Université de Montréal

Recommandation Conversationnelle: Écoutez avant de parlez

par

Nicholas Vachon

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

Orientation Intelligence Artificielle

May 5, 2021

Résumé

Dans un monde de mondialisation, où les offres continuent de croître, la capacité de référer les gens vers leurs besoins spécifiques est essentiel. Après avoir été un facteur de différenciation clé pour Netflix et Amazon, les systèmes de recommandation en général ne sont pas près de disparaître. Néanmoins, l'un des leurs inconvénients est qu'ils sont principalement basés sur des informations indirectes (notre comportement, principalement du passé) par opposition à une demande explicite à un moment donné.

Le développement récent de l'apprentissage automatique nous rapproche de la possibilité d'exprimer nos besoins spécifiques en langage naturel et d'obtenir une réponse générée par la machine. C'est ce en quoi consiste la recommandation conversationnelle. La recommandation conversationnelle englobe plusieurs sous-tâches d'apprentissage automatique. Dans ce travail, nous concentrons notre étude sur les méthodes entourant la tâche de recommandation d'item (dans notre cas, un film) à partir d'un dialogue.

Pour explorer cette avenue, nous adaptons et étendons les techniques de modélisation du langage basées sur les transformeurs à la tâche de recommandation à partir du dialogue. Nous étudions les performances de différentes méthodes à l'aide de l'ensemble de données ReDial [**25**], un ensemble de données de recommandation conversationnelle pour les films. Nous utilisons également une base de connaissances de films et mesurons sa capacité à améliorer les performances lorsque peu d'information sur les utilisateurs/éléments est disponible.

Ce mémoire par article est divisé comme suit. Tout d'abord, nous passons en revue tous les concepts de base et les travaux antérieurs nécessaires à cette lecture. Ensuite, nous élaborons les spécificités de notre gestion des données, les différents modèles que nous avons testés, la mise en place de nos expériences et les résultats que nous avons obtenus. Suit l'article original que nous avons soumis à la conférence RecSys 2020. Notez qu'il y a une incohérence mineure puisque tout au long du mémoire, nous utilisons v pour représenter les éléments mais dans l'article, nous avons utilisé i.

Dans l'ensemble, nous constatons que *les modèles de transformeurs pré-entraînés surpassent les modèles de bases même si les modèles de base ont accès aux préférences utilisateur extraites manuellement des dialogues.*

Mots-clés: Recommandation, systèmes de dialogue, transformeurs

Abstract

In a world of globalization, where offers continue to grow, the ability to direct people to their specific needs is essential. After being key differentiating factors for Netflix and Amazon, Recommender Systems are nowhere near a downfall. Still, one downside of the basic recommender systems is that they are mainly based on indirect feedback (our behaviour, mainly from the past) as opposed to explicit demand at a specific time.

Recent development in machine learning brings us closer to the possibility for a user to express its specific needs in natural language and get a machine generated reply. This is what Conversational Recommendation is about. Conversational recommendation encapsulates several machine learning sub-tasks. In this work, we focus our study on methods for the task of item (in our case, movie) recommendation from conversation.

To explore this setting, we use, adapt and extend state-of-the-art transformer based neural language modelling techniques to the task of recommendation from dialogue. We study the performance of different methods using the ReDial dataset [**25**], a conversational-recommendation dataset for movies. We also make use of a knowledge base of movies and measure their ability to improve performance for cold-start users, items, and/or both.

This master thesis is divided as follows. First, we review all the basics concepts and the previous work necessary to this lecture. When then dive deep into the specifics of our data management, the different models we tested, the set-up of our experiments and the results we got. Follows the original paper we submitted at RecSys 2020 Conference. Note that there is a minor inconsistency since throughout the thesis, we use v to represent items but in the paper, we used i.

Overall, we find that *pre-trained transformer models outperform baselines even if the baselines have access to the user preferences manually extracted from their utterances.*

Keywords: Recommendation, Dialogue Systems, Transformers

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et des abréviations

$m_u^{1:t}$	Conversation of user u including utterances 1 to t
$\mathbf{R}_u^{1:t}$	Observed ratings of user u for all items, up to utterance t
$\hat{\mathbf{R}}_u^{t+1}$	Predicted ratings of a user u for all items, at utterance $t+1$
$\hat{R}_{uv}^{t+1}$	Predicted rating of a user u for item v, at utterance $t+1$
u	A user
$\mathcal{U}$	Set of all users
v	An item (note: items are referenced by i in the article)
$\mathcal{V}$	Set of all items

Remerciements

I would like to thank Professor Laurent Charlin for supervising this research project. From the first time I met him with my interest in recommender systems and this vague plan to tell stories with artificial intelligence, I felt listened to, welcomed and supported. His passion and his experience were able to guide me wisely throughout my studies and even for further projects. I am also grateful to my co-director, Christopher Pal, for his critical eye which pushed me to move forward. For subsidizing my research grant, I'd like to thank Mila and HEC. My thanks also go to my lab comrades and to all the other people who advised or helped me during this master's degree, in particular Samuel and Joseph. On the administrative side, things wouldn't go as smoothly without the support of Céline and Linda. Finally, a special thanks to my husband, Gabriel Fortin, for his continuous support and patience in this crazy project of going back to school at 43 years old to complete a master's degree.

Chapitre 1

Reviewing the Basics

In this chapter, we review lines of works related to ours, including machine learning, recommender systems, natural language processing, transformers and conversational recommendation.

1.1. General Machine Learning Concepts

Machine learning is a set of statistical methods to help make sense of a data set, to gain useful insight or to automate a task. More specifically, it is the study of algorithms that are able to fit a model, often predictive, based on a data set. By extracting the structure of the data, such algorithms are able to solve tasks that might be complicated to code otherwise. Much of the machine learning algorithms, including those in this memoir, use parametric models. A model is a family of functions and the goal of training is to estimate the parameters giving the "best" function of this family for a given task.

Machine learning algorithms are commonly separated into three classes: supervised, unsupervised, and reinforcement learning. In this work, we are in a supervised setting and we review the basic concepts of supervised learning below.

1.1.1. Supervised learning

Supervised Learning is a type of machine learning in which an algorithm is fed some training data such that the algorithm has ex-ante 'rules' or a specific, known target to obtain with a specific data point. Suppose we observe the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i); i = 1,...,N\}$ which are independent and identically distributed (i.i.d) samples from a joint distribution over $(\mathbf{x}, y)$. The learning is said to be supervised because each $\mathbf{x}_i$ has a label y_i. Supervised learning consists of estimating a function $f : \mathbf{x} \to y$ using a dataset $\mathcal{D}$. When y is a categorical variable, we speak of a classification problem. For example, predicting whether the value of a financial security will go up or down is a classification problem: it is about deciding, for a set of features, whether it is positive or negative (2 possible outcomes). When

y takes continuous values, we speak of a regression problem. For example, predicting the price of a stock is a regression problem. Among the most efficient and studied supervised learning algorithms, one can cite Support Vector Machines [**9**], Random Forests [**4**] or neural networks, described in Section §1.1.4.

1.1.2. Optimization

1.1.2.1. Loss function. We often measure the performance of a supervised model f with a loss function or cost function, $l(y, f(\mathbf{x}))$. For example, a quadratic loss is written $l(y, f(\mathbf{x})) = ||y - f(\mathbf{x})||^2$. Facing a binary classification problem, using the cross-entropy loss function $l(y, f(\mathbf{x})) = -(y \log f(\mathbf{x}) + (1-y)\log(1-f(\mathbf{x})))$ is often more efficient. Given a loss function l, we try to find the function f which minimizes the risk $R(f)$ defined by:

$$R(f) = \mathbb{E}_{(\mathbf{x},y)}\left[l(y, f(\mathbf{x}))\right].$$

Since the distribution of the data is unknown, the empirical risk is rather minimized:

$$\hat{R}(f,\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} l(y_i, f(\mathbf{x}_i))$$

which calculates the average error over the entire data set D. Thus, if the function f is parameterized by θ, we then look for

$$\theta^* = \arg\min_{\theta} J(\theta) \tag{1.1.1}$$

where:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(y_i, f_\theta(\mathbf{x}_i))$$

$J(\theta) \in \mathbb{R}$ is called the objective function and training a model is equivalent to optimizing (here minimizing) this objective function.

In the case of an unsupervised problem, we may estimate a probability density $P_\theta(\mathbf{x})$ and seek to maximize its likelihood. Hence, the goal is still to minimize Equation 1.1.1, but with

$$J(\theta) = -\log \prod_{i=1}^{N} P_\theta(\mathbf{x}_i) = -\sum_{i=1}^{N} \log P_\theta(\mathbf{x}_i)$$

using the fact that $\mathcal{D}$ is i.i.d and log is a monotonic function.

1.1.2.2. Gradient Descent. The objective function J depends on the dataset and can have a very complex expression. There is usually no analytical solution to the $\arg\min_\theta J(\theta)$ optimization problem. Therefore, we use a technique called *gradient descent*. Gradient descent is an iterative approach that makes it possible to approximately find a local minimum of a differentiable function. The J function defines a cost surface and the idea is to follow the direction that locally gives the “steepest” slope, down the cost surface. Each update of the model parameters is written:

$$\theta \leftarrow \theta - \eta \nabla \mathbf{J}(\theta)$$

where η is the learning rate and $\nabla \mathbf{J}(\theta)$ is the gradient of J with respect to θ. The learning rate gives the size of the step to be taken in the direction of descent. In this configuration, the gradient must be calculated for the loss function J which takes into account the entire data set. To speed up the calculation and the update rate of the parameters, it is common to calculate the gradient using minibatches. A minibatch contains a small number of training examples. Several techniques aim to adapt the learning rate as training progresses [**29**].

1.1.3. Generalization

The capacity of a model designates its representation power. There are multiple ways to measure the capacity of a model, the Vapnik-Chervonenkis dimension [**45**] being one of them. A larger capacity model typically has a lot of parameters and will be able to approximate more complex functions. Consequently, high capacity models can achieve perfect precision on training data simply by memorizing it.

The goal of machine learning, however, is for a model to *generalize* well. That is to say, perform well on new data, data never seen during training. When an increase in the capacity of the model worsens performance on new data, we refer to this situation as *over-fitting*. Conversely, a model that is not complex enough (not enough capacity) can’t capture all the variations in the data and suffers from under-fitting.

A common way to identify the correct model is to divide the dataset into three subsets: the training set, the validation set, and the test set. The training set is used for optimizing model parameters. The validation set to identify the best hyper-parameters and stop training before over-fitting. Finally, the test set is used to report an unbiased estimator of the risk. Hyper-parameters control the general shape of the model and its capacity. It is important to adjust them to avoid over-fitting, under-fitting and to get a good generalization. The degree of polynomial regression, the number of epochs (the number of times we go through the entire data set during the gradient descent) or the learning rate are examples of such hyper-parameters.

Several methods, known as *regularization* methods, make it possible to reduce capacity. It is common to add a term proportional to the norm of the model parameters as a regularization. If $J(\theta)$ denotes the initial objective function, the new regularized objective function is written:

$$\tilde{J}(\theta) = J(\theta) + \lambda ||\theta||$$

where λ is a scalar hyper-parameter controlling the importance of the regularization and $||\cdot||$ is a norm, typically norm 1 or norm 2.

1.1.4. Neural Networks

Neural Networks (NN) are inspired by biological nervous systems in the sense that these models, capable of modelling very complex functions, are made up of a very large number of more basic computational units, called neurons.

Let $x \in \mathbb{R}^d$ be an input. A neuron calculates a linear combination of the components of x and then applies an activation function f to the result obtained:

$$y = f\left(\sum_{i=1}^{d} w_i x_i + b\right) = f(\mathbf{w}^T\mathbf{x} + b)$$

where $\mathbf{w}$ is the weight vector, and b is the bias.

Setting the activation function to be a nonlinear one is the key element that allows the neural network to learn nonlinear and increasingly complex functions. The choice of activation functions is made, among other things, on the values that the function can take, on the ranges where the derivative approaches zero or on the ease with which a computer can calculate its derivative. Frequently used activation functions include Sigmoid, tanh and Rectified Linear Unit (ReLU).

To build a layer of neurons, simply combine n neurons. The result, $\mathbf{y}$, becomes a vector in $\mathbb{R}^n$:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W} \in \mathbb{R}^{n\times d}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^n$ the bias vector of the neural layer and f is applied component by component. Here, the components of $\mathbf{W}$ and $\mathbf{b}$ constitute the parameters of the model which will be learned during training.

1.1.5. Deep Learning

Deep Learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input.

A deep neural network (DNN) is a neural network with multiple layers between the input and output layers. They can model complex non-linear relationships. DNN architectures are compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modelling more effectively complex data with fewer units than a similarly performing shallow network.

Examples of DNN include Convolutional Neural Networks (CNN), mainly used to treat images, but they also find applications in recommendation engines and financial time series, to name a few [**39, 47, 35**]. They can be seen as a regularized version of multilayer perceptrons as they limit the number of connections between neurons in different layers. They do so through a shared-weight architecture of convolutional kernels that shift over the features, taking advantage of their local connectivity. This technique also provides translation equivariance, which is useful when dealing with some problems, including image recognition.

