Foundations and Trends® in Information Retrieval

Deep Learning for Matching in Search and Recommendation

Suggested Citation: Jun Xu, Xiangnan He and Hang Li (2020), "Deep Learning for Matching in Search and Recommendation", Foundations and Trends® in Information Retrieval: Vol. 14, No. 2–3, pp 102–288. DOI: 10.1561/1500000076.

Jun Xu
Gaoling School of Artificial Intelligence
Renmin University of China
China
junxu@ruc.edu.cn

Xiangnan He
School of Information Science and Technology
University of Science and Technology of China
China
hexn@ustc.edu.cn

Hang Li
Bytedance AI Lab
China
lihang.lh@bytedance.com

now
the essence of knowledge
Boston — Delft

Contents

Deep Learning for Matching in Search and Recommendation

Jun Xu[1], Xiangnan He[2] and Hang Li[3]

[1] *Gaoling School of Artificial Intelligence, Renmin University of China, China; junxu@ruc.edu.cn*
[2] *School of Information Science and Technology, University of Science and Technology of China; hexn@ustc.edu.cn*
[3] *Bytedance AI Lab, China; lihang.lh@bytedance.com*

ABSTRACT

Matching is a key problem in both search and recommendation, which is to measure the relevance of a document to a query or the interest of a user to an item. Machine learning has been exploited to address the problem, which learns a matching function based on input representations and from labeled data, also referred to as "learning to match". In recent years, efforts have been made to develop deep learning techniques for matching tasks in search and recommendation. With the availability of a large amount of data, powerful computational resources, and advanced deep learning techniques, deep learning for matching now becomes the state-of-the-art technology for search and recommendation. The key to the success of the deep learning approach is its strong ability in learning of representations and generalization of matching patterns from data (e.g., queries, documents, users, items, and contexts, particularly in their raw forms).

Jun Xu, Xiangnan He and Hang Li (2020), "Deep Learning for Matching in Search and Recommendation", Foundations and Trends® in Information Retrieval: Vol. 14, No. 2–3, pp 102–288. DOI: 10.1561/1500000076.

This survey gives a systematic and comprehensive introduction to the deep matching models for search and recommendation developed recently. It first gives a unified view of matching in search and recommendation. In this way, the solutions from the two fields can be compared under one framework. Then, the survey categorizes the current deep learning solutions into two types: methods of representation learning and methods of matching function learning. The fundamental problems, as well as the state-of-the-art solutions of query-document matching in search and user-item matching in recommendation, are described. The survey aims to help researchers from both search and recommendation communities to get in-depth understanding and insight into the spaces, stimulate more ideas and discussions, and promote developments of new technologies.

Matching is not limited to search and recommendation. Similar problems can be found in paraphrasing, question answering, image annotation, and many other applications. In general, the technologies introduced in the survey can be generalized into a more general task of matching between objects from two spaces.

List of Acronyms

PLS Partial Least Square

RMLS Regularized Matching in Latent Space

SSI Supervised Semantic Indexing

BMF Biased Matrix Factorization

FISM Factored Item Similarity Model

FM Factorization Machine

FFN Feedforward Neural Network

MLP Multilayer Perceptron

CNN Convolutional Neural Networks

RNN Recurrent Neural Networks

GAN Generative Adversarial Network

AE Autoencoders

DAE Denoising Autoencoder

CBOW Continuous Bag of Words

SG Skip Gram

BERT Bidirectional Encoder Representations from Transformers

DSSM Deep Structured Semantic Models

CLSM Convolutional Latent Semantic Model

CNTN Convolutional Neural Tensor Network

LSTM-RNN Recurrent Neural Networks with Long Short-Term Memory cells

NVSM Neural Vector Space Model

SNRM Standalone Neural Ranking Model

ACMR Adversarial Cross Modal Retrieval

ARC-II Convolutional Matching Model II

DRMM Deep Relevance Matching Model

K-NRM Kernel Based Neural Ranking Model

DeepMF Deep Matrix Factorization

CDAE Collaborative Denoising Auto-Encoder

NAIS Neural Attentive Item Similarity

NARM Neural Attentive Recommendation Machine

DeepCoNN Deep Cooperative Neural Networks

NARRE Neural Attention Regression with Review-level Explanation

VBPR Visual Bayesian Personalized Ranking

CDL Comparative Deep Learning

ACF Attentive Collaborative Filtering

NGCF Neural Graph Collaborative Filtering

KGAT Knowledge Graph Attention Network

KPRN Knowledge Path Recurrent Network

NCF Neural Collaborative Filtering

ConvNCF Convolutional Neural Collaborative Filtering

GMF Generalized Matrix Factorization

NeuMF Neural Matrix Factorization

CML Collaborative Metric Learning

TransRec Translation-based Recommendation

LRML Latent Relational Metric Learning

NFM Neural Factorization Machine

AFM Attentional Factorization Machine

1

Introduction

1.1 Search and Recommendation

With the rapid growth of the internet, one of the fundamental problems in information science becomes even more critical today, that is, how to identify the information satisfying a user's need from a usually huge pool of information. The goal is to present the user only the information that is of interest and relevance, at the right time, place, and context. Nowadays, two types of information accessing paradigms, search and recommendation, are widely used in a great variety of scenarios.

In search, documents (e.g., web documents, Twitter posts, or E-commerce products) are first pre-processed and indexed in the search engine. After that, the search engine takes a query (a number of keywords) from the user. The query describes the user's information need. Relevant documents are retrieved from the index, matched with the query, and ranked according to their relevance to the query. For example, if a user is interested in news about quantum computing, the query "quantum computing" mat be submitted to a search engine and get news articles about the topic will be returned.

Different from search, a recommendation system typically does not take a query. Instead, it analyzes the user's profile (e.g., demographics and contexts) and historical interactions on items, and then makes recommendation on items to the user. The user features and item features are indexed and stored in the system in advance. The items are ranked according to the likelihood that the user is interested in them. For example, on a news website, when a user browses and clicks a new article, several news articles with similar topics or news articles that other users have clicked together with the current one may be shown.

Table 1.1 summarizes the differences between search and recommendation. The fundamental mechanism of search is "pull", because users first make specific requests (i.e., submit queries) and then receive information. The fundamental mechanisms of recommendation is "push", because users are provided information which they do not specifically request (e.g., submit queries). Here "beneficiary" means the people whose interests are to be met in the task. In a search engine, the results are typically created solely based on the user's needs, and thus the beneficiary is the users. In a recommendation engine, the results usually need to satisfy both the users and providers, and thus the beneficiary is all of them. However, the distinction is becoming blurred recently. For example, some search engines mix search results with paid advertisements, which benefits both the users and the providers. As for "serendipity", it means that conventional search focuses more on information that is clearly relevant. Conventional recommendation, on the other hand, is allowed to offer unexpected but useful information.

Table 1.1: Information-providing mechanisms of search and recommendation

	Search	**Recommendation**
Query available	Yes	No
Delivery model	Pull	Push
Beneficiary	User	User and provider
Serendipity	No	Yes

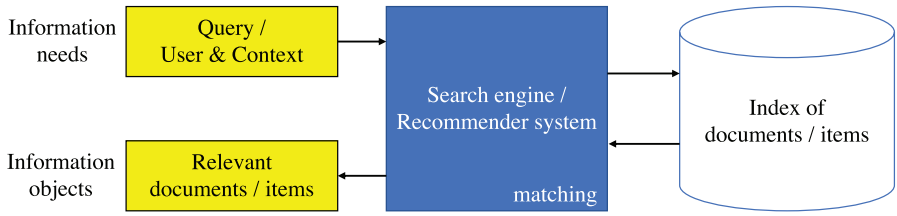

Figure 1.1: Unified view of matching in search and recommendation.

1.2 Unifying Search and Recommendation from Matching Viewpoint

Garcia-Molina *et al.* (2011) pointed out that the fundamental problems in search and recommendation are to identify information objects satisfying users' information needs. It is also indicated that search (information retrieval) and recommendation (information filtering) are the two sides of the same coin, having strong connections and similarities (Belkin and Croft, 1992). Figure 1.1 illustrates the unified matching view of search and recommendation. The goal in common is to present to the users the information they need.

Search is a retrieval task, which aims to retrieve the documents that are relevant to the query. In contrast, recommendation is a filtering task, which aims to filter out the items that are of interest to the user (Adomavicius and Tuzhilin, 2005). As such, search can be considered as conducting matching between queries and documents, and recommendation can be considered as conducting matching between users and items. More formally, both the matching in search and recommendation can be considered as constructing a matching model $f\colon \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{R}$ which calculates the matching degree between two input objects x and y, where $\mathcal{X}$ and $\mathcal{Y}$ denote two object spaces. $\mathcal{X}$ and $\mathcal{Y}$ are the spaces of queries and documents in search, or the spaces of users and items in recommendation.

Under the unified matching view in Figure 1.1, we use the term information objects to denote the documents/items to retrieve/recommend, and use information needs to denote the queries/users in the respective task. By unifying the two tasks under the same view of matching and

comparably reviewing existing techniques, we can provide deeper insights and more powerful solutions to the problems. Moreover, unifying the two tasks also has practical and theoretical implications.

Search and recommendation have already been combined in some practical applications. For example, at some E-commerce sites, when the user submits a query, a ranking list of products are presented based on not only relevance (query-product matching) but also user interest (user-product matching). In some lifestyle apps, when the user searches for restaurants, the results are returned based on both relevance (query-restaurant matching) and user interest (user-restaurant matching). There is a clear trend that search and recommendation will be integrated into a single system at certain scenarios to meet users' needs better, where matching plays an essential role.

Search and recommendation already have many shared technologies because of their similarities in matching. Some search problems can be solved by using recommendation techniques (Zamani *et al.*, 2016), and vice versa (Costa and Roda, 2011), on the basis of matching. With the use of deep learning technologies, the matching models for search and recommendation bear even more resemblance in architecture and methodology, as reflected in the techniques: embedding the inputs (queries, users, documents, and items) as distributed representations, combining neural network components to represent the matching function, and training the model parameters in an end-to-end manner. Moreover, search and recommendation can be jointly modeled and optimized if they share the same set of information objects (as in the above examples of E-commerce sites and lifestyle apps) (Schedl *et al.*, 2018; Zamani and Croft, 2018a, 2020). Therefore, in order to develop more advanced ones, it is necessary and advantageous to take a unified matching view to analyze and compare existing search and recommendation technologies.

The matching tasks in search and in recommendation face different challenges in practice. The underlying problem is essentially the same, however, that is, the mismatch challenge. Next, we introduce the key challenges of the two tasks, respectively.

1.3 Mismatching Challenge in Search

In search, queries and documents (usually their titles) are taken as texts. The relevance of a document to a query is mainly represented by the matching degree between the two. The document is considered relevant to the query if the matching degree is high. Natural language understanding by computer is still challenging, and thus the calculation of matching degree is still limited to the text level but not at the semantic level. A high match degree at the text level does not necessarily mean high relevance at the semantic level, and vice versa. Moreover, queries are issued by users, while documents are compiled by editors. Due to the ambiguity of natural language, users and editors are likely to use different language styles and expressions for presenting the same concepts or topics. As a result, the search system may suffer from the so-called query-document mismatch problem. Specifically, when the users of a search engine and the editors of the documents use different texts to describe the same concept (e.g., "ny times" vs. "new york times"), query-document mismatch may occur. This is still one of the main challenges for search. Moving to the cross-modal IR (e.g., using text queries to retrieve image documents), the query-document mismatch problem becomes even more severe, because different modalities have different types of representations. In cross-modal retrieval, one major challenge is how to construct a matching function that can bridge the "heterogeneity gap" amongst the modalities.

To address the query-document mismatch challenge, methods have been proposed to perform matching at the semantic level, referred to as semantic matching. The key idea in the solutions is either to perform more query and document understanding to better represent the meanings of the query and document, or to construct more powerful matching functions that can bridge the semantic gap between the query and document. Both traditional machine learning approaches (Li and Xu, 2014) and deep learning approaches (Guo *et al.*, 2019b; Mitra and Craswell, 2018; Onal *et al.*, 2018) have been developed for semantic matching.

1.4 Mismatching Challenge in Recommendation

The mismatching problem is even more severe in recommendation. In search, queries and documents consist of terms in the same language,[1] making it at least meaningful to conduct direct matching on their terms. In recommendation, however, users and items are usually represented by different types of features, for example, the features of users can be the user ID, age, income level, and recent behaviors, while the features for items can be the item ID, category, price, and brand name. Since the features of users and items are from the spaces of different semantics, the naive approaches based on the matching of superficial features do not work for recommendation. More challengingly, the items can be described by multi-modal features, e.g., images of clothing products and cover images of movies, which could play a pivotal role in affecting the decision-making of users. In such visually-aware scenarios, we need to consider the cross-modal matching between users and multi-modal content.

To address the mismatching challenge in recommendation, the collaborative filtering principle has been proposed (Shi *et al.*, 2014). Collaborative Filtering (CF), which works as the fundamental basis of almost all personalized recommender systems, assumes that a user may like (consume) the items that are liked (consumed) by the similar users, for which the similarity is judged from the historical interactions (Sarwar *et al.*, 2001). However, directly evaluating the similarity between users (items) suffers from the sparsity issue, since a user only consumed a few items in the whole item space. A typical assumption to address the sparsity issue is that the user-item interaction matrix is low-rank, which thus can be estimated from low-dimensional user (and item) latent feature matrix. Then the user (item) similarity can be more reliably reflected in the latent feature matrix. This leads to the effectiveness of matrix factorization for collaborative filtering (Koren *et al.*, 2009; Rendle *et al.*, 2009), which becomes a strong CF method and an essential design for many recommender models. Besides matrix factorization, many other types of CF methods have been developed

[1] Here we do not consider cross-language information retrieval.

like neural network-based methods (He *et al.*, 2017c; Liang *et al.*, 2018) and graph-based methods (Wang *et al.*, 2019b; Ying *et al.*, 2018).

To leverage the various side information beyond the interaction matrix, such as user profiles, item attributes, and the current contexts, many generic recommender models that follow the standard supervised learning paradigm have been proposed. These models can be used in the (re-)ranking stage of a recommendation engine, e.g., by predicting the click-through rate (CTR) of an item. A representative model is factorization machine (FM) (Rendle, 2010), which extends the low-rank assumption of matrix factorization to model feature interactions. Since the expressiveness of FM is limited by its linearity and second-order interaction modeling, many later efforts complement it with neural networks for nonlinear and higher-order interaction modeling (He and Chua, 2017; Lian *et al.*, 2018; Zhou *et al.*, 2018). These neural network models have now been intensively used in industrial applications. Batmaz *et al.* (2019) and Zhang *et al.* (2019) reviewed deep learning methods for recommendation systems.

Please note that though query-document matching and user-item matching are critical for search engines and recommendation systems, these systems also include other important components. Besides matching, web search engines also include crawling, indexing, document understanding, query understanding, and ranking, etc. Recommendation systems also include components such as user modeling (profiling), indexing, caching, diversity controlling, and online exploration, etc.

1.5 Recent Advances

Though traditional machine learning was successful for matching in search and recommendation, recent advances in deep learning have brought even more significant progress to the area with a large number of deep matching models proposed. The power of deep learning models lies in the ability to learn distributed representations from the raw data (e.g., text) for the matching problem, to avoid many limitations of hand-crafted features, and to learn the representations and matching networks in an end-to-end fashion. Moreover, deep neural networks have sufficient capacity to model complicated matching tasks. They have

the flexibility of extending to cross-modal matching naturally, where the common semantic space is learned to represent data of different modalities universally. All these characteristics are helpful in handling the complexity of search and recommendation.

In search, the mismatch between query and document is more effectively addressed by deep neural networks, including the feed-forward neural networks (FFNs), convolutional neural networks (CNNs), and Recurrent neural networks (RNNs), because they have stronger capabilities in representation learning and matching function learning. Most notably, Bidirectional Encoder Representations from Transformers (BERT) has significantly enhanced the accuracy of matching in search and stands out as the state-of-the-art technique now.

In recommendation, recent focus has shifted from behavior-centric collaborative filtering to information-rich user-item matching as in sequential, context-aware, and knowledge graph enhanced recommendations, which are all practical scenario-driven. In terms of techniques, graph neural networks (GNNs) become an emerging tool for representation learning (Wang *et al.*, 2019a,b), because recommendation data can be naturally organized in a heterogeneous graph and GNNs have the capability to exploit such data. To handle user behavior sequence data, self-attention and BERT are also adopted, which demonstrates promising results in sequential recommendation (Sun *et al.*, 2019; Yuan *et al.*, 2020).

1.6 About This Survey

This survey focuses on the fundamental problems of matching in search and recommendation. State-of-the-art matching solutions using deep learning are described. A unified view of search and recommendation from matching is provided. The ideas and solutions explained may motivate industrial practitioners to turn the research results into products. The methods and the discussions may help academic researchers to develop new approaches. The unified view may bring researchers in the search and the recommendation communities together and inspire them to explore new directions.

The survey is organized as follows: Section 2 describes the traditional machine learning approaches to matching for search and recommendation; Section 3 gives a general formulation of deep matching methods; Section 4 and Section 5 describe the details of the deep learning approaches to search and recommendation respectively. Each section includes the representation learning-based approaches and matching function learning-based approaches; Section 6 summarizes the survey and discusses open problems. Sections 2, 3, 4, and 5 are self-contained, and the readers can choose to read on the basis of their interest and need.

Note that deep learning for search and recommendation is a very hot topic of research. As such, this survey does not try to cover all related works in the fields of information retrieval and recommender systems. Instead, we discuss the most representative approaches of the two fields from the viewpoint of matching, aiming to summarize their key ideas which are general and essential. In particular, this survey covers the representative work before 2019.

Several previous FnTIR issues have given detailed introductions to related topics. One issue (Li and Xu, 2014) introduces the traditional machine learning approaches to the semantic matching problem, particularly in web search. Our survey in this issue is very different from it in the sense that (1) it focuses on the newly developed deep learning methods, and (2) it considers both search and recommendation. Mitra and Craswell (2018) conducted a comprehensive survey on deep neural networks for information retrieval, referred to as Neural IR. Bast *et al.* (2016) carries out a survey on the techniques and systems of semantic search, which means search with keyword queries, structured queries, and natural language queries, to documents, knowledge bases, and their combinations.

Several surveys and tutorials have been made on deep learning for information retrieval and recommendation. For example, Onal *et al.* (2018) have explained neural models for ad-hoc retrieval, query understanding, question answering, sponsored search, and similar item retrieval. Zhang *et al.* (2019) reviews deep learning-based recommendation methods according to the taxonomy of deep learning techniques, e.g., MLP, CNN, RNN, autoencoder-based, and so on. Other related

surveys and tutorials include Kenter *et al.* (2017), Li and Lu (2016), Guo *et al.* (2019b), Batmaz *et al.* (2019), and Zhang *et al.* (2019). They all quite differ from this survey, which summarizes existing work from the perspective of matching (e.g., input representations and the way for matching).

This survey focuses on state-of-the-art matching techniques using deep learning. We expect that the readers have a certain knowledge of search and recommendation. Those who are not familiar with the areas may consult existing materials (e.g., Adomavicius and Tuzhilin, 2005; Croft *et al.*, 2009; Li and Xu, 2014; Liu, 2009; Ricci *et al.*, 2015). We also assume that the readers have sufficient knowledge of machine learning, particularly deep learning.

2

Traditional Matching Models

Methods for conducting query-document matching in search and user-item matching in recommendation using traditional machine learning techniques have been proposed. The methods can be formalized within a more general framework, called by us "learning to match". Besides search and recommendation, it is also applicable to other applications such as paraphrasing, question answering, and natural language dialogue. This section first gives a formal definition of learning to match. Then, it introduces traditional learning to match methods developed for search and recommendation. Finally, it provides further reading in this direction.

2.1 Learning to Match

2.1.1 Matching Function

The learning to match problem can be defined as follows. Suppose that there are two spaces $\mathcal{X}$ and $\mathcal{Y}$. A class of matching functions $\mathcal{F} = \{f(x, y)\}$ is defined on two objects from the two spaces $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, where each function $f\colon \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{R}$ represents the matching

degree between the two objects x and y. The two objects x and y, and their relationship can be described with a set of features $\Phi(x, y)$.

The matching function $f(x, y)$ can be a linear combination of features:

$$f(x, y) = \langle \mathbf{w}, \Phi(x, y) \rangle,$$

where $\mathbf{w}$ is the parameter vector. It can also be a generalized linear model, a tree model, or a neural network.

2.1.2 Learning of Matching Functions

Supervised learning can be employed to learn the parameters of the matching function f, as shown in Figure 2.1. Supervised learning for matching typically consists of two phases: offline learning and online matching. In offline learning, a set of training instances $D = \{(x_1, y_1, r_1), \ldots, (x_N, y_N, r_N)\}$ is given, where r_i is a Boolean value or real number indicating the matching degree between objects x_i and y_i, and N is the size of training data. Learning is conducted to choose a matching function $f \in \mathcal{F}$ that can perform the best in matching. In online matching, given a test instance (a pair of objects) $(x, y) \in \mathcal{X} \times \mathcal{Y}$,

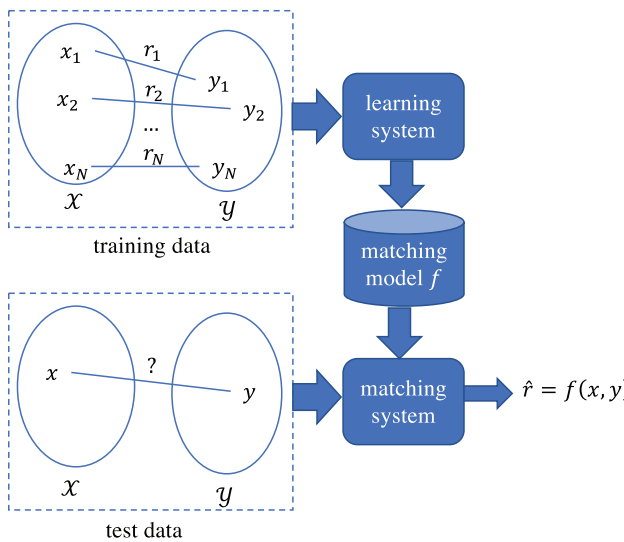

Figure 2.1: Supervised learning for matching.

the learned matching function f is utilized to predict the matching degree between the object pair denoted as $f(x, y)$.

Similar to other supervised learning problems, we can define the goal of learning to match as minimizing a loss function, which represents how much accuracy the matching function can achieve on the training data as well as the test data. More specifically, given the training data D, the learning amounts to solving the following problem:

$$\arg\min_{f \in \mathcal{F}} L(D, f) + \Omega(f).$$

The objective consists of two parts: the empirical loss $L(D, f)$ measures the overall loss incurred by the matching function f on training data, and the regularizer $\Omega(f)$ prevents overfitting to the training data. $\Omega(f)$ is typically chosen to impose a penalty on the complexity of f. Popular regularizers include ℓ_1, ℓ_2, and a mixture of them.

Different definitions of the empirical loss function $L(D, f)$ lead to different types of learning to match algorithms. Three types of loss functions, respectively referred to as pointwise loss function, pairwise loss function, and listwise loss function, have been popularly used in the literature (Cao *et al.*, 2006; He *et al.*, 2017c; Joachims, 2002; Nallapati, 2004; Rendle *et al.*, 2009). Next, we briefly describe the three types of loss functions.

Pointwise Loss Function

The pointwise loss function is defined only on one instance, i.e., a source object and a target object. Suppose that there is a pair of objects (x, y) with the true matching degree of r. Further, suppose the predicted matching degree of (x, y) given by the matching model is $f(x, y)$. The pointwise loss function is defined as a measure representing the disagreement between the matching degrees, denoted as $\ell^{\text{point}}(r, f(x, y))$. The closer $f(x, y)$ is to r, the less value the loss function has.

In learning, given the training dataset $D = \{(x_1, y_1, r_1), \ldots, (x_N, y_N, r_N)\}$, we are to minimize the total loss on the training data, or the sum of the losses of object pairs:

$$L^{\text{point}}(D, f) = \sum_{i=1}^{N} \ell^{\text{point}}(f(x_i, y_i), r_i), \tag{2.1}$$

where r_i is the ground-truth matching degree of training instance (x_i, y_i).

As an example of the pointwise loss, Mean Square Error (MSE) is a widely used loss function. Given a labeled instance (x, y, r) and the matching model f, the MSE is defined as:

$$\ell^{\text{MSE}} = (f(x, y) - r)^2.$$

Another example is the cross-entropy loss function. Cross-entropy loss function assumes that $r \in \{0, 1\}$ where 1 indicates relevant and 0 otherwise. It further assumes that $f(x, y) \in [0, 1]$ is the predicted probability that x and y are relevant. Then, the cross-entropy loss is defined as:

$$\ell^{\text{cross-entropy}} = -r \log f(x, y) - (1 - r) \log(1 - f(x, y)).$$

Pairwise Loss Function

Suppose that there are two pairs of objects (x, y^+) and (x, y^-), with one of the objects x being shared. We call x source object (e.g., query or user) and y^+ and y^- target objects (e.g., documents or items). Further suppose that there exists an order between the objects y^+ and y^- given the object x, denoted as $r^+ \succ r^-$. Here r^+ and r^- denote the matching degrees of (x, y^+) and (x, y^-) respectively. The order relations between objects can be explicitly or implicitly obtained.

We use $f(x, y^+)$ and $f(x, y^-)$ to denote the matching degrees of (x, y^+) and (x, y^-) given by the matching model f, respectively. The pairwise loss function is defined as a measure representing the disagreement between the matching degrees and the order relation, denoted as $\ell^{\text{pair}}(f(x, y^+), f(x, y^-))$. The larger $f(x, y^+)$ is than $f(x, y^-)$, the less value the loss function has.

In learning, given the training dataset D, a set of ordered object pairs P is derived as follows:

$$P = \{(x, y^+, y^-) \mid (x, y^+, r^+) \in D \wedge (x, y^-, r^-) \in D \wedge r^+ \succ r^-\},$$

The total empirical loss on the training data is the sum of the losses over the ordered object pairs:

$$L^{\text{pair}}(P, f) = \sum_{(x, y^+, y^-) \in P} \ell^{\text{pair}}(f(x, y^+), f(x, y^-)). \tag{2.2}$$

We can see that the pairwise loss function is defined on ordered pairs of objects.

As an example, the pairwise hinge loss is commonly adopted. Given a preference pair (x, y^+, y^-) and the matching model f, the pairwise hinge loss is defined as

$$\ell^{\text{pairwise-hinge}} = \max\{0, 1 - f(x, y^+) + f(x, y^-)\}.$$

Another common choice of pairwise loss in recommendation is the Bayesian Personalized Ranking (BPR) loss (Rendle *et al.*, 2009), which aims to maximize the margin between the prediction of the positive instance and that of negative instance:

$$\ell^{\text{pairwise-BPR}} = -\ln \sigma(f(x, y^+) - f(x, y^-)),$$

where $\sigma(\cdot)$ is the sigmoid function.

Listwise Loss Function

In search and recommendation, a source object (e.g., a query or a user) is usually related to multiple target objects (e.g., multiple documents or items). The evaluation measures for search and recommendation usually treat a list of target objects as a whole. It is reasonable, therefore, to define the loss function over a list of target objects, called listwise loss function. Suppose that a source object x is related to multiple target objects $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$, and the corresponding true matching degrees are $\mathbf{r} = \{r_1, r_2, \ldots, y_N\}$, respectively. The predicted matching degrees by f between x and $y_1, y_2, \ldots, y_N$ are $\hat{\mathbf{r}} = \{f(x, y_1), \ldots, f(x, y_N)\}$. The listwise loss function is defined as a measure to represent the disagreement between the true matching degrees and predicted matching degrees, denoted as $\ell^{\text{list}}(\hat{\mathbf{r}}, \mathbf{r})$. The more the predicted matching degrees in $\hat{\mathbf{r}}$ agree with the true matching degrees in $\mathbf{r}$, the lower value the loss function has. In learning, given the training data $D = \{(x_i, \mathbf{y}_i, \mathbf{r}_i)\}_{i=1}^M$, the empirical loss function is defined as the sum of the listwise losses over the training instances:

$$L^{\text{list}}(D, f) = \sum_{(x, \mathbf{y}, \mathbf{r}) \in D} \ell^{\text{list}}(\hat{\mathbf{r}}, \mathbf{r}). \tag{2.3}$$

As an example of listwise loss function, some methods define it as the negative probability of the relevant object given the other irrelevant objects. Specifically, let us assume that there exists only one relevant document in $\mathbf{y}$ denoted as y^+. Then, the list of labeled objects can be written as $(x, \mathbf{y} = \{y^+, y_1^-, \ldots, y_M^-\})$, where $y_1^-, \ldots, y_M^-$ are the M irrelevant objects. The list-wise loss function can be defined as the negative probability that y^+ is relevant given x:

$$\ell^{\text{prob}} = -P(y^+ \mid x) = -\frac{\exp(\lambda f(x, y^+))}{\sum_{y \in \mathbf{y}} \exp(\lambda f(x, y))},$$

where $\lambda > 0$ is a parameter.

Relation with Learning to Rank

We view learning to match and learning to rank as two different machine learning problems, although they are strongly related. Learning to rank (Li, 2011; Liu, 2009) is to learn a function represented as $g(x, y)$ where x and y can be query and document in search and user and item in recommendation respectively. In search, for example, the ranking function $g(x, y)$ may contain features about the relations between x and y, as well as features on x and features on y. In contrast, the matching function $f(x, y)$ only contains features about the relations between x and y.

Usually the matching function $f(x, y)$ is trained first and then the ranking function $g(x, y)$ is trained with $f(x, y)$ being a feature. For ranking, determination of the order of multiple objects is the key, while for matching, determination of the relation between two objects is the key. When the ranking function $g(x, y)$ only consists of the matching function $f(x, y)$, one only needs to employ learning to match.

In search, the features on x can be semantic categories of query x and the features on y can be PageRank score and URL length of document y. The features defined by the matching function $f(x, y)$ can be BM25 in traditional IR or a function learned by traditional machine learning or deep learning. The ranking function $g(x, y)$ can be implemented by LambdaMART (Burges, 2010) which is an algorithm of traditional machine learning. Table 2.1 lists some key differences between learning to matching and learning to rank.

Table 2.1: Learning to match vs. learning to rank

	Learning to Match	**Learning to Rank**
Prediction	Matching degree between query and document	Ranking list of documents
Model	$f(x, y)$	$g(x, y_1), \ldots, g(x, y_N)$
Challenge	Mismatch	Correct ranking on the top

Recently, researchers find that the univariate scoring paradigm in traditional IR is sub-optimal because it fails to capture inter-document relationships and local context information. Ranking models that directly rank a list of documents together with multivariate scoring functions have been developed (Ai *et al.*, 2018; Bello *et al.*, 2018; Jiang *et al.*, 2019b; Pang *et al.*, 2020). Similar efforts have been made in recommendation (Pei *et al.*, 2019). Therefore, the problems of matching and ranking can be even more distinctively separated in that sense.

2.2 Matching Models in Search and Recommendation

Next, we give an overview of matching models in search and recommendation, and introduce the approaches of matching in a latent space.

2.2.1 Matching Models in Search

When applied to search, learning to match can be described as follows. A set of query-document pairs $D = \{(q_1, d_1, r_1), (q_2, d_2, r_2), \ldots, (q_N, d_N, r_N)\}$ are given as training data, where q_i, d_i, and r_i $(i = 1, \ldots, N)$ denote a query, a document, and the query-document matching degree (relevance), respectively. Each tuple $(q, d, r) \in D$ is generated in the following way: query q is generated according to probability distribution $P(q)$, document d is generated according to conditional probability distribution $P(d \mid q)$, and relevance r is generated according to conditional probability distribution $P(r \mid q, d)$. This corresponds to the fact: queries are submitted to the search systems independently, documents associated with a query are retrieved with the query words, and the relevance of a document with respect to a query is determined by the contents

of the query and document. Human labeled data or click-through data can be used as training data.

The goal of learning to match for search is to automatically learn a matching model represented as a scoring function $f(q, d)$ (or as a conditional probability distribution $P(r \mid q, d)$). The learning problem can be formalized as minimization of the pointwise loss function in Equation (2.1), the pairwise loss function in Equation (2.2), or the listwise loss function in Equation (2.3). The learned model must have the generalization capability to conduct matching on unseen test data.

2.2.2 Matching Models in Recommendation

When applied to recommendation, learning to match can be described as follows. A set of M users $\mathcal{U} = \{u_1, \ldots, u_M\}$ and a set of N items $\mathcal{V} = \{i_1, \ldots, i_N\}$, as well as a rating matrix $R \in \mathbf{R}^{M \times N}$ are given, where each entry r_{ij} denotes the rating (interaction) of user u_i on item i_j and r_{ij} is set to zero if the rating (interaction) is unknown. We assume that each tuple (u_i, i_j, r_{ij}) is generated in the following way: user u_i is generated according to probability distribution $P(u_i)$, item i_j is generated according to probability distribution $P(i_j)$, and rating r_{ij} is generated according to conditional probability distribution $P(r_{ij} \mid u_i, i_j)$. This corresponds to the fact: users and items are presented in the recommender systems, and the interest of a user on an item is determined by the known interest of users on items in the system.

The goal of learning to match for recommendation is to learn the underlying matching model $f(u_i, i_j)$ that can make predictions on the ratings (interactions) of the zero entries in matrix R:

$$\hat{r}_{ij} = f(u_i, i_j),$$

where $\hat{r}_{ij}$ denotes the estimated affinity score between user u_i and item i_j. In this way, given a user, a subset of items with the highest scores with respect to the user can be recommended. The learning problem can be formalized as minimizing the regularized empirical loss function. Still, the loss function can be either pointwise loss, pairwise loss, or listwise loss as in Equation (2.1), Equation (2.2), or Equation (2.3). If the loss function is pointwise loss like square loss or cross-entropy, the

model learning becomes a regression or classification problem, where the prediction value indicates the strength of interest. If the loss function is pairwise loss or listwise loss, it becomes a genuine ranking problem, where the prediction value indicates the relative strengths of interest on items for a user.

2.2.3 Matching in Latent Space

As explained in Section 1, the fundamental challenge to matching in search and recommendation is the mismatch between objects from two different spaces (query and document, as well as user and item). One effective approach to dealing with the challenge is to represent the two objects in matching in a common space, and to perform the task of matching in the common space. As the space may not have an explicit definition, it is often referred to as "latent space". This is the fundamental idea behind the approach of matching in the latent space, for both search (Wu *et al.*, 2013b) and recommendation (Koren *et al.*, 2009).

Without loss of generality, let us take search as an example. Figure 2.2 illustrates query-document matching in a latent space. There are three spaces: query space, document space, and latent space, and there exist semantic gaps between the query space and document space. Queries and documents are first mapped to the latent space, and then matching is conducted in the latent space. Two mapping functions specify the mappings from the query space and document space into the latent space. The uses of different types of mapping functions (e.g., linear and non-linear) and similarity measures in the latent space (e.g., inner

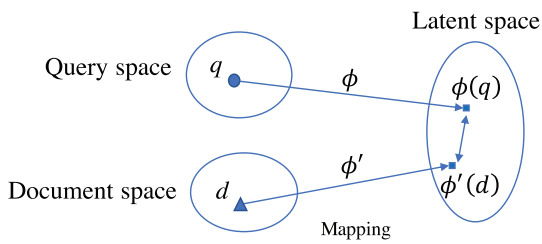

Figure 2.2: Query-document matching in latent space.

product and Euclidean distance) lead to different types of matching models.

Formally, let $\mathcal{Q}$ denote the query space (query $q \in \mathcal{Q}$) and $\mathcal{D}$ denote the document space (document $d \in \mathcal{D}$), respectively, and $\mathcal{H}$ denotes the latent space. The mapping function from $\mathcal{Q}$ to $\mathcal{H}$ is represented as ϕ: $\mathcal{Q} \mapsto \mathcal{H}$, where $\phi(q)$ stands for the mapped vector of q in $\mathcal{H}$. Similarly, the mapping function from $\mathcal{D}$ to $\mathcal{H}$ is represented as ϕ': $\mathcal{D} \mapsto \mathcal{H}$, where $\phi'(d)$ stands for the mapped vector of d in $\mathcal{H}$. The matching score between q and d is defined as the similarity between the mapped vectors (representations) of q and d in the latent space, i.e., $\phi(q)$ and $\phi'(d)$.

Before the prevalence of deep learning, most methods are "shallow", in the sense that linear functions and inner product are adopted as the mapping function and similarity, respectively,

$$s(q, d) = \langle \phi(q), \phi'(d) \rangle, \tag{2.4}$$

where ϕ and ϕ' denote linear functions and $\langle \cdot \rangle$ denotes inner product.

In learning of the model, training instances indicating the matching relations between queries and documents are given. For example, click-through data can be naturally used. The training data is represented as $(q_1, d_1, c_1), (q_2, d_2, c_2), \ldots, (q_N, d_N, c_N)$, where each instance is a triple of query, document, and click-number (or logarithm of click-number).

2.3 Latent Space Models in Search

Next, we introduce the matching models based on latent spaces as examples. A complete introduction to semantic matching in search can be found in Li and Xu (2014). Specifically, we briefly introduce representative methods for search that perform matching in a latent space, including Partial Least Square (PLS) (Rosipal and Krämer, 2006), Regularized Matching in Latent Space (RMLS) (Wu *et al.*, 2013b), and Supervised Semantic Indexing (SSI) (Bai *et al.*, 2009, 2010).

2.3.1 Partial Least Square

Partial Least Square (PLS) is a technique initially proposed for regression in statistics (Rosipal and Krämer, 2006). It is shown that PLS can

be employed in learning of latent space model for search (Wu *et al.*, 2013a).

Let us consider using the matching function $f(q, d)$ in Equation (2.4). Let us also assume that the mapping functions are defined as $\phi(q) = \mathbf{L}_q\mathbf{q}$ and $\phi'(d) = \mathbf{L}_d\mathbf{d}$, where $\mathbf{q}$ and $\mathbf{d}$ are feature vectors representing query q and document d, and $\mathbf{L}_q$ and $\mathbf{L}_d$ are orthonormal matrices. Thus, the matching function becomes

$$f(q, d) = \langle \mathbf{L}_q\mathbf{q}, \mathbf{L}_d\mathbf{d} \rangle, \tag{2.5}$$

where $\mathbf{L}_q$ and $\mathbf{L}_d$ are to be learned.

Given the training data, the learning of $\mathbf{L}_q$ and $\mathbf{L}_d$ amounts to optimizing the objective function (based on pointwise loss) with constraints:

$$\begin{aligned} \underset{\mathbf{L}_q, \mathbf{L}_d}{\text{argmax}} &= \sum_{(q_i, d_i)} c_i f(q_i, d_i), \\ s.t. \quad & \mathbf{L}_q\mathbf{L}_q^T = \mathbf{I}, \quad \mathbf{L}_d\mathbf{L}_d^T = \mathbf{I}, \end{aligned} \tag{2.6}$$

where (q_i, d_i) is a pair of query and document, c_i is the click number of the pair, and $\mathbf{I}$ is the identity matrix. This is a non-convex optimization problem, however, the global optimum exists and can be achieved by employing SVD (Singular Value Decomposition) (Wu *et al.*, 2013a,b).

2.3.2 Regularized Mapping to Latent Space

PLS assumes that the mapping functions are orthonormal matrices. When the training data size is large, learning becomes hard because it needs to solve SVD, which is of high time complexity. To address the issue, Wu *et al.* (2013b) propose a new method called Regularized Matching in Latent Space (RMLS), in which the orthonormality constraints in PLS are replaced with ℓ_1 and ℓ_2 regularizations, under the assumption that the solutions are sparse. In this way, there is no need to solve SVD, and the optimization can be carried out efficiently. Specifically, the optimization problem becomes that of minimizing the

objective function (based on pointwise loss) with ℓ_1 and ℓ_2 constraints:

$$\begin{aligned} \underset{\mathbf{L}_q, \mathbf{L}_d}{\text{argmax}} &= \sum_{(q_i, d_i)} c_i f(q_i, d_i), \\ s.t. \quad \forall j: \ |l_q^j| \leq \theta_q, \quad |l_d^j| \leq \theta_d, \quad &\|l_q^j\| \leq \tau_q, \quad \|l_d^j\| \leq \tau_d, \end{aligned} \tag{2.7}$$

where (q_i, d_i) is a pair of query and document, c_i is the click number of the pair, $\mathbf{L}_q$ and $\mathbf{L}_d$ are linear mapping matrices, l_q^j and l_d^j are the j-th row vectors of $\mathbf{L}_q$ and $\mathbf{L}_d$, and θ_q, θ_d, τ_q, and τ_d are thresholds. $|\cdot|$ and $\|\cdot\|$ denote ℓ_1 and ℓ_2 norms, respectively. Note that the regularizations are defined on the row vectors, not column vectors. The use of ℓ_2 norm is to avoid a trivial solution.

The learning in RMLS is also a non-convex optimization problem. There is no guarantee that a globally optimal solution can be found. One way to cope with the problem is to employ alternative optimization, that is, to first fix $\mathbf{L}_q$ and optimize $\mathbf{L}_d$, and then fix $\mathbf{L}_d$ and optimize $\mathbf{L}_q$, and repeat until convergence. One can easily see that the optimization can be decomposed and performed row by row and column by column of the matrices. That means that the learning in RMLS can be easily parallelized and scaled up.

The matching function in Equation (2.5) can be rewritten as a bilinear function

$$\begin{aligned} f(q, d) &= (\mathbf{L}_q \mathbf{q})^T (\mathbf{L}_d \mathbf{d}) \\ &= \mathbf{q}^T (\mathbf{L}_q^T \mathbf{L}_d) \mathbf{d} \\ &= \mathbf{q}^T \mathbf{W} \mathbf{d}, \end{aligned} \tag{2.8}$$

where $\mathbf{W} = \mathbf{L}_q^T \mathbf{L}_d$. Thus, both PLS and RMLS can be viewed as a method of learning a bilinear function with matrix $\mathbf{W}$ which can be factorized into two low-rank matrices $\mathbf{L}_q$ and $\mathbf{L}_d$.

2.3.3 Supervised Semantic Indexing

A special assumption can be made in PLS and RMLS; that is, the query space and the document space have the same dimensions. For example, when both queries and documents are represented as bag-of-words, they have the same dimensions in the query and document spaces. As a result, $\mathbf{W}$ in Equation (2.8) becomes a square matrix. The method of

Supervised Semantic Indexing (SSI) proposed by Bai *et al.* (2009, 2010) exactly makes the assumption. It further represents $\mathbf{W}$ as a low rank and diagonal preserving matrix:

$$\mathbf{W} = \mathbf{L}_q^T \mathbf{L}_d + \mathbf{I},$$

where $\mathbf{I}$ denotes the identity matrix. Thus, the matching function becomes

$$f(q, d) = \mathbf{q}^T (\mathbf{L}_q^T \mathbf{L}_d + \mathbf{I})\mathbf{d}.$$

The addition of the identity matrix means that SSI makes a tradeoff between the use of a low-dimensional latent space and the use of a classical Vector Space Model (VSM).[1] The diagonal of matrix $\mathbf{W}$ gives a score to each term which occurs in both query and document.

