Seq2Slate: Re-ranking and Slate Optimization with RNNs

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

Google

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. Such interactions between items can be especially dominant in the common case where display area is limited or when strong position bias is present, so that only a

*Corresponding authors: ibello@google.com, meshi@google.com

few highly ranked items get the user's attention. In this case it may be preferable, for example, to present a *diverse* set of items at the top positions of the slate in order to cover a wider range of user interests. Conversely, presenting multiple items with similar attributes may create "synergies" by drawing attention to the collection, amplifying user response beyond that of any individual item.

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 on scores of 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 and 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 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 in turn 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, including synergies—where items appearing together contribute to their joint appeal, and interference—where items decrease each other's appeal. In particular, 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*.

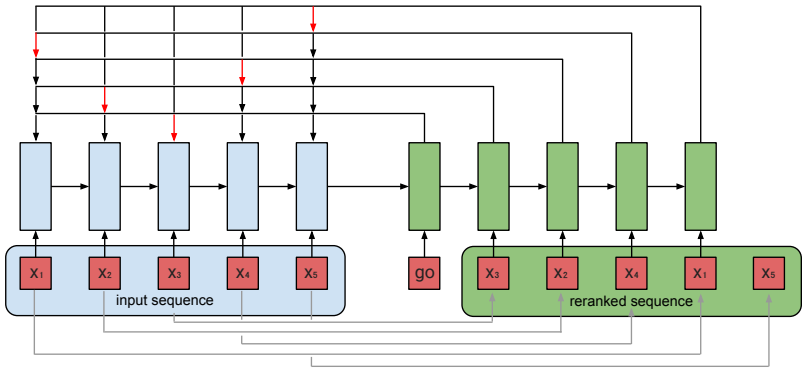

Figure 1: The seq2slate pointer network architecture for ranking.

To address the seq2seq problem, we build on the recent success of *recurrent neural networks (RNNs)* in a wide range of applications [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., 2015b]. We show how to train the network end-to-end to optimize several commonly used ranking measures. To this end, we adapt RNN training to use weak supervision in the form of click-through data obtained from logs, instead of relying on ground-truth rankings, which are much more expensive to obtain. Finally, we demonstrate the usefulness of the proposed approach in a number of learning-to-rank benchmarks and in a large-scale, real-world recommendation system.

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. For instance, given a specific user query, we might want to return an ordered set of music recommendations from a set of candidates that maximizes some measure of user engagement (e.g., number of tracks played).

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.

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.

Our setting is somewhat different than a standard seq2seq setting in that the output vocabulary is not fixed. In particular, unlike in e.g., machine translation, the same index (position) is populated by different items in different instances (queries). The vocabulary size n itself may also vary per instance in the common case where the number of items to rank can change. This is precisely the problem addressed by *pointer networks*, which we review next.

Pointer-Network Architecture for Ranking

We employ the *pointer-network architecture* of Vinyals et al. [2015b] 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 and 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.

We note the following. **(i)** Our formulation using sequential decoding lets the score of items (i.e., p_i^j) change depending on items previously placed on the slate, thereby allowing the model to account for high-order interactions in a natural way. **(ii)** The model makes no explicit assumptions about the type of interactions between items. If the learned conditional in Eq. (2) is close to the true conditional in Eq. (1), then the model can capture rich interactions—including diversity, similarity or others. Hence, our approach is data-driven rather than modeling specific types of interactions (such as multinomial logit), which is a key advantage. We demonstrate the benefits of this flexibility in our experiments (Section 4). **(iii)** The probability $p_\theta(\pi|x)$ is differentiable (in θ) for any fixed permutation π, which allows gradient-based learning (see Section 3). **(iv)** The computational cost of inference, dominated by the sequential decoding procedure, is $O(n^2)$, which is standard in seq2seq models with attention. We also consider a computationally cheaper single-step decoder with linear cost $O(n)$, which outputs a single vector $p^1 = p_\theta(\pi_1 = \cdot|x)$ (see Eq. (2)), from which we obtain π by sorting the values—similar to the approach taken in [Mottini and Acuna-Agost, 2017, Ai et al., 2018a]); we compare both approaches below.

