Ask the GRU: Multi-Task Learning for Deep Text Recommendations

Trapit Bansal
tbansal@cs.umass.edu

David Belanger
belanger@cs.umass.edu

Andrew McCallum
mccallum@cs.umass.edu

College of Information and Computer Sciences, University of Massachusetts Amherst

ABSTRACT

In a variety of application domains the content to be recommended to users is associated with text. This includes research papers, movies with associated plot summaries, news articles, blog posts, etc. Recommendation approaches based on latent factor models can be extended naturally to leverage text by employing an explicit mapping from text to factors. This enables recommendations for new, unseen content, and may generalize better, since the factors for all items are produced by a compactly-parametrized model. Previous work has used topic models or averages of word embeddings for this mapping. In this paper we present a method leveraging deep recurrent neural networks to encode the text sequence into a latent vector, specifically gated recurrent units (GRUs) trained end-to-end on the collaborative filtering task. For the task of scientific paper recommendation, this yields models with significantly higher accuracy. In cold-start scenarios, we beat the previous state-of-the-art, all of which ignore word order. Performance is further improved by multi-task learning, where the text encoder network is trained for a combination of content recommendation and item metadata prediction. This regularizes the collaborative filtering model, ameliorating the problem of sparsity of the observed rating matrix.

Keywords

Recommender Systems; Deep Learning; Neural Networks; Cold Start; Multi-task Learning

1. INTRODUCTION

Text recommendation is an important problem that has the potential to drive significant profits for e-businesses through increased user engagement. Examples of text recommendations include recommending blogs, social media posts [1], news articles [2, 3], movies (based on plot summaries), products (based on reviews) [4] and research papers [5].

Methods for recommending text items can be broadly classified into collaborative filtering (CF), content-based, and hybrid methods. Collaborative filtering [6] methods use the user-item rating matrix to construct user and item profiles from past ratings. Classical examples of this include matrix factorization methods [6, 7] which completely ignore text information and rely solely on the rating matrix. Such methods suffer from the *cold-start* problem – how to rank unseen or unrated items – which is ubiquitous in most domains. Content-based methods [8, 9], on the other hand, use the item text or attributes, and make recommendations based on similarity between such attributes, ignoring data from other users. Such methods can make recommendations for new items but are limited in their performance since they cannot employ similarity between user preferences [5, 10, 11]. Hybrid recommendation systems seek the best of both worlds, by leveraging both item content and user-item ratings [5, 10, 12, 13, 14]. Hybrid recommendation methods that consume item text for recommendation often ignore word order [5, 13, 14, 15], and either use bags-of-words as features for a linear model [14, 16] or define an unsupervised learning objective on the text such as a topic model [5, 15]. Such methods are unable to fully leverage the text content, being limited to bag-of-words sufficient statistics [17], and furthermore unsupervised learning is unlikely to focus on the aspects of text relevant for content recommendation.

In this paper we present a method leveraging *recurrent neural networks* (RNNs) [18] to represent text items for collaborative filtering. In recent years, RNNs have provided substantial performance gains in a variety of natural language processing applications such as language modeling [19] and machine translation [20]. RNNs have a number of noteworthy characteristics: (1) they are sensitive to word order, (2) they do not require hand-engineered features, (3) it is easy to leverage large unlabeled datasets, by pretraining the RNN parameters with unsupervised language modeling objectives [21], (4) RNN computation can be parallelized on a GPU, and (5) the RNN applies naturally in the cold-start scenario, as a feature extractor, whenever we have text associated with new items.

Due to the extreme data sparsity of content recommendation datasets [22], regularization is also an important consideration. This is particularly important for deep models such as RNNs, since these high-capacity models are prone to overfitting. Existing hybrid methods have used unsupervised learning objectives on text content to regularize the parameters of the recommendation model [4, 23, 24]. However, since we consume the text directly as an input for prediction, we can not use this approach. Instead, we provide regularization by performing multi-task learning combining collaborative filtering with a simple side task: predicting item meta-data such as genres or item tags. Here, the network producing vector representations for items directly from their text content is shared for both tag prediction and recommendation tasks. This allows us to make predictions in cold-start conditions, while providing regularization for the recommendation model.

We evaluate our recurrent neural network approach on the task of scientific paper recommendation using two publicly

available datasets, where items are associated with text abstracts [5, 13]. We find that the RNN-based models yield up to **34%** relative-improvement in Recall@50 for cold-start recommendation over collaborative topic regression (CTR) approach of Wang and Blei [5] and a word-embedding based model model [25], while giving competitive performance for warm-start recommendation. We also note that a simple linear model that represents documents using an average of word embeddings trained in a completely supervised fashion [25], obtains competitive results to CTR. Finally, we find that multi-task learning improves the performance of all of the models significantly, including the baselines.