Given click-through data, ordered query-document pairs are first derived, denoted as $P = \{(q_1, d_1^+, d_1^-), \ldots, (q_M, d_M^+, d_M^-)\}$ where d^+ is ranked higher than d^- and M is the number of pairs. The goal of learning is to choose $\mathbf{L}_q$ and $\mathbf{L}_d$ such that $f(q, d^+) > f(q, d^-)$ holds for all pairs. A pairwise loss function is utilized. The optimization problem becomes

$$\begin{aligned} &\underset{\mathbf{L}_q, \mathbf{L}_d}{\operatorname{argmin}} \sum_{(q, d^+, d^-) \in P} \max(0, 1 - (f(q, d^+) - f(q, d^-))), \\ &= \underset{\mathbf{L}_q, \mathbf{L}_d}{\operatorname{argmin}} \sum_{(q, d^+, d^-) \in P} \max(0, 1 - \mathbf{q}^T (\mathbf{L}_q^T \mathbf{L}_d + \mathbf{I})(\mathbf{d}^+ - \mathbf{d}^-)). \quad (2.9) \end{aligned}$$

The learning of SSI is also a non-convex optimization problem and there is no guarantee to find the global optimal solution. The optimization can be conduced in a way similar to that of RMLS.

2.4 Latent Space Models in Recommendation

Next, we briefly introduce representative methods for recommendation that perform matching in a latent space, including Biased Matrix Factorization (BMF) (Koren *et al.*, 2009), Factored Item Similarity Model (FISM) (Kabbur *et al.*, 2013), and Factorization Machine (FM) (Rendle, 2010).

[1] If $\mathbf{W} = \mathbf{I}$, then the model degenerates to VSM. If $\mathbf{W} = \mathbf{L}_q^T \mathbf{L}_d$, then the model is equivalent to the models of PLS and RMLS.

2.4.1 Biased Matrix Factorization

Biased Matrix Factorization (BMF) is a model proposed for predicting the ratings of users (Koren *et al.*, 2009), i.e., formalizing recommendation as a regression task. It is developed during the period of Netflix Challenge and quickly becomes popular due to its simplicity and effectiveness. The matching model can be formulated as:

$$f(u, i) = b_0 + b_u + b_i + \mathbf{p}_u^T \mathbf{q}_i, \tag{2.10}$$

where b_0, b_u, and b_i are scalars denoting the overall bias, user bias, and item bias in rating scores, and $\mathbf{p}_u$ and $\mathbf{q}_i$ are latent vectors denoting the user and the item. This can be interpreted as just using the IDs of users and items as features of them, and projecting the IDs into the latent space with two linear functions. Let $\mathbf{u}$ be the one-hot ID vector of user u and $\mathbf{i}$ be the one-hot ID vector of item i, and $\mathbf{P}$ be the user projection matrix and $\mathbf{Q}$ be the item projection matrix. Then we can express the model under the mapping framework of Equation (2.4):

$$f(u, i) = \langle \phi(u), \phi'(i) \rangle = \langle [b_0, b_u, 1, \mathbf{P} \cdot \mathbf{u}], [1, 1, b_i, \mathbf{Q} \cdot \mathbf{i}] \rangle, \tag{2.11}$$

where $[\cdot, \cdot]$ denotes vector concatenation.

Given the training data, the learning of model parameters ($\Theta = \{b_0, b_u, b_i, \mathbf{P}, \mathbf{Q}\}$) becomes optimizing the pointwise regression error with regularization:

$$\arg\min_{\Theta} \sum_{(u,i) \in \mathcal{D}} (R_{ui} - f(u, i))^2 + \lambda ||\Theta||^2, \tag{2.12}$$

where $\mathcal{D}$ denotes all observed ratings, R_{ui} denotes the rating for (u, i), and λ is the L_2 regularization coefficient. As it is a non-convex optimization problem, alternating least squares (He *et al.*, 2016b) or stochastic gradient decent (Koren *et al.*, 2009) are typically employed, which cannot guarantee to find a global optimum.

2.4.2 Factored Item Similarity Model

Factored Item Similarity Model (FISM) (Kabbur *et al.*, 2013) adopts the assumption of item-based collaborative filtering, i.e., users will prefer items that are similar to what they have chosen so far. To do so, FISM

uses the items that a user has chosen to represent the user and projects the combined items into the latent space. The model formulation of FISM is:

$$f(u,i) = b_u + b_i + d_u^{-\alpha}\Big(\sum_{j\in\mathcal{D}_u^+}\mathbf{p}_j\Big)^T\mathbf{q}_i, \tag{2.13}$$

where $\mathcal{D}_u^+$ denotes the items that user u has chosen, d_u denotes the number of such items, and $d_u^{-\alpha}$ represents normalization across users. $\mathbf{q}_i$ is the latent vector of target item i, and $\mathbf{p}_j$ is the latent vector of historical item j chosen by user u. FISM treats $\mathbf{p}_j^T\mathbf{q}_j$ as the similarity between items i and j, and aggregates the similarities of the target item i and the historical items of user u.

FISM employs a pairwise loss and learns a model from binary implicit feedback. Let $\mathcal{U}$ be all users, the total pairwise loss is given by

$$\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{D}_u^+}\sum_{j\notin\mathcal{D}_u^+}(f(u,i) - f(u,j) - 1)^2 + \lambda||\Theta||^2, \tag{2.14}$$

which forces the score of a positive (observed) instance to be larger than that of a negative (unobserved) instance with a margin of one. Another pairwise loss, the *Bayesian Personalized Ranking* (BPR) (Rendle *et al.*, 2009) loss, is also widely used,

$$\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{D}_u^+}\sum_{j\notin\mathcal{D}_u^+} -\ln\sigma(f(u,i) - f(u,j)) + \lambda||\Theta||^2, \tag{2.15}$$

where $\sigma(\cdot)$ denotes the sigmoid function that converts the difference of scores to a probability value between zero and one, and thus the loss has a probabilistic interpretation. The main difference between the two losses is that BPR enforces the differences between positive and negative instances as large as possible, without explicitly defining a margin. Both pairwise losses can be seen as a surrogate of the AUC metric, which measures how many pairs of items are correctly ranked by the model.

2.4.3 Factorization Machine

Factorization Machine (FM) (Rendle, 2010) is developed as a general model for recommendation. In addition to the interaction information

between users and items, FM also incorporates side information of users and items, such as user profiles (e.g., age, gender, etc.), item attributes (e.g., category, tags, etc.) and contexts (e.g., time, location, etc.). The input to FM is a feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ that can contain any features for representing the matching function, as described above. Consequently, FM casts the matching problem as a supervised learning problem. It projects the features into the latent space, modeling their interactions with the inner product:

$$f(\mathbf{x}) = b_0 + \sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{v}_i^T \mathbf{v}_j x_i x_j, \tag{2.16}$$

where b_0 is the bias, b_i is the weight of feature x_i, and $\mathbf{v}_i$ is the latent vector of feature x_i. Given the fact that the input vector $\mathbf{x}$ can be large but sparse, e.g., multi-hot encoding of categorical features, FM only captures interactions between non-zero features (with term $x_i x_j$).

FM is a very general model in the sense that feeding different input features into the model will lead to different formulations of the model. For example, when $\mathbf{x}$ only retains the user ID and target item ID, FM becomes the BMF model; and when $\mathbf{x}$ only keeps the IDs of user's historically chosen items and target item ID, FM becomes the FISM model. Other prevalent latent space models such as SVD++ (Koren, 2008) and Factorized Personalized Markov Chain (FPMC) (Rendle *et al.*, 2010) can also be subsumed by FM with proper feature engineering.

2.5 Further Reading

Query reformulation is another way to address the query-document mismatch in search, that is, to transform the query to another query which can do better matching. Query transformation includes spelling error correction of the query. For example, Brill and Moore (2000) propose a source channel model, and Wang *et al.* (2011) propose a discriminative method for the task. Query transformation also includes query segmentation (Bendersky *et al.*, 2011; Bergsma and Wang, 2007; Guo *et al.*, 2008). Inspired by Statistical Machine Translation (SMT), researchers also consider leveraging translation technologies to deal with query document mismatch, assuming that query is in one language and

document is in another. Berger and Lafferty (1999) exploit a word-based translation model to perform the task. Gao *et al.* (2004) propose using the phrase-based translation model to capture the dependencies between words in the query and document title. Topic models can also be utilized to address the mismatch problem. A simple and effective approach is to use a linear combination of term matching score and topic matching score (Hofmann, 1999). Probabilistic topic models are also employed to smooth document language models (or query language models) (Wei and Croft, 2006; Yi and Allan, 2009). Li and Xu (2014) provides a comprehensive survey on the traditional machine learning approaches to semantic matching in search.

In recommendation, besides the classical latent factor models introduced, other types of methods have been developed. For example, matching can be conducted on the original interaction space with predefined heuristics, like item-based CF (Sarwar *et al.*, 2001) and unified user-based and item-based CF (Wang *et al.*, 2006). User-item interactions can be organized as a bipartite graph, on which random walk is performed to estimate the relevance between any two nodes (a user and an item, two users, or two items) (Eksombatchai *et al.*, 2018; He *et al.*, 2017b). One can also model the generation process of user-item interactions using probabilistic graphical models (Salakhutdinov and Mnih, 2007). To incorporate various side information such as the user profiles and contexts, besides the FM model introduced, tensor factorization (Karatzoglou *et al.*, 2010) and collective matrix factorization (He *et al.*, 2014) are also exploited. We refer the readers to two survey papers on the traditional matching methods for recommendation (Adomavicius and Tuzhilin, 2005; Shi *et al.*, 2014).

3

Deep Learning for Matching

Recent years have observed tremendous progress in applications of deep learning into matching in search and recommendation (Guo *et al.*, 2019b; Naumov *et al.*, 2019). The main reason for the success is due to deep learning's strong ability in learning of representations for inputs (i.e., queries, documents, users, and items) and learning of nonlinear functions for matching. In this section, we first give an overview of Deep Learning (DL) techniques and then describe a general framework, typical architectures, and designing principles of deep learning for matching in search and recommendation.

3.1 Overview of Deep Learning

3.1.1 Deep Neural Networks

Deep neural networks are complicated nonlinear functions from input to output. In this subsection, we describe several neural network architectures that are widely used. Please refer to Goodfellow *et al.* (2016) for a more detailed introduction.

Feed-Forward Neural Networks

The Feed-forward Neural Networks (FFN), also called Multilayer Perceptron (MLP), are neural networks consisting of multiple layers of units, which are connected layer by layer without a loop. It is called feed-forward because the signal only moves in one direction in the network, from the input layer, through the hidden layers, and finally to the output layer. The feed-forward neural networks can be utilized to approximate any function, for example, a regressor $y = f(\mathbf{x})$ that maps a vector input $\mathbf{x}$ to a scalar output y.

Figure 3.1 shows a feedforward neural network with one hidden layer. For an input vector $\mathbf{x}$, the neural network returns an output vector $\mathbf{y}$. The model is defined as the following non-linear function

$$\mathbf{y} = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2),$$

where σ is the element-wise sigmoid function, $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1$ and $\mathbf{b}_2$ are model parameters to be determined in learning. To construct a deeper neural network, one only needs to stack more layers on the top of the network. Besides sigmoid function, other functions such as tanh and Rectified Linear Units (ReLU) are also utilized.

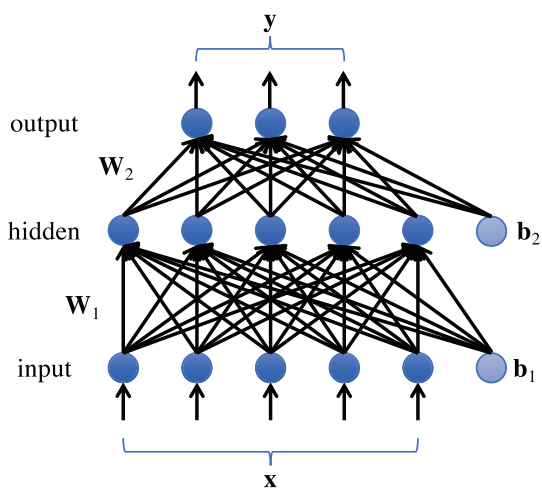

Figure 3.1: A simple feed-forward neural network.

In learning, training data of input-output pairs are fed into the network as ground-truth. A loss is calculated for each instance by contrasting the ground truth and the prediction by the network, and the training is performed by adjusting the parameters so that the total loss is minimized. The well-known back-propagation algorithm is employed to conduct the minimization.

Convolutional Neural Networks

Convolutional Neural Networks are neural networks that make use of convolution operations in at least one of the layers. They are specialized neural networks for processing data that has a grid-like structure, e.g., time serious data (1-D grid of time intervals) and image data (2-D grid of pixels).

As shown in Figure 3.2, a typical convolutional network consists of multiple stacked layers: convolutional layer, detector layer, and pooling layer. In the convolutional layer, convolution functions are applied in parallel to produce a set of linear activations. In the detector layer, the set of linear activations are run through a nonlinear activation function.

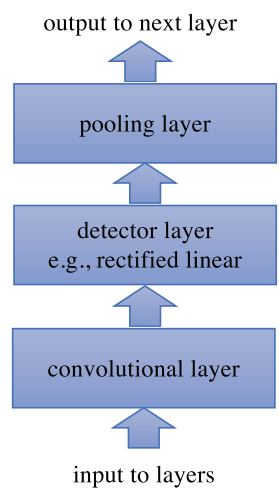

Figure 3.2: Typical convolutional layers.

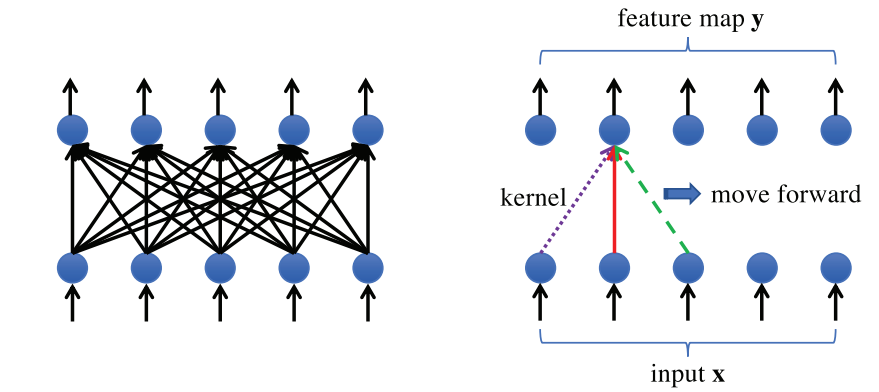

Figure 3.3: Fully connected layer and convolutional layer.

In the pooling layer, pooling functions are used to further modify the set of outputs.

Figure 3.3 shows a comparison between a fully connected layer and a convolutional layer. The fully connected layer uses a weight matrix to model the global features of all input units, and therefore it has dense connections between layers. The convolutional layer, on the other hand, uses convolutional kernel vectors (or matrices) to model the local features of each position (unit), where the weights of kernels are shared across positions (units). Thus it has much sparse connections between layers. Specifically, in one-dimension, given convolution kernel $\mathbf{w}$ and input vector $\mathbf{x}$, the output of position (unit) i is decided as:

$$y_i = (\mathbf{x} * \mathbf{w})(i) = \sum_a x_{i-a} w_a,$$

where "*" denotes the convolution operator.

In the two-dimension case, given the convolution kernel $\mathbf{K}$ and the input matrix $\mathbf{X}$, the output of position (unit) (i, j) is decided as:

$$Y_{i,j} = (\mathbf{X} * \mathbf{W})(i, j) = \sum_a \sum_b X_{i-a,j-b} W_{a,b}.$$

In the second detector layer, nonlinear activation functions such as sigmoid, tanh, and ReLU are usually utilized.

In the third pooling layer, pooling functions such as max-pooling, average-pooling, and min-pooling are exploited. For example, in max-pooling, the output at each position is determined as the maximum of outputs of a kernel in the neighborhood.

Recurrent Neural Networks

Recurrent Neural Networks are neural networks for processing sequence data $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$. Unlike FFNs which can only handle one instance at a time, RNN can handle a long sequence of instances with a variable length.

As shown in Figure 3.4, an RNN shares the same parameters at different positions. That is, at each position, the output is from the same function of input at the current position as well as output at the previous position. The outputs are determined with the same rule across positions. Specifically, at each position $t = 1, \ldots, T$, the output vector $\mathbf{o}^{(t)}$ and the state of the hidden unit $\mathbf{h}^{(t)}$ are calculated as

$$\mathbf{h}^{(t)} = \tanh(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_1),$$
$$\mathbf{o}^{(t)} = \text{softmax}(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{b}_2),$$

where $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}_1$, and $\mathbf{b}_2$ are model parameters.

Unfolding an RNN, we obtain a deep neural network with many stages (right part of Figure 3.4) that share parameters over all stages.

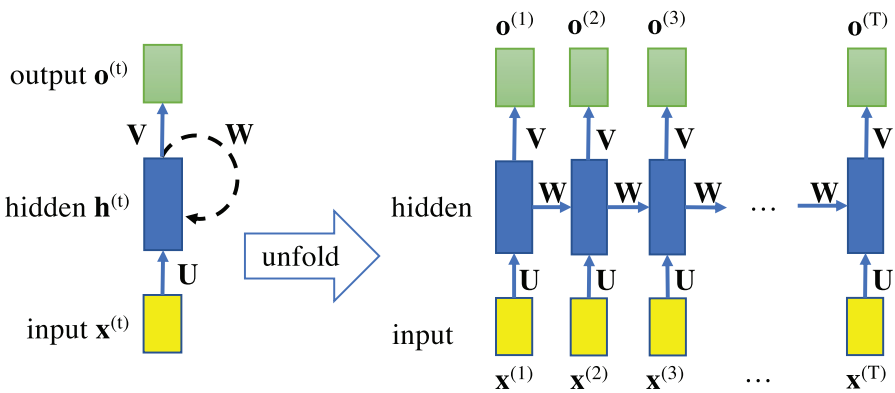

Figure 3.4: An RNN and its unfolded form.

The number of stages is determined by the length of the input sequence. This structure makes it challenging to learn the model of RNN, because the gradients propagated over the positions may either vanish or explode. To address the issue, variations of RNN such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) are proposed.

Attention-Based Neural Networks

Attention is a useful tool in deep learning. It is originally proposed to dynamically and selectively collect information from the source sentence in an encoder-decoder model in neural machine translation (NMT) (Bahdanau *et al.*, 2015).

Attention based Model: Figure 3.5 shows an encoder-decoder model with the additive attention mechanism. Suppose that there are an input sequence $(w_1, w_2, \ldots, w_M)$ of length M and an output sequence $(y_1, y_2, \ldots, y_N)$ of length N. The encoder (e.g., an RNN) creates a hidden state $\mathbf{h}_i$ at each input position w_i $(i = 1, \ldots, M)$. The decoder constructs a hidden state $\mathbf{s}_t = f(\mathbf{s}_{t-1}, y_{t-1}, \mathbf{c}_t)$ at output position t $(t = 1, \ldots, N)$, where f is the function of the decoder, $\mathbf{s}_{t-1}$ and y_{t-1} are the state and output of the previous position, and $\mathbf{c}_t$ is the context

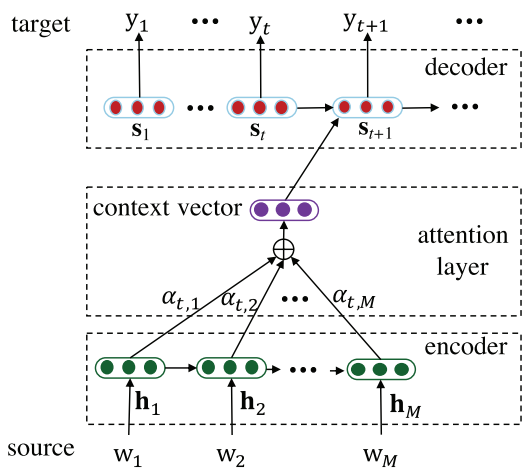

Figure 3.5: The encoder-decoder model with additive attention mechanism.

vector at the position. The context vector is defined as the sum of hidden states at all input positions, weighted by attention scores:

$$\mathbf{c}_t = \sum_{i=1}^{M} \alpha_{t,i} \mathbf{h}_i,$$

and the attention score $\alpha_{t,i}$ is defined as:

$$\alpha_{t,i} = \frac{\exp(g(\mathbf{s}_t, \mathbf{h}_i))}{\sum_{j=1}^{M} \exp(g(\mathbf{s}_t, \mathbf{h}_j))}.$$

The function g is determined by the hidden state of the previous output position and the context vector of the current output position. It can be defined as, for example, a feed-forward network with a single hidden layer:

$$g(\mathbf{s}_t, \mathbf{h}_i)) = \mathbf{v}_a^T \tanh(\mathbf{W}_a[\mathbf{s}_t, \mathbf{h}_i]),$$

where $\mathbf{v}_a$ and $\mathbf{W}_a$ are parameters.

We can see that the context vector $\mathbf{c}_t$ selectively and dynamically combines the information of the entire input sequence with the attention mechanism. Compared to the traditional encoder-decoder model in which only a single vector is used, multiple vectors are used to capture the information of the encoder regardless of the distance.

Transformer: Transformer (Vaswani *et al.*, 2017) is another attention-based neural network under the encoder and decoder framework. Different from the aforementioned model which sequentially reads the input sequence (left-to-right or right-to-left), Transformer reads the entire input sequence at once. The characteristic enables it to learn the model by considering both the left and the right context of a word.

As shown in Figure 3.6, Transformer consists of an encoder for transforming the input sequence of words into a sequence of vectors (internal representation) and a decoder for generating an output sequence of words one by one given the internal representation. The encoder is a stack of encoder components with an identical structure, and the decoder is also a stack of decoder components with an identical structure, where the encoder and decoder have the same number components.

Each encoder component or layer consists of a self-attention sub-layer and a feed-forward network sub-layer. It receives a sequence of

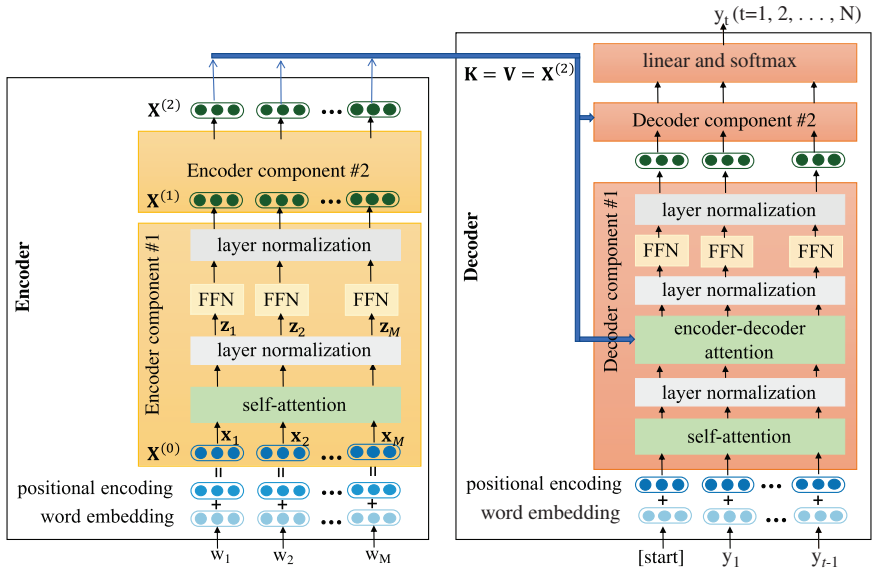

Figure 3.6: An example transformer with two-layers of encoder and decoder. The encoder is responsible for creating internal representations of input words. The decoder is applied multiple times to generate the output words one by one. Note that the residual connections around the sub-layers are not shown in the figure.

vectors (packed into a matrix) as input, processes the vectors with the self-attention sub-layer, and then passes them through the feed-forward network sub-layer. Finally, it sends the vectors as output to the next encoder component. Specifically, the input is a sequence of words $(w_1, w_2, \ldots, w_M)$ with length M. Each word w_i is represented by a vector $\mathbf{x}_i$ as sum of the word embedding and positional encoding of it. The vectors are packed into a matrix $\mathbf{X}^{(0)} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]^T$. The self-attention sub-layer converts $\mathbf{X}^{(0)}$ into $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_M]^T$ through self-attention defined as

$$\mathbf{Z} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V},$$

where $\mathbf{K}, \mathbf{V}$, and $\mathbf{Q}$ are matrices of key vectors, value vectors, and query vectors respectively; d_k is the dimensionality of key vector; $\mathbf{K}$ is the resulting matrix consisting of M vectors. The matrices $\mathbf{K}, \mathbf{V}$, and $\mathbf{Q}$

are calculated as

$$\mathbf{Q} = \mathbf{X}^{(0)}\mathbf{W}_Q,$$
$$\mathbf{K} = \mathbf{X}^{(0)}\mathbf{W}_K,$$
$$\mathbf{V} = \mathbf{X}^{(0)}\mathbf{W}_V,$$

where $\mathbf{W}_Q, \mathbf{W}_K$ and $\mathbf{W}_V$ are embedding matrices. After that, the vectors $\mathbf{z}_i$ in $\mathbf{Z}$ are independently processed by the feed-forward network sub-layer. In each sub-layer, a residual connection is employed, followed by layer-normalization. The output of the encoder component is represented as $\mathbf{X}^{(1)} = [\mathbf{x}_1^{(1)}, \mathbf{x}_2^{(1)}, \ldots, \mathbf{x}_M^{(1)}]^T$. $\mathbf{X}^{(1)}$ is then fed into the next encoder component. The encoder finally outputs the vectors (representations) corresponding to all input words, denoted as $\mathbf{X}^{enc}$. There are multiple heads at each attention sub-layer and we omit the description on it here.

Each decoder component or layer in the decoder consists of a self-attention sub-layer, an encoder-decoder attention sub-layer, and a feed-forward network sub-layer. The sub-layers have the same architecture as that of the encoder component. After encoding, the output of the encoder is used to represent the key and value vectors: $\mathbf{K} = \mathbf{V} = \mathbf{X}^{enc}$, which are then used for "encoder-decoder attention" in each decoder component. The decoder sequentially generates words for all output positions $1, 2, \ldots, N$. At each position $1 \leq t \leq N$, the bottom decoder component receives the previously outputted words "[start], $y_1, \ldots, y_{t-1}$", masks the future positions, and outputs internal representations for the next decoder component. Finally, the word at position t, denoted as v_t, is selected according to a probabilistic distribution generated by the softmax layer on the top decoder component. The process is repeated until a special symbol (e.g., "[end]") is generated, or the maximal length is reached.

Autoencoders

Autoencoders are neural networks that aim to learn the hidden information of the input, by compressing the input into a latent-space representation and then reconstructing the output from the representation. In the model, high-dimensional data is first converted into a

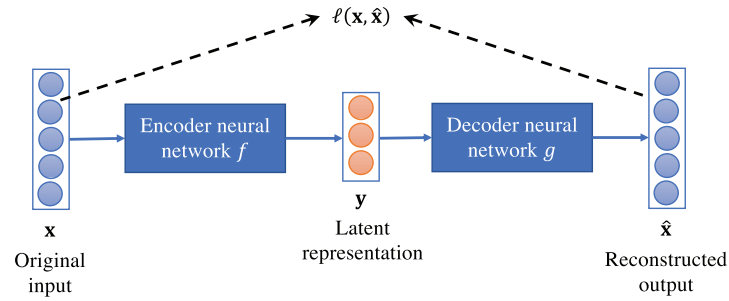

Figure 3.7: Architecture of an autoencoder.

low-dimensional latent representation by a multilayer encoder neural network. Then, the data is reconstructed from the latent representation by a multilayer decoder neural network. Therefore, it consists of two parts: an encoder $\mathbf{y} = f(\mathbf{x})$; and a decoder $\hat{\mathbf{x}} = g(\mathbf{y})$. The autoencoder as a whole can be described by the function $g(f(x)) = \hat{\mathbf{x}}$ where it is expected that $\hat{\mathbf{x}}$ is as close to the original input $\mathbf{x}$ as possible.

Vanilla Autoencoder (Hinton and Salakhutdinov, 2006): Figure 3.7 shows the architecture of the vanilla autoencoder model. The encoder and decoder can be neural networks with one hidden layer or deep neural networks with multiple hidden layers. The goal in learning is to construct the encoder and decoder so as to make the output $\hat{\mathbf{x}}$ as close to the input $\mathbf{x}$ as possible, i.e., $g(f(\mathbf{x})) \approx \mathbf{x}$. Suppose that we are given a training dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the encoder and decoder (f and g) can be learned through minimization of the squared-error loss function:

$$\min_{f,g} \sum_{i=1}^{N} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) = \sum_{i=1}^{N} \|\mathbf{x}_i - g(f(\mathbf{x}_i))\|^2.$$

An alternative is minimization of the reconstruction cross-entropy, where it is assumed that $\mathbf{x}$ and $\hat{\mathbf{x}}$ are either bit vectors or probability vectors,

$$\min_{f,g} \sum_{i=1}^{N} \ell_H(\mathbf{x}_i, \hat{\mathbf{x}}_i) = -\sum_{i=1}^{N} \sum_{k=1}^{D} [x_i^k \log \hat{x}_i^k + (1 - x_i^k) \log(1 - \hat{x}_i^k)],$$

where D is the dimensionality of input, and x_i^k and $\hat{x}_i^k$ are the k-th dimension of $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$, respectively. The optimization in learning of autoencoder is typically carried out by stochastic gradient descent.

By limiting the dimensionality of latent representation $\mathbf{y}$, autoencoder is forced to learn a representation in which the most salient features of data are uncovered in the "compressed" low dimensional representation.

Denoising Autoencoder (DAE): To deal with corruption in the data, DAE is also proposed (Vincent *et al.*, 2008) as an extension of vanilla autoencoder. It is assumed that DAE receives a corrupted data sample as input and predicts the original uncorrupted data sample as output. First, a corrupted input $\tilde{\mathbf{x}}$ is created from the original uncorrupted input $\mathbf{x}$ through a random process $\tilde{\mathbf{x}} \sim q_D(\tilde{\mathbf{x}} \mid \mathbf{x})$, where q_D is a conditional distribution over corrupted data samples conditioned on an uncorrupted data sample. Next, the corrupted input $\tilde{\mathbf{x}}$ is mapped into a latent representation $\mathbf{y} = f(\tilde{\mathbf{x}})$ and then the latent representation is mapped into a reconstructed output $\hat{\mathbf{x}} = g(f(\tilde{\mathbf{x}}))$, as shown in Figure 3.8. The parameters of the encoder and the decoder are learned to minimize the average reconstruction error (e.g., cross entropy error) between the input $\mathbf{x}$ and reconstructed output $\hat{\mathbf{x}}$.

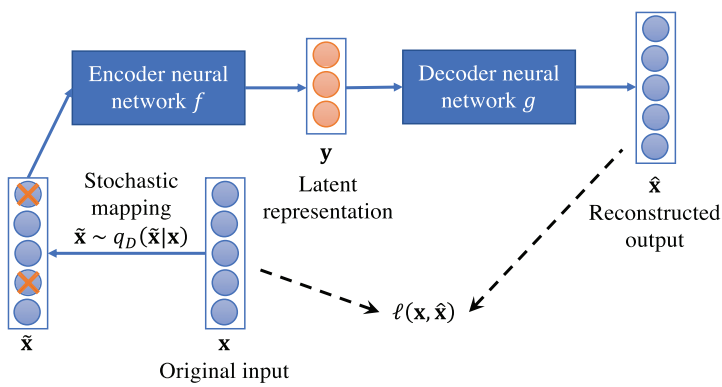

Figure 3.8: Architecture of a de-noising autoencoder.

Other types of autoencoder include Sparse autoencoder (Ranzato *et al.*, 2007), Variational Autoencoder (Kingma and Welling, 2014), and Convolutional Autoencoder (Masci *et al.*, 2011).

3.1.2 Representation Learning

The strong ability in representation learning is the primary reason for the big success of deep learning. In this subsection, we introduce several methods of representation learning that are successfully applied in matching, including methods of learning word embeddings and contextualized word representations.

Word Embeddings

Word embedding is a basic way of representing words in Natural Language Processing (NLP) and Information Retrieval (IR). Embeddings of words are usually created based on the assumption that the meaning of a word can be determined by its contexts in documents.

Word2Vec: Mikolov *et al.* (2013) proposed the Word2Vec tool and made word embedding popular. Word2Vec learns embeddings of words from a large corpus using shallow neural networks in an unsupervised manner. There are two specific methods in Word2Vec: Continuous Bag of Words (CBOW) and Skip Gram.

As shown in Figure 3.9, CBOW takes the context of a word as input and predicts the word from the context. It aims to learn two matrices, $\mathbf{U} \in R^{D \times |V|}$ and $\mathbf{W} \in R^{|V| \times D}$, where D is the size of embedding space, V is the vocabulary, and $|V|$ is the size of the vocabulary. $\mathbf{U}$ is the input word matrix such that the i-th column of $\mathbf{U}$, denoted as $\mathbf{u}_i$, is a D-dimensional embedding vector of input word w_i. Similarly, $\mathbf{W}$ is the output word matrix such that the j-th row of $\mathbf{W}$, denoted as $\mathbf{w}_j$, is a D-dimensional embedding vector of output word w_j. Note that each word w_i has two embedding vectors, i.e., the input word vector $\mathbf{u}_i$ and the output vector $\mathbf{w}_i$. Given a corpus, the learning of the $\mathbf{U}$ and $\mathbf{W}$ amounts to the following calculation:

1. Selecting a word sequence of size $2m+1$: $(w_{c-m}, \ldots, w_{c-1}, w_c, w_{c+1}, \ldots, w_{c+m})$ and generating a one-hot word vector for w_c, denoted

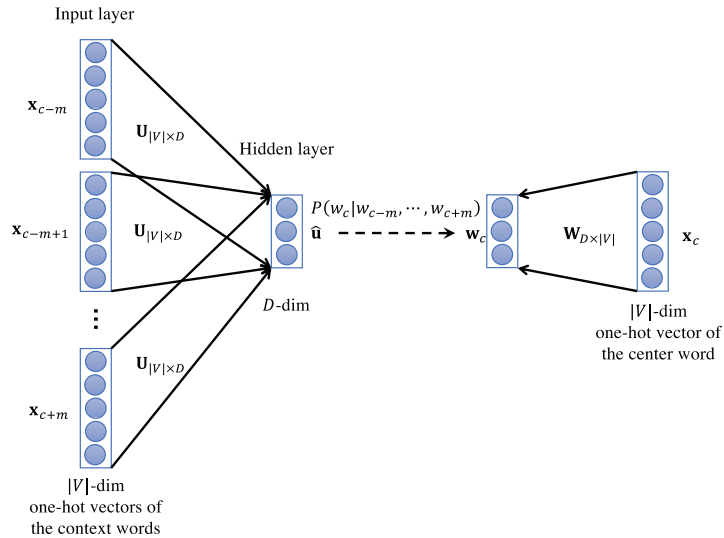

Figure 3.9: Architecture of CBOW.

as $\mathbf{x}_c$, and generating one-hot word vectors for the context of w_c, denoted as $(\mathbf{x}_{c-m}, \ldots, \mathbf{x}_{c-1}, \mathbf{x}_{c+1}, \ldots, \mathbf{x}_{c+m})$;

2. Mapping to the embedding word vectors of the context: $(\mathbf{u}_{c-m} = \mathbf{U}\mathbf{x}_{c-m}, \ldots, \mathbf{u}_{c-1} = \mathbf{U}\mathbf{x}_{c-1}, \mathbf{u}_{c+1} = \mathbf{U}\mathbf{x}_{c+1}, \mathbf{u}_{c+m} = \mathbf{U}\mathbf{x}_{c+m})$;

3. Getting the average context: $\hat{\mathbf{u}} = \frac{1}{2m}(\mathbf{u}_{c-m} + \cdots + \mathbf{u}_{c-1} + \mathbf{u}_{c+1} \ldots \mathbf{u}_{c+m})$;

4. Mapping to the embedding vectors of the center word: $\mathbf{w}_c = \mathbf{W}\mathbf{x}_c$;

5. Assuming that the center word is "generated" by the average context $\hat{\mathbf{u}}$.

CBOW adjusts the parameters in $\mathbf{U}$ and $\mathbf{W}$ such that

$$\begin{aligned}\underset{\mathbf{U},\mathbf{W}}{\operatorname{argmin}}\, \ell = & -\prod_c \log P(w_c \mid w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\ = & -\prod_c \log \frac{\exp\{\mathbf{w}_c^T \hat{\mathbf{u}}\}}{\sum_{k=1}^{|V|} \exp\{\mathbf{w}_k^T \hat{\mathbf{u}}\}}.\end{aligned}$$

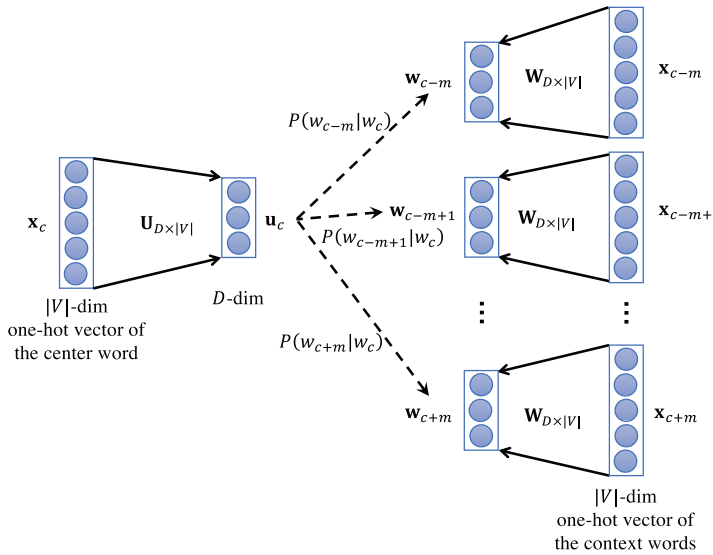

Figure 3.10: Architecture of skip gram.

The setting of Skip Gram is similar to that of CBOW, while input and output are swapped. As shown in Figure 3.10, the input of Skip Gram is the one-hot vector of the center word, and the outputs are the vectors of the words in the context. Skip Gram also aims to learn two matrices $\mathbf{U} \in R^{D\times|V|}$ and $\mathbf{W} \in R^{|V|\times D}$. Given a text corpus, the learning amounts to the following calculation:

1. Selecting a word sequence of size $2m+1$: $(w_{c-m}, \ldots, w_{c-1}, w_c, w_{c+1}, \ldots, w_{c+m})$ and generating a one-hot word vector for w_c, denoted as $\mathbf{x}_c$, and generating one-hot word vectors for the context of w_c, denoted as $(\mathbf{x}_{c-m}, \ldots, \mathbf{x}_{c-1}, \mathbf{x}_{c+1}, \ldots, \mathbf{x}_{c+m})$;

2. Mapping to the embedding word vector of the center word: $\mathbf{u}_c = \mathbf{U}\mathbf{x}_c$;

3. Mapping to the embedding word vectors of the context: $\mathbf{w}_{c-m} = \mathbf{W}\mathbf{x}_{c-m}, \ldots, \mathbf{w}_{c-1} = \mathbf{W}\mathbf{x}_{c-1}, \mathbf{w}_{c+1} = \mathbf{W}\mathbf{x}_{c+1}, \ldots, \mathbf{w}_{c+m} = \mathbf{W}\mathbf{x}_{c+m}$;

4. Assuming that the embedding vectors of the context $\mathbf{w}_{c-m}, \ldots, \mathbf{w}_{c-1}, \mathbf{w}_{c+1}, \ldots, \mathbf{w}_{c+m}$ are "generated" by the center word $\mathbf{u}_c$.

Skip Gram adjusts the parameters in $\mathbf{U}$ and $\mathbf{W}$ such that

$$\begin{aligned}\underset{\mathbf{U},\mathbf{W}}{\operatorname{argmin}}\, \ell &= -\log \prod_{c} \prod_{j=0;j\neq m}^{2m} P(w_{c-m+j} \mid w_c) \\ &= -\log \prod_{c} \prod_{j=0;j\neq m}^{2m} \frac{\exp\{\mathbf{w}_{c-m+j}^T \mathbf{u}_c\}}{\sum_{k=1}^{|V|} \exp\{\mathbf{w}_k^T \mathbf{u}_c\}}.\end{aligned}$$

Besides Word2Vec, a number of word embedding (and document embedding) models have been developed in recent years, including the GloVe (Global Vectors) (Pennington *et al.*, 2014), fastText,[1] and doc2Vec (Le and Mikolov, 2014).

Contextualised Word Representations

The classical word embedding models (e.g., Word2Vec and GloVe) have a fundamental shortcoming: they generate and utilize the same embeddings of the same words in different contexts. Therefore, they cannot effectively deal with the context-dependent nature of words. Contextualized word embeddings aim at capturing lexical semantics in different contexts. A number of models have been developed, including ULMFiT (Universal Language Model Fine-tuning), ELMo (Peters *et al.*, 2018), GPT (Radford *et al.*, 2018), GPT-2 (Radford *et al.*, 2019), Bidirectional Encoder Representations from Transformers (BERT) (Devlin *et al.*, 2019), and XLNet (Yang *et al.*, 2019c).

Among the models, BERT is the most widely used. BERT is a mask language model (a denoising auto-encoder) that aims to reconstruct the original sentences from the corrupted ones. That is, in the pre-train phase, the input sentence is corrupted by replacing some original words with "[MASK]". The learning objective, therefore, is to predict the masked words to get the original sentence.

As illustrated in Figure 3.11, BERT employs the Transformer to learn the contextual relations between words in a text. Specifically,

[1] https://fasttext.cc/.

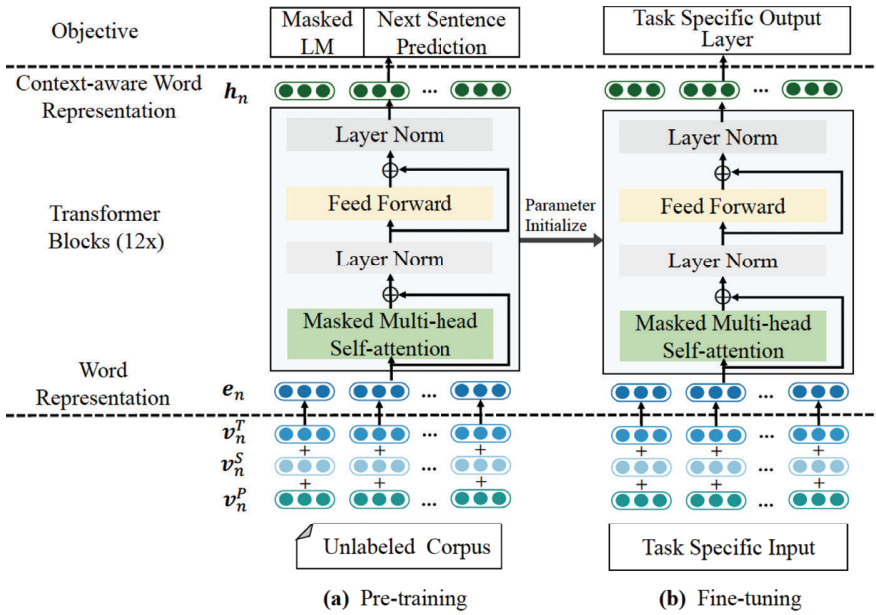

Figure 3.11: BERT training procedure: (a) The pre-training stage and transformer architecture; (b) the fine-tuning stage modifies the pre-trained parameters by task-specific training.

the model receives a pair of sentences separated by token "[SEP]" as input. The two sentences are referred to as the left context and the right context, respectively. BERT manages to capture the language representations of both the left and the right contexts by using a Transformer encoder. Formally, given a word sequence $W = \{w_1, w_2, \ldots, w_N\}$ of which consists of the words in the left context, the token "[SEP]", and the words in the right context, BERT first constructs an input representation $\mathbf{e}_n$ for each word w_n by summing the corresponding word embedding $\mathbf{v}_n^T$ (i.e., by Word2Vec), segment embedding $\mathbf{v}_n^S$ (i.e., indicate whether it is from the left context and or the right context), and position embedding $\mathbf{v}_n^P$ (i.e., indicate the position of the word in the sequence). The input representations are then fed into the layered blocks of Transformer encoder to obtain contextualized representations for the words of the pair of sentences. Each Transformer block is composed of a multi-head attention followed by a feed-forward layer.

