Noname manuscript No.
(will be inserted by the editor)

Neural Information Retrieval: At the End of the Early Years

Kezban Dilek Onal[⋆] · Ye Zhang[⋆] · Ismail Sengor Altingovde · Md Mustafizur Rahman · Pinar Karagoz · Alex Braylan · Brandon Dang · Heng-Lu Chang · Henna Kim · Quinten McNamara · Aaron Angert · Edward Banner · Vivek Khetan · Tyler McDonnell · An Thanh Nguyen · Dan Xu · Byron C. Wallace · Maarten de Rijke[†] · Matthew Lease[†]

Received: date / Accepted: date

Abstract A recent "third wave" of neural network (NN) approaches now delivers state-of-the-art performance in many machine learning tasks, spanning speech recognition, computer vision, and natural language processing. Because these modern NNs

I.S. Altingovde is supported by Turkish Academy of Sciences Distinguished Young Scientist Award (TÜBA-GEBİP 2016). P. Karagoz is partially funded by TUBITAK grant no. 112E275. M. de Rijke was supported by Ahold Delhaize, Amsterdam Data Science, the Bloomberg Research Grant program, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 612.001.116, HOR-11-10, CI-14-25, 652.002.001, 612.001.551, 652.001.003, and Yandex. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

⋆ † These authors contributed equally.

K.D. Onal · I.S. Altingovde · P. Karagoz
Middle East Technical University
E-mail: dilek@ceng.metu.edu.tr, altingovde@ceng.metu.edu.tr, karagoz@ceng.metu.edu.tr

K.D. Onal · M. de Rijke
University of Amsterdam
E-mail: k.d.onal@uva.nl, derijke@uva.nl

Y. Zhang · M.M. Rahman · A. Braylan · B. Dang · H. Chang · H. Kim · Q. McNamara · V. Khetan · T. McDonnell · A.T. Nguyen · D. Xu · M. Lease
University of Texas at Austin
E-mail: yezhang@utexas.edu, nahid@utexas.edu, braylan@cs.utexas.edu, budang@utexas.edu, hengluchang@utexas.edu, henna@utexas.edu, quinten.mcnamara@utexas.edu, vivek.khetan@utexas.edu, tmcdonnell@utexas.edu, atn@cs.utexas.edu, xudan0812@utexas.edu, ml@utexas.edu

A. Angert
IBM
E-mail: aarondangert@gmail.com

E. Banner · B.C. Wallace
College of Computer and Information Science, Northeastern University
E-mail: ebanner@ccs.neu.edu, byron@ccs.neu.edu

often comprise multiple interconnected layers, work in this area is often referred to as *deep learning*. Recent years have witnessed an explosive growth of research into NN-based approaches to information retrieval (IR). A significant body of work has now been created. In this paper, we survey the current landscape of *Neural IR* research, paying special attention to the use of learned distributed representations of textual units. We highlight the successes of neural IR thus far, catalog obstacles to its wider adoption, and suggest potentially promising directions for future research.

Keywords Deep learning · distributed representation · neural network · recurrent neural network · search engine · word embedding · semantic matching · semantic compositionality

1 Introduction

We are in the midst of a tremendous resurgence of interest and renaissance in research on artificial neural network (NN) models for machine learning, now commonly referred to as *deep learning*.[1] While many valuable introductory readings, tutorials, and surveys already exist for deep learning at large, we are not familiar with any existing literature review surveying NN approaches to Information Retrieval (IR). Given the great recent rise of interest in such *Neural IR* from academic and industrial researchers [67, 111] and practitioners [135, 154] alike, and given the significant body of work that has been created in just a few years, we believe that such a literature review is now timely. IR researchers interested in getting started with Neural IR currently must identify and compile many scattered works. Unifying these into a coherent resource provides a single point of reference for those interested in learning about these emerging approaches, as well as provide a valuable reference compendium for more experienced Neural IR researchers.

To address this need, this literature review surveys recent work in Neural IR. Our survey is intended for IR researchers (e.g., typical readers of this journal) already familiar with fundamental IR concepts and so requiring few definitions or explanations of these. Those less familiar with IR may wish to consult existing reference materials [47, 129] for unfamiliar terms or concepts. However, we anticipate many readers will have relatively less familiarity and experience with NNs and deep learning. Consequently, we point exemplary introductory resources on deep learning and briefly introduce key terms, definitions, and concepts in Section 3.

In terms of scope, the survey is limited to textual IR. For NN approaches to content-based image retrieval, see [190]. Similarly, we exclude work on NN approaches to acoustic or multi-modal IR, such as mixing text with imagery (see [125, 126]). We also intentionally focus on the current "third wave" revival of NN research, excluding earlier work.

For the purposes of this survey, the *early years of neural IR* refers to the period up to the end of 2016. Relevant research from this period has mainly been focused on the long standing vocabulary mismatch problem in textual IR: the phenomenon

[1] While not all NNs are 'deep' and not all 'deep' models are neural, the terms are often conflated in practice.

that the vocabulary of the searcher and the vocabulary used in relevant documents may be different. This focus was motivated by the success of neural network models in learning distributed representations for words [14, 17, 138, 161] and larger textual units [84, 103]. A *distributed representation* for a textual unit is a dense real-valued vector that somehow encodes the semantics of the textual unit [132]. Distributed representations hold the promise of aiding semantic matching: by mapping words and other textual units to their representations, semantic matches can be computed in the representation space [112]. Indeed, recent improvements in obtaining distributed representations using neural models have quickly been used for semantic matching of textual units in IR tasks.

The problem of mapping words to a representation that can capture their meanings is referred as *distributional semantics* and has been studied for a very long time; see [186] for an overview. *Neural language models*, which may be viewed as a particular flavor of distributional semantic models, so-called *context-predicting distributional semantic models*, have been shown to outperform so-called context-counting models such as Hyperspace Analog to Language (HAL) [122], Latent Semantic Analysis (LSA) [53], on word analogy and semantic relatedness tasks [14]. Moreover, Levy et al [108] improve context-counting models by adopting lessons from context-predicting models. Bengio et al [17] seem to have been the first to propose a neural language model; they introduce the idea of simultaneously learning a language model that predicts a word given its context and its representation, a so-called word embedding. This idea has since been adopted by many follow-up studies. The most well-known and most widely used context-predicting models, `word2vec` [138] and Global Vectors (GloVe) [161], have been used extensively in recent work on web search. The success of neural word embeddings has also given rise to work on computing context-predicting representations of larger textual units, including paragraphs and documents [84].

We review the literature in the area and consider both word representations and representations of larger textual units, such as sentences and paragraphs. We survey the use of neural language models and word embeddings, and detail applications of so-called neural semantic compositionality models that determine semantic representations of larger text units from smaller ones, and we present novel neural semantic compositionality models designed specifically for IR tasks. In addition to surveying relevant literature, we provide the reader with a foundation on neural language models and with pointers to relevant resources that those who are new to the area should appreciate.

Regarding the set of reviewed textual IR tasks, we largely follow the traditional divide between IR and Natural language processing (NLP), including search-related IR research and excluding syntactic and semantic NLP work. However, this division is perhaps most difficult to enforce in regard to two different classes of IR tasks. First is question answering (QA) and community question answering (CQA) tasks, which perhaps represent the greatest cross-over between the NLP and IR fields (e.g., see Dumais et al [59]). Recent years have shown that QA research is often more focused on semantic understanding, reasoning and generative models rather than on search over large collections. In this survey, we review QA work that is focused on retrieval of textual answers. For readers interested in further reading about deep QA, see [23, 67, 70, 100]. Secondly, textual similarity and document representations are crucial to many different applications such as document clustering and classification. In this survey, we review neural models for textual similarity only if the model is evaluated for retrieval of similar textual units. Limiting to *Similar Item Retrieval*, we exclude works on neural models for general purpose textual similarity such as Hill et al [84], Kenter and de Rijke [95].

Finally, regarding nomenclature, our use of *Neural IR*, referring to machine learning research on *artificial* NNs and deep learning, should not be confused with cognitive science research studying *actual* neural representations of relevance in the human brain (see [148]). In addition, note that the modern appellation of *deep learning* for the current "third wave" of NN research owes to these approaches using a large number of *hidden layers* in their NN architectures. For this literature review, we have chosen the term *neural* (rather than *deep*) because: (i) most current work on textual NNs in NLP and IR is often actually quite shallow in the number of layers used (typically only a few hidden layers and often only one, though some notable recent exceptions exist, such as Conneau et al [46]); (ii) the use of *neural* more clearly connects the current wave of *deep learning* to the long history of NN research; and (iii) our use of *Neural IR* is consistent with the naming of this journal's special issue and the other articles therein.

The remainder of this literature review is organized as follows. To illustrate the rough evolution of Neural IR research over time, we begin by providing a concise history of Neural IR in Section 2. Following this, in Section 3 we present background and terminology; readers with a background in neural methods can skip over this section. In Section 4, we detail the dimensions that we use for categorizing the publications we review. We classified textual IR work using five main classes of tasks. We review works on Ad-hoc retrieval, QA, Query Tasks, Sponsored Search and Similar Item Retrieval in Section 5, 6, 7, 8, and 9, respectively. So far, there have been few publications devoted to neural behavioral models; we cover them in Section 10. In Section 11 we present lessons and reflections, and we conclude in Section 12. We include multiple appendices, one with acronyms used in Appendix A, the second with a list of resources used in reviewed work; we believe this should help newcomers to the area; see Appendix B.

2 A Brief History of Neural IR

In this section we present a bird's eye view of key publications in neural IR up to the end of 2016. Introductory resources on deep learning cited in Section 3.1 (see LeCun et al [105] and Goodfellow et al [71]) explain how the "third wave" of interest in neural network approaches arose. Key factors include increased availability of "big data," more powerful computing resources, and better NN models and parameter estimation techniques. While early use in language modeling for automatic speech recognition (ASR) and machine translation (MT) might be loosely related to language modeling for IR [162], state-of-the-art performance provided by neural language models trained on vast amounts of data could not be readily applied to the more typical sparse data setting of training document-specific language models in

IR. Similarly, neural approaches in computer vision to learn higher-level representations (i.e., *visual concepts*) from low-level pixels were not readily transferable to text-based research on words.

In 2009, Salakhutdinov and Hinton [171] published the first "third wave" Neural IR publications that we are aware of, employing a deep *auto-encoder* architecture (Section 3.2) for semantic modeling for related document search (Section 9.1.2). They did not have relevance judgments for evaluation, so instead used document corpora with category labels and assumed relevance if a query and document had matching category labels. Little happened in terms of neural approaches to IR between 2009 and 2013.

In 2013, the Deep Structured Semantic Model (DSSM) [89] was introduced, a neural model that directly addresses the ad-hoc search task. Work by Clinchant and Perronnin [42] was the first to aggregate word embeddings for IR. Mikolov and Dean [136] and Mikolov et al [139] proposed `word2vec`. And Lu and Li [121] proposed *DeepMatch*, a deep matching method used on two datasets: CQA (matching questions with answers) and a Twitter-like micro-blog task (matching tweets with comments).

In 2014, the first two neural IR papers appeared in ACM SIGIR. Gupta et al [80] proposed an auto-encoder approach to mixed-script query expansion, considering transliterated search queries and learning a character-level "topic" joint distribution over features of both scripts. Zhang et al [216] considered the task of local text reuse, with three annotators labeling whether or not a given passage represents reuse. New variants of DSSM were proposed. And Sordoni et al [181] investigated a deep IR approach to query expansion, evaluating ad-hoc search on TREC collections. Le and Mikolov [103] proposed their Paragraph Vector (PV) method for composing word embeddings to induce semantic representations over longer textual units (see Section 11.2.2).

By 2015, work on neural IR had grown beyond what can be concisely described here. We saw `word2vec` enter wider adoption in IR research (e.g., [65, 75, 95, 222, 225]), as well as a flourishing of neural IR work appearing at SIGIR [65, 76, 143, 174, 189, 222], spanning ad-hoc search [65, 189, 222, 225], QA sentence selection and Twitter reranking [95], cross-lingual IR [189], paraphrase detection [95], query completion [143], query suggestion [182], and sponsored search [75, 76]. In 2015, we also saw the first workshop on Neural IR.[2]

In 2016, work on neural IR began to accelerate in terms of the volume of work, the sophistication of methods, and practical effectiveness (e.g., Guo et al [78]). SIGIR also featured its first workshop on the subject,[3] as well as its first tutorial on the subject Li and Lu [111]. To provide as current of coverage as possible in this literature review, we include articles appearing up through ACM ICTIR 2016 and CIKM 2016 conferences.

[2] `http://research.microsoft.com/en-us/um/beijing/events/DL-WSDM-2015/`

[3] `https://www.microsoft.com/en-us/research/event/neuir2016/`

3 Background

We assume that our readers are familiar with basic concepts from IR but possibly less versed in NN concepts; readers with a background in neural methods can skip ahead to Section 4. Below, in Section 3.1, we provide pointers to existing introductory materials that can help the reader get started. In Section 3.2, we present background material for the NN concepts that are crucial for the rest of the survey. In Section 3.3, we briefly review key concepts that underly the use of neural models for textual IR tasks, viz. distributional semantics, semantic compositionality, neural language models, training procedures for neural language models, `word2vec` and GloVe, and paragraph vectors.

