록 변경 했고, scoring 함수를 소개하는데 single 아이템의 스코어를 매길 때, 같은 list에 있는 다른 모든 아이템을 함께 고려해서 scoring을 한다.

4.1 Self-Attention Mechanism

attention 메커니즘은 Query Vector와 Key Vector 그리고 Value Vector를 입력으로 Output Vector를 출력하는 것이었다. 주어진 query에 대한 attention 메커니즘의 output은 Value Vector들의 가중 합계 다. weight 들은 해당하는 Value Vector의 Key가 Query와 얼마나 관계가 있는지가 수치화된 값이다. Self-attention은 Query, Key, Value가 모두 같은 것을 사용해서 Attention 되는 부분을 구한다. 논문의 경우에서는 list 내의 아이템 들의 벡터를 Query, Key, Value로 사용한다. Self-Attention 메커니즘의 목표는 list 내의 각 아이템 들의 새롭고, 고차원의 representation을 계산한다. list 내의 모든 아이템에 대해 가중치를 적용해 해당 아이템의 Query 아이템 과의 관련성을 계산에 사용한다.

Key 벡터 들과 Query 벡터 들의 관련성을 계산하는 많은 방법 들이 있다. 우리는 Scaled Dot-Product Attention 을 사용한다.



정리 : 이번 상태의 key와 value 페어인 {K, V}가 이전 상태인 Q와 어떤 연관이 있을것이다.
그러므로 K와 Q의 유사도를 계산하고 그걸 V에 반영
결국 V에 더 많은 Q의 정보가 전달된다.

- Scale 과 normalize의 차이
Scale – mean 0, Var 1으로 조정, 주로 overflow & underflow를 방지
Normalize – 전체 구간을 설정, 데이터 군 내에서 특정 데이터의 위치를 확인
  - 수식 : (요소값 – 최소값) / (최대값 – 최소값)

Q는 d_{model} 차원의 list 내의 모든 아이템(queries)을 표현한 것이라고 가정한다. K와 V를 매트릭스 들의 key와 value들로 둔다.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_{model}}})V$$

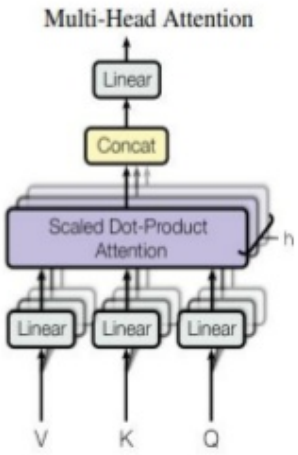
scaling factor $\frac{1}{\sqrt{d_{model}}}$가 d_{model}의 큰 값을 위한 softmax 연산에서 작은 Gradient를 방지하기 위해 추가됨

4.2 Multi-Headed Self-Attention

Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

• 지금까지 얘기했던 V, K, Q가 h개의 Scaled Dot-Product 를 통과하면서 h번의 선형 projection 뒤 결과들을 concat 해서 사용하면 더 좋다는 사실이 발견되었다.
Projection으로 인한 dimension decrease 때문에 계산량이 큰 폭으로 증가하지 않는다.



Self-Attention 메커니즘을 여러번 계산하고 출력 값을 합치는 방법은 좋다. 결과인 출력 벡터의 사이즈가 커지는 것을 방지하기 위해서, Q,K 그리고 V 매트릭스 들을 d_q, d_k, d_v 차원의 공간으로 선형적으로 투영하는 짓을 H (head) 만큼 진행한다. 일반적으로, $d_q = d_k = d_v = d_{model}$ /H 인 형태가 된다. Q, K, V를 linear projection 을 H 번 진행하는 것은 single attention 의 head 라고 한다. 각 head는 고유의 학습 가능한 projection matrices를 가진다. 각 head의 출력 들은 합쳐져서 다시 한번 선형적으로 project 되어서, 일반적으로는 입력 매트릭스 Q와 동일한 차원의 벡터 공간에 투영된다. Transformer와 유사하게, 본 논문의 모델도 multiple attention heads를 사용한다. 그래서

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_H)W^O$$
where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

그리고 projections 은 주어진 매트릭스 들로 된다.