The learning of BERT consists of two stages: pre-training and fine-tuning. In pre-training, sentence pairs collected from a large corpus are used as training data. The model parameters are determined using two training strategies: mask language modeling and next sentence prediction. (1) In mask language modeling, 15% of the randomly chosen words in the two sentences are replaced with token "[MASK]" before feeding them into the model. The training goal, then, is to predict the original masked words, based on the contexts provided by the non-masked words in the sentences. (2) In next sentence prediction, the model receives pairs of sentences as input. The training goal is to predict if the second sentence in the pair is the subsequent sentence in the original document. About 50% of the inputs are positive examples, while the other 50% of the inputs are negative examples. In pre-training, mask language modeling and next sentence prediction are conducted jointly, by minimizing the combined loss functions of the two strategies.

Fine-tuning of a pre-trained BERT model is conducted in a supervised learning manner, for generating word representations tailored for the specific task. Let us use text classification as an example. A classification layer is added on top of the BERT model to construct a classifier. Suppose that each instance in the training data consists of a word sequence $x_1, \ldots, x_M$, and a label y. The model feeds the sequence through the pre-trained BERT model, generates the representation for the "[CLS]" token, and predicts the label $\hat{y}$. The fine-tuning objective, therefore, can be defined upon the ground-truth label y and the predicted label $\hat{y}$.

3.2 Overview of Deep Learning for Matching

Deep learning for matching, referred to as deep matching, has become the state-of-the-art technologies in search and recommendation (Mitra and Craswell, 2019). Compared with the traditional machine learning approaches, the deep learning approaches improve the matching accuracy in three ways: (1) using deep neural networks to construct richer representations for matching of objects (i.e., query, document, user, and item), (2) using deep learning algorithms to construct more powerful functions for matching, and (3) learning the representations

and matching functions jointly in an end-to-end fashion. Another advantage of deep matching approaches is their flexibility of extending to multi-modal matching where the common semantic space can be learned to universally represent data of different modalities.

Various neural network architectures have been developed. Here, we give a general framework, typical architectures, and designing principles of deep matching.

3.2.1 General Framework for Deep Matching

Figure 3.12 shows a general framework of matching. The matching framework takes two matching objects as its input and outputs a numerical value to represent the matching degree. The framework has input and output layers at the bottom and the top. Between the input and output layers, there are three consecutive layers. Each layer can be implemented as a neural network or a part of a neural network:

The input layer receives the two matching objects which can be word embeddings, ID vectors, or feature vectors.

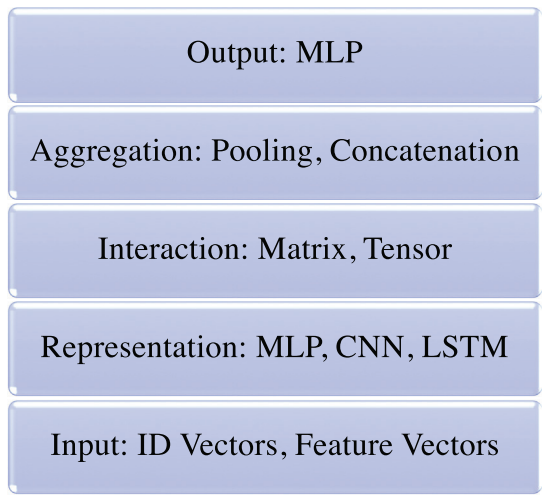

Figure 3.12: The general matching framework.

The representation layer converts the input vectors into the distributed representations. Neural networks such as MLP, CNN, and RNN can be used here, depending on the type and nature of the input.

The interaction layer compares the matching objects (i.e., two distributed representations) and outputs a number of (local or global) matching signals. Matrix and tensor can be used for storing the signals and their locations.

The aggregation layer aggregates the individual matching signals into a high-level matching vector. Operations in deep neural networks such as pooling and concatenation are usually adopted in this layer.

The output layer takes the high-level matching vector and outputs a matching score. Linear model, MLP, Neural Tensor Networks (NTN), or other neural networks can be utilized.

Neural network architectures developed so far for query-document matching in search and user-item matching in recommendation can be summarized in the framework.

3.2.2 Typical Architectures for Deep Matching

Figure 3.13 shows a typical architecture for deep matching in search (Huang *et al.*, 2013) and recommendation (He *et al.*, 2017c). In the architecture, the inputs X and Y are two texts in search or two feature vectors in recommendation. The two inputs are first processed with two neural networks independently, for creating their representations. Then, the architecture calculates the interactions between the two representations and outputs matching signals. Finally, the matching signals are aggregated to form the final matching score. A special case of the architecture is to let the two representation neural networks identical and their parameters shared (Huang *et al.*, 2013; Shen *et al.*, 2014). The simplification makes the network architecture easier to train and more robust, which is possible when both X and Y are texts.

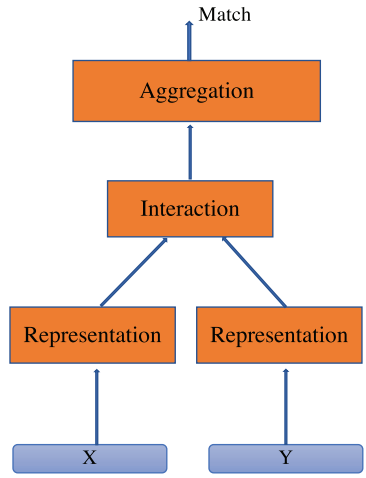

Figure 3.13: A common neural network architecture one for deep matching.

Figure 3.14 shows an architecture widely used for deep matching in search (Hu *et al.*, 2014; Pang *et al.*, 2016b). It takes two texts X and Y as input, and each word in the texts is represented by its embedding. The architecture first calculates *lexical interactions* between the two texts. The results of the lexical interactions are stored in a matrix or a tensor, for keeping the results as well as their locations. Then, the interactions at the lexical level are aggregated into the final matching score. Compared to the first architecture, this architecture has two remarkable properties: (1) conducting the interactions at the lexical level rather than at the semantic level, which is usually necessary for search; (2) storing and utilizing the location of each interaction at the next step.

Figure 3.15 shows an architecture widely used for deep matching in recommendation (He and Chua, 2017; Xin *et al.*, 2019a). The input matching objects are a user (query) and an item (document) with their feature vectors. The input vectors can be combined (concatenated). Then, the combined input vectors are processed with a neural network for creating a distributed representation (embeddings) of them. Next, the architecture calculates the interactions between the user and item, e.g., first-order interactions using linear regression, second-order interactions

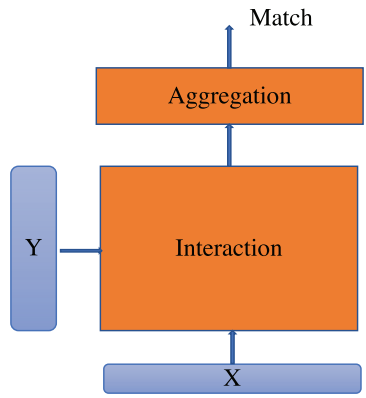

Figure 3.14: The neural network architecture two for query-document matching.

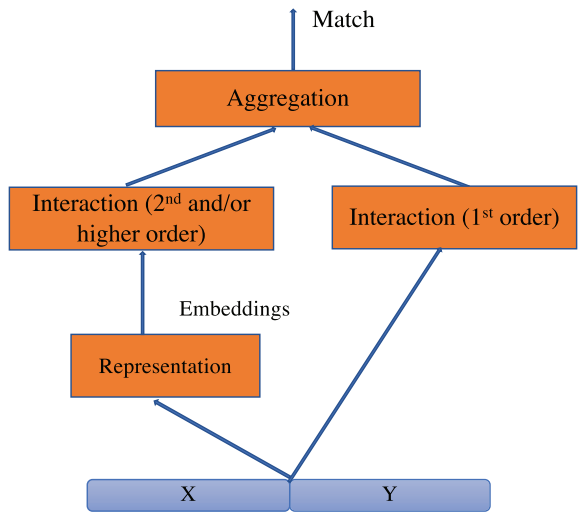

Figure 3.15: The neural network architecture for user-item matching.

using factorization machine, and higher-order interactions using MLP or CNN. Finally, these interactions are aggregated to obtain the final matching score. Please note that though it is originally designed in recommendation system literature, the hybrid structure is also popular in search. For example, the search model of Duet Mitra *et al.* (2017) adopts similar architecture for matching queries and documents.

3.2.3 Designing Principles of Deep Matching

We propose two designing principles for the development of deep matching models in search and recommendation: the modular principle and the hybrid principle.

The modular principle postulates that a matching model usually consists of multiple modules (functions), and thus the development of such a model should also take a modular approach. For example, the representation module can be implemented with CNN, RNN, or MLP, the interaction module can be a matrix or a tensor, and the aggregation module can be a pooling or concatenating operator. Different combinations of the modules result in different matching models.

The hybrid principle asserts that a combination of dichotomic techniques is helpful in the development of matching models. For example, in user-item matching in recommendation, the first-order, second-order, and higher-order interactions all contribute to the determination of the final matching degree. In query-document matching in search, the query and document can be represented with both bag-of-words and sequences of word embeddings. Furthermore, in both search and recommendation, the representation and interaction between objects can be combined using a combination of deep and wide neural networks, or nonlinear and linear models.

The next two sections will introduce the neural network architectures designed for search and recommendation with more details.

4

Deep Matching Models in Search

The deep learning approaches to query-document matching in search mainly fall into two categories: representation learning and matching function learning, whose architectures are depicted in Figure 3.13 and Figure 3.14, respectively. In the two categories, neural networks are utilized for representing queries and documents, for conducting the interactions between queries and documents, and for aggregating the matching signals. Different combinations of techniques result in different deep matching models. Table 4.1 summarizes the deep matching models in search. The first column categorizes the models as matching based on representation learning, matching based on matching function learning, and the models that combine the two approaches. The second column further categorizes the models according to how word order information is used in the initial representations. For example, "bag of letter tri-grams" and "bag of words means" means that the order of words in the queries and/or documents are not considered. "Sequence of words" means that the models utilize the word ordering information. The third column describes the types of the neural networks (or transformation functions) employed in the models. Note that we use the terms "representation learning" and "matching function learning"

Table 4.1: Deep learning approaches to query-document matching in search

	Input Representation	Network Architecture	Matching Models
Matching based on representation learning	Bag of letter tri-gram	MLP	DSSM (Huang *et al.*, 2013)
	Bag of words	Linear	NVSM (Gysel *et al.*, 2018)
	Sequence of words	MLP	SNRM (Zamani *et al.*, 2018b)
		CNN	CLSM (Shen *et al.*, 2014), ARC-I (Hu *et al.*, 2014), CNTN (Qiu and Huang, 2015), MACM (Nie *et al.*, 2018), NRM-F (Zamani *et al.*, 2018c), Multi-GranCNN (Yin and Schütze, 2015)
		RNN	LSTM-RNN (Palangi *et al.*, 2016), MV-LSTM (Wan *et al.*, 2016a)
		RNN+ Attention	MASH RNN (Jiang *et al.*, 2019a), CSRAN (Tay *et al.*, 2018b)
	Cross-modal	CNN	Deep CCA (Andrew *et al.*, 2013; Yan and Mikolajczyk, 2015), ACMR (Wang *et al.*, 2017a), m-CNNs (Ma *et al.*, 2015)
		RNN + CNN	BRNN (Karpathy and Li, 2015)
		Attention	RCM (Wang *et al.*, 2019d)

Continued.

Table 4.1: Continued

	Input Representation	Network Architecture	Matching Models
Matching based on matching function learning	Bag of words	MLP	DRMM (Guo *et al.*, 2016)
		RBF-Kernel	K-NRM (Xiong *et al.*, 2017)
		Attention	Decomposable Attention Model (Parikh *et al.*, 2016), aNMM (Yang *et al.*, 2016)
	Sequence of words	CNN	ARC-II (Hu *et al.*, 2014), MatchPyramid (Pang *et al.*, 2016b), DeepRank (Pang *et al.*, 2017a), PACRR (Hui *et al.*, 2017), Co-PACRR (Hui *et al.*, 2018)
		RNN	ESIM (Chen *et al.*, 2017b), BiMPM (Wang *et al.*, 2017c)
		Spatial RNN	Match-SRNN (Wan *et al.*, 2016b), HiNT (Fan *et al.*, 2018)
		Attention	BERT4Match (Nogueira and Cho, 2019), MIX (Chen *et al.*, 2018b), RE2 (Yang *et al.*, 2019a), ABCNN (Yin *et al.*, 2016), MCAN (Tay *et al.*, 2018d), HCRN (Tay *et al.*, 2018c), MwAN (Tan *et al.*, 2018), DIIN (Gong *et al.*, 2018), HAR (Zhu *et al.*, 2019)
		RBF-Kernel	Conv-KNRM (Dai *et al.*, 2018)
Combined	Sequence of words	CNN + MLP	Duet (Mitra *et al.*, 2017)

for ease of explanation. The matching function learning methods also learn and utilize word (and sentence) representations.

In the remaining of the section, we introduce the representative models based on representation learning in Subsection 4.1 and the models based on matching function learning in Subsection 4.2. Experimental results are also shown in each subsection.

4.1 Matching Based on Representation Learning

4.1.1 General Framework

The representation learning methods assume that queries and documents can be represented by low-dimensional and dense vectors. There are two key questions: (1) what kind of neural networks to use for creating the representations of query and document, and (2) what kind of function to use for calculating the final matching score based on the representations.

Let us suppose that there are two spaces: query space and document space. The query space contains all the queries, and the document space contains all the documents. The query space and document space can be heterogeneous, and the similarities between queries and documents across the spaces can be hard to calculate. Further, suppose that there is a new space in which both queries and documents can be mapped into, and a similarity function is also defined in the new space. In search, the similarity function can be utilized to represent the matching degrees between queries and documents. In other words, matching between query and document is conducted in the new space after mapping. Figure 4.1 shows the framework of query-document matching based on representation learning.

Formally, given query q in the query space $q \in \mathcal{Q}$ and document d in the document space $d \in \mathcal{D}$, functions ϕ_q: $\mathcal{Q} \mapsto \mathcal{H}$ and ϕ_d: $\mathcal{D} \mapsto \mathcal{H}$ represent mapping from the query space and mapping from the document space to the new space $\mathcal{H}$, respectively. The matching function between q and d is defined as

$$f(q, d) = F(\phi_q(q), \phi_d(d)),$$

where F is the similarity function defined in the new space $\mathcal{H}$.

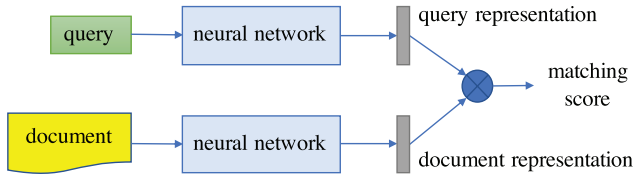

Figure 4.1: Query-document matching based on representation learning.

Different neural networks can be utilized for representing the queries and documents, as well as calculating the matching scores given the representations, resulting in different matching models. Most of the matching models (e.g., DSSM) use identical network structures for queries and documents (i.e., $\phi_q = \phi_d$). They can be generalized to having different network structures for queries and documents, respectively.

4.1.2 Representing with Feedforward Neural Networks

Feedforward neural networks are the first network architecture used to create semantic representations of queries and documents. For example, Huang *et al.* (2013) propose representing queries and documents with deep neural networks, using a model referred to as Deep Structured Semantic Models (DSSM). Figure 4.2 shows the architecture of DSSM.

DSSM first represents query q and its associated documents d's $(d_1, d_2, \ldots, d_n)$ as vectors of terms and takes the vectors as input. To overcome the difficulties resulting from the very large vocabulary size in web search, DSSM maps the term vectors to letter n-gram vectors. For example, word "good" is mapped into letter trigrams: ("#go", "goo", "ood", "od#"), where "#" denotes starting and ending marks. In this way, the dimensions of input vectors can be reduced from 500 k to 30 k, because the number of letter n-grams in English is limited. It then maps the letter n-gram vectors into output vectors of lower dimensions through deep neural networks:

$$\mathbf{y}_q = \text{DNN}(q)$$
$$\mathbf{y}_d = \text{DNN}(d),$$

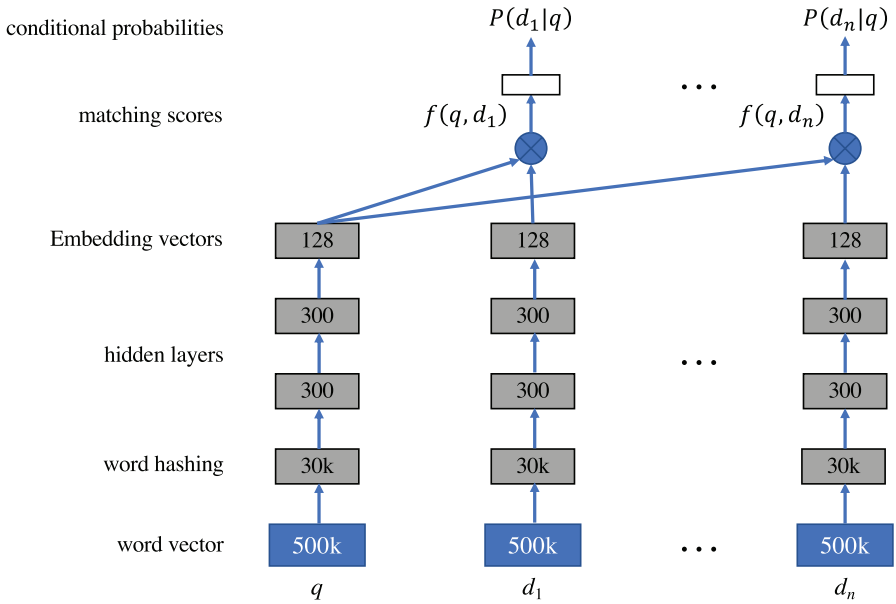

Figure 4.2: Deep structured semantic model.

where DNN$(\cdot)$ is the deep neural network used in DSSM, $\mathbf{y}_q$ and $\mathbf{y}_d$ are the output vectors that represent the hidden topics in query q and document d, respectively.

Next, DSSM takes the cosine similarity between the output vector of query (denoted as $\mathbf{y}_q$) and the output vector of document (denoted as $\mathbf{y}_d$) as matching score:

$$f(q,d) = \cos(\mathbf{y}_q, \mathbf{y}_d).$$

DSSM learns the model parameters by Maximum Likelihood Estimation (MLE) on the basis of queries, associated documents, and clicks. Specifically, given query q and a list of documents $\mathcal{D} = \{d^+, d_1^-, \ldots, d_k^-\}$, where d^+ is a clicked document and $d_1^-, \ldots, d_k^-$ are unclicked (shown but skipped) documents. The objective of learning amounts to maximizing the conditional probabilities of document d^+ given query q:

$$P(d^+ \mid q) = \frac{\exp(\lambda f(q,d^+))}{\sum_{d' \in \mathcal{D}} \exp \lambda f(q,d')},$$

where $\lambda > 0$ is a parameter.

4.1.3 Representing with Convolutional Neural Networks

Although successful in web search, researchers find that DSSM has two shortcomings. First, Deep Neural Networks contain too many parameters, which makes it difficult to train the model. Second, DSSM views a query (or a document) as a bag of words but not a sequence of words. As a result, DSSM is not effective in handling local context information between words. These two drawbacks can be addressed well with CNN. First, CNN has a smaller number of parameters than DNN, because its parameters are shared at different input positions (shift invariance), as shown in Figure 3.3. Second, the basic operations of convolution and max-pooling in CNN can keep the local context information. Therefore, CNN is a very effective architecture for representing queries and documents in search.

Convolutional Latent Semantic Model (CLSM)

Shen *et al.* (2014) propose to capture local context information for latent semantic modeling using a convolutional neural network referred to as CLSM. As shown in Figure 4.3, CLSM makes the follow modifications to DSSM for representing queries and documents:

- The input sentence (query or document) is represented as letter-trigram vectors based on word n-grams, which is a concatenation of the letter-trigram vectors of each word in a word n-gram.
- Convolutional operations are used to model context features of word n-grams. The context features of word n-grams are projected to vectors that are close to each other if they are semantically similar.
- Max-pooling operations are used to capture the sentence-level semantic features.

CLSM takes the cosine similarity between the representation vectors of query and document as the final matching score.

Similar to DSSM, the model parameters of CLSM are learned to maximize the likelihood of clicked documents given queries in the

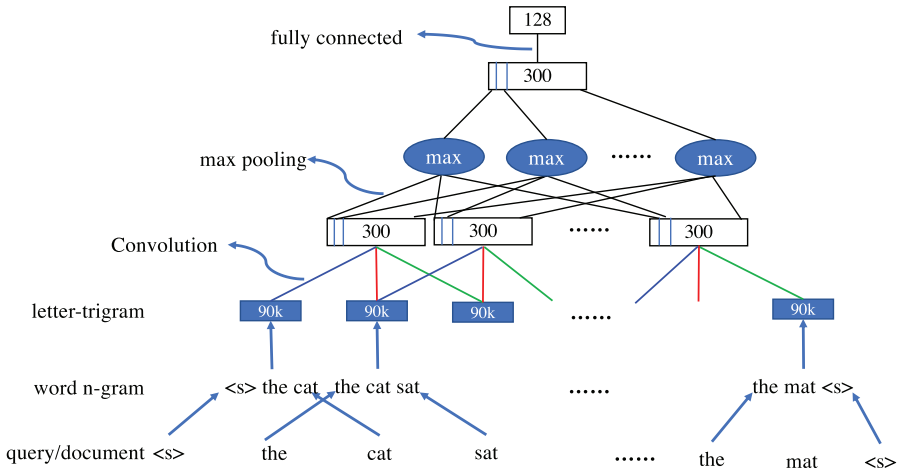

Figure 4.3: Convolutional latent semantic model.

training data. The method of calculating the conditional probability of document d^+ given query q is the same as that of DSSM.

Convolutional Matching Model (ARC-I)

Hu *et al.* (2014) propose to use convolutional architectures for matching two sentences. The model, called ARC-I, first finds the representation of each sentence with convolutional neural networks and then compares the representations of the two sentences with a multi-layer perceptron.

The ARC-I model takes the sequence of embeddings of words (i.e., word embeddings trained beforehand with word2vec (Mikolov *et al.*, 2013)) as input. The input is summarized through the layers of convolution and pooling to a fixed-length representation at the final layer. To address the problem that different sentences have different lengths, ARC-I puts zero to the elements after the last word of the sentence until the maximum length.

Formally, given query q and document d, ARC-I represents each of them as a sequence of embeddings of words. In this way, the word order information is kept. It then maps the sequence of embeddings into output vectors of lower dimensions with a 1-D convolutional neural

network:

$$\mathbf{y}_q = \text{CNN}(q)$$
$$\mathbf{y}_d = \text{CNN}(d),$$

where $\text{CNN}(\cdot)$ is the 1-D convolutional neural network, $\mathbf{y}_q$ and $\mathbf{y}_d$ are the output vectors of q and d, respectively.

To calculate the matching score, ARC-I utilizes a multiple layer perceptron:

$$f(q, d) = \mathbf{W}_2 \cdot \sigma\left(\mathbf{W}_1 \begin{bmatrix} \mathbf{y}_q \\ \mathbf{y}_d \end{bmatrix} + \mathbf{b}_1\right) + b_2,$$

where $\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2$, and b_2 are parameters, and $\sigma(\cdot)$ is the sigmoid function.

Figure 4.4 illustrates ARC-I with an example of two-layer convolutional neural networks. Given an input sentence, each word is first represented with word embedding. Then, the convolution layer generates context representations, which offer a variety of compositions of words within a three-word window and with different confidences (gray color indicates low confidence). The pooling layer then chooses between two adjacent context representations for each composition type. The output

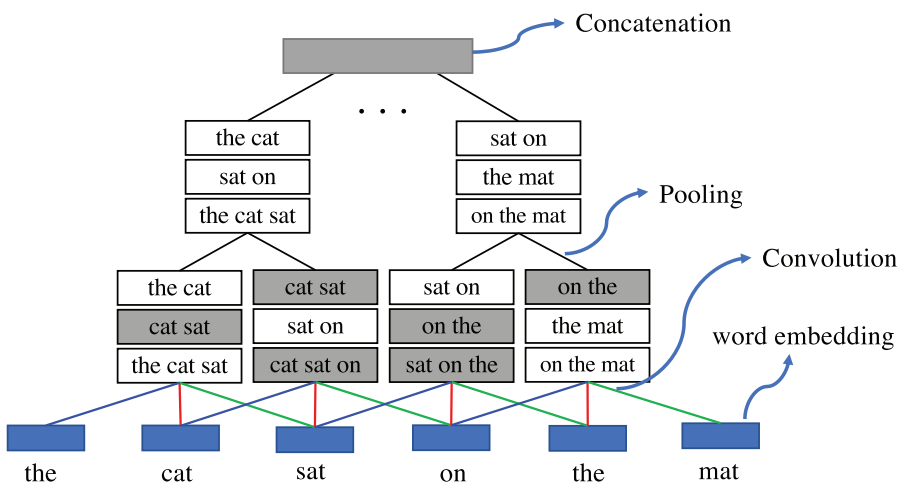

Figure 4.4: Convolutional matching model (Arc-I).

of the neural network (representation of sentence) is a concatenation of the pooling results.

Discriminative training with a large margin criterion is used in learning of the model parameters in ARC-I. Given query q, the relevant query-document pair (q, d) in the training data should receive a higher score than a query-document pair in which the relevant document is replaced with a random one, i.e., (q, d'). Thus, the ARC-I model minimizes the following objective:

$$\mathcal{L} = \sum_{(q,d)\in\mathcal{C}} \sum_{(q,d')\in\mathcal{C}'} \left[1 - f(q,d) + f(q,d')\right]_+,$$

where $\mathcal{C}$ and $\mathcal{C}'$ are collections of relevant query-document pairs and irrelevant query-document pairs, respectively.

Convolutional Neural Tensor Network (CNTN)

Neural Tensor Network (NTN) is originally proposed to explicitly model multiple interactions of relational data (Socher *et al.*, 2013). NTN has powerful representation ability and can represent multiple similarity functions, including cosine similarity, dot product, and bilinear product, etc. To model the complex interactions between query and document, Qiu and Huang (2015) propose to calculate the similarity between query and document with the tensor layer in NTN.

Similar to ARC-I, given query q and document d, CNTN first represents each of them as a sequence of word embeddings. Then each of the sequences is processed with a 1-D convolutional neural network, obtaining the low-dimensional representations:

$$\mathbf{y}_q = \text{CNN}(q)$$
$$\mathbf{y}_d = \text{CNN}(d).$$

As shown in Figure 4.5, in CNTN, the representations of the query and document are fed into NTN for calculating the matching score:

$$f(q,d) = \mathbf{u}^T \sigma\left(\mathbf{y}_q^T \mathbf{M}^{[1:r]} \mathbf{y}_d + \mathbf{V}\begin{bmatrix}\mathbf{y}_q \\ \mathbf{y}_d\end{bmatrix} + \mathbf{b}\right),$$

where σ is the element-wise sigmoid function, $\mathbf{M}^{[1:r]}$ is a tensor with r slices, and $\mathbf{V}, \mathbf{u}$, and $\mathbf{b}$ are parameters. The bilinear tensor product

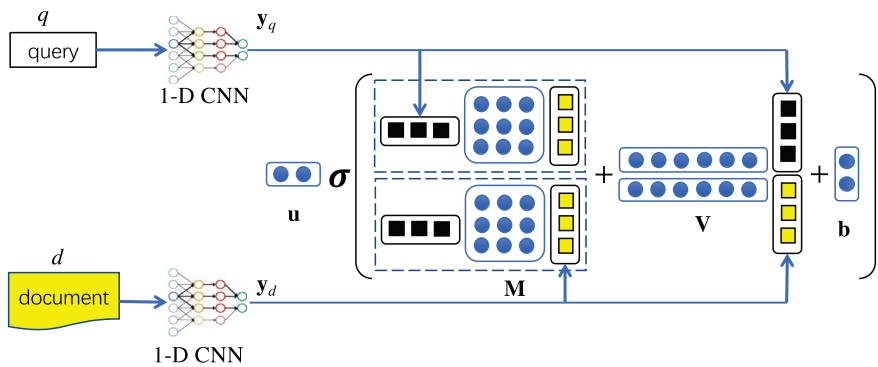

Figure 4.5: Convolutional neural tensor networks.

$\mathbf{y}_q^T \mathbf{M}^{[1:r]} \mathbf{y}_d$ returns a r-dimensional vector. One advantage of CNTN is that it can jointly model the representations and interactions. The representations of sentences are modeled with the convolutional layers, and the interactions between sentences are modeled with the tensor layer.

Similar to ARC-I, the learning of model parameters in CNTN also relies on discriminative training with a large margin criterion. Given the relevant pairs $\mathcal{C}$ and irrelevant pairs $\mathcal{C}'$, the learning amounts to minimizing:

$$\mathcal{L} = \sum_{(q,d)\in\mathcal{C}} \sum_{(q,d')\in\mathcal{C}'} [\gamma - f(q,d) + f(q,d')]_+ + \lambda \|\Theta\|^2,$$

where Θ includes the parameters in word embedding, CNN, and NTN; $\gamma > 0$ and $\lambda > 0$ are the margin and regularization hyper-parameters, respectively.

4.1.4 Representing with Recurrent Neural Networks

Given the fact that both queries and documents are texts, it is natural to apply RNN to represent the queries and documents (Palangi *et al.*, 2016). The main idea is to find a dense and low dimensional semantic representation of query (or document) by sequentially processing each word of the text. As shown in Figure 4.6, RNN sequentially processes

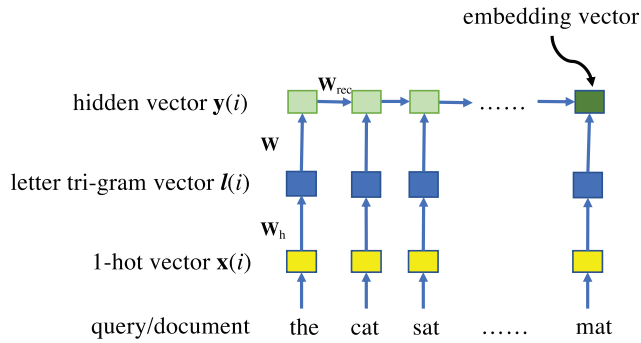

Figure 4.6: RNN for query/document representation.

each word in the input text, and the semantic representation of the last word becomes the semantic representation of the whole text.

To address the difficulty of learning long term dependency within the sequence due to the vanishing gradient problem, LSTM-RNN makes use of LSTM instead of the original RNN. During the scanning of the input text, the gates in LSTM store the long term dependencies into the cells. Specifically, the forward pass for LSTM-RNN is defined as

$$
\begin{aligned}
\mathbf{u}(t) &= \tanh(\mathbf{W}_4\mathbf{l}(t) + \mathbf{W}_{rec4}\mathbf{y}(t-1) + \mathbf{b}_4),\\
\mathbf{i}(t) &= \sigma(\mathbf{W}_3\mathbf{l}(t) + \mathbf{W}_{rec3}\mathbf{y}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3),\\
\mathbf{f}(t) &= \sigma(\mathbf{W}_2\mathbf{l}(t) + \mathbf{W}_{rec2}\mathbf{y}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2),\\
\mathbf{c}(t) &= \mathbf{f}(t) \odot \mathbf{c}(t-1) + \mathbf{i}(t) \odot \mathbf{u}(t),\\
\mathbf{o}(t) &= \sigma(\mathbf{W}_1\mathbf{l}(t) + \mathbf{W}_{rec1}\mathbf{y}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1),\\
\mathbf{y}(t) &= \mathbf{o}(t) \odot \tanh(\mathbf{c}(t)),
\end{aligned}
$$

where $\mathbf{i}(t), \mathbf{f}(t), \mathbf{o}(t), \mathbf{c}(t)$ are the input gate, forget gate, output gate, and cell state, respectively; "$\odot$" denotes the Hadamard (element-wise) product. Matrix $\mathbf{W}$ and vector $\mathbf{b}$ are model parameters. Vector $\mathbf{y}(t)$ is the representation until the t-th word. The representation of the last word $\mathbf{y}(m)$ is used as the representation of the entire text.

Given query q and document d, LSTM-RNN first creates their representation vectors $\mathbf{y}_q(|q|)$ and $\mathbf{y}_d(|d|)$, where $|\cdot|$ denotes the length of input. The matching score is defined as the cosine similarity between

the two vectors:

$$f(q, d) = \cos(\mathbf{y}_q(|q|), \mathbf{y}_d(|d|)).$$

Similar to DSSM and CLSM, LSTM-RNN also learns the model parameters by MLE on the basis of queries, associated documents, and clicked (positive) documents. Given query q and associated documents $\mathcal{D} = \{d^+, d_1^-, \ldots, d_k^-\}$, where d^+ is the clicked document and $d_1^-, \ldots, d_k^-$ are the unclicked documents. The conditional probability of document d^+ given query q is

$$P(d^+ \mid q) = \frac{\exp(\gamma f(q, d^+))}{\exp(\gamma f(q, d^+)) + \sum_{d^- \in \mathcal{D} \setminus \{d^+\}} \exp \gamma f(q, d^-)},$$

where $\gamma > 0$ is a parameter.

4.1.5 Representation Learning with Un-Supervision/Weak Supervision

Unsupervised learning and weakly supervised learning approaches are employed to learn representations of queries and documents.

Neural Vector Space Model (NVSM)

Traditionally, the low-dimensional representations of words and documents can be learned with topic models and word/document embedding methods. Gysel *et al.* (2018) present NVSM, which learns the low-dimensional representations of words and documents in a corpus using projection.

The model architecture is shown in Figure 4.7. Given a large corpus D with $|D|$ documents and the vocabulary V containing all words in the documents, the goal is to learn the representations of documents $R_D \in \Re^{|D| \times k_d}$ and the representations of words $R_V \in \Re^{|V| \times k_v}$. Note that the document representations have k_d dimensions while the word representations have k_v dimensions. NVSM first samples an n-gram with n contiguous words $B = (w_1, \ldots, w_n)$ from document d as a phrase. Then, it projects the n-gram phrase into the document space as:

$$\vec{h}(B) = \vec{h}(w_1, \ldots, w_n) = (f \circ \text{norm} \circ g)(w_1, \ldots, w_n),$$

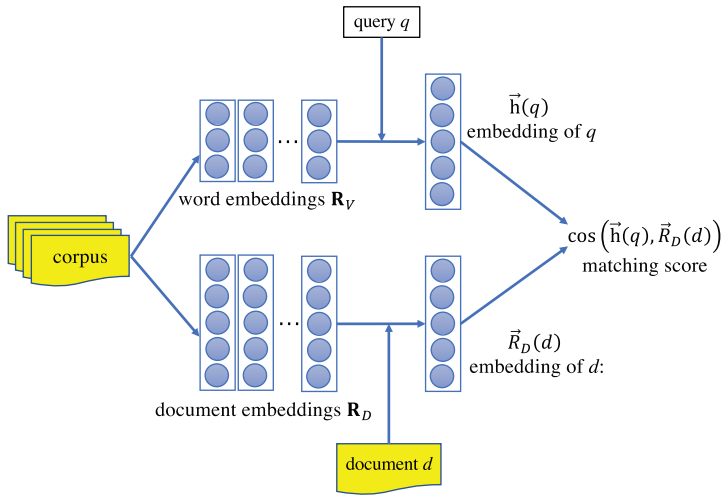

Figure 4.7: Neural vector space model.

where $g(w1, \ldots, w_n) = \frac{1}{n}\sum_{i=1}^{n} \vec{R}_V^{w_i}$ is an average of the word representations in the phrase, "norm" is the ℓ-2 normalization factor, and

$$f(\vec{x}) = \mathbf{W}\vec{x},$$

where $\mathbf{W}$ is a transformation matrix. The objective of learning is to maximize the similarity between the projected phrase representation and the document representation:

$$\max_{R_D, R_V, \mathbf{W}} \prod_{d \in D} \prod_{B:B \sim d} \sigma(\langle \vec{R}_D^d, \vec{h}(B) \rangle),$$

where "$B \sim d$" means that phrase B is sampled from document d.

In online matching, given query q and document d, NVSM projects the query to the document space, similar to that for the n-gram phrases. The matching score is calculated as the cosine similarity between the document representation and projected query representation:

$$f(q, d) = \cos(\vec{h}(q), \vec{R}_D^d).$$

Standalone Neural Ranking Model (SNRM)

Recently, researchers in the IR community propose to train neural matching and ranking models using weak supervision

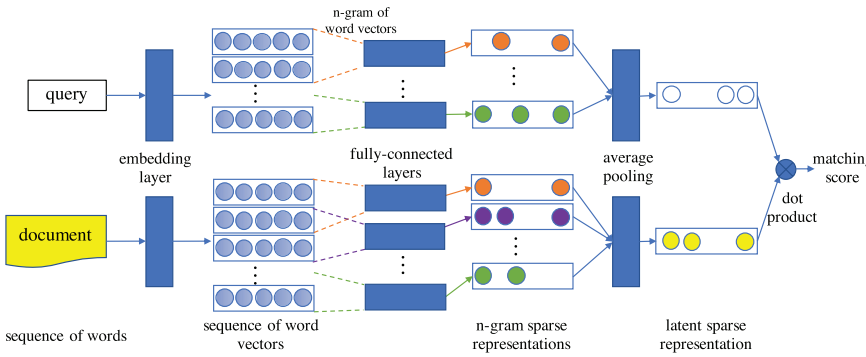

Figure 4.8: Standalone neural ranking model.

(Dehghani *et al.*, 2017), where the labels are obtained automatically without annotations by humans or exploitation of additional resources (e.g., click data). Zamani *et al.* (2018b) present SNRM by introducing sparsity into the learned latent presentations of query and document, and constructing an inverted index for the whole collection based on the representations. As shown in Figure 4.8, the SNRM network considers both query and document as a sequence of words. After going through an embedding layer, a word sequence becomes a sequence of word vectors. Then, the sequence is decomposed as a set of n-grams and processed by fully-connected layers with sparsity constraints, generating a set of high-dimensional sparse n-gram representations. Finally, an average pooling layer is used to aggregate the n-gram representations and generate the final sequence representations.

More specifically, the representation of a document d is defined as

$$\mathbf{y}_d = \frac{1}{|d| - n + 1} \sum_{i=1}^{|d|-n+1} \phi_{\text{ngram}}(w_i, w_{i+1}, \ldots, w_{i+n-1}),$$

where $w_1, w_2, \ldots, w_{|d|}$ denotes the word sequence in d and ϕ_{ngram} denotes a high-dimensional and sparse representation of the n-gram $w_i, w_{i+1}, \ldots, w_{i+n-1}$. That is, ϕ first converts the n-gram of words into an n-gram of word vectors, and then uses multiple feed-forward layers to generate the representation of the n-gram. Similarly, the representation

of a query q is defined as

$$\mathbf{y}_q = \frac{1}{|q| - n + 1} \sum_{i=1}^{|q|-n+1} \phi_{\text{ngram}}(q_i, q_{i+1}, \ldots, q_{i+n-1}),$$

where $q_i, q_{i+1}, \ldots, q_{|q|}$ denotes the word sequence in q and ϕ_{ngram} denotes a high-dimensional and sparse representation of the n-gram $q_i, q_{i+1}, \ldots, q_{i+n-1}$. The final matching score is defined as the dot product of the two representations:

$$f(q, d) = \langle \mathbf{y}_q, \mathbf{y}_d \rangle.$$

The model parameters in SNRM are trained with weak supervision using traditional IR models. Given a query q and a pair of documents d_1 and d_2, the preference label $z \in \{-1, 1\}$ indicates which document is more relevant to the query. In weak supervision, z is defined by the traditional IR model of Query Likelihood:

$$z = \text{sign}(p_{QL}(q, d_1) - p_{QL}(q, d_2)),$$

where p_{QL} denotes the query probability with the Dirichlet prior, and "sign" extracts the sign of a real number. Therefore, given a training instance (q, d_1, d_2, z), SNRM trains its parameters by minimizing the following loss function

$$\min \mathcal{L}(q, d_1, d_2, z) + \lambda \mathcal{L}_1([\mathbf{y}_q, \mathbf{y}_{d_1}, \mathbf{y}_{d_2}]),$$

where $\mathcal{L}(q, d_1, d_2, z) = \max\{0, \epsilon - z[f(q, d_1) - f(q, d_2)]\}$ is the pairwise hinge loss with margin, $\mathcal{L}_1$ is the ℓ_1 regularization over the concatenation of the representations $\mathbf{y}_q, \mathbf{y}_{d_1}$ and $\mathbf{y}_{d_2}$, and the hyper-parameter $\lambda > 0$ controls the sparsity of the learned representations.

4.1.6 Representing Multi-Modal Queries and Documents

In cross-modal search users conduct search across multiple modalities (e.g., the query is text and the documents are images). If the queries and documents are represented in different modalities, then there exists a significant gap between them. Thus, the key is to create common (modality agnostic) representations for queries and documents. Deep learning can indeed fulfill the need, and models are proposed for the purpose.

Deep CCA

One popular approach to multi-modal matching is to learn a latent embedding space where multimedia objects (e.g., images and texts) are uniformly represented. Canonical Correlation Analysis (CCA) (Hardoon *et al.*, 2004) is such a method that finds linear projections that maximize the correlation between the projected vectors of objects from the two original spaces.

To enhance the representation ability, Andrew *et al.* (2013) and Yan and Mikolajczyk (2015) propose to extend CCA into a deep learning framework. Deep CCA directly learns nonlinear mappings for the task of image-text matching. Specifically, it computes representations of objects in the two spaces (e.g., the text query and the image document) by passing them through multiple stacked layers of nonlinear transformations, as shown in Figure 4.9.