2. BACKGROUND AND RELATED WORK

2.1 Problem Formulation and Notation

This paper focuses on the task of recommending items associated with text content. The j-th text item is a *sequence* of n_j word tokens, $X_j = (w_1, w_2, \ldots, w_{n_j})$ where each token is one of V words from a vocabulary. Additionally, the text items may be associated with multiple *tags* (user or author provided). If item $j \in [N_d]$ has tag $l \in [N_t]$ then we denote it by $t_{jl} = 1$ and 0 otherwise.

There are N_u users who have liked/rated/saved some of the text items. The rating provided by user i on item j is denoted by r_{ij}. We consider the implicit feedback [26, 27] setting, where we only observe whether a person has viewed or liked an item and do not observe explicit ratings. $r_{ij} = 1$ if user i liked item j and 0 otherwise. Denote the user-item matrix of likes by R. Let R_i^+ denote the set of all items liked by user i and R_i^- denote the remaining items.

The recommendation problem is to find for each user i a personalized ranking of all unrated items, $j \in R_i^-$, given the text of the items $\{X_j\}$, the matrix of users' previous likes $\{R_i\}$ and the tagging information of the items $\{t_{il}\}$.

The methods we consider will often represent users, items, tags and words by K-dimensional vectors $\tilde{u}_i$, $\tilde{v}_j$, $\tilde{t}_l$ and $\tilde{e}_w \in \mathbb{R}^K$, respectively. We will refer to such vectors as *embeddings*. All vectors are treated as column vectors. $\sigma(.)$ will denote the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$.

2.2 Latent Factor Models

Latent factor models [6] for content recommendation learn K dimensional vector embeddings of items and users:

$$\hat{r}_{ij} = b_i + b_j + \tilde{u}_i^T \tilde{v}_j, \tag{1}$$

b_i, b_j are user and item specific biases, and $\tilde{u}_i$ is the vector embedding for user i and $\tilde{v}_j$ is the embedding of item j.

A simple method for learning the model parameters, $\boldsymbol{\theta} = \{b_i, b_j, \tilde{u}, \tilde{v}\}$, is to specify a cost function and perform stochastic gradient descent. For implicit feedback, an unobserved rating might indicate that either the user does not like the item or the user has never seen the item. In such cases, it is common to use a *weighted* regularized squared loss [5, 26]:

$$C_R(\boldsymbol{\theta}) = \frac{1}{|R|} \sum_{(i,j)\in R} c_{ij}(\hat{r}_{ij} - r_{ij})^2 + \Omega(\boldsymbol{\theta}) \tag{2}$$

Often, one uses $c_{ui} = a$ for observed items and $c_{ui} = b$ for unobserved items, with $b \ll a$ [5, 13], signifying the uncertainty in the unobserved ratings. $\Omega(\boldsymbol{\theta})$ is a regularization on the parameters, for example in PMF [7] the embeddings are assigned Gaussian priors, which leads to a ℓ_2 regularization. Some implicit feedback recommendation systems use a ranking-based loss instead [25, 27]. The methods we propose can be trained with any differentiable cost function. We will use a weighted squared loss in our experiments to be consistent with the baselines [5].

2.3 The Cold Start Problem

In many applications, the factorization (1) is unusable, since it suffers from the *cold-start* problem [11, 14]: new or unseen items can not be recommended to users because we do not have an associated embedding. This has lead to increased interest in hybrid CF methods which can leverage additional information, such as item content, to make cold-start recommendations. In some cases, we may also face a cold-start problem for new users. Though we do not consider this case, the techniques of this paper can be extended naturally to accommodate it whenever we have text content associated with users. We consider:

$$\hat{r}_{ij} = b_i + b_j + \tilde{u}_i^T f(X_j), \tag{3}$$

Where $f(\cdot)$ is a vector-valued function of the item's text. For differentiable $f(\cdot)$, (3) can also be trained using (2). Throughout the paper, we will refer to $f(\cdot)$ as an *encoder*. Existing hybrid CF methods [11, 16, 28, 29] which use item metadata take this form. In such cases, $f(.)$ is a linear function of manually extracted item features. For example, Agarwal and Chen [16], Gantner et al. [30] incorporate side information through a linear regression based formulation on metadata like category, user's age, location, etc. Rendle [29] proposed a more general framework for incorporating higher order interactions among features in a factor model. Refer to Shi et al. [28], and the references therein, for a recent review on such hybrid CF methods.

Our experiments compare to collaborative topic regression (CTR) [5], a state-of-the-art technique that simultaneously factorizes the item-word count matrix (through probabilistic topic modeling) and the user-item rating matrix (through a latent factor model). By learning low-dimensional (topical) representations of items, CTR is able to provide recommendations to unseen items.

2.4 Regularization via Multi-task Learning

Typical CF datasets are highly sparse, and thus it is important to leverage all available training signals [22]. In many applications, it is useful to perform multi-task learning [31] that combines CF and auxiliary tasks, where a shared feature representation for items (or users) is used for all tasks. Collective matrix factorization [32] jointly factorizes multiple observation matrices with shared entities for relational learning. Ma et al. [33] seek to predict side information associated with users. Finally, McAuley and Leskovec [4] used topic models and Almahairi et al. [24] used language models on review text.