$W_i^Q \in R^{d_{model} \times d_q}, W_i^K \in R^{d_{model} \times d_k},$
$W_i^V \in R^{d_{model} \times d_v}, W^O \in R^{H d_v \times d_{model}}.$

4.3 Permutation equivariance 순서에 관계 없이 같은 방식으로 아이템에 연산

원래 입력 순서에 의존하지 않고 아이템들에 점수를 매긴다. 실험적인 관측을 통해 permutation-equivariant 하다고 한다.

4.4 Positional Encodings

Transformer 구조는 원래 NMT 태스크를 해결 하기 위해 설계된 구조이다. NMT에서는 입력 토큰 들의 순서를 고려 해야 한다. RNN 와는 다르게, self-attention 기반 인코더는 입력 토큰 순서를 반영할 방법이 없다. permutation-equivariance 하기 때문이다. 그래서 Transformer의 저자는 고정적이고 학습 가능한 positional encodings을 input embeddings에 추가 했다. 고정적인 positiontional encoding 들은 다음 같은 frequencies를 보이는 sine과 cosine을 사용한다.

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}})$$

pos는 위치이고 i 는 차원이다.

랭킹 문제는 (case1)정렬 되어 있지 않은 아이템 들을 정렬하는 것 그리고 re-ranking(case2) 하는 것으로 볼 수 있다. 이미 입력 list가 상대적으로 weak 한 ranking model에 의해서 정렬이 되어져 있다. 전자의 경우에는 positional encoding을 사용하는 의미가 없다. 후자의 경우에는 모델의 성능을 증가 시킬 수 있다.

4.5 Model Architecture

Transformer 모델을 사용해서 ranking setting으로 사용했다. list 의 아이템 들은 토큰들로 다루어지고 아이템 feature들은 item token embedding으로 다루어 진다. 입력 list의 길이는 l 그리고 feature의 수를 d_f로 둔다. 각 아이템들은 d_{fc}의 크기를 가지는 fully connected layer로 보내진다. 다음에는 hideen representations 들이 N 개의 encoder block, H 헤드들 그리고 hidden dimension d_h들을 가지는 Transformer의 아키텍처 상 encoder 파트로 보내진다. 회상 해보면 Transformer 내의 encoder 블록은 입력에 skip-connection 과 multi-head attention layer로 layer normalization으로 구성된다. dropout은 summation 전에 residual 블록에서 사용이 된다. N개 인코더 블록을 지난 후에, 리스트 내의 모든 아이템들에 fully-connected layer를 거쳐서 각 아이템들의 점수가 계산된다. Transformer 의 encoder의 부분을 사용하는 것을 사용하는 것으로 볼 수 있다. 그래소 모델은 다음과 같이 표현된다.

$$f(x) = FC(Encoder(Encoder(...(Encoderr(FC(x))))))$$

N 번의 Encoder 블록을 거친다.

where

$$Encoder(x) = LayerNorm(z + Dropout(FC(z))),$$
$$z = LayerNorm(x + Dropout(MultiHead(x)))$$

Encoder 에서 self-attention을 사용함으로, 주어진 아이템에 대해서 점수를 계산할 때, 다른 모든 아이템들의 hidden representation을 사용하도록 한다. 획득된 점수는, ground truth label들과 함께 어떤 ranking loss 의 선택도 가능하게 한다. 만약 loss가 다른 기능을 하는 scoring function이라면, SGD를 사용해서 최적화 할 수 있다. 그래서 본 논문에서는 list에 대해 다른 ranking loss를 사용할 수 있는 일반화된, context-aware 모델을 가질 수 있다.

