Improving Abstractive Tips Generation for Explaining Recommendations

Master Thesis

June 14, 2018

Michael Kusk Christensen
Søren Nørgreen Gustafsson
Lasse Martin Lund Würtz

Supervisor: Peter Dolog

Department of Computer Science
Aalborg University
Denmark

AALBORG UNIVERSITY
DENMARK

Summary

In this thesis, we explore the problem of generating text based on user and item information. While many recommender systems can present a predicted rating to a user about an item, the lack of explanation for the user, makes it difficult to understand the reason for the predicted rating. Some researchers have explored the use of generated reviews, which can be useful for both companies and users to enhance the experience in recommender systems. However many problems can occur when writing long sequences of text. While a review gives a thorough explanation of the sentiment, some recommender systems ask the users to provide an additional small summary of the review called tips.

We try to reconstruct the model presented in state of the art research in order to examine how different components used for text generation can have an impact on the generated tips. We first present a description of the state of the art model. Later we propose three different ways of improving the model using attention, a matrix factorization model and lastly, we examine the use of a cost scaling depending on the rating attached to the text. Due to lack of resources, we first examine different configurations of the state of the art model in order to find a configuration that yields the best performance on all presented metrics. Later we examine the performance of all three extensions.

We find that the third extension that tries to reduce the inclination towards positive reviews show poor performance on a small dataset. We find that while this extension did reduce the inclination towards positive reviews, it was at the expense of the overall performance on the used metrics.

We then trained the attention and matrix factorization extensions along with the reconstructed state-of-the-art model on a larger dataset. This yielded results that show our proposed extensions outperform in rating prediction as well as text generation compared to the reconstructed state-of-the-art model. By also examining the diversity in the generated text, we find that our extensions show greater diversity in the generated tips proving useful for more context-aware text generation.

Lastly, we reflect on reproducibility of the state-of-the-art model and baselines and consider how the facets of preprocessing, dataset representativeness and implementation choices can have a great impact on how well a model can perform.

Improving Abstractive Tips Generation for Explaining Recommendations

Michael K. Christensen, Søren N. Gustafsson, and **Lasse M. L. Würtz**
Department of Computer Science
Aalborg University, Denmark
{mkch13, sgusta13, lwartz13}@student.aau.dk

Abstract

*Tips are an additional information users can provide when reviewing products as a short explanation of their review. This extra information can be used to improve recommender systems and even provide a generated tip as an explanation of the recommendation. We look at the Neural Rating and Tips Generation (NRT) model proposed by Li et al. and attempt to reconstruct it. We then look at ways to improve this model; with matrix factorization, with attention, and through rating based scaled loss. We evaluate the base model and the extensions against baselines with MAE and RMSE for rating prediction and with ROUGE-scores for tips generation. We also evaluate the diversity of the tips generated by measuring the frequency of bigrams. We find that we are not able to reconstruct the model by Li et al. with the same configuration as they show, but with a different configuration, we are able to achieve comparable results. Our experiments show that the extension with matrix factorization named NSVDT has improved performance on rating prediction, tips generation, and diversity. The extension with attention named NRT*A also improved on diversity and tips generation but shows a slightly worse performance on rating prediction. The extension with rating based scaled loss named NRT*RSL shows a worse overall performance on rating prediction and tips generation, but by inspecting the results for every rating individually it is clear that it has a more balanced performance across the ratings.*

1 Introduction

Recommender systems have become an integrated part of many IT solutions[1]. Over the years they have become very proficient, but often they lack an explanation to go with the recommendation[2]. This explanation should be presented in a human-like manner, for example as a relevant piece of text. Presenting information to a user can be troublesome when information about that user is limited. In order to solve this problem researchers have found ways to use information about other users to infer information about a specific user using similarities and latent representation models. The relation between users can be hard to grasp, but a set of methods called collaborative filtering models have proved to be effective at predicting how a user might rate a specific item based on how this user rated other items and how this item was rated by other users.

In Wang et al.[3] they present a model that combines the use of collaborative filtering with probabilistic topic modeling. Their model constructs latent representations of users and items and it extracts topics from documents. By using these latent representations and topics it is able to provide recommendations.

While recommendations give users a way to sort through a large collection of items there is a lack of explanations as to why the user is recommended these items. Instead, explaining what the user might say about a specific item has the possibility to give a user a greater understanding of the chosen recommendation. A good explanation of the recommendation could be a short generated text presented to the user together with the recommendation. Generating user reviews can be useful for both companies and users[4]. A company could perhaps promote an item to a user who has not bought it yet through personalized recommendations. From the users' point of view, the experience is improved as the system learns more about the users. In our last paper[5], we made a study on the different techniques, metrics and state-of-the-art models used to control text generation. When you are trying to output text, which is an alteration of the ground truth text, you have a task where measuring the performance is extremely hard. This was the case in Hu et al.[6] where changing source sentences to be written in another tense or fitted to another sentiment made it difficult to evaluate since no ground truth were present. Additionally, we found that some of the generative models have a problem with generating long paragraphs that respect some features like user and item because they will lose information. Keeping track of this information in longer sequences of text have a high resource demand and will require researchers to have expensive hardware and time available for the models to perform well.

How to control the text generation to get a desired output seems to have many answers but they are all fitted to the specific use case. In Liu et al.[7] they generate Wikipedia articles using a combination of extractive summarization of source documents and a neural abstractive model to generate text following the structure of a Wikipedia article. Li et al.[8] present a model based on deep learning that uses tips and user ratings to learn latent factors between users and items.

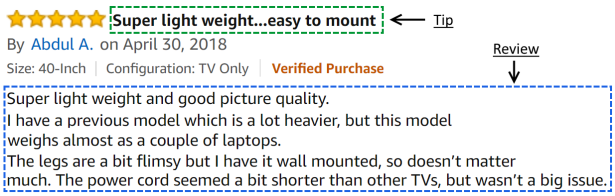

Figure 1: Tips are small pieces of information used to give a brief understanding of the opinion of an item. The tip can make it easy for a viewer to grasp the opinion of the item, without having to read a more detailed review.

A tip is a small and often single topic nugget of information, an example can be seen in fig. 1. The learned latent factors generated are then used to predict a rating and generate a tip for an item that has not yet been rated or reviewed by a user.

While there is similar research[4, 6, 9] that try to connect the relationship between the ratings given by users and the information present in the user reviews, we find the model presented by Li et al.[8] to be superior in the sense that their model achieves good statistical scores compared

to multiple baselines. Additionally, they do not attempt to generate long sequences of text but try to concentrate the core opinion into a tip.

The Neural Rating and Tips Generation (NRT) model presented by Li et al.[8] does not apply many of the components used in similar state-of-the-art models for both text generation and rating prediction. Furthermore, They address the reconstruction of ground truth and show to a small degree the sentiment of the text, but they do not address the diversity of the generated text.

Research like Dong et al.[4] shows that the addition of an attention component can have a large impact on the performance when generating novel text. The use of a similar attention component is therefore examined to review if it is applicable to the NRT model. While some research[10–12] show that review data will often be biased towards positive reviews, the research done by Li et al.[8] does not address this problem. We examine if the use of scaling in the cost function used to guide the multitask learning problem of the NRT model, inspired by Kotsiantis et al.[13], may improve the performance of the tips generation for the lower rated reviews. Lastly, we see that while Li et al.[8] address the use of Matrix Factorization (MF) models, they do not combine these fast and well-known models into their own. It is therefore interesting to examine if the use of MF for the rating prediction part of the model will provide better tips generation without a shared understanding of the user and item latent factors between the two parts.

Contributions

In this paper, we examine the model proposed by Li et al.[8] since it shows to provide state-of-the-art results regarding a combined rating prediction and tips generation. We test if it is possible for us to reconstruct their model with similar results. Additionally, we tackle the following three problems in order to try and improve tips generation and ensuring diversity in the generated tips:

1. Detaching the rating prediction component from the multitask learning goal and implementing it with matrix factorization. This model is used to pretrain the latent factors for the tips generation rather than random initialization.

2. Applying an attribute attention component to introduce more context awareness.

3. Introducing a scaling in the loss function based on ground truth ratings to counteract rating imbalance in the data set.

All these changes are built on the NRT model[8].

The rest of the paper is as follows. We describe the related work that inspired the contributions of this research. Next, we present our revision of the model proposed by Li et al.[8] We then introduce the three different extensions to the NRT model as described above. Succeeding that, we explain the research questions, baseline methods, metrics, datasets and experimental settings. Last we present the results and a discussion of the different research questioned followed by a reflection and a conclusion.

2 Related work

The use of collaborative filtering for rating prediction as part of recommender systems has been studied for numerous years[14]. Some state-of-the-art methods are based on Latent Factor Models (LFMs). Here a latent representation is learned and can be used to predict unknown relationships in the data. A group of LFMs is called Matrix Factorization[15] where a matrix is decomposed into a representation of smaller dimensionality. Many MF methods have been used in different applications, such as Probabilistic Matrix Factorization (PMF)[16], Non-Negative Matrix Factorization (NMF)[17] and SVD++[18]. For these methods, the goal is to decompose the original matrix containing ratings given by users to items into separate matrices containing user and item latent representations respectively. A user and item latent vector taken from these matrices can then be used to predict a value (a rating) in the original matrix.

While some models only use collaborative filtering to predict a rating[16–18], other models combine these with user reviews to improve the probability of correct prediction[3, 19]. This is seen in Wang et al.[3] where collaborative filtering is combined with topic modeling. This combined approach allows their model, named Collaborative Topic Regression (CTR), to construct top-n recommendations based on both rated research papers from users as well as the topic information. To construct CTR, they combine Latent Dirichlet Allocation (LDA) for topic extraction with PMF for collaborative filtering. They found that their model achieves higher average recall score regarding in-matrix prediction, and slightly better out-of-matrix prediction compared to LDA. Another advantage they found in their model is the construction of an interpretive user latent space since the topics for a specific user can explain what their interests are.

Many models have been used to summarize or generate text to better provide information to a user. Dong et al.[4] present an attribute to sequence model with an added attention component. Their model tackles the problem of generating user reviews from specified attributes such as users, items, and ratings. Their model represents attributes in a latent representation and feeds that to an LSTM network. In addition, they add an attention component to enhance the influence of the attributes to the output. They show that their model outperforms the baselines and that the attention significantly improves their model.

Li et al.[8] also presents a model for generating text, but instead of focusing on user reviews, they focus on tips, which are more concise reviews. Their model constructs a latent representation of users and items respectively. It then learns by solving a multitask learning problem with the following tasks:

1. Predict rating given to an item by a user.

2. Predict frequency of words used by the user to describe opinion about the item.

3. Generate the tip given by the user for the item.

This multitask learning problem results in a model that can predict the rating of an item in addition to generating a tip for an item given a user. They find that while collaborative filtering such as NMF shows good results their model gives a better rating prediction on the same dataset.

Li et al.[8] show that tips can be a very useful component to express user opinion and thereby provide additional

information in recommender systems besides ratings. We find that much can still be done to enhance the correlation between the generated text and the rating as well as the performance of the models compared to the training time and required resources. Collaborative filtering in the form of matrix factorization has been shown to give fairly good results with a small training time[15], and as such could be a useful tool to get faster results compared to using a neural network model.