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. In many real systems such base ranker is readily available. For example, the candidate set may be chosen from a huge item repository by an upstream model. Often candidate generator scores are available and can be used to obtain a base ranking via a simple sort. In this case we obtain the base ranking almost for free, as byproduct of candidate generation. Importantly, using a base ranker and focusing on re-ranking allows our seq2slate model to direct its modeling capacity at interactions between items rather than individual 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 approach can be easily extended to more informative feedback, such as the level of user engagement with the chosen item (e.g., time spent), but to simplify the presentation we focus on the binary case. 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, for example, mean average precision (MAP), precision at k, or normalized discounted cumulative

gain at k (NDCG@k). 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). In this case, the model is typically fed the ground-truth tokens as inputs to the next prediction step during training, an approach known as *teacher forcing* [Williams and Zipser, 1989]. Unfortunately, this approach cannot be applied in our setting since we only have access to weak supervision in the form of labels y (e.g., clicks), rather than ground-truth permutations. Instead, we next show how the seq2slate model can be trained directly from the labels y.

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:

$$\max_{\theta} \ \mathbb{E}_{\pi \sim p_\theta(.|x)}[\mathcal{R}(\pi, y)] \ .$$

One can use policy gradients and stochastic gradient ascent to optimize θ. The gradient is formulated using the popular REINFORCE update [Williams, 1992]:

$$\nabla_\theta \mathbb{E}_{\pi \sim p_\theta(.|x)}[\mathcal{R}(\pi, y)] = \mathbb{E}_{\pi \sim p_\theta(.|x)}\Big[\mathcal{R}(\pi, y) \nabla_\theta \log p_\theta(\pi \mid x)\Big] \ . \tag{3}$$

This can be approximated via Monte-Carlo sampling as follows:

$$\approx \frac{1}{B} \sum_{k=1}^{B} \Big(\mathcal{R}(\pi[k], y[k]) - b_{\mathcal{R}}(x[k])\Big) \nabla_\theta \log p_\theta(\pi[k] \mid x[k]) \ , \tag{4}$$

where k indexes ranking instances in a batch of size B, the $\pi[k]$ are permutations drawn from the model p_θ, and $b_{\mathcal{R}}(x)$ denotes a baseline function that estimates the expected rewards in order to reduce variance.

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 above, 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. Two natural, simple choices for ℓ are cross-entropy loss and hinge loss:

$$\ell_{xent}(s, y) = -\sum_i \hat{y}_i \log p_i \tag{5}$$
$$\ell_{hinge}(s, y) = \max\{0, 1 - \min_{i:y_i=1} s_i + \max_{j:y_j=0} s_j\} ,$$

where $\hat{y}_i = y_i / \sum_j y_j$, and p_i is a softmax of s, as in Eq. (2). Intuitively, with cross-entropy loss we try to assign high probabilities to positive labels [see also Kurata et al., 2016], while hinge loss is minimized when scores of items with positive labels are higher than scores of those with negative labels. Notice that both losses are convex functions of the scores s. To improve convergence, we consider a smooth version of the hinge loss where the maximum and minimum are replaced by their smooth counterparts: $\texttt{smooth-max}(s; \gamma) = \frac{1}{\gamma} \log \sum_i e^{\gamma s_i}$ (and smooth minimum is defined similarly, using $\min_i(s_i) = -\max_i(-s_i)$). Finally, we point out that any standard surrogate loss for ranking can be used as the per-step loss $\ell(s, y)$, including losses that depend on non-binary labels y, such as relevance scores.

As mentioned above, a main difference of seq2slate from previous approaches is its use of sequential decoding. This does complicate the training of the model somewhat relative to the the case of one-shot decoding [Mottini and Acuna-Agost, 2017, Ai et al., 2018a]. Specifically, if we simply apply a per-step loss from Eq. (5) to all steps of the output sequence while reusing the labels y at each step, then the loss is invariant to the resulting output permutation (i.e., predicting a positive item at the beginning of the sequence has the same cost as predicting it at the end). Instead, in order to train a seq2slate model we let the loss ℓ at each decoding step j ignore the items already chosen, so no further loss is incurred after a label is predicted correctly. In particular, for a *fixed* permutation π, define the *sequence loss*:

$$\mathcal{L}_\pi(S, y) = \sum_{j=1}^{n} w_j \, \ell_{\pi_{<j}}(s^j, y) , \tag{6}$$

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. (5)). Including a per-step weight w_j can encourage better performance earlier in the sequence. For example, we might set $w_j = 1/\log(j+1)$ (along the lines of DCG). Alternatively, if optimizing for a particular slate size k is desired, one can use the weights to restrict this loss to just the first k output steps.