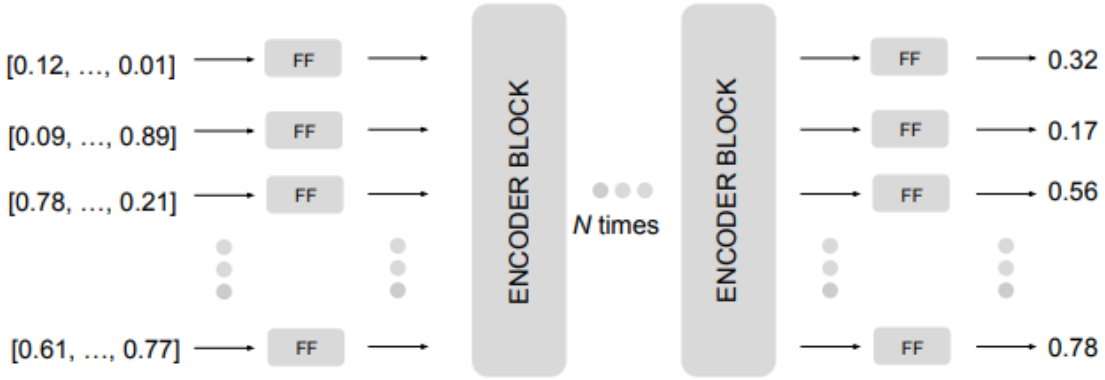


Figure 1: Schematic of the proposed model architecture. Input is a list of real-valued vectors. Output is the list of real-valued scores.

5. EXPERIMENTS

5.1 Dataset

Learning to rank 데이터셋 들은 두가지 맛이 있다. 1.Multi-level 과 2.Binary relevance label 이 있다. 보통 multi-level relevance 라벨은 사람이 만들고, binary 는 CTR 로그 기반으로 한다. 두 가지를 다 실험에 사용

Mutli-Level은

WEB30K 데이터셋을 5-fold cross-validation 으로 1 validation 1 test로 사용했다.

NDCG를 평가 지표로 LightGBM 과 비교

4. nDCG(normalizing Discounted Cumulative Gain)

MRR(Mean), MAP(Mean Average Precision), Top K Recall 등의 방법은 한 명의 사용자가 가진 relevant 아이템이 많을 수록 평가하기 수월하다. 하지만 impression에 대한 데이터가 없거나, 한 명의 사용자가 반응한 랭킹이 상위에만 지나치게 집중되어 있다면 이러한 방법들은 한계를 드러낸다. 그리고 실제로 대부분의 검색, 추천 랭킹 시스템은 극단적으로 skewed한 인기도를 가지고 있다. 그래서 nDCG는 순위에 가중치를 주기 위한 방법을 적용한다. 또한 하나의 랭킹으로 여러 명을 평가하기 위해 밀도(혹은 점수)를 추가하여, 2차원 점수를 평가한다는 의미를 가지기도 한다.

nDCG는 현재 점수(밀도)의 랭킹에서 가장 이상적인 랭킹과 현재 랭킹을 cummulative한 점수로 비교한다. 그리고 랭킹이 낮아짐에 따라 중요도가 감소하는 것은 log normalization으로 완화한다. nDCG를 계산하기 위한 수식들은 다음과 같다.

만약 4,3,4,2,1의 score를 가지고 있는 5개 Q의 반환 결과가 있다면, 이에 대한 ideal vector를 4,4,3,2,1 이라고 한다. 그리고 이 두 개의 relevant score list로 nDCG 스코어를 생성하는 것. 당연하게도 1에 가까울수록 좋은 랭킹이다. 더 직관적인 이해를 돕기 위해 파이썬 코드로 이를 표현하였다.

Binary Relevance Label은

Allegro.pl 의 검색 엔진은 이미 ranking model을 사용하고 있다. XGBoost 기반의 pairwise ranking 모델이다. 1M 데이터 사이즈, 가장 긴 시퀀스가 60길이로 관련 있는 건 1로 나머지 모든 아이템은 0으로, 아이템 임베딩 45 사이즈, cross validation 사용하지 않음.