Deep CCA represents the text (e.g., query q) as a vector of terms. Each element of the vector is the tf-idf value of the corresponding term. The vector is input into the text network that consists of n stacked triplets of fully connected (FC) layer, ReLU layer, and dropout layer. Deep CCA represents the image (e.g., image document d) as a raw image vector. The vector is input into the image network that consists of m stacked doubles of convolutional layer and ReLU layer, and a last fully connected layer.

The goal of learning is to jointly estimate the parameters in the text network and the image network so that the deep nonlinear mappings of the two types of data are maximally correlated. Assume that (Q, D) denotes the vectors of a text query and a relevant image document,

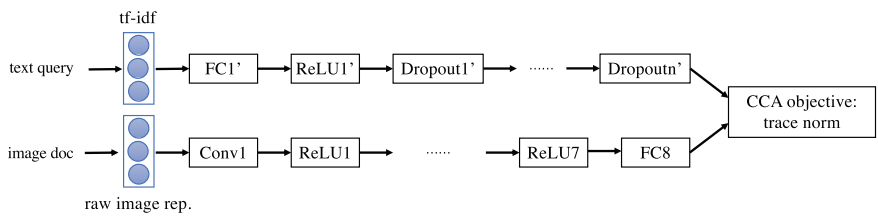

Figure 4.9: Deep CCA architecture. It consists of two deep networks for text and image.

respectively. Further, assume that Θ_1 and Θ_2 are the parameters of the text network and image network, respectively. Thus, deep CCA amounts to maximize the following objective function:

$$\max_{\Theta_1,\Theta_2} \text{corr}(\text{TextNN}(Q;\Theta_1), \text{ImageNN}(D;\Theta_2)),$$

where "corr" is the correlation of two vectors, "TextNN" and "ImageNN" are the text network and image network, respectively.

Adversarial Cross Modal Retrieval

Adversarial learning can be employed to construct a common space in which items in different modalities are represented and compared, as shown in Wang *et al.* (2017a). The method, called Adversarial Cross Modal Retrieval (ACMR), conducts a minimax game involving two players: a feature projector and a modality classifier. The feature projector is responsible for generating modality-invariant representations for items from different modalities in the common space. It has the objective of confusing the modality classifier as an adversary. The modality classifier is responsible for distinguishing items from their modalities. By bringing in the modality classifier, it is expected that the learning of feature projector can be performed more effectively, in the sense that modality invariance is obtained. Figure 4.10 illustrates the flowchart of ACMR.

Specifically, the text branch of ACMR takes bag-of-words features as input. A deep neural network, denoted as $f_T(\cdot;\theta_T)$, is used to conduct text feature projection. The image branch of ACMR takes CNN image features as input. A deep neural network, denoted as $f_V(\cdot;\theta_V)$, is used to conduct image feature projection. θ_V and θ_T are the parameters in the two networks.

Given a set of N training triples $\mathcal{D} = \{(v_i, t_i, y_i)\}_{i=1}^N$, where v_i is an image feature vector, t_i is a text feature vector, and y_i is the category of v_i and t_i, ACMR defines its modality classifier and feature projector as follows.

The modality classifier D is a feed forward neural network with parameters θ_D that predicts the probability of modality given an instance (image or text). The projected features of an image are assigned one-hot vector $[0, 1]$, while the projected features of a text are assigned one-hot

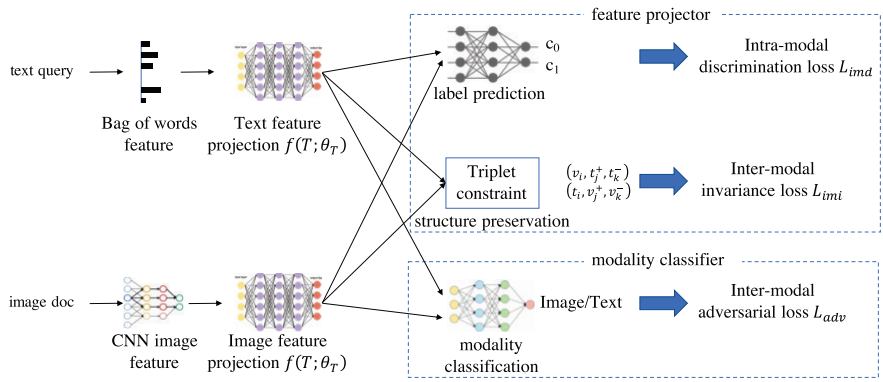

Figure 4.10: Flowchart of ACMR. The modality classifier tries to distinguish the items from their modalities. The features projector manages to confuse the modality classifier through generating modality-invariant and discriminative representations.

vector $[1, 0]$. The modality classifier acts as an adversary. It manages to minimize the adversarial loss:

$$L_{adv}(\Theta_D) = -\frac{1}{N}\sum_{i=1}^{N}(\mathbf{m}_i \cdot (\log D(f_V(v_i);\theta_D) + \log(\mathbf{1} - D(f_T(t_i);\theta_D)))),$$

where $\mathbf{m}_i$ is the modality label of the i-th instance, expressed as one-hot vector.

The feature projector conducts modality-invariant embedding of texts and images into a common space, consisting of two parts: label prediction and structure preservation. The label prediction minimizes a loss function L_{imd} to ensure that the feature representations belonging to the same categories are sufficiently close. The structure preservation minimizes a loss function L_{imi} to ensure that the feature representations belonging to the same categories are sufficiently close across modalities, and the feature representations belonging to different categories are sufficiently far apart within a modality. The overall generation loss, denoted as L_{emb}, is a combination of label prediction loss L_{imd}, structure preservation loss L_{imi}, and a regularization term L_{reg}:

$$L_{emb}(\theta_V, \theta_T, \theta_{imd}) = \alpha \cdot L_{imd} + \beta \cdot L_{imi} + L_{reg},$$

where $\alpha > 0$ and $\beta > 0$ are trading-off coefficients.

Finally, the learning of the ACMR model is conducted by jointly minimizing the adversarial and the generation losses, as a minimax game:

$$(\hat{\theta}_V, \hat{\theta}_T, \hat{\theta}_{imd}) = \underset{\theta_V, \theta_T, \theta_{imd}}{\operatorname{argmin}} (L_{emb}(\theta_V, \theta_T, \theta_{imd}) - L_{adv}(\hat{\theta}_D)),$$
$$\hat{\theta}_D = \underset{\theta_D}{\operatorname{argmax}}(L_{emb}(\hat{\theta}_V, \hat{\theta}_T, \hat{\theta}_{imd}) - L_{adv}(\theta_D)).$$

4.1.7 Experimental Results

We present the experimental results of search relevance by the methods of representation learning, reported in Yin and Schütze (2015) and Pang *et al.* (2017b). In the experiment, the benchmark data of MSRP[1] is utilized, and Accuracy and F_1 are adopted as evaluation measures. The results in Table 4.2 indicate that the methods based on representation learning can outperform the baseline of TF-IDF in terms of F_1.

We also present the experimental results of the multi-modal search in Table 4.3. The experiments are based on the Wikipedia dataset and reported in Wang *et al.* (2017a). In the experiment, Mean Average Precision (MAP) is used as the evaluation measure. The results indicate that the multi-modal matching method of ACMR can significantly outperform the baseline, especially when deep features are used.

Table 4.2: Performances of representation learning methods on MSRP dataset.

	Accuracy	$\boldsymbol{F_1}$
TF-IDF (baseline)	0.7031	0.7762
DSSM	0.7009	0.8096
CLSM	0.6980	0.8042
ARC-I	0.6960	0.8027

[1] https://www.microsoft.com/en-us/download/details.aspx?id=52398.

Table 4.3: Performances of multi-modal matching method on Wikipedia dataset in terms of MAP

	Image to Text	Text to Image	Average
CCA (shallow feature)	0.255	0.185	0.220
CCA (deep feature)	0.267	0.222	0.245
ACMR (shallow feature)	0.366	0.277	0.322
ACMR (deep feature)	0.619	0.489	0.546

4.2 Matching Based on Matching Function Learning

4.2.1 General Framework

The matching degree between query and document can be determined by aggregating the local and global matching signals between the query and document. The matching signals, as well as their locations, can be captured from the input query and document representations.

Researchers propose to use deep neural networks to automatically learn the matching patterns between query and document, referred to here as matching function learning. There are two critical problems in this approach: (1) how to represent and calculate the matching signals, and (2) how to aggregate the matching signals to calculate the final matching score.

Figure 4.11 shows the general framework. In the framework, the query and document are compared with each other to create matching signals, and the matching signals are aggregated to output the matching score, all in a single neural network.

One approach is first to let the query and document interact based on their raw representations, yielding a number of local matching signals, and then to aggregate the local matching signals to output the final matching score. Another approach is to create the representations of query and document as well as their interactions at both local and global levels with a single neural network usually using attention.

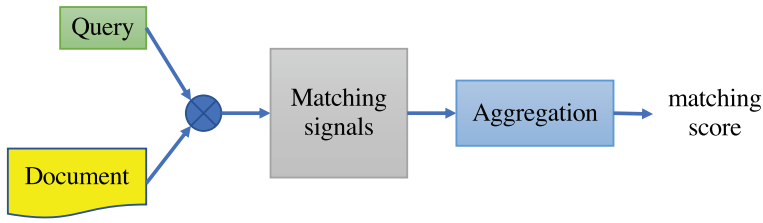

Figure 4.11: Query-document matching based on matching function learning.

4.2.2 Learning Matching Function with Matching Matrix

Matching matrix is used for storing word-level matching signals and their positions. The columns and rows of the matching matrix correspond to the words in the query and document, respectively. Each entry represents the location of matching, and the value of each entry represents the degree of matching. The matching matrix is input to a neural network as a whole.

Advantages of learning with the matching matrix are as follows:

(1) The matching matrix is precise in the sense that the local matching information (degree and location) is accurately represented in it.

(2) The matching matrix is intuitive in the sense that the local matching information can be easily visualized and interpreted.

Convolutional Matching Model (ARC-II)

The convolutional matching model (Arc-II) (Hu *et al.*, 2014) is directly built on the interactions between query and document. The idea is first to let the query and document interact with their raw representations and then capture the matching signals from the interactions.

As shown in Figure 4.12, in the first layer, ARC-II takes a sliding window on the query and document, and models the interactions within the window at each location using one-dimensional convolution. For segment i of query q and segment j of document d, ARC-II constructs the interaction representation:

$$\mathbf{z}_{i,j}^0 = [\mathbf{q}_{i:i+k_1-1}^T, \mathbf{d}_{j:j+k_1-1}^T]^T,$$

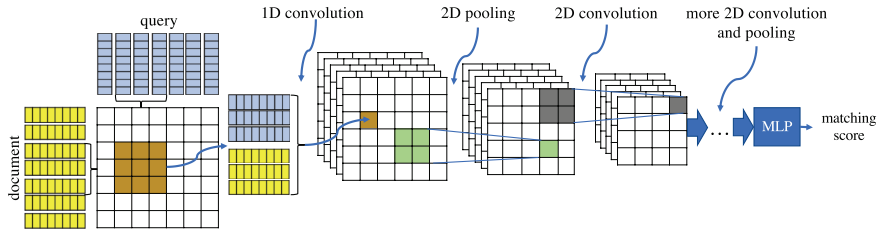

Figure 4.12: Convolution matching model (ARC-II).

where k_1 is the width of sliding window, and $\mathbf{q}_{i:i+k_1-1}^T = [\mathbf{q}_i^T, \mathbf{q}_{i+1}^T, \ldots, \mathbf{q}_{i+k_1-1}^T]^T$ (and $\mathbf{d}_{j:j+k_1-1}^T = [\mathbf{d}_j^T, \mathbf{d}_{j+1}^T, \ldots, \mathbf{d}_{j+k_1-1}^T]^T$) is the concatenation of embedding vectors of k_1 words in the query segment (and in the document segment). Thus, the corresponding value in the feature map f is

$$z_{i,j}^{(1,f)} = g(\mathbf{z}_{i,j}^0) \cdot \sigma(\mathbf{w}^{(1,f)} \mathbf{z}_{i,j}^0 + b^{(1,f)}),$$

where σ is the activation function, $\mathbf{w}^{(1,f)}$ and $b^{(1,f)}$ are the convolutional parameters, and $g(\cdot)$ is the gating function such that $g(\cdot) = 0$ if all the elements in the input vectors equal 0, otherwise $g(\cdot) = 1$. Here, $g(\cdot)$ works as a zero padding function. For all possible query words and document words, the one-dimensional convolutional layer outputs a two-dimensional matching matrix.

The next layer conducts two-dimensional max-pooling in every non-overlapping 2×2-window. The (i, j)-th entry in the output matrix is

$$z_{i,j}^{(2,f)} = \max(z_{2i-1,2j-1}^{(1,f)}, z_{2i-1,2j}^{(1,f)}, z_{2i,2j-1}^{(1,f)}, z_{2i,2j}^{(1,f)}).$$

The max-pooling layer quickly shrinks the size of matching matrix by filtering out weak (possibly noisy) matching signals.

Then, two-dimensional convolution is applied to the output matrix of the max-pooling layer. That is, interaction representations are created within the sliding window of size $k_3 \times k_3$ at each location on the matrix using two-dimensional convolution. The (i, j)-th value in the feature map f is

$$z_{i,j}^{(3,f)} = g(\mathbf{Z}_{i,j}^{(2)}) \cdot \sigma(\mathbf{W}^{(3,f)} \mathbf{Z}_{i,j}^{(2)} + b^{(3,f)}),$$

where $\mathbf{W}^{(3,f)}$ and $b^{(3,f)}$ are the convolutional parameters and $\mathbf{Z}_{i,j}^{(2)}$ is the input matrix. More layers of two-dimensional max-pooling and convolutional layers could be added afterward.

In the last layer, an MLP is utilized to summarize the matching signals and output the matching score

$$f(q,d) = \mathbf{W}_2\sigma\left(\mathbf{W}_1\mathbf{Z}^{(3)} + \mathbf{b}_1\right) + b_2,$$

where $\mathbf{Z}^{(3)}$ is the last layer feature map.

To train the model parameters, ARC-II makes use of the same discriminative strategy as in ARC-I. That is, given query q, relevant query-document pair (q,d) and irrelevant query-document pair (q,d'). ARC-II minimizes the objective:

$$\mathcal{L} = \sum_{(q,d)\in\mathcal{C}} \sum_{(q,d')\in\mathcal{C}'} \left[1 - f(q,d) + f(q,d')\right]_+,$$

where $\mathcal{C}$ and $\mathcal{C}'$ contain the relevant and irrelevant query-document pairs, respectively.

MatchPyramid

The convolutional matching model ARC-II makes early interactions between query and document. However, the meanings of the interactions (i.e., the one-dimensional convolution) are not clear. Pang *et al.* (2016b) point out that the matching matrix can be constructed more straightforwardly. The proposed model, called MatchPyramid, redefines the matching matrix as a word-level similarity matrix. Then, a two-dimensional convolutional neural network is exploited to extract query-document matching patterns, summarize the matching signals, and calculate the final matching score. The main idea of MatchPyramid is to view text matching as image recognition, by taking the matching matrix as an image, as shown in Figure 4.13. The input to the convolutional neural network is matching matrix $\mathbf{M}$, where element M_{ij} represents the basic interaction of the i-th query word q_i and the j-th document word d_j. In general, M_{ij} stands for the similarity between q_i and d_j, which can have different definitions. The indicate function $M_{ij} = 1_{q_i=d_j}$ can be used to produce either 1 or 0 to indicate whether

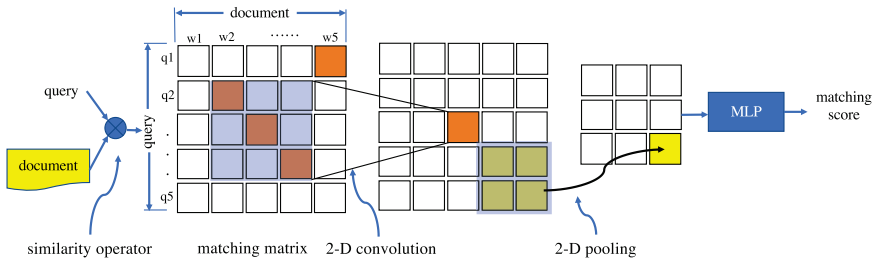

Figure 4.13: Architecture of MatchPyramid.

the two words are identical. The embeddings of query word and document word can also be used to represent semantic similarities between the two words. For example, cosine similarity: $M_{ij} = \cos(\mathbf{q}_i, \mathbf{d}_j)$ where $\mathbf{q}_i$ and $\mathbf{d}_j$ are the embeddings of q_i and d_j respectively, and dot product: $M_{ij} = \mathbf{q}_i^T \mathbf{d}_j$.

MatchPyramid is a two-dimensional convolutional neural network with the matching matrix $\mathbf{M}$ as input. Let $\mathbf{z}^{(0)} = \mathbf{M}$. The k-th kernel $\mathbf{w}^{(1,k)}$ scans the matching matrix $\mathbf{z}^{(0)}$ and generates a feature map $\mathbf{z}^{(1,k)}$ whose values are

$$z_{ij}^{(1,k)} = \sigma \left(\sum_{s=0}^{r_k-1} \sum_{t=0}^{r_k-1} w_{s,t}^{(1,k)} z_{i+s,j+t}^{(0)} + b^{(1,k)} \right),$$

where r_k is the size of the k-th kernel. Dynamic pooling is then utilized to deal with the variability in text length. The fixed-size feature maps outputted by dynamic pooling is:

$$z_{ij}^{(2,k)} = \max_{0 \le s \le d_k} \max_{0 \le t \le d'_k} z_{i \cdot d_k + s, j \cdot d'_k + t}^{(1,k)},$$

where d_k and d'_k are the width and length of the pooling kernel. With dynamic pooling the output feature map becomes fixed-sized. More layers of convolution and dynamic pooling can be stacked.

In the last layer, MatchPyramid utilizes an MLP to produce the final matching score:

$$[s_0, s_1]^T = f(q, d) = \mathbf{W}_2 \sigma \left(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1 \right) + \mathbf{b}_2,$$

where $\mathbf{z}$ is the input feature map and $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$ and $\mathbf{b}_2$ are the parameters.

To learn the model parameters, MatchPyramid utilizes the softmax function and cross entropy loss. Given a set of N training triples $\mathcal{D} = \{(q_n, d_n, r_n)\}_{n=1}^{N}$ where $r_n \in \{0, 1\}$ is the ground truth relevance label, 1 for relevant and 0 for irrelevant. Cross entropy loss is defined as:

$$\mathcal{L} = -\sum_{(q,d,r)\in\mathcal{D}} [r \log(p(\text{rel} \mid q, d)) + (1 - r) \log(1 - p(\text{rel} \mid q, d))],$$

where $p(\text{rel} \mid q, d) = \frac{e^{s_1}}{e^{s_0}+e^{s_1}}$ is the probability that document d is relevant to query q, given by the softmax function.

One attractive characteristic of the two-dimensional convolution is that it is capable to automatically extract high level (soft) matching patterns and store them in the kernels, which is similar to visual pattern extraction in image recognition. Figure 4.14 illustrates an example, with a handcrafted matching matrix based on the indicator function. Given two kernels, it is clear that the first convolutional layer can capture both n-gram matching signals such as "down the ages" and n-term matching signals such as "(noodles and dumplings) vs. (dumplings and noodles)", as shown in the feature maps of the first layer. Then, the second convolution layer makes compositions and forms higher level matching patterns, as shown in the feature map of the second layer.

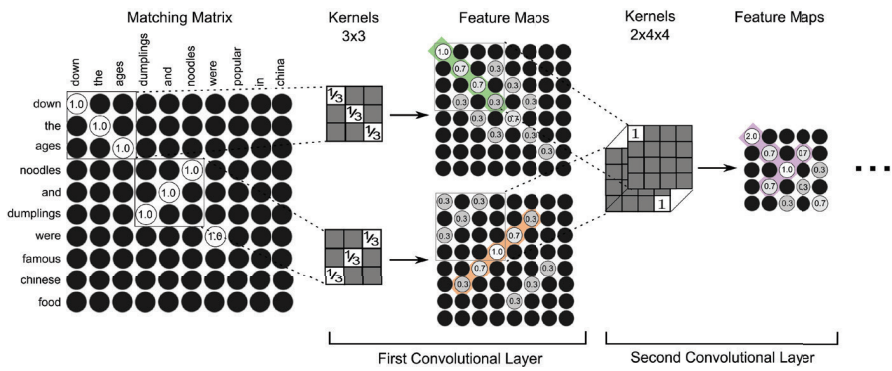

Figure 4.14: The effects of the 2-dimensional convolution kernel. Figure from Pang *et al.* (2016b).

Match-SRNN

Two-dimensional Recurrent Neural Networks (RNN) can also be used to discover matching patterns in the matching matrix. Wan *et al.* (2016b) propose a method to divide the matching of two texts into a series of sub-problems of matching, and solve the sub-problems recursively. The proposed model, called MatchSRNN, applies a two-dimensional RNN (Graves *et al.*, 2007) to scan the matching matrix from the top-left to the bottom-right. The state at the last (bottom-right) position is considered as the overall representation of matching.

As shown in Figure 4.15, Match-SRNN consists of three components: an NTN for discovering word-level matching signals, a spatial RNN for summarizing the sentence-level matching representation, and a linear layer for calculating the final matching score.

First, given query q and document d, an NTN is utilized to calculate the similarity between i-th query word q_i and the j-th document word d_j:

$$s(q_i, d_j) = \mathbf{u}^T \sigma \left(\mathbf{q}_i^T \mathbf{M}^{[1:r]} \mathbf{d}_j + \mathbf{V} \begin{bmatrix} \mathbf{q}_i \\ \mathbf{d}_j \end{bmatrix} + \mathbf{b} \right),$$

where $\mathbf{q}_i$ and $\mathbf{d}_j$ are the embeddings of the i-th query word and the j-th document word.

Next, a spatial RNN (two-dimensional RNN) is employed to scan the outputted matching matrix recursively. Specifically, to calculate the matching representation between the query prefix $\mathbf{q}_{[1:i]}$ and document

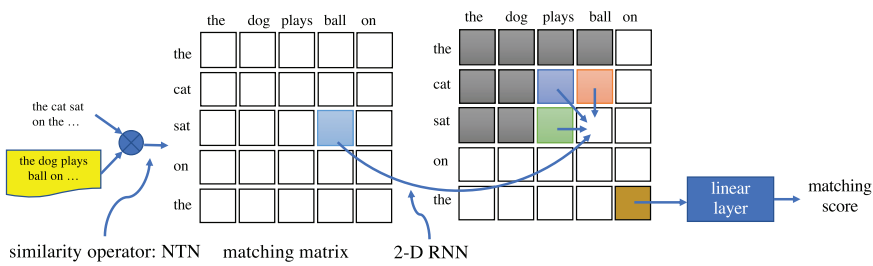

Figure 4.15: Match-SRNN model.

prefix $\mathbf{q}_{[1:j]}$, the representations of their prefixes are first calculated:

$$\begin{aligned}\mathbf{h}_{i-1,j} &= \text{SpatialRNN}(\mathbf{q}_{[1:i-1]}, \mathbf{d}_{[1:j]}),\\ \mathbf{h}_{i-1,j-1} &= \text{SpatialRNN}(\mathbf{q}_{[1:i-1]}, \mathbf{d}_{[1:j-1]}),\\ \mathbf{h}_{i,j-1} &= \text{SpatialRNN}(\mathbf{q}_{[1:i]}, \mathbf{d}_{[1:j-1]}),\end{aligned}$$

where $\text{SpatialRNN}(\cdot,\cdot)$ is the two-dimensional RNN applied to the prefixes. Then, the matching representation is calculated as

$$\mathbf{h}_{i,j} = \text{SpatialRNN}(\mathbf{q}_{[1:\ i]}, \mathbf{d}_{[1:\ j]}) = f(\mathbf{h}_{i-1,j}, \mathbf{h}_{i,j-1}, \mathbf{h}_{i-1,j-1}, s_{q_i,d_i}),$$

where f represents the model of two-dimensional RNN. Instead of two-dimensional RNN more powerful models such as two-dimensional GRU and LSTM can also be exploited.

The last representation at the right bottom corner, $\mathbf{h}_{|q|,|d|}$, reflects the global matching representation between the query and document. Finally, a linear function is used to calculate the final matching score:

$$f(q,d) = \mathbf{w}\mathbf{h}_{|q|,|d|} + b,$$

where $\mathbf{w}$ and b are parameters.

To learn the model parameters, Match-SRNN utilizes the pairwise hinge loss. Given query q, the relevant query-document pair (q, d^+) in the training data should receive a higher score than the irrelevant query-document pair (q, d^-), defined as:

$$\ell(q, d^+, d^-) = \max(0, 1 - f(q, d^+) + f(q, d^-)).$$

Given the training data, all the parameters in the Match-SRNN model are trained by BackPropagation.

4.2.3 Learning Matching Function with Attention

A recent trend is to leverage attention, which is inspired by the attention mechanism in human cognition. Attention is successfully applied to tasks in NLP and IR, including query-document matching.

Decomposable Attention Model

Parikh *et al.* (2016) point out that matching signals can be captured and represented with a decomposable attention mechanism. As shown in Figure 4.16, the model consists of three steps: attend, compare, and aggregate. Given a query and a document, where each word in the query and the document is represented by an embedding vector, the model first creates a soft alignment matrix using attention; then it uses the (soft) alignment to decompose the task into subproblems; finally, it merges the results of the subproblems to produce the final matching score.

Specifically, given a query-document pair (q, d) where each word in q is represented as an embedding vector $q = (\mathbf{q}_1, \ldots, \mathbf{q}_{|q|})$ and $|q|$ is the number of words in q, and each word in d is represented as an embedding vector $d = (\mathbf{d}_1, \ldots, \mathbf{d}_{|d|})$ and $|d|$ is the number of words in d. In the attend step an attention matrix between each query word and document word is constructed. The unnormalized attention weight e_{ij} is calculated with a decomposable function:

$$e_{ij} = F'(\mathbf{q}_i, \mathbf{d}_j) = F(\mathbf{q}_i)^T F(\mathbf{d}_j),$$

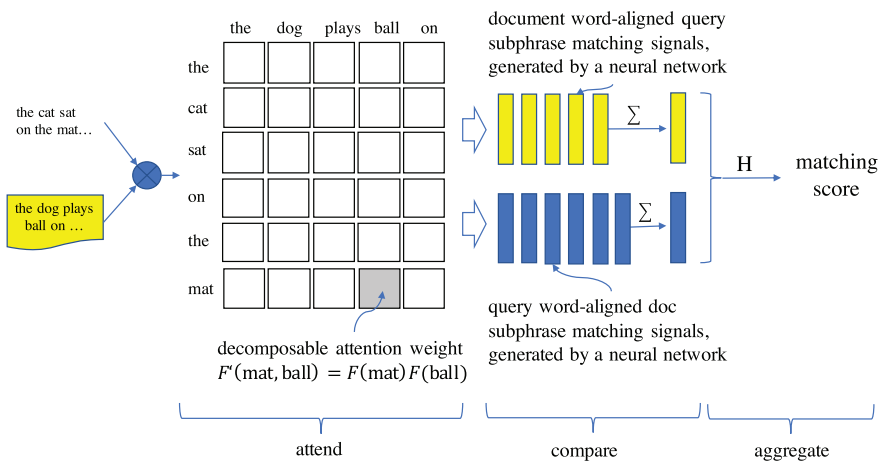

Figure 4.16: Decomposable attention model for matching.

where F is a feed-forward neural network. With the attention weights, the whole document aligned to the i-th query word is

$$\beta_i = \sum_{j=1}^{|d|} \frac{\exp(e_{ij})}{\sum_{k=1}^{|d|} \exp(e_{ik})} \mathbf{d}_j.$$

Similarly, the whole query aligned to the j-th document word is

$$\alpha_j = \sum_{i=1}^{|q|} \frac{\exp(e_{ij})}{\sum_{k=1}^{|q|} \exp(e_{kj})} \mathbf{q}_i.$$

In the compare step, each query word and its aligned version $\{(\mathbf{q}_i, \beta_i)\}_{i=1}^{|q|}$ are compared separately with a feed-forward network G:

$$\mathbf{v}_{1,i} = G([\mathbf{q}_i^T, \beta_i^T]^T), \qquad \forall i = 1, \ldots, |q|,$$

where $[\cdot, \cdot]$ concatenates two vectors. Each document word and its aligned version $\{(\mathbf{d}_j, \alpha_j)\}_{j=1}^{|d|}$ are compared separately with the same feed-forward network G:

$$\mathbf{v}_{2,j} = G([\mathbf{d}_j^T, \alpha_j^T]^T), \qquad \forall j = 1, \ldots, |d|.$$

Finally in the aggregate step, the two sets of comparison signals $\{\mathbf{v}_{1,i}\}$ and $\{\mathbf{v}_{2,j}\}$ are summed separately:

$$\mathbf{v}_1 = \sum_{i=1}^{|q|} \mathbf{v}_{1,i}, \qquad \mathbf{v}_2 = \sum_{j=1}^{|d|} \mathbf{v}_{1,j}.$$

The two aggregated vectors are then input to a feed forward network followed by a linear layer H, giving multiple-class scores:

$$\hat{\mathbf{y}} = H([\mathbf{v}_1^T, \mathbf{v}_2^T]^T).$$

The predicted class (e.g., relevant or irrelevant) is decided by $\hat{y} = \arg\max_i \mathbf{y}_i$.

In training of the model, cross-entropy loss is utilized:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} y_c^{(n)} \log \frac{\exp(\hat{y}_c^{(n)})}{\sum_{c'=1}^{C} \exp(\hat{y}_{c'}^{(n)})},$$

where C is the number of classes,[2] N is the number of instances in the training data.

[2]There are two classes for the query-document matching task: relevant and irrelevant.

Matching with BERT

Recently, BERT (the Bidirectional Encoder Representations from Transformers) becomes the state-of-the-art model for language understanding tasks with its better performances (Devlin *et al.*, 2019). In pre-training of BERT the representations of two texts are learned from a large amount of unlabeled data through mask language modeling and next sentence prediction. In fine-tuning the representations are further refined for the downstream task with an output layer added on top of the model and a small amount of task-specific labeled data.

When applied to search, BERT can be utilized to calculate the matching degree between query and document, as long as training data is provided (Nogueira and Cho, 2019). That is, a pre-trained BERT model can be adapted to query-document matching with fine-tuning.

Figure 4.17 shows a popular method of using BERT for query-document matching. Given a query-document pair (q, d), the input

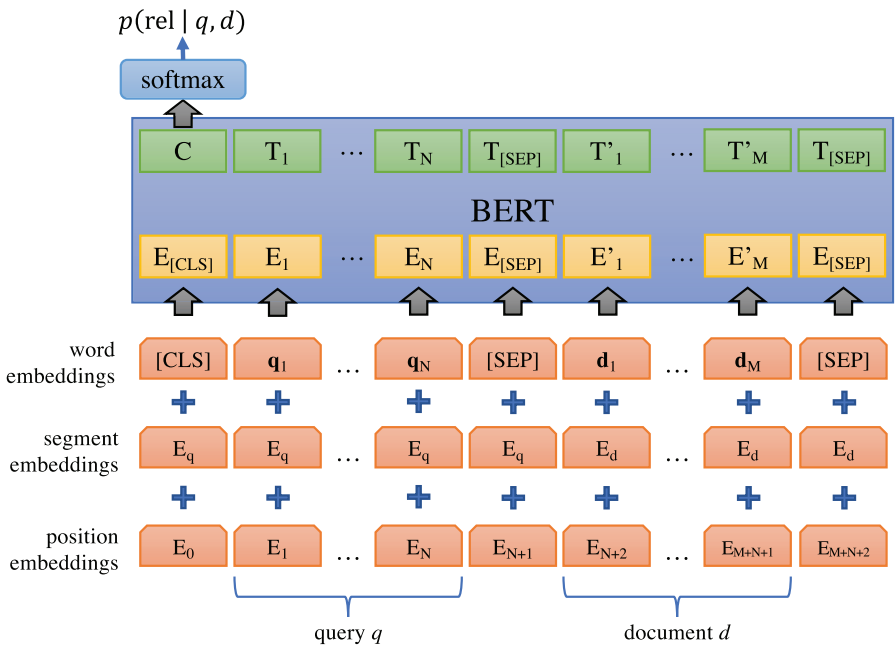

Figure 4.17: Fine-tuning BERT for query-document matching.

to the BERT model includes query words, document words: "[CLS], $\mathbf{q}_1, \ldots, \mathbf{q}_N$, [SEP], $\mathbf{d}_1, \ldots, \mathbf{d}_M$, [SEP]", where "[CLS]" is the token to indicate whether the query-document pair is relevant or not, "[SEP]" is the token to indicate the separation of query and document, and $\mathbf{q}_i$ and $\mathbf{d}_j$ are the i-th query word and the j-th document word, respectively. The query (and document) is padded or truncated to have N (and M) words. Each word is represented with its embedding. The input embedding of a word is the sum of the corresponding word embedding, the segment embedding, and the position embedding.

The model of BERT is the encoder of Transformer (Vaswani *et al.*, 2017), which outputs a sequence of high level semantic representations for the special input tokens and query and document words: "$C, T_1, \ldots, T_N, T_{[\text{SEP}]}, T'_1, \ldots, T'_M, T'_{[\text{SEP}]}$", where C is the representation of the token [CLS], $T_1, \ldots, T_N$ of the query words, $T'_1, \ldots, T'_M$ of the document words, $T_{[\text{SEP}]}$ and $T'_{[\text{SEP}]}$ of the two separators. The representation of the [CLS] token C is fed into an output layer (e.g., single layer neural network) to obtain $p(\text{rel} \mid q, d)$, which represents the probability of document's being relevant to query.

The $\text{BERT}_{\text{LARGE}}$ model released by Google is widely used as pre-trained model.[3] It is then employed in the fine-tuning for query document matching. Suppose that we are given a set of training triples $\mathcal{D} = \{(q_n, d_n, r_n)\}_{n=1}^{N}$ where $r_n \in \{0, 1\}$ is the ground truth label. The cross-entropy loss is calculated:

$$\mathcal{L} = \sum_{(q,d,r)\in\mathcal{D}} -r \log(p(\text{rel} \mid q, d)) - (1 - r) \log(1 - p(\text{rel} \mid q, d)).$$

Compared to the existing models, BERT offers several advantages for query-document matching. First, in BERT, the query and document are jointly input to the model, making it possible to simultaneously represent the intra-query, intra-document, and inter query-document interactions. Second, in BERT, the representations of query and document as well as query-document interaction are transformed multiple times in the hierarchical architecture. As a result, complicated local and global matching patterns can be represented. Third, BERT uses a pre-training/fine-tuning framework, where a pre-trained BERT model

[3] https://github.com/google-research/bert.

can leverage the information in a large amount of unlabeled data. Other matching models, however, have less powerful representation abilities and thus cannot achieve similar performances. Studies show that pre-training of BERT can make the model favor text pairs that are semantically similar and thus can make the model perform very well in matching (Nogueira and Cho, 2019; Nogueira *et al.*, 2019; Qiao *et al.*, 2019).

4.2.4 Learning Matching Functions in Search

There exist differences between the matching tasks in search and those in NLP. The matching tasks in search are mainly about topic relevance, while the matching tasks in NLP are mainly concerned with semantics. For example, matching models in search should be able to handle exact matching signals very well, such as query term importance and diversity of matching (Guo *et al.*, 2016). Several matching models tailored for search are developed and proved to be effective.

Deep Relevance Matching Model (DRMM)

In Guo *et al.* (2016), a relevance matching model called DRMM is proposed. Figure 4.18 shows the model architecture. The query q and the document d are represented as two sets of word vectors respectively: $q = \{q_1, q_2, \ldots, q_{|q|}\}$ and $d = \{w_1, w_2, \ldots, w_{|d|}\}$, where q_i and w_j denote a query word vector and a document word vector both generated by Word2Vec. For each query word q_i, a matching histogram $\mathbf{z}_i^{(0)}$ is constructed to characterize the interaction between q_i and the whole document:

$$\mathbf{z}_i^{(0)} = h(q_i \bigotimes d),$$

for $i = 1, \ldots, |q|$, where $\bigotimes$ denotes cosine similarity calculation between q_i and all words in d, outputting a set of cosine similarity scores in the interval of $[-1, 1]$. Then, function h discretizes the interval into a set of ordered bins, counts the number of similarity scores in each bin, and calculates the logarithm of the counts, generating the histogram vector $\mathbf{z}_i^{(0)}$.

The vector $\mathbf{z}_i^{(0)}$ is then passed through L-layer feed-forward layers to generate a matching score for each query word q_i, denoted as $z_i^{(L)}$. Given the matching scores of individual query words, the final matching score between q and d, $f(q, d)$, is calculated as a weighted sum of the matching scores:

$$f(q, d) = \sum_{i=1}^{|q|} g_i z_i^{(L)},$$

where the weight of query word q_i, g_i, is generated by a term gating network:

$$g_i = \frac{\exp(\mathbf{w}_g \mathbf{x}_i)}{\sum_{j=1}^{|q|} \exp(\mathbf{w}_g \mathbf{x}_j)},$$

where $\mathbf{w}_g$ is the parameter vector in the term gating network and $\mathbf{x}_j$ $(j = 1, \ldots, |q|)$ is the feature vector to characterize the importance of query word q_i. The features can be term vector or inverse document frequency (IDF).

The parameters in the feed-forward network and term gating network are jointly learned. Given a training example (q, d^+, d^-), where d^+ and d^- respectively denote a relevant document and a non-relevant document, the learning of DRMM amounts to minimizing the pairwise hinge loss with margin:

$$\mathcal{L} = \max(0, 1 - f(q, d^+) + f(q, d^-)).$$

The stochastic gradient descent method of Adagrad with mini-batches is applied to conduct the minimization. Early stopping strategy is adopted for regularization.

Kernel Based Neural Ranking Model (K-NRM)

DRMM is effective in modeling the interactions between query words and document words. However, the histogram pooling part (i.e., counting of similarity values in each bin) is not a differentiable function, which hinders end-to-end learning of the matching model. To cope with the problem, DRMM makes use of pre-trained word vectors to represent the words in the queries and documents. Xiong *et al.* (2017) propose a relevance matching model called K-NRM. In the model, instead of

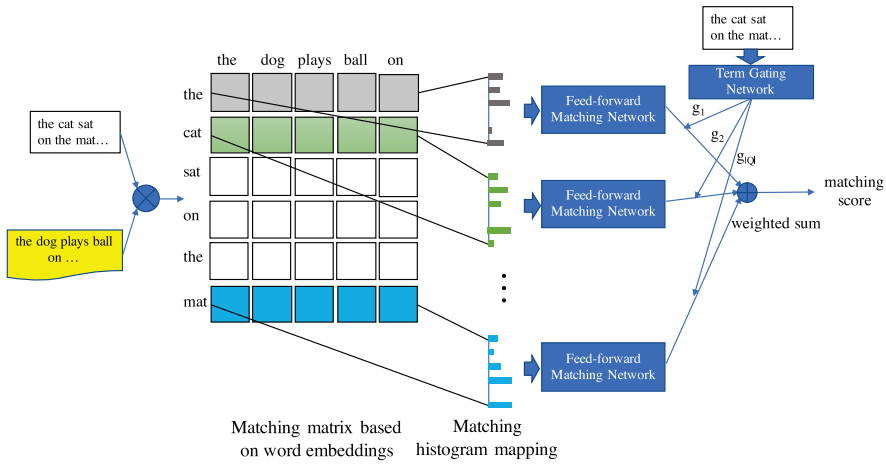

Figure 4.18: Deep relevance matching model.

counting the similarity scores, kernel pooling is employed to characterize the matching degree between each query word and the document. As a result, the model can be trained end-to-end.

As shown in Figure 4.19, given the query q and the document d, K-NRM first uses an embedding layer to map each word (in q and d) into an embedding vector. Then, it constructs an translation matrix (i.e., matching matrix) $\mathbf{M}$ where the (i, j)-th element in M_{ij} is the embedding similarity (cosine similarity) between the i-th query word and the j-th document word. Then, it applies the kennel pooling operator to each row

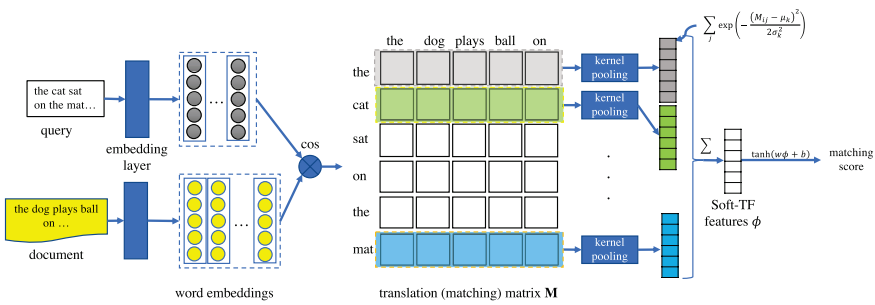

Figure 4.19: Kernel based neural ranking model.

of M (corresponding to each query word), generating a K-dimensional vector $\vec{K}$. Specifically, the k-th dimension of the pooling vector of the i-th query word is defined as an RBF Kernel function:

$$K_k(M_i) = \sum_j \exp\left(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2}\right),$$

where M_i is the i-th row of $\mathbf{M}$, M_{ij} is the j-th element in M_i, and μ_k and σ_k are the mean and variance of the RBF kernel, respectively.

Given the pooling vectors of all query words, the pooling vectors are summed to create the soft-TF features:

$$\phi(M) = \sum_{i=1}^{|q|} \log \vec{K}(M_i),$$

where $\vec{K}(M_i) = [K_1(M_i), \ldots, K_K(M_i)]$ and log is applied to each dimension of $\vec{K}(M_i)$. Finally, the soft-TF features are combined together, yielding a final matching score

$$f(q, d) = \tanh(\langle \mathbf{w}, \phi(M) \rangle + b),$$

where $\mathbf{w}$ and b are weights and bias respectively.

One advantage of K-NRM is that learning can be conducted in an end-to-end manner. Given a set of training examples $D = \{(q_i, d_i^+, d_i^-)\}_{i=1}^N$, where d_i^+ and d_i^- respectively denote a relevant document and a non-relevant document w.r.t. q_i, the learning of K-NRM mounts to minimizing the pairwise hinge loss function:

$$\mathcal{L}(\mathbf{w}, b, \mathcal{V}) = \sum_{i=1}^{N} \max(0, 1 - f(q_i, d_i^+) + f(q_i, d_i^-)).$$

Back propagation can be employed in learning of the kernels, which makes it possible to learn both the parameters $\mathbf{w}, b$ and word embeddings $\mathcal{V}$ during training.

