Seq2Slate: Re-ranking and Slate Optimization with RNNs

Irwan Bello [1] **Sayali Kulkarni** [1] **Sagar Jain** [1] **Craig Boutilier** [1] **Ed Chi** [1] **Elad Eban** [1] **Xiyang Luo** [1] **Alan Mackey** [1] **Ofer Meshi** [1]

Abstract

Ranking is a central task in machine learning and information retrieval. In this task, it is especially important to present the user with a slate of items that is appealing as a whole. This in turn requires taking into account interactions between items, since intuitively, placing an item on the slate affects the decision of which other items should be placed alongside it. In this work, we propose a sequence-to-sequence model for ranking called *seq2slate*. At each step, the model predicts the next "best" item to place on the slate given the items already selected. The sequential nature of the model allows complex dependencies between the items to be captured directly in a flexible and scalable way. We show how to learn the model end-to-end from weak supervision in the form of easily obtained click-through data. We further demonstrate the usefulness of our approach in experiments on standard ranking benchmarks as well as in a real-world recommendation system.

1. Introduction

Ranking a set of candidate items is a central task in machine learning and information retrieval. Many existing ranking systems are based on pointwise estimators, where the model assigns a score to each item in a candidate set and the resulting *slate* is obtained by sorting the list according to item scores (Liu et al., 2009). Such models are usually trained from click-through data to optimize an appropriate loss function (Joachims, 2002). This simple approach is computationally attractive as it only requires a sort operation over the candidate set at test (or serving) time, and can therefore scale to large problems. On the other hand, in terms of modeling, pointwise rankers cannot easily express dependencies between ranked items. In particular, the score of an item (e.g., its probability of being clicked) often depends on the other items in the slate and their joint placement. For example, in the common case where only a few highly ranked items get the user's attention, it may be better to present a *diverse* set of items at the top positions of the slate in order to cover a wider range of user interests.

A significant amount of work on learning-to-rank does consider interactions between ranked items when *training* the model. In *pairwise* approaches a classifier is trained to determine which item should be ranked first within a pair of items (e.g., Herbrich et al., 1999; Joachims, 2002; Burges et al., 2005). Similarly, in *listwise* approaches the loss depends on the full permutation of items (e.g., Cao et al., 2007; Yue et al., 2007). Although these losses consider inter-item dependencies, the ranking function itself is pointwise, so at inference time the model still assigns a score to each item which does not depend directly on the other items (i.e., an item's score will not change if it is placed in a different set).

There has been some work on trying to capture interactions between items in the ranking scores themselves (e.g., Qin et al., 2008; 2009; Zhu et al., 2014; Rosenfeld et al., 2014; Dokania et al., 2014; Borodin et al., 2017; Ai et al., 2018b). Such approaches can, for example, encourage a pair of items to appear next to (or far from) each other in the resulting ranking. Approaches of this type often assume that the relationship between items takes a simple form (e.g., submodular (Borodin et al., 2017)) in order to obtain tractable inference and learning algorithms. Unfortunately, this comes at the expense of the model's expressive power. Alternatively, greedy or approximate procedures can be used at inference time, though this often introduces approximation errors, and many of these procedures are still computationally expensive (e.g., Rosenfeld et al., 2014).

More recently, neural architectures have been used to extract representations of the entire set of candidate items for ranking, thereby taking into consideration all candidates when assigning a score for each item (Mottini & Acuna-Agost, 2017; Ai et al., 2018a). This is done by an encoder which processes all candidate items sequentially and produces a compact representation, followed by a scoring step in which pointwise scores are assigned based on this joint representation. This approach can in principle model rich

[1]Google, Mountain View, California, USA.
Correspondence to: Ofer Meshi <meshi@google.com>.

Proceedings of the Workshop on Negative Dependence in Machine Learning at the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

dependencies between ranked items, however its modeling requirements are quite strong. In particular, all the information about interactions between items needs to be stored in the intermediate compact representation and extracted in one-shot when scoring (decoding).

Instead, in this paper we propose a different approach by applying *sequential decoding*, which assigns item scores conditioned on previously chosen items. Our decoding procedure lets the score of an item change depending on the items already placed in previous positions. This allows the model to account for high-order interactions in a natural and scalable manner. Moreover, our approach is purely data-driven so the model can adapt to various types of inter-item dependencies, for example, negative dependence – where items decrease each other's appeal. We apply a *sequence-to-sequence (seq2seq)* model (Sutskever et al., 2014) to the ranking task, where the input is the list of candidate items and the output is the resulting ordering. Since the output sequence corresponds to ranked items on the slate, we call this approach *sequence-to-slate*, or in short *seq2slate*.