Another significant family of DNN is the Recurrent Neural Networks (RNN), used to treat sequential data, as we'll see in Section 1.2.2.

1.2. Natural Language Processing

Natural Language Processing (NLP) studies how machines understand human language. Its goal is to build systems that can make sense of text and perform tasks like translation or topic classification, among others.

Throughout history, approaches to solving this problem varied.

From the '50s to the late '80s, many language-processing systems were designed by symbolic methods, implying the hand-coding of a set of rules, coupled with a dictionary lookup. These methods are often referred to as rule based methods. Even though the popularity of rule based methods has strongly declined these days in favour of neural systems, they are still relevant in some cases. For example, when the amount of data is low, for some pre-processing or post-processing steps or when interpretability and transparency are required. Major drawbacks of the technique include the fact that to increase the accuracy of the system, you need to add rules, which becomes harder and harder to manage. Also, rare cases or errors (which can be common in multiple natural language datasets, think social media) get complex to handle.

During the '90s and up to around 2010, a shift towards statistical methods happened. Here, statistical inference is used to learn rules through the analysis of large corpora. At first, the decisions were hard, using, for example, decision tree models. Then, models that make soft, probabilistic decisions took the lead, as they were better with unfamiliar inputs and errors. Moreover, they provided better results when integrated into larger systems. A drawback from these techniques is the elaborated feature engineering process, which brings us to the most popular techniques used today.

Since 2010 or so, neural networks took the lead, replacing feature engineering with word embeddings (see §1.2.1.2). Another important change resides in an end-to-end approach: from embedding to downstream task. These methods are not relying on separate intermediate tasks (e.g. part-of-speech). They also brought sequence-to-sequence models, as in neural machine translation.

1.2.1. Text Representations

Using simple statistics and computers to process raw text is limited. In this section, we first consider the possibility to upgrade raw text with features accounting for syntax and semantic. Then, we explore how text can be turned into numbers, which opens up a world of opportunities but brings the new challenge of finding the best numbers to represent text.

1.2.1.1. NLP Processing Pipeline. Text pipelines can be elaborate. An example is presented in Figure 1.1.[1] Since the goal of this work is not to overview the basics of NLP, we won't go into a description of each of these. The main idea is that executing a subset of all these steps, depending on the desired downstream task, provides a new representation for text, adding syntax and semantic features.

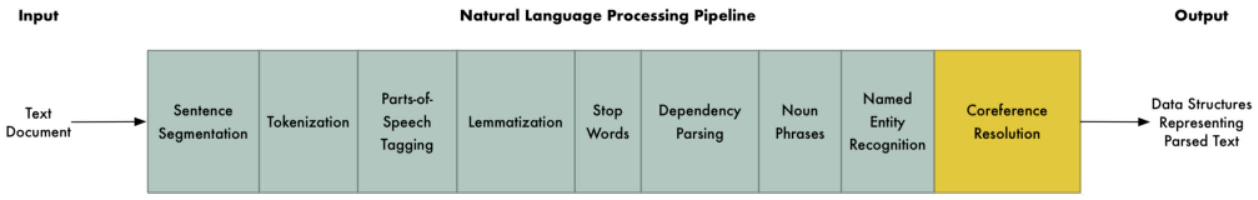

Fig. 1.1. Traditional pipeline in NLP.

1.2.1.2. Word Embeddgings. The first way one may think to represent words is to sequentially associate a value to each word. For example, if 'to' = 1, 'be' = 2, 'or' = 3 and 'not' = 4, the sentence «To be or not to be» could be expressed as (1,2,3,4,1,2). But, this representation has many pitfalls, namely that it's introducing size and ordering aspect, which will impact the statistics calculated. Why is 'to' smaller than 'not' or why is 'to' before 'not'?

One way to alleviate this problem is to represent text as a set of words, counting how many times a word appears in the text. Hence, «To be or not to be» becomes {'to'=2, 'be'=2, 'or'=1, 'not'=1}. This is what is referred to as a *Bag of Words* (BoW) representation. Since mathematical operations on sets are limited, we can now move to a one-hot encoding. Then, if 'to' = (1,0,0,0), 'be' = (0,1,0,0), 'or' = (0,0,1,0), 'not' = (0,0,0,1), «To be or not to be» = (2,2,1,1). Drawbacks for this representation include sparsity and high dimensionality. Note also that if it's not combined with pre-processing steps displayed in Figure 1.1, all the context of the words is lost, therefore the syntax and semantic too.

n-grams try to better integrate the context by combining words, where n is the number of words combined. The BoW representation can be considered as a n-grams representation where $n = 1$. As an example, a tri-grams ($n = 3$) representation of «To be or not to be» implies sequences 'to be or', 'be or not', 'or not to' and 'not to be'. One can easily see that the gain in context is making the sparsity and high dimensionality issues even worse.

[1]Taken from `https://medium.com/@ageitgey/natural-language-processing-is-fun-9a0bff37854e`

Word embeddings address many issues presented above by bringing the high dimensional sparse representation into a continuous vector space with a much lower dimension. It does so using the context, i.e. the words surrounding the one concerned by the embedding. Many techniques can be used, including Dimensionality Reduction, Co-occurrence Matrix [**22**] and probabilistic models. Word embeddings were introduced in neural networks by Bengio et al. at NIPS 2000 [**3**].

We choose to present a Neural Networks approach called Word2Vec [**31**]. Let's say you want to embed the t^{th} word of a sentence w_t. First, you specify a context c which is the number of words before and after w_t. You also fix the vector you want to have for the embedding d. The neural network has only one hidden layer of size d. Then, there are two approaches to choose from as presented in Figure 1.2. In the CBoW one, you input the c words before and the c words after w_t and you try to output w_t. In the Skip-gram approach, you feed w_t and learn to output the c words before and the c words after w_t. In general, CBoW converges faster while Skip-gram finds better representations.

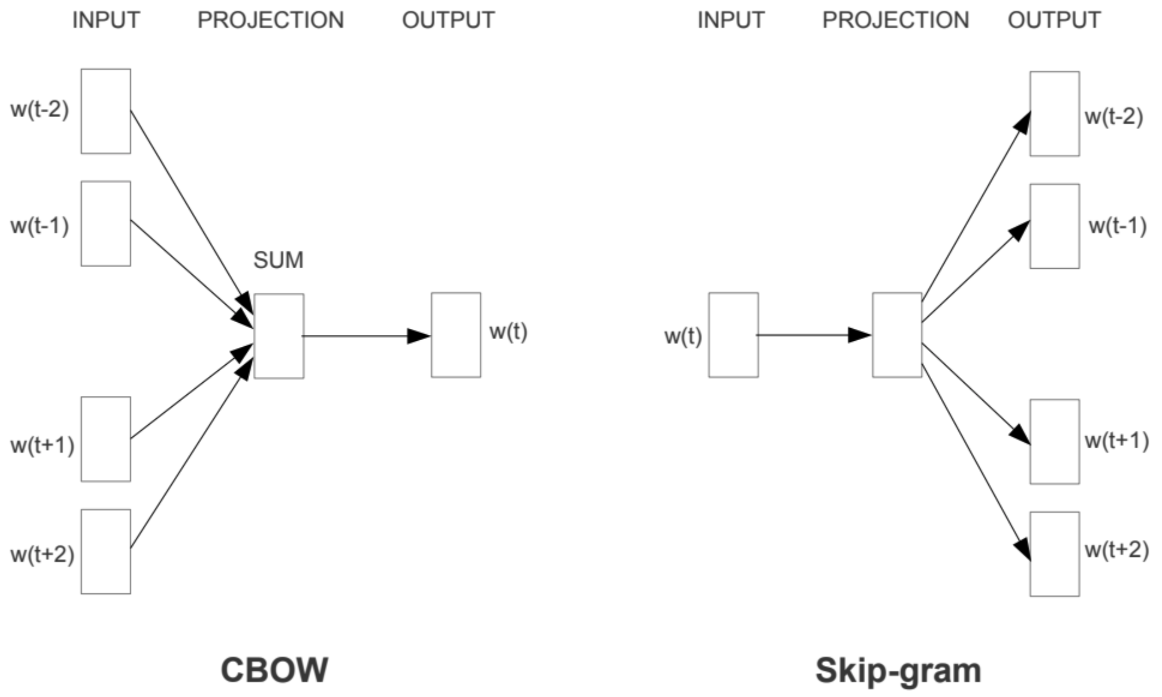

Fig. 1.2. Word2Vec. CBoW and Skip-gram approaches. The CBoW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. Image taken from [**31**]

It's worth noting that performing simple math operations on Word2Vec representations preserves some kind of logic. For example, it was observed that Word2Vec('king') – Word2Vec('man') + Word2Vec('woman') $\approx$ Word2Vec('queen').

Unfortunately, once the training completed, a word always has the same embedding (representation), whatever the context it's used in. We'll see in Section 1.3.3 that this is not the case for BERT's embedding. Still, one can use different NLP processing techniques (e.g. part-of-speech) to disambiguate the different meanings of a word.

Word and phrase embeddings, when used as the underlying input representation, have shown to boost the performance in numerous NLP tasks.

1.2.2. Models

Fundamentally, language is a sequential type of data. Words are formed by a sequence of letters, sentences by a sequence of words and documents by a sequence of sentences. The task to accomplish and the language representation you choose influences the type of models one can use. For example, BoW representations are of fixed length. This means that any traditional machine learning model can be used. But, if you want to process sentences and you are using word embeddings, the size of your input will vary. One way to manage this problem is to use sequential models.

A type of DNN known as Recurrent Neural Networks (RNN) has been used to treat sequential data. Because of vanishing or exploding gradient issues (the fact that the gradient tends to 0 or goes to infinity as it flows back in the sequence), new models like LSTM [**17**] and GRU [**7**] were created and alleviate these issues.

More recently, another type of DNN, transformers (introduced in Section 1.3.2), have been extensively used to process sequential data even though transformers are not specifically sequential models, they are more general, i.e. set of processing models.

1.3. Transformers for Language Modelling

1.3.1. Attention

Ever since its introduction, the attention mechanism [**2**] provided great results in modelling sequential data. Informally, the attention mechanism equips neural networks with the ability to focus on a subset of its features. Compared to recurrent models, it allows the modelling of dependencies without regard to their distance in sequences. Also, it expands the capabilities of neural networks by allowing the approximation of more complicated functions.

More specifically, in a traditional RNN encoder-decoder, for an input sentence represented by the sequence of vectors $x_1, x_2, ... x_T$, the encoder builds hidden states $h_1, h_2, ... h_T$ and the decoder uses the same context vector c for all predictions it makes. Usually, the context vector is the last hidden layer of the encoder (i.e. $c = h_T$). The addition of attention implies different context vectors c_i for each prediction. These context vectors are weighted sums over all the hidden states of the encoder.

Formally,

$$c_i = \sum_{i=1}^{T} \alpha_{ij} h_j$$

where $\alpha_{ij} = \text{softmax}[a(s_{i-1}, h_j)]_j$, s_{i-1} is the hidden state of the decoder producing word $i-1$ and a is a feedforward neural network jointly trained with all the other components of the system.

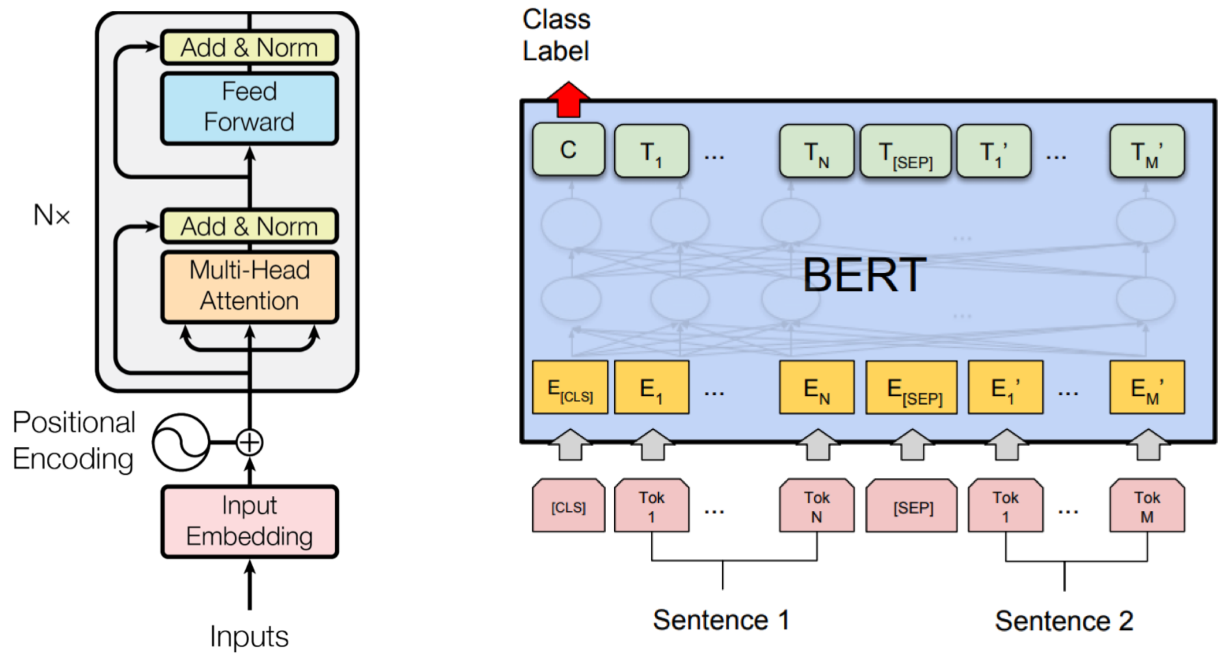

Fig. 1.3. On the left, transformer encoder as presented in [**46**]. Right side shows a BERT model used for pair classification task as presented in [**12**]

1.3.2. Transformers

At first, attention was incorporated as an additional feature to existing models. In contrast, transformers as presented in [**46**] are sequence to sequence models built on multi-head self-attention. They have an encoder-decoder structure. In this work, we solely use the encoder part, our goal being to find a dense representation of natural language text to make the recommendation, not to output a sequence. The encoder part is composed of a stack of layers. Each of these layers is composed of two different sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position wise fully connected feed-forward network. Residual connection around each of the two sub-layers, followed by layer normalization is also used as shown on the left of Figure 1.3 taken in [**46**].