Duet

Matching based on representation learning relies on the distributed representations of queries and documents. In contrast, matching based on

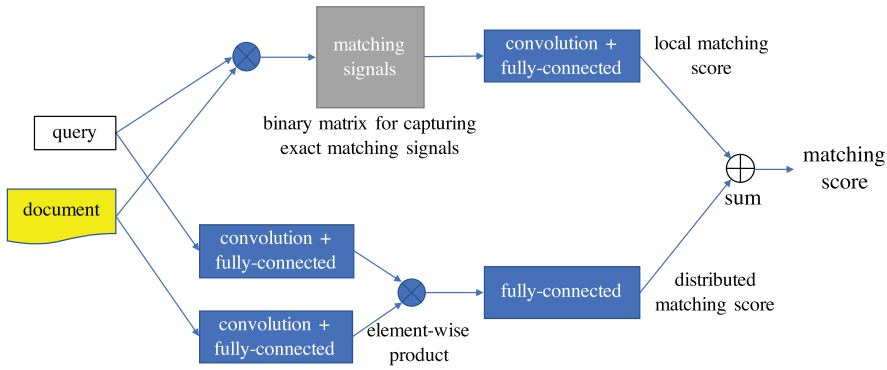

Figure 4.20: Model architecture of duet.

matching function learning relies on the local matching representations of queries and documents. In the matching model called Duet (Mitra *et al.*, 2017), a hybrid approach is taken and the advantages of the two approaches are both leveraged.

As shown in Figure 4.20, Duet consists of two separate deep neural networks, one that matches the query and the document using local representations, and the other that matches the query and the document using distributed representations. Given the query q and the document d, the final Duet matching score $f(q, d)$ is defined as

$$f(q, d) = f_l(q, d) + f_d(q, d),$$

where $f_l(q, d)$ and $f_d(q, d)$ respectively denote the local and distributed matching scores.

In the model of $f_l(q, d)$, each query word (and each document word) is represented by a one-hot vector. The model then creates a binary matching matrix $\mathbf{X} \in \{0, 1\}^{|d| \times |q|}$ where the (i, j)-th entry represents the exact matching relation of the i-th document word and the j-th query word. The matching matrix $\mathbf{X}$ is first passed through a convolutional layer. The output is then passed through two fully-connected layers, a dropout layer, and a fully-connected layer to produce a single matching score.

In the model of $f_d(q, d)$, (similar to DSSM Huang *et al.*, 2013), the words in q and d are respectively represented as frequency vectors

of letter n-grams. Then, the vector of query q is passed through a convolution layer, a max-pooling layer, and a fully-connected layer, yielding a query representation of vector $\mathbf{Q}$. Similarly, the vector of document d is passed through a convolution layer, a max-pooling layer, and a fully-connected layer, yielding a document representation of matrix $\mathbf{D}$. Next, element-wise product is calculated between $\mathbf{D}$ and an extended $\mathbf{Q}$. The resulting matrix is passed through fully connected layers, and a dropout layer to produce a single matching score.

The two networks in Duet are jointly trained as a single neural network. Given a training example which consists of the query q, the relevant document d^+, and the set of non-relevant documents $\mathcal{D} = \{d_1^-, \ldots, d_k^-\}$, the objective of learning is defined as the conditional probability of document d^+ given query q:

$$P(d^+ \mid q) = \frac{\exp(f(q, d^+))}{\sum_{d' \in \mathcal{D}} \exp f(q, d')}.$$

Stochastic gradient descent is utilized to maximize the log likelihood $\log P(d^+ \mid q)$.

4.2.5 Experimental Results

We present the experimental results by matching function learning methods, reported in Wan *et al.* (2016b). In the experiment, the benchmark of Yahoo! Answers is utilized, and P@1 and Mean Reciprocal Rank (MRR) are adopted as evaluation measures. The results in Table 4.4 indicate that both the approach of representation learning and the approach of matching function learning can outperform the baseline of BM25. The matching function learning methods, in general, perform better than the representation learning methods. Table 4.5 presents some experimental results of the matching methods on ad hoc retrieval. The results are based on the experiments conducted in Dai *et al.* (2018) and Mitra *et al.* (2017). We also represent the experimental results of BERT reported in Nogueira and Cho (2019), for the task of passage ranking. In the experiment, the benchmark of MS MARCO is utilized, and MRR@10 is adopted as the evaluation measure. The results in Table 4.6 indicate that fine-tuned $\text{BERT}_{\text{LARGE}}$ significantly outperforms the state-of-the-art passage ranking models.

Table 4.4: Performances of some representation learning methods and matching function learning methods on Yahoo! answers

		P@1	MRR
	BM25 (baseline)	0.579	0.726
Representation learning	ARC-I	0.581	0.756
	CNTN	0.626	0.781
	LSTM-RNN	0.690	0.822
Matching function learning	ARC-II	0.591	0.765
	MatchPyramid	0.764	0.867
	Match-SRNN	0.790	0.882

Table 4.5: Performances of matching function learning methods on ad hoc retrieval, based on bing search log and Sogou log

	Bing Search Log		Sogou Log	
	NDCG@1	NDCG@10	NDCG@1	NDCG@10
DSSM	0.258	0.482	–	–
Duet	0.322	0.530	–	–
DRMM	0.243	0.452	0.137	0.315
MatchPyramid	–	–	0.218	0.379
K-NRM	–	–	0.264	0.428

Table 4.6: Performances of the fine-tuned $\text{BERT}_{\text{LARGE}}$ and other methods on MS MARCO

	MRR@10 (Dev)	MRR@10 (Eval)
BM25	0.167	0.165
K-NRM (Xiong *et al.*, 2017)	0.218	0.198
Conv-KNRM (Dai *et al.*, 2018)	0.290	0.271
$\text{BERT}_{\text{LARGE}}$	0.365	0.358

4.3 Discussions and Further Reading

In this subsection, we discuss the characteristics of the two matching approaches and give more references for further reading.

4.3.1 Discussions

Both the approach of representation learning and the approach of matching function learning have been intensively studied. The two approaches have both advantages and limitations and have strong connections to traditional matching and ranking models in IR.

Representation learning gives the final matching score based on the semantic representations of query and document, which are respectively learned from the raw representations of query and document. The semantic representations of query and document are embedding vectors (real-valued vectors), which means that we represent the query and the document in a common semantic space. This approach is natural and has a strong connection with the conventional latent space models. The approach can effectively address the term mismatch problem if the semantics of query and document are represented very well. However, there also exist limitations in the approach. The queries and documents are represented independently before the final step of matching score calculation. The underlying assumption is that there exist universal representations for queries and documents, and the representations can be compared for determination of relevance. However, queries and documents usually have semantics at multiple levels (e.g., local and global levels). It is better, therefore, if queries and the documents can be compared at different levels. In other words, the representations of queries and documents and the interactions of queries and documents at multiple levels are better to be modeled.

Traditional latent space matching models for search (e.g., PLS, RMLS) and matching methods using topic models (e.g., LSI, PLSA, LDA) (Li and Xu, 2014) also learn the semantic representations of queries and documents. From this viewpoint, the approach of representation learning has similarities with the traditional approaches. However, the deep learning models have advantages, because (1) they employ deep neural networks to map the queries and documents into the semantic space and thus can obtain richer representations; and (2) the mapping functions and embeddings of words in the queries and documents can be jointly learned in an end-to-end fashion.

Matching function learning, on the other hand, generates the final matching score on the basis of both representations and interactions of query and document. Since the basic matching signals are modeled with both high-level representations (e.g., semantic representations) and low-level representations (e.g., term-level representations), the approach has the ability to conduct more accurate matching.

Traditional IR models (e.g., VSM, BM25, and LM4IR) also compare query words and document words and aggregate the matching signals. From this viewpoint, the approach of matching function learning has similarities with the traditional IR approach. The deep learning models are superior to the traditional IR models, however, because (1) they can capture matching signals not only at the local level (e.g., term level) but also at the global level (e.g., semantic level); (2) they can naturally keep and take into consideration the positions of matching; (3) it is possible to conduct end-to-end learning and achieve better performance; and (4) they can more easily leverage weak-supervision data (e.g., clickthrough logs).

Representation learning and matching function learning are not mutually exclusive. Matching models have also been developed that can take the advantages of both approaches. Some methods directly combine the scores from the representation learning and the matching function learning, as in Mitra *et al.* (2017). Other methods utilize the attention mechanism to alternatively construct the representations of queries and documents and make interactions between the representations (Yang *et al.*, 2019a).

4.3.2 Further Reading

Semantic matching in search is a very active research topic. Here we list other related work on text matching and cross-modal matching and the benchmark datasets and open-source software packages.

Papers

A large number of models are proposed for conducting matching in search. One research direction is to learn more sophisticated representations. For the representation learning approach, Yin and Schütze (2015)

propose the CNN-based Multi-GranCNN model, which learns query representations and document representations at multiple levels, including words, phrases, and the entire text. Wan *et al.* (2016a) present MV-LSTM, an LSTM-based model to achieve multiple positional sentence representations, for capturing the local information as well as the global information in the sentences. Nie *et al.* (2018) point out that different levels of matching, from low-level term matching to high-level semantic matching, are required due to the nature of natural language. A Multi-level Abstraction Convolutional Model (MACM) is proposed for generating multi-levels of representations and aggregating the multi-levels of matching scores. Huang *et al.* (2017) also present an approach that combines CNN and LSTM to exploit from character-level to sentence-level features for performing matching. Jiang *et al.* (2019a) present MASH RNN for matching of long documents. In the method, bidirectional GRU and the attention mechanism are adopted as the encoders for constructing the document representations at the sentence level, passage level, and document level. Liu *et al.* (2019a) propose to encode news articles with a concept interaction graph and conduct matching based on the sentences that enclose the same concept vertices.

For the matching function learning approach, the attention mechanism is intensively used. For example, Attention Based Convolutional Neural Network (ABCNN) (Yin *et al.*, 2016) integrates the attention mechanism into CNNs for general sentence pair modeling. In the model, the representation of each sentence can take its counterpart into consideration. The Bilateral Multi-Perspective Matching (BiMPM) (Wang *et al.*, 2017c) model matches the two encoded sentences in two directions. In each matching direction, each position of one sentence is matched against (attended to) all positions of the other sentence from multiple perspectives. Multi-Cast Attention Networks (MCAN) (Tay *et al.*, 2018d) performs a series of soft attention operations for question-answer matching. One advantages of MCAN is that it allows an arbitrary number of attention mechanisms to be casted, and allows multiple attention types (e.g., co-attention, intra-attention) and attention variants (e.g., alignment-pooling, max-pooling, mean-pooling) to be executed simultaneously. Also, Tay *et al.* (2018c) argue that the co-Attention models in asymmetrical matching tasks require different treatments to the

attentions for symmetrical tasks. They propose Hermitian Co-Attention Recurrent Network (HCRN) in which the attention mechanism is based on the complex-valued inner product (Hermitian products). Tan *et al.* (2018) propose the Multiway Attention Networks (MwAN) in which multiple attention functions are employed to match sentence pairs under the matching-aggregation framework. Chen *et al.* (2018b) propose the Multi-channel Information Crossing (MIX) model to compare the query and document at various granularities, forming a series of matching matrices. Attention layers are then imposed for capturing the interactions and producing the final matching score. Attention-based Neural Matching Model (aNMM) (Yang *et al.*, 2016) is another attention-based neural matching model. Given the matching matrix, for each query word, a value-shared weighting scheme instead of the position-shared weighting scheme is used for calculating the matching signals for the word. The signals of different query words are aggregated with an attention network. Nogueira *et al.* (2019) propose a three-stage ranking architecture for search. The first stage is implemented with BM25, and the second stage and the third stage are respectively implemented with pointwise and pairwise BERT based on learning to rank. Yang *et al.* (2019b) and Qiao *et al.* (2019) also apply the BERT model to ad-hoc retrieval and passage retrieval. Reimers and Gurevych (2019) propose Sentence-BERT for reducing the computational overhead for text matching. For the task of natural language inference, Chen *et al.* (2017b) propose a sequential inference models based on chain LSTMs, called Enhanced Sequential Inference Model (ESIM). ESIM explicitly considers recursive architectures in both local inference modeling and inference composition. Gong *et al.* (2018) propose the Interactive Inference Network (IIN) which hierarchically extracts semantic features from interaction space and performs high-level understanding of the sentence pair. It is shown that the interaction tensor (attention weight) can capture the semantic information to solve the natural language inference task.

For unsupervised matching models, Van Gysel *et al.* (2017) propose to adapt NVSM to the entity space, which can be used for product search (Van Gysel *et al.*, 2016a, 2018) and expert search (Van Gysel *et al.*, 2016b). Zamani and Croft (2017) also present an approach to learning

the word and query representations in an unsupervised manner. Weakly supervised models are also proposed in the IR community for the tasks of text representation, matching, and ranking. As for training models using weak supervision, Zamani and Croft (2016) propose a framework in which the queries can be represented based on the individual word embedding vectors. The parameters are estimated using pseudo-relevant documents as training signals. Zamani and Croft (2017) propose to train a neural ranking model using weak supervision. Still, the labels are obtained automatically from pseudo-relevance feedback and without the need of using human annotation or external resource. Dehghani *et al.* (2017) propose to use the output of BM25 as a weak supervision signal to train neural ranking models. Haddad and Ghosh (2019) suggest to leverage multiple unsupervised rankers to generate soft training labels and then learn neural ranking models based on the generated data. Zamani *et al.* (2018a) propose to train the query performance prediction model with multiple weak supervision signals. Zamani and Croft (2018b) provides a theoretical justification for weak supervision for information retrieval.

Usually, the matching models assume that query and document are homogeneous (e.g., short texts[4]), and symmetric matching functions are utilized. Pang *et al.* (2016a) study matching between short queries and long documents, on the basis of the aforementioned MatchPyramid model (Pang *et al.*, 2016b). The results show that queries and documents in web search are heterogeneous in nature: queries are short while documents are long, and thus learning an asymmetric matching model is a better choice. Convolutional Kernel-based Neural Ranking Model (Conv-KNRM) (Dai *et al.*, 2018) extends the K-NRM model, makes use of CNNs to represent n-grams of various lengths, and performs soft matching of the n-grams in a unified embedding space. Researchers also observe that a long document consists of multiple passages and matching with the query can be determined by some of them. Thus, the document can be split into multiple passages and matched individually for capturing fine-grained matching signals. In DeepRank (Pang *et al.*, 2017a), the document is split into term-centric contexts, each

[4]For documents, only the titles, anchors, or clicked queries are used.

corresponding to a query term. The local relevance between each (query, term-centric context) pair is calculated with the MatchPyramid model. The local relevance scores are then aggregated as the query-document matching score.

Similarly, Position-Aware Convolutional-Recurrent Relevance Matching (PACRR) (Hui *et al.*, 2017) splits the document with a sliding window. The focused region can be the first-k words in the document or the most similar context positions in the document (k-window). The context-aware PACRR (Co-PACRR) (Hui *et al.*, 2018) extends PACRR by incorporating the components that can model the context information of the matching signals. Fan *et al.* (2018) propose the Hierarchical Neural maTching model (HiNT) model in which the document is also split into passages. The local relevance signals are calculated between the query and the passages of the document. The local signals are accumulated into different granularities and finally combined into the final matching score. Commercial Web search engines need to consider more than just one document field. Incorporating different sources of document description (e.g., title, URL, body, anchor, etc.) is useful to determine the relevance of the document to the query (Robertson *et al.*, 2004). To address the problem of leveraging multiple fields, Zamani *et al.* (2018c) propose NRM-F which introduces a masking method to handle missing information from one field, and a field-level dropout method to avoid relying too much on one field. In Hierarchical Attention Retrieval Model (HAR) (Zhu *et al.*, 2019), word-level cross-attention is conducted to identify the words that most relevant for a query, and hierarchical attention is conducted at the sentence and document levels.

As for cross-modal query-document matching, CCA (Hardoon *et al.*, 2004) and semantic correlation matching (Rasiwasia *et al.*, 2010) are traditional models. Both of the models aim to learn linear transformations to project two objects in different modalities into a common space such that their correlation is maximized. To extend the linear transformations into non-linear transformations, Kernel Canonical Correlation Analysis (KCCA) (Hardoon and Shawe-Taylor, 2003) is proposed, which finds maximally correlated projections in the reproducing kernel Hilbert space using a kernel function. A method is introduced in Karpathy *et al.* (2014) to embed fragments of images (objects in an image) and fragments of

sentences into a common space, and calculate their similarities as dot products. The final image-sentence matching score is defined as the average thresholded score of their pairwise fragment matching scores. Multimodal Convolutional Neural Networks (m-CNNs) (Ma *et al.*, 2015) adopts CNN to compute the multi-modal matching scores at the word-level, phrase-level, and sentence-level. To better uniformly represent multi-media objects with embeddings, a variety of multi-modal deep neural networks are developed, including the models proposed in Wang *et al.* (2016), Eisenschtat and Wolf (2017), Liu *et al.* (2017), Wang *et al.* (2018b), Huang *et al.* (2018), Balaneshin-Kordan and Kotov (2018), and Guo *et al.* (2018).

Benchmark Datasets

A number of publicly available benchmark datasets are used for training and testing semantic matching models. For example, the traditional information retrieval datasets such as the TREC collections[5] (e.g., Robust, ClueWeb, and Gov2 etc.), the NTCIR collections,[6] and Sogou-QCL (Zheng *et al.*, 2018b) are suitable for experiments on query-document matching. Question Answering (QA) (and community-based QA) collections such as TREC QA,[7] WikiQA (Yang *et al.*, 2015), WikiPassageQA (Cohen *et al.*, 2018), Quora's 2017 question dataset,[8] Yahoo! Answers collection (Surdeanu *et al.*, 2011), and MS MARCO (Nguyen *et al.*, 2016) are also used for research on deep matching models as well. Other natural language processing datasets such as MSRP (Dolan and Brockett, 2005) and SNLI (Bowman *et al.*, 2015) are also exploited.

Open Source Packages

A number of open-source packages for matching are available on the web. MatchZoo[9] is an open-source project dedicated to research on deep

[5] https://trec.nist.gov/data.html.
[6] http://research.nii.ac.jp/ntcir/data/data-en.html.
[7] https://trec.nist.gov/data/qamain.html.
[8] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs.
[9] https://github.com/NTMC-Community/MatchZoo.

text matching (Guo *et al.*, 2019a). TensorFlow Ranking[10] is a subproject of TensorFlow that aims at solving large-scale search ranking problems in a deep learning framework (Pasumarthi *et al.*, 2019). Anserini[11] is an open-source information retrieval toolkit built on Lucene that aims at bridging the gap between academic research and real-world applications (Yang *et al.*, 2018).

[10] https://github.com/tensorflow/ranking.
[11] https://github.com/castorini/Anserini.

5

Deep Matching Models in Recommendation

In this section, we introduce representative deep matching methods in recommendation. As in Section 4, we categorize the methods into two groups: (1) methods of representation learning, and (2) methods of matching function learning. In the first category, neural networks are employed to create representations of users and items to make a comparison between them and generate a final matching score. In the second category, neural networks are utilized to conduct interactions between users and items (and possibly contexts) to generate matching signals and aggregate them to a final matching score. Table 5.1 shows a taxonomy of the representative deep matching models in recommendation.

5.1 Matching Based on Representation Learning

Matching models based on representation learning adopt the general framework of latent space model as described in Figure 4.1. In short, given user u in the user space $u \in \mathcal{U}$ and item i in the item space $i \in \mathcal{I}$, functions ϕ_u: $\mathcal{U} \mapsto \mathcal{H}$ and ϕ_i: $\mathcal{I} \mapsto \mathcal{H}$ stand for mappings from the user space $\mathcal{U}$ and from the item space $\mathcal{I}$ to a new space $\mathcal{H}$, respectively.

Table 5.1: Deep matching models in recommendation

	Input Representation	**Technique Categorization**	**Models**
Methods Based on Representation Learning	Unordered Interactions	MLP-based	DeepMF (Xue *et al.*, 2017), YouTubeDNN (Covington *et al.*, 2016), MV-DNN (Elkahky *et al.*, 2015)
		Autoencoder-based	AutoRec (Sedhain *et al.*, 2015), CDAE (Wu *et al.*, 2016b), Mult-VAE (Liang *et al.*, 2018)
		Attention-based	NAIS (He *et al.*, 2018a), ACF (Chen *et al.*, 2017a), DIN (Zhou *et al.*, 2018)
	Sequential Interactions	RNN-based	GRU4Rec (Hidasi *et al.*, 2016), NARM (Li *et al.*, 2017), RRN (Wu *et al.*, 2017), Latent Cross (Beutel *et al.*, 2018)
		CNN-based	Caser (Tang and Wang, 2018), NextItNet (Yuan *et al.*, 2019), GRec (Yuan *et al.*, 2020)
		Attention-based	SASRec (Kang and McAuley, 2018), Bert4Rec (Sun *et al.*, 2019)
	Multi-modal Content	Categorical Attributes	NSCR (Wang *et al.*, 2017b), DeepCF (Li *et al.*, 2015)
		User Reviews	DeepCoNN (Zheng *et al.*, 2017), NARRE (Chen *et al.*, 2018a), CARP (Li *et al.*, 2019)
		Multimedia Content	VBPR (He and McAuley, 2016a), CDL (Lei *et al.*, 2016), ACF (Chen *et al.*, 2017a), MMGCN (Wei *et al.*, 2019), PinSage (Ying *et al.*, 2018)
	Graph Data	User-item Graph	NGCF (Wang *et al.*, 2019b), PinSage (Ying *et al.*, 2018), LightGCN (He *et al.*, 2020)
		Knowledge Graph	KGAT (Wang *et al.*, 2019a), RippleNet (Wang *et al.*, 2018a), KPRN (Wang *et al.*, 2019c)
		Social Network	DiffNet (Wu *et al.*, 2019b), GraphRec (Fan *et al.*, 2019)

Continued.

Table 5.1: Continued

	Input Representation	**Technique Categorization**	**Models**
Methods Based on Matching Function Learning	Two-way Matching	Similarity Learning	NCF (He *et al.*, 2017c), ConvNCF (He *et al.*, 2018b), DeepICF (Xue *et al.*, 2019), NNCF (Bai *et al.*, 2017)
		Metric Learning	CML (Hsieh *et al.*, 2017), TransRec (He *et al.*, 2017a), LRML (Tay *et al.*, 2018a)
	Multi-way Matching	Implicit Interaction Modeling	YouTubeDNN (Covington *et al.*, 2016), Wide&Deep (Cheng *et al.*, 2016), Deep Crossing (Shan *et al.*, 2016), DIN (Zhou *et al.*, 2018)
		Explicit Interation Modeling	NFM (He and Chua, 2017), AFM (Xiao *et al.*, 2017), HoAFM (Tao *et al.*, 2019), CIN (Lian *et al.*, 2018), TransFM (Pasricha and McAuley, 2018)
		Combination of Explicit and Implicit Interaction Modeling	Wide&Deep (Cheng *et al.*, 2016), DeepFM (Guo *et al.*, 2017), xDeepFM (Lian *et al.*, 2018)

The matching model between u and i is defined as

$$f(u, i) = F(\phi_u(u), \phi_i(i)), \tag{5.1}$$

where F is the similarity function over $\mathcal{H}$, such as inner product or cosine similarity. Different neural networks can be used to realize the representation functions ϕ_u and ϕ_i, depending on the forms of input data and the data properties of interest. We further categorize the methods into four types based on the forms of input data: (1) unordered interactions, (2) sequential interactions, (3) multi-modal content, and (4) linked graph.

The remainder of this subsection is organized to present each type of the methods in one subsection. In Subsection 5.1.1, we describe methods that represent a user with his/her unordered interactions with the system, such as deep matrix factorization and auto-encoder based methods. In Subsection 5.1.2, we explain methods that represent a user with the sequence of his/her interactions (ordered interactions), such as RNN-based and CNN-based sequential recommendation methods. In Subsection 5.1.3, we present methods that incorporate multi-modal content into the learning of representations, such as user/item attributes, texts, and images. In Subsection 5.1.4, we introduce recently developed methods that perform the learning of representations on graph data, such as user-item graph and knowledge graph.

5.1.1 Representation Learning from Unordered Interactions

The traditional matrix factorization model utilizes a one-hot ID vector to represent a user (an item), and performs one-layer linear projection to obtain a user (item) representation. As a one-hot vector only contains an ID information, it is not meaningful to perform multiple layers of non-linear transformation on the vector. Given that an abundance of user-item interaction data is available at the recommendation system, a natural idea is to represent a user with his/her interaction history, which encodes richer information. If we ignore the order of user-item interactions, an interaction history can be considered as an unordered set of interactions. Each interaction can be represented as a multi-hot vector denoting the interacted items by the user, where each dimension

corresponds to an item. We next review three types of methods that learn a user representation from unordered interactions: MLP-based, Auto-Encoder-based, and attention-based methods.

MLP-Based Methods

Deep Matrix Factorization (DeepMF) (Xue *et al.*, 2017) adopts the architecture of DSSM (Huang *et al.*, 2013). It has a two-tower structure, where one tower is for learning user representation, and the other is for learning item representation. In each tower, an MLP is employed to learn a representation from a multi-hot vector. The expectation is that the multi-layer nonlinear transformations on the interaction history can learn better representations to bridge the semantic gap between the user and item. Figure 5.1 illustrates the architecture of the DeepMF model.

Let the user-item interaction matrix be $\mathbf{Y} \in \mathcal{R}^{M \times N}$ where M and N denote the number of users and items, respectively; for explicit feedback, each entry y_{ui} is a rating score, and the score of 0 means that user u has not rated on item i before; for implicit feedback, each entry is a binary value, and the values of 1 and 0 denote whether or not user u has interacted with item i before. Let $\mathbf{y}_{u*} \in \mathcal{R}^N$ denote the u-th row of $\mathbf{Y}$, i.e., the multi-hot history vector of user u, and $\mathbf{y}_{*i} \in \mathcal{R}^M$ denote

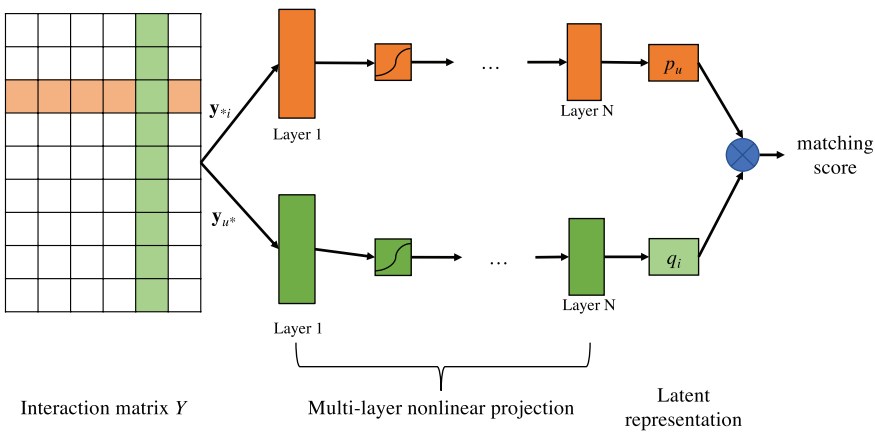

Figure 5.1: Model architecture of DeepMF.

the i-th column of $\mathbf{Y}$, i.e., the multi-hot history vector of item i. Then, we can express the matching function of DeepMF as:

$$\begin{aligned} \mathbf{p}_u &= \text{MLP}_1(\mathbf{y}_{u*}), \quad \mathbf{q}_i = \text{MLP}_2(\mathbf{y}_{*i}), \\ f(u,i) &= \text{cosine}(\mathbf{p}_u, \mathbf{q}_i) = \frac{\mathbf{p}_u^T \mathbf{q}_i}{||\mathbf{p}_u||^2 ||\mathbf{q}_i||^2}. \end{aligned} \tag{5.2}$$

As the spaces of users and items are different, DeepMF uses two MLPs with different parameters to represent users and items. Note that due to the sparse nature of $\mathbf{y}_{u*}$ and $\mathbf{y}_{*i}$, the overall complexity of the model is acceptable if we omit the zero entries in $\mathbf{Y}$ in implementation. It is also worth mentioning that it is not compulsory to use two towers — one can use MLP_1 only to obtain $\mathbf{p}_u$ and use simple embedding lookup for $\mathbf{q}_i$. Such a simplification is essentially equivalent to the auto-encoder based architecture, which is introduced next.

Auto-Encoder Based Methods

Auto-encoder is another choice to build a recommender model from the interaction history. Auto-encoder transforms the input data into a hidden representation, such that from a good hidden representation, one is almost able to recover the input data. In item-based AutoRec (Sedhain *et al.*, 2015), the input is the user history vector $\mathbf{y}_{u*} \in \mathcal{R}^N$, and the reconstruction of the input is:

$$\hat{\mathbf{y}}_{u*} = \sigma_2(\mathbf{W} \cdot \sigma_1(\mathbf{V}\mathbf{y}_{u*} + \mathbf{b}_1) + \mathbf{b}_2), \tag{5.3}$$

where σ_2 and σ_1 are activation functions, $\mathbf{V} \in \mathcal{R}^{d \times N}$ and $\mathbf{W} \in \mathcal{R}^{N \times d}$ are weight matrices, $\mathbf{b}_1 \in \mathcal{R}^d$ and $\mathbf{b}_1 \in \mathcal{R}^N$ are bias vectors. The reconstruction vector $\hat{\mathbf{y}}_{u*}$ is a N-dimension vector which stores the predicted matching scores of all items for user u. To learn parameters $\theta = \{\mathbf{V}, \mathbf{W}, \mathbf{b}_1, \mathbf{b}_2\}$, AutoRec minimizes the total loss over all inputs (users) with L_2 regularization:

$$L = \sum_{u=1}^{M} ||\mathbf{y}_{u*} - \hat{\mathbf{y}}_{u*}||^2 + \lambda ||\theta||^2.$$

Given that recommendation is essentially a matching plus ranking task, other loss functions like cross-entropy, hinge loss and pairwise loss can also be employed here, as demonstrated by Wu *et al.* (2016b).

In fact, we can view the AutoRec model as one using MLP on interaction history to learn a user representation, and using embedding lookup to obtain an item representation. To be more specific, we can reformulate Equation (5.3) to get the element-wise matching function:

$$f(u,i) = \hat{\mathbf{y}}_{u*,i} = \sigma_2(\underbrace{\mathbf{w}_{i*}}_{\mathbf{q}_i} \cdot \underbrace{\sigma_1(\mathbf{V}\mathbf{y}_{u*} + \mathbf{b}_1)}_{\Longrightarrow \mathbf{p}_u = \text{MLP}(\mathbf{y}_{u*})} + \mathbf{b}_2), \tag{5.4}$$

where $\mathbf{w}_{i*}$ denotes the i-th row of $\mathbf{W}$, which can be seen as the ID embedding of item i, and user representation $\mathbf{p}_u$ is equivalent to the output of a one-layer MLP with $\mathbf{y}_{u*}$ as input. The matching score is essentially the inner product of user representation $\mathbf{p}_u$ and item ID embedding $\mathbf{q}_i$, which falls into the latent space framework defined in Equation (5.1). If multiple hidden layers are used to build a "deep" auto-encoder, we can interpret it as replacing one-layer MLP with multi-layer MLP to obtain the user representation. As such, the auto-encoder architecture can be seen as a simplified variant of DeepMF.

Some later variants of AutoRec include Collaborative Denoising Auto-Encoder (CDAE) (Wu *et al.*, 2016b), which extends AutoRec by corrupting the input $\mathbf{y}_{u*}$ with random noises to prevent the model from learning a simple identity function and to discover a more robust representation. Liang *et al.* (2018) propose extending variational auto-encoder for recommendation, solving the representation learning problem from the perspective of generative probabilistic modeling.

Attention-Based Methods

One observation in the learning of user representation is that historical items may not equally contribute to the modeling of the user's preference. For example, a user may choose a trendy item based on its high popularity rather than his/her own interest. Although, in principle, an MLP learned from interaction history may be able to capture the complicated relationships (c.f. the universal approximation theorem of neural networks (Hornik, 1991)), the process is too implicit and there is no guarantee for that. To solve the problem, the Neural Attentive Item Similarity (NAIS) model (He *et al.*, 2018a) employs a neural attention

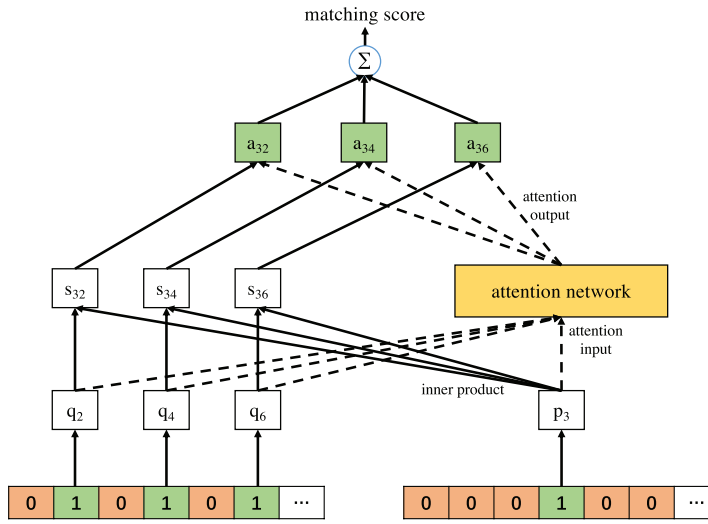

Figure 5.2: Model architecture of NAIS.

network to explicitly learn the weight of each historical item. Figure 5.2 shows the architecture of the model.

In short, NAIS is an extension of FISM by using a learnable weight on each interacted item of a user. Let $\mathcal{Y}_u$ be the set of interacted items of user u, and each item i is associated with two ID embedding vectors $\mathbf{p}_i$ and $\mathbf{q}_i$ to represent its role as a target item and a historical item, respectively. The matching function in NAIS is formulated as

$$\begin{aligned} f(u,i) &= \left(\sum_{j \in \mathcal{Y}_u \setminus \{i\}} a_{ij}\mathbf{q}_j\right)^T \mathbf{p}_i, \\ a_{ij} &= \frac{\exp(g(\mathbf{p}_i, \mathbf{q}_j))}{[\sum_{j \in \mathcal{Y}_u \setminus \{i\}} \exp(g(\mathbf{p}_i, \mathbf{q}_j))]^\beta}, \end{aligned} \tag{5.5}$$

where a_{ij} is an attention weight that controls the weight of the historical item j in estimation of the user u's matching score on the target item i. The attention network g is typically implemented as a one-layer MLP which outputs a scalar value (e.g., the MLP takes concatenation or element-wise product of $\mathbf{p}_i$ and $\mathbf{q}_j$ as input). The output of g is further processed by a smoothed softmax function where β is in $(0, 1)$ to smooth the weighted sum of active users (the default value of β is

0.5). By explicitly learning the weight of each interacted item with an attention network, the interpretability of representation learned from interaction history can also be improved. One can further enhance the non-linearity of representation learning by stacking an MLP above the sum pooling, such as in the deep neural network architecture for YouTube recommendation (Covington *et al.*, 2016).

It is worth highlighting that the NAIS attention is aware of the target item i when estimating the weight of a historical item j. This purposeful design is to address the limitation of static user representation when interacting with different items. For example, when a user considers whether to purchase a clothing item, the historical behaviors on the fashion category are more reflective of his/her aesthetic preference than the historical behaviors on the electronic category. The Deep Interest Network (DIN) model (Zhou *et al.*, 2018), which is independently proposed by the Alibaba team at the same time, adopts the same way of dynamic (target item-aware) user representation. It is shown to be useful to distill useful signals from user behavior history in large-scale e-commerce CTR prediction.

5.1.2 Representation Learning from Sequential Interactions

User-item interactions are naturally associated with timestamps, which record when an interaction happens. If the order of user-item interactions is considered, an interaction history becomes a sequence of item IDs. Modeling such a sequence can be useful for prediction of user behavior in the future, for example, purchase transition patterns from one item (e.g., phone) to another item (e.g., phone case) exist and recent purchases are more predictive of next purchases. Next, we present two types of sequential order based recommendation methods: RNN-based and CNN-based methods.

RNN-Based Methods

As one of the pioneering work on RNN for session-based recommendation, Hidasi *et al.* (2016) propose a GRU-based RNN for summarizing the sequential interactions (e.g., the sequence of the clicked items) in a session and making recommendation, called GRU4Rec. The input

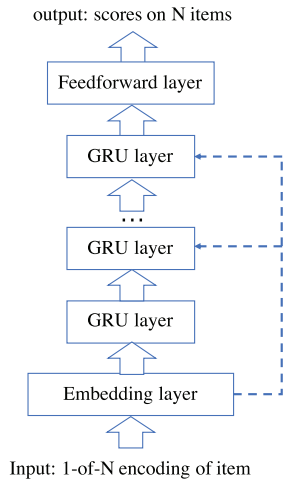

Figure 5.3: Model architecture of GRURec. It processes one item of the item sequence at once.

to the GRU4Rec model is a sequence of r clicked items in a session $\mathbf{x} = (x_1, \ldots, x_{r-1}, x_r)$, where each item is represented as a one-hot N-dimensional vector and N is the number of items. The output is the next event (clicked item) in the session. More specifically, at each position i in the sequence $\mathbf{x}$, the input is the state of the session, which can be a one-hot representation of the current item, or a weighted sum of representations of items so far, as shown in Figure 5.3. As the core of the network, a multi-layer GRU is used to receive the embeddings of the input representations, and the output of each GRU layer is the input to the next layer. Finally, feedforward layers are added between the last GRU layer and the output layer. The output is an N-dimensional vector, each representing the probability of the corresponding item being clicked in the next event of the session.

During training, pairwise ranking loss is utilized to learning the model parameters. Two types of loss functions are used. The BPR loss compares the score of a positive (preferred) item with those of several

sampled negative items. Thus, the BPR loss at a position is defined as:

$$L_s = -\frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j})),$$

where N_s is the number of sampled negative items, $\hat{r}_{s,i}$ (or $\hat{r}_{s,j}$) is the predicted score of item i (or j), i is the positive item, and j is the negative item. The other type of loss called TOP1 is also devised, which is the ratio of correctly ranked pairs with regularization. The TOP1 loss at a position is defined as:

$$L_s = \frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \sigma(\hat{r}_{s,j} - \hat{r}_{s,i}) + \sigma(\hat{r}_{s,j}^2).$$

To address the problem that the lengths of sessions vary, GRU4Rec employs session-parallel mini-batches for the optimization. In training, it uses popularity-based negative sampling, which assumes that the more popular an item is, the more likely the user knows about it, for generating the negative items.

One problem with the RNN-based models (including the above introduced GRU4Rec) is that they only consider the user's sequential behaviors (short-term interest) in the current session, and do not put enough emphasis on the user's general interest. To address the issue, Li *et al.* (2017) propose to combine the attention mechanism with RNN, called Neural Attentive Recommendation Machine (NARM). As shown in Figure 5.4, NARM employs an encoder-decoder framework for session-based sequential recommendation. Given a user's click sequence $\mathbf{x} = (x_1, x_2, \ldots, x_t)$ consisting of t clicked items, the global encoder in NARM scans the input sequence with a GRU, and uses the final hidden state $\mathbf{c}_t^g = \mathbf{h}_t$ as the representation of the user's sequential behavior. The local encoder in NARM also scans the input sequence with another GRU, and takes the weighted sum of the hidden states as the representation of the user's main intent:

$$\mathbf{c}_t^l = \sum_{j=1}^{t} \alpha_{tj} \mathbf{h}_j,$$

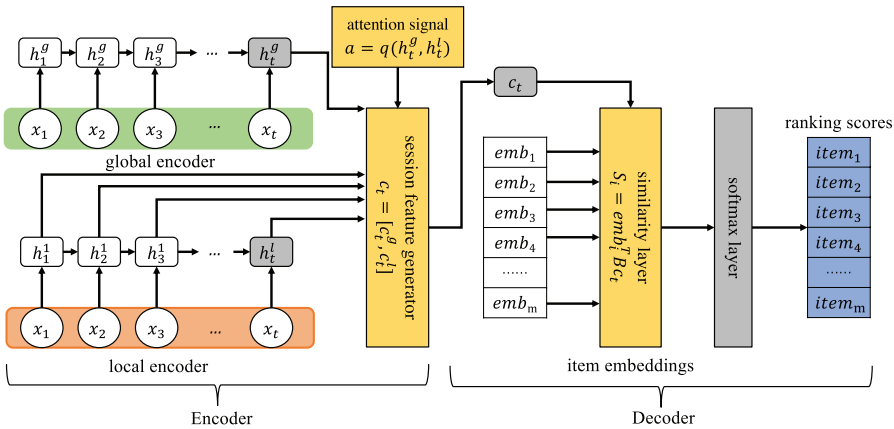

Figure 5.4: Model architecture of NARM.

where α_{tj} is the attention between the positions j and t. The unified sequence representation is formed as a combination of $\mathbf{c}_t^g$ and $\mathbf{c}_t^l$:

$$\mathbf{c}_t = \begin{bmatrix} \mathbf{c}_t^g \\ \mathbf{c}_t^l \end{bmatrix}.$$

The unified sequence representation at position t, as well as embedding of candidate item, are fed into the decoder. The similarity between the sequence representation at position t and embedding of candidate item i is calculated as a bilinear function:

$$s_i = \mathbf{e}_i^T \mathbf{B} \mathbf{c}_t,$$

where $\mathbf{B}$ is the matrix to be learned. A softmax layer is further imposed on the m item scores to generate a distribution (of click) over all candidate items, where m is the number of candidates.

To learn the model parameters, cross-entropy loss is used. Specifically, given a training sequence, at the position t NARM first predicts the probability distribution over the m items $\mathbf{q}_t$. From the log we know that the ground-truth probability distribution at t is $\mathbf{p}_t$. Thus, the cross-entropy loss is defined as

$$\mathcal{L} = \sum_{i=1}^{m} p_t^i \log q_t^i,$$

where p_t^i and q_t^i are the predicted probability and ground-truth probability for item i, respectively. The loss function can be optimized with the standard mini-batch SGD.

CNN-Based Methods

A representative CNN-based sequential recommendation method is Caser (Convolutional Sequence Embedding Recommendation Model) (Tang and Wang, 2018). The basic idea is to treat the interacted items in the embedding space as an "image", and then perform 2D convolution on the image. Figure 5.5 shows the structure of the Caser model.

Let $\mathbf{E} \in \Re^{t \times k}$ be the embedding matrix of interacted items, where t is the number of interacted items (length) and k is the dimension of embeddings (width). Each row of the matrix is the embedding vector of an item. Unlike a real image in computer vision, there are two difficulties in applying convolution operations into $\mathbf{E}$ for sequential recommendation. First, the "image" length t can be different for different users. Second, E may not have spatial relations like real images in terms of the width of embedding space. Hence, it is not suitable to employ the standard 2D CNN filters, such as 3×3 or 5×5.