We note that the loss above differs from the actual ranking measures used in evaluation (i.e., MAP, NDCG@k, etc.). On the other hand, any permutation that places the positive labels at the first positions gets 0 loss and optimizes all ranking measures, so in that sense the losses are aligned. This situation is quite common for surrogate losses in machine learning.

Using the definition of the sequence loss above, our goal is to optimize the expected loss:

$$\min_{\theta} \ \mathbb{E}_{\pi \sim p_\theta(\cdot|x)}[\mathcal{L}_\pi(\theta)] \ ,$$

where

$$\mathbb{E}_{\pi \sim p_\theta(\cdot|x)}[\mathcal{L}_\pi(\theta)] = \sum_{\pi} p_\theta(\pi|x)\mathcal{L}_\pi(\theta) \ . \tag{7}$$

This corresponds to sampling the permutation π according to the model, where π_j is drawn from $p_\theta(\cdot|\pi_{<j}, x)$ for each position j. For completeness, we derive the expected loss as a function of the model scores S in Appendix A.

Notice that the expected loss in Eq. (7) 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:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_\pi[\mathcal{L}_\pi(\theta)] &= \nabla_\theta \sum_{\pi} p_\theta(\pi|x)\mathcal{L}_\pi(\theta) \\
&= \sum_{\pi} \left[(\nabla_\theta p_\theta(\pi|x))\mathcal{L}_\pi(\theta) + p_\theta(\pi|x)(\nabla_\theta \mathcal{L}_\pi(\theta))\right] \\
&= \mathbb{E}_{\pi \sim p_\theta(\cdot|x)} \left[\mathcal{L}_\pi(\theta) \cdot \nabla_\theta \log p_\theta(\pi|x) + \nabla_\theta \mathcal{L}_\pi(\theta)\right] \ ,
\end{aligned} \tag{8}$$

which can be approximated from samples by:

$$\begin{aligned}
\approx \ \frac{1}{B} \sum_{k=1}^{B} \Bigg[&\Big(\mathcal{L}_{\pi[k]}(S(\theta), y[k]) - b_{\mathcal{L}}(x[k])\Big) \nabla_\theta \log p_\theta(\pi[k] \mid x[k]) \\
&+ \nabla_\theta \mathcal{L}_{\pi[k]}(S(\theta), y[k])\Bigg] \ .
\end{aligned} \tag{9}$$

Here $b_{\mathcal{L}}(x[k])$ is a baseline that approximates $\mathcal{L}_{\pi[k]}(\theta)$, introduced for variance reduction. This gradient is analogous to the REINFORCE update from Eq. (3)–(4), but where the loss $\mathcal{L}$ subsumes the role of the reward $\mathcal{R}$. Notice, however, that since the loss depends on the model parameters θ while the reward does not, the resulting update is quite different. Specifically, applying stochastic gradient descent intuitively decreases the probability of drawing samples with high losses (left term in Eq. (8)), as in REINFORCE, but in addition also reduces the loss of any sample (right term in Eq. (8)), which differs from REINFORCE [see also Schulman et al., 2015, Eq. (4)].

Greedy Decoding

In many seq2seq applications, using greedy decoding at test time performs better than sampling from the model [e.g., Ranzato et al., 2016, Leblond et al., 2018]. Therefore, it makes sense to also consider training the model using a greedy decoding policy, which is an alternative approach to sampling (cf. Eq. (9)). The greedy policy consists of selecting the item that maximizes $p_\theta(\cdot|\pi_{<j}, x)$ at every step j. The resulting permutation π^* then satisfies $\pi^*_j = \operatorname{argmax}_i \ p_\theta(\pi_j = i|\pi^*_{<j}, x)$ and our loss simply becomes $\mathcal{L}_{\pi^*}(\theta)$. Unlike the sampling-based loss in Eq. (7), the greedy policy loss is not continuous everywhere since a small change in the scores S may result in a jump between permutations π^*, and therefore a jump in the value of $\mathcal{L}_{\pi^*}(\theta)$. Specifically, the

loss is non-differentiable when any s^j has multiple maximizing arguments. Outside this measure-zero subspace, the loss is continuous (almost everywhere), and the gradient is well-defined.

For both training policies (sampling and greedy), we minimize the loss via stochastic gradient descent over mini-batches in an *end-to-end* fashion.

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.