Self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Learning long-range dependencies is a key challenge. An important factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and the output, the easier it is to learn long-range dependencies, hence the benefits of self-attention over recurrence.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

Since transformers contain no recurrence nor convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative position of the tokens in the sequence. To this end, they add positional encodings to the input embeddings at the bottoms of the encoder stack.

Finally, it's important to notice the huge speed benefit which comes from how transformers lend themselves to parallelization.

1.3.3. BERT

The architecture of BERT [**12**] is a multi-layer Transformer encoder based on the original implementation described in [**46**]. It is pre-trained on two unsupervised tasks in a bidirectional manner to obtain an encoded representation of a sequence of words. The first task is masked language modelling (MLM), where some tokens in the input sentence are masked and the goal is to predict those masked tokens. The second task is next sentence prediction (NSP) where two sentences are input and the goal is to predict if the second sentence follows the first one. This pre-trained version can be trained further on another dataset and/or another downstream task. This process is known as model fine-tuning.

To make BERT as versatile as possible on downstream tasks, the input representation is able to unambiguously represent both a single sentence and a pair of sentences in one token sequence, thanks to their embedding strategy. A lot can be said about their embedding strategy. Let's just say that it's composed of three parts: the token embedding, the positional embedding and a segment embedding. WordPiece [**53**] embedding strategy was used for tokenization. The positional embedding is managed as in [**46**]. Segment embedding are added to manage the fact that two sentences can be input in the model in conjunction with the special token [SEP] put in between tokens of both sentences. Figure 1.4 taken in [**12**] nicely shows this input representation strategy.

We should also mention another special token of importance, [CLS]. [CLS] is always the first token in BERT. It is mainly used for classification tasks. From the last hidden layer of

BERT, the [CLS] token is input in a fully connected layer which output is the size of the number of labels in the classification task. A Tanh activation is used. During the pre-training phase, it's trained through the NSP task to predict whether or not the two sentences input are following one another (output of size two). We will refer to it as the pooler output *pool*.

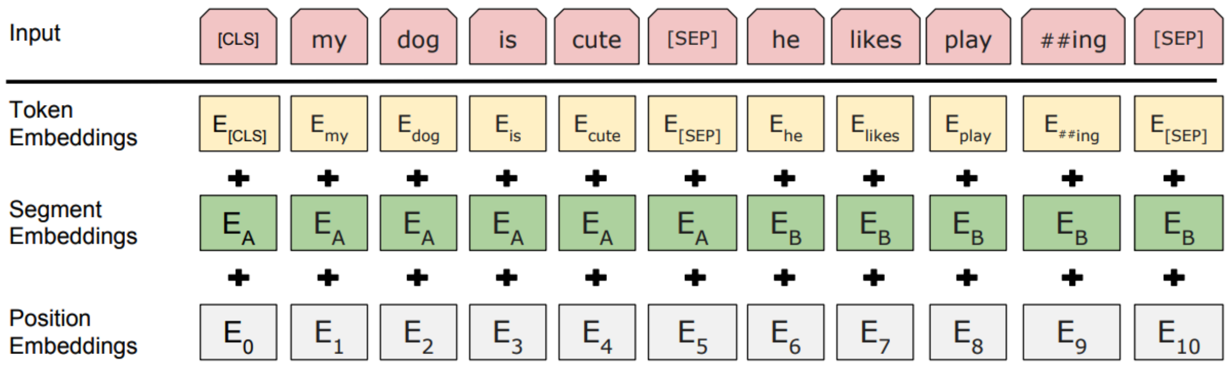
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Fig. 1.4. BERT input representation strategy. Image taken from [**12**]

Because of this specific set-up, BERT will provide *contextualized word embeddings* that will be different according to the sentence. For instance, whereas the vector for "running" will have the same Word2Vec representation for both of its occurrences in the sentences "He is running a company" and "He is running a marathon", they will differ if BERT is used.

Thanks to the Hugging Face group [**51**], pre-trained BERT models are easily available and can be fine-tuned for a variety of tasks, including sentence classification and readily available pooler output.

1.4. Recommender Systems

Recommender surface products in large online catalogues. Obvious examples of such catalogues include Netflix and Amazon.

Typically, to achieve such recommendations, a representation of the user (their preferences) or the item (referred to as their profile) or both is built. These representations are either based on the information available (content based) or past users' behaviour (Collaborative Filtering (CF)). Furthermore, these past behaviours can be either explicit (a rating provided by the user) or implicit (completed viewing of a movie or watched a preview, click or not,...) or a combination of both. Content filtering suffers from a lack of access to the information, need for domain knowledge and the difficulty to address some elusive or complex data aspects. Collaborative filtering alleviates most of these problems but suffers from cold start issues, that being the difficulty to represent new users or new items. Since there are downsides to both techniques, hybrid methods are also used.

1.4.1. Collaborative Filtering (CF)

Among various CF methods, one can find item-based neighbourhood [**11**] or Restricted Boltzmann Machines methods [**37**]. Matrix Factorization (MF) techniques are also very popular. They project both users and items into a shared vector space and estimate a user's preference on an item by the inner product between their vectors [**21, 55**].

Deep learning techniques tried to improve this conventional MF by estimating user preferences via Multi-Layer Perceptron (MLP) instead of inner product, as in Neural Collaborative Filtering (NCF) [**15**].

Other examples, inspired by the Auto-Encoder framework include CDAE [**52**] and AutoRec [**38**] where only the user representation is found and from it, ratings for all items are predicted. Note that this corresponds to the user-based AutoRec. It's also possible to build an item representation and predict ratings users would give to this item (item-based AutoRec).

1.4.2. Hybrid Methods

Numerous deep learning based methods seek to improve the recommendation performance by integrating the distributed item representations learned from auxiliary information into CF models. For example, this information can be found in knowledge bases specific to each domain and in forms as various as text [**20**], images [**48**] or acoustic features [**44**]. Moreover, propagating this information through graphs, either by convolution techniques [**43**] or attention mechanism [**49**] has been shown to boost performance and in some models, to explain recommendations [**50**].

1.4.3. Sequential Recommendation

Most methods above aren't taking into account the temporality of recommendations. Time effects can come from a variety of sources. Items may have a seasonality (boots), lose relevance as they age (news) or simply depend on the user's specific mood or need at this moment (interactive setup). Moreover, as shown in [**10**], setting the problem as a prediction of the next movie to watch instead of trying to predict one held out movie, as it is often the case in non-sequential settings, provides better performance. In that sense, a sequence of videos watched on YouTube has similarities with conversation patterns for movie recommendations.

Adding temporal biases to MF techniques showed interesting results. Deep learning techniques like RNN and its variants, Gated Recurrent Unit (GRU) [**7**] and Long Short-Term Memory (LSTM) [**17**], are used to model user behaviour sequences [**14, 24, 33**]. The idea is to encode a user's previous representation into a single vector and combine it with the actual state.

More recently, transformers have been used to model sequences in a bidirectional way as in BERT4Rec [**41**].

1.4.4. Conversational Recommendation

Traditional recommender systems work in a static way, estimating user preferences on items from past interaction history. This has a fundamental intrinsic limitation: the system is not able to capture a user's current preferences, which may have shifted away from their historical ones.

In conversational recommendation, the process becomes an interactive one through which users converse with the recommender using natural language. In this manner, a conversational recommender system should be able to obtain or infer the dynamic preferences of users from their utterances, which might consist of their direct or indirect descriptions of their needs or of answers to questions posed by the recommender system in a mixed-initiative setup. Such information can help the system to adjust its recommendation strategy effectively and enhance its performance.

The rapid development of recommender systems and dialogue agents in recent years provides an opportunity to help define how these two technologies can be combined. As we see, conversational recommendation incorporates a lot of cutting edge technology all dealing with technical difficulties, so the field has various approaches to tackle them.

Some of them take advantage of bandit approaches to balance the exploration and exploitation trade-offs for cold-start users in conversational recommendation scenarios [**56**]. Christakopoulou et al. [**8**] make the first step to adapt a typical bandit-based method like Upper Confidence Bound (UCB) and Thompson Sampling to the conversational recommendation scenario.

In a question driven approach, the estimation and utilization of a user's preferences towards attributes become a key research issue of asking an attribute-centric question. Zhang et al. [**57**] use a multi-memory network to estimate user's preferred attributes to consult the user from their profile and product description in an e-commerce scenario.

Our line of work is oriented on Dialogue Understanding so we focus on how to understand a user's preferences and intentions from their utterances. Many ([**18**], [**23**], [**26**]) combine this with Dialogue Generation, an important and complex field in itself. We decided to take one step at a time. Listen before you talk.

Chapitre 2

Recommendation From Dialogues

In this chapter, we will be focusing on dialogue understanding and the specific task of providing recommendations based on dialogues.

Data are discussed first. Then we tackle the models we use. Our experimental setup follows. Finally, we share and discuss our results.

Note that in order to limit duplication, this chapter uses several references to chapter 3, which presents the original version of the paper we submitted to the Conference RecSys 2020.

2.1. Data

2.1.1. Conversational Recommendation Data

One of the many difficulties to do research in this domain is the lack of publicly available data. This is why a lot of initiatives use machine generated text based on well-known recommendation datasets, like MovieLens. The Facebook Research team, as part of The bAbI project, proposes The Movie Dialogue dataset as introduced in their paper [**13**]. Another example of a synthetic language dataset is created in the paper Conversational Recommender System [**42**], which mainly focuses on attributes of items.

To the best of our knowledge, the first dataset with human generated dialogues and recommendation is ReDial, as introduced [**25**]. ReDial was used in the framework of KBDR [**6**] in and provided state-of-the-art results on both recommendation and dialogue generation.

Because our work is specifically focused on the task of making recommendations based on natural language, we solely used the ReDial Dataset. Machine generated text, as it's done in [**13**] for example, has a strong repetitive structure that doesn't correspond to the goal we are trying to achieve.

More recently, the dataset GoRecDial [**19**] is available through the Parlai framework from Facebook. Due to time restrictions, we have to leave experiments on this dataset for future work.

2.1.2. ReDial Dataset

The ReDial Dataset[1] [**25**] was created to allow research at the intersection of goal-directed dialogue systems (such as restaurant recommendation) and free-form (also called “chit-chat”) dialogue systems.

In the dataset, users talk about which movies they like and which ones they do not like, which ones they have seen or not etc. It contains a total of 11,348 conversations, 10,006 for training and model selection and 1,342 for testing.

One person in the conversation is the *recommendation seeker* and the other is the *recommender*. The movie seeker has to explain what kind of movie he/she likes and asks for movie suggestions. The recommender tries to understand the seeker’s movie tastes and recommends movies. All exchanges of information and recommendations are made using natural language.

Each conversation contains roughly ten messages minimum and at least four different movie mentions. In addition, every movie mention is tagged using the ‘@’ symbol, which simplifies extraction (during data gathering, users were asked to tag movies after each conversation).

ReDial also comes with movie dialogue forms (*ReDial’s forms*). They include answers from both seeker and recommenders to the following questions:

(1) Whether the movie was mentioned by the seeker or was a suggestion from the recommender (“suggested” label)
(2) Whether the seeker has seen the movie (“seen” label). One of: Seen it, Haven’t seen it or Didn’t say
(3) Whether the seeker liked the movie or the suggestion (“liked” label). One of: Liked, Didn’t like, Didn’t say.

Ideally, seeker and recommender have the same answers, but it is possible that their answers do not coincide due to the carelessness of the participants or to the dialogue ambiguity.

The liked/disliked/did not say label is highly imbalanced, 94% of ratings correspond to liked movies. This is standard for recommendation data [**30**], since people are naturally more likely to talk about movies that they like and the recommender’s objective is to recommend movies that the seeker is likely to appreciate.

An example of a conversation in ReDial is in Table 2.1.

2.1.3. Knowledge Base

Relevant knowledge bases have been useful in various machine learning setups [**1, 28**].