To solve the two issues, Caser introduces "full-width" CNN filters and max-pooling operations. Specifically, the convolutional operations

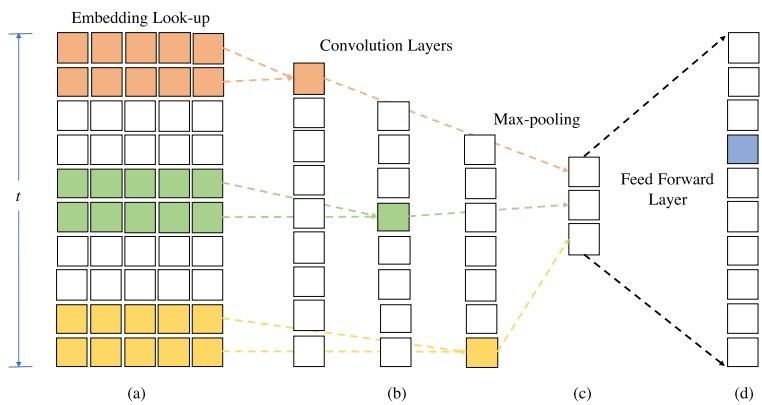

Figure 5.5: Architecture of caser.

in Caser cover the full columns of the sequence "image". That is, the width of the filter has the same size as the embedding dimension, and the height of the filter varies (see the different colors of Figure 5.5(a)). As a result, filters of different sizes produce feature maps with different lengths. To ensure all feature maps having the same size, a max-pooling operation is then performed on each feature map by extracting only the largest value. As shown in Figure 5.5(b), a number of 1×1 feature maps are produced after max pooling. Following the concatenation operation (Figure 5.5(c)) and softmax layer (5.5(d)), Caser outputs the probabilities of next items. Note that in addition to the horizontal filter, Caser also utilizes a vertical filter with the size of t, which is omitted in Figure 5.5. The feature maps $1 \times k$ are concatenated together with other feature maps.

In fact, due to the max-pooling operations, Caser is not well-suited to model long-range sequences or repeated sequences. To alleviate the problem, Caser employs a data augmentation method by sliding a window over the original sequence to create a set of subsequences. For example, assume that the original sequence is $\{x_1, \ldots, x_{10}\}$ and the sliding window size is 5, and then the subsequences are generated as $\{x_1, \ldots, x_5\}, \{x_2, \ldots, x_6\}, \ldots, \{x_6, \ldots, x_{10}\}$, which are fed into model training together with the original sequence.

After Caser is proposed, several methods are developed to improve the CNN framework for long-range sequential recommendation. A representative method is NextItNet (Yuan *et al.*, 2019), which differs from Caser in two ways: (1) NextItNet models the user sequence in an autoregressive manner, i.e., sequence-to-sequence (seq2seq); (2) NextItNet exploits the stacked dilated CNN layers to increase the model receptive field, and thus omits the use of max-pooling. Let $p(\mathbf{x})$ be the joint distribution of item sequence $\{x_0, \ldots, x_t\}$. According to the chain rule, $p(\mathbf{x})$ can be modeled as:

$$p(\mathbf{x}) = \prod_{i=1}^{t} p(x_i \mid x_0, \ldots, x_{i-1}, \theta) p(x_0), \tag{5.6}$$

where θ denotes the model parameters, and $\prod_{i=1}^{t} p(x_i \mid x_0, \ldots, x_{i-1}, \theta)$ denotes the probability of the i-th item x_i conditioned on all preceding items $\{x_0, \ldots, x_{i-1}\}$. For clarity, we make a comparison between

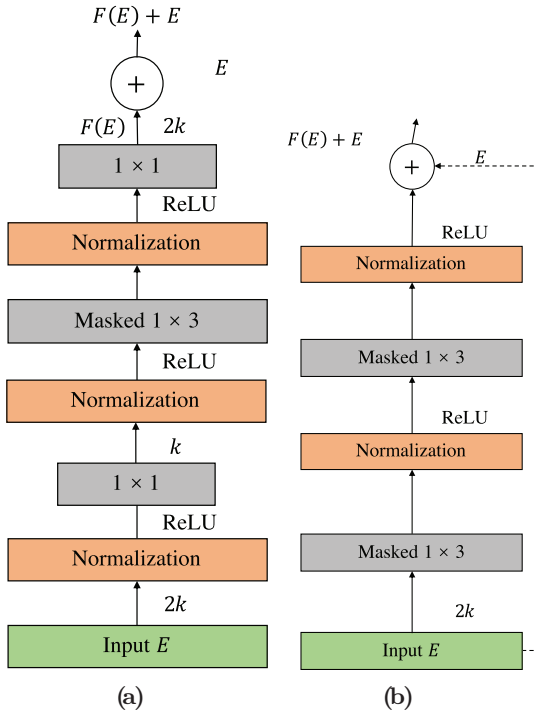

Figure 5.6: Dilated residual blocks (a), (b).

NextItNet and Caser in the generative process:

$$\begin{aligned} \textit{Caser}&: \underbrace{\{x_0, x_1, \ldots, x_{i-1}\}}_{input} \Rightarrow \underbrace{x_i}_{output} \\ \textit{NextItNet}&: \underbrace{\{x_0, x_1, \ldots, x_{i-1}\}}_{input} \Rightarrow \underbrace{\{x_1, x_2, \ldots, x_i\}}_{output} \end{aligned} \tag{5.7}$$

where $\Rightarrow$ represents "predict". In fact, the final objective function of NextItNet is a combination of all losses of tokens in the entire output sequence. Hence, NextItNet is usually not sensitive to the batch size.

Moreover, NextItNet introduces two types of dilated residual blocks, as illustrated in Figure 5.6. The dilation factors are doubled for every convolution layer and then repeated, e.g., $\{1, 2, 4, 8, 16, \ldots, 1, 2, 4, 8, 16\}$. The design allows an exceptional increase of receptive fields. Hence, NextItNet is well-suited to model long-range user sequences and capture

long-distance item dependencies. In addition, unlike the RNN models, the CNN models based on the seq2seq framework face the data leakage issue because the future data can be observed by the higher-layers of the network. To overcome this problem, NextItNet introduces the masking technique, by which the predicted item itself and future items are hidden to the higher-layers. Masking can be simply implemented by padding the input sequence.

Attention-Based Methods

Attention is also used for learning representations from sequential interactions. A well-known method is the *Self-Attention based Sequential Recommender* (SASRec) (Kang and McAuley, 2018) model. It is inspired by the Transformer (Vaswani *et al.*, 2017), taking the core design of self-attention to assign a weight to an item in the sequence adaptively. Figure 5.7 shows the structure of the SASRec model.

Let $\mathbf{E} = \mathbf{V} + \mathbf{P} \in \mathcal{R}^{t \times k}$ be the embedding matrix of the input sequence, where each row represents an interacted item. The two constituent matrices, $\mathbf{V}$ denotes the embeddings of the items and $\mathbf{P}$ denotes the embeddings of the positions of the corresponding items in the sequence. The reason for injecting $\mathbf{P}$ is to augment the attention mechanism with the sequential order of the items, since the attention mechanism by nature is not aware of the sequential order. Then, $\mathbf{E}$ is fed into a stack of self-attention blocks, where each block has two parts: a Self-Attention (SA) layer and a point-wise Feed-Forward Network (FFN):

$$\mathbf{S}^{(l)} = SA(\mathbf{F}^{(l-1)}), \qquad \mathbf{F}^{(l)} = FFN(\mathbf{S}^{(l)}), \tag{5.8}$$

where $\mathbf{F}^{(0)} = \mathbf{E}$. The SA layer is defined as:

$$\begin{aligned} SA(\mathbf{F}) &= Attention(\mathbf{F}\mathbf{W}^Q, \mathbf{F}\mathbf{W}^K, \mathbf{F}\mathbf{W}^V) \\ Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \end{aligned} \tag{5.9}$$

where $\mathbf{W}^Q$, $\mathbf{W}^K$ and $\mathbf{W}^V$ are the weight matrices of queries, keys and values, respectively. The SA layer uses the same objects $\mathbf{F}$ as quires, keys, and values, which are projected by different weight matrices to improve model flexibility. Intuitively, the attention calculates a weighted

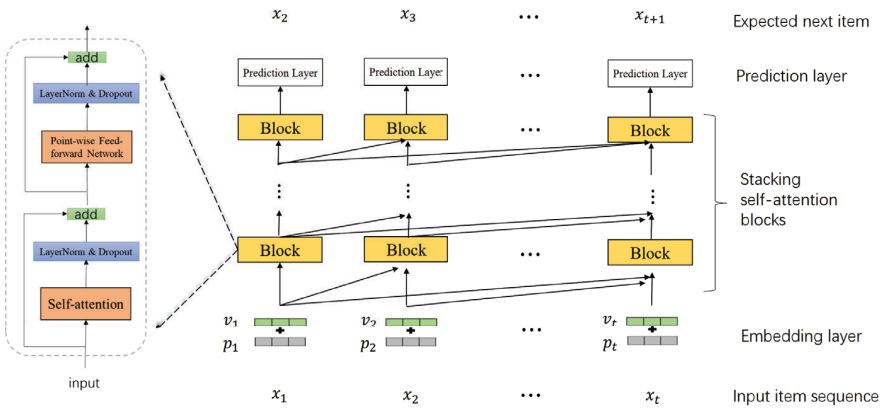

Figure 5.7: The structure of SASRec.

sum of all value vectors, where the weight between query i and value j relates to the interaction between query i and key j, i.e., the result of $softmax(\cdot)$. The denominator $\sqrt{d}$ is to avoid overly large values of the inner products that may cause gradient problems.

Point-wise Feed-Forward Network has the following form:

$$FFN(S) = Relu(SW_1 + b_1)W_2 + b_2 \tag{5.10}$$

where W_1, W_2 and b_1, b_2 are weights and bias, and Relu is activation function. FFN is used to enable nonlinearity and consider the interactions between different latent dimensions.

Another issue to be noted is that when recommending the next item of one sequence, only the previous items are known (see Figure 5.7). This is achieved by forbidding the link between $\mathbf{Q}_i$ (the i-th query) and $\mathbf{K}_j$ (the j-th key) for $j > i$, i.e., setting the corresponding attention weights to 0. When the network goes deeper, the model becomes more difficult to train. To solve the problem, SASRec employs layer normalization (Ba *et al.*, 2016), dropout (Srivastava *et al.*, 2014), and residual connection (He *et al.*, 2016a) on each SA layer and FFN layer:

$$\begin{aligned} \mathbf{S}^{(l)} &= \mathbf{F}^{(l-1)} + Dropout(SA(LayerNorm(\mathbf{F}^{(l-1)}))), \\ \mathbf{F}^{(l)} &= \mathbf{S}^{(l)} + Dropout(FFN(LayerNorm(\mathbf{S}^{(l)}))). \end{aligned} \tag{5.11}$$

At last, the output of the last Self-attention block is used for prediction. Given the historical item sequence $\{v_1, v_2, \ldots, v_t\}$, the next item

need be predicted based on $\mathbf{F}_t^{(L)}$, where L is the number of blocks. The predicted score of the target item i is:

$$\hat{r}_i = \mathbf{N}_j^T \mathbf{F}_i^{(L)}, \tag{5.12}$$

$\mathbf{N} \in \mathcal{R}^{(|I| \times d)}$ is the embedding matrix of target items, which can either be trained end-to-end, or the same as the item embeddings in the input layer. The authors show that sharing the item embeddings could be beneficial. The objective function is pointwise cross-entropy loss, which is similar to Caser: predicting v_2 based on the sub-sequence $\{v_1\}$, predicting v_3 based on the sub-sequence $\{v_1, v_2\}$, and so on.

In addition to SASRec, another representative attention-based method for learning the representation of sequential interactions is BERT4Rec (Sun *et al.*, 2019). The main difference is that it takes a bi-directional self-attention model to process the sequence, which can utilize both left (previous) and right (future) interactions. The future interactions are arguably useful for prediction (Yuan *et al.*, 2020) because they also reflect the preference of the user, and the rigid order of interactions may not be so important (the order is derived from the interaction timestamp). To this end, they revise SASRec in two ways: (1) revise self-attention to remove the zero constraints on the attention weights of $\mathbf{Q}_i$ and $\mathbf{K}_j$ for $j > i$, and (2) randomly mask some items in the sequence and predict the masked items based on the left and right interactions, to avoid information leakage.

5.1.3 Representation Learning from Multi-Modal Content

In addition to user-item interactions, users and items are often associated with descriptive features such as categorical attributes (e.g., age, gender, product category) and texts (e.g., product description, user reviews). Besides, in a recommender system for multi-modal items like images, videos, and music, their multi-modal descriptive features are readily available. Leveraging such side information is beneficial for learning better representations, especially for sparse users and items that have few interactions. In this subsection, we review neural recommendation models that integrate multi-modal side information for representation

learning. The representation learning component can be abstracted as:

$$\begin{aligned} \phi_u(u) &= \text{COMBINE}(\mathbf{p}_u, f(\mathbf{F}_u)), \\ \phi_i(i) &= \text{COMBINE}(\mathbf{q}_i, g(\mathbf{G}_i)), \end{aligned} \tag{5.13}$$

where $\mathbf{p}_u$ denotes the embedding of user u that is learned from historical interactions (e.g., the ID embedding and the embeddings from the previous subsections can be used), $\mathbf{F}_u$ denotes the side features of user u that can be a matrix or a vector, and $f(\cdot)$ is the representation learning function for side features; similar notations apply to $\mathbf{q}_i, \mathbf{G}_i$, and $g(\cdot)$ for the item side. COMBINE$(\cdot, \cdot)$ is a function that combines the embedding from historical interactions and the side features. The functions $f(\cdot)$, $g(\cdot)$, and COMBINE$(\cdot, \cdot)$ can all be realized as a deep neural network. In the next, we introduce specific methods and divide them into three types: learning from categorical attributes, user reviews, and multimedia content such as image and video.

Learning from Categorical Attributes

Wang *et al.* (2017b) propose an attribute-aware deep CF model, which is illustrated in Figure 5.8. It projects each categorical feature into an embedding vector and then performs bi-interaction pooling (He and Chua, 2017) with the user (item) ID embedding. Finally, the pooled user vector and item vector are combined into an MLP to obtain the prediction score:

$$\begin{aligned} \phi_u(u) &= \text{BI-Interaction}(\mathbf{p}_u, \{\mathbf{f}_t^u\}_{t=1}^{V_u}) = \sum_{t=1}^{V_u} \mathbf{p}_u \odot \mathbf{f}_t^u + \sum_{t=1}^{V_u} \sum_{t'=t+1}^{V_u} \mathbf{f}_t^u \odot \mathbf{f}_{t'}^u, \\ \phi_i(i) &= \text{BI-Interaction}(\mathbf{q}_i, \{\mathbf{g}_t^i\}_{t=1}^{V_i}) = \sum_{t=1}^{V_i} \mathbf{q}_u \odot \mathbf{g}_t^i + \sum_{t=1}^{V_i} \sum_{t'=t+1}^{V_i} \mathbf{g}_t^i \odot \mathbf{g}_{t'}^i, \\ \hat{y}_{ui} &= \text{MLP}(\phi_u(u) \odot \phi_i(i)), \end{aligned} \tag{5.14}$$

where $\mathbf{f}_t^u$ and $\mathbf{g}_t^i$ respectively denote the embeddings of user attribute and item attribute, V_u and V_i denote the numbers of attributes for user u and item i, respectively. The bi-interaction pooling operation considers all pairwise interactions among user ID embedding and attribute embeddings. The combined user representation $\phi_u(u)$ and item

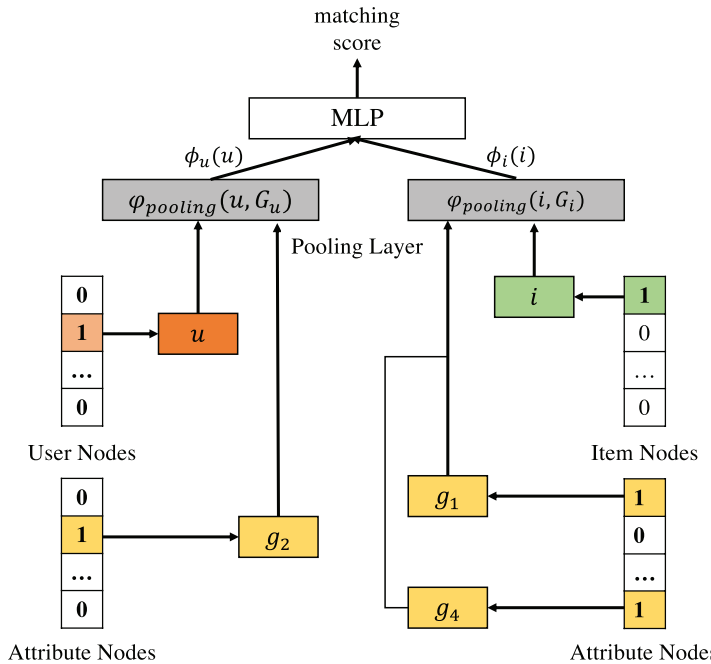

Figure 5.8: Model architecture of attribute-aware deep CF model.

representation $\phi_i(i)$ are interacted by element-wise product, followed by a MLP for the final prediction. The MLP is a learnable matching function (more details will be introduced in Subsection 5.2), which can also be replaced with simple inner product. The advantage of this architecture is that the interactions between user (item) attributes and the cross-interactions between user attributes and item attributes are well captured.

Li *et al.* (2015) propose a regularization-based approach to incorporate attributes into recommendation. The idea is to first learn representations from user features and item features with two auto-encoders, respectively, and then jointly train the representations within the recommendation task. The auto-encoder loss can be treated as a regularization term for recommendation. Figure 5.9 shows the model architecture. The left auto-encoder is built from user feature X with the hidden layer U as user representation and $L(X, U)$ as loss function; the right

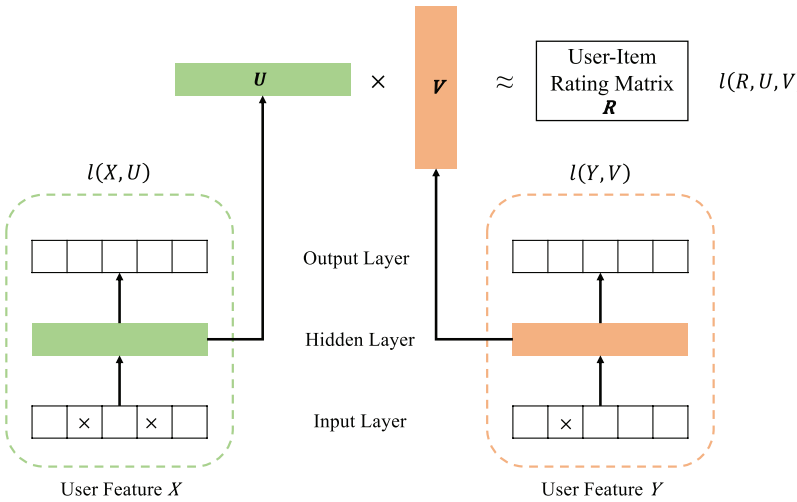

Figure 5.9: Model architecture of attribute-aware deep CF model.

auto-encoder is built from item feature Y with the hidden layer V as item representation and $L(Y, V)$ as the loss function. Then U and V are used to reconstruct the user-item rating matrix R for recommendation with $l(R, U, V)$ as loss function. The whole model is trained by joint optimization of the three loss functions $L(X, U)$, $L(Y, V)$, and $l(R, U, V)$.

Learning from User Reviews

The reviews given by other users often significantly influence the users' online purchasing decisions in a recommender system. Recent studies find that leveraging the information in the reviews can help the system to not only improve the accuracy but also enhance the explainability in recommendation.

As one of the representative works, Zheng *et al.* (2017) propose a deep learning model to jointly learn item properties and user opinions from the reviews, called Deep Cooperative Neural Networks (DeepCoNN). As shown in Figure 5.10, DeepCoNN consists of two parallel neural networks. One focuses on learning of user opinions from the reviews (called Net_u), and the other learning of item properties from the reviews

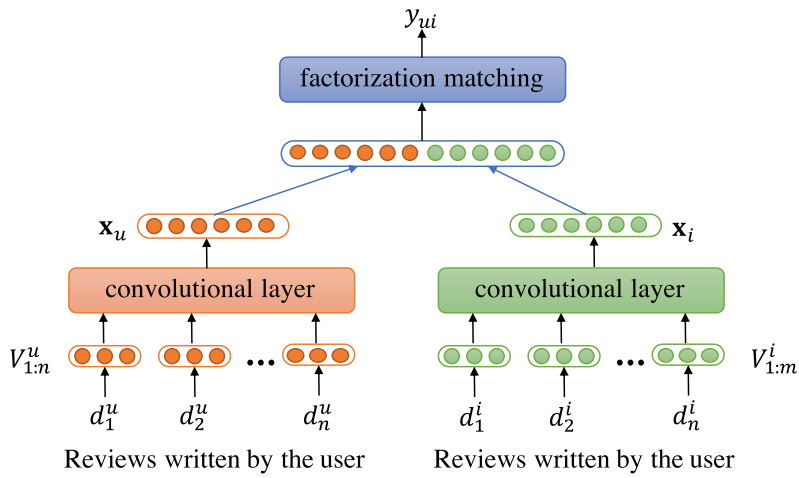

Figure 5.10: Model architecture of DeepCoNN.

(called Net_i). The two networks are coupled together in their last layers and jointly learned. Given all the reviews written by a user u, Net_u first merges the reviews into a single document $d^u_{1:n}$ with n words. Then, the document is represented as a matrix of word vectors $V^u_{1:n}$:

$$V^u_{1:n} = [\phi(d^u_1), \phi(d^u_2), \ldots, \phi(d^u_n)],$$

where d^u_k denotes the k-th word in the document $d^u_{1:n}$, the look-up function $\phi(d^u_k)$ returns the embedding of the input word d^u_k, and c is the dimension of embeddings. Then, a one-dimensional CNN is employed to summarize the reviews into a representation vector $\mathbf{x}_u$:

$$\mathbf{x}_u = Net_u(d^u_{1:n}) = CNN(V^u_{1:n}).$$

Similarly, given all the reviews for an item i, Net_i also merges the reviews into a single document $d^i_{1:m}$ of m words, creates a matrix of word vectors $V^i_{1:m}$, and employs a one-dimensional CNN to summarize the reviews into a representation vector:

$$\mathbf{x}_i = Net_i(d^i_{1:m}) = CNN(V^i_{1:m}).$$

The final matching score of user u and item i is calculated on the basis of the two representation vectors. Specifically, $\mathbf{x}_u$ and $\mathbf{x}_i$ are concatenated

into one vector $\mathbf{z} = [\mathbf{x}_u^T, \mathbf{x}_i^T]^T$ and a factorization machine (FM) is used to calculate the score:

$$y_{ui} = w_0 + \sum_{k=1}^{|\mathbf{z}|} w_k z_k + \sum_{k=1}^{|\mathbf{z}|} \sum_{l=k+1}^{|\mathbf{z}|} w_{kl} z_k z_l,$$

where w_0, w_k, w_{kl}'s are parameters of FM.

Chen *et al.* (2018a) point out that simple concatenation of reviews as in DeepCoNN means equal treatments of informative reviews and non-informative reviews. To address the problem, they propose the Neural Attention Regression with Review-level Explanation (NARRE) in which reviews are assigned weights and informative reviews are emphasized.

The model architecture of NARRE is shown in Figure 5.11. Specifically, given all the m reviews written for an item i, the reviews are first transformed into matrices $V_{i,1}, V_{i,2}, \ldots, V_{i,m}$. The matrices are then sent to a convolutions layer obtaining the feature vectors $O_{i,1}, O_{i,2}, \ldots, O_{i,m}$. After that, an attention-based pooling layer is exploited to aggregate informative reviews to characterize the item i. The attention weight for item i's l-th review is defined as

$$a_{i,l} = \frac{\exp(a_{il}^*)}{\sum_{k=1}^{m} \exp(a_{ik}^*)},$$

where a_{il}^* is the attention weight

$$a_{il}^* = \mathbf{h}^T \text{ReLU}(\mathbf{W}_O O_{i,l} + \mathbf{W}_u u_{il} + \mathbf{b}_1) + b2,$$

where u_{il} is the embedding of the user who writes the l-th review; $\mathbf{W}_O, \mathbf{W}_u, \mathbf{h}, \mathbf{b}_1$, and b_2 are model parameters. The final representation of item i is written as

$$\mathbf{x}_i = \mathbf{W}_0 \sum_{l=1}^{m} a_{i,l} O_{i,l} + \mathbf{b}_0,$$

where $\mathbf{W}_0$ and $\mathbf{b}_0$ are model parameters. Given all the m reviews written by a user u, its representation, denoted as $\mathbf{x}_u$, is calculated similarly.

In NARRE, an extended latent factor model is used as the prediction layer for calculating the final user-item matching score:

$$y_{ui} = \mathbf{w}_1^T((\mathbf{q}_u + \mathbf{x}_u) \odot (\mathbf{p}_i + \mathbf{x}_i)) + b_u + b_i + \mu,$$

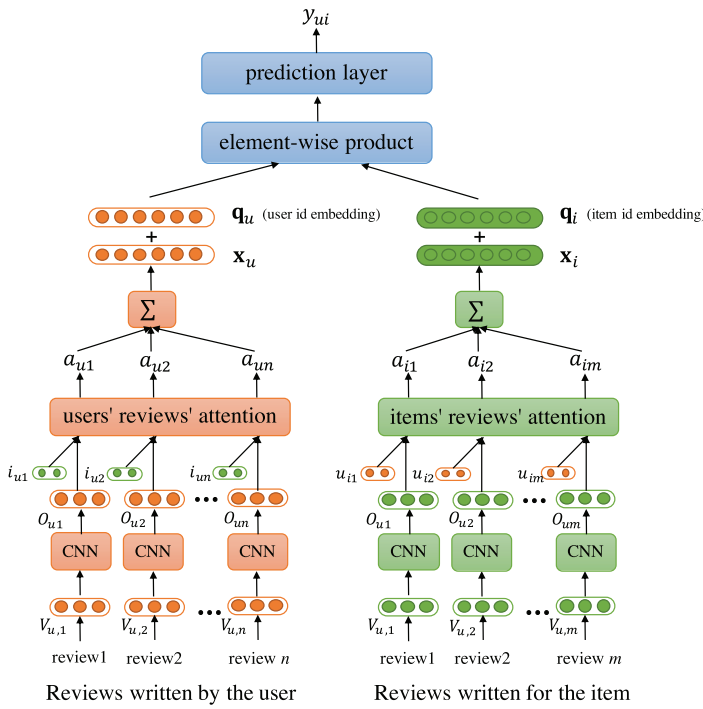

Figure 5.11: Model architecture of NARRE.

where $\odot$ denotes element-wise product, $\mathbf{q}_u$ and $\mathbf{p}_i$ respectively represent the user preferences and item features, $\mathbf{w}_1$ is the weight vector, b_u, b_i and μ are the user bias, item bias, and global bias, respectively.

Learning from Multimedia Content

CNN is known to be an effective feature extractor from multimedia content such as image and video and is widely used in multimedia recommendation. An early work is *Visual Bayesian Personalized Ranking* (VBPR) (He and McAuley, 2016a), which uses a Deep CNN to extract a 4096-dimension feature vector $\mathbf{g}_i$ from each product i's image. As the dimension of $\mathbf{g}_i$ is higher than the dimension of embeddings in collaborative filtering, which is typically in order of hundreds, VBPR projects $\mathbf{g}_i$ into the embedding space with a feature transformation matrix $\mathbf{E}$, that is, $\theta_i = \mathbf{E}\mathbf{g}_i$. It then concatenates θ_i with the item ID

embedding $\mathbf{q}_i$ to form the final item representation. Finally, it interacts the item representation with the user representation with inner product to obtain the prediction score, that is, $\hat{y}_{ui} = \phi_u(u)^T[\mathbf{q}_i, \mathbf{Eg}_i]$. Note that the bias terms are omitted for clarity. The model is learned with the pairwise BPR loss.

It is worth noting that in VBPR, the Deep CNN is pre-trained as a feature extractor, which is not updated during recommendation training. Since Deep CNN is typically trained from a general image corpus like ImageNet, it may not be suitable for the recommendation tasks like clothing recommendation. To address the problem, three solutions are proposed:

- Lei *et al.* (2016) propose the *Comparative Deep Learning* (CDL) method for content-based image recommendation. Instead of fixing the parameters of Deep CNN, it also updates them in training. The objective function is tailored for recommendation, more specifically, a variant of the pairwise BPR loss based on user interactions. As the whole model is trained in an end-to-end fashion, the features extracted by Deep CNN is more suitable for the recommendation task. A recent work employs adversarial training and learns both Deep CNN parameters and recommendation parameters in a similar way (Tang *et al.*, 2020). However, as the number of user-item interactions is typically much larger than the number of labeled instances in an image corpus, this solution may suffer from long training time.

- Ying *et al.* (2018) propose PinSage for image recommendation and Wei *et al.* (2019) propose MMGCN for micro-video recommendation, which share the same idea — refining the extracted image representations on the user-item interaction graph with a graph convolution network. The extracted image representations are treated as the initial features of item nodes, which are propagated on the interaction graph with the graph convolution operation. Since the interaction graph structure contains user preference on items especially the collaborative filtering signals, this method can make the refined visual features more suitable for personalized

recommendation. In the next subsection (Representation Learning from Graph Data), we will introduce details of how it works.

- Different from the above two solutions that learn the whole image representation with a Deep CNN, the *Attentive Collaborative Filtering* (ACF) method (Chen *et al.*, 2017a) cuts an image into 49 (7×7) regions. It employs a pre-trained Deep CNN to extract features from each region and an attention network to learn the weight of it, where the underlying assumption is that different users may be interested in different regions of an image. The 49 regions are finally pooled to obtain the image representation. As the attention network is trained based on user-item interactions, the image representation is adapted for the recommendation task. The framework of ACF is also applied to video recommendation by the authors, where a region is replaced with a frame sampled from the video.

5.1.4 Representation Learning from Graph Data

The above-mentioned representation learning methods have a drawback — learning from the information of a user (an item) separately, while the relations among users and items are ignored. A user-item interaction graph provides rich information on user and item relations, and an item knowledge graph provides rich information on item relations, and thus learning representations from such graphs can overcome the drawback and has the potential to improve the accuracy of recommendation. Several recent works try to leverage this information and developed graph representation learning based recommender systems (Wang *et al.*, 2019a,b,c; Ying *et al.*, 2018). User-item interactions are organized as a bipartite graph, social relations among users are presented in social networks, and item knowledge (*e.g.*, item attributes and relations) is represented in the form of knowledge graph (*aka.* heterogeneous information network). Such a graph structure connects users and items, opening up possibilities to exploit high-order relationships among them, captures meaningful patterns on them (*e.g.*, collaborative filtering, social influence effect, and knowledge-based reasoning), and improves the representation learning of them.

We can categorize existing work into two groups: (1) two-stage learning approach (Gao *et al.*, 2018; Wang *et al.*, 2019c), which first extracts relations as triples or paths, and then learns node representations using the relations, and (2) end-to-end learning approach (Wang *et al.*, 2019a,b) which directly learns representations of nodes where propagation of information is carried out among nodes.

End-to-End Modeling: Neural Graph Collaborative Filtering (NGCF)

Since user-item interactions can be represented in a bipartite graph, Neural Graph Collaborative Filtering (NGCF) (Wang *et al.*, 2019b) revisits collaborative filtering (CF) by defining CF signals as high-order connectivities in the graph. Intuitively, direct connections explicitly characterize users and items — a user's interacted items give supporting evidence on the user's preference, while an item's associated users can be viewed as features of the item. Furthermore, high-order connectivities reflect more complex patterns — the path $u_1 \leftarrow i_1 \leftarrow u_2$ indicates the behavioral similarity between users u_1 and u_2, as both have interacted with item i_1; the longer path $u_1 \leftarrow i_1 \leftarrow u_2 \leftarrow i_2$ suggests u_1's preference on i_2, since his/her similar user u_2 has adopted i_2. Figure 5.12 shows an example of such high-order connectivities, which reflect the user–user and item–item dependencies. NGCF aims to inject such signals into the representations of users and items.

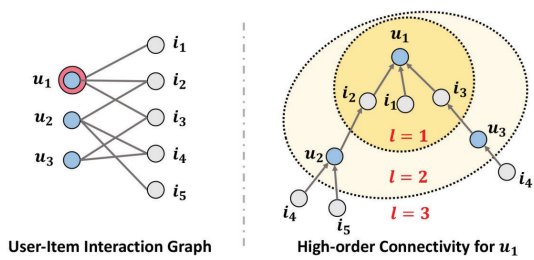

Figure 5.12: An example of high-order connectivity revealed in user-item interaction graph. The figure is taken from Wang *et al.* (2019b).

Inspired by the recent success of graph neural networks (GNNs) (Ying *et al.*, 2018), which are built upon information propagation (or messaging passing) on graphs, NGCF performs embedding propagation on the bipartite user-item interaction graph. Figure 5.13 shows its framework.

Formally, a graph convolutional layer of GNN is composed of two components: (1) message construction, which defines the message being propagated from a neighbor node to the current node, and (2) message aggregation, which aggregates the messages propagated from the neighbor nodes to update the representation of the current node. One widely-used implementation at the l-th layer is as follows:

$$\mathbf{p}_u^{(l)} = \rho(\mathbf{m}_{u\leftarrow u}^{(l)} + \sum_{j\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow j}^{(l)}), \quad \mathbf{m}_{u\leftarrow j}^{(l)} = \alpha_{uj}\mathbf{W}^{(l)}\mathbf{q}_j^{(l-1)}, \tag{5.15}$$

where $\mathbf{p}_u^{(l)}$ denotes the representation of user u after l-layer propagation, $\rho(\cdot)$ is the nonlinear activation function, and $\mathcal{N}_u$ is the neighbor set of u; $\mathbf{m}_{u\leftarrow j}^{(l)}$ is the message being propagated, α_{uj} is the decay factor for propagation on edge (u, j) which is heuristically set as $1/\sqrt{|\mathcal{N}_u||\mathcal{N}_j|}$, and $\mathbf{W}^{(l)}$ is a learnable transformation matrix at the l-th layer. As such, the L-order connectivity is encoded into the updated representation. Thereafter, NGCF concatenates representations from different layers

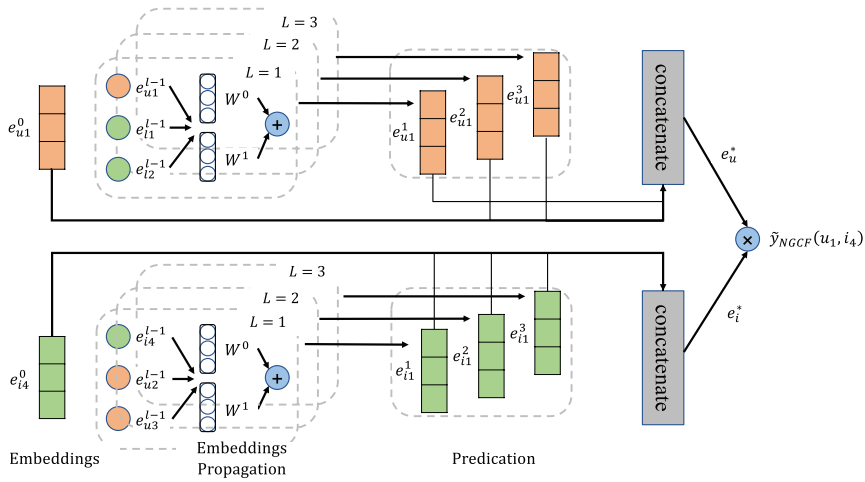

Figure 5.13: Model architecture of NGCF.

which reflect varying contributions to user preferences and performs prediction as:

$$f(u,i) = \mathbf{p}^{*\top}_u \mathbf{q}^*_i, \quad \mathbf{p}^*_u = \mathbf{p}^{(0)}||\cdots||\mathbf{p}^{(L)}, \quad \mathbf{q}^*_i = \mathbf{q}^{(0)}||\cdots||\mathbf{q}^{(L)}, \tag{5.16}$$

where $||$ denotes the concatenation operation.

It is worth mentioning that MF and SVD++ can be viewed as special cases of NGCF with no and one-order propagation layer, respectively. Moreover, one can implement the graph convolutional layer in different ways. For example, SpectralCF (Zheng *et al.*, 2018a) uses a spectral convolution operation to perform information propagation; GC-MC (Berg *et al.*, 2017) combines MLP with Equation (5.15) to capture nonlinear and complex patterns.

While NGCF has demonstrated the strengths of using the interaction graph structure for representation learning, a recent research (He *et al.*, 2020) (LightGCN) shows that many designs in NGCF are redundant, especially the nonlinear feature transformations. The main argument is that in the user-item interaction graph, each node (user or item) is only described by a one-hot ID, which has no semantics besides being an identifier. In such a case, performing multiple layers of nonlinear feature transformation, which is the standard operation of neural networks, will bring no benefit. To validate this argument, they propose a simple model named LightGCN, which retains only the neighborhood aggregation in graph convolution:

$$\begin{aligned} \mathbf{p}^{(l)}_u &= \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{q}^{(l-1)}_i, \\ \mathbf{q}^{(l)}_i &= \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{p}^{(l-1)}_u, \end{aligned} \tag{5.17}$$

where $\mathbf{p}^{(0)}_u$ and $\mathbf{q}^{(0)}_i$ are the model parameters of ID embeddings. We can see that in the light graph convolution, nonlinear feature transformations and self-connections are removed. After L such layers to aggregate high-order neighborhood, LightGCN sums up the representations of all layers as the final representation for a user/item:

$$\mathbf{p}^*_u = \sum_{l=0}^{L} \alpha_l \mathbf{p}^{(l)}_u; \qquad \mathbf{q}^*_i = \sum_{l=0}^{L} \alpha_l \mathbf{q}^{(l)}_i, \tag{5.18}$$

where α_l denotes the importance of the representation of the l-th layer, which is pre-defined. The authors prove in theory that the sum aggregator subsumes the self-connections in graph convolution. Thus, the self-connections can be safely removed from graph convolution. With the same data and evaluation method of NGCF, LightGCN obtains about 15% relatively improvements, which are very significant.

End-to-End Modeling: Knowledge Graph Attention Network (KGAT)

In addition to user-item interactions, more recent works also take into consideration the relations among items in a knowledge graph. Knowledge graph (KG) is a powerful resource which provides rich side information on items (i.e., item attributes and item relations), where nodes are entities and edges represent the relations between them. Usually KG organizes facts or beliefs in a heterogeneous directed graph $\mathcal{G} = \{(h, r, t) \mid h, t \in \mathcal{E}, r \in \mathcal{R}\}$, where the triplet (h, r, t) indicates that there is a relationship r from head entity h to tail entity t. For example, (*Hugh Jackman*, *ActorOf*, *Logan*) states the fact that *Hugh Jackman* is an actor of the movie *Logan*.

The use of the knowledge graph can enhance the learning of item representations and modeling of user-item relationships. In particular, direct connections of an entity — more specifically its associated triples — profile its features. For example, a movie can be characterized by its director, actors, and genres. Moreover, the connections between entities, especially multi-hop paths, stand for complex relationships, and capture complex association patterns. In movie recommendation, for example, users are connected to *Logan* because they like *The Greatest Showman* acted by the same actor *Hugh Jackman*. Obviously, such connections can help to reason about unseen user-item interactions (i.e., a potential recommendation).

Towards the end, Knowledge Graph Attention Network (KGAT) (Wang *et al.*, 2019a) extends NGCF by adaptively extracting information from the neighborhood of high-order connectivity. Different from NGCF where the decay factor α_{ht} on propagation of information on edge (h, t) is fixed, KGAT employs a relational attention mechanism taking into consideration the relation r of edge (h, r, t). Figure 5.14 shows the

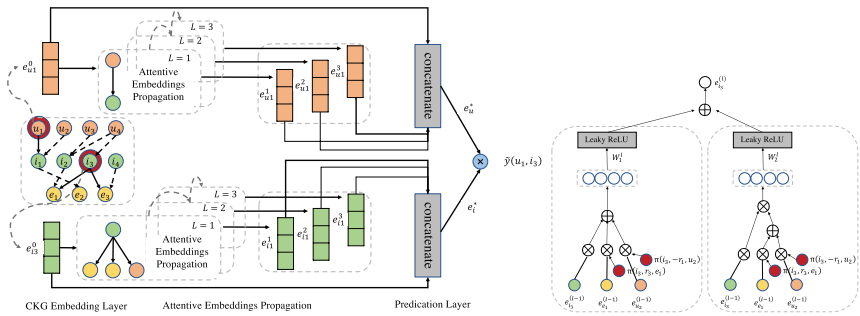

Figure 5.14: Model architecture of KGAT. The left subfigure illustrates the overall model framework, and the right subfigure illustrates the graph convolution operation in KGAT.

framework. The attentive embedding propagation layer is formulated as:

$$\mathbf{p}_h^{(l)} = f_1(\mathbf{p}_h^{(l-1)}, \{\mathbf{m}_{(h,r,t)}^{(l)} \mid (h,r,t) \in \mathcal{N}_h\}), \tag{5.19}$$

$$\mathbf{m}_{(h,r,t)}^{(l)} = f_2(\mathbf{q}_t^{(l-1)}, \alpha_{(h,r,t)}), \quad \alpha_{(h,r,t)} = \frac{\exp g(\mathbf{p}_h, \mathbf{e}_r, \mathbf{q}_t)}{\sum_{(h,r',t')} \exp g(\mathbf{p}_h, \mathbf{e}_{r'}, \mathbf{q}_{t'})},$$

where $f_1(\cdot)$ denotes the message aggregation function, which updates the representation of the head entity h, $f_2(\cdot)$ is the attention message construction function, yielding messages from tail entity t to head entity h, $\alpha_{(h,r,t)}$ is the attentive decay factor derived from the attention network $g(\cdot)$, indicating how much information is propagated and identifying importance of neighbors with regard to relation r. After establishing the representations, KGAT uses the same prediction model as Equation (5.16) to estimate how likely a user would adopt an item.

Two-Stage Modeling: Knowledge Path Recurrent Network (KPRN)

Besides end-to-end modeling to enhance representation learning with high-order connectivity, some works (Gao *et al.*, 2018; Wang *et al.*, 2019c) introduce meta-paths or paths to directly refine the similarities between users and items. In particular, the models first either define meta-path patterns (Gao *et al.*, 2018) or extract qualified paths (Wang *et al.*, 2019c), and then feed them into a supervised learning model to predict the score. Such an approach can be formulated as follows. Given

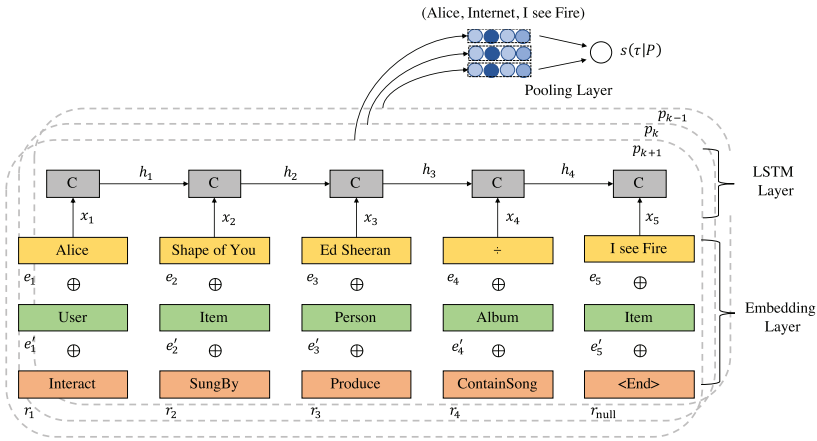

Figure 5.15: Model architecture of KPRN.

the user u, the target item i, and a set of paths $\mathcal{P}(ui) = \{p_1, \ldots, p_K\}$ connecting u and i, their matching score is calculated as $f(u, i \mid \mathcal{P}(u, i))$.