Implementation details We set hyperparameters of our model to values inspired by the literature. All experiments use mini-batches of 128 training examples and LSTM cells with 128 hidden units. We train our models with the Adam optimizer [Kingma and Ba, 2014] and an initial learning rate of 0.0003 decayed every 1000 steps by a factor of 0.96. Network parameters are initialized uniformly at random in $[-0.1, 0.1]$. To improve generalization, we regularize the model by using dropout with probability of dropping $p_{dropout} = 0.1$ and L2 regularization with a penalty coefficient $\lambda = 0.0003$. Unless specified otherwise, all results use supervised training with cross-entropy loss ℓ_{xent} and the sampling policy. At inference time, we report metrics for the greedy policy. We use an exponential moving average with a decay rate of 0.99 as the baseline functions $b_{\mathcal{R}}(x)$ and $b_{\mathcal{L}}(x)$ in Eq. (4) and (9), respectively. When training the seq2slate model with REINFORCE, we use $\mathcal{R} = \text{NDCG@}10$ as the reward function and do not regularize the model (since we observed no overfitting during training with the noisy policy gradients). We also considered a bidirectional encoder RNN [Schuster and Paliwal, 1997], a stacked LSTM, and models with more hidden units, but found that these did not lead to significant improvements in our experiments.

4.1 Learning-to-Rank Benchmarks

To understand the behavior of the proposed model, we conduct experiments using two learning-to-rank datasets. We use two of the largest publicly available benchmarks: the Yahoo Learning to Rank Challenge data (set 1),[2] and the Microsoft Web30k dataset.[3] These datasets only provide feature vectors for each query-document pair, so all context (query) features are embedded within the item feature vectors themselves.

We adapt the procedure proposed by Joachims et al. [2017] to generate click data. The original procedure is as follows: first, a base ranker is trained from the raw data. We select this base ranker by training all models in the RankLib package,[4] and choosing the one with the best performance on each data set (MART for Yahoo and LambdaMART for Web30k). We generate an item ranking using the base model, which is then used

[2] https://webscope.sandbox.yahoo.com/catalog.php?datatype=c

[3] https://www.microsoft.com/en-us/research/project/mslr/

[4] https://sourceforge.net/p/lemur/wiki/RankLib/

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.

to generate training data by simulating a user "cascade" model: a user observes each item with decaying probability $1/i^\eta$, where i is the base rank of the item and η is a parameter of the generative model. This simulates a noisy sequential scan by the user. An observed item is clicked if its ground-truth relevance score is above a threshold (relevant: $\{2, 3, 4\}$, irrelevant: $\{0, 1\}$), otherwise no click is generated.

Unfortunately, the original datasets only include a per-item relevance score, which is independent of the other items. This means that there are no direct high-order interactions between the clicks, and therefore the joint probability in Eq. (1) is just $p(\pi|x) = \prod_{j=1}^{n} p(\pi_j|x)$. In this case a pointwise ranker is optimal so there would be no need for seq2slate. Therefore, in order to introduce high-order dependencies, we augment the above procedure as follows, creating a generative process dubbed *diverse-clicks*. When observing a relevant item, the user will only click if it is not too similar to previously clicked items (i.e, diverse enough), thus reducing the total number of clicks. 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: $D_{ij} = \|x_i - x_j\|$. We use $\eta = 0$ (no decay, since clicks are sparse anyway due to the diversity term) and $q = 0.5$. We also discuss variations of this model below. Since our focus is on modeling high-order interactions, all results reported in this section are w.r.t. the generated binary labels and not the original relevance scores.

Using the generated training data, we train both our seq2slate model and baseline rankers from the RankLib package: AdaRank [Xu and Li, 2007], Coordinate Ascent [Metzler and Croft, 2007], LambdaMART [Wu et al., 2010], ListNet [Cao et al., 2007], MART [Friedman, 2001], Random Forests [Breiman, 2001], RankBoost [Freund et al., 2003], RankNet [Burges et al., 2005]. Some of these baselines use deep neural networks (e.g., RankNet, ListNet), so they are strong state-of-the-art models with comparable complexity to seq2slate. 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.