3.1 Introductory Tutorials

For general introductions to deep learning, see [7, 54, 71, 105, 172, 210], and Bengio [16]. See Broder et al [27] for a recent panel discussion on deep learning. For introductions to deep learning approaches to other domains, see [85] for ASR, [70] for NLP, and [198] for MT. Goldberg [70] covers details on training neural networks and a broader set of architectures including feed-forward NN, convolutional neural network (CNN), recurrent neural network (RNN) and recursive neural network (RecNN). Cho [38] focuses on language modeling and machine translation, sketches a clear picture of encoder-decoder architectures, recurrent networks and attention modules.

Regarding NN approaches to IR, informative talks and tutorials have been presented by Gao [67], Li and Lu [111], and Li [109, 110]. Many other useful tutorials and talks can be found online for general deep learning and specific domains. A variety of resources for deep learning, including links to popular open-source software, can be found at `http://deeplearning.net`.

3.2 Background on Neural Networks

3.2.1 Neural networks

The neural models we will typically consider in this survey are *feed-forward* networks, which we refer to as NN for simplicity. A simple example of such an NN is shown in Figure 1. Input features are extracted, or *learned*, by NNs using multiple, stacked fully connected layers. Each layer applies a linear transformation to the vector output of the last layer (performing an affine transformation). Thus each layer is associated with a matrix of parameters, to be estimated during learning. This is followed by element-wise application of a non-linear activation function. In the case of IR, the output of the entire network is often either a vector representation of the input or some predicted scores. During training, a loss function is constructed by contrasting the prediction with the ground truth available for the training data, where training adjusts network parameters to minimize loss. This is typically performed via the classic back-propagation algorithm [170]. For further details, see [71].

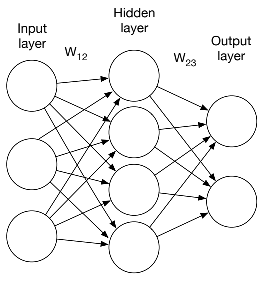


Fig. 1: Feed-forward fully connected neural network.

3.2.2 Auto-encoder

An *auto-encoder* NN is an unsupervised model used to learn a representation for data, typically for the purpose of dimensionality reduction. Unlike typical NNs, an auto-encoder is trained to reconstruct the input, and the output has the same dimension as the input. For more details, see [62, 86]. *auto-encoder* was applied in IR in [171].

3.2.3 Restricted Boltzman Machine (RBM)

An RBM is a stochastic neural network whose binary activations depend on its neighbors and have a probabilistic binary activation function. RBMs are useful for dimensionality reduction, classification, regression, collaborative filtering, feature learning, topic modeling, etc. The RBM was originally proposed by Smolensky [178] and further popularized by Nair and Hinton [149].

3.2.4 Convolutional neural network

In contrast to the densely-connected networks described above, a convolutional neural network (CNN) [104] defines a set of linear filters (kernels) connecting only spatially local regions of the input, greatly reducing computation. These filters extract locally occurring patterns. CNNs are typically built following a “convolution+pooling” architecture, where a pooling layer following convolution further extracts the most important features while at the same time reducing dimensionality. We show a basic example of a CNN in Figure 2. CNNs were first established by their strong performance on image classification [99], then later adapted to text-related tasks in NLP and IR [45, 93, 97, 219, 221]. As discussed in Section 11.2.2, Yang et al [205], Guo et al [79], and Severyn and Moschitti [174] question whether CNN models developed in computer vision and NLP to exploit spatial and positional information are equally-well suited to the IR domain.

3.2.5 Recurrent neural network

A recurrent neural network (RNN) [61] models sequential inputs, e.g., sequences of words in a document. We show a basic example of an RNN in Figure 3. Individual

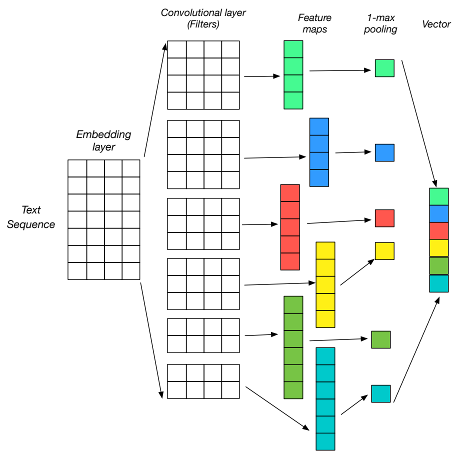


Fig. 2: One-dimensional CNN with only one convolution layer (six filters and six feature maps), followed by a 1-max-pooling layer.

input units (e.g., words) are typically encoded in vector representations. RNNs usu-

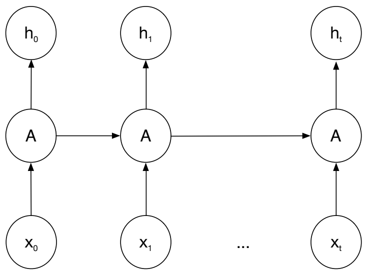


Fig. 3: recurrent neural network (RNN). Here, x is the input, A is the computation unit shared across time steps, and h is the hidden state vector.

ally read inputs sequentially; one can think of the input order as indexing “time.” Thus, the first word corresponds to an observation at time 0, the second at time 1, and so on. The key component of RNNs is a hidden state vector that encodes salient information extracted from the input read thus far. At each step during traversal (at each time point t), this state vector is updated as a function of the current state vector and the input at time t. Thus, each time point is associated with its own unique state vector, and when the end of a piece of text is reached, the state vector will capture the context induced by the entire sequence.

A technical problem with fitting the basic RNN architecture just described is the “vanishing gradient problem” [160] inherent to parameter estimation via back-propagation “through time.” The trouble is that gradients must flow from later time steps of the sequence back to earlier bits of the input. This is difficult for long sequences, as the gradient tends to degrade, or "vanish," as they are passed backwards through time. Fortunately, there are two commonly used variants of RNNs that aim to mitigate this problem (and have proven empirically successful in doing so): LSTM and GRU.

Long Short Term Memory (LSTM) [87] was the first approach introduced to address the vanishing gradient problem. In addition to the hidden state vector, LSTMs have a *memory cell* structure, governed by three gates. An *input gate* is used to control how much the memory cell will be influenced by the new input; a *forget gate* dictates how much previous information in the memory cell will be forgotten; and an *output gate* controls how much the memory cell will influence the current hidden state. All three of these gates depend on the previous hidden state and the current input. For a more detailed description of LSTM and more LSTM variants, see [73, 77].

Bahdanau et al [10]'s *Gated Recurrent Unit (GRU)* is a more recent architecture, similar to the LSTM model but simpler (and thus with fewer parameters). Empirically, GRUs have been found to perform comparably to LSTMs, despite their comparative simplicity [41]. Instead of the memory cell used by LSTMs, an *update gate* is used to govern the extent to which the hidden gate will be updated, and a *reset gate* is used to control the extent to which the previous hidden state will influence the current state.

3.2.6 Attention

The notion of *attention* has lately received a fair amount of interest from NN researchers. The idea is to imbue the model with the ability to *learn* which bits of a sequence are most important for a given task (in contrast, e.g., to relying only on the final hidden state vector).

Attention was first proposed in the context of neural machine translation model by Bahdanau et al [10]. The original RNN model for machine translation [184] encodes the source sentence into a fixed-length vector (by passing an RNN over the input, as described in Section 3.2.5). This is then accepted as input by a *decoder* network, which uses the single encoded vector as the only information pertaining to the source sentence. This means that all relevant information required for the translation must be stored in a single vector — a difficult aim.

The attention mechanism was proposed to alleviate this requirement. At each time step (as the decoder generates each word), the model identifies a set of positions in the source sentence that is most relevant to its current position in the output (a function of its index and what it has generated thus far). These positions will be associated with corresponding state vectors. The current contextualizing vector (to be used to generate output) can then be taken as a sum of these, weighted by their estimated relevance. This attention mechanism has also been used in image caption generation [202]. A similar line of work includes *Neural Turing Machines* by Graves et al [74] and *Memory Networks* by Weston et al [196].

3.3 Word Embeddings and Semantic Compositionality

3.3.1 Distributional semantics

A *distributional semantic model* (DSM) is a model that relies on the *distributional hypothesis* [83], according to which *words that occur in the same contexts tend to have similar meanings*, for associating words with vectors that can capture their meaning. Statistics on observed contexts of words in a corpus is quantified to derive word vectors. The most common choice of context is the set of words that co-occur in a context window.

Baroni et al [14] classify existing DSMs into two categories: *context-counting* and *context-predicting*. The context-counting category includes earlier DSMs such as Hyperspace Analog to Language (HAL) [122], Latent Semantic Analysis (LSA) [53]. In these models, low-dimensional word vectors are obtained via factorisation of a high-dimensional sparse co-occurrence matrix. The *context-predicting* models are neural language models in which word vectors are modelled as additional parameters of a neural network that predicts co-occurrence likelihood of context-word pairs. Neural language models comprise an embedding layer that maps a word to its distributed representation. A *distributed representation* of a symbol is a vector of features that characterize the meaning of the symbol and are not mutually exclusive [132].

Early neural language models were not aimed at learning representations for words. However, it soon turned out that the embedding layer component, which addresses the curse of dimensionality caused by one-hot vectors [17], yields useful distributed word representations, so-called *word embeddings*. Collobert and Weston [44] are the first ones to show the benefit of word embeddings as features for NLP tasks. Soon afterwards, word embeddings became widespread after the introduction of the shallow models Skip-gram and Continuous Bag of Words (CBOW) in the `word2vec` framework by Mikolov et al [138, 140]; see Section 3.3.5.

Baroni et al [14] report that context-predicting models outperform context-counting models on several tasks, including question sets, semantic relatedness, synonym detection, concept categorization and word analogy. In contrast, Levy et al [108] point out that the success of the popular context-predicting models `word2vec` and GloVe does not originate from the neural network architecture and the training objective but from the choices of hyper-parameters for contexts. A comprehensive analysis reveals that when these hyper-parameter choices are applied to context-counting models, no consistent advantage of context-predicting models is observed over context-counting models.

3.3.2 Semantic compositionality

Compositional distributional semantics or *semantic compositionality* (SC) is the problem of formalizing how the meaning of larger textual units such as sentences, phrases, paragraphs and documents are built from the meanings of words [142]. Work on SC studies are motivated by the *Principle of Compositionality* which states that the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.

A *neural SC* model maps the high-dimensional representation of a textual unit into a distributed representation by forward propagation in a neural network. The neural network parameters are learned by training to optimize task-specific objectives. Both the granularity of the target textual unit and the target task play an important role for the choice of neural network type and training objective. A SC model that considers the order of words in a sentence and aims to obtain a deep understanding may fail in an application that requires representations that can encode high-level concepts in a large document. A comparison of neural SC models of sentences learned from unlabelled data is presented in [84]. Besides the models reviewed by Hill et al [84], there exist sentence-level models that are trained using task-specific labelled data. For instance, a model can be trained to encode the sentiment of a sentence using a dataset of sentences annotated with sentiment class labels [113].

To the best of our knowledge, there is no survey on neural SC models for distributed representations of long documents, although the representations are useful not only for document retrieval but also for document classification and recommendation. In Section 4.2.2 we review the subset of neural SC models and associated training objectives adopted specifically in IR tasks.

3.3.3 Neural language models

A *language model* is a function that predicts the acceptability of pieces of text in a language. Acceptability scores are useful for ranking candidates in tasks like machine translation or speech recognition. The probability of a sequence of words $P(w_1, w_2, \ldots, w_n)$ in a language, can be computed by Equation 1, in accordance with the chain rule:

$$P(w_1, w_2, \ldots, w_{t-1}, w_t) = P(w_1)P(w_2 \mid w_1) \cdots P(w_t \mid w_1, w_2, \ldots, w_{t-1}) \quad (1)$$

Probabilistic language models mostly approximate Equation 1 by $P(w_t \mid w_{t-n}, \ldots, w_{t-1})$, considering only a limited context of size n, that immediately precedes w_t. In neural language models the probability $P(w \mid c)$ of a word w to follow the context c is computed by a neural network. The neural network takes a context c and outputs the conditional probability $P(w \mid c)$ of every word w in the vocabulary V of the language:

$$P(w \mid c, \theta) = \frac{\exp(s_\theta(w, c))}{\sum_{w' \in V} \exp(s_\theta(w', c))}. \quad (2)$$

Here, $s_\theta(w, c)$ is an unnormalized score for the compatibility of w given the context c; $s_\theta(w, c)$ is computed via forward propagation of the context c through a neural network defined with the set of parameters θ. Note that $P(w \mid c)$ is computed by the normalized exponential (softmax) function in Equation 2, over the $s_\theta(w, c)$ scores for the entire vocabulary.