Among them, Knowledge Path Recurrent Network (KPRN) (Wang *et al.*, 2019c) is a representative model, which is shown in Figure 5.15. Given a path among entities, KPRN employs recurrent networks like LSTM to encode the elements on the path to capture the compositional semantics of entities and relations. Thereafter, KPRN exploits a pooling layer to combine multiple path representations into a single vector, and then feeds it into an MLP to obtain the final score for the user-item pair. Formally, the prediction model is defined as:

$$\begin{aligned} \mathbf{x}_k &= \text{LSTM}([\mathbf{p}_{h_1} || \mathbf{e}_{r_1}, \ldots, \mathbf{p}_{h_L} || \mathbf{e}_{r_L}]), \\ f(u, i) &= \text{MLP}\Big(\sum_{k \in \mathcal{P}(u,i)} \mathbf{x}_k \Big), \end{aligned} \tag{5.20}$$

where $p_k = [h_1, r_1, \ldots, h_L, r_L]$ is the k-th path, (h_l, r_l, h_{l+1}) is the l-th triplet in p_k, and L denotes the triplet number. As such, KPRN can employ an LSTM model to leverage the sequential information on the knowledge graph and to enhance the recommender model's explanation ability, revealing why a recommendation is made.

5.2 Matching Based on Matching Function Learning

The matching function outputs a matching score between the user and the item, with user-item interaction information as input, along with possible side information including user attributes, item attributes, contexts, and others. We categorize the methods into two types based on the inputs to the matching function — two-way matching (only user information and item information are provided) and multi-way matching (other side information is also provided).

5.2.1 Two-Way Matching

Traditional latent space models calculate inner product or cosine similarity between user and item to obtain the matching score. However, such a simple way of matching has limitations in model expressiveness. For example, (He *et al.*, 2017c) show that it may incur a large ranking loss due to their inability of maintaining the triangle inequality (Hsieh *et al.*, 2017). Therefore, it is necessary to develop more complicated and expressive matching functions. We categorize existing work along this line into two types: similarity learning methods and metric learning methods.

Similarity Learning Methods

Neural Collaborative Filtering (NCF) (He *et al.*, 2017c) exploits a general neural network framework for collaborative filtering. The idea is to place a multi-layer neural network above user embedding and item embedding to learn their interaction score:

$$f(u, i) = F(\phi_u(u), \phi_i(i)),$$

where F is the interaction neural network to be specified, $\phi_u(u)$ and $\phi_i(i)$ denote the embeddings of user u and item i, respectively. Several instances are proposed under the NCF framework:

- Multi-Layer Perception (MLP). A straightforward way is to stack an MLP above the concatenation of user embedding and item embedding, leveraging the non-linear modeling capability of MLP

to learn the interaction function: $F(\phi_u(u), \phi_i(i)) = MLP([\phi_u(u), \phi_i(i)])$. Although theoretically sound (since MLP can approximate any continuous function in theory), this method does not perform well in practice and underperforms the simple MF model most of the time (for evidence see He *et al.*, 2017c). As revealed in Beutel *et al.* (2018), the key reason is that it is practically difficult for MLP to learn the multiplication operation, which is, however, very important for modeling of interaction in CF (corresponding to the low-rank assumption of user-item interaction matrix). It is important, therefore, to explicitly express the multiplication or similar effect in the matching network.

- Generalized Matrix Factorization (GMF). To generalize MF under the NCF framework, the authors of NCF first calculate element-wise product on user embedding and item embedding, then output the prediction score with a fully connected layer: $F(\phi_u(u), \phi_i(i)) = \sigma(\mathbf{w}^T([\phi_u(u) \odot \phi_i(i)]))$. $\mathbf{w}$ is the trainable weight vector of the layer, which assigns different weights to interactions of different dimensions. Fixing $\mathbf{w}$ as an all-one vector $\mathbf{1}$ can fully recover the MF model. Thus, in principle, GMF can achieve better performance than MF (note that the choices of loss function may affect the results). It is also reasonable to further stack an MLP above the element-wise product in GMF, which is a natural way to address the inability of multiplication learning by MLP. This method appears in Zhang *et al.* (2017) and demonstrates good performance.

- Neural Matrix Factorization (NeuMF). The use of MLP can endow the interaction function with non-linearity. To complement GMF with MLP and combine their strengths, the authors of NCF propose an ensemble model, as illustrated in Figure 5.16. It uses separated embedding sets for GMF and MLP, concatenating the last hidden layers of the two models before projecting to the final matching score. This model has higher representation capability. However, it is also hard to train if training is conducted from the scratch. Empirically, initializing the parameters with pre-trained GMF and MLP leads to better performance, which is highly

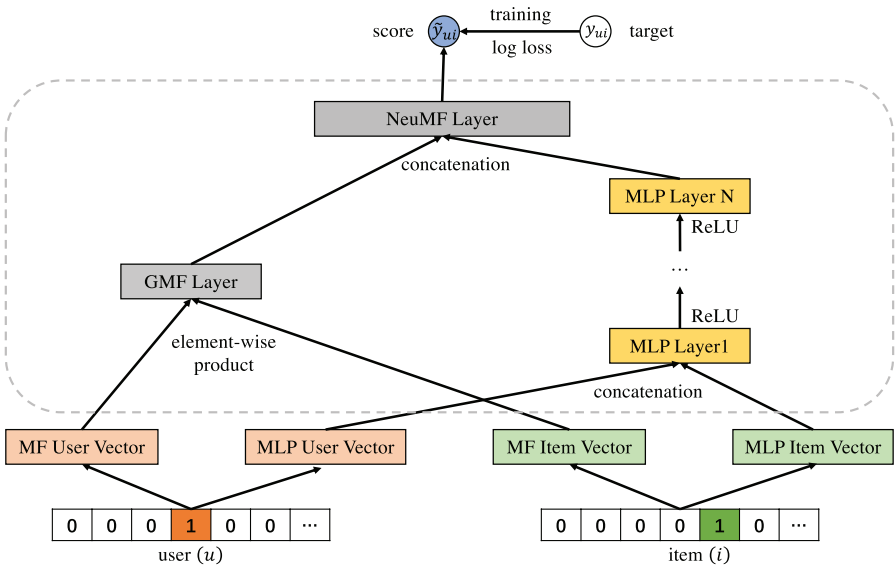

Figure 5.16: Model architecture of NeuMF.

encouraged in practice. Moreover, sharing the embedding layer is also reasonable to reduce the number of parameters, which is subjected to design (Guo *et al.*, 2017).

- Convolutional NCF (ConvNCF) (He *et al.*, 2018b). To explicitly model the correlations (interactions) between embedding dimensions, He *et al.* (2018b) propose to use outer product on user embedding and item embedding, followed by a CNN to aggregate the interactions hierarchically. Figure 5.17 illustrates the model. The output of the outer product is a 2D matrix, where the (k, t)-th entry is $(\mathbf{p}_u \otimes \mathbf{q}_i)_{kt} = p_{uk} \cdot q_{it}$, capturing the interaction between the k-th dimension and the t-th dimension ($\mathbf{p}_u$ and $\mathbf{q}_i$ denote user embedding and item embedding). As the 2D matrix encodes pairwise interactions between embedding dimensions, stacking a CNN above it can capture high-order interactions among embedding dimensions, because each higher layer has a larger receptive field on the matrix. Moreover, CNN has fewer parameters than MLP, which could be difficult to train and is not encouraged.

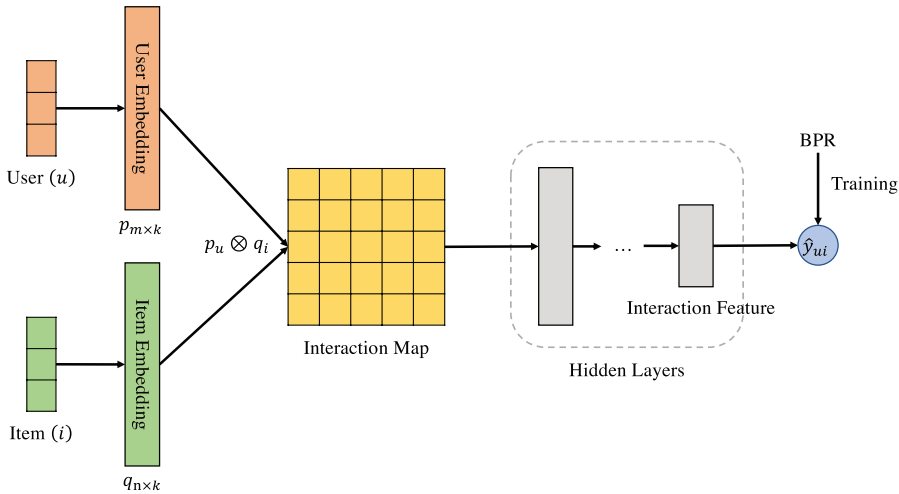

Figure 5.17: Model architecture of ConvNCF.

Metric Learning Methods

Metric learning methods aim to learn and utilize *distance* metrics to quantitatively measure the relationships among data instances. Mathematically, a distance metric needs to satisfy several conditions, and among them the triangle inequality is an important one for generalization (Kulis *et al.*, 2013). An early and representative work that introduces metric learning into recommendation is *Collaborative Metric Learning* (CML) (Hsieh *et al.*, 2017), which points out several limitations of using the inner product for collaborative filtering because of the dissatisfaction of triangle inequality. As a result, it is not able to capture finer-grained user preferences and user–user and item–item relations (because the similarity relation cannot be properly propagated with the inner product). They then formulate a basic metric learning framework for collaborative filtering, which is extended by some later work like (He *et al.*, 2017a; Tay *et al.*, 2018a). Next, we briefly introduce the methods.

- Collaborative Metric Learning (CML) (Hsieh *et al.*, 2017). The user-item metric in CML is defined as the Euclidean distance between user embedding and item embedding:

$$d(u, i) = ||\mathbf{p}_u - \mathbf{q}_i||, \tag{5.21}$$

where $\mathbf{p}_u$ is the user embedding vector, $\mathbf{q}_i$ is the item embedding vector, and $d(u, i)$ is the distance between user u and item i, the smaller, the more similar. An advantage of learning and utilizing the metric is that the similarity among instances can be propagated. For example, if it is known that "$\mathbf{p}_u$ is similar to both $\mathbf{q}_i$ and $\mathbf{q}_j$", then the learned metric will not only make $\mathbf{p}_u$ get closer to $\mathbf{q}_i$ and $\mathbf{q}_j$, but also make $\mathbf{q}_i$ and $\mathbf{q}_j$ themselves closer. This property is fairly useful to capture user–user and item–item relationships from user-item interactions. The intuition is that an item that a user likes is close to the user than the other items that the user does not like. Thus, a margin-based pairwise loss is defined as:

$$L = \sum_{(u,i)\in D^+} \sum_{(u,j)\in D^-} w_{ui}[\delta + d(u,i)^2 - d(u,j)^2]_+, \tag{5.22}$$

where i denotes an item that u likes, j denotes an item that u does not like, $\delta > 0$ is the predefined margin size, $[z]_+ = max(z, 0)$ denotes the hinge loss function, and w_{ui} is the weight of training instance which is predefined. The authors propose several additional constraints to improve the quality of the learned metrics, including a bound of user embedding and item embedding within a unit sphere (i.e., $||p_*|| \leq 1$ and $||q_*|| \leq 1$), and a regularizer to de-correlate the dimensions of the learned metric. We refer the readers to the original paper for more details (Hsieh *et al.*, 2017).

- Translation-based Recommendation (TransRec) (He *et al.*, 2017a). TransRec can be seen as an extension of CML for next-item recommendation, which accounts for user sequential behavior by modeling the third-order interaction among the user, the previously visited item, and the next item to visit (Rendle *et al.*, 2010). The idea is that, the user is represented as a "translation vector", which translates the previous item to the next item, i.e., $\mathbf{q}_j + \mathbf{p}_u \approx \mathbf{q}_i$. The distance metric to realize the translation is:

$$d(\mathbf{q}_j + \mathbf{p}_u, \mathbf{q}_i) = ||\mathbf{q}_j + \mathbf{p}_u - \mathbf{q}_i||, \tag{5.23}$$

where all embedding vectors are re-scaled in the unit length. The authors then estimate the likelihood that the user makes a

transition from item j to item i as:

$$prob(i \mid u, j) = \beta_i - d(\mathbf{q}_j + \mathbf{p}_u, \mathbf{q}_i), \tag{5.24}$$

where β_i is a bias term to capture the item popularity. The TransRec model is learned with the pairwise BPR loss. Compared with CML, TransRec considers the previous item and the transition relation between it and the next item. Recently, Wu *et al.* (2019a) extend TransRec by modeling multiple previous items and high-order interactions with them.

- Latent Relational Metric Learning (LRML) (Tay *et al.*, 2018a). LRML advances TransRec by further learning the relation between the user and the next item. An advantage is that the metric is more geometrically flexible. The metric of LRML is:

$$d(u, i) = ||\mathbf{p}_u + \mathbf{r} - \mathbf{q}_i||, \tag{5.25}$$

where $\mathbf{r}$ is the latent relation vector to be learned. Instead of learning a uniform $\mathbf{r}$ for all user-item pairs, LRML parameterizes it as an attentive sum over external memory vectors. Figure 5.18 shows the model architecture. Let the external memory vectors be $\{\mathbf{m}_t\}_{t=1}^T$, and the keys of the memory vectors be $\{\mathbf{k}_t\}_{t=1}^T$, and both of which are model free parameters to be learned. The relation vector $\mathbf{r}$ is parameterized as:

$$\begin{aligned} \mathbf{r} &= \sum_{t=1}^{M} a_t \mathbf{m}_t, \\ a_t &= softmax((\mathbf{p}_u \odot \mathbf{q}_i)^T \mathbf{k}_t), \end{aligned} \tag{5.26}$$

where a_t is the attentive weight of memory $\mathbf{m}_t$ generated by an attention network that takes the interaction between user embedding and item embedding as input. In this way, the relation vector is user-item interaction aware, which increases the geometrical flexibility of the metric. The model is learned by optimizing the pairwise hinge loss which is the same as that in CML.

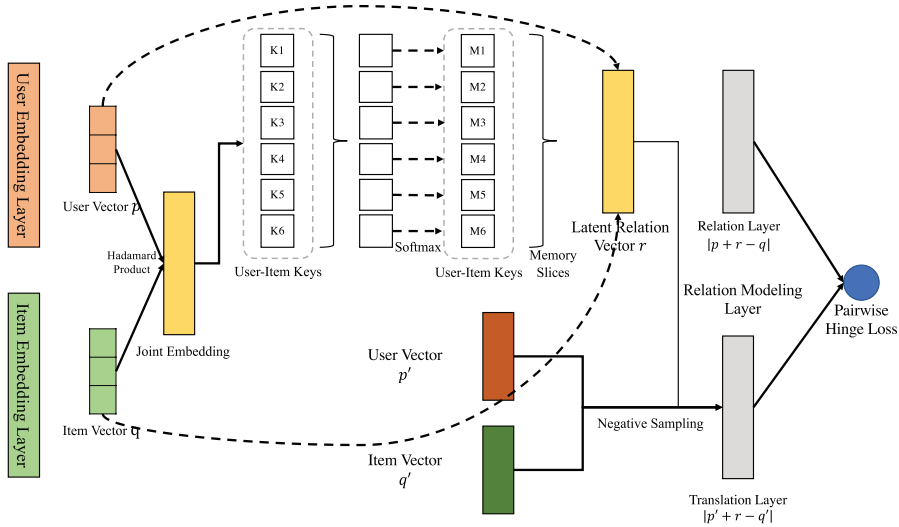

Figure 5.18: Model architecture of LRML.

5.2.2 Multi-Way Matching

Methods of multi-way matching are generic feature-based methods, just like the FM model which takes features as input and incorporates feature interactions in the matching function. The methods allow the utilization of any kind of side information and context. However, they may have higher complexity compared to two-way matching and representation learning-based methods. As such, they are more often used in the ranking stage rather than in the candidate retrieval stage, such as Click-Through Rate (CTR) prediction.

Feature interaction modeling aims to capture cross-feature effects, i.e., signals from multiple features. For example, users of age 20–25 (feature 1) and gender female (feature 2) are more likely to purchase iPhones of pink color (feature 3). A naive solution for capturing such effects is to manually construct cross features, feeding them into a linear model that can learn and memorize the importance of the cross features (Cheng *et al.*, 2016). The issue is that it can only memorize the seen cross features (in training data), and cannot generalize to unseen cross features. Moreover, the number of cross features increases polynomially with the order of crossing. Thus it requires domain knowledge to

select useful cross features instead of using all cross features. Therefore, we need more effective and efficient techniques for feature interaction modeling.

We categorize existing work into three types based on how feature interactions are modeled: implicit interaction modeling, explicit interaction modeling, and the combination of implicit and explicit interaction modeling.

Implicit Interaction Modeling

At Recsys 2016, the YouTube team presented a deep neural network model for YouTube recommendation (Covington *et al.*, 2016). It projects each categorical feature as an embedding vector (for sequence features like watched items, it performs average pooling to obtain a sequence embedding vector). It then concatenates all embeddings, feeding the concatenated vector into a three-layer MLP to obtain the final prediction. The MLP is expected to learn the interactions among feature embeddings, because of its strong representation capability in approximating any continuous function. However, the feature interaction modeling is rather an implicit process since the interactions are encoded in the hidden units of MLP, and there is no way to identify which interactions are important for a prediction after the model is trained. Moreover, it is practically difficult for MLP to learn multiplication effect (Beutel *et al.*, 2018), which is important to capture cross features.

This simple architecture becomes a pioneer work of utilizing deep neural networks for recommendation, and many later works make extensions on it. For example, Wide&Deep (Cheng *et al.*, 2016) ensembles the deep model (i.e., the deep part) with a linear regression model (i.e., the wide part), which contains sophisticated features including manually constructed cross features. Deep Crossing (Shan *et al.*, 2016) deepens the MLP to ten layers with residual connections between layers. As will be introduced next, many ensemble models like DeepFM (Guo *et al.*, 2017) and xDeepFM (Lian *et al.*, 2018) integrate the deep architecture into a shallow architecture to augment implicit interaction modeling with explicit interaction modeling.

Explicit Interaction Modeling

FM is a traditional model that performs second-order interaction modeling (we have introduced the model in Subsection 2.4.3). Specifically, it projects each non-zero feature x_i into an embedding $\mathbf{v}_i$, performs inner product on each pair of non-zero feature embeddings, and sums over all inner products (the first-order linear regression part is omitted for clarity). Due to its effectiveness, FM is extended under the neural network framework for explicit interaction modeling.

Figure 5.19 shows the Neural Factorization Machine (NFM) model (He and Chua, 2017). The idea is to replace the inner product with the element-wise product, which outputs a vector rather than a scalar, and then stacks an MLP above the sum of element-wise products. The core operation in NFM is called *Bi-Interaction Pooling*, defined as:

$$f_{BI}(\mathcal{V}_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i \mathbf{v}_i \odot x_j \mathbf{v}_j, \tag{5.27}$$

where x_i denotes the value of feature i, $\mathcal{V}_x$ denotes the set of embeddings of non-zero features, and n denotes the number of non-zero features. The vector obtained by Bi-Interaction pooling encodes second-order interactions. By stacking an MLP above it, the model has the ability to learn high-order feature interactions.

One issue with FM and NFM is that all second-order interactions are considered equally important, and they contribute evenly to the

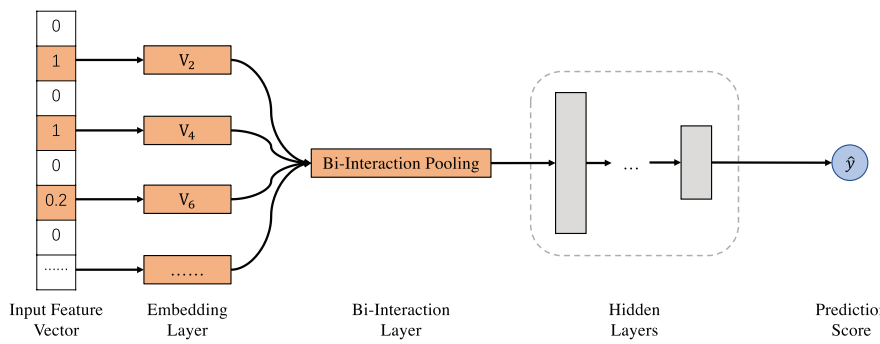

Figure 5.19: Model architecture of NFM.

prediction. To address this issue, Attentional Factorization Machine (AFM) (Xiao *et al.*, 2017) is proposed to differentiate the importance of interactions with an attention network. Figure 5.20 shows the architecture of AFM. The input and embedding layers are the same as those in the standard FM. The pair-wise interaction layer performs the element-wise product on each pair of feature embeddings to obtain the interaction vectors; this step is equivalent to those of FM and NFM. The attention network takes each interaction vector $\mathbf{v}_i \odot \mathbf{v}_j$ as input and outputs the importance weight a_{ij} with a two-layer MLP. Then, the model uses the importance weight to re-weight each interaction vector and sum up all interaction vectors to obtain the final score. The equation of AFM is:

$$\begin{aligned} \hat{y}_{AFM}(\mathbf{x}) &= \mathbf{p}^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j, \\ \text{where} \quad a_{ij} &= softmax(\mathbf{h}^T MLP(\mathbf{v}_i \odot \mathbf{v}_j)). \end{aligned} \tag{5.28}$$

The attention weight a_{ij} can be used to interpret the importance of each second-order interaction for the prediction. It is straightforward to further leverage the strengths of NFM in high-order interaction modeling and AFM in second-order interaction modeling, by appending an MLP onto the attention-based pooling layer. This naturally leads to Deep AFM, which has better representation ability and may yield better performance. A recent work (Tao *et al.*, 2019) proposes a high-order attentive FM (HoAFM), which has linear complexity in order size.

Orthogonal to the work of FM, Lian *et al.* (2018) propose Compressed Interaction Network (CIN), which explicitly models high-order feature interactions in a recursive way. Let the embeddings of non-zero input features be a matrix $\mathbf{V}^0 \in \mathbf{R}^{n \times D}$, where n is the number of nonzero features and D is the embedding size; the i-th row in $\mathbf{V}^0$ is the embedding vector of the i-th nonzero feature: $\mathbf{V}^0_{i,*} = \mathbf{v}_i$. Let the output of the k-th layer in CIN be a matrix $\mathbf{X}^k \in \mathbf{R}^{H_k \times D}$, where H_k denotes the number of embedding vectors in the k-th layer and is an architecture choice to specify (note that $H_0 = n$). The recursive definition of CIN

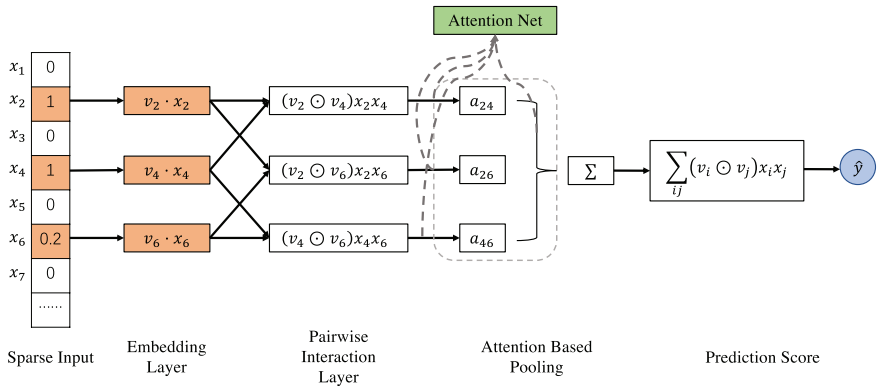

Figure 5.20: Model architecture of AFM.

that can capture high-order feature interaction is defined as:

$$\mathbf{V}_{h,*}^{k} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{n} \mathbf{W}_{ij}^{k,h} (\mathbf{V}_{i,*}^{k-1} \odot \mathbf{V}_{j,*}^{0}), \tag{5.29}$$

where $1 \leq h \leq H_k, \mathbf{W}^{k,h} \in \mathbf{R}^{H_{k-1} \times n}$ is the parameter matrix for the h-th feature vector. As $\mathbf{V}^k$ is derived via the interaction between $\mathbf{V}^{k-1}$ and $\mathbf{V}^0$, the order of feature interactions increases with the layer depth. Assuming that the model stacks K such layers, the final prediction is based on the output matrix $\mathbf{X}^k$ of all K layers, which unifies the feature interactions up to K orders. Note that the time complexity of high-order interaction modeling increases linearly with the number of orders, which is smaller than that of high-order FM. However, CIN introduces more parameters to train — for layer k, it has H_k trainable weight matrices of size $H_{k-1} \times n$, which amounts to a parameter tensor of size $H_k \times H_{k-1} \times n$.

Combination of Explicit and Implicit Interaction Modeling

As implicit interaction modeling and explicit interaction modeling work in different ways, integrating them into a unified framework has the potential to improve the performance. In the recent literature, the reported best performances are obtained by hybrid models that combine

multiple interaction models (Guo *et al.*, 2017; Lian *et al.*, 2018). We briefly review the ensemble methods in this subsection.

Wide&Deep (Cheng *et al.*, 2016) ensembles a linear regression model which utilizes manually constructed cross features (the wide part) and an MLP model which implicitly uses the interactions of features (the deep part). The wide part is to memorize the seen cross features, and the deep part is to generalize to unseen cross features. Let $\mathbf{x}$ be the original input features and $\phi(\mathbf{x})$ be the constructed cross features. Then the model for predicting the click-through rate is:

$$p(click \mid \mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T \mathbf{a}^{(L)} + b), \tag{5.30}$$

where $\sigma(\cdot)$ is the sigmoid function to output a probability value, $\mathbf{w}_{wide}$ denotes the weights of the wide part, $\mathbf{w}_{deep}$ denotes the weights of the deep part, $\mathbf{a}^{(L)}$ denotes the last hidden layer of the MLP model, and b is the bias term.

The work of Wide&Deep inspired many later works to employ similar ensembles, but on different models. For example, DeepFM (Guo *et al.*, 2017) ensembles FM and MLP via: $\hat{y}_{DeepFM} = \sigma(\hat{y}_{FM} + \hat{y}_{MLP})$, where $\hat{y}_{FM}$ and $\hat{y}_{MLP}$ denote the predictions of FM and MLP, respectively. The FM model learns second-order feature interactions explicitly, and the MLP model learns high-order feature interactions implicitly. In addition, DeepFM shares the embedding layer of FM and MLP to reduce model parameters. xDeepFM (Lian *et al.*, 2018) further ensembles DeepFM with CIN, which explicitly models high-order feature interactions. There are other ensemble models. We do not describe them here due to space limitations. The general observation is that combining models that account for different types of interactions usually yields better performances.

5.3 Further Reading

Recommendation remains to be an important and hot topic in information retrieval and data mining. New techniques are constantly developed, and new methods are evolving fast. Here we give more references for further reading.

5.3.1 Papers

With regard to learning representations from user sequential interactions, some recent work (Sun *et al.*, 2019; Yuan *et al.*, 2020) argues that user behaviors may not be strictly formalized as sequences of interactions. That is, a sequence of interactions does not necessarily encode strong semantics as a sentence of words, and a recorded sequence only reflects the user's one choice and other choices may also be possible. As a result, the "left-to-right" training paradigm, such as RNN, may not be optimal because the future interactions are ignored in the prediction of interactions. Such information is also indicative of user's preference and should be leveraged. The key question in utilization of future information is how to avoid information leakage. To address this challenge, (Sun *et al.*, 2019; Yuan *et al.*, 2020) employ the fill-in-the-blank training paradigm inspired by BERT, which randomly masks some interactions in the encoder with the aim of predicting the masked interactions either by the encoder itself (Sun *et al.*, 2019) or by an additional decoder (Yuan *et al.*, 2020).

For learning representations from multi-modal content, some recent work exploits an interaction graph to propagate multimedia features on the graph with graph convolution network (Wei *et al.*, 2019; Ying *et al.*, 2018). In this way, the multimedia features are smoothed and become more useful for recommendation. For example, the multi-modal GCN (MMGCN) (Wei *et al.*, 2019) constructs a modality-aware GCN on the user-item graph with features of each modality (visual, textual, and acoustic), and fuses the output of each modality to get the final representation of a micro-video. With regard to representation learning on graph data, besides the user-item graph and knowledge graph as we have described, social network (Wu *et al.*, 2019b) and session graph (Qiu *et al.*, 2019; Wu *et al.*, 2019c) are also used for learning of GCN. As a graph provides a formal way of describing different types of entities and their relations, matching based on a heterogeneous graph is a promising solution for recommender systems in different applications. As for learning dynamic representations of users with respect to different items, attention networks are designed to learn user's specific preferences

to different aspects of items by leveraging reviews and images (Cheng *et al.*, 2018; Liu *et al.*, 2019b).

Due to the diversity of recommendation scenarios in practice, researchers develop neural recommender models from different perspectives. For example, (Gao *et al.*, 2019b) model multiple cascading user behaviors like click, add-to-cart, and purchase, in a sequential multi-task framework. Li *et al.* (2019) develop a capsule network to recognize sentiment from user reviews for recommendation. Xin *et al.* (2019b) takes into account relations of multiple items (e.g., same category, shared attribute, etc.) for item-based CF. Gao *et al.* (2019a) develop a privacy-preserving method for cross-domain recommendation by transferring user embeddings rather than raw behavior data. Pan *et al.* (2019) tailor the learning of embedding parameters for item cold-start recommendation via meta-learning.

All the above-discussed neural network models are offline recommendation methods, which exploit the offline historical data to estimate user preference. Another thriving area is online recommendation, for which the bandit-based methods are prevalent (Li *et al.*, 2010; Wu *et al.*, 2016a; Zhang *et al.*, 2020). They aim to pursue the exploitation-exploration trade-off when interacting with users in online recommendations. Two common types of bandit methods are Upper Confidence Bound (UCB)-based and Thompson Sampling (TS)-based, and both methods have pros and cons. Besides interacting with users with item recommendations, recent work (Zhang *et al.*, 2020) considers asking attribute preference, which can find the most related items more effectively. Neural networks can serve as the exploitation component for the bandit-based methods, and more investigations remain to be done towards combining offline deep models with online exploration strategies.

5.3.2 Benchmark Datasets

There are a number of benchmark datasets available for training and testing recommender models in different scenarios. For instance, the MovieLens collection,[1] Amazon product collection[2] (He and McAuley, 2016),

[1] https://grouplens.org/datasets/movielens.
[2] http://jmcauley.ucsd.edu/data/amazon/.

and Gowalla[3] (Liang *et al.*, 2016) are benchmark datasets that consist of user-item interaction data. Yelp,[4] Ciao,[5] and Epinions[6] (Richardson *et al.*, 2003) are datasets that additionally include social relations among users which are useful for social recommendation. Yoochoose[7] and Diginetica[8] contain streams of user clicks in e-commerce, and thus are suitable for session-based (sequential) recommendation. Criteo,[9] Avazu,[10] and Frappe[11] (Baltrunas *et al.*, 2015) are comprised of context information of interactions, and are widely used in CTR prediction and feature-based recommendation. Moreover, as Amazon, Yelp, and TripAdvisor[12] (Wang *et al.*, 2018c) provide rich reviews and comments on items, they are widely utilized in review-based recommender models. Besides, there are several datasets presenting a knowledge graph for recommendation, such as KB4Rec[13] (Zhao *et al.*, 2019) and KGAT[14] (Wang *et al.*, 2019a).

5.3.3 Open Source Packages

Several open-source packages or libraries for recommendation are publicly available, with the aim of facilitating related research. Microsoft Recommenders[15] offers tens of example models for building recommendation systems. NeuRec[16] is an open-source library that includes a large number of state-of-the-art recommender models, ranging from collaborative filtering and social recommendation to sequential recommendation. It is worthwhile highlighting that NeuRec is a modular framework in which a model can be built upon reusable modules with

[3] https://snap.stanford.edu/data/loc-gowalla.html.
[4] https://www.yelp.com/dataset.
[5] http://www.ciao.co.uk.
[6] https://snap.stanford.edu/data/soc-Epinions1.html.
[7] https://2015.recsyschallenge.com/.
[8] https://competitions.codalab.org/competitions/11161.
[9] http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset.
[10] https://www.kaggle.com/c/avazu-ctr-prediction/data.
[11] http://baltrunas.info/research-menu/frappe.
[12] https://github.com/xiangwang1223/tree_enhanced_embedding_model.
[13] https://github.com/RUCDM/KB4Rec.
[14] https://github.com/xiangwang1223/knowledge_graph_attention_network.
[15] https://github.com/microsoft/recommenders.
[16] https://github.com/NExTplusplus/NeuRec.

standard interfaces. Thus it allows users to build their own models easily. Similarly, OpenRec[17] is an open-source project that contains several recommendation methods.

[17] https://github.com/ylongqi/openrec.

6

Conclusion and Future Directions

6.1 Summary of the Survey

How to bridge the semantic gap between two matching entities is the most fundamental and challenging issue in search and recommendation. In search, the searchers and the authors of documents may use different expressions to represent the same meanings, resulting in the most undesirable outcomes in which relevant documents exist but cannot be found. In recommendation, the users and the items belong to different types of entities and are represented by different superficial features, making it difficult to conduct matching between the features and thus offer satisfactory recommendations on items to users. To bridge the semantic gap, researchers in both search and recommendation have proposed to construct and utilize matching models with machine learning techniques.

In recent years, deep learning has been applied to search and recommendation, and great success has been achieved. In this survey, we have first introduced a unified view on matching in search and recommendation. Under this view, we have then categorized the learning solutions to query-document matching in search and user-item matching in recommendation into two types: methods of representation learning

and methods of matching function learning. After that, representative traditional matching methods, as well as deep matching methods, have been explained with details. Experimental results, benchmarks, and software packages have also been introduced.

The unified view of matching provides a new means to compare and analyze the machine learning approaches, particularly deep learning approaches, developed for search and recommendation. Although existing matching models for search and for recommendation are developed for different purposes within different communities (e.g., SIGIR and RecSys), they bear similar design principles and model properties. This survey can be beneficial for people in communities with its unified view. In fact, the boundary between search and recommendation becomes blurry, and there emerges a trend to unify the two paradigms (Schedl *et al.*, 2018; Zhang *et al.*, 2018). The unified view provides a new angle to devise novel models for search and recommendation.

One can see that deep learning for matching has made and is making significant progress in search and recommendation. One can also foresee that it has the potential to make impact on similar problems in other fields, including online advertising, question answering, image annotation, and drug design.

6.2 Matching in Other Tasks

Semantic matching is a fundamental issue in other tasks beyond search and recommendation. Since matching is conducted between two sets of objects, it can be categorized as text matching and entity matching. In text matching, there exists an order between the elements within each object (e.g., words in a sentence). Query-document matching is a typical example of text matching. In entity matching, there is no order exits between the objects. User-item matching in recommendation is an example of entity matching. Other matching tasks have also been studied. We list some of them here.

Paraphrase detection Determining whether two sentences are with the same meaning is an important topic of semantic matching in

natural language processing. The matching is conducted at the semantic level, and the learned matching function is symmetric.

Community QA Given a question, the goal is to find questions with the same meaning from the knowledge-base in community QA. The task is similar to paraphrase detection, while the two sentences are questions. The matching between two questions is conducted at the semantic level.

Text entailment Text entailment refers to the problem of determining implication or none-implication relation between two statements. Though similar, entailment is different from paraphrase detection in that it focuses on determining the logical relation between two texts. The matching should also be conducted at the semantic level, and the matching function is not symmetric.

Retrieval-based dialogue One key issue in retrieval-based dialogue is to find the most suitable response given utterances in the context of the conversation. The response is usually a sentence while the utterance can be, for example, one single utterance or all utterances in the context (in multi-turn dialog). It is obvious that the matching is conducted between texts at the semantic level.

Online advertising In search ads, how to match a user's search query to advertisers' keywords greatly affects the probability that the user will see and click the ads. In contextual ads, matching is conducted between the keywords and the contents of webpages. In both cases, semantic matching is helpful in choosing right ads and constructing a right order by which the ads are displayed.

6.3 Open Questions and Future Directions

There are many open questions with regard to deep matching for search and recommendation. Here, we only list some of them.

1. Lack of training data (i.e., supervised learning data) is still one of the key challenges. In contrast, deep matching models need a large

amount of data to train. How to leverage unsupervised learning, weakly-supervised learning, semi-supervised learning, and distant supervised learning techniques to deal with the problem is an important question.

2. A large fraction of deep matching models are trained with click data. Existing studies show that directly using click data as training signals often yields suboptimal results. In learning to rank, the counterfactual inference framework is proposed to derive unbiased learning to rank models (Joachims *et al.*, 2017). How to overcome the bias problem in deep matching is an exciting future direction.

3. The learning of existing deep matching models is purely data-driven. Sometimes, rich prior knowledge does exist (e.g., domain knowledge, knowledge-base, matching rules), and the use of it should be helpful in improving the performances of matching. How to integrate prior knowledge into matching models is an important direction to explore.

4. Matching models are usually learned with one single objective, i.e., "similarity". There may need to exploit multiple objectives in learning (e.g., induction ability, fairness) according to applications. How to add other criteria into the learning of matching models is another important issue to investigate.

5. To a large extent, current deep matching models are black boxes. In real search and recommender systems, however, it is often required that the matching models not only achieve high accuracy, but also give intuitive explanations of the results. Such explainability is helpful to improve the transparency, persuasiveness, and trustworthiness of the system. How to create the explanation ability in deep matching models is still an open question.

6. Most deep matching models only learn correlations from the data. However, correlation is not causality, and it falls short in revealing the reasons behind the data (e.g., the reasons that a user prefers an item over another one). To enhance a matching model with causal reasoning ability, we need to introduce the mechanisms of

intervention and counterfactual reasoning into the model (Pearl, 2019). Moreover, the collected data is usually biased by many factors, like the position bias, exposure bias, and so on. It is an emerging direction to develop causal methods for matching, which are robust to the various data bias and able to reveal the reasons behind the data.

7. In search and recommender systems, the processes of matching and ranking are usually separated: first matching and then ranking. Therefore, the results of matching are naturally used as features of ranking. However, the separation of ranking and matching may not be necessary sometimes. One natural question is whether it is possible to build an end-to-end system in which the matching and ranking models are jointly learned.

8. Search and recommender systems are becoming more and more interactive, which can help users to find relevant or interesting information in an exploratory way. For example, some search engines let the users to refine the queries after checking the initial results. Similarly, some recommendation systems recommend items based on what the users have chosen, or through asking users what kind of item attributes they prefer (Lei *et al.*, 2020). Therefore, how to structure the user-system interactions and conduct query-document (or user-item) matching in the interactive and conversational scenarios is an important and interesting research topic.

Acknowledgements

We thank the editors and the three anonymous reviewers for their valuable comments to improve the manuscript. We thank Dr. Wang Xiang and Dr. Yuan Fajie for providing materials for the writing of the survey. The work is supported by the National Natural Science Foundation of China (61872338, 61972372, U19A207, 61832017), Beijing Academy of Artificial Intelligence (BAAI2019ZD0305), and Beijing Outstanding Young Scientist Program (BJJWZYJH012019100020098).

References

Adomavicius, G. and A. Tuzhilin (2005). "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". *IEEE Transactions on Knowledge and Data Engineering*. 17(6): 734–749.

Ai, Q., K. Bi, J. Guo, and W. B. Croft (2018). "Learning a deep listwise context model for ranking refinement". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: Association for Computing Machinery. 135–144.

Andrew, G., R. Arora, J. Bilmes, and K. Livescu (2013). "Deep canonical correlation analysis". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning – Volume 28. ICML'13*. Atlanta, GA, USA: JMLR.org. III-1247–III-1255. URL: http://dl.acm.org/citation.cfm?id=3042817.3043076.

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). "Layer normalization". *CoRR*. abs/1607.06450. arXiv: 1607.06450.

Bahdanau, D., K. Cho, and Y. Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*. URL: http://arxiv.org/abs/1409.0473.

Bai, B., J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger (2009). "Supervised semantic indexing". In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management. CIKM '09*. Hong Kong, China: ACM. 187–196.

Bai, B., J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger (2010). "Learning to rank with (a lot of) word features". *Information Retrieval*. 13(3): 291–314.

Bai, T., J.-R. Wen, J. Zhang, and W. X. Zhao (2017). "A neural collaborative filtering model with interaction-based neighborhood". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17*. Singapore, Singapore: ACM. 1979–1982.

Balaneshin-Kordan, S. and A. Kotov (2018). "Deep neural architecture for multi-modal retrieval based on joint embedding space for text and images". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: ACM. 28–36.

Baltrunas, L., K. Church, A. Karatzoglou, and N. Oliver (2015). "Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild". *CoRR*. abs/1505.03014. arXiv: 1505.03014.

Bast, H., B. Björn, and E. Haussmann (2016). "Semantic search on text and knowledge bases". *Foundations and Trends in Information Retrieval*. 10(2–3): 119–271.

Batmaz, Z., A. Yurekli, A. Bilge, and C. Kaleli (2019). "A review on deep learning for recommender systems: Challenges and remedies". *Artificial Intelligence Review*. 52(1): 1–37.

Belkin, N. J. and W. B. Croft (1992). "Information filtering and information retrieval: Two sides of the same coin?" *Communications of the ACM*. 35(12): 29–38.

Bello, I., S. Kulkarni, S. Jain, C. Boutilier, E. H. Chi, E. Eban, X. Luo, A. Mackey, and O. Meshi (2018). "Seq2Slate: Re-ranking and slate optimization with RNNs". In: *Proceedings of the Workshop on Negative Dependence in Machine Learning at the 36th International Conference on Machine Learning*. Long Beach, CA. PMLR 97, 2019.

Bendersky, M., W. B. Croft, and D. A. Smith (2011). "Joint annotation of search queries". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies – Volume 1. HLT '11*. Portland, OR, USA: Association for Computational Linguistics. 102–111. URL: http://dl.acm.org/citation.cfm?id=2002472.2002486.

Berg, R. van den, T. N. Kipf, and M. Welling (2017). "Graph convolutional matrix completion". *CoRR*. abs/1706.02263. arXiv: 1706.02263.

Berger, A. and J. Lafferty (1999). "Information retrieval as statistical translation". In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '99*. Berkeley, CA, USA: ACM. 222–229.

Bergsma, S. and Q. I. Wang (2007). "Learning noun phrase query segmentation". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics. 819–826. URL: https://www.aclweb.org/anthology/D07-1086.

Beutel, A., P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi (2018). "Latent cross: Making use of context in recurrent recommender systems". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: ACM. 46–54.

Bowman, S. R., G. Angeli, C. Potts, and C. D. Manning (2015). "A large annotated corpus for learning natural language inference". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics. 632–642. URL: https://www.aclweb.org/anthology/D15-1075.

Brill, E. and R. C. Moore (2000). "An improved error model for noisy channel spelling correction". In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. ACL '00*. Hong Kong: Association for Computational Linguistics. 286–293.