[1]available at `https://redialdata.github.io/website/`

SEEKER:	hi ! i 'm looking for an action filled movie similar to jurassic park
RECOMMENDER:	ok , have you seen jurassic world ?
SEEKER:	yes i loved that movie as well . are there any good movies without dinosaurs ?
RECOMMENDER:	have you seen king kong ?
SEEKER:	yes ! that was another great giant monster movie :)
RECOMMENDER:	what about jaws & amp ; jaws 2 ? oldies but goodies scary action ? suspense gives me the rumblies
SEEKER:	i absolutely loved the first one . one of my favorite movies ever :) honestly i can't say i remember much about the second one . jaws 3-d was quite funny and jaws : the revenge was just weird . i do love suspense too ...
RECOMMENDER:	i like suspense but sometimes i can barely handle it ! it gives me anxiety ! lol
SEEKER:	that 's understandable . as long as it 's not too gory i don't generally have a problem with those things .
RECOMMENDER:	well , it was great chatting with you ! have a great one !
SEEKER:	you too ! thanks !

Table 2.1. Example of conversation in ReDial's dataset

To build a knowledge base relevant to our work, we used information from the Internet Movie Database (IMDb). We accessed the data using the IMDb python package[2]. For every movie in ReDial, we match the title and the release date to retrieve IMDb_id, genres, actors, directors and plot.

2.2. Modelling

In this section, following a brief introduction, we first present our baseline models, both using extracted entities from text. We then move on to models using full text as input.

[2]https://pypi.org/project/IMDb/

Finally, we discuss the output of the models being the ratings for all the items at once or only for one item.

As a summary, you can find a visual of all models studied in Figure 3.1 as well their mathematical description in Table 3.1.

2.2.1. Overview

Traditional recommendation engines do not rely on inputs in the form of text (never mind dialogue) to make recommendations. Likewise, both of our baseline models use historical ratings of a user and try to predict their future ratings.

The *Basic Encoder Decoder* (BED) model simply takes the item's rating as inputs. Note that we use ReDial's forms to get the ratings.

The *Knowledge Base Augmented Encoder Decoder* (KBAED) model augments the input using the item's attributes. To do so, we extract the attributes mentioned in the seeker's dialogue (see Section 2.3.1.2) and identify the items related to the extracted attributes (using the knowledge base). The items identified this way are then augmented. Implementation details are in Section 2.2.2.

The goal of conversational recommendation is to directly use all the information in the text to make predictions. The four models we propose (BERT U, BERT MF Fixed, BERT MF Learned and BERT UnV) all proceed this way. The first step is always to find a dialogue embedding, which we use BERT for. In most cases, the dialogue embedding we use is the one corresponding to the [CLS] token. We also ran some tests using the average of all the tokens from the last layer of BERT (*avg*).

When the time comes and the model has to make predictions, there are two main options.

If the representation before the prediction corresponds to the user, creating a decoder that will learn to predict the ratings of all items will make inference efficient. The major downside of this method being its impossibility to predict a rating for new items. It's the approach we used for BED, KBAED and BERT U.

On the other hand, if the learned representation is a combination of both the user and the item, you are left with predicting a single rating, the one this user would have for this item. This is the approach you'll find in the BERT MF Fixed, BERT MF Learned and BERT UnV.

2.2.2. Baselines

Both models in this category, BED and KBAED are strongly inspired by AutoRec [**38**], an autoencoder model for rating prediction, and adapted for sequential recommendations. This implies that the ratings we use as input are only those mentioned in the conversation up to the point were the next item will be recommended. The output is the predicted ratings for all items. The only *observed* items are the item that will be further recommended in the sequence.

Formally, we consider a sequence of T events, indexed $\{1,...,T\}$, a set of users $\mathcal{U}$, a set of items $\mathcal{V}$ and a partially observed Rating Matrix a time t, $\mathbf{R}^{1:t} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}|}$, which includes all ratings from timesteps 1 to t. To lighten the text, the superscript referring to timesteps will be omitted when not significant.

Our loss function differs from the one proposed in AutoRec [**38**]. Since we only have 0 and 1 ratings, we used the binary cross entropy (BCE) to evaluate our error instead of Mean Square Error (MSE). But, as in AutoRec, while training, we only update the weight related to observed items. For this reason, we use the *Masked* Binary Cross Entropy (MBCE). Its value is the same as BCE for *observed* items, 0 elsewhere.

Finally, we needed to manage the strong imbalance of ratings in ReDial: 94% of ratings are 1 (liked) and only 6% are 0. This corresponds to our intuition that people tend to talk about movies they like and want to see more than those they didn't like. Without any adjustment, the model could easily learn to predict 1 for all items. There are multiple ways this can be handled. We used a softmax on the output, forcing the model to favour a small subset of items. This technique was also used in [**27**] and has shown the best results in our set-up.

Considering a user u at time t and its processed input $\mathbf{x}_u^{1:t}$ (which depends on the model used, details in each model's section), we formally have that both models solve:

$$\min_{\theta} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}, t \in \{1:T-1\}} \mathrm{MBCE}(\mathbf{y}^{t+1} - \mathrm{softmax}(\mathbf{W}_d \cdot \mathrm{Encoder}(\mathbf{x}^{1:t}) + \mathbf{b}_d)) \tag{2.2.1}$$

where θ includes $\mathbf{W}_d$ and $\mathbf{b}_d$ (the trainable parameters of the Decoder) plus the trainable parameters of the *Encoder*, $\mathbf{y}^{t+1}$ is the vector of item ratings at time $t+1$ and $\mathcal{D}$ is the dataset. Note the values of $\mathbf{y}^{t+1}$ of items that won't be mentioned in the next sequence are undetermined, but it's not an issue since we use MBCE, they are masked in the objective.

The only difference between the two baseline models (BED and KBAED) is the *Encoder*$(\cdot)$ part, i.e. the way we encode the user u at time t to find its latent representation.

2.2.2.1. Basic Encoder Decoder (BED) (Figure 3.1a top). For the BED model, we use as input the ratings of the observed movies. As a reminder, during the ReDial data collection, after all conversations, both users were asked to extract all ratings available in the conversation (ReDial's forms, see §2.1.2). Here, we use these extracted ratings.

Formally, following Equation 2.2.1, we have that user u at time t can be represented as $\mathbf{x}_u^{1:t} = \mathbf{R}_u^{1:t} = \mathbf{R}_{u,1}^{1:t}, ..., \mathbf{R}_{u,|\mathcal{V}|}^{1:t}$ and:

$$\mathrm{Encoder}(\mathbf{x}_u^{1:t}) = \mathrm{ReLU}(\mathbf{W}_e \mathbf{x}_u^{1:t} + \mathbf{b}_e) \tag{2.2.2}$$

where $\mathbf{W}_e$ and $\mathbf{b}_e$ are the trainable parameters of the *Encoder* (included in θ of Equation 2.2.1). For the details of how we built $\mathbf{R}$, see the Rating Matrix Section 2.3.1.1.

2.2.2.2. Knowledge Base Augmented Encoder Decoder (KBAED) (Figure 3.1a). With KBAED, one goal is to alleviate the well known user cold start issue of collaborative filtering, while keeping a similar structure. Hence, we added a content portion to the user's input. Content information typically comes from a knowledge base of items and their associated attributes (for movies we use as attributes: genres, actors, directors, plots, etc.). In order to consider attributes mentioned by the user u at time t, we introduce a vector of *attributes mentioned* $\mathbf{A}_u^{1:t} \in \mathbb{R}^{1\times|\mathcal{V}|}$ (see §2.3.1.2 for details on how we established attributes mentioned). This vector is filled with zeros, except at index corresponding to items which attributes include *all* attributes mentioned by user u at time t. For example, if at some point, attributes mentioned by the seeker are "comedy" and "action", all indices corresponding to movies having attributes "comedy" and "action"(and maybe more) are set to 1.

Formally, if $\mathcal{A}_u^{1:t}$ represents the set of attributes mentioned by user u at time t and $\mathcal{A}_v$ the set of attributes associated with item v in the knowledge base, we have that $A_{uv}^{1:t} = \mathbf{1}_{\mathcal{A}_u^{1:t} \subseteq \mathcal{A}_v}$, for all $v \in |\mathcal{V}|$, where $\mathbf{1}$ is the Indicator function.

Depending on the attributes mentioned by user u, the number of ones in this vector can vary a lot. For example, there are many more comedies than there are western movies. To mitigate the unbalanced quantity of ones in vector $\mathbf{A}_u^{1:t}$, we define the attribute input vector $\mathbf{a}_u^{1:t} = \parallel \text{top}_{100}(\mathbf{A}_u^{1:t} * \mathbf{p}) \parallel$, where $\parallel \cdot \parallel$ is the norm, the top_{100} function zeroes out entries that correspond to movies outside of the top 100 most popular ones and $\mathbf{p}$ is a popularity of movies vector defined by the sum of ratings from all users in ReDial, formely, $\mathbf{p} = \sum_{u \in \mathcal{U}} \mathbf{R}_{u,1}^{1:T}, ..., \mathbf{R}_{u,|\mathcal{V}|}^{1:T}$.

Hence, for the KBAED model, at time t, we augment our user u input with $\boldsymbol{\alpha}\mathbf{a}_u^{1:t}$, yielding

$$\mathbf{x}_u^{1:t} = \mathbf{R}_{u,1}^{1:t}, ..., \mathbf{R}_{u,|\mathcal{V}|}^{1:t} + \boldsymbol{\alpha}\mathbf{a}_u^{1:t}$$

where $\boldsymbol{\alpha} \in \theta$ is also vector of trainable parameters. The rest of the *Encoder* is as in Equation 2.2.2.

2.2.3. Language Models for Recommendations from Dialogue

We now move our attention to our four main models. Each uses natural language input. We introduce m to denote the text of a user or an item.

At time t, all the *messages* (m) exchanged, in text, between a seeker and a recommender are $m_u^{1:t}$. We include seekers and recommenders messages in this representation since it is impossible for the model to know which movie the seeker referred to in a case like: "Yes, I really liked that one." Please see §2.3.1.3 for more all details regarding $m_u^{1:t}$.

For an item v, we use m_v, which is not time dependant. However, we use different textual representations of the movies so a superscript helps distinguish them. For example, an item having a full representation f (using all the attributes in the knowledge base) is noted as m_v^f. The details are presented in §2.3.1.4,

One limitation of our baseline models (and BERT U, to be introduced) is that they cannot recommend new (cold-start) items. As a remedy we introduce three models (BERT MF Fixed, BERT MF Learned and BERT UnV) all based on Matrix Factorization (MF). MF learns a representation for the user and one for the item and combines them to output a scalar interpreted as their associated rating. Formally, it solves:

$$\min_{\theta} \sum_{(u,v)\in\mathcal{D}, t\in\{1,\ldots,T-1\}} BCE(\mathbf{R}_{u,v}^{t+1} - \sigma(\text{Combine}(m_u^{1:t}, m_v))) \tag{2.2.3}$$

where θ includes all trainable parameters and σ is the sigmoid function. The *Combine* function, $m_u^{1:t}$ and m_v vary from model to model and are specified in the sections introducing each of them.

Here again, we need to manage the strong imbalance of ratings in ReDial: 94% of ratings are 1 (liked) and only 6% are 0, but cannot use the softmax technique. We revert to the common practice of negative sampling, i.e. impute ratings of 0 to a random subset of unobserved items [**15, 54**]. We refer to this subset of ratings randomly set to 0 as *random ratings* and the number of random ratings is R^-. The random ratings are then included in the rating matrix $\mathbf{R}$.

2.2.3.1. BERT U (Figure 3.1b). BERT U is an interesting mix. It also solves Equation 2.2.1, as our baseline models, but uses text as input.

To manage the text, as it is commonly done, we leverage transfer learning from a pre-trained BERT model and fine-tune it. Specifically, we add a fully connected layer on the pooler output (*pool*, see BERT description §1.3.3) and train all the parameters with the ReDial dataset. In our case, the fully connected layer goes from dimension 768 (token size in BERT base) to 6924 (the number of movies mentioned in ReDial's dataset). As with both baselines models, the output is the predicted ratings for all items.

Hence, referring to Equation 2.2.1, our processed input $\mathbf{x}_u^{1:t}$ of a user u at time t simply becomes $m_u^{1:t}$ and

$$\text{Encoder}(\mathbf{x}_u^{1:t}) = \text{pool}(\text{BERT}(m_u^{1:t})).$$

Finally, note that we can also instantiate a BERT V model which takes as input a natural language representation of the items, for example m_v^f, and tries to predict the ratings all users would give this item.

2.2.3.2. BERT MF Fixed (Figure 3.1d). For BERT MF Fixed, we are inspired by MF techniques, more specifically the neural collaborative filtering model [**16**]. We concatenate the BERT representations of a user and an item and input them to a neural network that predicts their rating. Each representation is the pooler output of optimized (fixed) BERT U and BERT V models.

Hence, following Equation 2.2.3:

$$\text{Combine}(m_u^{1:t}, m_v) = \text{MLP}[\text{pool}(\text{BERT U}^*(m_u^{1:t})) : \text{pool}(\text{BERT V}^*(m_v))].$$

.

2.2.3.3. BERT MF Learned. In BERT MF Learned, we go back to the basic approach of matrix factorization where a simple dot product combination of a user and an item representation is used to predict their rating. This forces all the learning to take place in the representation learning. We use two BERT models, one for the users (BERT) and one for the items (BERT') and combine their output with a dot product.