Parameters of the neural network are learned by training on a text corpus using gradient-descent based optimization algorithms to maximize the likelihood function L in Equation 3, on a given corpus T:

$$L(\theta) = \sum_{(t,c) \in T} P(t \mid c, \theta). \quad (3)$$

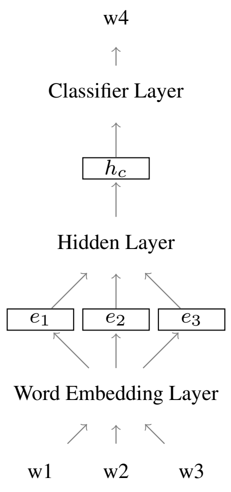


Fig. 4: Architecture of neural language models.

The first neural language model published is the Neural Network Language Model (NNLM) [17]. The common architecture shared by neural language models is depicted in Figure 4, with example input context $c = w_1, w_2, w_3$ and the word to predict being w_4, extracted from the observed sequence $c = w_1, w_2, w_3, w_4$. Although a probability distribution over the vocabulary V is computed, the word that should have the maximum probability is shown at the output layer, for illustration purposes.

The neural network takes one-hot vectors w_1, w_2, w_3 of the words in the context. The dimensionality of the one-hot vectors is $1 \times |V|$. The *embedding layer E* in Figure 4 is indeed a $|V| \times d$-dimensional matrix whose i-th row is the d-dimensional word embedding for the i-th word in the vocabulary. The embedding vector of the i-th word in the vocabulary is obtained by multiplying the one-hot vector of the word with the E matrix or simply extracting the i_{th} row of the embedding matrix. Consequently, high dimensional one-hot vectors of words w_1, w_2, w_3 are mapped to their d-dimensional embedding vectors e_1, e_2, e_3 by the embedding layer. Note that d is usually chosen to be in the range 100–500 whereas $|V|$ can go up to millions.

The *hidden layer* in Figure 4 takes the embedding vectors e_1, e_2, e_3 of the context words and creates a vector h_c for the input context. This layer differs between neural language model architectures. In NNLM [17] it is a non-linear neural network layer whereas in the CBOW model of `word2vec` [138], it is vector addition over word embeddings. In the Recurrent Neural Network Language Model (RNNLM) [137] the hidden context representation is computed by a recurrent neural network. Besides the hidden layer, the choice of context also differs among models. In [44] and in the CBOW model [138] context is defined by the words that surround a center word in a symmetric context. In the NNLM [17] and RNNLM [137] models, the context is defined by words that precede the target word. The Skip-gram [138] takes a single word as input and predicts words from a dynamically sized symmetric context window around the input word.

The *classifier layer* in Figure 4, which is composed of a weights matrix C of dimension $d \times |V|$ and a bias vector of dimension $|V|$, is used to compute $s_\theta(w,c)$ using Equation 4:

$$s_\theta(w,c) = h_c C + b. \tag{4}$$

To sum up, the neural network architecture for a Neural Language Model (NLM) is defined by $|V|$, d, the context type, the context size $|c|$ and the function in the hidden layer. The parameter set θ to be optimized includes the embedding matrix E, parameters from the hidden layer, the weights matrix C and the bias vector b of the classifier layer. The embedding layer E is treated as an ordinary layer of the network, its weights are initialized randomly and updated with back-propagation during training of the neural network.

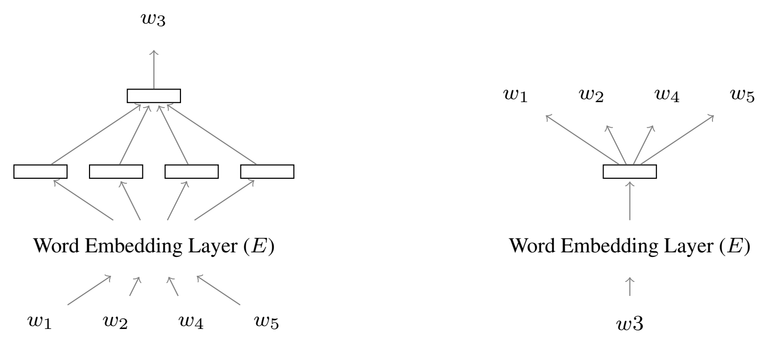
3.3.4 Efficient training of NLMs

As mentioned in our discussion of Equation 1, the output of a NLM is a normalized probability distribution over the entire vocabulary. Therefore, for each training sample (context pair (t,c)), it is necessary to compute the softmax function in Equation 2 and consider the whole vocabulary for computing the gradients of the likelihood function in back-propagation. This makes the training procedure computationally expensive and prevents the scalability of the models to very large corpora.

Several remedies for efficiently training NLMs have been introduced. The reader may refer to Chen et al [36] for a comparison of these remedies for the NNLM. The first group of remedies such as Hierarchical Softmax [147] and differentiated softmax [36] propose updates to the softmax layer architectures for efficient computation. The second approach, adopted by methods like Importance Sampling (IS) [18] and Noise Contrastive Estimation (NCE) [146], is to avoid the normalization by using modified loss functions to approximate the softmax. Collobert and Weston [44] propose the cost function in Equation 5, which does not require normalization over the vocabulary. The NLM is trained to compute higher s_θ scores for observed context-word pairs (c,t) compared to the negative samples constructed by replacing t with any other word w in V. The context is defined as the words in a symmetric window around the center word t.

$$\sum_{(t,c)\in T} \sum_{w \in V} \max(0, 1 - s_\theta(t,c) + s_\theta(w,c)) \tag{5}$$

Mnih and Teh [146] apply NCE [81] to NLM training. By using NCE, the probability density estimation problem is converted to a binary classification problem. A two-class training data set is created from the training corpus by treating the observed context-word pairs (t,c) as positive samples and k noisy pairs (t',c) constructed replacing t with a word t' sampled from the noise distribution q.



(a) Continuous Bag of Words (CBOW) (b) Skip-Gram

Fig. 5: Models of the `word2vec` framework.

3.3.5 `word2vec` *and GloVe*

Mikolov et al [138] introduce the Skip-gram and CBOW models that follow the NLM architecture with a linear layer for computing a distributed context representation. Figure 5 illustrates the architecture of `word2vec` models with context windows of size five. The CBOW model is trained to predict the center word of a given context. In the CBOW model, the hidden context representation is computed by the sum of the word embeddings. On the contrary, the Skip-gram model is trained to predict words that occur in a symmetric context window given the center word. The name Skip-gram is used since the size of symmetric context window is selected randomly from the range $[0, c]$ for each center word. Skip-gram embeddings are shown to outperform embeddings obtained from NNLMs and RNNLMs in capturing the semantic and syntactic relationships between the words.

`word2vec` owes its widespread adoption in the NLP and IR communities to its scalability. Efficient training of Skip-gram and CBOW models is achieved by hiearchical softmax [147] with a Huffman tree [138]. In follow-up work [140], Negative Sampling (NEG) is proposed for efficiently training the Skip-Gram model. NEG is a variant of NCE. NEG, differently from NCE, assumes the noise distribution q to be uniform and $k = |V|$ while computing the conditional probabilities. For a detailed discussion of NCE and NEG, see notes by Dyer [60]. It is crucial to note that the Skip-Gram with Negative Sampling (SGNS) departs from the goal of learning a language model and only embedding layer of the model is used in practice.

Subsampling frequent words is another extension introduced in [140] for speeding up training and increasing the quality of embeddings. Each word w_i in the corpus is discarded with probability $p(w_i)$, computed as a function of its frequency $f(w_i)$ in the corpus, given in Equation 6:

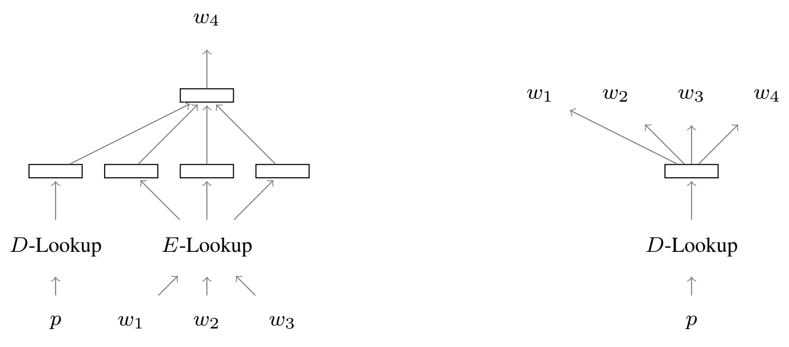
$$p(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}. \tag{6}$$

GloVe. GloVe [161] combines global context and local context in the training objective for learning word embeddings. In contrast to NLMs, where embeddings are optimized to maximize the likelihood of local contexts, GloVe embeddings are trained to fit the co-occurrence ratio matrix.

Context-counting vs. context-predicting. Levy et al [108] discuss that diluting frequent words before training enlarges the context window size in practice. Experiments show that the hyper-parameters about context-windows, like dynamic size and subsampling frequent words, have a notable impact on the performance of SGNS and GloVe [108]. Levy et al show that when these choices are applied to traditional DSMs, no consistent advantage of SGNS and GloVe is observed. In contrast to the conclusions obtained in [14], the success of context-predicting models is attributed to choice of hyper-parameters, which can also be used for context-counting DSMs, rather than to the neural architecture or the training objective.

3.3.6 Paragraph Vector

The PV [103] extends `word2vec` in order to learn representations for so-called *paragraph*, textual units of any length. Similar to `word2vec`, it is composed of two separate models, namely Paragraph Vector with Distributed Memory (PV-DM) and Paragraph Vector with Distributed Bag of Words (PV-DBOW). The architectures of PV-DM and PV-DBOW are illustrated in Figure 6. The PV-DBOW model is a Skip-Gram model where the input is a paragraph instead of a word. The PV-DBOW is trained to predict a sample context given the input paragraph. In contrast, the PV-DM model is trained to predict a word that is likely to occur in the input paragraph after the sample context. The PV-DM model is a CBOW model extended with a paragraph in the input layer and a document embedding matrix. In the PV-DBOW model, only paragraph embeddings are learned whereas in the PV-DM model word embeddings and paragraph embeddings are learned, simultaneously.



(a) PV-DM (b) PV-DBOW

Fig. 6: Models of the paragraph vector framework.

In Figure 6, p stands for the index of the input paragraph and w_1, w_2, w_3, w_4 represent the indices of the words in a contiguous sequence of words sampled from this paragraph. A sequence of size four is selected just for illustration purposes. Also, D represents the paragraph embedding matrix and E stands for the word embedding matrix. At the lowest layer, the input paragraph p is mapped to its embedding by a lookup in the D matrix. The hidden context representation is computed by summing the embeddings of the input words and paragraph, which is the same as in the CBOW model.

Paragraph vector models are trained on unlabelled *paragraph* collections. An embedding for each paragraph in the collection is learned at the end of training. The embedding for an unseen paragraph can be obtained by an additional inference stage. In the inference stage, D is extended with columns for new paragraphs; D is updated using gradient descent while other parameters of the model are kept fixed.

4 Taxonomy

To organize the material surveyed in this paper, we use a simple taxonomy. We classify publications based on the IR *task* to which neural models are applied and *how* neural network models are utilized for solving the target IR task. In the next subsections, we first detail the sub-categories of the *Task* and *How* features. Secondly, we provide a roadmap for the rest of the survey in Section 4.3.

4.1 Task

As mentioned previously, the majority of reviewed work is about textual IR tasks. Each of the tasks is concerned with a different *target textual unit* (TTU) or with different TTU pairs. We group publications on textual IR into five classes given in Table 1, considering a TTU or TTU pairs with different characteristics.

Table 1: TTU or TTU pairs for the textual IR tasks covered in the survey.

Task	**TTU or TTU pair**	**Relation**
Ad-hoc retrieval	Query-document	relevant
Query understanding	Query	similar
Sponsored search	Query-ad	relevant
Question answering	Question-answer	answer
Similar item retrieval	Document-document	similar

The hierarchy of tasks in the rest of the survey is as follows:

1. **Ad-Hoc retrieval**: Ad-hoc retrieval refers to a single search performed by a user: a single query, with no further interaction or feedback, on the basis of which an

IR system strives to return an accurate document ranking. This comprises of the following tasks:
   - Document ranking
   - Query expansion
   - Query re-weighting
   - Result diversification
   - Semantic expertise retrieval
   - Product search
2. **Query understanding**: This category includes IR tasks concerned with understanding the user intent in order to assist the user in typing queries or improving document retrieval. Publications here are focused on distributed representations of queries and finding similar queries that can better express user intent. We distinguish the following tasks:
   - Query suggestion
   - Query auto completion
   - Query classification
3. **Question answering**: This class includes tasks that are focused on retrieval of text segments that answers the user question. Answer segments may have different granularities, such as sentence, passage or even the entire document. Here we identify two tasks:
   - Answer sentence retrieval
   - Conversational agents
4. **Sponsored search**: This category includes tasks related to retrieval of ads relevant to user queries.
5. **Similar item retrieval**: This category includes tasks related to retrieving similar items of the same type as the query. There exist a large number of neural semantic compositionality models for textual similarity. To remain focused we limit ourselves to publications that focus on the retrieval effectiveness for similar item search. The following tasks are considered:
   - Related document search
   - Detecting text re-use
   - Similar Question Retrieval in CQA
   - Content-based recommendation

4.2 How

The *How* feature defines how neural models and distributed representations are adopted and utilized in an IR task. The methods can be grouped in two main classes: *Aggregate* and *Learn*. Availability of code and existing embeddings from `word2vec` [136] and GloVe [161] (see Appendix B.1) motivated the **Aggregate** approaches for Neural IR, especially for extending traditional IR models to integrate word embeddings. In contrast, the **Learn** category covers conceptually different approaches which directly incorporate word embeddings within NN models, reflecting a more significant shift toward pursuing *end-to-end* NN architectures in IR.