To better understand the behavior of the model, we visualize the probabilities of the attention from Eq. (2) for one of the test instances in Fig. 2. Interestingly, the model

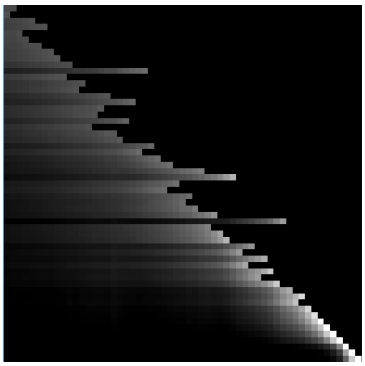

Figure 2: Visualization of attention probabilities on benchmark data. Intensities correspond to p_i^j for each item i in step j.

Ranker	Yahoo				Web30k			
	MAP	NDCG@5	NDCG@10	rank-gain	MAP	NDCG@5	NDCG@10	rank-gain
seq2slate	**0.67**	**0.69**	**0.75**	**7.4**	**0.51**	**0.53**	**0.59**	**18.3**
Greedy policy	0.66	0.69	0.75	7.2	0.50	0.52	0.59	18.3
smooth-hinge	0.66	0.69	0.75	7.1	0.49	0.51	0.58	17.9
REINFORCE	0.66	0.68	0.75	5.7	0.44	0.47	0.53	-0.5
one-step decoder	0.66	0.69	0.75	6.4	0.49	0.51	0.58	16.5
shuffled data	0.57	0.60	0.67	–	0.40	0.40	0.48	–
base ranker (no-op)	0.58	0.61	0.69	0	0.45	0.48	0.54	0

Table 2: Comparison of model and data variants for seq2slate on data generated with diverse-clicks.

produces slates that are close to the input ranking, but with some items demoted to lower positions, presumably due to the interactions with previous items.

We next consider several variations of the generative model and of the seq2slate model itself. Results are reported in Table 2. The rank-gain metric per example is computed by summing the positions change of all positive labels in the re-ranking, and this is averaged over all examples (queries).

Comparison of training variants In Table 2, we compare the different training variants outlined in Section 3, namely, cross entropy with the greedy or sampling policy, a smooth hinge loss with $\gamma = 1.0$, and REINFORCE. We find that supervised learning with cross entropy generally performs best, with the smooth hinge loss doing slightly worse. Our weakly supervised training methods have positive rank gain on all datasets, meaning they improve over the base ranker. The results from Table 2 suggest that training with REINFORCE yields comparable results on Yahoo but significantly worse results on the more challenging Web30k dataset. In terms of training time, REINFORCE needed 4X more time till convergence. We find no significant difference in performance between relying on the greedy and sampling policies during training.

One-step decoding We compare seq2slate to the model which uses a single decoding step, referred to as *one-step decoder* (see Section 2). In Table 2 we see that this model

Ranker	Yahoo			Web30k		
	MAP	NDCG@5	NDCG@10	MAP	NDCG@5	NDCG@10
seq2slate	0.82	0.82	0.84	**0.44**	**0.54**	**0.50**
AdaRank	0.83	0.81	0.84	0.41	0.52	0.48
Coordinate Ascent	0.83	0.82	0.85	0.39	0.47	0.44
LambdaMART	**0.84**	**0.83**	**0.85**	0.41	0.52	0.48
ListNet	0.83	0.83	0.85	0.41	0.53	0.49
MART	0.83	0.82	0.85	0.41	0.52	0.48
Random Forests	0.83	0.82	0.84	0.40	0.48	0.45
RankBoost	0.83	0.83	0.85	0.38	0.43	0.41
RankNet	0.83	0.82	0.84	0.35	0.36	0.35

Table 3: Performance of seq2slate and other baselines on data generated with similar-clicks.

has comparable performance to the sequential decoder. One possible explanation for the comparable performance of the one-step decoder is that the interactions in our generated data are rather simple and can be effectively learned by the encoder. By contrast, in Section 4.2 we show that on more complex real-world data, sequential decoding can perform significantly better than one-step decoding. In terms of runtime, we observed a 4X decrease in training time and a 3X decrease in inference time for the one-step decoder compared to sequential decoding (for the real-world data in Section 4.2 below, one-step decoding was 2.5X faster per iteration in both training and inference). This suggests that when inference time is crucial, as in many real-world systems, one might prefer the faster single-shot option. Having said that, we point out that even with sequential decoding the runtime was not a bottleneck in our case and we were able to train a seq2slate model on millions of examples in a couple of hours, and serve live traffic in $O(10)$ milliseconds. For this reason we also did not make an effort to optimize the code, so the numbers above can probably be reduced significantly.