In this paper, we present several models that are extensions of the generative model proposed by Li et al.[8]. We introduce MF and use the latent representations from this as the initial latent representation for users and items. We take this a step further and replace the rating prediction part of the model with MF. This provides us with a simpler network where there is no latent representation shared between the rating prediction and tips generation. Furthermore, we extend their model with an attention component similar to the one presented in Dong et al.[4]. Lastly, we add scaling of the loss based on the rating during training to attempt to balance the dataset. This extension is inspired by the work of Kotsiantis et al.[13] and their review of different methods for handling imbalanced datasets.

3 Proposed Model

First, we explain the original model by Li et al.[8]. Then we will present the first extension named NSVDT. Later we present the extension with attention called NRT*A. Lastly, we explain the use of scaled loss as the model NRT*RSL. Symbols used to describe the models can be seen in table 1.

Symbol	Description
$\mathcal{X}$	training set
$\mathcal{V}$	vocabulary
$\mathcal{U}$	set of users
$\mathcal{I}$	set of items
$\mathcal{S}$	set of tips
$\mathbf{C_{ctx}}$	context for tips decoder
$\mathbf{E}$	word embedding
$\mathbf{H}$	neural hidden states
$\mathbf{W}$	mapping matrix
$\mathbf{U}$	user latent factors
$\mathbf{V}$	item latent factors
$\mathbf{R}$	rating latent factors
$\mathbf{b}$	bias
$\mathbf{h}$	hidden layer
L	Number of ratings
Θ	set of neural parameters
$r_{u,i}$	rating of user u to item i
σ	sigmoid function
ς	softmax function
tanh	hyperbolic tangent function

Table 1: Glossary table showing symbols with descriptions for symbols used in this paper.

Neural Rating and Tips Generation (NRT)

The model introduced by Li et al.[8] is a model build for predicting ratings as well as abstractively generating a small tip for the item from a jointly learned latent representation of users and items. They implement this model using a combination of two Multi-Layer Perceptrons (MLPs) as well as a Gated Recurrent Unit (GRU). They mention that superscripts are used to denote which part of the network the different layers belong to. A graphical representation of the NRT model can be seen in fig. 2.

Rating Regression

To predict the rating an MLP is used. As seen in the left part of fig. 2 the MLP maps the user and item latent spaces into a hidden space and back into a real-valued rating. First, they map from the latent spaces into the hidden space:

$$\mathbf{h}^r = \sigma(\mathbf{W}^r_{uh}\mathbf{u} + \mathbf{W}^r_{vh}\mathbf{v} + \mathbf{b}^r_h) \tag{1}$$

where $\mathbf{W}^r_{uh} \in \mathbb{R}^{d \times k_u}$ and $\mathbf{W}^r_{uv} \in \mathbb{R}^{d \times k_v}$ are matrices that map the user latent vector $\mathbf{u}$ and item latent vector $\mathbf{v}$ from the latent spaces given by $\mathbf{U} \in \mathbb{R}^{k_u \times m}$ and $\mathbf{V} \in \mathbb{R}^{k_v \times n}$ for user and item respectively. Here m is the number of users and n the number of items. k_u and k_v are latent factor dimensions. $\mathbf{b}^r_h \in \mathbb{R}^d$ is the bias where d is the size of the hidden vector $\mathbf{h}^r$. $\sigma(\cdot)$ is the sigmoid activation function.

To improve performance they add additional hidden layers to their model:

$$\mathbf{h}^r_l = \sigma(\mathbf{W}^r_{hh_l}\mathbf{h}^r_{l-1} + \mathbf{b}^r_{h_l}) \tag{2}$$

where $\mathbf{W}^r_{hh_l} \in \mathbb{R}^{d \times d}$ is a mapping matrix for variables in the hidden layers. l is an index indicating which hidden layer, while $\mathbf{h}_L$ is the output of the last hidden layer. They transform $\mathbf{h}_L$ into the real-valued predicted rating:

$$\hat{r} = \mathbf{W}^r_{hr}\mathbf{h}^r_L + \mathbf{b}^r \tag{3}$$

where $\mathbf{W}^r_{hr} \in \mathbb{R}^d$ and $\mathbf{b}^r \in \mathbb{R}$.

The loss function for their regression problem is a mean squared error defined as:

$$\mathcal{L}^r = \frac{1}{2\,|\chi|} \sum_{u \in \mathcal{U},\, i \in \mathcal{I}} (\hat{r}_{u,i} - r_{u,i})^2 \tag{4}$$

where χ is the training set and $r_{u,i}$ is the ground truth rating for user-item pair (u, i).

Review Regression

Together with the rating regression, the review regression is part of a context used by the abstractive tips generation network. The review regression is a generative model based again on an MLP. To generate the review content $c_{u,i}$ they use the MLP in a similar manner as when they predict a rating. This part of the model is indicated in green in fig. 2. First they map the latent spaces of users and items into the hidden space of the network:

$$\mathbf{h}^c = \sigma(\mathbf{W}^c_{uh}\mathbf{u} + \mathbf{W}^c_{vh}\mathbf{v} + \mathbf{b}^c_h) \tag{5}$$

They also add additional layers according to the technique in eq. (2). Again $\mathbf{h}_L$ is the output of the last hidden layer. They map this to a $|\mathcal{V}|$-size vector $\hat{c}$, where $\mathcal{V}$ is the vocabulary of words in the reviews and tips:

$$\hat{c} = \varsigma(\mathbf{W}^c_{hc}\mathbf{h}^c_L + \mathbf{b}^c) \tag{6}$$

here $\mathbf{W}^c_{hc} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $\mathbf{b}^c \in \mathbb{R}^{|\mathcal{V}|}$. $\varsigma(\cdot)$ is the softmax function. They regard $\hat{c}$ as a multinomial distribution over $|\mathcal{V}|$ and they can draw words from this distribution to generate the review $c_{u,i}$.

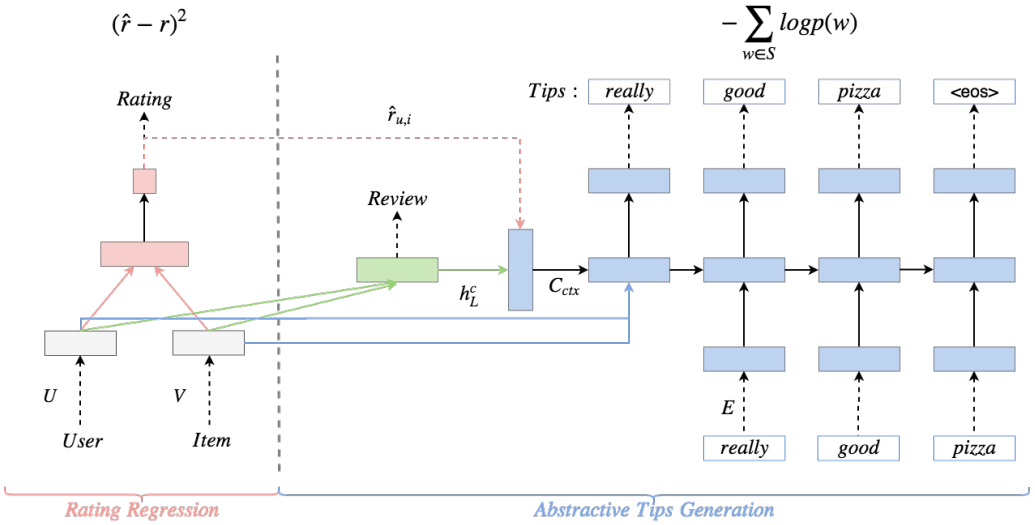

Figure 2: The NRT model by Li et al.[8] takes user and item latent factors as input and produces a rating and a tip. On the left, the latent factors are used for rating regression and on the right for tip generation. First, the rating regression and review regression are used to create C_{ctx}, which is the context for the RNN on the right. This is comprised of GRU cells that through C_{ctx} and the latent factors generate a tip.

To train this network they use the Negative Log-Likelihood (NLL) as the loss function:

$$\mathcal{L}^c = -\sum_{k=1}^{|\mathcal{V}|} \mathbf{c}^{(k)} \log \hat{c}^{(k)} \tag{7}$$

Where $\mathbf{c}$ is the ground truth and $\mathbf{c}^{(k)}$ is the frequency of the word k. So this loss function tries to minimize the NLL of having the right frequency for the right words in a review.

Abstractive Tips Generation

The abstractive tips generation is presented in the right part of fig. 3 can be seen as sequence modeling:

$$p(s_t|s_1, s_2, ..., s_{t-1}, C_{ctx}) = \varsigma(\mathbf{h}_t^s) \tag{8}$$

where s_t is the t-th word of the tip. $C_{ctx} = \{\hat{\mathbf{r}}, \mathbf{h}_L^c\}$ denotes the context generated by one hot vectorization of the output $\hat{\mathbf{r}}$ from the rating regression and the last hidden layer $\mathbf{h}_L^c$ of the review regression. $\varsigma(\cdot)$ is the softmax function. $\mathbf{h}_t^s$ denotes the hidden state of the sequence at time t. This state is dependent on the previous hidden state $\mathbf{h}_{t-1}^s$:

$$\mathbf{h}_t^s = f(\mathbf{h}_{t-1}^s, s_t) \tag{9}$$

$f(\cdot)$ is a GRU. The GRU gives the state update as:

$$\begin{aligned}
\mathbf{r}_t^s &= \sigma(\mathbf{W}_{sr}^s \mathbf{s}_t + \mathbf{W}_{hr}^s \mathbf{h}_{t-1}^s + \mathbf{b}_r^s) \\
\mathbf{z}_t^s &= \sigma(\mathbf{W}_{sz}^s \mathbf{s}_t + \mathbf{W}_{hz}^s \mathbf{h}_{t-1}^s + \mathbf{b}_z^s) \\
\mathbf{g}_t^s &= \tanh(\mathbf{W}_{sh}^s \mathbf{s}_t + \mathbf{W}_{hh}^s (\mathbf{r}_t^s \odot \mathbf{h}_{t-1}^s) + \mathbf{b}_h^s) \\
\mathbf{h}_t^s &= \mathbf{z}_t^s \odot \mathbf{h}_{t-1}^s + (1 - \mathbf{z}_t^s) \odot \mathbf{g}_t^s
\end{aligned} \tag{10}$$

where $\mathbf{s}_t \in \mathbf{E}$ is an embedding of the word s_t. $\mathbf{r}_t^s$ and $\mathbf{z}_t^s$ is the reset gate and update gate respectively. $\odot$ is the element-wise multiplication.

Because there is no input for $t = 1$ they utilize C_{ctx} to initialize $\mathbf{h}_0^s$:

$$\mathbf{h}_0^s = \tanh(\mathbf{W}_{uh}^s \mathbf{u} + \mathbf{W}_{vh}^s \mathbf{v} + \mathbf{W}_{rh}^s \hat{\mathbf{r}} + \mathbf{W}_{ch}^s \mathbf{h}_L^c + \mathbf{b}_c^s) \tag{11}$$

Here $\mathbf{W}_{rh}^s \in \mathbb{R}^{d \times L}$ and $\mathbf{W}_{ch}^s \in \mathbb{R}^{d \times d}$. L is the number of different ratings. The context C_{ctx} is what allows the tips generation network to write tips that are relevant for the user and item pairs.

When all sequence hidden states are found, they feed them through the output layer:

$$\hat{\mathbf{s}}_{t+1} = \varsigma(\mathbf{W}_{hs}^s \mathbf{h}_t^s + \mathbf{b}^s) \tag{12}$$

where $\mathbf{W}_{hs}^s \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $\mathbf{b}^s \in \mathbb{R}^{|\mathcal{V}|}$.