4.2.1 Aggregate

Publications in this category are focused on the following research question raised by Clinchant and Perronnin [42]: If we were provided with an embedding of words in a continuous space, how could we best use it in IR/clustering tasks? The methods in this category rely on pre-trained word embeddings as external resources in order to build or extend relevance matching functions. Existing work can be split into two sub-categories depending on how the embeddings are utilized:

Explicit Word embeddings are considered as building blocks for distributed TTU representations. Publications that follow this pattern treat a TTU as a *Bag of Embedded Words* (BoEW) or a set of points in the word embedding space. The BoEW is aggregated to build a single vector for the TTU. The most common aggregation method is averaging or summing the vectors of the terms in the TTU.

Implicit Here, one utilizes the vector similarity in the embedding space in language modeling frameworks without explicit computation of distributed representations for TTUs pairs. For instance, Zuccon et al [225] compute translation probabilities of word pairs in a translation language model retrieval framework with cosine similarity of Skip-Gram vectors.

4.2.2 Learn

This category covers work on learning *end-to-end* neural models for IR, specifically for semantic matching of TTU pairs. The neural models in this category are designed and trained to learn word embeddings and semantic compositionality functions for building distributed representations for TTUs or TTU pairs, simultaneously, from scratch, given only the raw text of TTUs. We observed that in some recent publications, the similarity or relevance function on top of distributed representations is also modeled with a neural network and learned, simultaneously.

In the rest of the survey, *Semantic Compositionality Network* (SCN) refers to a neural network that composes distributed TTU representations based on either the embeddings of words in the TTU or the one-hot term vector. Publications in this category mainly vary in the training objectives defined based on the distributed representation for TTUs. Four separate training objectives are observed in the reviewed work. Based on these objectives, we define the four sub-categories, namely

– *Learn to autoencode*,
– *Learn to match*,
– *Learn to predict* and
– *Learn to generate*,

all of which are detailed below.

Learn to autoencode. This category covers mostly relatively early work that relies on *auto-encoder* (see Section 3.2.2) architectures for learning TTU representations. As depicted in Figure 7, the training objective is to restore the input x based on the distributed representation h_x learned by the encoder. Generally, the encoder and decoder are mirrored neural networks with different architectures.

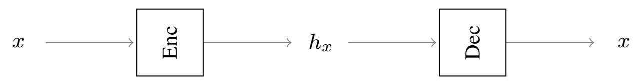


Fig. 7: Auto-encoder architecture. Note that the encoder is the Semantic Compositionality Network (SCN).

Learn to match. *Learning to match* [112] is the problem of learning a matching function $f(x,y)$ that computes a degree of similarity between two objects x and y from two different spaces X and Y. Given training data T composed of triples (x,y,r), *learning to match* is the optimization problem in Equation 7:

$$\arg\min_{f \in F} \sum_{(x,y,r)\in T} L(r, f(x,y)). \tag{7}$$

Here, L denotes a loss function between the actual similarity score r and the score predicted by the f function. Learn to match models introduce neural architectures for computing the relevance matching function f.

Learn to match models require similarity assessments for training. Since it is difficult to obtain large amounts of supervised data, click information in click-through logs are exploited to derive similarity assessments for query-document and query-ad pairs. If a pair (x,y) is associated with clicks, the objects are assumed to be similar; they are dissimilar in the absence of clicks. In the case of query-query pairs, co-occurrence in a session is accepted as similarity signal that can be extracted from query logs. We replace x and y of Equation 7 with q and d to represent a query and a document object, respectively. The document object can be any textual unit that needs to be matched against a query, such as a document, query or ad; (q,d^+) denotes a (query-clicked document) pair extracted from logs.

A training objective adopted by the majority of the publications in this category is to minimize the negative log likelihood function in Equation 8:

$$L = -\log \prod_{(q,d^+)} P(d^+ \mid q) \tag{8}$$

The likelihood of a document d given a query q, is computed by Equation 9 with a softmax over similarity scores of distributed representations:

$$P(d \mid q) = \frac{\exp(f(q,d))}{\sum_{d' \in D} \exp(f(q,d'))} \tag{9}$$

Here, f is a relevance matching function; L requires the computation of a probability distribution over the entire document collection D for each (q,d^+) pair. Since this is computationally expensive, D is approximated by a randomly selected small set of unclicked documents, similar to the softmax approximation methods for neural language models in Section 3.3.4.

There are two patterns of neural architectures for the relevance matching function f, namely *Representation Based* and *Interaction Based*, as illustrated in Figure 8. In *Representation Based* architectures, the query and document are independently run through mirrored neural models (typically this would be a Siamese architecture [28], in which weights are shared between the networks); relevance scoring is then performed by a model operating over the two induced representations. In contrast, in an *Interaction Based* architecture, one first constructs a joint representation of the query and document pair and then runs this joint input through a network. The architectures differ in their inputs and the semantics of the hidden/representational layers. Guo et al [78] point out that *Representation Based* architectures fail to catch exact matching signals which are crucial for retrieval tasks.

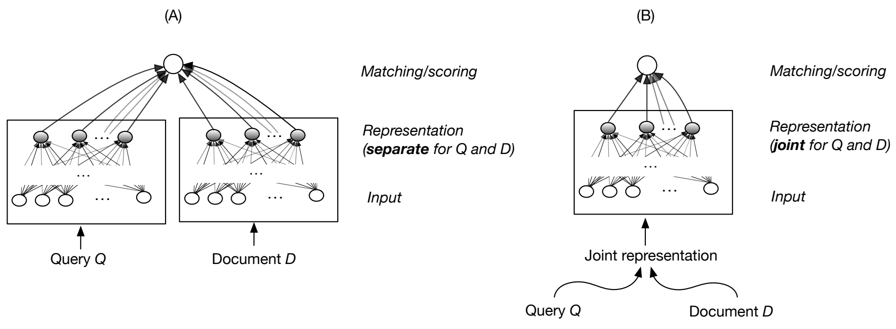


Fig. 8: Two basic neural architectures for scoring the relevance of a TTU pair, query and document for illustrative purposes. (A) *Representation Based* (B) *Interaction Based*. This figure is inspired by Figure 1 in Guo et al. [78].

In *Representation Based* architectures, input vector representations of q, d are mapped into distributed representations h_q, h_d by the *Semantic Compositionality Network* (SCN) and the similarity score of the objects is computed by the similarity of the distributed representations. Usually, cosine similarity is used to compute the similarity of representation vectors created by the SCN, as given in Equations 10 and 11:

$$h_q = SCN(q), h_d = SCN(d) \tag{10}$$

$$f(q,d) = \frac{h_q \cdot h_d}{|h_q|\,|h_d|}. \tag{11}$$

In Figure 9, the training objective in Equation 8 is illustrated on a *Representation Based* architecture with four randomly selected non-relevant documents. Let d_1 be a relevant document and d_2, d_3, d_4, d_5 non-relevant documents for the query q. The Siamese network is applied to compute similarity scores for each pair (q, d_i). The target values for the output nodes of the entire network is forced to be [1.0, 0.0, 0.0, 0.0, 0.0] during training.

Learn to predict. The success of word-based neural language models has motivated models for learning representations of larger textual units from unlabelled data [84, 103]. As mentioned previously, neural language models are context-predicting distributional semantic models (DSMs). Learn to predict models rely on an extension of the context-prediction idea to larger linguistic units, which is that *similar textual units*

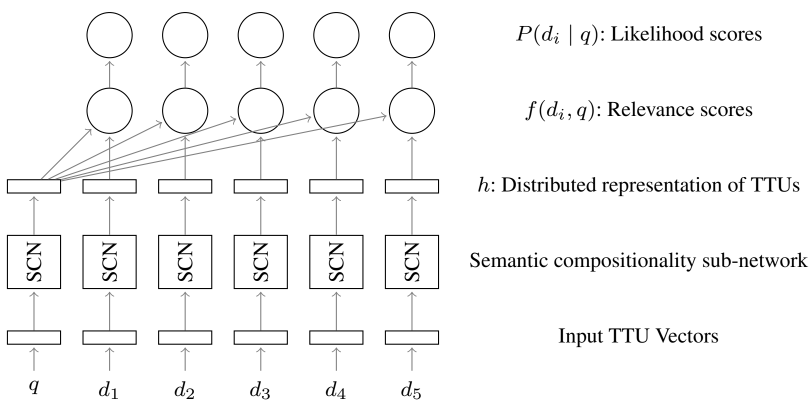

$P(d_i \mid q)$: Likelihood scores
$f(d_i, q)$: Relevance scores
h: Distributed representation of TTUs
Semantic compositionality sub-network
SCN
Input TTU Vectors
q
d_1
d_2
d_3
d_4
d_5

Fig. 9: Architecture of *Representation Based* Learn to match models adapted from [55].

occur in similar contexts. For instance, sentences within a paragraph and paragraphs within a document are semantically related. The organization of textual units—of different granularity—in corpora can be used to learn distributed representations. Contextual relationships of textual units are exploited to design training objectives similar to neural language models.

We refer to models that rely on the extended distributional semantics principle to obtain distributed TTU representations as Learn to predict models, in accordance with the *context-predicting* label used for neural language models. Learn to predict models are neural network models trained using unlabelled data to maximize the likelihood of the context of a TTU. Among the models reviewed in [84], Skip-thought Vector [98] and Paragraph Vector (PV) [103] are successful representatives of the *learn to predict context* idea. The training objective of the Skip-thought Vector is to maximize the likelihood of the previous and the next sentences given an input sentence. The context of the sentence is defined as its neighboring sentences. For details of the PV model, see Section 3.

The main context of a textual unit consists of the words it contains. Containment should be seen as a form of co-occurrence and the content of a document is required to define its textual context. Two documents are similar if they contain similar words. Besides the content, temporal context is useful for defining TTU contexts. For instance, the context of the query can be defined by other queries in the same session [76]. Finally, the context of a document is defined by joining its content and the neighboring documents in a document stream in [58].

Learn to generate. This category covers work in which a synthetic textual unit is generated based on the distributed representation of an input TTU. Studies in this category are motivated by successful applications of RNNs, LSTM networks, and encoder-decoder architectures, illustrated in Figure 10, to sequence-to-sequence learning [72] tasks such as machine translation [39] and image captioning [202]. In these applications, an input textual or visual object is encoded into a distributed representation with a neural network and a target sequence of words—a translation in the target language or a caption—is generated by a decoder network.

$$\begin{tikzcd} x \arrow[r] & \boxed{\text{Enc}} \arrow[r] & h_x \arrow[r] & \boxed{\text{Dec}} \arrow[r] & y \end{tikzcd}$$

Fig. 10: Encoder decoder architecture. Note that the encoder is the SCN.

4.3 Roadmap

In Table 2, we provide a classification of reviewed work with respect to the features *Task* and *Approach*.

Table 2: Classification of reviewed work.