To address the seq2seq problem, we build on the recent success of *recurrent neural networks (RNNs)* (e.g., Sutskever et al., 2014). This allows us to use a deep model to capture rich dependencies between ranked items, while keeping the computational cost of inference manageable. More specifically, we use *pointer networks*, which are seq2seq models with an attention mechanism for pointing at positions in the input (Vinyals et al., 2015). We show how to train the network end-to-end from click-through data to optimize several commonly used ranking measures. Finally, we demonstrate the usefulness of the proposed approach in experiments on benchmark and real-world data.

2. Ranking as Sequence Prediction

The *ranking problem* is that of computing a ranking of a set of items (or ordered list or *slate*) given some query or context. We formalize the problem as follows. Assume a set of n items, each represented by a feature vector $x_i \in \mathbb{R}^m$ (which may depend on a query or context).[1] Let $\pi \in \Pi$ denote a permutation of the items, where each $\pi_j \in \{1, \ldots, n\}$ denotes the index of the item in position j, for example, $\pi = (3, 1, 2, 4)$ for $n = 4$. Our goal is to predict an "optimal" output ranking π given the input items x.

In the seq2seq framework, the probability of an output permutation, or slate, given the inputs is expressed as a product of conditional probabilities according to the chain rule:

$$p(\pi|x) = \prod_{j=1}^{n} p(\pi_j|\pi_1, \ldots, \pi_{j-1}, x) \,, \tag{1}$$

[1] x_i can represent either raw inputs or learned embeddings.

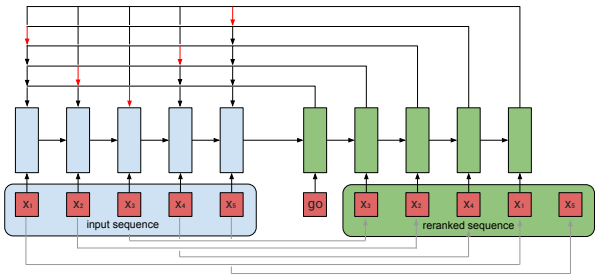

Figure 1. The seq2slate pointer network architecture for ranking.

This expression is completely general and does not make any conditional independence assumptions. In our case, the conditional $p(\pi_j|\pi_{<j}, x) \in \Delta^n$ (a point in the n-dimensional simplex) models the probability of any item being placed at the j'th position in the ranking given the items already placed at previous positions. For brevity, we have denoted the prefix permutation $\pi_{<j} = (\pi_1, \ldots, \pi_{j-1})$. Therefore, this conditional can exactly capture *all* high-order dependencies between items in the ranked list, including those due to diversity, similarity or other interactions.

Pointer-Network Architecture for Ranking

We employ the *pointer-network architecture* of Vinyals et al. (2015) to model the conditional $p(\pi_j|\pi_{<j}, x)$. A pointer network uses non-parametric softmax modules, akin to the attention mechanism of Bahdanau et al. (2015), and learns to point to items in its input sequence rather than predicting an index from a fixed-sized vocabulary.

Our *seq2slate* model, illustrated in Fig. 1, consists of two *recurrent neural networks* (RNNs): an encoder and a decoder, both of which use Long Short-Term Memory (LSTM) cells (Hochreiter & Schmidhuber, 1997). At each encoding step $i \leq n$, the encoder RNN reads the input vector x_i and outputs a ρ-dimensional vector e_i, thus transforming the input sequence $\{x_i\}_{i=1}^n$ into a sequence of latent memory states $\{e_i\}_{i=1}^n$. These latent states can be seen as a compact representation of the entire set of candidate items. At each decoding step j, the decoder RNN outputs a ρ-dimensional vector d_j which is used as a query in the attention function. The attention function takes as input the query $d_j \in \mathbb{R}^\rho$ and the set of latent memory states computed by the encoder $\{e_i\}_{i=1}^n$ and produces a probability distribution over the next item to include in the output sequence as follows:

$$s_i^j = v^\top \tanh\left(W_{enc} \cdot e_i + W_{dec} \cdot d_j\right) \tag{2}$$

$$p_\theta(\pi_j = i|\pi_{<j}, x) \equiv p_i^j = \begin{cases} e^{s_i^j} / \sum_{k \notin \pi_{<j}} e^{s_k^j} & \text{if } i \notin \pi_{<j} \\ 0 & \text{if } i \in \pi_{<j} \end{cases} .$$