During training the word with the largest probability is the decoded result for time step $t + 1$:

$$w_{t+1}^* = \underset{w_i \in \mathcal{V}}{\operatorname{argmax}} \, \hat{\mathbf{s}}_{t+1}^{(w_i)} \tag{13}$$

The loss function is the NLL:

$$\mathcal{L}^s = -\sum_{w \in Tips} \log \hat{\mathbf{s}}^{(I_w)} \tag{14}$$

where we try to the make sure that the index I_w to the ground truth word w is the most likely in $\hat{\mathbf{s}}$ at the given time step. The sequence s^* with the best log-likehood is found with the following equation:

$$s^* = \underset{s \in \mathcal{S}}{\operatorname{argmax}} \sum_{w \in s} \log \hat{\mathbf{s}}^{(I_w)} \tag{15}$$

When testing they apply beam search. Here we have to notice that they are not exploring the full search space of all possible sequences but instead beam search will limit the search frontier to a certain beam size, making it likely to find the optimal sequence but not guaranteed. We apply length normalization to the beam search like in Li et al.[8].

Multitask Learning

Li et al.[8] learn the user and item latent features jointly from the different networks as well as unifying the learning of the subtasks of rating, review and tips generation in a multi-task learning problem where the objective function is given by:

$$\mathcal{J} = \min_{U,V,E,\Theta} (\lambda_r \mathcal{L}^r + \lambda_c \mathcal{L}^c + \lambda_s \mathcal{L}^s + \lambda_n (\|U\|_2^2 + \|V\|_2^2 + \|\Theta\|_2^2)) \tag{16}$$

$\mathcal{L}^r$ is the loss function of the rating regression network defined in eq. (4), $\mathcal{L}^c$ is the loss function of the review generation network define in eq. (7) and $\mathcal{L}^s$ is the loss function from the tips generation network define in eq. (14). Θ is the set of neural parameters, meaning all the weights of the rating, review and tips generation networks. U and V refer to the user and item latent features. λ_r, λ_c, λ_s are weight proportions that serve to weight different parts of the network differently. λ_n controls the weight of the L2 regularization term.

Non-Negative Singular Value Decomposition and Tips Generation (NSVDT)

In this extension to NRT we replace the MLP used for rating prediction with Singular Value Decomposition (SVD). We call this model NSVDT. In fig. 3 we have a matrix factorization problem on the left and an abstractive tips generation problem on the right. We have replaced the rating regressions of the NRT model with an SVD component that is used for rating prediction. One of the big differences is the latent user and item factors (U, V) of the tips generation model that are pretrained and copied from the feature matrices (W, H). This means that the SVD model keeps (W, H) stable and does not need backpropagation, denoted by the double-crossed line. The review and tips generation network will do their backpropagation on (U, V), which are no longer randomly initialized but pretrained by the SVD.

We use a gradient descent based algorithm to solve the SVD as well as adding a non-negativity constraint to the algorithm. This algorithm was originally popularized by Simon Funk[20] in the Netflix competition [21]. It works by minimizing the sum of squared errors against only known ratings. Where the update for latent variables for users $\mathbf{u} \in \mathbf{U}$ and items $\mathbf{v} \in \mathbf{V}$ is,

$$\begin{aligned} \mathbf{u} &\leftarrow \mathbf{u} - lr\,\frac{d(\hat{r}_{u,i} - r_{u,i})^2}{d\mathbf{u}} = \mathbf{u} + lr * 2(\hat{r}_{u,i} - r_{u,i})^2 * \mathbf{v} \\ \mathbf{v} &\leftarrow \mathbf{v} - lr\,\frac{d(\hat{r}_{u,i} - r_{u,i})^2}{d\mathbf{v}} = \mathbf{v} + lr * 2(\hat{r}_{u,i} - r_{u,i})^2 * \mathbf{u} \end{aligned} \tag{17}$$

Where lr is the learning rate and d denotes partial derivative. This is done for each user and item pair $(u \in \mathcal{U}, i \in \mathcal{I})$ and rating $r_{u,i}$ in the set of known ratings from users to items.

We use this kind of gradient descent based approach for updating our SVD model because it allows us to consider and hold only the known user and item pairs and their associated rating in memory. The loss function is the Sum of Squared Errors (SSE) of reconstructing the known ratings. We add the non-negativity constraint to the loss function as,

$$constraint = 10\text{e}{+12}\left(\left(\sum_{h}^{|H|} |h| - h\right) + \left(\sum_{w}^{|W|} |w| - w\right)\right) \tag{18}$$

If all latent variables are non-negative the constraint is 0, otherwise we add an extremely large penalty to the loss. The optimization algorithm used for updating is the Adam[22] algorithm. More formally the update for a latent variable lv with gradient g becomes,

$$\begin{aligned} &\forall lv \in \mathbf{U} \cup \mathbf{V}: \\ &lr_t \leftarrow lr\frac{\sqrt{1-\beta_1^{t+1}}}{1-\beta_1^{t+1}} \\ &m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g \\ &v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g^2 \\ &lv \leftarrow lv - lr_t\frac{m_t}{\sqrt{v_t}+\epsilon} \end{aligned} \tag{19}$$

m_0 and v_0 are initialized as zeros and gradient g is calculated from the SSE like in eq. (17). lr_t is the learning rate for the current time step calculated from the hyperparameter lr. β_1 and β_2 are hyperparameters.

Neural Rating and Tips Generation with Attention (NRT*A)

In the NRT model the latent factors are used as inputs to the network but in the GRU and in the MLP it is possible that the information gained from the latent factors vanish in between the layers. Therefore we wanted to extend the model with the attribute attention introduced in Dong et al.[4]. This kind of attention revolves around applying the latent knowledge to all the outputs of the GRU and thereby minimize the vanishing of their importance. This kind of attention is illustrated in fig. 4 and we will briefly explain the different calculations that are added in the network. In fig. 4 the NRT C_{ctx} box is the part of the NRT network that constructs the context C_{ctx} from the user and item latent factors. This part consists of rating regression and review regression.

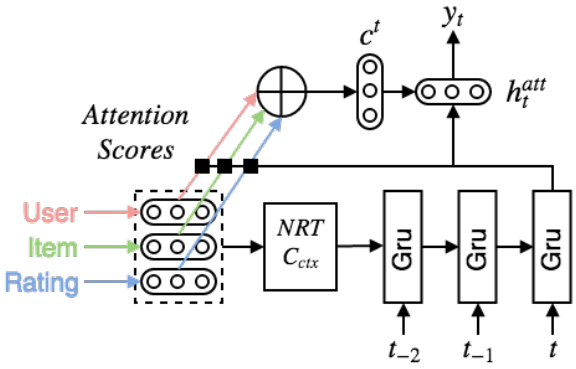

Figure 4: Illustration of attribute attention inspired by Dong et al.[4]. We retrieve C_{ctx} from NRT as normal. For user, item and rating an attention score is calculated. This score and the hidden output is used to calculate c^t. c^t and the hidden output is used to find $\mathbf{h}_t^{att}$, which is used to find the output.

On the left, in fig. 4 we see that we calculate attention scores by concatenating the latent factors with the hidden layers from the GRU and then multiply them with a weight matrix. The attention score for the attributes are calculated as,

$$s_i^t = \exp(\tanh(\mathbf{W}_{hsc}^a[\mathbf{h}_t^s, a_i]))/Z \tag{20}$$

where $[\cdot,\cdot]$ denotes concatenation, $\mathbf{h}_t^s$ is the hidden output from the GRU at time t, a_i represents attributes $u_i \in U$, $v_i \in V$ and a new latent representation of rating $r_i \in R^{k_r \times L}$, where k_r is the rating latent size and L is the number of different ratings. Z is a normalization term that ensures

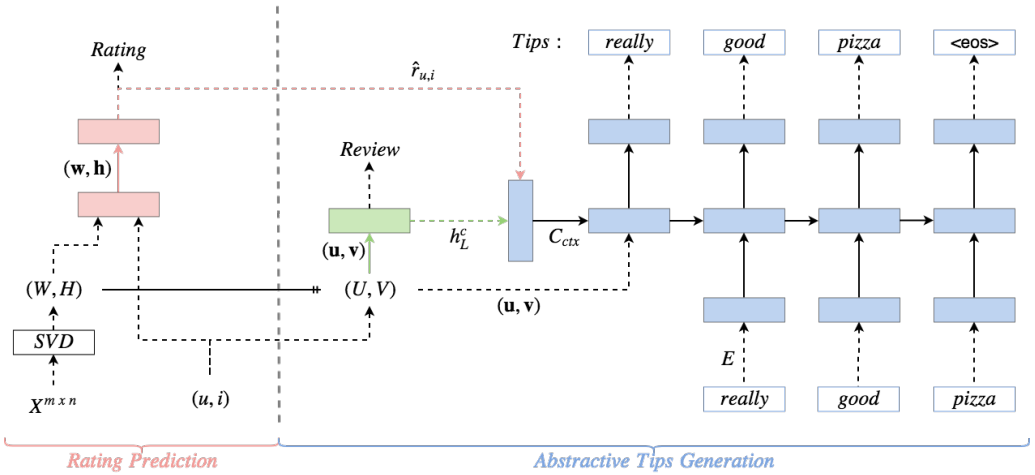

Figure 3: NSVDT for rating prediction and abstractive tips generation. We train the SVD by using the user-item-rating matrix X. This provides us with a latent representation **W** *and* **H** *that is used to predict the rating for a given user-item pair.* **W** *and* **H** *are copied into* **U** *and* **V** *in the tips generation part of the model. Each user and item pair* $(\mathbf{u}, \mathbf{v})$ *from* **U** *and* **V** *are used to predict frequencies of words in the corresponding review. These outputs are used to generate the tip.*

$\sum_{i=1}^{|a|} s_i^t = 1$. $\mathbf{W}_{hsc}^a \in \mathbb{R}^{1\times(d+k_i)}$ is a mapping matrix where k_i is the latent size of attribute i. The attention scores are used to create a weighted sum of the latent attribute vectors. We call this the context vector $\mathbf{c}^t$,

$$\mathbf{c}^t = \sum_{i=1}^{|a|} s_i^t a_i \tag{21}$$

The final hidden layer for the attention $\mathbf{h}_t^{att}$ is found by a combination of the hidden layer and this context vector,

$$\mathbf{h}_t^{att} = \tanh(\mathbf{W}_{ch}^a \mathbf{c}^t + \mathbf{W}_{hh}^a \mathbf{h}_t^s + \mathbf{b}^a) \tag{22}$$

where $\mathbf{W}_{ch}^a \in \mathbb{R}^{d\times k_i}$ and $\mathbf{W}_{hh}^a \in \mathbb{R}^{d\times d}$. The hidden layer $\mathbf{h}_t^{att}$ is used to predict the token s_t,

$$p(s_t|s_t|s_1, s_2, ..., s_{t-1}, a) = \varsigma(\mathbf{W}_{hs}^a \mathbf{h}_t^{att} + \mathbf{b}^{att}), \tag{23}$$

where $\varsigma(\cdot)$ is the softmax function and $\mathbf{W}_{hs}^a \in \mathbb{R}^{|V|\times d}$ is a mapping matrix that maps from the hidden space to the vocabulary, much like in eq. (12).