In many applications, text items are associated with tags, including research papers with keywords, news articles with user or editor provided labels, social media posts with hashtags, movies with genres, etc. These can be used as features X_j in (3) [16, 29]. However, there are considerable drawbacks to this approach. First, tags are often assigned by users, which may lead to a cold-start problem [34], since new items have no annotation. Moreover, tags can be noisy, especially if they are user-assigned, or too general [3].

While tag annotation may be unreliable and incomplete as input features, encouraging items' representations to be predictive of these tags can yield useful regularization for the CF problem. Besides providing regularization, this multi-task learning approach is especially useful in cold-start scenarios, since the tags are only used at train time and hence need not be available at test time. In Section 3.3 we employ this approach.

2.5 Deep Learning

In our work, we represent the item-to-embedding mapping $f(\cdot)$ using a deep neural network. See [35] for a comprehensive overview of deep learning methods. We provide here a brief review of deep learning for recommendation systems.

Neural networks have received limited attention from the recommendation systems community. [36] used restricted Boltzmann machines as one of the component models to tackle the Netflix challenge. Recently, [37, 38] proposed denoising auto-encoder based models for collaborative filtering which are trained to denoise corrupted versions of entire sparse vectors of user-item likes or item-user likes (i.e. rows or columns of the R matrix). However, these models are unable to handle the cold-start problem. Wang et al. [13] addresses this by incorporating a bag-of-words autoencoder in the model within a Bayesian framework. Elkahky et al. [39] proposed to use neural networks on manually extracted user and item feature representations for content based multi-domain recommendation. Dziugaite and Roy [40] proposed to use a neural network to learn the similarity function between user and item latent factors. Van den Oord et al. [41], Wang and Wang [42] developed music recommender systems which use features extracted from the music audio using convolutional neural networks (CNN) or deep belief networks. However, these methods process the user-item rating matrix in isolation from the content information and thus are unable to exploit the direct interaction between item content and ratings [13]. Weston et al. [43] proposed a CNN based model to predict hashtags on social media posts and found the learned representations to also be useful for document recommendation. Recently, He and McAuley [44] used image-features from a separately trained CNN to improve product recommendation and tackle cold-start. Almahairi et al. [24] used neural network based language models [19, 45] on review text to regularize the latent factors for product recommendation, as opposed to using topic models, as in McAuley and Leskovec [4]. They found that RNN based language models perform poorly as regularizers and word embedding models Mikolov et al. [45] perform better.

3. DEEP TEXT REPRESENTATION FOR COLLABORATIVE FILTERING

This section presents neural network-based encoders for explicitly mapping an item's text content to a vector of latent factors. This allows us to perform cold-start prediction on new items. In addition, since the vector representations for items are tied together by a shared parametric model, we may be able to generalize better from limited data.

As is standard in deep learning approaches to NLP, our encoders first map input text $X_j = (w_1, w_2, \ldots, w_{n_j})$ to a sequence of K_w-dimensional embeddings [45], $(e_1, e_2, \ldots, e_{n_j})$, using a lookup table with one vector for every word in our vocabulary. Then, we define a transformation that collapses the sequence of embeddings to a single vector, $g(X_j)$.

In all of our models, we maintain a separate item-specific embedding $\tilde{v}_j$, which helps capture user behavior that cannot be modeled by content alone [5]. Thus, we set the final document representation as:

$$f(X_j) = g(X_j) + \tilde{v}_j \tag{4}$$

For cold-start prediction, there is no training data to estimate the item-specific embedding and we set $\tilde{v}_j = 0$ [5, 13].

3.1 Order-Insensitive Encoders

A simple order-insensitive encoder of the document text can be obtained by averaging word embeddings:

$$g(X_j) = \frac{1}{|X_j|} \sum_{w \in X_j} e_w. \tag{5}$$

This corresponds exactly to a linear model on a bag-of-words representation for the document. However, using the representation (5) is useful because the word embeddings can be pre-trained, in an unsupervised manner, on a large corpus [46]. Note that (5) is similar to the embedding-based model used in Weston et al. [43] for hastag prediction.

Note that CTR [5], described in 2.3, also operates on bag-of-words sufficient statistics. Here, it does not have an explicit parametric encoder $g(\cdot)$ from text to a vector, but instead defines an implicit mapping via the process of doing posterior inference in the probabilistic topic model.

3.2 Order-Sensitive Encoders

Bag-of-words models are limited in their capacity, as they cannot distinguish between sentences that have similar unigram statistics but completely different meanings [17]. As a toy example, consider the research paper abstracts: "This paper is about deep learning, not LDA" and "This paper is about LDA, not deep learning". They have the same unigram statistics but would be of interest to different sets of users. A more powerful model that can exploit the additional information inherent in word order would be expected to recognize this and thus perform better recommendation.