Section	**Task**	**Approach**		**Publications**
5	**Ad-hoc retrieval**			
5.1.1	Document ranking	Aggregate	Explicit	[26, 42, 66, 145, 150, 189]
			Implicit	[65, 165, 168, 225]
5.1.2	Document ranking	Learn	Learn to match	[79, 89, 114, 152, 156, 157, 175, 176, 207]
			Learn to predict	[1, 58]
			Learn to generate	[116]
5.2	Query re-weighting		Explicit	[222]
5.3.1	Query expansion	Aggregate	Explicit	[3, 5, 164, 169]
			Implicit	[57, 212, 213]
5.3.2		Learn	Learn to autoencode	[80]
			Learn to match	[181]
5.4.1	Diversification	Aggregate	Explicit	[153]
5.4.2		Learn	Learn to match	[200]
5.5.1	Expertise retrieval	Learn	Learn to predict	[188]
5.6.1	Product search	Learn	Learn to match	[187]

6	**Query tasks**			
6.1	Query auto completion	Aggregate	Implicit	[29]
		Learn	Learn to match	[143, 144]
6.2.1	Query suggestion	Learn	Learn to generate	[182]
6.3	Query classification	Aggregate	Explicit	[213]
6.4	Proactive search	Learn	Learn to generate	[123]
7	**Question answering**			
7.1.1	Answer sentence retrieval	Learn	Learn to match	[174, 183, 191, 203, 211]
7.2.1	Conversational agents	Learn	Learn to match	[203]
8	**Sponsored search**			
8.1	Sponsored search	Learn	Learn to match	[8, 214, 215]
8.2		Learn	Learn to predict	[75, 76, 217]
9	**Similar item retrieval**			
9.1.1	Related document search	Aggregate	Explicit	[96, 101]
9.1.2		Learn	Learn to autoencode Learn to predict	[171] [48, 58, 103]
9.2	Detecting text reuse	Aggregate	Explicit	[216]
9.3.1	Similar Question Retrieval	Aggregate	Explicit	[223]
9.3.2		Learn	Learn to match Learn to generate	[107] [107]
9.4.1	Content-based recommendation	Aggregate	Explicit	[130]
9.4.2		Learn	Learn to match	[69]
10	**Non-textual/Behavioural**			[25, 217]

In Section 5, 6, 7, 8 and 9 below we survey work on neural models for Ad-hoc retrieval, query understanding, question answering, sponsored search and similar item retrieval, respectively.

5 Ad-hoc Retrieval

In this section we survey work on neural models for ad-hoc retrieval tasks. We devote a subsection per task listed in Table 2 under ad-hoc retrieval and follow the *How* feature as explained in Section 4 for organizing the subsections.

5.1 Document ranking

5.1.1 Aggregate

In this section, we present publications that rely on pre-trained word embeddings for ad-hoc retrieval under the Implicit and Explicit categories.

Explicit. Clinchant and Perronnin [42] present the earliest work using word embeddings in IR. The context-counting model Latent Semantic Indexing (LSI) [53] is used to induce word embeddings, which are then transformed into fixed-length Fisher Vectors (FVs) via Jaakkola et al [90]'s Fisher Kernel (FK) framework. The FVs are then compared via cosine similarity for document ranking. Experiments on ad-hoc search using Lemur are reported for three collections: TREC ROBUST04, TREC Disks 1&2, and English CLEF 2003 Ad-hoc. While results show improvements over standard LSI on all three collections, standard TF-IDF performs slightly better on two of the three collections, and Divergence From Randomness (DFR) [4] performs far better on all collections. The authors note that thse "results are not surprising as it has been shown experimentally in many studies that latent-based approaches such as LSI are generally outperformed by state-of-the-art IR models in Ad-Hoc tasks."

Vulic and Moens [189] are the first to aggregate word embeddings learned with a context-predicting distributional semantic model (DSM). Query and document are represented as a sum of word embeddings learned from a pseudo-bilingual document collection with a Skip-gram model (Ganguly et al [66] discusses why such a representation for documents could be noisy). Each document pair in a document-aligned translation corpus is mapped to a pseudo-bilingual document by merging source and target documents, removing sentence boundaries and shuffling the complete document. Owing to these shuffled pseudo-bilingual documents, words from the source and target language are mapped to the same embedding space. This approach for learning bilingual word embeddings is referred to as Bilingual word Embeddings Skip-Gram (BWESG). Documents are ranked by the cosine similarity of their embedding vector to the query vector. The query-document representations are evaluated both on cross-lingual and mono-lingual retrieval tasks. For the monolingual experiments, ranking the proposed distributed representations outperforms ranking LDA representations. While this approach is simple and able to benefit from a potentially vast body of comparable vs. parallel corpora for training, random shuffling loses the precise local context windows exploited by `word2vec` training (which parallel corpora would provide), effectively setting context window size to the length of the entire document. The approach and experimental setup otherwise follows the monolingual version of the authors' method. Cross-lingual results for CLEF 2001–2003 Ad-hoc

English-Dutch show that the embedding approach outperforms the unigram baseline and is comparable to the LDA baseline. As with monolingual results, the mixture models perform better, with a cross-lingual three way mixture of unigram, LDA, and embedding. No experiments are reported with parallel corpora for comparison, which would be interesting for future work.

Mitra et al [145], Nalisnick et al [150] propose the Dual Embedding Space Model (DESM), writing that "a crucial detail often overlooked when using `word2vec` is that there are two different sets of vectors ... IN and OUT embedding spaces [produced by `word2vec`].... By default, `word2vec` discards W_{OUT} at the end of training and outputs only W_{IN}..." In contrast, the authors retain both input and output embeddings. Nalisnick et al [150] point out that within the same embedding space, either IN or OUT, the neighbors are functionally similar words. However, the neighbors of a word represented with its IN embedding vector in the OUT space, are topically similar words. Topically similar words are likely to co-occur in a local context whereas functionally similar words are likely to occur in similar contexts. For instance, for the term *harvard*, the terms *faculty*, *alumni*, and *graduate* are topically similar terms and *yale*, *stanford*, *cornell* are functionally similar terms. Motivated by this observation, Nalisnick et al [150] propose the DESM.

In the DESM, query terms are mapped to the IN space and document words to the OUT space. Documents are embedded by taking an average (weighted by document term frequency) over embedding vectors of document terms. Query-document relevance is computed by average cosine similarity between each query term and the document embedding. The authors induce word embeddings via `word2vec` CBOW only, though they note that Skip-gram embeddings could be used interchangeably. Experiments with ad-hoc search are carried out on a proprietary Web collection using both explicit and implicit relevance judgments. In contrast with DSSM [89], full web page documents are indexed instead of only page titles. DESM's IN and OUT embedding space combinations are compared to baseline retrieval by BM25 and latent semantic analysis (LSA) [53]. Out Of Vocabulary (OOV) query terms are ignored for the DESM approach but retained for the baselines. Results show "DESM to be a poor standalone ranking signal on a larger set of documents," so a re-ranking approach is proposed in which the collection is first pruned to all documents retrieved by an initial Bing search, and then re-ranked by DESM. Re-ranking results show improvements over baselines, especially on the implicit feedback test set, with best performance obtained when word embeddings are trained on queries and using IN-OUT embedding spaces in document ranking. The authors surmise that training on queries performs better due to users tending to include only significant terms from their queries. Learned word embeddings are shared online (see Appendix B.1).

In [66], documents are modelled as a mixture distribution that generates the observed terms in the document. Ganguly et al [66] estimate this distribution with k-means clustering of embeddings of the terms in the document. The likelihood of a query to be generated by the document is computed by the average inter-similarity of the set of query terms to the centroids of clusters in the document, in the word embedding space. For efficiency, the global vocabulary is clustered using `word2vec` embeddings in advance and document specific clusters are created by grouping the terms according to their global cluster ids. A centroid-based query likelihood function is evaluated in combination with language modeling with Jelinek-Mercer smoothing on the TREC 6-7-8 and TREC Robust data sets. A significant improvement is observed by the inclusion of word embedding-based query-likelihood function over the standalone language model (LM) baseline.

Boytsov et al [26] consider the cosine similarity between the averaged word embedding vectors of the query and document as a similarity function, in k-NN based retrieval. The authors propose to replace the traditional term-based search by k-NN based retrieval. Exact k-NN search fails to be efficient yet approximation algorithms, such as Small-World Graph and Neighbourhood Approximations (NAPP), are proposed as remedies. Experiments are performed on the Yahoo Answers and Stack Overflow data sets. The cosine-similarity between averaged word embeddings is found to be less effective than BM25 and cosine similarity of TF-IDF vectors.

Implicit. Zuccon et al [225] propose a Neural Translation Language Model (NLTM), which integrates word embeddings into Berger and Lafferty [21]'s classic translation model approach to query-likelihood IR. They estimate the translation probability between terms as the cosine similarity of the two terms divided by the sum of the cosine similarities between the translating term and all of the terms in the vocabulary. Previous state-of-the-art translation models use mutual information (MI) embeddings to estimate translation probabilities. Experiments evaluating NLTM on ad-hoc search are reported on the TREC datasets AP87-88, WSJ87-92, DOTGOV, and MedTrack. Results indicate that NLTM provides moderate improvements over the MI and classic TM systems, based on modest improvements to a large number of topics, rather than large differences on a few topics. Sensitivity analysis of the various model hyper-parameters for inducing word embeddings shows that manipulations of embedding dimensionality, context window size, and model objective (CBOW vs Skip-gram) have no consistent impact upon NLTM's performance vs. the baselines. Regarding the choice of training corpus for learning embeddings vs. search effectiveness, although effectiveness typically appears highest when embeddings are estimated using the same collection in which search is to be performed, the differences are not statistically significant. Source code and learned embeddings are shared online (see Appendix B.1).

Rekabsaz et al [165] investigate a set of existing models including Pivoted Document Normalization, BM25, BM25 Verboseness Aware, Multi-Aspect TF, and Language Modeling by generalizing the translation model to the probabilistic relevance framework. They extend the translation models in Pseudo-Relevance (PR) framework by integrating the effect of changing term frequencies. For experimental evaluation 6 test collections are used: a combination of TREC 1 to 3, TREC-6, TREC-7, and TREC-8 of the AdHoc track, TREC-2005 HARD track, and CLEF eHealth 2015 Task 2 User-Centred Health Information Retrieval. In terms of baseline models, Rekabsaz et al use (1) the original version of the extended models; (2) the query expanded model using the logarithm weighting model; and (3) the query expanded model with the normalization over the expanded terms. The evaluation metrics are MAP and NDCG@20. Experimental results show that the newly proposed models achieve state-of-the-art results.

Cosine similarity of `word2vec` embeddings is used in a similar way in the Generalized Language Model (GLM) [65]. Ganguly et al [65] propose GLM for integrating word embeddings with the query-likelihood language modelling. Semantic similarity between query and document/collection terms is measured by cosine similarity between word embeddings induced via `word2vec` CBOW. The authors frame their approach in the context of classic *global* vs. *local* term similarity, with word embeddings trained without reference to queries representing a global approach akin to the Latent Dirichlet Allocation (LDA) of Wei and Croft [193]. Like Rekabsaz et al [164] and Zuccon et al [225], the authors build on Berger and Lafferty [21]'s "noisy channel" translation model. Smoothing of mixture model components resembles classic cluster-based LM smoothing of Liu and Croft [119]. Ad-hoc search results reported for TREC 6-8 and Robust using Lucene show improvements over both unigram query-likelihood and LDA. However, the parameters appear to be tuned on the test collections, and LDA results are much lower than Wei and Croft [193]'s, which the authors hypothesize is due to training LDA on the entire collection rather than on collection subsets. The authors do not compare their global approach vs. local pseudo-relevance feedback (PRF) [102], which prior work has shown to outperform LDA [209]; while typically a query-time technique, it can be approximated for greater efficiency [34].

Roy et al [168] proposes a relevance feedback model that employs word embeddings, based on the intuition that adding word embeddings can result in semantic composition of individual words. They buid a kernel functions around the query word embeddings that are treated as data points. Then they estimate the kernel density of the probability density function that generates the query word embeddings. The query term can control the shape of the estimated probability density function. In this way, they develop a relevance feedback model that integrates semantic relationships and compositionality of words. Documents are then ranked according to their KL divergence from the estimated kernel density function. They evaluate their approach on TREC 6–8 and Robust adhoc news retrieval and the TREC 9-10 WT10G web retrieval test collections. They compare their method with the standard relevance model that only uses statistical co-occurrences between query terms and words in top ranked documents, and find that it significantly outperforms the baseline. A future direction might be exploring larger units embeddings such as sentence and paragraph embeddings.

5.1.2 Learn

Learn to match

Representation Based. The Deep Structured Semantic Model (DSSM) [89] is the earliest neural *Representation Based* Learn to match model for document ranking. DSSM was one of the pioneering models incorporating click-through data in deep NNs. It has been built on by a variety of others [143, 144, 175, 176, 207]. Other work in this category is either an architectural variant of DSSM with different SCNs or they propose novel ways of using distributed representations by DSSM variants in order to improve retrieval effectiveness. Architectural variants such as the Convolutional Latent Semantic Model (CLSM) [175] and LSTM Deep Structured Semantic Model (LSTM-DSSM) [156, 157] differ from DSSM in the input representations and architecture of the SCN component. Besides architectural variants with different SCN types, Nguyen et al [152] propose two high level views of how to incorporate a knowledge base (KB) graph into DSSM [89]. Li et al [114] utilize distributed representations produced by DSSM and CLSM in order to re-rank documents based on in-session contextual information. Ye et al [207] question the assumptions about clicked query-document pairs in order to derive triplets for training variants of the DSSM models.