Here $W_{enc}, W_{dec} \in \mathbb{R}^{\rho \times \rho}$ and $v \in \mathbb{R}^\rho$ are learned parameters in our network, denoted collectively by parameter vector θ, and s_i^j are *scores* associated with placing item i in position j. The probability $p_i^j = p_\theta(\pi_j = i|\pi_{<j}, x)$, is obtained via a softmax over the remaining items and represents the

degree to which the model points to input i at decoding step j. In order to output a permutation, the probabilities p_i^j are set to 0 for items i that already appear on the slate. Once the next item π_j is selected, typically greedily or by sampling (see below), its embedding x_{π_j} is fed as input to the next decoder step. This way the decoder states hold information on the items already placed on the slate. The input to the first decoder step is a learned m-dimensional vector, denoted as '*go*' in Fig. 1.

Previous studies have shown that the order in which the input is processed can significantly affect the performance of sequential models (Vinyals et al., 2016; Nam et al., 2017; Ai et al., 2018a). For this reason, we will assume here the availability of a base (or "production") ranker with which the input sequence is ordered (e.g., a simple pointwise method that ignores the interactions we seek to model), and view the output of our model as a *re-ranking* of the items.

3. Training with Click-Through Data

We now turn to the task of training the seq2slate model from data. A typical approach to learning in ranking systems is to run an existing ranker "in the wild" and log click-through data, which are then used to train an improved ranking model. This type of training data is relatively inexpensive to obtain, in contrast to human-curated labels such as relevance scores, ratings, or full rankings (Joachims, 2002).

Formally, each training example consists of a sequence of items $x = \{x_1, \ldots, x_n\}$, with $x_i \in \mathbb{R}^m$ and binary labels $y = (y_1, \ldots, y_n)$, with $y_i \in \{0, 1\}$, representing user feedback (e.g., click/no-click). Our goal is to learn the parameters θ of $p_\theta(\pi_j|\pi_{<j}, x)$ (Eq. (2)) such that permutations π corresponding to "good" rankings are assigned high probabilities. Various performance measures $\mathcal{R}(\pi, y)$ can be used to evaluate the quality of a permutation π given the labels y. Generally speaking, permutations where the positive labels rank higher are considered better.

In the standard seq2seq setting, models are trained to maximize the likelihood of a target sequence of tokens given the input, which can be done by maximizing the likelihood of each target token given the previous target tokens using Eq. (1). Unfortunately, this approach cannot be applied in our setting since it requires ground-truth permutations while we only have access to weak supervision in the form of labels y (e.g., clicks).

3.1. Training Using REINFORCE

One viable approach, which has been applied successfully in related tasks (Bello et al., 2017; Zhong et al., 2017), is to use *reinforcement learning (RL)* to directly optimize for the ranking measure $\mathcal{R}(\pi, y)$. In this setup, the objective is to maximize the expected ranking metric obtained by sequences sampled from our model: $\max_\theta \ \mathbb{E}_{\pi\sim p_\theta(.|x)}[\mathcal{R}(\pi, y)]$. Policy gradients and stochastic gradient ascent can be used to optimize θ. The gradient is formulated using the REINFORCE update (Williams, 1992):

$$\nabla_\theta \mathbb{E}_{\pi\sim p_\theta(.|x)}[\mathcal{R}(\pi, y)] = \mathbb{E}_{\pi\sim p_\theta(.|x)}\Big[\mathcal{R}(\pi, y)\nabla_\theta \log p_\theta(\pi \mid x)\Big] ,$$

which can be approximated via Monte-Carlo sampling.

3.2. Supervised Training

Policy gradient methods like REINFORCE are known to induce challenging optimization problems and can suffer from sample inefficiency and difficult credit assignment. As an alternative, we propose *supervised learning* using the labels y. In particular, rather than waiting until the end of the output sequence as in RL, we can give feedback to the model at each decoder step.

Consider the first step, and recall that the model assigns a score s_i to each item in the input (see Eq. (2)); to simplify notation we omit the position superscript j for now. Letting $s = (s_1, \ldots, s_n)$, we define a per-step loss $\ell(s, y)$ which essentially acts as a multi-label classification loss with labels y as ground truth. Any ranking loss can be used for ℓ, here we employ two simple choices, cross-entropy and hinge:

$$\ell_{xent}(s, y) = -\sum_i \hat{y}_i \log p_i \tag{3}$$

$$\ell_{hinge}(s, y) = \max\{0, 1 - \min_{i:y_i=1} s_i + \max_{j:y_j=0} s_j\} ,$$

where $\hat{y}_i = y_i / \sum_j y_j$, and p_i is a softmax of s, as in Eq. (2).