Also, we use the average (avg) of the last hidden layer in BERT, not the pooler output. Average yielded better empirical results (see Table 3.3) as suggested in the documentation of the Hugging Face group [**51**].

Again, referring to Equation 2.2.3:

$$\text{Combine}(m_u^{1:t}, m_v) = \text{avg}(\text{BERT}(m_u^{1:t})) \cdot \text{avg}(\text{BERT'}(m_v)).$$

2.2.3.4. BERT UnV. BERT UnV is inspired by the Next Sentence Prediction (NSP) task presented in Figure 1.3, where two sentences are input in BERT and the model predicts if they come one after the other. We adapt it so that both sentences input are now the natural language representations of a user and item and the goal is to predict the rating associated with them at time t. As in the NSP setup, a [SEP] token is added between the user and item representation. Finally, an affine transformation is added on top of BERT's pooler output to predict the particular user-item rating.

Therefore, in combination with Equation 2.2.3, the model is

$$\text{Combine}(m_u^{1:t}, m_v) = \mathbf{w}[\text{pool}(\text{BERT}(m_u^{1:t} : [\text{SEP}] : m_v)] + b.$$

2.3. Experiment Set-up

We now discuss the specifics of how we tested our models using the ReDial dataset.

2.3.1. Data Pre-Processing

In all our experiments, it is important to keep in mind that the task is one of sequential recommendation.

We split every conversation in ReDial into timesteps. We segmented the conversations such that each time step ends by the recommender mentioning a movie (or more). Note that

Table 2.2. Descriptive statistics of the sequential ReDial data.

	Conversations	Timesteps	Target movies
Train	9,006	23,499	26,872
Valid	1,000	2,569	2,958
Test	1,342	3,177	3,646
Total	11,348	29,245	33,476

it is possible to have multiple new movies mentioned in the recommender's next message. Recall that we only considered movies for which we have a rating from the ReDial's forms. In particular, there are 6,924 movies discussed in ReDial but only 5,908 of these have at least one associated rating (from movie forms).

In summary, each conversation has multiple timesteps. Each time step has at least one target movie but can have more since the recommender can mention multiple movies in a single message. Table 2.2 provides descriptive statistics of our resulting train/validation/test splits.

An important thing to consider in regards to our performance is whether or not the rated movie is in the training set. There are 5,281 movies that are discussed in the train set, either as an input or a target and for which we have a rating, For the test set, 803 movies have never been seen in the train set, 299 of which are targets. Note that it is also possible to consider unseen movies like new movies release in real life, relating them to the cold-item case.

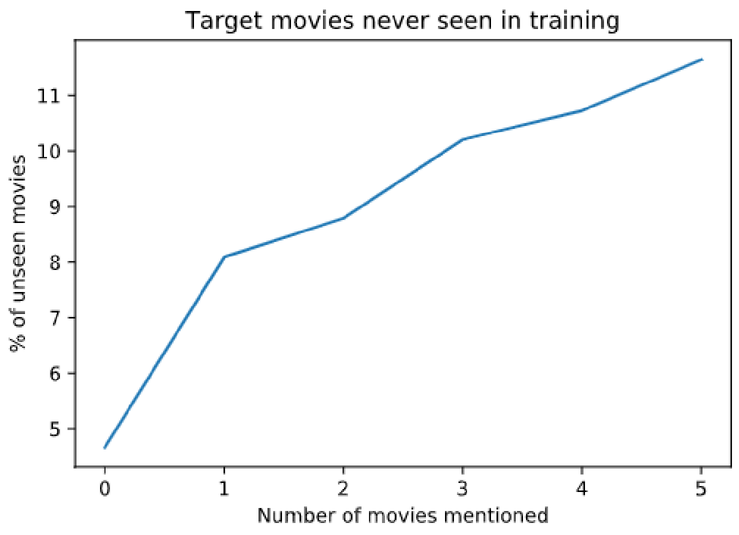

Fig. 2.1. Test set statistics. The x-axis represents the number of movies mentioned in the input. The y-axis represents the percentage of target movies that were never seen during training.

Furthermore, Figure 2.1 presents the percentage of target movies that were never seen during training, as a function of the number of movies mentioned in the input. We see that there is an important increase further down in the conversation when there are more movies mentioned in the input. Our intuition is that at the beginning of a conversation with little knowledge of the seeker's preferences, the recommender suggests more popular, well-known or general movies. As the recommender better understand what the seeker wants, they can provide more targeted, specific, and so more unseen movies. This has important implications for the performance of our models through time. Predicting movies unseen in the train set is hard. This may explain the decrease in performance as the number of movies mentioned in the input increases (see Figure 3.2b). We will further discuss this issue in Section 2.4.2.

2.3.1.1. Rating Matrix. We build our rating matrix $\mathbf{R}$ using the ratings available in the ReDial's forms (§2.1.2). Users were asked to rate the movies that were mentioned in the conversation as liked (1), disliked (0) or don't know (2).

We used only observed ratings to build $\mathbf{R}$ and so discarded the "don't know." This causes a change in the number of movies mentioned for different models. For models based on BERT, it is possible to use all movies mentioned, even if they were not rated in the ReDial. But, for BED and KBAED, it is only possible to input movies with an explicit rating.

To denote ratings at a particular time step t we use $\mathbf{R}^t$. We use $\mathbf{R}^{t_1:t_2}$ to denote all observed ratings from t_1 to t_2, inclusively.

2.3.1.2. Attributes. For simplicity, when considering attributes mentioned by a user $\mathbf{A}_u$ we limited ourselves to the genre attribute, as it is the most common one in the ReDial conversations. We used an exact string match to find them.

2.3.1.3. User text. For a user u at time t, what we define as its representation in natural language, $m_u^{1:t}$, is the concatenation of all the messages that were exchanged between the seeker and the recommender up to time t.

BERT is pre-trained with different sentences having different segment tokens (recall Figure 1.4). We decided not to use these segment tokens as they were used only with one change of speaker: one to start and another one that ends. Since our conversation can go on for multiple rounds, we opted for a simple solution where we added "S::" before each message by the seeker and "R::" in front of messages by the recommender. Experiments showed better performance when these separators were added. An example is: "S:: Hello, I'm looking for movie recommendations R:: Hey! Happy to help. What kind of movies do you like S:: I'm really into action movies. I loved The Matrix R:: Who about superhero movies? S:: Not that much into superheroes."

2.3.1.4. Item text. For an item v, we define its natural language representation m_v as information that we have about it in the knowledge base. The information is encoded as a single string. We used three different subsets of information. First, the short one, m_v^s, contains the movie title, its genre and the three leading actors. The medium one m_v^m is as m_v^s, but with all the actors. Finally, m_v^f corresponds to all information in the movie available in the knowledge base.

Using the movie "The Matrix" as an example, we would have the following text for its medium representation: "The Matrix. Genres: ['Action', 'Sci-Fi']. Actors: ['Keanu Reeves', 'Laurence Fishburne', 'Carrie-Anne Moss', 'Hugo Weaving', 'Gloria Foster', 'Joe Pantoliano', 'Marcus Chong', 'Julian Arahanga', 'Matt Doran', 'Belinda McClory']."

2.3.2. Metrics

In all experiments, we used *recall@k* and *NDCG@k*. Both are standard for the evaluation of recommender systems.

We used recall@k since it is used in other papers on conversational recommender systems and thus and allows us to compare our work. Recall@k is the number of items retrieved among all the items there are. Since we are in a setup where we only consider the next movie recommended, we have that recall@k is 1 if the item recommended is in the first k positions and 0 if not. We multiplied by 100 to get percentages.

The problem with *recall@k* is that it gives equal importance to a movie regardless of its actual ranking (as long as it's below k). In most interfaces items ranked closer to the top have much more importance. for example, in an interface that would show ten movies at a time, movies recommended in 27^{th} position have barely any chance compared to those in the top 10 spots.

Normalized discounted cumulative gain (NDCG) is a more complete metric. It is based on the discounted cumulative gain (DCG) that not only accounts for the presence of items, but also for their rank and the relevance of the item. The value is reduced logarithmically proportional to the position of the item

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}$$

where

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

and with k the max rank considered and rel_i the relevance of item at rank i. In our case, the relevance is 1 for a good movie recommendation (the next one to be recommended), 0 if not. The ideal discounted cumulative gain (IDCG) is just the DCG of the best possible solution

(IDCG normalized DCG), which in our case corresponds to the next movie recommended to be in the first position.

2.4. Results

Generally, our results are presented from two points of view: the overall performance of the models and their performance through time. We then study the specific results in regards of user cold start, items cold star, the different latent representations, negative sampling implications and time considerations.

When considering the overall performance, we look at recall@1, recall@10, recall@50 and ndcg@100. The scores for all our models can be found in Table 3.2 and general discussion about it in Section 3.5.1.

To better understand the impact of time, Figure 3.2b presents all models using recall@50 scores (y-axis) in relation to the number of movies mentioned (x-axis). Note that KBRD results were taken directly from [**6**]. To our knowledge, our setup is the same as theirs and so the results are comparable.

2.4.1. User Cold Start

Figure 3.2b reports the performance of all models according to the number of movies mentioned. To analyze the user cold start issue, we focus on 0 or 1 movie mention, as a completely new user has no movie mentions.

For our baseline model BED, where only movie mentions are input, this corresponds to the best guess of the model for no input (probably related to the popularity of the first movies recommended in our database). KBAED can have no movie mention input, but still have a genre input, which explains the relative gain over BED. Furthermore, models using the complete text available through different implementations of BERT show improved performances since they use much more information. For example, they can use all the mentions of movie's attributes, establish a connection between words like "comedy" and "funny", differentiate between "I like this" or "I never want to see this again".

After two movie mentions, the information provided by full text usage in the user's input provides less differentiation with our baseline than at the beginning. This is to be expected since at that stage the models benefit from collaborative filtering in the composition of their latent representations.

2.4.2. Item Cold Start

To understand the impact of item cold start, we consider the ability of models to predict movies that were never seen in the train set. (See Section 2.3.1 for a discussion of cold-start movies.)

First, we consider the prediction performance overall timesteps. In Table 3.2, the scores in parentheses are the performance of the models only considering movies never mentioned in training. We observe that the impact of unseen movies is more important on models employing only user information as input (BED, KBAED, BERT U) with a score of 0 across all metrics. There is no way for these models to know about new movies (even though they might be similar to other movies and the users would like them). On the other hand, models based on MF combine user and item information for prediction. Hence, a movie may never have been seen in the train set, but its representation can still be inferred as long as it has similar attributes to movies that were seen during training. Hence, these models have some information about the unseen movies through their attributes and they can infer some relations between movies and even recommend unseen ones. Table 3.2 shows better performance for the MF models.

Now, consider the cold-start issue as it relates to time. Remember that the percentage of movies that were never seen during training grows with time (Figure 2.1). This growing ratio partly explains why the performance decreases over time (Figure 3.2b). Even if the model has more information about the user in later timesteps, the task might become harder because it has to predict more unseen movies.

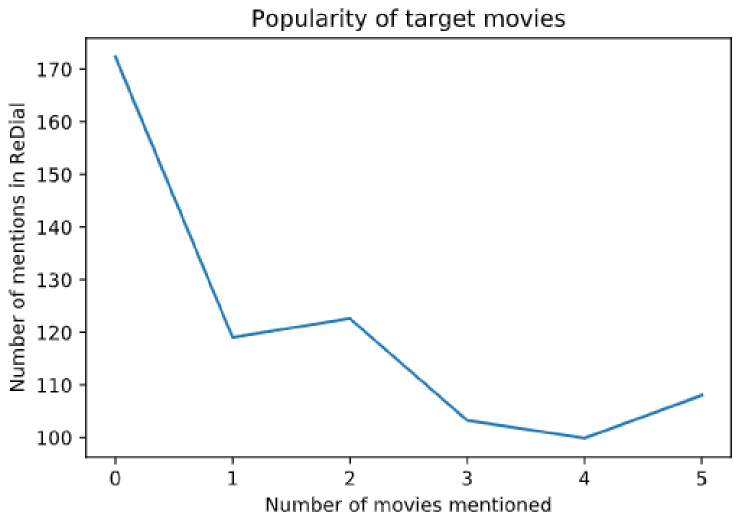

Fig. 2.2. Test set statistics. The x-axis represents the number of movies mentioned in the input. The y-axis represents the average number of time target movies were mentioned in the ReDial dataset.

The rest of the performance decrease can be explained by the fact that in ReDial, movies mentioned at beginning of conversations are, on average, more popular. But, as the conversation progresses recommendations are better informed and become more specific, hence harder to predict in absolute terms. Figure 2.2 shows the decrease in popularity of target movies as time goes by.

2.4.3. MF: Pooler Output VS Last Hidden layer Average

We now compare the two different outputs from BERT that can be used as a latent representation. For so, we stick to BERT MF Learned models, with 40 negative samples ($R^- = 40$). Results in Table 3.3 suggest that averaging the last hidden layer of BERT outperforms the usage of the pooler output.