Neural Rating and Tips Generation with Rating Scaled Loss (NRT*RSL)

An inherent characteristic of most review datasets is the distribution of ratings which is highly imbalanced with a large number of positive ratings compared to negative ones. As can be seen in fig. 5 this is the case for our datasets as well. For the NRT model, this means that it will see positive review and tip text much more often than negative ones. This means that in general the loss throughout the training will be influenced a lot more by positive sentiment text and the model's statistical knowledge will come mostly from these. Inspired by Kotsiantis et al.[13] we have come up with a counteractive measure that scales the loss based on whether it comes from a positive sentiment or negative sentiment review. This means that the model will be punished much harder for having errors in the reconstruction of negative tips. Scaling the loss based on the ground truth rating is done by extending eq. (14) to the following,

$$\mathcal{L}^s = -\sum_{w\in Tips} \mathbf{W}_{sc}^{(I_r)} \log \hat{\mathbf{s}}^{(I_w)} \tag{24}$$

Where the new addition $\mathbf{W}_{sc}^{(I_r)} \in \mathbb{R}^L$ is a weight matrix containing the scales and I_r is an index to the scaling matrix for the ground truth rating. We call this extension of the NRT model for NRT*RSL.

4 Experimentation

Research Questions

In this paper we will assess the following research questions:

- **RQ1:** Can we reconstruct the work of Li et al.[8]?
- **RQ2:** How will the integration of a matrix factorization component for rating prediction and pretraining of the latent factors affect the performance of the model?
- **RQ3:** How will the addition of attention affect the performance of the model?
- **RQ4:** Can we make the tips generation correlate more with the ground truth rating prediction by the addition of rating scaling?

Comparative Methods

In this section we will give an overview of the comparative methods used to compare the performance of our model against well-known methods. For the evaluation of the rating prediction we compare to the following methods:

NMF - Non-Negative Matrix Factorization[17] is a decomposition algorithm that ensures non-negative user and item features and takes only the rating matrix as input. It takes the rating values for already rated items in a sparse user-item matrix and learns the missing rating values using decomposition.

SVD++ - Singular Value Decomposition++[20] is an extension of SVD where binary implicit feedback about whether a user rated an item or not is taken into consideration for latent factor modeling.

URP - User Rating Profile[19] takes the rating matrix as input and models users as a mixture of attitudes. A full user profile is created from the relationship between a users attitude towards an item and the preference pattern for that attitude.

CTR - Collaborative Topic Regression[23] is a model used for recommending scientific papers. It combines collaborative filtering with probabilistic topic modeling. It uses the topic distribution of the words used about an item and the user latent features to predict the rating a user would give that item.

For the evaluation of the tip generation we compare to the following methods:

LexRank[24] is a well-known method for summarizing text. It works by ranking sentences in a text corpus based on a similarity graph. The similarity between nodes is calculated as a idf-modified-cosine formula, which measure the distance between two sentences. Li et al.[8] creates a summarization problem by creating a set of sentences $C_{u,i}$ for each user and item pair in the test dataset. $C_{u,i}$ is created by:

1. Retrieving C_u as the set of reviews written by user u and C_i as the set of reviews written about item i. These reviews are retrieved from the train dataset.
2. Filtering C_u and C_i by removing any reviews that do no match the ground truth rating $r_{u,i}$, which is the rating given by user u to item i. These values come from the test dataset.
3. All reviews from C_u and C_i whose words only appear in one set are removed.
4. Extract the tip by merging C_u and C_i into $C_{u,i}$ and extract the top ranked sentence from $C_{u,i}$ given by LexRank.

They note that this gives LexRank an advantage compared to the other methods, as it uses the ground truth ratings to generate tips. We use this method for our LexRank as well.

CTR-t - The CTR model contains a topic model that can be sampled to get the topics for an item. This topic model is used to generate tips. The generation can be described as:

1. Get the latent topic representation of an item i and draw the topic z with the highest probability.
2. Select top 50 words from ϕ_z which is a multinomial distribution of the topic z on the vocabulary $|V|$. Giving us the 50 most likely words given a topic.
3. The sentence most similar to the top 50 words from $C_{u,i}$ is then extracted as the tip. Similarity is based on bag-of-words overlap.

Datasets

For our experiments, we use a benchmark dataset from Amazon, namely the Electronics 5-core dataset[25]. We will call this dataset Electronics-Full. In Li et al.[8] they use several other datasets, but due to hardware limitations, we are unable to run the model on these datasets. The dataset is a 5-core dataset which means that all users have written at least 5 reviews and there is at least 5 reviews written for each items.

From the dataset, we extract the following for each sample: user ID, item ID, review, rating, and tip. The user and item ID's are integers where each user and item will have a unique ID. The rating is an integer in the range [1,5]. The review and tip are texts. The vocabulary V is built from all unique words with a term frequency of 20 or above.

	Full	**200k**
users	192,403	97,809
items	63,001	8,381
reviews	1,684,779	199,576
$\|V\|$	69,138	18,971

Table 2: Overview of the datasets. We see that the number of users far exceeds the number of items for both datasets, but the ratios are not the same.

For parameter tuning and testing we developed a smaller dataset, Electronics-200K, which is a subset of the Electronics-Full dataset and is created by taking the first 200k reviews from the full dataset. As the Electronics-Full dataset is sorted by items this ensures that we maintain 5-core on the items. We can however not ensure that the same is true for users.

Figure 5: Distribution of the reviews based on rating. It is clear that both datasets are very similar in distribution of rating. Additionally, the figure shows the large proportion of positive reviews compared to negative.

Information about the number of *users*, *items*, *reviews*, and the size of the vocab can be found in table 2. It is clear that we are not able to maintain the same proportion of users and items for the smaller dataset.
An important feature of these datasets is the distribution of the ratings. If the distribution of ratings differs between the datasets, it will not be representative. As seen in fig. 5 the rating distribution are very similar for all three datasets with only a small variation for each rating. Additionally, we note that a large proportion of the reviews are positively rated.
The frequencies in the Electronics-200K dataset matches the ones of Electronics-Full in fig. 6. Electronics-200K has a

significantly smaller amount of bigrams, but this behavior is expected from lowering the amount of data. They still share the same distribution and the top bigrams are all the same bar a few changes, see table 3. Given the attributes for the smaller dataset, we deem the Electronics-200K to be representative of the Electronics-Full dataset, and we will use this dataset for further evaluation.

Ranking	Electronics-200K	Electronics-Full
1	(for, the)	(for, the)
2	(this, is)	(this, is)
3	(for, my)	(for, my)
4	(i, have)	(i, have)
5	(i, bought)	(i, bought)
6	(is, a)	(is, a)
7	(for, a)	(for, a)
8	(price, <eos>)	(price, <eos>)
9	(<unk>, <eos>)	(bought, this)
10	(bought, this)	(the, price)
11	(the, price)	(of, the)
12	(of, the)	(it, <eos>)
13	(it, <eos>)	(it, is)
14	(it, is)	(a, great)
15	(a, great)	(and, it)

Table 3: Comparison of top bigrams on Electronics-Full and Electronics-200K.

Preprocessing the Data

In this section we will look at how we can go from the raw data in the datasets to the information we are interested in using. In the datasets users and items have a unique ID. We provide every user and item with a new ID when processing the dataset, such that we ensure that the first user has $u_{ID} = 0$ and every following user will have an incremented ID. We do exactly the same for items. We do this, so we can use the IDs for indexing. The rating is extracted from the dataset as an integer in the range [1,5]. For the tip we use the summary text. The remaining preprocessing is different for the various models and will be described in the following sections.

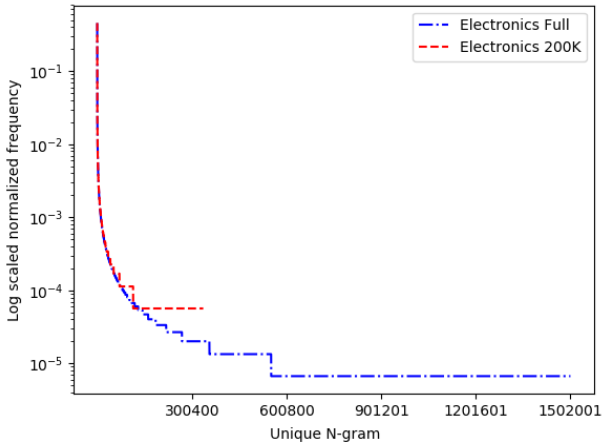

Figure 6: Bigram frequencies for Electronics-200K and Electronics-Full plotted in descending order. The x-axis is an index denoting the ranking of the bigrams. The y-axis shows the normalized frequency of a bigram on a logarithmic scale. We see that both datasets share a similar regression in frequencies.

NRT & Extensions

For NRT and our extensions we process the data following these steps:

1. For all samples with a summary of less than five words, use the first sentence from the review text, with five or more words. If no sentence exists, we remove this sample from the dataset.

2. We remove all punctuation from the reviews and tips. We do this to avoid that the model sees words like "today!" with an exclamation mark as different from the word "today", which is not desirable both due to an explosion in vocabulary size and because there is little difference in the meaning of the two.

3. We reduce the length of the tips to be no larger than 20.

4. We append an end-of-string <eos> token to each tip, this is so that the recurrent network in the tips generation part will learn the ending token for all tips.

5. We remove stopwords from all the reviews. Stopwords are words that does not contain much meaning, and due to their frequent appearance will make it harder for the model to learn the latent information in the review texts.

6. We add all words that appear more than 20 times to a vocabulary. All words that appear less than 20 times in the entire set will be replaced with an unknown <unk> token. This is done to avoid an unnecessary vocabulary size and to not carry meaningless or misspelled words into the generative model.

7. The output is a file containing the processed review information (userID, itemID, rating, review, and tip).

From Li et al.[8] we received a preprocessing file outlining that they did similar for their research of the NRT model.

URP, NMF & SVD++

For all the collaborative filtering we use the same processed review information as for NRT and our extensions. This means that if a sample has been discarded in the preprocessing it is not used here either. These models are implemented using LibRec library[26] that assumes the data to be formatted as (user, item, rating). Therefore, we output the data following this structure such that it can be loaded properly by the LibRec library.

LexRank & CTR

For LexRank and CTR we make some small changes to the preprocessing to better fit to the models and their implementations. The first difference is that we find the individual sentences in each review before removing punctuation. This is done since the LexRank model use the individual sentences for the similarity graph. Furthermore, we do not append an <eos> token to the tips since none of the models need it. Lastly we do not replace infrequent words with an <unk> token. We choose to not do this since we figure that the <unk> token will have a negative impact on the semantic graph connecting entire sentences based on only a single word namely the <unk> token.

	RMSE	MAE	ROUGE-1	ROUGE-2	ROUGE-l	ROUGE-SU4
NRT	[2]1.19	0.93	0.01	0.00	0.00	0.00
NRT′	1.27	0.91	8.85	0.01	3.64	2.42
NRT″	1.29	[2]0.87	**[1]16.41**	**[1]2.01**	[2]10.59	**[1]4.57**
NRT*	1.43	1.03	14.96	1.69	10.02	4.09
NRT*A	1.32	0.90	15.25	1.77	10.16	4.18
NSVDT	**[1]1.15**	**[1]0.85**	[2]15.72	[2]1.95	**[1]10.62**	[2]4.36
NRT*RSL	1.31	0.93	13.78	1.37	8.49	3.76

Table 4: Results on the Electronics-200K dataset. All ROUGE values in the table are F1-scores. Superscript denotes best and second best in each column. We see that the NSVDT has the best performance on both RMSE and MAE, and achieves the second best ROUGE scores on all metrics. Additionally we see that the normal NRT configuration presented by Li et al.[8] is not able to get competitive ROUGE scores, but still achieves high rating prediction scores.