In response, we parametrize $g(\cdot)$ as a recurrent neural network (RNN). It reads the text one word at a time and produces a single vector representation. RNNs can provide impressive compression of the salient properties of text. For example, accurate translation of an English sentence can be performed by conditioning on a single vector encoding [47].

The extracted item representation is combined with a user embedding, as in (3), to get the predicted rating for a user-item pair. The model can then be trained for recommendation in a completely supervised manner, using a differentiable cost function such as (2). Note that a key difference between this approach and the existing approaches which use item content [5, 13], apart from sensitivity to word order, is that we do not define an unsupervised objective (like likelihood of observing bag-of-words under a topic model) for extracting a text representation. However, our model can benefit from unsupervised data through pre-training of word embeddings [45] or pre-training of RNN parameters using language models [21] (our experiments use embeddings).

3.2.1 *Gated Recurrent Units (GRUs)*

Traditional RNN architectures suffer from the problem of vanishing and exploding gradients [48], rendering optimiza-

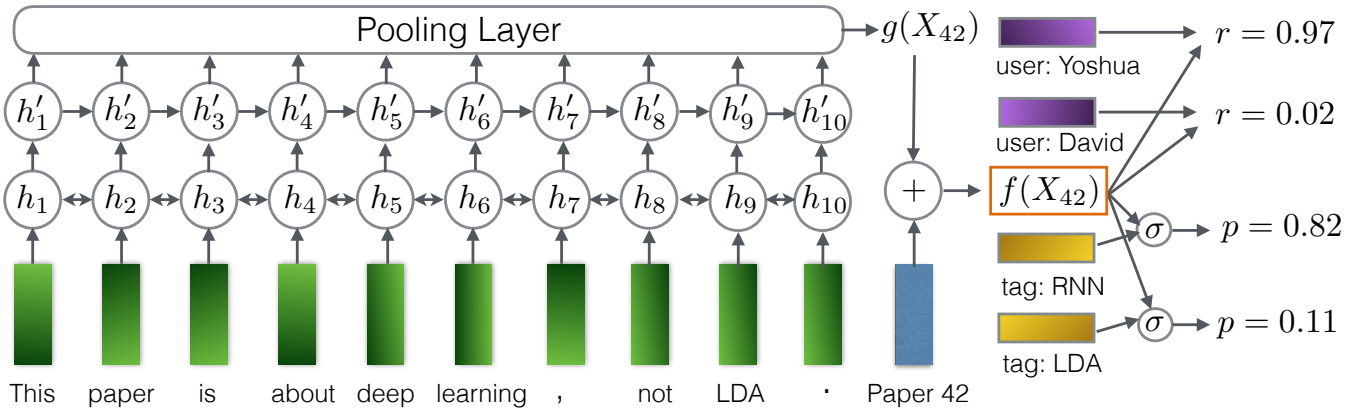

Figure 1: Proposed architecture for text item recommendation. Rectangular boxes represent embeddings. Two layers of RNN with GRU are used, where the first layer is a bi-directional RNN. The output of all the hidden units at the second layer is pooled to produce a text encoding which is combined with an item-specific embedding to produce the final representation $f(X)$. Users and tags are also represented by embeddings, which are combined with the item representation to do tag predection and recommendation.

tion difficult and prohibiting them from learning long-term dependencies. There have been several modifications to the RNN proposed to remedy this problem, of which the most popular are *long short-term memory units* (LSTMs) [49] and the more recent *gated recurrent units* (GRUs) [20]. We use GRUs, which are simpler than LSTM, have fewer parameters, and give competitive performance to LSTMs [50, 51].

The GRU hidden vector output at step t, h_t, for the input sequence $X_j = (w_1, \ldots, w_t, \ldots, w_{n_j})$ is given by:

$$\begin{bmatrix} f_t \\ o_t \end{bmatrix} = \sigma\left(\theta^1 \begin{bmatrix} \tilde{e}_{w_t} \\ h_{t-1} \end{bmatrix} + b\right) \quad (6)$$

$$c_t = \tanh(\theta^2_w \tilde{e}_{w_t} + f_t \odot \theta^2_h h_{t-1} + b_c) \quad (7)$$

$$h_t = (1 - o_t) \odot h_{t-1} + o_t \odot c_t \quad (8)$$

where $\theta^1 \in \mathbb{R}^{2K_h \times (K_w + K_h)}, \theta^2_w \in \mathbb{R}^{K_h \times K_w}, \theta^2_h \in \mathbb{R}^{K_h \times K_h}$ and $b, b_c \in \mathbb{R}^{K_h}$ are parameters of the GRU with K_w the dimension of input word embeddings and K_h the number of hidden units in the RNN. $\odot$ denotes element-wise product. Intuitively, f_t (6) acts as a 'forget' (or 'reset') gate that decides what parts of the previous hidden state to consider or ignore at the current step, c_t (7) computes a candidate state for the current time step using the parts of the previous hidden state as dictated by f_t, and o_t (6) acts as the output (or update) gate which decides what parts of the previous memory to change to the new candidate memory (8). All forget and update operations are differentiable to allow learning through backpropagation.