The SCN of the DSSM model is composed of a deep neural network with three non-linear layers placed on top of a word hashing layer. Documents are indexed only by title text rather than the entire body text. The query and the document are first modeled as two high dimensional term vectors (i.e., a bag-of-words representation). Each term vector is mapped to a trigram vector by the word hashing layer, in order to cope with a large vocabulary. For instance, the word *vector* is mapped to $\{$*.ve, vec, ect, cto, tor ,or.*$\}$ where the dot sign is used as the start and end character. This low-dimensional trigram vector is then considered as input to the SCN. The vocabulary size is reduced from 500K to 30K by replacing each term with its letter trigrams. Trigram hashing also helps to address out of vocabulary (OOV) query terms not seen in training data. The authors do not discuss how to mitigate hashing collisions; while they show that such collisions are relatively rare (e.g., 0.0044% for the 500K vocabulary size), this stems in part from indexing document titles only.

Convolutional Deep Structured Semantic Models (C-DSSM) [176] extend DSSM by introducing a CNN with max-pooling as the SCN in the DSSM architecture (C-DSSM). It first uses word hashing to transform each word into a vector. A convolutional layer then projects each word vector within a context window to a local contextual feature vector. It also incorporates a max-pooling layer to extract the most salient local features to form a fixed-length global feature vector for queries and web documents. The main motivation for the max-pooling layer is that because the overall meaning of a sentence is often determined by a few key words, simply mixing all words together (e.g., by summing over all local feature vectors) may introduce unnecessary divergence and hurt overall semantic representation effectiveness. This is a key difference between DSSM and C-DSSM.

Both DSSM [89] and C-DSSM [176] fail to capture contextual information of queries and documents. To address this, Shen et al [175] propose a Convolutional Latent Semantic Model (CLSM) built on top of DSSM. CLSM captures contextual information by a series of projections from one layer to another in a CNN architecture [104]. The first layer consists of a word n-gram followed by a letter trigram layer where each word n-gram is composed of its trigram representation, a form of word hashing technique developed in DSSM. Then, a convolution layer transforms these trigrams into contextual feature vectors using the convolution matrix W_c, which is shared among all word n-grams. Max-pooling is then performed against each dimension on a set of contextual feature vectors. This generates the global sentence level feature vector v. Finally, using the non-linear transformation $\tanh$ and the semantic projection matrix W_s, they compute the final latent semantic vector for the query/document. The parameters W_c and W_s are optimized to maximize the same loss function used by Huang et al [89] for DSSM. Even though CLSM introduces word n-grams to capture contextual information, it suffers from the same problems as DSSM, including scalability. For example, CLSM performs worse when trained on a whole document than when trained on only the document title Guo et al [78].

To sum up the architectural variants, DSSM takes a term vector of the textual unit and treats it as a bag of words. In contrast, C-DSSM, CLSM and LSTM-DSSM take a sequence of one-hot vectors of terms and treat the TTUs as a sequence of words. CLSM includes a convolutional neural network and LSTM-DSSM includes an LSTM network as SCN. The word hashing layer is common to all of these models. DSSM, CLSM and LSTM-DSSM are evaluated on large-scale data sets from Bing in [89, 175, 176]. In [89], DSSM is trained on document title-query pairs and is shown to outperform the Word Translation Model, BM25, TF-IDF and Bilingual Topic Models with posterior regularization in terms of NDCG at cutoff values 1, 3 and 10. However, later work [78] performs two further experiments with DSSM: indexing only document titles vs. indexing entire documents (i.e., full-text search). Guo et al [78]'s results indicate that full-text search with DSSM does not perform as well as traditional IR models. In [175], CLSM is shown to be more effective with document titles. Finally, Shen et al [175] and Palangi et al [157] report that CLSM outperforms DSSM and LSTM-DSSM outperforms CLSM when document titles are used instead of full documents.

Liu et al [120] propose a neural model with multi-task objectives. The model integrates a deep neural network for query classification and the DSSM model for web document ranking via shared layers. The word hashing layer and semantic representation layer of DSSM are shared between the two models. The integrated network comprises separate task-specific semantic representation layers and output layers for two different tasks. A separate cost function is defined for each task. During training, in each iteration, a task is selected randomly and the model is updated only according to the selected cost function. The proposed model is evaluated on large-scale commercial search logs. Experimental results show improvements by the integrated model over both standalone deep neural networks for query classification and a standalone DSSM for web search ranking.

Li et al [114] utilize distributed representations produced by DSSM and CLSM in order to re-rank documents based on in-session contextual information. Similarity of query-query and query-document pairs extracted from the session context is computed using DSSM and CLSM vectors. These similarity scores are included as additional features to represent session context in a context-aware learning to rank framework. They evaluate XCode (internally developed by the authors), DSSM [89], and C-DSSM [176] as models for deriving these contextual features and find that DSSM offers the highest performance, followed by C-DSSM. Though they expected C-DSSM to offer the highest performance, they note that C-DSSM could only be trained on a small dataset with bounded size, in contrast to DSSM, which could be trained on a larger dataset. Additionally, they note that observed performance differences may be due to imperfect tuning of model parameters, such as sliding window size for C-DSSM. Nevertheless, contextual features derived using both DSSM and C-DSSM offer performance benefits for re-ranking.

Nguyen et al [152] propose two high level views of how to incorporate a knowledge base (KB) graph into a ranking model like DSSM [89]. To the best of our knowledge, this is one of the first IR studies that tries to incorporate KBs into a deep neural structure. The authors' first model exploits KBs to enhance the representation of a document-query pair and its similarity score by exploiting concept embeddings learned from the KB distributed representation [63, 201] as input to deep NNss like DSSM. The authors argue that this hybrid representation of the distributional semantics (namely, word embeddings) and the symbolic semantics (namely, concept embeddings taking into account the graph structure) would enhance document-query matching. Their second model uses the knowledge resource as an intermediate component that helps to translate the deep representation of the query towards the deep representation of documents for an ad-hoc IR task. Strong empirical evidence is still needed to demonstrate that adding a KB does in indeed benefit the deep neural architecture for capturing semantic similarity.

Ye et al [207] question assumptions about clicked query-document pairs in order to derive triplets for training DSSM variant models. According to Ye et al [207] these three assumptions are: (i) each clicked query-document pair is equally weighted; (ii) each clicked query-document pair is a 100% positive example; and (iii) each click is solely due to semantic similarity. The authors relax these assumption and propose two generalized extensions to DSSM: GDSSM1 and GDSSM2. While DSSM models the probability of a document being clicked given a query and the semantic similarity between document and query, GDSSM1 uses more information in its loss function, incorporating the number of users seeing and the proportion clicking for each query-document pair. GDSSM2 conditions on lexical similarity in addition to semantic similarity.

Interaction Based. Guo et al [78]'s Deep Relevance Matching Model (DRMM) is one of the first NN IR models to show improvement over traditional IR models (though the comparison is against bag-of-words approaches rather than term proximity baselines). The authors hypothesize that deep learning methods developed and/or commonly applied in NLP for semantic matching may not be suited for ad-hoc search, which is most concerned with relevance matching. They articulate three key differences they perceive between semantic and relevance matching:

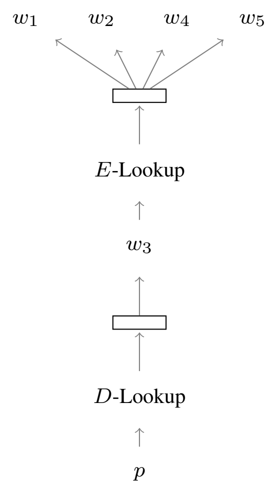
1. Semantic matching looks for semantic similarity between terms; relevance matching puts more emphasis on exact matching.
2. Semantic matching is often concerned with how composition and grammar help to determine meaning; varying importance among query terms is more crucial than grammar in relevance matching.
3. Semantic matching compares two whole texts in their entirety; relevance matching might only compare parts of a document to a query.

The raw inputs to the DRMM are term embeddings. The first layer consists of *matching histograms* of each query term's cosine similarity scores with each of the document terms. On top of this histogram layer is an NN outputting a single node for each query term. These outputs are multiplied by query term importance weights calculated by a *term gating network* and then summed to produce the final predicted matching score for the query/document pair. The whole network is trained using a margin ranking loss function. Ad-hoc search experiments are reported on TREC Robust04 and ClueWeb09-Cat-B. Baselines include: (i) traditional retrieval models such as BM25 and query likelihood; (ii) representation-focused NN models, including DSSM [89] and C-DSSM [176] (indexing titles vs. full documents), ARC-I [88]; and (iii) interaction-focused NN models, such as ARC-II [88] and MatchPyramid [159]. OOV terms are represented by random vectors (as in [95]), effectively allowing only exact matching. Results show that as the CBOW dimension increases (50, 100, 300 and 500), the performance of DRRM first increases then slightly drops. DRRM using IDF and Log-Count Histogram (LCH) also significantly outperforms all baselines. In addition, none of the representation-focused and interaction-focused baselines are competitive with traditional retrieval models, supporting the authors' hypothesis that deep matching models focused on semantic matching may not be well-suited to ad-hoc search.

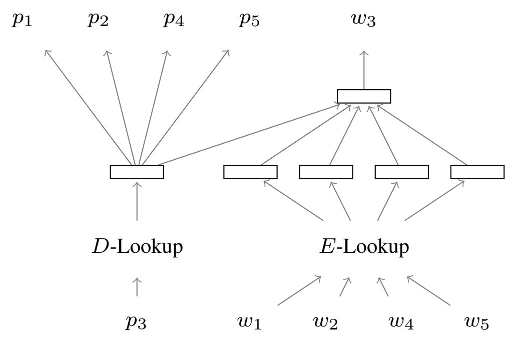
5.1.3 Learn to predict

Ai et al [1, 2] investigate the use of the PV-DBOW model as a document language model for retrieval. Three shortcomings of the PV-DBOW model are identified and an extended Paragraph Vector (PV) model is proposed with remedies for these shortcomings. First, the PV-DBOW model is found to be biased towards short documents due to overfitting in training and the training objective is updated with L2 regularization. Secondly, the PV-DBOW model trained with NEG implicitly weights terms with respect to Inverse Corpus Frequencies (ICF) which has been shown to be inferior to Inverse Document Frequency (IDF) in [166]. A document frequency based negative sampling strategy, which converts the problem into factorization of a shifted TF-IDF matrix, is adopted. Thirdly, the two layer PV-DBOW architecture depicted in Figure 11a is introduced since word substitution relations, such as the relation in *car-vehicle*, *underground-subway* pairs, are not captured by PV-DBOW. The Extended Paragraph Vector (EPV) is evaluated in re-ranking the set of top 2,000 documents retrieved by a query likelihood retrieval function. Document language models based on EPV and LDA are compared on TREC Robust04 and GOV2 data sets. An EPV-based model yields higher effectiveness scores than the LDA-based model.

The Hierarchical Document Vector (HDV) [58] model extends the PV-DM model to predict not only words in a document but also its temporal neighbors in a document stream. The architecture of this model is depicted in Figure 11b with a word context and document context size of five. There, w_1, w_2, w_3, w_4, w_5 represent a sample context of words from the input paragraph; p_1, p_2, p_3, p_4, p_5 represent a set of documents that occur in the same context. In HDV, the content of the documents in the temporal context also contributes to the document representation. Similar to PV-DM, the words and documents are mapped to d-dimensional embedding vectors. Djuric et al [58] point out that words and documents are embedded in the same space and this makes the model useful for both recommendation and retrieval tasks including document retrieval, document recommendation, document tag recommendation and keyword suggestion. Given a keyword, titles of similar documents in the embedding space are presented to give an idea of the effectiveness of the model on the ad-hoc



(a) Two Layer PV-DBOW [1].



(b) HDV [58] (and `context-content2vec` [76, Section 8.2]).

Fig. 11: Learn to predict context models for document retrieval.

retrieval task. However, a quantitative evaluation is not provided for the document retrieval and keyword suggestion tasks.

5.1.4 Learn to generate

Lioma et al [116] ask whether it is possible to generate relevant documents given a query. A character level LSTM network is optimized to generate a synthetic document. The network is fed with a sequence of words constructed by concatenating the query and context windows around query terms in all relevant documents for the query. For each query, a separate model is generated and a synthetic document is generated for the same query with the learned model. The synthetic document was evaluated in a crowdsourcing setting. Users are provided with four word clouds that belong to three known relevant documents and the synthetic document. Each word cloud is built by selection of top frequent terms from the document. Users are asked to select the most relevant word cloud. Author report that the word cloud of the synthetic document ranked the first or second for most of the queries. Experiments were performed on the TREC Disks 4, 5 test collection with title-only queries from TREC 6, 7, 8.

5.2 Query re-weighting

5.2.1 Aggregate

Implicit Palakodety and Callan [155]'s work on query expansion is based on word embeddings; although originally proposed for result merging in federated web search, we note it here since it introduces the idea of an *importance vector* in the query re-weighting method in [222]. Palakodety and Callan [155] represent a query by the mean vector of query term embeddings and k-nearest neighbors of the query vector are selected to expand the query. Expansion terms are re-weighted with respect to their distance to the query vector. A query term that is more distant to the query vector is assumed to contain more information and is assigned a higher weight. Query re-weighting is modelled as a linear regression problem from importance vectors to term weights in [222]. The importance vector, which is the offset between the query vector and the term vector, is used as the feature vector for the query term. A linear regression model is trained using the ground-truth term weights computed by term recall weights which is the ratio of relevant documents that contain the query term t to the total number of relevant documents to the query q. Weighted queries based on a learned regression model are evaluated in a retrieval setting and compared against the LM and BM-25 models using the data sets ROBUST04, WT10g, GOV2, ClueWeb09B. Two variants of the model, DeepTR-BOW and DeepTR-SD, are compared against unweighted queries, sequential dependency models and weighted sequential dependency models. Statistically significant improvements are observed at high precision levels and throughout the rankings compared to the first two methods.