Evaluation Metrics

This section introduces different measures that capture different aspects of the models. We want to measure the performance of rating prediction, tips generation, and diversity of the generated text. In the following sections, we will show the metrics used for each of these.

Rating Prediction Metrics

To evaluate the performance of the rating prediction, we will use Root Mean Square Error (RMSE) and Mean Absolute Error (MAE)[8, 27]. Given a predicted rating $\hat{r}_{u,i}$ for user u to item i and a ground-truth rating $r_{u,i}$, RMSE is calculated in the following way:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(r_{u,i} - \hat{r}_{u,i})^2} \tag{25}$$

where N is the number of ratings between users and items. The strength of RMSE is that it is good at describing errors following a normal distribution[27]. RMSE Shows how much the data is concentrated or spread out from the correct value. Because the error is squared before it is averaged it gives a high weight to large errors. With the same information MAE can be calculated in the following way:

$$MAE = \frac{1}{N}\sum_{u,i}\left|r_{u,i} - \hat{r}_{u,i}\right| \tag{26}$$

The MAE is a good metric to evaluate average model performance and is good when used to describe uniformly distributed errors because it does not penalize large errors like RMSE[27]. In MAE it is simply twice as bad being off by 4 as being off by 2, but in RMSE it is more than twice as bad.

The choice of evaluating rating prediction using both MAE and RMSE is due to the convenience of being able to easily compare our model to the original NRT model[8].

ROUGE Measures

To evaluate the performance of the tip generation we will use the following scores from ROUGE[28]; ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-SU4.

ROUGE-1 and **ROUGE-2** are both versions of ROUGE-N, which is an n-gram recall between a generated tip and a set of reference tips. In our case we always have only one reference, which gives us the equation shown below:

$$ROUGE\text{-}N = \frac{\sum_{gram_n} Count_{match}(gram_n)}{\sum_{gram_n} Count(gram_n)} \tag{27}$$

where n is the size of the n-gram. $Count_{match}(gram_n)$ is the number of n-grams co-occurring in the generated tip and in the reference tip. $Count(gram_n)$ refers to the total number of n-grams in the reference or the generated tip depending on whether we are calculating recall or precision respectively.

ROUGE-L score is a measure of the longest common sub-sequence between the generated tip and the reference tip. The longer the longest common sub-sequence is, the higher the similarity will be between two tips. Having a ROUGE-L score of zero means that there is no sub-sequence in common between two tips.

ROUGE-SU4 is a measure where we count unigrams and skip-bigrams with a skip size of 4. A skip-bigram is a bigram where you allow n wrong words to be between the words in the bigram you are looking for. This measure is more forgiving when it comes to matching bigrams compared to ROUGE-2. Additionally this metric rewards for also having single words co-occurring in two tips. This can handle situations, where a single word may be correct, but the structure of the tips limit the co-occurrence of bigrams in the two tips.

The ROUGE scores are measured in precision and recall. Precision measure how much of the generated tip is relevant. Recall measures how much of the reference tip was captured by the generated tip. These measures can be combined in a F1-score[29], which is a harmonic mean over precision and recall. We use an implementation of ROUGE in Python created by Tardy[30]. This implementation comes with a disclaimer that the results may differ slightly from the original ROUGE implementation.

Diversity Measure

Explaining a recommendation to a user as a piece of text becomes interesting when we are able to diversify the text. If we merely wanted some text that matches the rating, we could have a fixed sentence for items with a high predicted rating like: *"This is a great product for the price"* and perhaps: *"A waste of time and money"* for an item with a low rating.

To measure the diversity in the generated tips we introduce a diversity measure based on bigram occurrence. We find the different bigrams in the predicted tips and measure their normalized frequency. This is plotted to visually display the distribution of bigrams. This measure should help us see patterns of whether the model is producing

diversified outputs for the different user and item pairs and their predicted rating or if the model learns a limited number of bigrams that are repeated. A very high frequency of the most used bigrams together with a low amount of total bigrams should help us identify a low diversity while a more flat curve show a higher diversity in the predicted tips.

Parameter Tuning & Configuration

In this section, we will examine the preliminary results used for finding good parameters for the models. We will then explore gradient descent algorithms, parameter tuning, and regularization.

Gradient Descent Algorithm

Li et al.[8] uses the AdaDelta[31] gradient descent algorithm in their NRT model. This algorithm is quite computationally heavy because of the way learning rate, decay and momentum are calculated. However, it has shown good results on Recurrent Neural Networks (RNNs) as reported in Li[32]. In our preliminary results we were unable to achieve any useful results using AdaDelta, and as such we investigated other optimization algorithms. To get a noticeable improvement in computation time we have chosen to use another algorithm reported in Li[32], namely the Adam algorithm[22]. This algorithm has shown good performance on MLPs and it is a much simpler algorithm to run. In our testing on the smaller dataset the Adam algorithm showed on par performance with AdaDelta, but the time it takes to compute the gradients for an epoch is less than half the time used by AdaDelta.

Parameter Tuning

Parameter tuning was done on the Electronics-200K dataset and evaluated on the development set. We mainly focused on finding a good optimizer and a good value for λ_n, the scaling of the regularization term. We evaluated a number of different configurations which is shown in table 5. NRT is the setup that was used by Li et al.[8]. NRT′ and NRT″ are setups where the Adam optimizer is tested with two different values for λ_n. NRT* is a setup where the regularization is removed by setting $\lambda_n = 0$. All the extensions (NRT*A, NSVDT, NRT*RSL) are built on top of the NRT* version.

Name	Optim.	λ_n
NRT	AdaDelta	1e−4
NRT′	Adam	1e−4
NRT″	Adam	1e−8
NRT*	Adam	0
NRT*A	Adam	0
NSVDT	Adam	0
NRT*RSL	Adam	0

Table 5: Overview of model configurations on the Electronics-200K dataset. We see that the choice of the Adam optimizer along with the exclusion of regularization has been used on the extensions to the NRT model.

Regularization

As shown in eq. (16) there is a regularization term as part of the objective function, but during preliminary experimentation, we noticed that the regularization proposed by Li et al.[8] had a very bad impact on the results of the model. This can be seen in table 4 where NRT is the same configuration as Li et al.[8] used. The configurations of the models can be seen in table 5. We then experimented with various values for λ_n to adjust the impact the regularization had on the model. An example of this can again be seen in table 4 where the only difference between NRT′ and NRT″ is the impact of the regularization and as we can see, the model performs better when λ_n gets smaller.

Experimental Settings

The dataset is shuffled with a random seed of 1 and then split into three parts; train, development, and test with ratio 8:1:1. The latent factors for NMF and SVD++ is set to 10. The number of iterations for NMF is 10 and for SVD++ it is 200.

For NRT and our extensions, we set user, item and rating latent factors to K=300. We set the size of all hidden layers to 400 and the size of the word embedding to 300. The learning rate is set to 1e−4 The beam size is set to $\beta = 4$ and the maximum tip length is set to 20 (excluding <eos> token). The weight parameters are set to $\lambda_c = \lambda_r = \lambda_s = 1$ and $\lambda_n = 0$. NRT and our extensions are trained for 20 epochs with a batch size of 30.

For NSVDT the SVD is trained for 17 iterations with a learning rate of 1e−3. The initial values in the SVD is a uniform distribution with values in the range $[0, 0.23]$.

For NRT*RSL the scaling of the loss for the ratings was done with the following vector $\langle 3.33, 4.00, 2.20, 1.00, 0.33 \rangle$.

Implementation

We have implemented the NRT model as well as our extensions in Tensorflow 1.5.0[33]. LexRank uses LexRank[34] and CTR uses CollaborativeTopicModel[35] and both are implemented in Python 3.5. NMF, SVD++, and URP use LibRec[26], and are implemented in Java. LexRank and CTR was run on an Intel Xeon E5-2690 at 2.60GHz with 378GB system memory. NMF, SVD++ and URP was run on another server with an Intel Xeon E5-2680 v3 at 2.50GHz with 32GB memory. The various NRT models was run on an Intel i7 880 2.93GHz with 24GB system memory and an Nvidia GTX 1060 with 6GB memory.

5 Results & Discussions

In this section, we will look at the results from our experiments and discuss these. Results for the NRT model and extensions can be found at [36]. First, we will present the overall results for the various models. Then we will look at the different research questions and present additional results relevant only to that question. For each of the research question we will end with a discussion of the results. It should be noted that we chose not to run NRT*RSL on Electronics-Full due to resource limitations and because it showed poor performance regarding text generation on Electronics-200K as shown in table 4. First we will take a look at the performance of the rating prediction for the various models and the appropriate baseline methods. NSVDT has the best performance on both RMSE and MAE compared to the other methods which can be seen in table 7. We also note that besides NSVDT the other models are outperformed by the baseline methods, where URP has

	ROUGE-1			ROUGE-2			ROUGE-L			ROUGE-SU4		
	F1	P	R	F1	P	R	F1	P	R	F1	P	R
LexRank	8.78	8.25	13.60	0.56	0.54	0.99	5.87	6.73	11.32	2.15	2.24	4.04
CTR	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
NRT*	15.63	13.65	20.87	1.76	1.49	2.57	10.43	10.07	17.29	4.24	3.55	6.91
NRT*A	^{1}**16.01**	^{2}13.91	^{1}**21.61**	^{2}1.79	^{2}1.49	^{2}2.69	^{2}10.64	^{2}10.22	^{1}**17.91**	^{2}4.35	^{2}3.60	^{1}**7.21**
NSVDT	^{2}15.99	^{1}**14.22**	^{2}20.99	^{1}**1.95**	^{1}**1.68**	^{1}**2.80**	^{1}**10.75**	^{1}**10.49**	^{2}17.57	^{1}**4.45**	^{1}**3.81**	^{2}7.10

Table 6: ROUGE-scores on Electronics-Full. NSVDT is far superior compared to NRT on all ROUGE scores, but in some cases beaten by NRT*A. NRT*A extension outperforms NRT*. CTR was implemented, but did not finish in time.*

the second best performance on RMSE and SVD++ has the second best performance on MAE.

	RMSE	MAE
NMF	1.226	0.867
SVD++	1.183	^{2}0.845
URP	^{2}1.126	0.857
CTR	N/A	N/A
NRT*	1.414	0.960
NRT*A	1.464	0.995
NSVDT	^{1}**1.116**	^{1}**0.805**

Table 7: Rating prediction measured with MAE and RMSE on Electronics-Full. While Li et al.[8] outperformed the presented baselines on rating prediction, we were not able to achieve the same conclusion with NRT. NSVDT outperforms all baselines and extensions.*

If we look at the performance for tips generation we can see in table 6 that NSVDT is the overall best performer followed by NRT*A. As we can see all the models outperforms the baseline methods on tips generation. Note that there are no results for CTR as the model did not complete in time, and as such the results are missing.