Sensitivity to input order Previous work suggests that the performance of seq2seq models is often sensitive to the order in which the input is processed [Vinyals et al., 2016, Nam et al., 2017, Ai et al., 2018a]. To test the sensitivity of seq2slate to the order in which items are processed, we consider the use of seq2slate without relying on the base ranker to order the input. Instead, items are fed to the model in random order. Since learning the correct ranking from a single example may be hard, we generate multiple copies of each training example, each with a different randomly shuffled input order. Specifically, in Table 2 we show results for 10 generated examples per original example under 'shuffled data'. The results show that the performance is indeed significantly worse in this case, which is consistent with previous studies. It suggests that reranking is an easier task than ranking from scratch.

Adaptivity to the type of interaction To demonstrate the flexibility of seq2slate, we generate data using a variant of the diverse-clicks model above. Specifically, in the *similar-clicks* model, the user also clicks on observed irrelevant items if they are similar to previously clicked items (increasing the number of total clicks). As above, we use the pairwise distances in feature space D_{ij} to determine similarity. For this model we use

Ranker	Yahoo				Web30k			
	MAP	NDCG@5	NDCG@10	rank-gain	MAP	NDCG@5	NDCG10	rank-gain
seq2slate	**0.82**	**0.82**	**0.84**	**8.5**	**0.44**	**0.54**	**0.50**	**16.0**
Greedy policy	**0.82**	**0.82**	**0.84**	**8.5**	0.44	0.54	0.50	15.9
smooth-hinge	0.80	0.80	0.82	7.7	0.44	0.54	0.50	15.9
REINFORCE	**0.82**	**0.82**	**0.84**	**8.5**	0.42	0.53	0.49	-14.8
one-step decoder	0.81	0.81	0.82	7.7	0.44	0.53	0.49	15.5
shuffled data	0.79	0.78	0.79	–	0.42	0.48	0.46	–
base ranker (no-op)	0.78	0.76	0.79	0	0.43	0.53	0.49	0

Table 4: Comparison of model and data variants for seq2slate on data generated with similar-clicks.

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 5: Performance compared to a competitive base production ranker on real data.

$q = 0.5$, and $\eta = 0.3$ for Web30k, $\eta = 0.1$ for Yahoo, to keep the proportion of positive labels similar.[5] The results in Table 3 show that seq2slate has comparable performance to the baseline rankers, with slightly lower performance on Yahoo and significantly better performance on the harder Web30k data. This demonstrates that our model can adapt to various types of interactions in the data. Notice that no changes to the model or training algorithm were necessary for seq2slate. In contrast, if one used a specific interaction model for 'diverse-clicks', then a different model would be required for the 'similar-clicks' data, a distinction not needed with seq2slate.

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) with cross-entropy loss, the greedy policy, L2-regularization and dropout. 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 5 shows the performance of seq2slate and the one-step decoder compared to the production base ranker on test data (of roughly the same size as the training data). 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.

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 per position for this traffic as well. Fig. 3 shows the difference in CTR per position,

[5]The value of η was chosen such that the percentage of examples with no positive labels (clicks) at all remained small enough and roughly the same in all datasets (around 1.15% of all examples).

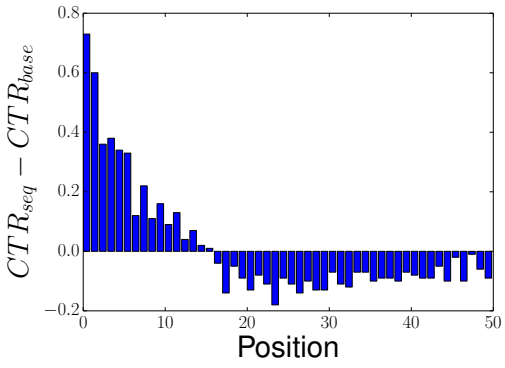

Figure 3: Difference in CTR per position between a seq2slate model and a base production ranker in a live experiment.

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.

5 Related Work

In this section we discuss additional related work. We build on the recent impressive success of seq2seq models in complex prediction tasks, including machine translation [Sutskever et al., 2014, Bahdanau et al., 2015], parsing [Vinyals et al., 2015a], combinatorial optimization [Vinyals et al., 2015b, Bello et al., 2017], multi-label classification [Wang et al., 2016, Nam et al., 2017], and others. Our work differs in that we explicitly target the ranking task, which requires a novel approach to training seq2seq models from weak feedback (click-through data).