5.3 Query expansion

5.3.1 Aggregate

Explicit. ALMasri et al [3] propose a heuristic method VEXP for term-by-term query expansion using embedding vectors. For each query term, the most similar terms in the embedding space are added to the query. Expansion terms are weighted in proportion to their frequency in the expanded query. Experiments with ad-hoc search use Indri on four CLEF medical test collections: Image2010-2012 (short documents and queries, text-only) and Case2011 (long documents and queries). Baselines include pseudo-relevance feedback [102] and mutual information. They evaluate both CBOW and Skip-gram `word2vec` embeddings (using default dimensionality and context window settings) but present only Skip-gram results, noting "there was no big difference in retrieval performance between the two." The authors consider adding a fixed number of 1–10 expansion terms per query term and also compare two smoothing methods: linear Jelineck-Mercer vs. Dirichlet. Results show VEXP achieves higher Mean Average Precision (MAP) than other methods.

Roy et al [169] propose a set of query expansion methods, based on selecting k nearest neighbors of the query terms in the word embedding space and ranking these terms with respect to their similarity to the whole query. For each nearest neighbor, they calculate the average cosine similarity vs. all query terms, selecting the top-K terms according to average cosine score. The second approach (reduces the vocabulary space considered by k-NN by only considering terms appearing in the top M pseudo-relevant documents retrieved by the query. In the third approach, an iterative (and computationally expensive) pruning strategy is applied to reduce the number of nearest neighbors, assuming that nearest neighbors are similar to one another. All expansion methods yielded lower effectiveness scores than a statistical co-occurrence based feedback method, in experiments with the TREC 6, 7 and 8 and TREC Robust data set and the LM with Jelinek Mercer smoothing as the retrieval function.

Amer et al [5] investigate word embeddings for personalized query expansion in the domain of social book search.[4] While personalized query expansion is not new [32, 37], use of word embeddings for personalization is novel. The proposed method consists of three steps: user modeling, term filtering and selection of expansion terms. A user is modeled as a collection of documents, and query terms are filtered to remove adjectives, which may lead to noisy expansion terms. For each remaining query term, similar to Roy et al [169]'s k-NN approach, the top-K most similar terms are selected based on cosine similarity in the embedding space. Evaluation on the social book search task compares `word2vec` trained on personalized vs. non-personalized training sets. However, results show that expansion via word embeddings strictly hurts performance vs. no expansion at all, in contrast to [144, 169]. This may stem from training `word2vec` embeddings only on social book search documents. Results further suggest that personalized query expansion does not provide improvements over non-personalized query expansion using word embedding. The authors postulate that sparse training data for personalization is the main problem here.

Rekabsaz et al [164] recommend choosing similar terms based on a global similarity threshold rather than by k-NN because some terms should naturally have more similar terms than others. They choose the threshold by setting it such that for any term, the expected number of related terms within the threshold is equal to the average number of synonyms over all words in the language. This method avoids having to constrain or prune the k-NN technique as in [169]. They use multiple initializations of the `word2vec` Skip-gram model to produce a probability distribution used to calculate the expected cosine similarity, making the measure more robust against noise. Experiments on TREC 6–8 and HARD 2005 incorporate this threshold-setting method into a translation language model [21] for ad-hoc retrieval and compare against both a language model baseline and a translation language model that uses k-NN to select similar words. The threshold-based translation language model achieves the highest Mean Average Precision (MAP).

Implicit. In [57, 212], word embeddings are used for defining a new query language model (QLM). A QLM specifies a probability distribution $p(w \mid q)$ over all terms in the vocabulary. In query expansion with language modeling, the top m terms w that have the highest $p(w \mid q)$ value are selected as expansion terms [33]. Diaz et al [57], propose a query expansion language model based on word embeddings learned from topic-constrained corpora. When word embeddings are learned from a topically-

[4] `http://social-book-search.humanities.uva.nl`

unconstrained corpora, they can be very general. Therefore, a query language model is defined based on word embeddings learned using a subset of documents sampled from a multinomial created by applying softmax on KL divergence scores of all documents in the corpus. The original query language model is interpolated with the query expansion language model which is defined by weights of terms computed by the UU^Tq where U is the $|V| \times d$ dimensional embedding matrix and q is the $|V| \times 1$ dimensional term matrix. Locally-trained embeddings are compared against global embeddings on TREC12, Robust and ClueWeb 2009 Category B Web corpus. Local embeddings are shown to yield higher NDCG@10 scores. Besides the local and global option, the authors also investigate the effect of using the target corpus versus an external corpus for learning word embeddings. A topically-constrained set of documents sampled from a general-purpose large corpus achieves the highest effectiveness scores.

Zamani and Croft [212] propose two separate QLMs and an extended relevance model [102] based on word embeddings. In the first QLM, $p(w \mid q)$ is computed by multiplying likelihood scores $p(w \mid t)$ given individual query terms whereas in the second QLM, $p(w \mid q)$ is estimated by an additive model over $p(w \mid t)$ scores. The $p(w \mid t)$ scores are based on similarity of word embeddings. For measuring similarity of embeddings, a sigmoid function is applied on top of the cosine similarity in order to increase the discriminative ability. The reason for this choice is the observation that the cosine similarity of the 1,000-th closest neighbor to a word is not much lower than the similarity of the first closest neighbor. Besides the QLMs, a relevance model [102], which computes a feedback query language model using embedding similarities in addition to term matching, is introduced. The proposed query language models are compared to Maximum Likelihood Estimation (MLE), GLM [3, 65] on AP, Robust and GOV2 collections from TREC. The first QLM is shown to be more effective in query expansion experiments. Regarding the PRF experiments, the embedding based relevance model combined with expansion using the first QLM produces the highest scores.

Zamani and Croft [213] develop a method for query embedding based on the embedding vectors of its individual words. The intuitiion is that a good query vector should yield a probability distribution $P(w \mid q) = \delta(w, q)/Z$ induced over the vocabulary terms similar to the query language model probability distribution (as measured by the KL Divergence). Here, w is the embedding vector of the word, q is the query embedding vector, $\delta(w, q)$ is the similarity between the two embedding vectors, and Z is a normalization constant. For similarity, they use the softmax and sigmoid transformations of cosine similarity. They show that the common heuristic of averaging individual query term vectors to induce the overall query embedding is actually a special case of their proposed theoretical framework for the case when vector similarity is measured by softmax and the query language model is estimated by maximum likelihood. Experiments with ad-hoc search using Galago are reported for three TREC test collections: AP, Robust04, and GOV2, using keyword TREC `title` queries. Word embeddings are induced by GloVe [161]. The GloVe are trained from a 6 billion token collection (Wikipedia 2014 plus Gigawords 5). Two different methods are used for estimating the query language model: via Pseudo Relevance Feedback (PRF) [102], which they refer to as *pseudo query vector* (PQV), and via maximum-likelihood estimation (MLE). Two methods are used for calculating similarity: softmax and sigmoid. Softmax is easy to use because of the closed form solution, but the sigmoid function consistently showed better performance, likely due to the flexibility provided by its two free parameters. Although PQV shows higher mean average precision, precision of top documents appears to suffer. Results also emphasize the importance of training data selection. Embeddings trained from the same domain as the documents being searched perform better than embeddings trained on a larger dataset from a different domain.

5.3.2 Learn

5.3.3 Learn to autoencode

Gupta et al [80] explore query expansion in mixed-script IR (MSIR), a task in which documents and queries in non-Roman written languages can contain both native and transliterated scripts together. Stemming from the observation that transliterations generally use Roman letters in such a way as to preserve the original-language pronunciation, the authors develop a method to convert both scripts into a bilingual embedding space. Therefore, they convert terms into feature vectors of the count of each Roman letter. An *auto-encoder* is then trained for query expansion by feeding in the concatenated native term feature vector and transliterated term feature vector. The output is the low-dimensional embedding in the abstract space. The training of the autoencoder includes greedy layer-wise pretraining and fine-tuning through backpropagation. Experiments are conducted on shared task data from FIRE (see Appendix B.2). Results suggest that their method significantly outperforms other state-of-the-art methods. Source code for the model is shared online (see Appendix B.4).

5.3.4 Learn to match

Sordoni et al [181] propose a supervised Quantum Entropy Minimization (QEM) approach for finding semantic representations of *concepts*, such as words or phrases, for query expansion. The authors suggest that text sequences should not lie in the same semantic space as single terms because their information content is higher. To this end, concepts are embedded in rank-one matrices while queries and documents are embedded as a mixture of rank-one matrices. This allows documents and queries to lie in a larger space and carry more semantic information than concepts, thereby achieving greater semantic resolution. For learning parameters of *density matrices*, QEM's gradient updates are a refinement of updates in both Bai et al [11]'s Supervised Semantic Indexing (SSI) and Rocchio [167]; a query concept embedding moves toward the embeddings of relevant documents' concepts and away from the embeddings of non-relevant documents' concepts. This has the effect of selecting which document concepts the query concept should be aligned with and also leads to a refinement of Rocchio: the update direction for query expansion is obtained by weighting relevant and non-relevant documents according to their similarity to the query. Due to lack of public query logs, QEM is trained on Wikipedia *anchor logs* [49]. Experiments conducted on ClueWeb09-Cat-B with TREC 2010-2012 Web Track topics show QEM outperforms Gao et al [68]'s concept translation model (CTM) (statistically significant differences), and SSI (occasionally statistically significant differences). QEM notably achieves improved precision at top-ranks. Also notable is QEM's ability to find useful expansion terms for longer queries due to higher semantic resolution. Additional preliminary experiments with Weston et al [195]'s *Weighted Approximate-Rank Pairwise* loss yields further improvements for QEM over baselines.

5.4 Result diversification

5.4.1 Aggregate

Explicit. Onal et al [153] propose to utilize GloVe embeddings in order to overcome the vocabulary gap between various TTU pairs, such as tweet-tweet, query-tweet and aspect-tweet pairs, in tweet search result diversification. Diversification algorithms rely on textual similarity functions in order to select the optimal set of results that maximizes both novelty and relevance. Exact matching functions fail to distinguish the aspects expressed with very short textual content. TTUs are expanded with k nearest neighbors of the terms in the GloVe embedding space. As similarity functions cannot distinguish details at the aspect level, marginal improvement are not observed in α-NDCG scores when explicit and implicit diversification algorithms are run with the expanded TTUs.

5.4.2 Learn

Learn to match. Prior state-of-the-art methods for diversifying search results include the Relational Learning-to-Rank framework (R-LTR) [224] and the Perceptron Algorithm using Measures as Margins (PAMM) [199]. These prior methods either use a heuristic ranking model based on a predefined document similarity function, or they automatically learn a ranking model from predefined novelty features often based on cosine similarity. In contrast, Xia et al [200] take automation a step further, using Neural Tensor Networks (NTN) to learn the novelty features themselves. The NTN architecture was first proposed to model the relationship between entities in a knowledge graph via a bilinear tensor product [179]. The model here takes a document and a set of other documents as input. The architecture uses a tensor layer, a max-pooling layer, and a linear layer to output a document novelty score. The NTN augmentations of R-LTR and PAMM perform at least as well as those baselines, showing how the NTN can remove the need for manual design of functions and features. It is not clear yet whether using a full tensor network works much better than just using a single slice of the tensor.

5.5 Expertise retrieval

5.5.1 Learn

Learn to predict. Van Gysel et al [188] propose an unsupervised discriminative log-linear model for the retrieval of the authors with the most expertise, given a topic as the query. Contextual relations to be used for training are derived from the author-document relations. Authors split each document into n-grams and compute probability distribution of authors given an n-gram. The training objective is to optimize the cross-entropy between the predicted distribution and the oracle distribution derived from the contextual relations, over the collection of n-grams. The neural model is a single layer neural network which takes a word as the input and predicts a probability distribution over the authors. The probability of an author given a sequence of words is computed by the multiplication of the probability values given each word, assuming conditional independence of expertise given words. Word embeddings and the author embeddings are learned jointly. The proposed log-linear model is shown to outperform the state-of-the-art vector space-based entity ranking and language modeling approaches, in terms of precision, in experiments on the TREC Enterprise Track 2005–2008 data sets and a University of Tilburg dataset. The authors note the need for improving the scalability of the model with respect to the number of authors.