We define the *sequence loss* for a *fixed* permutation π as:

$$\mathcal{L}_\pi(S, y) = \sum_{j=1}^{n} \ell_{\pi_{<j}}(s^j, y) , \tag{4}$$

where $S = \{s^j\}_{j=1}^n$ are the model scores (see Eq. (2)), and each $s^j = (s_1^j, \ldots, s_n^j)$ is the item-score vector for position j. In the sequel we will also use the abbreviation: $\mathcal{L}_\pi(\theta) \equiv \mathcal{L}_\pi(S(\theta), y)$. Importantly, the per-step loss $\ell_{\pi_{<j}}(s^j, y)$ depends only on the indices in s^j and y which are not in the prefix $\pi_{<j}$ (cf. Eq. (3)).

Using the definition of the sequence loss above, our goal is to optimize the expected loss: $\min_\theta \ \mathbb{E}_{\pi\sim p_\theta(\cdot|x)}[\mathcal{L}_\pi(\theta)]$, which corresponds to sampling the permutation π according to the model. Notice that this expected loss is differentiable everywhere since both $p_\theta(\pi|x)$ and $\mathcal{L}_\pi(\theta)$ are differentiable for any permutation π. In this case, the gradient is formulated as (see also Schulman et al., 2015, Eq. (4)):

$$\nabla_\theta \mathbb{E}_\pi[\mathcal{L}_\pi(\theta)] = \mathbb{E}_{\pi\sim p_\theta(\cdot|x)}\left[\mathcal{L}_\pi(\theta) \cdot \nabla_\theta \log p_\theta(\pi|x) + \nabla_\theta \mathcal{L}_\pi(\theta)\right] ,$$

which again can be approximated from samples.

Ranker	Yahoo			Web30k		
	MAP	NDCG@5	NDCG10	MAP	NDCG@5	NDCG@10
seq2slate	**0.67**	**0.69**	**0.75**	**0.51**	**0.53**	**0.59**
AdaRank	0.58	0.61	0.69	0.37	0.38	0.46
Coordinate Ascent	0.49	0.51	0.59	0.31	0.33	0.39
LambdaMART	0.58	0.61	0.69	0.42	0.46	0.52
ListNet	0.49	0.51	0.59	0.43	0.47	0.53
MART	0.58	0.60	0.68	0.39	0.42	0.48
Random Forests	0.54	0.57	0.65	0.36	0.39	0.45
RankBoost	0.50	0.52	0.60	0.24	0.25	0.30
RankNet	0.54	0.57	0.64	0.43	0.47	0.53

Table 1. Performance of seq2slate and other baselines on data generated with diverse-clicks.

4. Experimental Results

We evaluate the performance of our seq2slate model on a collection of ranking tasks. In Section 4.1 we use learning-to-rank benchmark data to study the behavior of the model. We then apply our approach to a large-scale commercial recommendation system and report the results in Section 4.2.

4.1. Learning-to-Rank Benchmarks

We conduct experiments using two learning-to-rank datasets, the Yahoo Learning to Rank Challenge data (set 1), and the Microsoft Web30k dataset.We adapt the procedure proposed by Joachims et al. (2017) to generate click data. A base ranker is first trained from the raw data. The base ranking is then used to generate training data by simulating a user "cascading" through the results and clicking on items (for full details see Joachims et al. (2017)). In order to introduce high-order dependencies, we augment the procedure in Joachims et al. (2017) by generating clicks only if an item is not too similar to previously clicked items (i.e, diverse enough). Similarity is defined as being in the smallest q percentile (i.e., $q = 0.5$ is the median) of Euclidean distances between pairs of feature vectors within the same ranking instance: $D_{ij} = \|x_i - x_j\|$.

Using the generated training data, we train both our seq2slate model and baseline rankers from the RankLib package. The results in Table 1 show that seq2slate significantly outperforms all the baselines, suggesting that it can better capture and exploit the dependencies between items in the data.

4.2. Real-World Data

We also apply seq2slate to a ranking problem from a large-scale commercial recommendation system. We train the model using massive click-through logs (comprising roughly $O(10^7)$ instances). The data has item sets of varying size, with an average n of 10.24 items per example. We learn embeddings of the raw inputs as part of training.