The final architecture, shown in Figure 1, consists of two stacked layers of RNN with GRU hidden units. We use a bi-directional RNN [52] at the first layer and feed the concatenation of the forward and backward hidden states as the input to the second layer. The output of the hidden states of the second layer is pooled to obtain the item content representation $g(X_j)$. In our experiments, mean pooling performs best. Models that use the final RNN state take much longer to optimize. Following (4), the final item representation is obtained by combining the RNN representation with an item-specific embedding v_j. We now describe the multi-task learning setup.

3.3 Multi-Task Learning

The encoder $f(\cdot)$ can be used as a generic feature extractor for items. Therefore, we can employ the multi-task learning approach of Section 2.4. The tags associated with papers can be considered as a (coarse) summary or topics of the items and thus forcing the encoder to be predictive of the tags will provide a useful inductive bias. Consider again the toy example of Figure 1. Observing the tag "RNN" but not "LDA" on the paper, even though the term LDA is present in the text, will force the network to pay attention to the sequence of words "not LDA" in order to explain the tags.

We define the probability of observing tag l on item j as: $P(t_{jl} = 1) = p_{jl} = \sigma(f(X_j)^T \tilde{t}_l)$, where $\tilde{t}_l$ is an embedding for tag l. The cost for predicting the tags is taken as the sum of the weighted binary log likelihood of each tag:

$$C_T(\boldsymbol{\theta}) = \frac{1}{|T|} \sum_j \sum_l \{t_{jl} \log p_{jl} + c'_{jl}(1 - t_{jl}) \log(1 - p_{jl})\}$$

where c'_{jl} down-weights the cost for predicting the unobserved tags. The final cost is $C(\boldsymbol{\theta}) = \lambda C_R(\boldsymbol{\theta}) + (1 - \lambda) C_T(\boldsymbol{\theta})$ with C_R defined in (2), and λ is a hyperparameter.

It is worth noting the differences between our approach and Almahairi et al. [24], who use language modeling on the text as an unsupervised multi-task objective with the item latent factors as the shared parameters. Almahairi et al. [24] found that the increased flexibility offered by the RNN makes it too strong a regularizer leading to worse performance than simpler bag-of-words models. In contrast, our RNN is trained fully supervised, which forces the item representations to be discriminative for recommendation and tag prediction. Furthermore, by using the text as an input to $g(\cdot)$ at test time, rather than just for train-time regularization, we can alleviate the cold-start problem.

4. EXPERIMENTS

4.1 Experimental Setup

Datasets: We use two datasets made available by Wang

Table 1: % Recall@50 for all the methods (higher is better).

	Citeulike-a			Citeulike-t		
	Warm Start	Cold Start	Tag Prediction	Warm Start	Cold Start	Tag Prediction
GRU-MTL	**38.33**	**49.76**	**60.52**	45.60	**51.22**	**62.32**
GRU	36.87	46.16	—	42.59	47.59	—
CTR-MTL	35.51	39.87	48.95	**46.82**	34.98	46.66
CTR	31.10	39.00	—	40.44	33.74	—
Embed-MTL	36.64	41.71	60.36	43.02	38.16	62.29
Embed	33.95	38.53	—	37.98	35.85	—

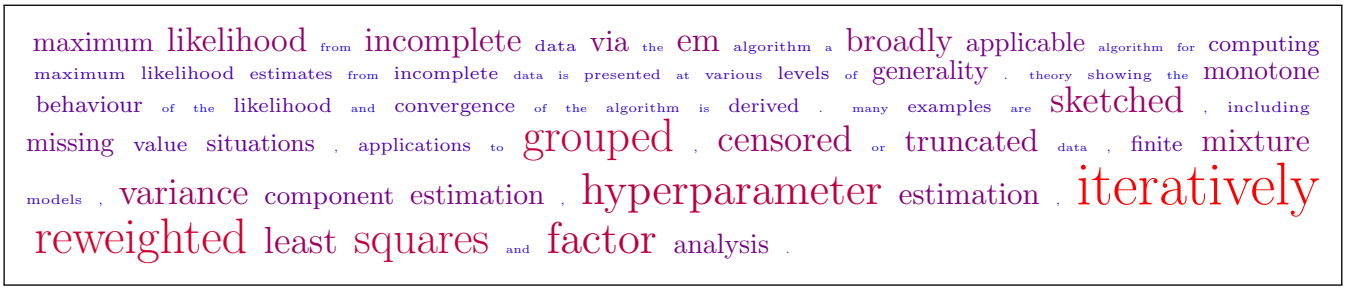

Figure 2: Saliency of each word in the abstract of the EM paper [53]. Size and color of the words indicate their leverage on the final rating. The model learns that chunks of word phrases are important, such as "maximum likelihood" and "iteratively reweighted least squares", and ignores punctutations and stop words.