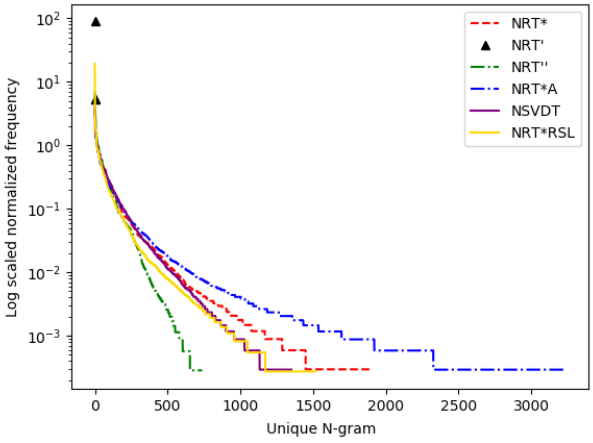

Figure 7: Bigram frequencies in descending rank for configurations in 4 on Electronics-200K. The x-axis is an index denoting the ranking of the bigrams. The y-axis shows the normalized frequency of a bigram on a logarithmic scale. NRT is not present in the figure since it produced no bigrams. NRT' is presented as dots since it had only three unique bigrams for that model.

We also looked at the diversity of the different models, which can be seen in fig. 8 for the results on Electronics-Full and in fig. 7 for the results on Electronics-200K. These figures show normalized frequencies on a logarithmic scale on the y-axis and an identifier on the x-axis that ranks the bigrams by frequency. We are not interested in the bigrams themselves but rather the trend in frequency.

In fig. 7 we see how diverse the generated tips are for each configuration when trained on the Electronics-200K dataset. Notice that NRT is not included in this figure since it did not generate any bigrams but simply one-word tips. Additionally, we see that also NRT' and NRT'' did not perform well, with NRT' only generating three unique bigrams, and therefore plotted as dots instead of lines, while NRT'' contain very low bigram diversity compared to other configurations and extensions. We can see in fig. 7 that NRT* achieved a good bigram diversity compared to many other configurations and only NRT*A writes more diverse.

In fig. 8 we see the results on Electronics-Full where both NRT*A and NSVDT are writing more diverse tips compared to the base model NRT* on Electronics-Full, where only the NRT*A was better on Electronics-200K.

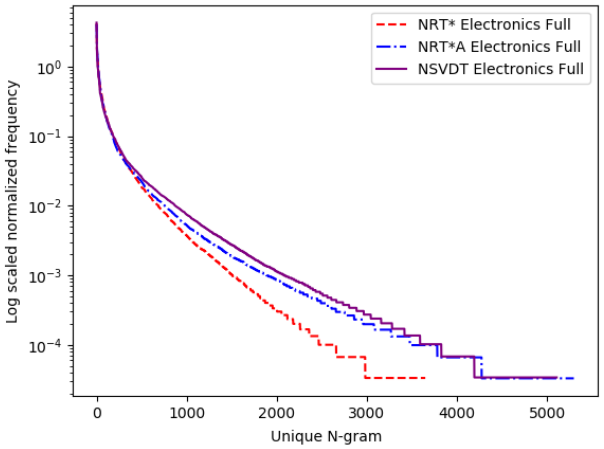

Figure 8: Bigram frequencies in descending rank for NRT, NRT*A and NSVDT on Electronics-Full. The x-axis is an index denoting the ranking of the bigrams. The y-axis shows the normalized frequency of a bigram on a logarithmic scale. Diversity is higher for both NRT*A and NSVDT compared to NRT*. Both extensions contains a higher number of unique bigrams compared to NRT*.*

Reconstruction of the NRT Model (RQ1)

The results presented by Li et al.[8] for their model NRT is seen in table 8 as NRTo. For ROUGE we show only the F-1 scores. Additionally we see that our own implementation of NRT on Electronics-200K did not provide any acceptable ROUGE scores, however, we did get comparable RMSE and MAE scores. By changing the optimizer of the network from Adadelta to Adam, and removing the regu-

	RMSE	MAE	ROUGE-1	ROUGE-2	ROUGE-l	ROUGE-SU4
NRTo (Li et al.[8])	1.107	0.806	13.95	2.72	12.67	4.68
NRT	1.189	0.927	0.01	0.00	0.00	0.00
NRT″	1.287	0.869	16.41	2.01	10.59	4.57
NRT*	1.414	0.960	15.63	1.76	10.43	4.24

Table 8: Table showing RMSE, MAE and F1-scores on ROUGE for our configurations of the NRT model and NRTo is results as in Li et al[8]. NRTo achieved better results than our implementations, but our configurations NRT″ and NRT get comparable results.*

larization term, we find that we can achieve comparable results on Electronics-Full between NRT* and NRTo and also between NRT″ and NRTo, but with NRTo being better in both comparisons on almost all metrics.

From these results, we see the reconstruction challenge of the NRT model by Li et al.[8] can be seen from two viewpoints:

1. Reconstruction of their model with similar results using their experimental settings.
2. Reconstruction of their model with similar results using our experimental settings.

5	**"perfect for office server and photoshop use"**
4.41	"i bought this for my wife and it has been very happy with it"
5	**"i consider myself an audiophile i have a huge system that ive built over the years but its so big"**
4.22	"i bought this for my wife and it has been very happy with it"
4	**"i have a maze of cables behind my living room tv and i wanted to clean it up"**
4.81	"i bought this for my laptop and it worked great for me and it works great and it works great"
5	**"user for years with zero problems"**
3.89	"i bought this for my laptop and it worked great for me and it has been working great for me"

Table 9: Tips that are very similar or the same as produced by the NRT″ on the test set for Electronics-200K. Ground truth is marked in bold.

Reconstruction of NRT Compared to Viewpoint 1

NRT with Adadelta and regularization performs very poorly, and we suspect that the AdaDelta[31] algorithm might be faulty in its GPU implementation in Tensorflow. When we tried to only change the optimizer, we still had problems regarding regularization. Even though NRT″ achieves good ROUGE scores on the Electronics-200K dataset, as seen in table 4, we still find a problem with diversity of tips for NRT″. While the diversity of the tips are not addressed in Li et al.[8] we know from their results, that their model contained more than single word sentences and that their presented results showed a good variation in their tips. Due to lack of diversity for NRT″ as presented in fig. 7, we provide a further examination of tips provided by NRT″ and find that many of the generated tips are very similar in their content as seen in table 9 where the ground truth is marked with bold. This points to a problem regarding evaluation of ROUGE score without also looking at diversity. Given all of these results it is evident, that we did not successfully reconstruct the NRT model when compared using viewpoint 1.

Reconstruction of NRT Compared to Viewpoint 2

While NRT″ achieves the highest ROUGE scores on the Electronics-200K dataset, but with low diversity, we found that NRT* could achieve comparable ROUGE scores while still having a high diversity. By training NRT* on the Electronics-Full dataset we found that the ROUGE scores were close to the ones by Li et al.[8] but with theirs being better especially for ROUGE-2. This difference can of course be due to the difference in experimental settings and implementations.

For rating prediction we have measured the performance of the NRT model we have reconstructed. In table 7 we see that we are not able to achieve similar performance on rating prediction as Li et al.[8] achieved. We see that the SVD++ and the URP model are performing the best of the baseline models.

We see in table 10 that sometimes there is still a coherence between the predicted rating and the textual output, but with a higher error we can expect that we are further from the ground truth tip. In our case we predict much more positive ratings and thus write more positive tips. Considering the overall performance of NRT* compared to the NRT model implemented by Li et al.[8] we think we are able to reconstruct the model by Li et al.[8] with similar results when comparing using viewpoint 2.

5	**"great for nuke and photoshop"**
4.67	"this is the best stylus i have ever owned and i have had it for a couple of months now"
3	**"i was very excited when i got this pc"**
0.90	"i bought this to use with my laptop and it worked great for a few months then it stopped working"
5	**"a great performance keyboard for a good price"**
4.80	"i bought this keyboard because i wanted a keyboard that i could use with my ipad and it works great"
1	**"i have tried it in my set up"**
1.23	"i bought this unit to replace my old one that had died on me after a few months of use"

Table 10: Tips where the sentiment is coherent with the rating for NRT on Electronics-Full. Ground truth is marked in bold.*

Model with Matrix Factorization (RQ2)

We see in table 11 the sentiment from the predicted rating is not carried very well to the generated tip. It is carried in some cases as seen in 11 and Li et al.[8] also show that they can vary their tips based on the sentiment. However they show a very limited number of examples as we do in

table 10. This can be misleading and by further inspection it seems that the general trend is that the sentiment is not learned that well.

5	**"i love this replacement cap for my canon especially the <unk> to keep it safe and secure"**
1.38	"i bought this to use with my canon eos rebel t3i and it works perfectly with my canon eos rebel"
2	**"mostly work as advertised one major problem"**
2.98	"i bought this for my wife to use with her ipod touch and it works great and she loves it"
2	**"the sound wasnt particularly great"**
4.56	"i have been using this product for about a month now and it has been working great so far"
1	**"this drive for some stupid reason requires two usb ports to work and comes with bloatware preinstalled which pissed me"**
2.64	"i bought this to use with my macbook pro and it worked great for the price i paid for it"

Table 11: Tips produced by the NRT that do not adhere to their rating from the test set for Electronics-Full. Ground truth is marked in bold.*

Based on this detaching the latent factors for the rating prediction model seems like a good thing to do. As the sentiment is not encoded in the latent features but rather the tip generation model will learn that it is merely a numeric input.

We see that the NSVDT model perform well both in rating prediction(table 7) and in ROUGE score(table 6). On rating, it significantly outperforms all other methods both on MAE and RMSE. On the ROUGE scores, it is also the best performing model. It achieves the best performance on almost every ROUGE score and otherwise, it achieves the second best performance. The NRT*A model performs slightly better in ROUGE-1 due to its higher recall.

Looking at these results it appears that the modification of the NRT model with the use of MF for rating prediction has provided us with a model that is both simpler to train and has an improved performance. And we can see in table 12 that we are able to generate tips that match a sentiment and a context to some degree. It is hard to say exactly why this modification has an improved performance, but we will try to address the matter anyways. Perhaps the shared latent representation between rating prediction and tips generation is not a good idea. Maybe the model is better because it has a latent representation in the SVD to predict ratings and a different latent representation for the rest of the model used for tips generation. One could argue that it is different features that determine what rating a user would give an item and the features that determine what tip a user would give that same item. When they are detached the model is free to learn different features for these and as we see on the results, this appears to have a positive impact on the performance.

A benefit of the NSVDT mode is that it would be possible to add more users and items to the SVD much more easily and only have to retrain the SVD and not the entire model. Then the pretrained latent factors for the new users and items could be used in the rest of the network to generate tips. This would greatly reduce the training time to update this model when new users and items needs to be added. However, we can only speculate that the performance for users or items that are only trained by the SVD will be lower. As the SVD is separate from the rest of the model it also enables us to do a thorough parameter sweep in a reasonable time horizon. This makes it possible to find good parameters for the SVD which improves the performance of this model.

5	**"my purchase of this bag the timbuk2 swig backpack in black was impulsive"**
4.57	"this is a great bag for the price and it fits my laptop perfectly and has a lot of pockets"
2	**"64mb of software constant checks for updates items in my system tray just for a keyboard and mouse"**
2.63	"i bought this keyboard to replace my old logitech keyboard that i had for a year and a half ago"
1	**"a prime example of how to ruin a barely working product"**
2.77	"i bought this to use with my mac pro and it worked great for about a year then it died"
5	**"the best canon flash to date"**
5.08	"i bought this to use with my canon eos rebel t3i and it works great"

Table 12: Tips generated by NSVDT on Electronics-Full. Ground truth is marked in bold.