5.2 Loss Function

5.2.1 Pointwise RMSE

가장 간단한 pointwise로 아이템 간의 상호 작용을 고려하지 않는다.

$$l(\boldsymbol{s}, \boldsymbol{y}) = \sqrt{\sum_i (y_i - s_i)^2}$$

마지막 activation으로 sigmoid를 사용했고 maximum relevance value(4) 를 곱해서 rescale한 결과

5.2.2 Ordinal Loss

Multi-level Ground truth를 사용했다. 다음과 같이 벡터로 변환해서 진행함.

$$
\begin{aligned}
0 &\mapsto [0, 0, 0, 0], \\
1 &\mapsto [1, 0, 0, 0], \\
2 &\mapsto [1, 1, 0, 0], \\
3 &\mapsto [1, 1, 1, 0], \\
4 &\mapsto [1, 1, 1, 1].
\end{aligned}
$$

activation으로 sigmoid로 사용해서 모든 아이템에 대해서 결과를 내보낸다. 마지막 loss value는 mean of binary cross entropy 를 사용한다. 랭킹 setting에서 클래식한 ordinal loss

5.2.3 LambdaLoss, RankNet and LambdaRank, NDCG-Loss2++

$$
l(\boldsymbol{s}, \boldsymbol{y}) = -\sum_{y_i > y_j} \log_2 \sum_{\pi} \left(\frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right)^{(\rho_{ij} + \mu\delta_{ij})|G_i - G_j|} H(\pi|\boldsymbol{s})
$$

where

$$
G_i = \frac{2^{y_i} - 1}{\text{maxDCG}},
$$

$$
\rho_{ij} = \left| \frac{1}{D_i} - \frac{1}{D_j} \right|,
$$

$$
\delta_{ij} = \left| \frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j|} + 1} \right|,
$$

$$
D_i = log_2(1 + i)
$$

and $H(\pi|\boldsymbol{s})$ is a hard assignment distribution of permutations, i.e.

$$
H(\hat{\pi}|\boldsymbol{s}) = 1 \text{ and } H(\pi|\boldsymbol{s}) = 0 \text{ for all } \pi \neq \hat{\pi}
$$

$\hat{\pi}$는 모든 아이템이 score s를 감소 시키면서 가지게 되는 정렬되는 아이템 순서이다.

. Fixed parameter μ is set to 10.0.

l(s, y) 공식에서 지수를 제거 해서 RankNet Loss 함수를 얻을 수 있고, 각 점수 pair를 동일한 가중치로 맞춘다. 비슷하게도, 지수 내의 공식을 변경해서 다르게 weighted 된 변형된 RankNet을 얻을 수 있다.

LambdaRank 공식은 지수를 변경해서

$$\Delta\text{NDCG}(i,j) = |G_i - G_j|\rho_{ij}.$$

를 가질 수 있다.

5.2.4 ListNet and ListMLE. ListNet loss

$$l(\boldsymbol{s},\boldsymbol{y}) = -\sum_j \text{softmax}(\boldsymbol{y})_j \times \log(\text{softmax}(\boldsymbol{s})_j)$$

binary 버전에서는, 소프트 맥스 ground truth y는 single-click lists에서 도출되고 multiple-click list에서는 클릭 수의 normalization으로 교체한다.

ListMLE [35] is given by:

$$l(\boldsymbol{s},\boldsymbol{y}) = -\log P(\boldsymbol{y}|\boldsymbol{s})$$

where

$$P(\boldsymbol{y}|\boldsymbol{s}) = \prod_i^n \frac{\exp(f(x_{y(i)}))}{\sum_{k=i}^n \exp(f(x_{y(k)}))}$$

and $y(i)$ is the index of object which is ranked at position i.

5.3 Experimental setup

validation set으로 hyper-parameter optimization을 진행함

Adam optimizer를 사용해서 learning rate는 고려하지 않았다.

Positional Encoding을 썼을 때 때 눈에 띄는 성능 개선은N 없었다.

Table 5: Ablation study

Parameter	Value	Params	WEB30K NDCG@5
baseline		950K	49.01
H	1	950K	51.99
	4	950K	51.96
N	1	250K	50.33
	2	490K	51.72
d_h	64	430K	50.92
	128	509K	51.31
	256	650K	51.81
	1024	1540K	**52.11**
p_{drop}	0.0	950K	42.18
	0.1	950K	50.36
	0.2	950K	51.84
	0.3	950K	51.99
	0.5	950K	51.32
l	30	950K	50.48
	60	950K	51.14
	120	950K	51.72
	360	950K	51.89