et al. [13] from CiteULike[1]. CiteULike is an online platform which allows registered users to create personal libraries by saving papers which are of interest to them. The datasets consist of the papers in the users' libraries (which are treated as 'likes'), user provided tags on the papers, and the title and abstract of the papers. Similar to Wang and Blei [5], we remove users with less than 5 ratings (since they cannot be evaluated properly) and removed tags that occur on less than 10 articles. *Citeulike-a* [5] consists of 5551 users, 16980 papers and 3629 tags with a total of 204,987 user-item likes. *Citeulike-t* [5] consists of 5219 users, 25975 papers and 4222 tags with a total of 134,860 user-item likes. Note *Citeulike-t* is much more sparse (99.90%) than *Citeulike-a* (99.78%).

Evaluation Methodology: Following, Wang and Blei [5], we test the models on held-out user-article likes under both warm-start and cold-start scenarios.

Warm-Start: This is the case of in-matrix prediction, where every test item had at least one like in the training data. For each user we do a 5-fold split of papers from their like history. Papers with less than 5 likes are always kept in the training data, since they cannot be evaluated properly. After learning, we predict ratings across all active test set items and for each user filter out the items in their training set from the ranked list.

Cold-Start: This is the task of predicting user interest in a new paper with no existing likes, based on the text content of the paper. The set of all papers is split into 5 folds. Again, papers with less than 5 likes are always kept in training set. For each fold, we remove all likes on the papers in that fold forming the test-set and keep the other folds as training-set. We fit the models on the training set items for each fold and form predictive per-user ranking of items in the test set.

Evaluation Metric: Accuracy of recommendation from implicit feedback is often measured by recall. Precision is not reasonable since the zero ratings may mean that a user either does not like the article or does not know of it. Thus, we use Recall@M [5] and average the per-user metric:

$$\text{Recall@}M = \frac{\text{number of articles user liked in top } M}{\text{total number of articles user liked}}$$

4.1.1 Methods

We compare the proposed methods with *CTR*, which models item content using topic modeling. The approach put forth by CTR [5] cannot perform tag-prediction and thus, for a fair comparison, we modify CTR to do tag prediction. This can be viewed as a probabilistic version of collective matrix factorization [32]. Deriving an alternating least squares inference algorithm along the line of [5] is not possible for a sigmoid loss. Thus, for CTR, we formulate tag prediction using a weighted squared loss instead. Learning this model is a straightforward extension of CTR: rather than performing alternating updates on two blocks of parameters, we rotate among three. We call this *CTR-MTL*. The word embedding-based model with order-insensitive document encoder (section 3.1) is *Embed*, and the RNN-based model (section 3.2) is *GRU*. The corresponding models trained with multi-task learning are *Embed-MTL* and *GRU-MTL*.

4.1.2 Implementation Details

For CTR, we follow Wang and Blei [5] for setting hyper-parameters. We use latent factor dimension $K = 200$, regularization parameters $\lambda_u = 0.01$, $\lambda_v = 100$ and cost weights $a = 1$, $b = 0.01$. The same parameters gave good results for CTR-MTL. CTR and CTR-MTL are trained using the EM algorithm, which updates the latent factors using alternating least squares on full data [5, 26]. CTR is sensitive to good pre-processing of the text, which is common in topic modeling [5]. We use the provided pre-processed text for CTR,

[1]http://www.citeulike.org/

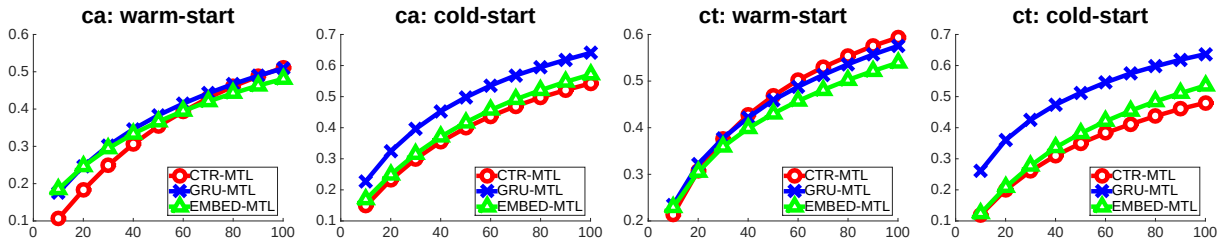

Figure 3: Recall@M for the models trained with multi-task learning. x-axis is the value of $M \in [100]$

.

which was obtained by removing stop-words and choosing top words based on tf-idf. We initialized CTR with the output of a topic model trained only on the text. We used the CTR code provided by the authors.