In this model, we also tried to initialize the latent factors in the network with the latent factors we have in the SVD after it has trained. From what we can see on these results it appears as if this works quite well, but as we have not trained a model without using the latent factors from the SVD as the initialization for the latent factors in the network we do not have a good way to measure how this impacts the model.

NRT with Attention (RQ3)

The assessment of RQ3 is based on the model NRT*A. We see the results for RMSE and MAE in table 7. The model is slightly worse than NRT*, however, if we look at the ROUGE scores in table 6 we see that NRT*A is the second best overall. In fig. 8 we also see that NRT*A show good diversity close to that of NSVDT. In table 13 we see tips that show a very good coherence between the items or topics in the ground truth and the generated sentence and the generated tips also carry the sentiment of the predicted rating well.

We are interested in the context awareness that NRT*A brings. It has a good diversity and achieves good ROUGE scores and in table 13 we see that there is a good coherence and context awareness between predicted rating and generated tip. As earlier mentioned a generative model that merely describes the sentiment or rating in a generic way is not always interesting and could simply be replaced by a set of manually generated generic sentences. We find NRT*A interesting because it provides results that are more context aware and will possibly give a better explanation of a recommendation than a sentence that could be said about any item like, *"This is a great product"*. If a user can identify that a tip is talking about the item he is recommended or looking at, one could imagine that he might be more prone

to believe it or find that it describes a certain feature of the item that is interesting to him.

5	**"great price charged fine and then held the charge"**
4.14	"i bought this to replace the original battery that came with my camera and it worked great for the price"
5	**"i just received my new er6i isolator headphones"**
1.01	"i bought these headphones for my wife to use with her ipod touch and they are very uncomfortable and uncomfortable"
2	**"i had high expectations for this product given the sennheiser brand"**
2.34	"i have had these for a few months now and i am very disappointed with the sound quality of the"
5	**"great all around bag very versatile"**
4.97	"i bought this bag to use with my new camera and it works great for the price i paid for"

*Table 13: Tips produced by the NRT*A from the test set for Electronics-Full that show a big correlation between the products mentioned in the ground truth as well as adhering to the predicted rating. Ground truth is marked in bold.*

The rating prediction of NRT*A is slightly worse than that of NRT*. Already in our attempt to reconstruct the work of Li et al.[8] we noticed that the rating prediction suffered higher errors when the network achieved better ROUGE scores. Our hypothesis is that because the tips generation part of the network gets more complex it has a larger influence on the latent factors and the gradient descent, the model will have a harder time converging to a point where both the rating and the tips generation part are at a minimum.

Scaling Loss Based on Rating for Better Tip Generation (RQ4)

We evaluate whether the scaling of the loss has a positive effect on the generation of tips by evaluating the ROUGE scores on each rating separately. In fig. 9 we see the distribution of ROUGE scores based on each different rating. It is evident that the performance in ROUGE score follows the trend of the dataset. We are significantly better at generating tips when the predicted rating belongs to the majority sentiment.

In fig. 9 we see that by scaling the loss we get a more even curve. We punish the model in such a manner that every rating have an equal impact on the total loss. This means that when we predict a minority sentiment wrong it is punished more than when we predict a majority sentiment wrong. In general, we get a lower ROUGE score overall. We attribute this to the fact that finding the right descent in our gradient descent might be hard when we choose to change certain weights much more aggressively than others and we speculate that we end up in a different local minimum.

Another approach for tackling this problem could be through preprocessing. We could construct a dataset where there is an even distribution of reviews on all ratings, which would prevent the model from focusing heavily on the most frequent rating. But as we can see in fig. 5 rating 1, 2 and 3 are significantly less frequent compared to rating 4 and 5. So if this approach was used we would have to discard a high amount of the reviews, which would result in a much smaller dataset. Another problem that occurs when we remove reviews from the dataset is the guarantee that all users and items are 5-core. This guarantee will be hard to maintain when removing most of the reviews, and it might even be impossible to maintain. This smaller dataset would most likely result in a lower performance when trained compared to the original larger dataset. To avoid having to modify the dataset we could also have used under- or oversampling to achieve a similar effect as shown in Kotsiantis et al.[13]. The first option would be to oversample reviews from rating 1, 2 and 3 in such a way that the class distribution in the dataset would appear to be even. The second option would be to undersample reviews from rating 4 and 5, such that we again achieve an even class distribution.

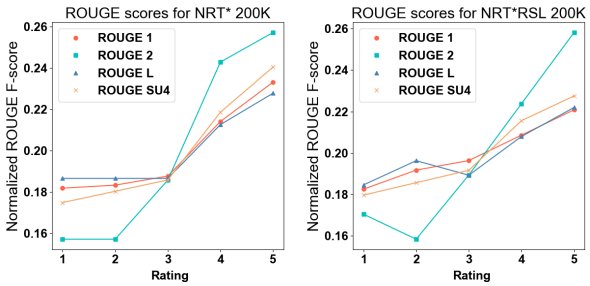

Figure 9: Normalized ROUGE scores for each rating for NRT and NRT*RSL. NRT* achieves high scores for rating 4 and 5, but falls behind at lower ratings, with ROUGE-2 being very low for rating 1 and 2. NRT*RSL achieves slightly greater scores on lower ratings, but it has a negative impact on the ROUGE scores for the higher ratings.*

Finally there is the question if it is even necessary to force the model to be better at generating more negative tips for lower ratings. If the model is used to generate tips for items recommended to a user because we predict that the user would be interested in the item, then the rating will always be high. This results in the model only having to show the tips it generates when there is a positive rating, and in this case you would prefer that the model performs better on these ratings as this is what will actually be seen by the users. On the other hand the model could also be used to predict a rating and generate a tip for any item the user decides to look at. In this case the user might look at an item that would not be recommended for him, and in this case the model would have to generate a negative tip. In this case it would be desired to have a model that is good at both positive and negative.

6 Reflections

In the following sections we will reflect upon the results and more general discussions of this research as well as the findings we have come across working on this.

Representativeness of Electronics-200K Dataset

We developed Electronics-200K for our preliminary experiments to find good configurations of the model to train on Electronics-Full. We concluded that this dataset appeared

to be representative of Electronics-Full as it maintained 5-core on items and had a similar class distribution. Now that we have the results from Electronics-Full we note that some of the models do not appear to follow the same pattern on both datasets. If we look at how diverse the various models are on both datasets, we notice that this varies for the models. If we look at fig. 8 we can see that NRT* is the least diverse on Electronics-Full, but if we look at fig. 7 we can see that it is actually the second most diverse model on Electronics-200K. We also note that NSVDT is diverse on Electronics-Full, but on Electronics-200K that is not the case. The only model that has a similar diversity on both datasets is NRT*A, which appears to have a good diversity on both datasets. On the other hand if we look at ROUGE scores the models appear to have a similar performance on both datasets as can be seen in table 4 and table 6, where the models NRT*, NRT*A and NSVDT all have similar performances on both datasets. These results make us think that the features of the datasets must still be similar, and even though the diversity of the models variate on the datasets, we still think that the preliminary results performed on Electronics-200K provides us with a good indication of how the models will perform on Electronics-Full. We also experimented with another small dataset with 250K reviews. We did some preliminary experiments with this dataset which is described in Appendix B, but found that that all results gave bad results regarding the quality of the generated text and ROUGE scores. The results from Electronics-200K made us more certain that changing to this dataset, was better for experimentation.

Precision/Recall

When we compared the ROUGE score results from Li et al.[8] with the results in this paper we noticed that the values for precision and recall appear to have been swapped. When we first noticed this we went back into our code to verify that we did everything correct, which appeared to be the case. From this, we can conclude that the results obtained by Li et al.[8] must differ a lot from the results we have achieved or they must have accidentally swapped the values for precision and recall in their results. We cannot say for sure that they are swapped, as a model that generates a different kind of sentences could achieve different scores. If a model often generates long tips it will be easier to achieve a high recall, but not a high precision. Whereas a model that often generates short tips will most likely achieve a higher precision at the cost of a lower recall. As Li et al.[8] do not show a lot of the output generated by their model it is hard to know if their model often writes long or short tips, and hence it is hard to say if they swapped their precision and recall, or if they just managed to train a model that differs from the one we trained.

Beam Search

When generating tips we are using a beam search algorithm similar to the one used by Li et al.[8] to explore various possible sequences. This helps us generate a more likely tip, which also helps us improve our ROUGE scores compared to the results without beam search. As the beam search with the same configuration as used by Li et al. already improved the model's performance we did not look further into this. This could easily be a place where the model's performance could be improved. If we look at the tips we are generating we have an issue that the ending of the tip is often repeating a meaning mentioned earlier in the same tip which can be seen in table 9 or that the tip is cut off mid-sentence which can be seen in table 13. This is most likely a result of the scaling of our length normalization for the beam search, where we reward the model for writing a longer sequence. If the reward is too high the model will always write 20 words, even when there is no meaning left to write. If the reward is too small the model will write very short tips because the probability of the longer sentences will become too small. As we did not try to modify the parameters for length normalization, we do not know if the current setting is the optimal one.

The importance of Preprocessing

We want to address some experiences we have gathered regarding the reproduction and development of the models presented in this paper. It was evident that the quality of the dataset can have a large impact on the performance of individual models. The difference can, for example, be seen between Electronics-200K and Electronics-Full where the diversity of words vary between the different models respectively. Additionally, since we extracted sentences from reviews where an acceptable tip was not available, the content of some tips may not be structured or as informative as a normal tip would be.

How one goes about preprocessing the data can also have a great impact on how well the final model can perform. A distinct choice for the problem with text generation is the inclusion or exclusion of features in the text such as stopwords and replacement of infrequent words with an <unk>token. Which stopwords are excluded and the threshold for infrequent word replacements can have an impact on how well the model understands relations between words and latent representations.

Lastly, we address the problem of reproducibility of research in the field of text generation. It is our experience that reproducing the result of Li et al.[8] proved troublesome, since many parameters and aspects were not addressed in detail. This can result in a different implementation. For example Li et al.[8] wrote, *"For samples without tips, the first sentence of review texts is extracted and regarded as tips"*. However, in the document, we received regarding their preproccesing they actually extracted the first sentence with 5 or more words. We have tried to state our assumptions in order to give a throughout the description of our implementations. The problem seems to be that the many layers of abstractions present when building machine learning models makes it difficult to perform a scientific reconstruction. We think that the choice of implementing the models in Tensorflow[33] instead of Theano[37] as stated in Li et al.[8] may have had a large impact on the results. For example, we know that in Tensorflow the Frobenius norm is implemented as $\frac{1}{2}\sum_{k=1}^{n}|x_k|^2$ rather than $\sqrt{\sum_{k=1}^{n}|x_k|^2}$. The way Tensorflow calculates gradient it cannot solve the gradient for a square root of the square. We do not know how this is implemented in Theano and there are most likely many more differences like this that are done in order to optimize the libraries.

First Token Repetition

The results reveal that the NRT model, as well as our extensions, have a tendency to start tips with "i". The reason is that the tips generation part learns the statistics of the dataset and that beam search might be more prone to find the same sentences if some sentences are statistically much more likely. The ground truth contains "i" as the first word 17.33% of the time compared to the next most frequent word "this" that appears 8.36% of the time. This means that by sheer statistics we end up with a result that contains "i" as the first word a lot of the time. However, it is not necessarily a bad thing as it might contribute to users being able to identify with a recommendation better and experience it more personalized. If the use case was to produce a sample tip for a user, it would also make sense that it starts with "i" as it is something a user would say.