Note that the difference between BERT MF Learned results in Tables 3.2 and 3.3 is caused by using i_f in the former and i_m in the latter, emphasizing the importance of using all available item features.

2.4.4. Impact of Random Negative Samples

We now move our attention to the impact of the number of random negative samples. Figure 3.2a shows the evolution of our metrics as a function of R^- (x-axis). Since scale varies by metric, we present four different graphs. For all, we used BERT UnV models.

We see a clear benefit in augmenting R^- for Recall@1. The same goes for Recall@10 at least until after 20 R^-. The story differs for Recall@50 whose performance decreases after 20. NDCG@100, accounting strongly for top ranking predictions but also being affected by last tier ranking, shows mixed results. Overall, it seems like having more negative examples clarifies the top choices the model should make (steady increase for recall@1 and recall@10), but we can imagine that this gain in specificity has a cost over popular choices, hence the decrease for recall@50 and NDCG@100 for $R^- > 20$.

To clarify this first impression, future work should include exploration with a greater number of random negative samples, even though training time is seriously impacted, as we will see in the following section.

2.4.5. Time Considerations

Fine-tuning a BERT model can be resource and time consuming. Table 3.4 displays time and resource consumptions. For BERT MF and BERT UnV, training time depends on the number of random ratings (RR). We present results for 20 RR. It is also important to remember that BERT MF Learned needs to learn 2 BERT models, one for users, one for items, which impacts memory consumption, forcing us to reduce batch size and increase training time.

Prediction and ranking (for example, at test time) are also important to consider resource wise since there are important variations between models. Let us consider making predictions for $|\mathcal{U}|$ users and $|\mathcal{V}|$ items. Remembering that BERT is composed of thousands of feed-forward networks and millions of dot products, we assume negligible time to go through our MLP and executing dot products, focusing solely on the resources needed to go through BERT once, denoted B_r. For BERT U, a single pass of a user conversation provides rating predictions for all items, hence a cost of $|\mathcal{U}|B_r$. BERT MF Fixed first needs representation of all users and all items from BERT models and are then combined through the MLP, hence $|\mathcal{U}|B_r + |\mathcal{V}|B_r$. The same goes for BERT MF Learned since we assume MLP and dot product time to be negligible. With BERT UnV, each user-item pair has to be processed independently by BERT which results in a $|\mathcal{U}||\mathcal{V}|B_r$ complexity. These results appear in the last column of Table 3.4.

Chapitre 3

Paper 1 - Conversational Recommendation: Listen Before You Talk

Authors: Nicholas Vachon, Christopher Pal, Laurent Charlin.
Submitted to Conference: The ACM Conference Series on Recommender Systems 2020.
Contribution: Implemented the models and ran the experiments. I wrote the original draft and contributed to its polishing.

3.1. Introduction

Conversational recommendations encapsulate several machine learning sub-tasks including preference elicitation, item recommendation, and language generation.

In this work, we focus on item recommendation as the core task for a conversational recommender system and we study methods for the task of item (in our case, movie) recommendation from conversation.

This task comes with challenges, for example, a user may express preferences about items and their attributes (I enjoy action movies such as Wonder Woman). Further, the dialogue component introduces challenges beyond standard settings like recommendations from user reviews [**32**], since models have to understand dialogue dynamics. Fully interactive dialogue systems allow models to also alter dialogue dynamics; however, here we focus on the retrospective analysis of recorded dialogues to evaluate model performance in a controlled manner.

To explore this setting, we use, adapt and extend state-of-the-art transformer based neural language modelling techniques to the task of recommendation from dialogue. We study the performance of different methods using the ReDial dataset [**25**], a conversational-recommendation dataset for movies. We also make use of a knowledge base of movies and measure their ability to improve performance for cold-start users, items, and/or both.

We find that *pre-trained transformer models outperform non-transformer baselines even if the baselines have access to the user preferences manually extracted from their utterances.*

3.2. Transformers for Language Modeling

We provide a succinct review of transformer models focusing on the BERT model [**12**] which we use in this paper. The architecture of BERT is a multi-layer transformer encoder based on the original implementation described in [**46**]. It is pre-trained on two unsupervised tasks to obtain an encoded representation of language. The first task is masked language modelling (MLM), where the goal is to predict tokens that were masked from the input sequence. The second task is next sentence prediction (NSP) where two sentences are input and the goal is to predict if the second sentence follows the first one. Pre-trained BERTs can then be used for downstream tasks, either in a feature-based manner or by fine-tuning, fine-tuning showing better results.

In this work, we use two outputs from BERT. First, we used the [CLS] token, which is always the first token in a BERT model. We refer to this output as the pooler output (*pool*). Second, we averaged the tokens across all positions (*avg*) in the last hidden layer.

3.3. Recommendations from Dialogues

Problem definition. Our task is item recommendations for text-based conversational recommender systems. A conversation consists of a sequence of utterances (a user utterance followed by a system utterance) indexed $\{1,\ldots,T\}$. The system can recommend an item i after each user u's utterance. The task then consists of predicting user preference or a rating R_u^{t+1} at time $t+1$ given a conversation $c_u^{1:t}$ up to utterance t. In a complete system, this predicted rating would then be used downstream to compose the next utterance.
Data. We assume that we have access to the text of each conversation which we denote $c_u^{1:T}$. Each utterance is prefixed by a special token to denote the corresponding speaker's role ("S::" for the user seeking the recommendation and "R::" for the recommender).

We also assume access to a user's preferences or rating vector $\mathbf{R}_u^{1:t} \in \mathbb{R}^{|\mathcal{V}|}$ extracted from the dialogues up to utterance t (for example, labelled by the users at the end of the dialogue). In practice, the data we study has binary labels (liked/disliked), but our models generalize beyond that.

In addition, to model cold-start items, some models will use item features. We assume access to a knowledge base of movies from which we can extract these features and encode them as text. We denote the resulting string as i_f. Further, in conversations, users can also express their preferences toward item attributes. Such preferences can be extracted. For example, in our target dataset users often mention movie genres they like. We encode the attributes a user mentions in a sparse vector $\mathbf{A}_u \in \mathbb{R}^{|\mathcal{V}|}$ (or $\mathbf{A}_u^{1:t}$ to denote available attributes following utterances 1 through t) where non-zero entries correspond to movies that have all attributes mentioned by user u. If $\mathcal{A}_u$ represents the set of attributes mentioned by user u and $\mathcal{A}_i$ the set of attributes associated with movie i in the knowledge base, we have that, for i $\in \mathcal{V}$, $A_{ui} = \mathbf{1}_{\mathcal{A}_u \subseteq \mathcal{A}_i}$, where $\mathbf{1}$ is the indicator function.

3.4. Modelling

We use and develop several models for the task of recommendations from dialogue utterances. We first discuss models which assume that user preferences (ratings) have been pre-extracted from the conversation and are available in a ratings-vector format ($\mathbf{R}_u^{1:t}$). We then introduce several variations of transformer models based on BERT that directly model the dialogue text. We also consider methods for modelling movie features to address the cold-start problem.

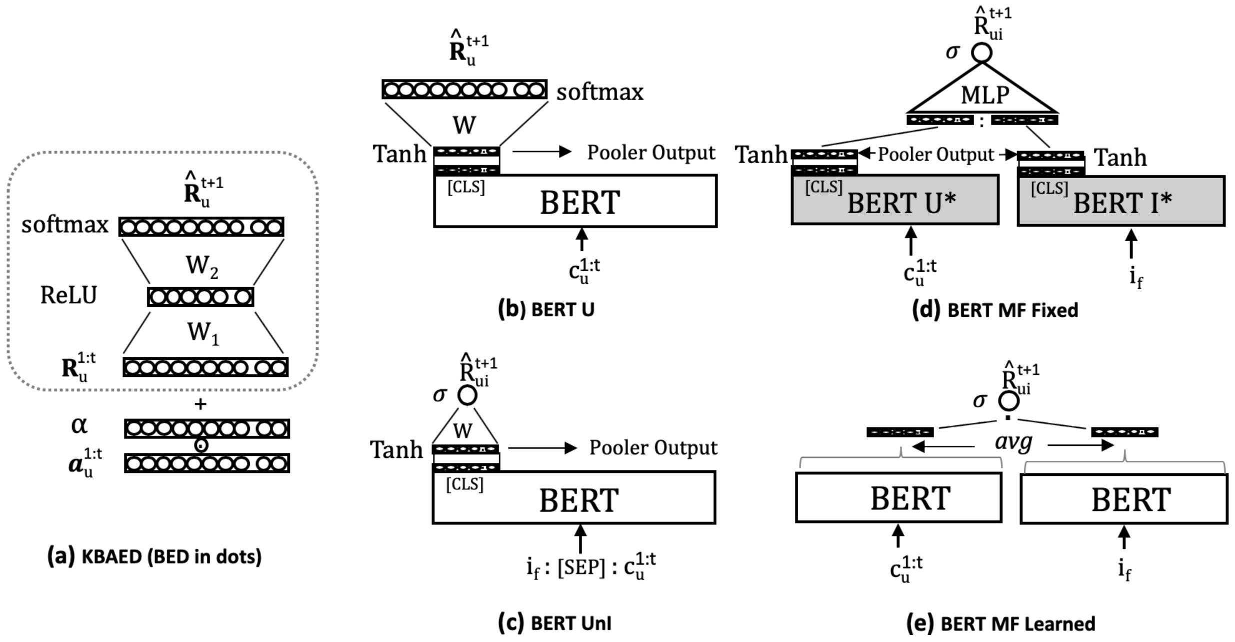

Fig. 3.1. We study different classes of models for item recommendation from conversation ($c_u^{1:t}$). (a) Encoder-decoders that rely on ratings $\mathbf{R}_u^{1:t}$ and attributes $\mathbf{a}_u^{1:t}$ having been extracted from the conversation. (b–e) Transformer-based models that directly model the (text) conversation. Models (b) and (c) are instances of BERT trained for recommendations. Models (d) and (e) are inspired by matrix factorization approaches and use two BERTs to learn a user and an item embedding. For (d) we reuse the BERTs trained using (b) (and denote them BERT U* and I*).

Visuals of all models contributed in this paper.

3.4.1. Baselines

Our *Basic Encoder Decoder BED (Figure 3.1a top)* is an instance of an auto-encoder for recommendations [**38**]. It uses the vector of observed ratings $\mathbf{R}_u^{1:t}$ as input. Ratings must then be extracted from the dialogue utterances upstream. BED serves as a baseline method with which to compare models of conversation (§3.4.2). Ratings $\mathbf{R}_u^{t+1}$ are computed as $\text{softmax}(\mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1\mathbf{R}_u^{1:t} + \mathbf{b}_1) + \mathbf{b}_2)$ with parameters $\theta := \mathbf{W}_{\{1,2\}}, \mathbf{b}_{\{1,2\}}$. The parameters are optimized by minimizing a masked binary cross-entropy (MBCE) on the training data. It is masked since only the output dimensions that are observed are used in the loss. (It is not standard to combine a softmax with a BCE loss, but we found that it performs best in our study and is also supported by other studies [**27**].)

We also examine a *Knowledge-base augmented BED model (KBAED) (Figure 3.1a).* In conversational recommender systems, users can express preferences toward attributes of movies. For example, the dataset we study in this paper contains many conversations in which movie genres (e.g., comedy, drama) are used to steer recommendations. KBAED extends BED to model such attribute preferences. The assumption we make is that a user expressing a preference for one or more attributes is expressing a positive preference toward movies that contains these attributes.

The input of KBAED consists of a vector of available attributes $\mathbf{A}_u^{1:t}$—which encodes the preferences of user u for item attributes—summed with the ratings vector $\mathbf{R}_u^{1:t}$. For genres, we found that limiting the attribute preferences to the top movies (we use 100) and weighting the level of the preference using movie popularity was beneficial: $\mathbf{a}_u^{1:t} = \| \text{top}_n(\mathbf{A}_u^{1:t} * \mathbf{p}) \|$, where top_n returns its arguments for the top n values and zero otherwise. We estimate a movie's popularity using its number of (training) ratings $\mathbf{p} = \sum_{u \in \mathcal{U}} [\mathbf{R}_{u,1}, \ldots, \mathbf{R}_{u,|\mathcal{V}|}]$. Hence, $\mathbf{R}_u^{t+1} = \text{softmax}(\mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1(\mathbf{R}_u^{1:t} + \boldsymbol{\alpha}\mathbf{a}_u^{1:t}) + \mathbf{b}_1) + \mathbf{b}_2)$ with parameters $\theta := \{\mathbf{W}_{\{1,2\}}, \mathbf{b}_{\{1,2\}}, \boldsymbol{\alpha}\}$. KBAED trains with the MBCE loss.

3.4.2. Language Models for Recommendations from Dialogue

Here we introduce a set of recommendation models based on BERT. Compared to the above baselines these models have the ability to make recommendations directly from the text of the conversations $c_u^{1:t}$. We explore different methods for transforming representations inferred through BERT into recommendations. We present two methods inspired by matrix-factorization (MF) modelling [**36**] that learn a separate user and item representation which is then combined into a rating (prediction). We also explore methods for modelling the item attributes i_f in the conversation (in addition to the conversation).