For the *Embed* and *GRU* models, we used word embeddings of dimension $K_w = K = 200$, in order to be consistent with CTR. For *GRU* models, the first layer of the RNN has hidden state dimension $K_{h1} = 400$ and the second layer (the output layer) has hidden state dimension $K_{h2} = 200$. We pre-trained the word embeddings using CBOW [45] on a corpus of 440,756 ACM abstracts (including the Citeulike abstracts). Dropout is used at every layer of the network. The probabilities of dropping a dimension are 0.1, 0.5 and 0.3 at the embedding layer, the output of the first layer and the output of the second layer, respectively. We also regularize the user embeddings with weight 0.01. We do very mild preprocessing of the text. We replace numbers with a <NUM> token and all words which have a total frequency of less than 5 by <UNK>. Note that we don't remove stop words or frequent words. This leaves a vocabulary of 21,129 words for *Citeulike-a* and 24,697 words for *Citeulike-t*.

The models are optimized via stochastic gradient descent, where mini-batch randomly samples a subset of B users and for each user we sample one positive and one negative example. We set the weights c_{ij} in (2) to $c_{ij} = 1 + \alpha \log(1 + \frac{|R_i|}{\epsilon})$, where $|R_i|$ is the number of items liked by user i, with $\alpha = 10$, $\epsilon = 1e-8$. Unlike Wang and Blei [5] we do not weight the cost function differently for positive and negative samples. Since the total number of negative examples is much larger than the positive examples for each user, stochastically sampling only one negative per positive example implicitly down-weights the negatives. We used a mini-batch size of $B = 512$ users and used Adam [54] for optimization. We run the models for a maximum of 20k mini-batch updates and use early-stopping based on recall on a validation set from the training examples.

4.2 Quantitative Results

Table 1 summarizes Recall@50 for all the models, on the two CiteULike datasets, for both warm-start and cold-start. Figure 3, further shows the variation of Recall@M for different values of M for the multi-task learning models.

Cold-Start: Recall that for cold-start recommendation, the item-specific embeddings $\tilde{v}_j$ in (4) are identically equal to zero, and thus the items' representations depend solely on their text content. We first note the performance of the models without multi-task learning. The GRU model is better than the best score of either the CTR model or the Embed model by **18.36%** (relative improvement) on CiteUlike-a and by **32.74%** on CiteULike-t. This significant gain demonstrates that the GRU model is much better at representing the content of the items. Improvements are higher on the CitULike-t dataset because it is much more sparse, and so models which can utilize content appropriately give better recommendations. CTR and Embed models perform competitively with each other.

Next, observe that multi-task learning uniformly improves performance for all models. The GRU model's recall improves by 7.8% on Citulike-a and by 7.6% on Citeulike-t. This leads to an overall improvement of **19.30%** on Citeulike-a and **34.22%** on Citeulike-t, over best of the baselines. Comparatively, improvement for CTR is smaller. This is expected since the Bayesian topic model provides strong regularization for the model parameters. Contrary to this, Embed models also benefits a lot by MTL (up to 8.2%). This is expected since unlike CTR, all the $K_w \times V$ parameters in the Embed model are free parameters which are trained directly for recommendation, and thus MTL provides necessary regularization.

Warm-Start: Collaborative filtering methods based on matrix factorization [6] often perform as well as hybrid methods in the warm-start scenario, due to the flexibility of the item-specific embeddings $\tilde{v}_j$ in (4) [5, 42]. Consider again the models trained without MTL. GRU model performs better than either the CTR or the Embed model, with relative improvement of **8.5%** on CiteULike-a and **5.3%** on CiteULike-t, over the best of the two models. Multi-task learning again improves performance for all the models. Improvements are particularly significant for CTR-MTL over CTR (up to 15.8%). Since the tags associated with test items were observed during training, they provide a strong inductive bias leading to improved performance. Interestingly, the GRU-MTL model performs slightly better than the CTR-MTL model on one dataset and slightly worse on the other. The first and third plots in Figure 3 demonstrate that the GRU-MTL performs slightly better than the CTR-MTL for smaller M, i.e. more relevant articles are ranked toward the top. To quantify this, we evaluate average reciprocal Hit-Rank@10 [3]. Given a list of M ranked articles for user i , let $c_1, c_2, \ldots, c_h$ denote the ranks of h articles in $[M]$ which the user actually liked. HR is then defined as $\sum_{t=1}^{h} \frac{1}{c_t}$ and tests whether top ranked articles are correct. GRU-MTL gives HR@10 of **0.098** and CTR-MTL gives HR@10 to be 0.077, which confirms that the top of the list for GRU-MTL contains more relevant recommendations.

Tag Prediction: Although the focus of the models is recommendation, we evaluate the performance of the multi-task models on tag prediction. We again use Recall@50 (defined per article) and evaluate in the cold-start scenario, where there are no tags present for the test article. The GRU and

Embed models perform similarly. CTR-MTL is significantly worse, which could be due to our use of the squared loss for training or because hyperparameters were selected for recommendation performance, not tag prediction.