Table 2 shows the performance of seq2slate compared to the production base ranker on test data (of roughly the same size as the training data). We also compare to a computationally cheaper one-step decoder, which outputs a single vector $p^1 = p_\theta(\pi_1 = \cdot|x)$ (see Eq. (2)), from which π is obtained by sorting—similar to the approach taken in (Mottini & Acuna-Agost, 2017; Ai et al., 2018a)). Significant gains are observed in all performance metrics, with sequential decoding outperforming the one-step decoder. This suggests that sequential decoding may more faithfully capture complex dependencies between the items.

Ranker	MAP	NDCG@5	NDCG@10	rank-gain
one-step decoder	+26.79%	+10.69%	+40.67%	0.83
seq2slate	+31.32%	+14.47%	+45.77%	1.087

Table 2. Performance compared to a competitive base production ranker on real data.

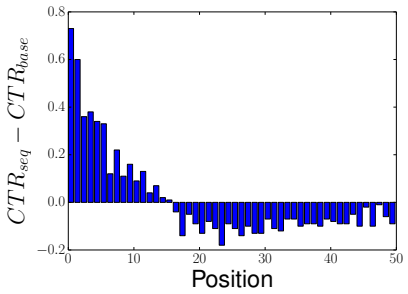

Figure 2. Difference in CTR per position between a seq2slate model and a base production ranker in a live experiment.

Finally, we let the learned seq2slate model run in a live experiment (A/B testing) and re-rank the result of the current production recommender system. We compute the click-through rate (CTR) in each position (#clicks/#examples) for seq2slate. The production base ranker serves traffic outside the experiment, and we compute CTR for this traffic as well. Fig. 2 shows the difference in CTR per position, indicating that seq2slate has significantly higher CTR in the top positions. This suggests that seq2slate indeed places items that are likely to be chosen higher in the ranking.

References

Ai, Q., Bi, K., Guo, J., and Croft, W. B. Learning a deep listwise context model for ranking refinement. In *SIGIR*, pp. 135–144, 2018a.

Ai, Q., Wang, X., Golbandi, N., Bendersky, M., and Najork, M. Learning groupwise scoring functions using deep neural networks. 2018b.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *ICLR 2017 – Workshop Track*, 2017.

Borodin, A., Jain, A., Lee, H. C., and Ye, Y. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 13(3):41:1–41:25, 2017.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, 2005.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136. ACM, 2007.

Dokania, P. K., Behl, A., Jawahar, C., and Kumar, M. P. Learning to rank using high-order information. In *European Conference on Computer Vision*, pp. 609–623. Springer, 2014.

Herbrich, R., Graepel, T., and Obermayer, K. Support vector learning for ordinal regression. In *ICANN*, pp. 97–102, 1999.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computations*, 1997.

Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142. ACM, 2002.

Joachims, T., Swaminathan, A., and Schnabel, T. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 781–789. ACM, 2017.

Liu, T.-Y. et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3): 225–331, 2009.

Mottini, A. and Acuna-Agost, R. Deep choice model using pointer networks for airline itinerary prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

Nam, J., Loza Mencía, E., Kim, H. J., and Fürnkranz, J. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in Neural Information Processing Systems 30*, pp. 5413–5423, 2017.

Qin, T., Liu, T.-Y., Zhang, X.-D., Wang, D.-S., Xiong, W.-Y., and Li, H. Learning to rank relational objects and its application to web search. In *Proceedings of WWW*, pp. 407–416. ACM, 2008.

Qin, T., Liu, T.-Y., Zhang, X.-D., Wang, D.-S., and Li, H. Global ranking using continuous conditional random fields. In *Advances in neural information processing systems*, pp. 1281–1288, 2009.

Rosenfeld, N., Meshi, O., Tarlow, D., and Globerson, A. Learning structured models with the auc loss and its generalizations. In *Artificial Intelligence and Statistics*, pp. 841–849, 2014.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems 28*, 2015.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *NIPS*, pp. 3104–3112, 2014.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NIPS*, pp. 2692–2700, 2015.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2016. URL http://arxiv.org/abs/1511.06391.

Williams, R. Simple statistical gradient following algorithms for connectionnist reinforcement learning. In *Machine Learning*, 1992.

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. A support vector method for optimizing average precision. In *Proceedings of SIGIR*, pp. 271–278. ACM, 2007.

Zhong, V., Xiong, C., and Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Zhu, Y., Lan, Y., Guo, J., Cheng, X., and Niu, S. Learning for search result diversification. In *Proceedings of SIGIR*, pp. 293–302. ACM, 2014.