Burges, C. J. (2010). "From RankNet to LambdaRank to LambdaMART: An overview". *Technical report*. MSR-TR-2010-82. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/.

Cao, Y., J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon (2006). "Adapting ranking SVM to document retrieval". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '06*. Seattle, Washington, DC, USA: ACM. 186–193.

Chen, J., H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua (2017a). "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: ACM. 335–344.

Chen, Q., X. Zhu, Z.-H. Ling, S. Wei, H. Jiang, and D. Inkpen (2017b). "Enhanced LSTM for natural language inference". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics. 1657–1668. URL: https://www.aclweb.org/anthology/P17-1152.

Chen, C., M. Zhang, Y. Liu, and S. Ma (2018a). "Neural attentional rating regression with review-level explanations". In: *Proceedings of the 2018 World Wide Web Conference. WWW '18*. Lyon, France. 1583–1592.

Chen, H., F. X. Han, D. Niu, D. Liu, K. Lai, C. Wu, and Y. Xu (2018b). "MIX: Multi-channel information crossing for text matching". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18*. London, UK: ACM. 110–119.

Cheng, H.-T., L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah (2016). "Wide & deep learning for recommender systems". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. DLRS 2016*. Boston, MA, USA: ACM. 7–10.

Cheng, Z., Y. Ding, X. He, L. Zhu, X. Song, and M. S. Kankanhalli (2018). "A3NCF: An adaptive aspect attention model for rating prediction". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. 3748–3754.

Cohen, D., L. Yang, and W. B. Croft (2018). "WikiPassageQA: A benchmark collection for research on non-factoid answer passage retrieval". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: ACM. 1165–1168.

Costa, A. and F. Roda (2011). "Recommender systems by means of information retrieval". In: *Proceedings of the International Conference on Web Intelligence, Mining and Semantics. WIMS '11*. Sogndal, Norway: ACM. 57:1–57:5.

Covington, P., J. Adams, and E. Sargin (2016). "Deep neural networks for youtube recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. 191–198.

Croft, W. B., D. Metzler, and T. Strohman (2009). *Search Engines: Information Retrieval in Practice*. 1st Edn. USA: Addison-Wesley Publishing Company. I–XXV, 1–524.

Dai, Z., C. Xiong, J. Callan, and Z. Liu (2018). "Convolutional neural networks for soft-matching N-grams in ad-hoc search". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: ACM. 126–134.

Dehghani, M., H. Zamani, A. Severyn, J. Kamps, and W. B. Croft (2017). "Neural ranking models with weak supervision". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: Association for Computing Machinery. 65–74.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). "BERT: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics. 4171–4186. URL: https://www.aclweb.org/anthology/N19-1423.

Dolan, B. and C. Brockett (2005). "Automatically constructing a corpus of sentential paraphrases". In: *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing. URL: https://www.microsoft.com/en-us/research/publication/automatically-constructing-a-corpus-of-sentential-paraphrases/.

Eisenschtat, A. and L. Wolf (2017). "Linking image and text with 2-way nets". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1855–1865.

Eksombatchai, C., P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec (2018). "Pixie: A system for recommending 3+ Billion items to 200+ Million users in real-time". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France*. 1775–1784.

Elkahky, A. M., Y. Song, and X. He (2015). "A multi-view deep learning approach for cross domain user modeling in recommendation systems". In: *Proceedings of the 24th International Conference on World Wide Web*. Republic and Canton of Geneva, CHE. 278–288.

Fan, Y., J. Guo, Y. Lan, J. Xu, C. Zhai, and X. Cheng (2018). "Modeling diverse relevance patterns in ad-hoc retrieval". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: ACM. 375–384.

Fan, W., Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin (2019). "Graph neural networks for social recommendation". In: *The World Wide Web Conference. WWW '19*. San Francisco, CA, USA: Association for Computing Machinery. 417–426.

Gao, J., J.-Y. Nie, G. Wu, and G. Cao (2004). "Dependence language model for information retrieval". In: *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '04*. Sheffield, UK: ACM. 170–177.

Gao, L., H. Yang, J. Wu, C. Zhou, W. Lu, and Y. Hu (2018). "Recommendation with multi-source heterogeneous information". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization. 3378–3384.

Gao, C., X. Chen, F. Feng, K. Zhao, X. He, Y. Li, and D. Jin (2019a). "Cross-domain recommendation without sharing user-relevant data". In: *The World Wide Web Conference. WWW'19*. New York, NY, USA: Association for Computing Machinery. 491–502.

Gao, C., X. He, D. Gan, X. Chen, F. Feng, Y. Li, and T.-S. Chua (2019b). "Neural multi-task recommendation from multi-behavior data". In: *Proceedings of IEEE 35th International Conference on Data Engineering (ICDE),* Macao, China. 1554–1557.

Garcia-Molina, H., G. Koutrika, and A. Parameswaran (2011). "Information seeking: Convergence of search, recommendations, and advertising". *Communications of the ACM*. 54(11): 121–130.

Gong, Y., H. Luo, and J. Zhang (2018). "Natural language inference over interaction space". In: *6th International Conference on Learning Representations, ICLR 2018*.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Graves, A., S. Fernández, and J. Schmidhuber (2007). "Multi-dimensional recurrent neural networks". In: *Artificial Neural Networks – ICANN 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg. 549–558.

Guo, J., G. Xu, H. Li, and X. Cheng (2008). "A unified and discriminative model for query refinement". In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '08*. Singapore, Singapore: ACM. 379–386.

Guo, J., Y. Fan, Q. Ai, and W. B. Croft (2016). "A deep relevance matching model for ad-hoc retrieval". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, IN, USA: ACM. 55–64.

Guo, H., R. Tang, Y. Ye, Z. Li, and X. He (2017). "DeepFM: A factorization-machine based neural network for CTR prediction". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI'17*. Melbourne, Australia: AAAI Press. 1725–1731.

Guo, Y., Z. Cheng, L. Nie, X. Xu, and M. S. Kankanhalli (2018). "Multi-modal preference modeling for product search". In: *Proceedings of the 26th ACM International Conference on Multimedia*. 1865–1873.

Guo, J., Y. Fan, X. Ji, and X. Cheng (2019a). "MatchZoo: A learning, practicing, and developing system for neural text matching". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR'19*. Paris, France: ACM. 1297–1300.

Guo, J., Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng (2019b). "A deep look into neural ranking models for information retrieval". *Information Processing and Management*: 102067. URL: https://doi.org/10.1016/j.ipm.2019.102067.

Gysel, C. V., M. de Rijke, and E. Kanoulas (2018). "Neural vector spaces for unsupervised information retrieval". *ACM Transactions on Information Systems*. 36(4): 38:1–38:25.

Haddad, D. and J. Ghosh (2019). "Learning more from less: Towards strengthening weak supervision for ad-hoc retrieval". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR'19*. Paris, France: Association for Computing Machinery. 857–860.

Hardoon, D. R. and J. Shawe-Taylor (2003). "KCCA for different level precision in content-based image retrieval". Event Dates: 22–24 September 2004. URL: https://eprints.soton.ac.uk/259596/.

Hardoon, D. R., S. R. Szedmak, and J. R. Shawe-Taylor (2004). "Canonical correlation analysis: An overview with application to learning methods". *Neural Computation*. 16(12): 2639–2664.

He, R. and J. McAuley (2016a). "VBPR: Visual Bayesian personalized ranking from implicit feedback". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI'16*. Phoenix, AZ: AAAI Press. 144–150. URL: http://dl.acm.org/citation.cfm?id=3015812.3015834.

He, R. and J. J. McAuley (2016b). "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering". In: *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada*. 507–517.

He, X. and T.-S. Chua (2017). "Neural factorization machines for sparse predictive analytics". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: ACM. 355–364.

He, X., M.-Y. Kan, P. Xie, and X. Chen (2014). "Comment-based multi-view clustering of web 2.0 items". In: *Proceedings of the 23rd International Conference on World Wide Web. WWW '14*. Seoul, Korea: ACM. 771–782.

He, K., X. Zhang, S. Ren, and J. Sun (2016a). "Deep residual learning for image recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

He, X., H. Zhang, M.-Y. Kan, and T.-S. Chua (2016b). "Fast matrix factorization for online recommendation with implicit feedback". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '16*. Pisa, Italy: ACM. 549–558.

He, R., W.-C. Kang, and J. McAuley (2017a). "Translation-based recommendation". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems. RecSys '17*. Como, Italy: ACM. 161–169.

He, X., M. Gao, M.-Y. Kan, and D. Wang (2017b). "BiRank: Towards ranking on bipartite graphs". *IEEE Transactions on Knowledge and Data Engineering*. 29(1): 57–71.

He, X., Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua (2018a). "NAIS: Neural attentive item similarity model for recommendation". *IEEE Transactions on Knowledge and Data Engineering*. 30(12): 2354–2366.

He, X., X. Du, X. Wang, F. Tian, J. Tang, and T.-S. Chua (2018b). "Outer product-based neural collaborative filtering". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization. 2227–2233.

He, X., L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua (2017c). "Neural collaborative filtering". In: *Proceedings of the 26th International Conference on World Wide Web. WWW '17*. Perth, Australia. 173–182.

He, X., K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang (2020). "LightGCN: Simplifying and powering graph convolution network for recommendation". In: *The 43rd International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '20*. New York, NY, USA.

Hidasi, B., A. Karatzoglou, L. Baltrunas, and D. Tikk (2016). "Session-based recommendations with recurrent neural networks". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico*. URL: http://arxiv.org/abs/1511.06939.

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". *Science*. 313(5786): 504–507. URL: https://science.sciencemag.org/content/313/5786/504.

Hofmann, T. (1999). "Probabilistic latent semantic indexing". In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '99*. Berkeley, CA, USA: ACM. 50–57.

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". *Neural Networks*. 4(2): 251–257. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

Hsieh, C.-K., L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin (2017). "Collaborative metric learning". In: *Proceedings of the 26th International Conference on World Wide Web. WWW '17*. Perth, Australia. 193–201.

Hu, B., Z. Lu, H. Li, and Q. Chen (2014). "Convolutional neural network architectures for matching natural language sentences". In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 2042–2050. URL: http://papers.nips.cc/paper/5550-convolutional-neural-network-architectures-for-matching-natural-language-sentences.pdf.

Huang, P.-S., X. He, J. Gao, L. Deng, A. Acero, and L. Heck (2013). "Learning deep structured semantic models for web search using clickthrough data". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management. CIKM '13*. San Francisco, CA, USA: ACM. 2333–2338.

Huang, J., S. Yao, C. Lyu, and D. Ji (2017). "Multi-granularity neural sentence model for measuring short text similarity". In: *Database Systems for Advanced Applications*. Cham: Springer International Publishing. 439–455.

Huang, Y., Q. Wu, W. Wang, and L. Wang (2018). "Image and sentence matching via semantic concepts and order learning". *IEEE Transactions on Pattern Analysis and Machine Intelligence*: 1–1.

Hui, K., A. Yates, K. Berberich, and G. de Melo (2017). "PACRR: A position-aware neural IR model for relevance matching". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics. 1049–1058. URL: http://aclweb.org/anthology/D17-1110.

Hui, K., A. Yates, K. Berberich, and G. de Melo (2018). "Co-PACRR: A context-aware neural IR model for ad-hoc retrieval". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: ACM. 279–287.

Jiang, J.-Y., M. Zhang, C. Li, M. Bendersky, N. Golbandi, and M. Najork (2019a). "Semantic text matching for long-form documents". In: *The World Wide Web Conference. WWW '19*. San Francisco, CA, USA: Association for Computing Machinery. 795–806.

Jiang, R., S. Gowal, Y. Qian, T. A. Mann, and D. J. Rezende (2019b). "Beyond greedy ranking: Slate optimization via list-CVAE". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA*. OpenReview.net. URL: https://openreview.net/forum?id=r1xX42R5Fm.

Joachims, T. (2002). "Optimizing search engines using clickthrough data". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '02*. Edmonton, Alberta, Canada: ACM. 133–142.

Joachims, T., A. Swaminathan, and T. Schnabel (2017). "Unbiased learning-to-rank with biased feedback". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. WSDM '17*. Cambridge, UK. 781–789.

Kabbur, S., X. Ning, and G. Karypis (2013). "FISM: Factored item similarity models for top-N recommender systems". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '13*. Chicago, IL, USA: ACM. 659–667.

Kang, W. and J. J. McAuley (2018). "Self-attentive sequential recommendation". In: *IEEE International Conference on Data Mining (ICDM), Singapore*. 197–206.

Karatzoglou, A., X. Amatriain, L. Baltrunas, and N. Oliver (2010). "Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering". In: *Proceedings of the Fourth ACM Conference on Recommender Systems. RecSys '10*. Barcelona, Spain: ACM. 79–86.

Karpathy, A. and F. Li (2015). "Deep visual-semantic alignments for generating image descriptions". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA*. IEEE Computer Society. 3128–3137.

Karpathy, A., A. Joulin, and L. Fei-Fei (2014). "Deep fragment embeddings for bidirectional image sentence mapping". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems – Volume 2. NIPS'14*. Montreal, Canada: MIT Press. 1889–1897. URL: http://dl.acm.org/citation.cfm?id=2969033.2969038.

Kenter, T., A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra (2017). "Neural networks for information retrieval". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: ACM. 1403–1406.

Kingma, D. P. and M. Welling (2014). "Auto-encoding variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings*. URL: http://arxiv.org/abs/1312.6114.

Koren, Y. (2008). "Factorization meets the neighborhood: A multifaceted collaborative filtering model". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '08*. Las Vegas, NV, USA: ACM. 426–434.

Koren, Y., R. Bell, and C. Volinsky (2009). "Matrix factorization techniques for recommender systems". *Computer*. 42(8): 30–37.

Kulis, B. (2013). "Metric learning: A survey". *Foundations and Trends in Machine Learning*. 5(4): 287–364.

Le, Q. and T. Mikolov (2014). "Distributed representations of sentences and documents". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning – Volume 32. ICML'14*. Beijing, China: JMLR.org. II–1188–II–1196.

Lei, C., D. Liu, W. Li, Z. Zha, and H. Li (2016). "Comparative deep learning of hybrid representations for image recommendations". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA*. 2545–2553.

Lei, W., X. He, Y. Miao, Q. Wu, R. Hong, M.-Y. Kan, and T.-S. Chua (2020). "Estimation-action-reflection: Towards deep interaction between conversational and recommender systems". In: *Proceedings of the 13th ACM International Conference on Web Search and Data Mining. WSDM '20*. New York, NY, USA: ACM.

Li, H. (2011). "Learning to rank for information retrieval and natural language processing". *Synthesis Lectures on Human Language Technologies*. 4(1): 1–113.

Li, H. and J. Xu (2014). "Semantic matching in search". *Foundations and Trends in Information Retrieval*. 7(5): 343–469.

Li, H. and Z. Lu (2016). "Deep learning for information retrieval". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '16*. Pisa, Italy: ACM. 1203–1206.

Li, L., W. Chu, J. Langford, and R. E. Schapire (2010). "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th International Conference on World Wide Web. WWW '10*. Raleigh, NC, USA: Association for Computing Machinery. 661–670.

Li, S., J. Kawale, and Y. Fu (2015). "Deep collaborative filtering via marginalized denoising auto-encoder". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. CIKM '15*. Melbourne, Australia: ACM. 811–820.

Li, J., P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma (2017). "Neural attentive session-based recommendation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17*. Singapore, Singapore: ACM. 1419–1428.

Li, C., C. Quan, L. Peng, Y. Qi, Y. Deng, and L. Wu (2019). "A capsule network for recommendation and explaining what you like and dislike". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21–25, 2019*. 275–284.

Lian, J., X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun (2018). "xDeepFM: Combining explicit and implicit feature interactions for recommender systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18*. London, UK: ACM. 1754–1763.

Liang, D., L. Charlin, J. McInerney, and D. M. Blei (2016). "Modeling user exposure in recommendation". In: *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11–15, 2016*. 951–961.

Liang, D., R. G. Krishnan, M. D. Hoffman, and T. Jebara (2018). "Variational autoencoders for collaborative filtering". In: *Proceedings of the 2018 World Wide Web Conference. WWW '18*. Lyon, France: International World Wide Web Conferences Steering Committee. 689–698.

Liu, B., D. Niu, H. Wei, J. Lin, Y. He, K. Lai, and Y. Xu (2019a). "Matching article pairs with graphical decomposition and convolutions". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics. 6284–6294. URL: https://www.aclweb.org/anthology/P19-1632.

Liu, F., Z. Cheng, C. Sun, Y. Wang, L. Nie, and M. S. Kankanhalli (2019b). "User diverse preference modeling by multimodal attentive metric learning". In: *Proceedings of the 27th ACM International Conference on Multimedia*. 1526–1534.

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

Liu, Y., Y. Guo, E. M. Bakker, and M. S. Lew (2017). "Learning a recurrent residual fusion network for multimodal matching". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 4127–4136.

Ma, L., Z. Lu, L. Shang, and H. Li (2015). "Multimodal convolutional neural networks for matching image and sentence". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV). ICCV '15*. Washington, DC, USA: IEEE Computer Society. 2623–2631.

Masci, J., U. Meier, D. Cireşan, and J. Schmidhuber (2011). "Stacked convolutional auto-encoders for hierarchical feature extraction". In: *Proceedings of the 21th International Conference on Artificial Neural Networks – Volume Part I. ICANN'11*. Espoo, Finland: Springer-Verlag. 52–59. URL: http://dl.acm.org/citation.cfm?id=2029556.2029563.

Mikolov, T., I. Sutskever, K. Chen, G. Corrado, and J. Dean (2013). "Distributed representations of words and phrases and their compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems – Volume 2. NIPS'13*. Lake Tahoe, Nevada: Curran Associates Inc. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.

Mitra, B. and N. Craswell (2018). "An introduction to neural information retrieval". *Foundations and Trends in Information Retrieval*. 13(1): 1–126.

Mitra, B. and N. Craswell (2019). "Duet at Trec 2019 deep learning track". In: *Proceedings of the Twenty-Eighth Text REtrieval Conference, TREC 2019, Gaithersburg, MD, USA*. URL: https://trec.nist.gov/pubs/trec28/papers/Microsoft.DL.pdf.

Mitra, B., F. Diaz, and N. Craswell (2017). "Learning to match using local and distributed representations of text for web search". In: *Proceedings of the 26th International Conference on World Wide Web. WWW '17*. Perth, Australia. 1291–1299.

Nallapati, R. (2004). "Discriminative models for information retrieval". In: *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '04*. Sheffield, UK: ACM. 64–71.

Naumov, M., D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy (2019). "Deep learning recommendation model for personalization and recommendation systems". *CoRR*. abs/1906.00091. arXiv: 1906.00091.

Nguyen, T., M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng (2016). "MS MARCO: A human generated MAchine Reading COmprehension dataset". In: *Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches 2016 Co-Located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain*.

Nie, Y., A. Sordoni, and J.-Y. Nie (2018). "Multi-level abstraction convolutional model with weak supervision for information retrieval". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: ACM. 985–988.

Nogueira, R. and K. Cho (2019). "Passage re-ranking with BERT". *CoRR*. abs/1901.04085. arXiv: 1901.04085.

Nogueira, R., W. Yang, K. Cho, and J. Lin (2019). "Multi-stage document ranking with BERT". *CoRR*. abs/1910.14424. arXiv: 1910.14424.

Onal, K. D., Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H.-L. Chang, H. Kim, Q. Mcnamara, A. Angert, E. Banner, V. Khetan, T. Mcdonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. Rijke, and M. Lease (2018). "Neural information retrieval: At the end of the early years". *Information Retrieval Journal*. 21(2–3): 111–182. ISSN: 1573-7659.

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

Pan, F., S. Li, X. Ao, P. Tang, and Q. He (2019). "Warm up cold-start advertisements: Improving CTR predictions via learning to learn ID embeddings". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France*. 695–704.

Pang, L., Y. Lan, J. Guo, J. Xu, and X. Cheng (2016a). "A study of MatchPyramid models on ad-hoc retrieval". *CoRR*. abs/1606.04648. arXiv: 1606.04648.

Pang, L., Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng (2016b). "Text matching as image recognition". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI'16*. Phoenix, AZ: AAAI Press. 2793–2799. URL: http://dl.acm.org/citation.cfm?id=3016100.3016292.

Pang, L., Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng (2017a). "DeepRank: A new deep architecture for relevance ranking in information retrieval". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17*. Singapore, Singapore: ACM. 257–266.

Pang, L., Y. Lan, J. Xu, J. Guo, S.-X. Wan, and X. Cheng (2017b). "A survey on deep text matching". *Chinese Journal of Computers*. 40(4): 985–1003. URL: http://cjc.ict.ac.cn/online/onlinepaper/pl-201745181647.pdf.

Pang, L., J. Xu, Q. Ai, Y. Lan, X. Cheng, and J.-R. Wen (2020). "SetRank: Learning a permutation-invariant ranking model for information retrieval". In: *The 43rd International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '20*. Association for Computing Machinery.

Parikh, A., O. Täckström, D. Das, and J. Uszkoreit (2016). "A decomposable attention model for natural language inference". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, TX: Association for Computational Linguistics. 2249–2255. URL: http://www.aclweb.org/anthology/D16-1244.

Pasricha, R. and J. McAuley (2018). "Translation-based factorization machines for sequential recommendation". In: *Proceedings of the 12th ACM Conference on Recommender Systems. RecSys '18*. Vancouver, British Columbia, Canada: Association for Computing Machinery. 63–71.

Pasumarthi, R. K., S. Bruch, X. Wang, C. Li, M. Bendersky, M. Najork, J. Pfeifer, N. Golbandi, R. Anil, and S. Wolf (2019). "TF-ranking: Scalable TensorFlow library for learning-to-rank". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19*. Anchorage, AK, USA: ACM. 2970–2978.

Pearl, J. (2019). "The seven tools of causal inference, with reflections on machine learning". *Communications of the ACM*. 62(3): 54–60.

Pei, C., Y. Zhang, Y. Zhang, F. Sun, X. Lin, H. Sun, J. Wu, P. Jiang, J. Ge, W. Ou, and D. Pei (2019). "Personalized re-ranking for recommendation". In: *Proceedings of the 13th ACM Conference on Recommender Systems. RecSys '19*. Copenhagen, Denmark: Association for Computing Machinery. 3–11.

Pennington, J., R. Socher, and C. Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics. 1532–1543. URL: https://www.aclweb.org/anthology/D14-1162.

Peters, M., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). "Deep contextualized word representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, LA: Association for Computational Linguistics. 2227–2237. URL: https://www.aclweb.org/anthology/N18-1202.

Qiao, Y., C. Xiong, Z. Liu, and Z. Liu (2019). "Understanding the behaviors of BERT in ranking". *CoRR*. abs/1904.07531. arXiv: 1904.07531.

Qiu, R., J. Li, Z. Huang, and H. Yin (2019). "Rethinking the item order in session-based recommendation with graph neural networks". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3–7, 2019*. 579–588.

Qiu, X. and X. Huang (2015). "Convolutional neural tensor network architecture for community-based question answering". In: *Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI'15*. Buenos Aires, Argentina: AAAI Press. 1305–1311. URL: http://dl.acm.org/citation.cfm?id=2832415.2832431.

Radford, A., K. Narasimhan, T. Salimans, and I. Sutskever (2018). "Improving language understanding by generative pre-training". *Technical report*, OpenAI. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2019). "Language models are unsupervised multitask learners". *Technical report*, OpenAI. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Ranzato, M. A., Y.-L. Boureau, and Y. LeCun (2007). "Sparse feature learning for deep belief networks". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems. NIPS'07*. Vancouver, British Columbia, Canada: Curran Associates Inc. 1185–1192. URL: http://dl.acm.org/citation.cfm?id=2981562.2981711.

Rasiwasia, N., J. Costa Pereira, E. Coviello, G. Doyle, G. R. Lanckriet, R. Levy, and N. Vasconcelos (2010). "A new approach to cross-modal multimedia retrieval". In: *Proceedings of the 18th ACM International Conference on Multimedia. MM '10*. Firenze, Italy: ACM. 251–260.

Reimers, N. and I. Gurevych (2019). "Sentence-BERT: Sentence embeddings using siamese BERT-networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, Hong Kong, China*. Association for Computational Linguistics. 3982–3992. URL: http://arxiv.org/abs/1908.10084.

Rendle, S. (2010). "Factorization machines". In: *Proceedings of the 2010 IEEE International Conference on Data Mining. ICDM '10*. Washington, DC, USA: IEEE Computer Society. 995–1000.

Rendle, S., C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme (2009). "BPR: Bayesian personalized ranking from implicit feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. UAI '09*. Montreal, Quebec, Canada: AUAI Press. 452–461. URL: http://dl.acm.org/citation.cfm?id=1795114.1795167.

Rendle, S., C. Freudenthaler, and L. Schmidt-Thieme (2010). "Factorizing personalized Markov chains for next-basket recommendation". In: *Proceedings of the 19th International Conference on World Wide Web. WWW '10*. Raleigh, NC, USA: ACM. 811–820.

Ricci, F., L. Rokach, and B. Shapira (2015). *Recommender Systems Handbook*. 2nd Edn. Springer Publishing Company, Incorporated.

Richardson, M., R. Agrawal, and P. M. Domingos (2003). "Trust management for the semantic web". In: *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA*. 351–368.

Robertson, S., H. Zaragoza, and M. Taylor (2004). "Simple BM25 extension to multiple weighted fields". In: *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management. CIKM '04*. Washington, D.C., USA: ACM. 42–49.

Rosipal, R. and N. Krämer (2006). "Overview and recent advances in partial least squares". In: *Proceedings of the 2005 International Conference on Subspace, Latent Structure and Feature Selection. SLSFS'05*. Bohinj, Slovenia: Springer-Verlag. 34–51.

Salakhutdinov, R. and A. Mnih (2007). "Probabilistic matrix factorization". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems. NIPS'07*. Vancouver, British Columbia, Canada: Curran Associates Inc. 1257–1264. URL: http://dl.acm.org/citation.cfm?id=2981562.2981720.

Sarwar, B., G. Karypis, J. Konstan, and J. Riedl (2001). "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of the 10th International Conference on World Wide Web. WWW '01*. Hong Kong, Hong Kong: ACM. 285–295.

Schedl, M., H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi (2018). "Current challenges and visions in music recommender systems research". *International Journal of Multimedia Information Retrieval*. 7(2): 95–116.

Sedhain, S., A. K. Menon, S. Sanner, and L. Xie (2015). "AutoRec: Autoencoders meet collaborative filtering". In: *Proceedings of the 24th International Conference on World Wide Web. WWW '15 Companion*. Florence, Italy: ACM. 111–112.

Shan, Y., T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao (2016). "Deep crossing: Web-scale modeling without manually crafted combinatorial features". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16*. San Francisco, CA, USA: ACM. 255–262.

Shen, Y., X. He, J. Gao, L. Deng, and G. Mesnil (2014). "A latent semantic model with convolutional-pooling structure for information retrieval". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. CIKM '14*. Shanghai, China: ACM. 101–110.

Shi, Y., M. Larson, and A. Hanjalic (2014). "Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges". *ACM Computing Surveys*. 47(1): 3:1–3:45.

Socher, R., D. Chen, C. D. Manning, and A. Y. Ng (2013). "Reasoning with neural tensor networks for knowledge base completion". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems – Volume 1. NIPS'13*. Lake Tahoe, Nevada: Curran Associates Inc. 926–934. URL: http://dl.acm.org/citation.cfm?id=2999611.2999715.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research*. 15(1): 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Sun, F., J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang (2019). "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management. CIKM '19*. Beijing, China: ACM. 1441–1450.

Surdeanu, M., M. Ciaramita, and H. Zaragoza (2011). "Learning to rank answers to non-factoid questions from web collections". *Computational Linguistics*. 37(2): 351–383.

Tan, C., F. Wei, W. Wang, W. Lv, and M. Zhou (2018). "Multiway attention networks for modeling sentence pairs". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence. IJCAI'18*. Stockholm, Sweden: AAAI Press. 4411–4417.

Tang, J., X. Du, X. He, F. Yuan, Q. Tian, and T. Chua (2020). "Adversarial training towards robust multimedia recommender system". *IEEE Transactions on Knowledge and Data Engineering*. 32(5): 855–867.

Tang, J. and K. Wang (2018). "Personalized top-N sequential recommendation via convolutional sequence embedding". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: Association for Computing Machinery. 565–573.

Tao, Z., X. Wang, X. He, X. Huang, and T.-S. Chua (2019). "HoAFM: A High-Order Attentive Factorization Machine for ctr Prediction". *Information Processing & Management*: 102076. URL: http://www.sciencedirect.com/science/article/pii/S0306457319302389.

Tay, Y., L. Anh Tuan, and S. C. Hui (2018a). "Latent relational metric learning via memory-based attention for collaborative ranking". In: *Proceedings of the 2018 World Wide Web Conference. WWW '18*. Lyon, France. 729–739.

Tay, Y., A. T. Luu, and S. C. Hui (2018b). "Co-stack residual affinity networks with multi-level attention refinement for matching text sequences". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics. 4492–4502. URL: https://www.aclweb.org/anthology/D18-1479.

Tay, Y., A. T. Luu, and S. C. Hui (2018c). "Hermitian co-attention networks for text matching in asymmetrical domains". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization. 4425–4431.

Tay, Y., L. A. Tuan, and S. C. Hui (2018d). "Multi-cast attention networks". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18*. New York, NY, USA: Association for Computing Machinery. 2299–2308.

Van Gysel, C., M. de Rijke, and E. Kanoulas (2016a). "Learning latent vector spaces for product search". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, IN, USA: ACM. 165–174.

Van Gysel, C., M. de Rijke, and E. Kanoulas (2017). "Structural regularities in text-based entity vector spaces". In: *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval. ICTIR '17*. Amsterdam, The Netherlands: ACM. 3–10.

Van Gysel, C., M. de Rijke, and E. Kanoulas (2018). "Mix 'N match: Integrating text matching and product substitutability within product search". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management. CIKM '18*. Torino, Italy: ACM. 1373–1382.

Van Gysel, C., M. de Rijke, and M. Worring (2016b). "Unsupervised, efficient and semantic expertise retrieval". In: *Proceedings of the 25th International Conference on World Wide Web. WWW '16*. Montral, Qubec, Canada. 1069–1079.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning. ICML '08*. Helsinki, Finland: ACM. 1096–1103.

Wan, S., Y. Lan, J. Guo, J. Xu, L. Pang, and X. Cheng (2016a). "A deep architecture for semantic matching with multiple positional sentence representations". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI'16*. Phoenix, AZ: AAAI Press. 2835–2841. URL: http://dl.acm.org/citation.cfm?id=3016100.3016298.

Wan, S., Y. Lan, J. Xu, J. Guo, L. Pang, and X. Cheng (2016b). "Match-SRNN: Modeling the recursive matching structure with spatial RNN". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. IJCAI'16*. New York, NY, USA: AAAI Press. 2922–2928. URL: http://dl.acm.org/citation.cfm?id=3060832.3061030.

Wang, B., Y. Yang, X. Xu, A. Hanjalic, and H. T. Shen (2017a). "Adversarial cross-modal retrieval". In: *Proceedings of the 25th ACM International Conference on Multimedia. MM '17*. Mountain View, CA, USA: ACM. 154–162.

Wang, H., F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, and M. Guo (2018a). "RippleNet: Propagating user preferences on the knowledge graph for recommender systems". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery. 417–426.

Wang, J., A. P. de Vries, and M. J. T. Reinders (2006). "Unifying user-based and item-based collaborative filtering approaches by similarity fusion". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '06*. Seattle, Washington, DC, USA: ACM. 501–508.

Wang, L., Y. Li, J. Huang, and S. Lazebnik (2018b). "Learning two-branch neural networks for image-text matching tasks". *IEEE Transactions on Pattern Analysis and Machine Intelligence*: 1–1.

Wang, L., Y. Li, and S. Lazebnik (2016). "Learning deep structure-preserving image-text embeddings". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 00. 5005–5013. URL: doi.ieeecomputersociety.org/10.1109/CVPR.2016.541.

Wang, X., X. He, Y. Cao, M. Liu, and T. Chua (2019a). "KGAT: Knowledge graph attention network for recommendation". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4–8, 2019*. 950–958.

Wang, X., X. He, F. Feng, L. Nie, and T. Chua (2018c). "TEM: Tree-enhanced embedding model for explainable recommendation". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. 1543–1552.

Wang, X., X. He, L. Nie, and T.-S. Chua (2017b). "Item silk road: Recommending items from information domains to social users". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: ACM. 185–194.

Wang, X., X. He, M. Wang, F. Feng, and T.-S. Chua (2019b). "Neural graph collaborative filtering". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR'19*. Paris, France: Association for Computing Machinery. 165–174.

Wang, X., D. Wang, C. Xu, X. He, Y. Cao, and T. Chua (2019c). "Explainable reasoning over knowledge graphs for recommendation". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*. 5329–5336. URL: https://aaai.org/ojs/index.php/AAAI/article/view/4470.

Wang, X., Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang (2019d). "Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, Z., W. Hamza, and R. Florian (2017c). "Bilateral multi-perspective matching for natural language sentences". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 4144–4150.

Wang, Z., G. Xu, H. Li, and M. Zhang (2011). "A fast and accurate method for approximate string search". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies – Volume 1. HLT '11*. Portland, OR, USA: Association for Computational Linguistics. 52–61. URL: http://dl.acm.org/citation.cfm?id=2002472.2002480.

Wei, X. and W. B. Croft (2006). "LDA-based document models for ad-hoc retrieval". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '06*. Seattle, Washington, DC, USA: ACM. 178–185.

Wei, Y., X. Wang, L. Nie, X. He, R. Hong, and T.-S. Chua (2019). "MMGCN: Multi-modal graph convolution network for personalized recommendation of micro-video". In: *Proceedings of the 27th ACM International Conference on Multimedia. MM '19*. Nice, France: ACM. 1437–1445.

Wu, B., X. He, Z. Sun, L. Chen, and Y. Ye (2019a). "ATM: An attentive translation model for next-item recommendation". *IEEE Transactions on Industrial Informatics*: 1–1.

Wu, C.-Y., A. Ahmed, A. Beutel, A. J. Smola, and H. Jing (2017). "Recurrent recommender networks". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. New York, NY, USA. 495–503.

Wu, L., P. Sun, Y. Fu, R. Hong, X. Wang, and M. Wang (2019b). "A neural influence diffusion model for social recommendation". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21–25, 2019*. 235–244.

Wu, Q., H. Wang, Q. Gu, and H. Wang (2016a). "Contextual bandits in a collaborative environment". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '16*. Pisa, Italy. 529–538.

Wu, S., Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan (2019c). "Session-based recommendation with graph neural networks". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, HI, USA, 2019*. 346–353.

Wu, W., H. Li, and J. Xu (2013a). "Learning query and document similarities from click-through bipartite graph with metadata". In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. WSDM '13*. Rome, Italy: ACM. 687–696.

Wu, W., Z. Lu, and H. Li (2013b). "Learning bilinear model for matching queries and documents". *Journal of Machine Learning Research*. 14(1): 2519–2548. URL: http://dl.acm.org/citation.cfm?id=2567709.2567742.

Wu, Y., C. DuBois, A. X. Zheng, and M. Ester (2016b). "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*. San Francisco, CA, USA: ACM. 153–162.

Xiao, J., H. Ye, X. He, H. Zhang, F. Wu, and T.-S. Chua (2017). "Attentional factorization machines: Learning the weight of feature interactions via attention networks". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI'17*. Melbourne, Australia: AAAI Press. 3119–3125. URL: http://dl.acm.org/citation.cfm?id=3172077.3172324.

Xin, X., B. Chen, X. He, D. Wang, Y. Ding, and J. Jose (2019a). "CFM: Convolutional factorization machines for context-aware recommendation". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI'19. International Joint Conferences on Artificial Intelligence Organization*. 3119–3125. URL: https://doi.org/10.24963/ijcai.2019/545.

Xin, X., X. He, Y. Zhang, Y. Zhang, and J. M. Jose (2019b). "Relational collaborative filtering: Modeling multiple item relations for recommendation". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21–25, 2019*. 125–134.

Xiong, C., Z. Dai, J. Callan, Z. Liu, and R. Power (2017). "End-to-end neural ad-hoc ranking with kernel pooling". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: ACM. 55–64.

Xue, F., X. He, X. Wang, J. Xu, K. Liu, and R. Hong (2019). "Deep item-based collaborative filtering for top-N recommendation". *ACM Transactions on Information Systems*. 37(3).

Xue, H.-J., X. Dai, J. Zhang, S. Huang, and J. Chen (2017). "Deep matrix factorization models for recommender systems". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3203–3209.

Yan, F. and K. Mikolajczyk (2015). "Deep correlation for matching images and text". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3441–3450.

Yang, L., Q. Ai, J. Guo, and W. B. Croft (2016). "aNMM: Ranking short answer texts with attention-based neural matching model". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, IN, USA: ACM. 287–296.

Yang, P., H. Fang, and J. Lin (2018). "Anserini: Reproducible ranking baselines using lucene". *J. Data and Information Quality*. 10(4): 16:1–16:20.

Yang, R., J. Zhang, X. Gao, F. Ji, and H. Chen (2019a). "Simple and effective text matching with richer alignment features". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics. 4699–4709. URL: https://www.aclweb.org/anthology/P19-1465.

Yang, W., H. Zhang, and J. Lin (2019b). "Simple applications of BERT for ad hoc document retrieval". *CoRR*. abs/1903.10972. arXiv: 1903.10972.

Yang, Y., S. W.-T. Yih, and C. Meek (2015). "WikiQA: A challenge dataset for open-domain question answering". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. ACL – Association for Computational Linguistics. URL: https://www.microsoft.com/en-us/research/publication/wikiqa-a-challenge-dataset-for-open-domain-question-answering/.

Yang, Z., Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le (2019c). "XLNet: Generalized autoregressive pretraining for language understanding". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc. 5753–5763. URL: http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf.

Yi, X. and J. Allan (2009). "A comparative study of utilizing topic models for information retrieval". In: *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval. ECIR '09*. Toulouse, France: Springer-Verlag. 29–41.

Yin, W. and H. Schütze (2015). "MultiGranCNN: An architecture for general matching of text chunks on multiple levels of granularity". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics. 63–73. URL: https://www.aclweb.org/anthology/P15-1007.

Yin, W., H. Schütze, B. Xiang, and B. Zhou (2016). "ABCNN: Attention-based convolutional neural network for modeling sentence pairs". *Transactions of the Association for Computational Linguistics*. 4: 259–272.

Ying, R., R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec (2018). "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18*. London, UK: ACM. 974–983.

Yuan, F., X. He, H. Jiang, G. Guo, J. Xiong, Z. Xu, and Y. Xiong (2020). "Future data helps training: Modeling future contexts for session-based recommendation". In: *Proceedings of the Web Conference 2020. WWW '20*. Taipei, Taiwan: Association for Computing Machinery. 303–313.

Yuan, F., A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He (2019). "A simple convolutional generative network for next item recommendation". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. WSDM '19*. Melbourne VIC, Australia: Association for Computing Machinery. 582–590.

Zamani, H. and W. B. Croft (2016). "Estimating embedding vectors for queries". In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval. ICTIR '16*. Newark, DE, USA: Association for Computing Machinery. 123–132.

Zamani, H. and W. B. Croft (2017). "Relevance-based word embedding". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17*. Shinjuku, Tokyo, Japan: Association for Computing Machinery. 505–514.

Zamani, H. and W. B. Croft (2018a). "Joint modeling and optimization of search and recommendation". In: *Proceedings of the First Biennial Conference on Design of Experimental Search & Information Retrieval Systems. DESIRES '18*. Bertinoro, Italy: CEUR-WS. 36–41. URL: http://ceur-ws.org/Vol-2167/paper2.pdf.

Zamani, H. and W. B. Croft (2018b). "On the theory of weak supervision for information retrieval". In: *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval. ICTIR '18*. Tianjin, China: Association for Computing Machinery. 147–154.

Zamani, H. and W. B. Croft (2020). "Learning a joint search and recommendation model from user-item interactions". In: *Proceedings of the 13th International Conference on Web Search and Data Mining. WSDM '20*. Houston, TX, USA: Association for Computing Machinery. 717–725.

Zamani, H., W. B. Croft, and J. S. Culpepper (2018a). "Neural query performance prediction using weak supervision from multiple signals". In: *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: Association for Computing Machinery. 105–114.

Zamani, H., J. Dadashkarimi, A. Shakery, and W. B. Croft (2016). "Pseudo-relevance feedback based on matrix factorization". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, IN, USA: ACM. 1483–1492.

Zamani, H., M. Dehghani, W. B. Croft, E. Learned-Miller, and J. Kamps (2018b). "From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management. CIKM '18*. Torino, Italy: Association for Computing Machinery. 497–506.

Zamani, H., B. Mitra, X. Song, N. Craswell, and S. Tiwary (2018c). "Neural ranking models with multiple document fields". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. WSDM '18*. Marina Del Rey, CA, USA: ACM. 700–708.

Zhang, S., L. Yao, A. Sun, and Y. Tay (2019). "Deep learning based recommender system: A survey and new perspectives". *ACM Computing Surveys*. 52(1): Article 5.

Zhang, X., H. Xie, H. Li, and J. C. S. Lui (2020). "Conversational contextual bandit: Algorithm and application". In: *Proceedings of the Web Conference 2020. WWW '20*. Taipei, Taiwan: Association for Computing Machinery. 662–672.

Zhang, Y., Q. Ai, X. Chen, and W. B. Croft (2017). "Joint representation learning for top-N recommendation with heterogeneous information sources". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17*. Singapore, Singapore: ACM. 1449–1458.

Zhang, Y., X. Chen, Q. Ai, L. Yang, and W. B. Croft (2018). "Towards conversational search and recommendation: System ask, user respond". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management. CIKM '18*. Torino, Italy: ACM. 177–186.

Zhao, W. X., G. He, K. Yang, H. Dou, J. Huang, S. Ouyang, and J. Wen (2019). "KB4Rec: A data set for linking knowledge bases with recommender systems". *Data Intelligence*. 1(2): 121–136.

Zheng, L., C. Lu, F. Jiang, J. Zhang, and P. S. Yu (2018a). "Spectral collaborative filtering". In: *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2–7, 2018*. 311–319.

Zheng, L., V. Noroozi, and P. S. Yu (2017). "Joint deep modeling of users and items using reviews for recommendation". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. WSDM '17*. Cambridge, UK: ACM. 425–434.

Zheng, Y., Z. Fan, Y. Liu, C. Luo, M. Zhang, and S. Ma (2018b). "Sogou-QCL: A new dataset with click relevance label". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '18*. Ann Arbor, MI, USA: ACM. 1117–1120.

Zhou, G., X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai (2018). "Deep interest network for click-through rate prediction". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18*. London, UK: Association for Computing Machinery. 1059–1068.

Zhu, M., A. Ahuja, W. Wei, and C. K. Reddy (2019). "A hierarchical attention retrieval model for healthcare question answering". In: *The World Wide Web Conference. WWW '19*. San Francisco, CA, USA: Association for Computing Machinery. 2472–2482.