4.3 Interpreting Prediction Decisions

We employ a simple, easy-to-implement tool for analyzing RNN predictions, based on Denil et al. [55] and Li et al. [56]. We produce a heatmap where every input word is associated with its leverage on the output prediction. Suppose that we recommended item j to user i. In other words, suppose that $\hat{r}_{ij}$ is large. Let $E_j = (e_{j,1}, e_{j,2}, \ldots, e_{j,n_j})$ be the sequence of word embeddings for item j. Since $f(\cdot)$ is encoded as a neural network, $\frac{d\hat{r}_{ij}}{de_{j,t}}$ can be obtained by backpropagation. To produce the heatmap's value for word t, we convert $\frac{d\hat{r}_{ij}}{de_{j,t}}$ into a scalar. This is not possible by backpropagation, as $\frac{d\hat{r}_{ij}}{dx_{j,t}}$ is not well-defined, since $x_{j,t}$ is a discrete index. Instead we compute $\|\frac{d\hat{r}_{ij}}{de_{j,t}}\|$. An application is in Figure 2.

5. CONCLUSION & FUTURE WORK

We employ deep recurrent neural networks to provide vector representations for the text content associated with items in collaborative filtering. This generic text-to-vector mapping is useful because it can be trained directly with gradient descent and provides opportunities to perform multi-task learning. For scientific paper recommendation, the RNN and multi-task learning both provide complementary performance improvements. We encourage further use of the technique in a variety of application domains. In future work, we would like to apply deep architectures to users' data and to explore additional objectives for multi-task learning that employ multiple modalities of inputs, such as movies' images and text descriptions.

6. ACKNOWLEDGMENT

This work was supported in part by the Center for Intelligent Information Retrieval, in part by The Allen Institute for Artificial Intelligence, in part by NSF grant #CNS-0958392, in part by the National Science Foundation (NSF) grant number DMR-1534431, and in part by DARPA under agreement number FA8750-13-2-0020. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

[1] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *SIGIR*, 2010.

[2] Owen Phelan, Kevin McCarthy, and Barry Smyth. Using twitter to recommend real-time topical news. In *RecSys*, 2009.

[3] Trapit Bansal, Mrinal Das, and Chiranjib Bhattacharyya. Content driven user profiling for comment-worthy recommendations of news and blog articles. In *RecSys*, 2015.

[4] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*, 2013.

[5] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *SIGKDD*, 2011.

[6] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[7] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, 2007.

[8] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[9] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *ACM conference on Digital libraries*, 2000.

[10] Chumki Basu, Haym Hirsh, William Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI*, 1998.

[11] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, 2002.

[12] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *ICML*, 2004.

[13] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *SIGKDD*, 2015.

[14] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI*, 2002.

[15] Prem K Gopalan, Laurent Charlin, and David Blei. Content-based recommendations with poisson factorization. In *NIPS*, 2014.

[16] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *SIGKDD*, 2009.

[17] Hanna M Wallach. Topic modeling: beyond bag-of-words. In *ICML*, 2006.

[18] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990.

[19] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. *INTERSPEECH*, 2010.

[20] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[21] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *NIPS*, pages 3061–3069, 2015.

[22] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

[23] Guang Ling, Michael R Lyu, and Irwin King. Ratings meet reviews, a combined approach to recommend. In *RecSys*, 2014.

[24] Amjad Almahairi, Kyle Kastner, Kyunghyun Cho, and Aaron Courville. Learning distributed representations from reviews for collaborative filtering. In *RecSys*, 2015.

[25] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

[26] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[27] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[28] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, 47(1):3, 2014.

[29] Steffen Rendle. Factorization machines. In *ICDM*, 2010.

[30] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *ICDM*, 2010.

[31] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[32] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *SIGKDD*, 2008.

[33] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*, 2008.

[34] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In *RecSys*, 2009.

[35] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in prep. for MIT Press, 2016.

[36] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, 2007.

[37] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW*, 2015.

[38] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*, 2016.

[39] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW*, 2015.

[40] Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[41] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NIPS*, 2013.

[42] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *International Conference on Multimedia*, 2014.

[43] Jason Weston, Sumit Chopra, and Keith Adams. # tagspace: Semantic embeddings from hashtags. 2014.

[44] R. He and J. McAuley. VBPR: visual bayesian personalized ranking from implicit feedback. In *AAAI*, 2016.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[46] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12: 2493–2537, 2011.

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

[48] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks*, 5(2):157–166, 1994.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[50] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[51] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

[52] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing*, 45(11): 2673–2681, 1997.

[53] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society.*, pages 1–38, 1977.

[54] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Misha Denil, Alban Demiraj, and Nando de Freitas. Extraction of salient sentences from labelled documents. *arXiv preprint arXiv:1412.6815*, 2014.

[56] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. 2016.