Most of the work on ranking mentioned above uses shallow representations. However, in recent years deep models have been used for information retrieval, focusing on embedding queries, documents and query-document pairs [Huang et al., 2013, Guo et al., 2016, Palangi et al., 2016, Wang and Klabjan, 2017, Pang et al., 2017] (see also recent survey by Mitra and Craswell [2017]). Rather than embedding individual items, in seq2slate a representation of the entire slate of items is learned and encoded in the RNN state. Moreover, learning the embeddings (x) can be easily incorporated into the training of the sequence model to optimize both simultaneously end-to-end.

Closest to ours are the recent works of Mottini and Acuna-Agost [2017] and Ai et al. [2018a], where an RNN is used to encode a set of items for ranking. There are some differences between the approach of Ai et al. [2018a] and ours, including using GRU cells instead of LSTM cells, reversing the input order (the highest ranking item is fed to the encoder last), and training from relevance scores instead of click-through data. More importantly, both works [Mottini and Acuna-Agost, 2017, Ai et al., 2018a] use a single decoding step. In contrast, we apply sequential decoding, which directly allows item scores to change based on previously chosen items. We believe that this significantly

simplifies modeling and inference with complex high-order interactions between items, and indeed show that it performs much better in practice (see Section 4.2).

Finally, Santa Cruz et al. [2017] recently proposed an elegant deep learning framework for learning permutations based on the so called Sinkhorn operator, building on prior work by Adams and Zemel [2011]. Their approach uses a continuous relaxation of permutation matrices (i.e., the set of doubly-stochastic matrices, or the Birkhoff polytope). Followup work has focused on improved training and inference procedures, including a Gumbel softmax distribution to enable efficient learning [Mena et al., 2018], a reparameterization of the Birkhoff Polytope for variational inference [Linderman et al., 2018], and an Actor-Critic policy gradient training procedure [Emami and Ranka, 2018]. However, these works are focused on reconstruction of scrambled objects (i.e., matchings), and it is not obvious how to extend it to our ranking setting, where no ground-truth permutation is available.

6 Conclusion

We presented a novel approach to ranking sets of items called seq2slate. We found the formalism of pointer-networks particularly suitable for this setting. We emphasized the modeling and computational advantages of using sequential decoding, which allowed the model to dynamically adjust placement of items on the slate given previous choices. We addressed the challenge of training the model from weak user feedback (click-trough logs) to improve the ranking quality. To this end, we proposed new sequence losses along with corresponding gradient-based updates. Our experiments show that the proposed approach is highly scalable and can deliver significant improvements in ranking results.

Our work can be extended in several directions. In terms of architecture, we aim to explore the *Transformer* network [Vaswani et al., 2017, Dehghani et al., 2019] in place of the RNN. Several algorithmic variants can potentially improve the performance of our model. For inference, beam-search has been shown to improve predictions of several seq2seq models [Wiseman and Rush, 2016], and we believe can do the same for seq2slate. For training, several approaches have been recently proposed for seq2seq models, including Actor-Critic [Bahdanau et al., 2017] and more recently SeaRNN [Leblond et al., 2018], and it will be interesting to test their performance in the ranking setting.

Finally, an interesting future direction is to study off-policy correction for seq2slate [Joachims et al., 2018, Chen et al., 2019]. In this setting, training examples are assigned *importance weights* in order to account for the fact that the labels were obtained using a different policy than the one we wish to evaluate during training. In particular, the expected sequence loss is adjusted to account for this mismatch as follows:[6]

$$\mathbb{E}_{\pi\sim p_\theta(.|x)}[\mathcal{L}_\pi(\theta)] = \mathbb{E}_{\pi\sim p_{\text{base}}(.|x)}\left[\frac{p_\theta(\pi|x)}{p_{\text{base}}(\pi|x)}\mathcal{L}_\pi(\theta)\right] ,$$

[6]Substituting $\mathcal{L}_\pi(\theta)$ by $\mathcal{R}(\pi, y)$ yields an equivalent formulation for the expected reward from Section 3.1.

where p_{base} is the probability of π under the base ranker (i.e., logging policy). This expectation can then be approximated from logged samples as in Section 3. We leave this extension to future work.