7 Conclusion

We propose three different methods of extending or changing the work of Li et al. [8]. The proposed models NSVDT, NRT*A and NRT*RSL are all based on the NRT model. We try to reconstruct the model NRT and cannot achieve similar scores. We alter the setup under which we run experiments and achieve similar results. The proposed models show better performance in rating prediction and abstractive generation of tips under our setup. NRT*A and NRT*RSL are extensions of the NRT model that do not alter the multitask learning approach proposed by Li et al.[8]. In NRT*A we propose to add an attention mechanism in order to diversify the results and gain context awareness. In NRT*RSL we propose to scale the loss function of the rating prediction in order to achieve a better ROUGE score on negative sentiment tips. The NSVDT model introduces pretraining of the latent factors and detaches the learning of the rating prediction component from the multitask learning. This shows useful for diversification and simplification of the network. We show that the proposed models are able to diversify the generated tips and avoid generating generic and unrelatable tips while achieving better RMSE, MAE and ROUGE scores. In the future, we propose to run the models on the different benchmark datasets used in Li et al.[8]. Another proposal is to granularize the attributes for the attention. This could extend on the idea that some users might be looking for different features than others and find a more granular tip interesting. Lastly, we want to test the proposed models on data evenly distributed in rating or perhaps look at other ways to compensate for the imbalance in the datasets.

8 Acknowledgments

We would first and foremost like to thank our supervisor Peter Dolog for advice and guidance throughout the process of writing this paper. We would like to thank Piji Li for clarifications regarding the NRT model. Last we would also like to thank our fellow students for reflections and participation in discussions regarding our work.

References

[1] A. Sharma, *How much traffic do recommender systems actually cause?* Blog, Apr. 2016. [Online]. Available: https://medium.com/@amit_sharma/how-much-traffic-do-recommender-systems-actually-cause-2e0c6708801f (visited on 06/05/2018).

[2] G. Sielis, A. Tzanavari, and G. Papadopoulos, "A review of recommender systems: Types, techniques and applications," pp. 329–339, Jul. 2014.

[3] C. Wang and D. M. Blei, "Collaborative Topic Modeling for Recommending Scientific Articles," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11, New York, NY, USA: ACM, 2011, pp. 448–456, ISBN: 978-1-4503-0813-7. DOI: 10.1145/2020408.2020480. [Online]. Available: http://doi.acm.org/10.1145/2020408.2020480 (visited on 05/04/2018).

[4] L. Dong, S. Huang, F. Wei, M. Lapata, M. Zhou, and K. Xu, "Learning to generate product reviews from attributes," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Association for Computational Linguistics, 2017. DOI: 10.18653/v1/e17-1059. [Online]. Available: https://doi.org/10.18653/v1/e17-1059.

[5] M. K. Christensen, S. N. Gustafsson, and L. M. L. Würtz, *A Study on Controlled Text Generation*, ENG, Jan. 2018. [Online]. Available: https://goo.gl/TY9TAc (visited on 05/04/2018).

[6] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward Controlled Generation of Text," *arXiv:1703.00955 [cs, stat]*, Mar. 2017, arXiv: 1703.00955. [Online]. Available: http://arxiv.org/abs/1703.00955 (visited on 05/04/2018).

[7] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating Wikipedia by Summarizing Long Sequences," *arXiv:1801.10198 [cs]*, Jan. 2018, arXiv: 1801.10198. [Online]. Available: http://arxiv.org/abs/1801.10198 (visited on 05/04/2018).

[8] P. Li, Z. Wang, Z. Ren, L. Bing, and W. Lam, "Neural Rating Regression with Abstractive Tips Generation for Recommendation," *arXiv:1708.00154 [cs]*, pp. 345–354, 2017, arXiv: 1708.00154. DOI: 10.1145/3077136.3080822. [Online]. Available: http://arxiv.org/abs/1708.00154 (visited on 05/04/2018).

[9] Z. C. Lipton, S. Vikram, and J. McAuley, "Capturing meaning in product reviews with character-level generative text models," *arXiv preprint arXiv:1511.03683*, 2015. [Online]. Available: https://arxiv.org/pdf/1511.03683.pdf.

[10] R. Longadge and S. Dongre, *Class imbalance problem in data mining review*, 2013. eprint: arXiv:1305.1707.

[11] L. Zhang, K. Hua, H. Wang, G. Qian, and L. Zhang, "Sentiment analysis on reviews of mobile users," vol. 34, pp. 458–465, Dec. 2014.

[12] S. Visa and A. Ralescu, "Issues in mining imbalanced data sets - a review paper," Jan. 2005.

[13] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: A review," vol. 30, pp. 25–36, Nov. 2005.

[14] A. van Duijnhoven, J. Korst, and W. Verhaegh, "Collaborative filtering with privacy," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[15] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009, ISSN: 0018-9162. DOI: 10.1109/MC.2009.263.

[16] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proceedings of the 20th International Conference on Neural Information Processing Systems*, ser. NIPS'07, Vancouver, British Columbia, Canada: Curran Associates Inc., 2007, pp. 1257–1264, ISBN: 978-1-60560-352-0. [Online]. Available: http://dl.acm.org/citation.cfm?id=2981562.2981720.

[17] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *In NIPS*, MIT Press, 2000, pp. 556–562.

[18] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *In Proc. of the 14th ACM SIGKDD conference*, 2008, pp. 426–434.

[19] B. M. Marlin, "Modeling user rating profiles for collaborative filtering," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds., MIT Press, 2004, pp. 627–634. [Online]. Available: http://papers.nips.cc/paper/2377-modeling-user-rating-profiles-for-collaborative-filtering.pdf.

[20] S. Funk, *Incremental svd*, ENG, May 2018. [Online]. Available: https://github.com/aaw/IncrementalSVD.jl (visited on 05/22/2018).

[21] Netflix, *Netflix prize*, ENG, May 2018. [Online]. Available: https://www.netflixprize.com/index.html (visited on 05/22/2018).

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. eprint: arXiv:1412.6980.

[23] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, ACM Press, 2011. DOI: 10.1145/2020408.2020480. [Online]. Available: https://doi.org/10.1145/2020408.2020480.

[24] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *J. Artif. Int. Res.*, vol. 22, no. 1, pp. 457–479, Dec. 2004, ISSN: 1076-9757. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622487.1622501.

[25] J. McAuley, *Amazon 5-core datasets*, 2018. [Online]. Available: http://jmcauley.ucsd.edu/data/amazon/.

[26] G. Guo, *LibRec - A Java Library for Recommender Systems*. [Online]. Available: https://www.librec.net/ (visited on 05/28/2018).

[27] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?" *Geosci. Model Dev.*, vol. 7, Jan. 2014. DOI: 10.5194/gmdd-7-1525-2014.

[28] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Proc. ACL workshop on Text Summarization Branches Out*, 2004, p. 10. [Online]. Available: http://research.microsoft.com/~cyl/download/papers/WAS2004.pdf.

[29] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," vol. Vol. 4304, pp. 1015–1021, Jan. 1970.

[30] P. Tardy, *Rouge*, ENG, May 2018. [Online]. Available: https://github.com/pltrdy/rouge (visited on 05/28/2018).

[31] M. D. Zeiler, *Adadelta: An adaptive learning rate method*, 2012. eprint: arXiv:1212.5701.

[32] P. Li, *Optimization algorithms for deep learning*, 2018. [Online]. Available: http://lipiji.com/docs/li2017optdl.pdf.

[33] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[34] L. Shostenko, *Lexrank: Graph-based lexical centrality as salience in text summarization*. ENG, May 2018. [Online]. Available: https://pypi.org/project/lexrank/ (visited on 05/31/2018).

[35] D. Kim, *Collaborative topic modeling for recommending scientific articles*, ENG, Jun. 2018. [Online]. Available: https://github.com/dongwookim-ml/python-topic-model (visited on 06/01/2018).

[36] M. K. Christensen, S. N. Gustafssonand, and L. M. L. Würtz, *Results from our models*. [Online]. Available: https://github.com/lwartz13/p10speciale (visited on 06/14/2018).

[37] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688.

Appendices

A Overview of configurations for preliminary experiments

Config	Optimizer	Distribution	λ_n	Extension
C1	AdaDelta	Uniform	1e−4	
C2	Adam	Normal	1e−4	
C3	Adam	Mixed	1e−4	
C4	Adam	Uniform	1e−4	
C5	Adam	Normal	1e−8	
C6	Adam	Mixed	1e−4	
C7	Adam	Uniform	1e−8	
C8	Adam	Uniform	1e−8	s1
C9	Adam	Uniform	1e−8	s2
C10	Adam	Uniform	1e−8	at
C11	Adam	Uniform	1e−8	svd
C12	Adam	Uniform		nR
C13	Adam	Uniform	1e−8	lfi

Table 14: Overview of model configurations. Below is a table showing the meaning of the extension acronyms.

s1	*scaled cost based on rating*
s2	*scaled cost based on rating*
at	*attention*
svd	*the extended SVDNRT model*
nR	*no regularization*
lfi	*user and item latent instantiated from uniform distribution [-1, 1]*

B Results on Electronics-250K

In the preliminary experiments we also experimented with a different small dataset. This dataset was build with the algorithm seen in fig. 10. We also tried running various configurations on this dataset, but as we concluded that the Electronics-200K was a better representation of Electronics-Full, we did not use the results on Electronics-250k in the paper.

	RMSE	MAE	ROUGE-1	ROUGE-2	ROUGE-l	ROUGE-SU4
C1	1.1446	0.8951	0.01	0.00	0.00	0.00
C2	1.2323	0.9098	5.22	0.01	2.00	1.41
C3	1.1446	0.8949	8.47	0.01	3.28	2.33
C4	1.1446	0.8959	5.22	0.01	2.00	1.41
C5	1.2372	0.9218	8.47	0.01	3.28	2.33
C6	1.1861	0.8262	13.77	1.36	7.08	4.02
C7	1.1884	0.8280	**[1]16.80**	**[1]1.86**	**[1]10.32**	**[1]4.43**
C8	1.4762	1.1472	14.35	1.42	8.14	4.05
C9	1.4727	1.1715	[2]15.45	[2]1.69	[2]10.15	[2]4.15
C10	[2]1.1635	**[1]0.8181**	13.92	1.25	9.05	3.64
C11	1.6254	1.1336	11.47	0.93	5.55	3.49
C12	1.1886	[2]0.8257	12.47	1.17	6.45	3.75
C13	**[1]1.1413**	0.8238	0.1322	0.0121	0.0706	0.0383

Table 15: Results of configurations seen in table 14 on Electronics-250K dataset.

```
    input: reviews: List, limit: int

    s = Set()
    userQueue = Queue()
    itemQueue = Queue()

    userQueue.push(//user from first review)

    while s.size < limit:
        currentUser = userQueue.pop()
        reviewsForUser = currentUser.getReviews()
        s.add(reviewsForUser)
        for review in reviewsForUser:
            itemQueue.push(review.item)
        nextItem = itemQueue.pop()
        usersForItem = nextItem.getUsers()
        for user in usersForItem:
            if user.hasBeenVisited == False:
                userQueue.push(user)
```

Listing 1: Pseudocode for Retrieving Reviews for the Small Dataset.

Figure 10: Algorithm used to generate the Electronics-250k dataset.