BERT U (Figure 3.1b) is a vanilla BERT which models the text in the conversation. Its pooler output (*pool*) is used as input to an affine transformation followed by a softmax over all movies $\mathbf{R}_u^{t+1} = \text{softmax}(\mathbf{W}(\textit{pool}(\text{BERT}(c_u^{1:t})) + \mathbf{b})$ with parameters $\theta := \mathbf{W}, \mathbf{b}$ and BERT parameters. Training is done with MBCE. In a similar fashion, we can also instanciate BERT I which takes as input i_f and predicts ratings for all users.

One limitation of BERT U is that it cannot recommend new (cold-start) items. As a remedy we can use BERT to also model item attributes. The input of BERT UnI (Figure 3.1c) consists of the conversation $c_u^{1:t}$ concatenated with the item features i_f separated by a special token ([SEP]). An affine transformation is added on top of BERT's pooler output to predict the particular user-item rating $R_{ui}^{t+1} = \sigma(\mathbf{w} \cdot pool(\text{BERT}(c_u^{1:t} : [SEP] : i_f)) + b)$ with parameters $\theta := \mathbf{W}, \mathbf{b}$ and BERT parameters all learned with BCE loss.

BERT MF Fixed (Figure 3.1d) is inspired by the neural collaborative filtering model [**16**]. We concatenate the BERT representations of a user and an item and input them to a neural network that predicts their rating. Each representation is the pooler output of optimized BERT U and BERT I models. The predicted rating R_{ui}^{t+1} is given by $\sigma(\text{MLP}([pool(\text{BERT U}^*), pool(\text{BERT I}^*)]))$ where learning optimizes the MLP parameters only, using BCE loss.

BERT MF Learned (Figure 3.1e) is similar to BERT MF Fixed, but 1) it combines the user and item representations through a dot product, and 2) both BERTs are trained using the BCE loss. Preliminary results showed benefits in using the average of the last hidden layer (*avg* instead of *pool*) to obtain representations: $R_{ui}^{t+1} = \sigma(\text{avg}(\text{BERT}(c_u^{1:t})) \cdot avg(\text{BERT'}(i_f)))$.

3.5. Study

Dataset. We use the ReDial dataset which contains 11,348 conversations between a user seeking a recommendation and a user acting as a recommendation provider [**25**].

Binary ratings have been extracted at the utterance level. We split the conversations 80/10/10 in train/valid/test. We further segment conversations into multiple instances, one for each user utterance that was followed by a recommendation from the recommender. There are 6,924 movies (items) discussed in ReDial. Our test set has 299 target movies that were never seen in the training set. These will be considered as new movies for studying item cold-start and we use a movie knowledge base from IMDB[1] to create movie item embeddings. Experimental Details: When considering the attributes $\mathbf{A}_u^{1:t}$ mentioned by a user we limit ourselves to the genre attribute, as it is the most common one in ReDial's conversations. We matched them using a list of 24 popular genres from IMDB. [2] We studied three different subsets of item information: i_f corresponds to a text representation of the full knowledge base we collected about item i, i_m is a shorter version with only title, genres, and actors, and i_s, is the same as i_m but with only up to three (leading) actors.

[1]we accessed the data using the IMDb python package https://pypi.org/project/IMDb/
[2]"Popular TV Series by Genre" from `https://www.imdb.com/feature/genre/?ref_=nv_ch_gr`

In Table 3.1 we report the exact attributes that were selected (based on cross-validation using ndcg@100 for early stopping) in each model.

Model	Model Description
BED	$\text{softmax}(\mathbf{W}_2 f(\mathbf{W}_1 \mathbf{R}_u^{1:t} + \mathbf{b}_1) + \mathbf{b}_2)$
KBAED	$\text{softmax}(\mathbf{W}_2 f(\mathbf{W}_1 (\mathbf{R}_u^{1:t} + \boldsymbol{\alpha}\mathbf{a}_u^{1:t}) + \mathbf{b}_1) + \mathbf{b}_2)$
BERT U	$\text{softmax}(\mathbf{W}(pool(\text{BERT}(c_u^{1:t})) + \mathbf{b})$
BERT MF Fixed	$\sigma(\text{MLP}(\text{BERT U}^* : \text{BERT I}^*))$
BERT MF Learned	$\sigma(avg(\text{BERT}(c_u^{1:t}))) \cdot avg(\text{BERT'}(i_f)))$
BERT UnI ($R^{-}20$)	$\sigma(\mathbf{w}(pool(\text{BERT}(i_s : [\text{SEP}] : c_u^{1:t}))) + b$
BERT UnI ($R^{-}40$)	$\sigma(\mathbf{w}(pool(\text{BERT}(i_s : [\text{SEP}] : c_u^{1:t}))) + b$

Table 3.1. Models description. $\mathbf{W}, \mathbf{b}, \boldsymbol{\alpha}$ are trainable parameters as those in BERT models, f is the ReLU activation function, σ is sigmoid, $\cdot$ is dot product, : concatenation and * stands for optimized models.

We need to manage the rating imbalance in ReDial: 94% of ratings are 1s (liked) and only 6% are 0s.

For models predicting a vector of ratings (BED, KBAED, and BERT U), it helps to use a softmax output forcing the model to keep a fixed probability mass over all items [**27**]. For the others, we used the common practice of imputing zero ratings to a random subset of unobserved items [**15, 54**]. We denote the number of random ratings as R^{-}.

Metrics. We use two standard metrics in our experiments: *Recall@k* and *NDCG@k*. Both evaluate the quality of the top-k ranked items by a model. Recall considers that items ranked below rank k have uniform importance, while NDCG exponentially reduces the importance of lower-ranked items.

3.5.1. Results

We first report overall results and then focus on different facets of the tasks and the models. Table 3.2 reports the average test recommendation performance of all models. In addition to the model we propose, we also compare it with KBRD [**6**]. KBRD extracts entities from the dialogue and learns their embeddings using an external knowledge graph. Predictions are obtained through attention on these learned entities. The results of the ReDial model (the model accompanying the ReDial paper[**25**]) and KBRD were taken directly from [**6**]. To our knowledge, our setup is the same as theirs and the results comparable (up to a random seed).

The best recommendations are obtained by BERT UnI (only outperformed by BERT U in Recall@1). The fact that it infers a joint user-item distribution (as opposed to MF models that infer them independently) may explain this result. Note that we used i_s for BERT UnI for computational reasons since its input sequences are longer (concatenation of user and item text).

Still, BERT UnI comes with a larger computational cost at prediction time since it needs to do a BERT forward pass for all combinations of users and items (Table 3.4). BERT MF provides good performance at a reasonable cost since inferred user and item embeddings can be reused across predictions.

Model	Recall@1	Recall@10	Recall@50	NDCG@100
ReDial model[**25**]	2.3	12.9	28.7	-
KBRD[**6**]	3.0	16.3	33.8	-
BED	2.96	13.26	30.02	0.1262
KBAED	3.05	15.33	34.16	0.1402
BERT U	**3.13** (0)	15.36 (0)	34.25 (0)	0.1365 (0)
BERT MF Fixed	2.93 (0)	13.33 (0)	32.54 (1.06)	0.1300 (0.0078)
BERT MF Learned	3.08 (0)	15.16 (0)	36.10 (0.71)	0.1432 (0.0048)
BERT UnI ($R^{-}20$)	2.03 (0.24)	15.28 (4.33)	**40.59** (9.38)	0.1471 (0.0320)
BERT UnI ($R^{-}40$)	3.02 (0.24)	**16.54** (4.33)	38.67 (6.73)	**0.1511** (0.0231)

Table 3.2. Models recommendation test results on ReDial. Scores in parentheses correspond to recommendation score on new items (we kept as targets only movies never seen in training). Results for ReDial Model and KBRD copied from [**6**].

3.5.1.1. Sequential and User Cold Start Evaluation. In conversational recommendations, the system gathers information throughout the dialogue. Figure 3.2b presents a temporal view of the experiment reported in Table 3.2. We group conversations in terms of how many movie preferences were available before making the recommendation. We note that overall the relative performance of models are similar in Table 3.2 with BERT UnI outperforming all approaches.

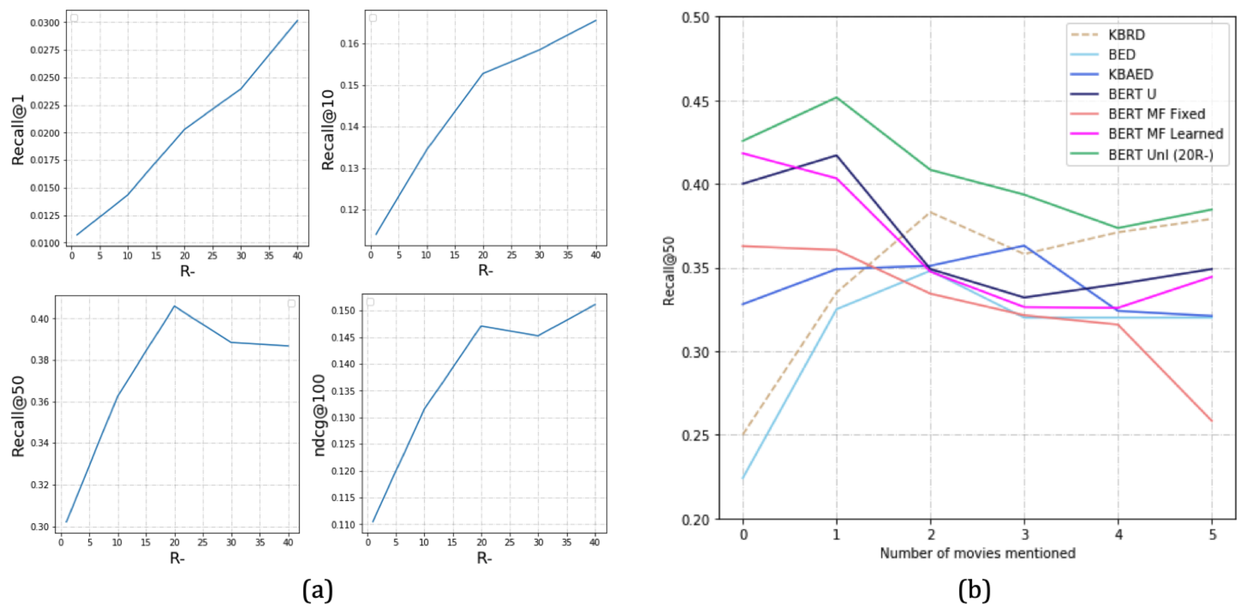

Fig. 3.2. (a) Impact in variation of R^{-} on four metrics. (b) Recall@50 for all models, KBRD[**6**] is the previous state of the art.

The trend that performance degrades over time is surprising. However, in ReDial, movies mentioned at beginning of conversations are, on average, very popular. But, as the conversation progresses recommendations are better informed and become more specific and hence harder to predict in absolute terms.

At the start of the conversation, we can evaluate the performance of models in the user cold-start setting. Without surprise, BED's performance is the lowest since it only uses previous ratings. The results also show that KBRD is unable to use the little information available.

The performance of KBAED is also low early in the conversation, but since it can use a genre input when available, it outperforms BED. Further, models using the complete text available through different implementations of BERT show even better performances as they can use all available information (e.g. all available attributes of movies and their associated sentiments) even including the nuances expressible in language.

We notice that two-movie mentions provide enough data for the encoder-encoder models (BED and KBAED) to match the performance of most text-based ones except for the performance of BERT UnI which is only matched by the end of the conversation by KBRD.

3.5.1.2. Item Cold Start. To analyze the performance of the models on cold-start items, we consider the ability of models to predict movies that were never seen in the train set, as discussed above. These results are presented next to the full results in Table 3.2 (in parentheses). We see that BERT U cannot rank unseen movies well, which is to be expected since it does not learn an explicit item representation.

Results suggest that BERT UnI would provide the best prediction for unseen items given its joint representation of users and items. Further, the performance of BERT UnI on new items contributes to its overall superior performance (Table 3.2).

3.5.1.3. Hyper-Parameter Exploration: Pooler Output VS Last Hidden Layer Average. Table 3.3 reports a comparison between two methods for obtaining representations from BERT: the pooler and averaging its last hidden layer. We compare these two strategies using BERT MF Learned models with 40 R^- and i_m. Results suggest that averaging the last hidden layer provides a better representation, as suggested in the documentation of the Hugging Face group [**51**]. Note that the difference between BERT MF Learned results in Tables 3.2 and 3.3 is caused by using i_f in the former and i_m in the latter, emphasizing the importance of using all available item features.

Model	Output	R@1	R@10	R@50	NDCG@100
BERT MF Learned	Pooler	0.88	6.66	15.52	0.0679
BERT MF Learned	Average	2.10	12.02	28.79	0.1176

Table 3.3. BERT MF Learned output - Pooler VS Learned.

Model	Train (hours)	Prediction
BERT U	19.2	$\lvert\mathcal{U}\rvert B_r$
BERT MF Fixed	38.7	$\lvert\mathcal{U}\rvert B_r + \lvert\mathcal{V}\rvert B_r$
BERT MF Learned	172	$\lvert\mathcal{U}\rvert B_r + \lvert\mathcal{V}\rvert B_r$
BERT UnI	44	$\lvert\mathcal{U}\rvert\lvert\mathcal{V}\rvert B_r$

Table 3.4. Training time (20 R^-) and time complexity of BERT models to obtain predictions for $|U|$ users and $|V|$ items, where B_r is the complexity of the forward-pass of BERT.