A Derivation of the Expected Loss

Here we show the expected loss as a function of the model scores S,

$$
\begin{aligned}
\mathbb{E}[\mathcal{L}(\theta)] &= \sum_{\pi} p(\pi)\mathcal{L}_{\pi}(\theta) \\
&= \sum_{\pi} p(\pi) \sum_{j} \ell_{\pi_{<j}}(\theta) \\
&= \sum_{j} \sum_{\pi} p(\pi_{<j}) p(\pi_{\geq j}|\pi_{<j}) \ell_{\pi<j}(\theta) \\
&= \sum_{j} \sum_{\pi_{<j}} p(\pi_{<j}) \ell_{\pi_{<j}}(\theta) \overbrace{\sum_{\pi_{\geq j}} p(\pi_{\geq j}|\pi_{<j})}^{1} \\
&= \sum_{j} \sum_{\pi_{<j}} \left(\prod_{k=1}^{j-1} e^{s^k_{\pi_k}} / \sum_{i \notin \pi_{<k}} e^{s^k_i} \right) \ell_{\pi<j}(s^j, y) \ .
\end{aligned}
$$

References

Ryan Prescott Adams and Richard S Zemel. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. Learning a deep listwise context model for ranking refinement. In *SIGIR*, pages 135–144, 2018a.

Qingyao Ai, Xuanhui Wang, Nadav Golbandi, Michael Bendersky, and Marc Najork. Learning groupwise scoring functions using deep neural networks. 2018b.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. In *ICLR*, 2017.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *ICLR 2017 – Workshop Track*, 2017.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*, 2015.

Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 13 (3):41:1–41:25, 2017.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *Internation Conference on Learning Representations (ICLR)*. 2019.

Puneet Kumar Dokania, Aseem Behl, CV Jawahar, and M Pawan Kumar. Learning to rank using high-order information. In *European Conference on Computer Vision*, pages 609–623. Springer, 2014.

Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. 2018.

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov): 933–969, 2003.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *International Conference on Information and Knowledge Management (CIKM)*, pages 55–64, 2016.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Support vector learning for ordinal regression. In *ICANN*, pages 97–102, 1999.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computations*, 1997.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

T. Joachims, A. Swaminathan, and M. de Rijke. Deep learning with logged bandit feedback. In *International Conference on Learning Representations (ICLR)*, 2018.

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

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

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Gakuto Kurata, Bing Xiang, and Bowen Zhou. Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *ACL*, 2016.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. SEARNN: Training rnns with global-local losses. In *ICLR*, 2018.

Scott Linderman, Gonzalo Mena, Hal Cooper, Liam Paninski, and John Cunningham. Reparameterizing the birkhoff polytope for variational permutation inference. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*. PMLR, 2018.

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

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Donald Metzler and W Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. `arXiv:1705.01509`, 2017.

Alejandro Mottini and Rodrigo Acuna-Agost. 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.

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

H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(4):694–707, 2016.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, 2017.

Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, Wen-Ying Xiong, and Hang Li. Learning to rank relational objects and its application to web search. In *Proceedings of WWW*, pages 407–416. ACM, 2008.

Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. Global ranking using continuous conditional random fields. In *Advances in neural information processing systems*, pages 1281–1288, 2009.

MarcAurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *Proceedings of ICLR*, 2016.

Nir Rosenfeld, Ofer Meshi, Danny Tarlow, and Amir Globerson. Learning structured models with the auc loss and its generalizations. In *Artificial Intelligence and Statistics*, pages 841–849, 2014.

Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Visual permutation learning. In *CVPR*, 2017.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient Estimation Using Stochastic Computation Graphs. In *NIPS*, 2015.

M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, 1997. ISSN 1053-587X. doi: 10.1109/78.650093.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017.

O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *NIPS*, 2015a.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, pages 2692–2700, 2015b.

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

Baiyang Wang and Diego Klabjan. An attention-based deep net for learning to rank. *arXiv preprint arXiv:1702.06106*, 2017.

Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. CNN-RNN: A unified framework for multi-label image classification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2285–2294. IEEE, 2016.

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

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, June 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL `http://dx.doi.org/10.1162/neco.1989.1.2.270`.

Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *ACL*, 2016.

Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM, 2007.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of SIGIR*, pages 271–278. ACM, 2007.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Yadong Zhu, Yanyan Lan, Jiafeng Guo, Xueqi Cheng, and Shuzi Niu. Learning for search result diversification. In *Proceedings of SIGIR*, pages 293–302. ACM, 2014.