3.5.1.4. Hyper-Parameter Exploration: Impact of Random Ratings (R^-). We now explore the impact of the number of items that we randomly set to 0 for training. Figure 3.2a shows the performance of BERT UnI as a function of R^- (x-axis).

We see a monotonic benefit in augmenting R^- for Recall@1 and Recall@10. However, the performance of Recall@50 actually diminishes when increasing R^- above 20. NDCG@100 which gives more importance to the top ranks shows mixed results. Overall, it seems like having more counter-examples (higher R^-) is beneficial to ensure the quality of short recommendation lists. This intuition is confirmed by the best results for Recall@1 obtained by BERT U, a model that uses softmax, which is similar to setting all unmentioned items at 0 (i.e. huge R^-).

3.5.1.5. Computational resources for training and serving recommendations. Even BERT fine-tuning uses significant computing resources. In Table 3.4 we report training time and resource consumption of the BERT-based models. For BERT MF and BERT UnI, the time depends on the number of Random Ratings (R^-). We present results for 20 R^-. BERT MF Learned requires learning two separate BERT models at the same time, one for users, one for items, which uses more memory. It forces us to reduce batch size and increases wall-clock training time.

We report the time complexity of making predictions for $|\mathcal{U}|$ user and $|\mathcal{V}|$ items in the last column of Table 3.4. Recall that BERT is composed of thousands of feed-forward networks and millions of dot products and so the extra time to computer the MLP and a small number of extra dot products is negligible. We denote the time complexity of a forward pass through BERT as B_r. For BERT U, a single forward-pass gives an embedding that can be decoded to obtain predicted ratings for all items, hence it costs $|\mathcal{U}|B_r$ to compute. For both BERT MF models, all users and all items have to go through their respective BERT once to get their representations and their combination is obtained through a dot product or MLP (both negligible time), hence $|\mathcal{U}|B_r + |\mathcal{V}|B_r$. For BERT UnI, since each combination of user and item has to go through BERT, we have $|\mathcal{U}||\mathcal{V}|B_r$.

3.6. Conclusion and Future Work

We studied the task of recommendations from dialogue, a core component of conversational recommender systems. We show that new advances in language modelling provide strong performance in several settings including user and item cold-start at different training and serving costs.

Chapitre 4

Conclusion and Future Work

4.1. Conclusion

Our study shows that recent improvements in language modelling can benefit conversational recommender systems by improving recommendations made to users, especially in cold-start settings. In particular, we show that pre-trained transformer models outperform baselines even if the baselines have access to the user preferences manually extracted from their utterances.

According to our results, there isn't one go-to model to use in all cases. Here are some of the things one might want to consider when deciding which model is better suited for its situation:

- New users. If new users are important in your situation, all BERT models are showing stronger performances. Since very low information about the user is available at these stages, it's better to use everything you have. Language may contain a lot of information that you don't want to turn your eyes of
- New items. In a situation where no new items are added or are important, BERT U offers great results, especially for the first few ranks, plus it is super efficient. On the other end, if new items are important to you, BERT MF methods or BERT UnV should be preferred. For short dialogues, depending on the computing power available, go from BERT MF Fixed, BERT MF Learned to BERT UnV. As dialogues start to grow, all BERT models become problematic, BERT UnV being the first one affected. See Future Work (section 4.2) for potential solutions
- Number of recommendations. When focusing solely on the first few items recommended (e.g. less than 10), we haven't found a limit to the amount of negative sampling, so don't be afraid to try above 40. On the other end, when considering wider ranges of recommendations (e.g. 50), negative sampling seems better when limited to around 20.

4.2. Future Work

Some aspects of our research could benefit from more exploration including which latent representation to use (pooler output VS average of last hidden layer) and the number of negative samples (R^-). These might lead to further performance improvements. Another obvious next step would be to explore our proposed approaches on other datasets to see how our actual observations hold and possibly gain information on further directions to investigate.

The length of the dialogues has an important impact on the models presented in this research. Because we limited ourselves to the ReDial dataset and that the dialogues it contains are relatively small, this topic is not covered by our research. But, finding user representation in real life would rapidly outgrow the capacity of BERT (only 512 tokens). Therefore, future work should include such considerations. A typical approach is to divide the text into subsets of approximately 512 tokens. At training time, all subsets have the same target. For prediction, an average of ratings over all the subsets could be used. Depending on the situation, aggregating subset's results through a maximum can provide better performances. This approach might lead to a very confused BERT since a sample (a subset of the full conversation) might not contain information about the recommendation to make. Complexity grows even more when the same user returns for a completely different type of need. What training target do you use then? Another approach could be to use a latent representation of the user and combine it with the representation of the last conversation to get an actualized user representation. Then, new questions arise as to how to obtain these latent representations, if they are fixed (easier to train) or part of an end-to-end model, and how to combine them.

Finally, all our work in this paper focuses on the receptive part of a conversation: "Listen before you talk". Future work necessarily includes the end-to-end dialogue system. A lot of challenges remain there. First, we could choose the intent, for example, whether or not the text output will include a recommendation, a clarification question or a combination of both. We could also compare retrieval versus generative dialogue models. Finally, the evaluation of this type of work is also trickier, especially when the time comes to evaluate language generation [**5, 40, 34**].

Références bibliographiques

[1] K M Annervaz, Somnath Basu Roy Chowdhury et Ambedkar Dukkipati : Learning beyond datasets: Knowledge graph augmented neural networks for natural language processing, 2018.

[2] Dzmitry Bahdanau, Kyunghyun Cho et Yoshua Bengio : Neural machine translation by jointly learning to align and translate, 2014.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent et Christian Janvin : A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[4] Leo Breiman : Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[5] Ozan Caglayan, Pranava Madhyastha et Lucia Specia : Curious case of language generation evaluation metrics: A cautionary tale. *arXiv preprint arXiv:2010.13588*, 2020.

[6] Qibin Chen, Junyang Lin, Yichang Zhang, Ming Ding, Yukuo Cen, Hongxia Yang et Jie Tang : Towards knowledge-based recommender dialog system, 2019.

[7] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk et Yoshua Bengio : Learning phrase representations using RNN encoder–decoder for statistical machine translation. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, octobre 2014. Association for Computational Linguistics.

[8] Konstantina Christakopoulou, Filip Radlinski et Katja Hofmann : Towards conversational recommender systems. *In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 815–824, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Corinna Cortes et Vladimir Vapnik : Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[10] Paul Covington, Jay Adams et Emre Sargin : Deep neural networks for youtube recommendations. *In Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[11] Christian Desrosiers et George Karypis : *A Comprehensive Survey of Neighborhood-based Recommendation Methods*, pages 107–144. Springer US, Boston, MA, 2011.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee et Kristina Toutanova : Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[13] Jesse Dodge, Andreea Gane, Xiang Zhang, Antoine Bordes, Sumit Chopra, Alexander Miller, Arthur Szlam et Jason Weston : Evaluating prerequisite qualities for learning end-to-end dialog systems, 2015.

[14] Tim Donkers, Benedikt Loepp et Jürgen Ziegler : Sequential user-based recurrent neural network recommendations. *In Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 152–160, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu et Tat-Seng Chua : Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu et Tat-Seng Chua : Neural collaborative filtering. *In Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[17] Sepp Hochreiter et Jürgen Schmidhuber : Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, novembre 1997.

[18] Shaojie Jiang, Pengjie Ren, Christof Monz et Maarten de Rijke : Improving neural response diversity with frequency-aware cross-entropy loss. *In The World Wide Web Conference*, WWW '19, page 2879–2885, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Dongyeop Kang, Anusha Balakrishnan, Pararth Shah, Paul Crook, Y-Lan Boureau et Jason Weston : Recommendation as a communication game: Self-supervised bot-play for goal-oriented dialogue, 2019.

[20] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee et Hwanjo Yu : Convolutional matrix factorization for document context-aware recommendation. *In Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 233–240, New York, NY, USA, 2016. Association for Computing Machinery.

[21] Yehuda Koren et Robert Bell : *Advances in Collaborative Filtering*, pages 145–186. Springer US, Boston, MA, 2011.

[22] Rémi Lebret et Ronan Collobert : Word emdeddings through hellinger pca, 2017.

[23] Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He et Dawei Yin : Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. *In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447, Melbourne, Australia, juillet 2018. Association for Computational Linguistics.

[24] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian et Jun Ma : Neural attentive session-based recommendation. *In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1419–1428, New York, NY, USA, 2017. Association for Computing Machinery.

[25] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin et Chris Pal : Towards deep conversational recommendations. *In Advances in Neural Information Processing Systems 31 (NIPS 2018)*, 2018.

[26] Ziming Li, Julia Kiseleva et Maarten de Rijke : Dialogue generation: From imitation learning to inverse reinforcement learning, 2018.

[27] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman et Tony Jebara : Variational autoencoders for collaborative filtering, 2018.

[28] Angli Liu, Jingfei Du et Veselin Stoyanov : Knowledge-augmented language model and its application to unsupervised named-entity recognition, 2019.

[29] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao et Jiawei Han : On the variance of the adaptive learning rate and beyond, 2020.

[30] Benjamin M. Marlin, Richard S. Zemel, Sam T. Roweis et Malcolm Slaney : Collaborative filtering and the missing at random assumption. *CoRR*, abs/1206.5267, 2012.

[31] Tomas Mikolov, Kai Chen, Greg Corrado et Jeffrey Dean : Efficient estimation of word representations in vector space, 2013.

[32] Julian McAuley Noveen Sachdeva : How useful are reviews for recommendation? a critical review and potential improvements. *In SIGIR*, 2020.

[33] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi et Paolo Cremonesi : Personalizing session-based recommendations with hierarchical recurrent neural networks. *In Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 130–137, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Ehud Reiter et Anja Belz : An investigation into the validity of some metrics for automatically evaluating natural language generation systems. *Computational Linguistics*, 35(4):529–558, 2009.

[35] Lamyaa Sadouk : Cnn approaches for time series classification. 2018.

[36] Ruslan Salakhutdinov et Andriy Mnih : Probabilistic matrix factorization. *In Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 1257–1264, Red Hook, NY, USA, 2007. Curran Associates Inc.

[37] Ruslan Salakhutdinov, Andriy Mnih et Geoffrey Hinton : Restricted boltzmann machines for collaborative filtering. *In Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 791–798, New York, NY, USA, 2007. Association for Computing Machinery.

[38] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner et Lexing Xie : Autorec: Autoencoders meet collaborative filtering. *In Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 111–112, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus et Yann LeCun : Overfeat: Integrated recognition, localization and detection using convolutional networks, 2014.

[40] Shikhar Sharma, Layla El Asri, Hannes Schulz et Jeremie Zumer : Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *arXiv preprint arXiv:1706.09799*, 2017.

[41] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou et Peng Jiang : Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer, 2019.

[42] Yueming Sun et Yi Zhang : Conversational recommender system, 2018.

[43] Rianne van den Berg, Thomas N. Kipf et Max Welling : Graph convolutional matrix completion, 2017.

[44] Aaron van den Oord, Sander Dieleman et Benjamin Schrauwen : Deep content-based music recommendation. *In* C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani et K. Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013.

[45] V Vapnik et A Ya Chervonenkis : Algorithms with complete memory and recurrent algorithms in the problem of learning pattern recognition. *Avtomat. i Telemekh*, 4:95–106, 1968.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser et Illia Polosukhin : Attention is all you need, 2017.

[47] Jingjing Wang, Haoran Xie, Oliver Tat Sheung Au, Di Zou et Fu Lee Wang : Attention-based cnn for personalized course recommendations for mooc learners. *In 2020 International Symposium on Educational Technology (ISET)*, pages 180–184, 2020.

[48] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath et Huan Liu : What your images reveal: Exploiting visual contents for point-of-interest recommendation. *In Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 391–400, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[49] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu et Tat-Seng Chua : Kgat: Knowledge graph attention network for recommendation. *In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 950–958, New York, NY, USA, 2019. Association for Computing Machinery.

[50] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao et Tat-Seng Chua : Explainable reasoning over knowledge graphs for recommendation, 2018.

[51] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz et Jamie Brew : Huggingface's transformers: State-of-the-art natural language processing, 2019.

[52] Yao Wu, Christopher DuBois, Alice X. Zheng et Martin Ester : Collaborative denoising auto-encoders for top-n recommender systems. *In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, page 153–162, New York, NY, USA, 2016. Association for Computing Machinery.

[53] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes et Jeffrey Dean : Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[54] Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang et Jiajun Chen : Deep matrix factorization models for recommender systems. *In Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 3203–3209. AAAI Press, 2017.

[55] Robert Bell Yehuda Koren et Chris Volinsky : Matrix factorization techniques for recommender systems, 2009.

[56] Xiaoying Zhang, Hong Xie, Hang Li et John C. S. Lui : Conversational contextual bandit: Algorithm and application, 2020.

[57] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang et W. Bruce Croft : Towards conversational search and recommendation: System ask, user respond. *In Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 177–186, New York, NY, USA, 2018. Association for Computing Machinery.