5.6 Product Search

5.6.1 Learn

Learn to match. Van Gysel et al [187] describe a Latent Semantic Entities (LSE) method for learning a latent space model for product entity retrieval. Products are entities that are each associated with a text description accompanied by user reviews. LSE is a discriminative model that predicts probability distributions over a collection of product entities given a descriptive text. The LSE model comprises a word embedding matrix, an entity embedding matrix and a mapping from words to entities. Word embeddings and entity embeddings are learned jointly. Word embeddings and entity embeddings have different dimensionality. For a given textual description, embedding of the predicted entity is computed by a hidden neural network over the average of word embeddings. The objective function maximizes similarity between the predicted and actual entity embeddings while minimizing similarity between the predicted embeddings and randomly sampled negative instances. For scalability, NCE is used to approximate the probability distribution over the products. Although this work is classified as Learn to match due to the training objective, embeddings for a fixed collection products are learned. As an outlier in the Learn to match category, the input to the model is not a pair of textual units. One element of the pair is a product-id. Experiments are conducted on Amazon product data, comparing NDCG scores against three baseline methods including LSI, Latent Dirichlet Allocation (LDA), and `word2vec`. In all four product domains from the dataset, LSE outperforms baselines over all tested entity dimensionality sizes. LSE is also found to provide useful additional features to a Query-likelihood Language Model in learning-to-rank experiments.

6 Query Tasks

6.1 Query auto completion

6.1.1 Aggregate

Implicit. The work by Cai and de Rijke [29] on query auto completion introduces semantic features computed using Skip-Gram embeddings, for learning to rank query auto completion candidates. Query similarity computed by sum and maximum of the embedding similarity of term-pairs from queries are used as two separate features. The maximal embedding similarity of term pairs is found to be the most important feature in a diverse feature set including popularity-based features, a lexical similarity and another semantic similarity feature based on co-occurrence of term pairs in sessions.

6.1.2 Learn

Learn to match. The Convolutional Latent Semantic Model (CLSM) has been used to learn distributed representations of query reformulations [143] and of queries [144]. In both studies, CLSM representations are used to build additional features in an existing learning to rank framework for query auto completion.

Mitra [143] learns embedding vectors for query reformulation based on query logs, representing query reformulation as a vector using CLSM. In [143], a CLSM model is trained on query pairs that are observed in succession in search logs. This work provides an analysis of CLSM vectors for queries similar to the word embedding space analysis in [141]. Mitra [143] found that offsets between CLSM query vectors can represent intent transition patterns. To illustrate, the nearest neighbor query of the vector computed by $vector(university\ of\ washington) - vector(seattle) + vector(chicago)$ is found to be $vector(university\ of\ chicago)$. Besides, the offset of the vectors for $university\ of\ washington$ and $seattle$ is similar to the offset of the vectors for $chicago\ state\ university$ and $chicago$. Motivated by this feature of the CLSM vectors, query reformulations are represented as the offset vector from the source query to target query [143]. Clustering of query reformulations represented by the offset vectors yields clusters that contain pairs with similar intent transitions. For instance, the query reformulations in which there is an intent jump like *avatar dragons* $\rightarrow$ *facebook*, are observed grouped in a cluster. Two sets of features are then developed. The first feature set captures topical similarity via cosine similarity between the candidate embedding vector and embedding vectors of some past number of queries in the same session. The second set of *reformulation features* captures the difference between the candidate embedding vector and that of the immediately preceding query. Other features used include non-contextual features, such as most popular completion, as well as contextual features, such as n-gram similarity features and pairwise frequency features. LambdaMART [197] is trained on click-through data and features. Results suggest that embedding features give considerable improvement over Most Probable Completion (MPC) [13], in addition to other models lacking the embedding-based features.

Whereas popularity query auto completion methods perform well for head queries having abundant training data, prior methods often fail to recommend completions for tail queries having less usual prefixes, including both probabilistic algorithms [13, 31, 177] and learning-based QAC approaches [29, 91]. To address this, Mitra and Craswell [144] develop a query auto-completion procedure for such rare prefixes which enables query auto completion even for query prefixes never seen in training. The authors mine the most popular query suffixes (e.g., n-grams that appear at the end of a query) and append them to the user query prefixes, thus generating possible synthetic candidate solutions. To recommend possible query expansion, they use LambdaMART [197] with two sets of ranking features. The first feature set is the frequency of query n-grams in the search log. The second feature set is generated by training CLSM on the prefix-suffix dataset, sampling queries in the search logs and segmenting each query at every possible word boundary. Results on the AOL search log show significant improvements over MPC [13]. The authors also find n-gram features to be more important than CLSM features in contributing to overall model performance.

6.2 Query suggestions

6.2.1 Learn to generate

We know of no work using RNNs for query suggestion prior to Hierarchical Recurrent Encoder Decoder (HRED) by Sordoni et al [182] that trains a hierarchical GRU model to generate context-aware suggestions. They first use a GRU layer to encode the queries in each session into vector representations, then build another GRU layer on sequences of query vectors in a session and encode the session into a vector representation. The model learns its parameters by maximizing the log-likelihood of observed query sessions. To generate query suggestions, the model uses a GRU decoder on each word conditioned on both the previous words generated and the previous queries in the same session. The model estimates the likelihood of a query suggestion given the previous query. A learning-to-rank system is trained to rank query suggestions, incorporating the likelihood score of each suggestion as a feature. Results on the AOL query log show that the proposed approach outperforms several baselines that use only hand-crafted features. The model is also seen to be robust when the previous query is noisy. The authors then conduct a user study to evaluate the synthetic suggestions generated by the HRED model. Users are asked to classify the suggested queries as *useful*, *somewhat useful*, *not useful* categories. A total of 64% of the queries generated by the HRED model was found to be either useful or somewhat useful by users. This score is higher than all the other baselines where the highest score for “useful or somewhat useful” is about 45%. Because the model can generate synthetic queries, it can effectively handle long tail queries. However, only previous queries from the same session are used to provide the contextual query suggestion; the authors do not utilize click-through data from previous sessions. Because click-through data provides important feedback for synthetic query suggestions, incorporating such click-through data from previous sessions represents a possible direction for future work.

6.3 Query classification

6.3.1 Aggregate

Explicit. Using data from the KDD Cup 2005 Task,[5] Zamani and Croft [213] propose an embedding method for categorizing queries. See Section 5.1.1 for a description of Zamani and Croft [213]'s overall method and results for ad-hoc search. For query classification, given an item from the training data query-category, the authors first calculate the centroid vector of all query embedding vectors under a category. Then for a test query, they use the query embedding form and calculate the distance to the K nearest neighbor centroid vector. Finally, they use a softmax function over the set of calculated distances to determine the final set of categories for the test query. Because only embedding-based baselines are included in the evaluation, it is unclear how the proposed approach would perform vs. traditional IR models.

6.4 Proactive search

6.4.1 Learn to generate

In the only work we know of investigating Neural IR for proactive retrieval, Luukkonen et al [123] propose LSTM-based text prediction for query expansion. Intended to better support proactive intelligent agents such as Apple's *Siri*, *Ok Google*, etc., the LSTM is used to generate sequences of words based on all previous words written by users. A beam search is used to prune out low probability sequences. Finally, words remaining in the pruned tree are used for query expansion. They evaluate their method on the abstracts of the Computer Science branch of the arXiv preprint database, downlaoded on October 28, 2015. Experimental results show that the proposed method can proactively generate relevant resources and improve retrieval precision. The authors provide several possible future directions, such as using the model to automatically suggest different continuations for user text as it is written, as done in the Reactive Keyboard [50] and akin to query auto completion in search engines. User studies are also needed to test the system's effectiveness in the context of users' real-world tasks.

[5] `http://www.kdd.org/kdd-cup/view/kdd-cup-2005`

7 Question Answering

7.1 Answer sentence selection

7.1.1 Learn to match

Methods proposed in this section all evaluate on Wang et al [192]'s dataset derived from TREC QA 8–13. Wang and Nyberg [191] use a stacked bi-directional LSTM (BLSTM) to read a question and answer sequentially and then combine the hidden memory vectors from LSTMs of both question and answer. They use mean, sum and max-pooling as features. This model needs to incorporate key-word matching as a feature to outperform previous approaches that do not utilize deep learning. They use BM25 as the key-word matching feature and use Gradient boosted regression tree (GBDT) [64] to combine it with the LSTM model.

To rank pairs of short texts, Severyn and Moschitti [174] propose a Convolutional Deep Neural Network (CDNN). Their deep learning architecture has 2 stages. The first stage is a sentence embedding model using a CNN to embed question and answer sentences into intermediate representative vectors, which are used to compute their similarity. The second stage is a NN ranking model whose features include intermediate representative sentence vectors, similarity score, and some additional features such as word overlap between sentences. Results show improvements of about 3.5% in MAP vs. results reported by Yu et al [211], in which a CNN is used followed by logistic regression (LR) to rank QA pairs. The authors attribute this improvement to the larger width (of 5) of the convolutional filter in their CNN for capturing long term dependencies, vs. the unigram and bigram models used by Yu et al [211]. Beyond the similarity score, their second stage NN also takes intermediate question and answer representations as features to constitute a much richer representation than that of Yu et al [211].

Yang et al [205]'s approach starts by considering the interaction between question and answer at the word embedding level. They first build a question-answer interaction matrix using pre-trained embeddings. They then use a novel value-shared weight CNN layer (instead of a position-shared CNN) in order to induce a hidden layer. The motivation for this is that different matching ranges between a question term and answer will influence the later ranking score differently. After this, they incorporate an *attention network* for each question term to explicitly encode the importance of each question term and produce the final ranking score. They rank the answer sentences based on the predicted score and calculate MAP and MRR. Whereas Severyn and Moschitti [174] and Wang and Nyberg [191] need to incorporate additional features in order to achieve comparative performance, Yang et al [205] do not require any feature engineering.

Suggu et al [183] propose a Deep Feature Fusion Network (DFFN) to exploit both hand-crafted features (HCF) and deep learning based systems for Answer Question Prediction. Specifically, query/answer sentence representations are embedded using a CNN. A single feature vector of 601 dimensions serves as input to a second stage fully-connected NN. Features include sentence representations, HCF (e.g., TagMe, Google Cross-Lingual Dictionary (GCD), and Named Entities (NEs)), similarity measures between questions and answers, and metadata such as an answer author's reputation score. The output of the second stage NN is a score predicting the answer quality. They compare their approach to the top two best performing HCF based systems from SemEval 2015 and a pure deep learning system. For SemEval 2016, DFFN was compared with their corresponding top two best performing system. Results show that DFFN performs better than the top systems across all metrics (MAP, F1 and accuracy) in both SemEval 2015 and 2016 datasets. The authors attribute this to fusing the features learned from deep learning and HCF, since some important features are hard to learn automatically. As an example, question and answer often consists of several NEs along with variants, which are hard to capture using deep learning. However, NEs can be extracted from QA and their similarity used as a feature. Their use of HCF was also motivated by the many available similarity resources, such as Wikipedia, GCD, and click-through data, which could be leveraged to capture richer syntactic and semantic similarities between QA pairs.

7.2 Conversational agents

7.2.1 Learn

Learn to match. An automatic conversation response system called Deep Learning-to-Respond (DL2R) is proposed by Yan et al [203]. They train and test on 10 million posting-reply pairs of human conversation web data from various sources, including microblog websites, forums, Community Question Answering (CQA) bases, etc. For a given query they reformulate it using other contextual information and retrieve the most likely candidate reply. They model the total score as a function of three scores: query-reply, query-posting, and query-context, each fed into a neural network consisting of bi-directional LSTM RNN layers, convolution and pooling layers, and several feed-forward layers. The strength of DL2R comes from the incorporation of reply, posting, and context with the query.

8 Sponsored Search

8.1 Learn to match

Azimi et al [8] use DSSM represetantations for ad keyword re-writing. In paid search, each ad is associated with a set of keywords called *bided keywords*. The ads are ranked against a query and the ads at high rank are displayed to the user. In order to overcome the vocabulary mismatch between user queries and bided keywords, bided keywords are replaced with more common keywords. A set of candidate keywords are extracted from the set of documents returned by a search engine in response to the bided keyword query. The DSSM model is leveraged to rank the candidate keywords against the original keywords.

The *Deep-Intent* model proposed by Zhai et al [214, 215] comprises a Bidirectional Recurrent Neural Network (BRNN) combined with an attention module as the SCN. The attention module, first introduced in [10] for neural machine translation, is referred to as *attention pooling layer*. This is the first work that employs an attention module for a web search task. A recurrent neural network takes a sequence of words and generates a sequence of distributed representations, so-called *context-vectors*, aligned with each word. Each context vector encodes the semantics of the context from the start to the corresponding word. A Bidirectional Recurrent Neural Network (BRNN) processes the input word sequence in both forward and backward directions.

Context vectors generated by a BRNN encode the context after and before the associated word. The pooling strategy is merging the context vectors vectors into a single vector that encodes the semantics of the whole sequence. The sequence vector is assigned the last context vector in *last pooling*, whereas an element-wise max operation is applied on context vectors in *max-pooling*. In *attention pooling*, the sequence vector is computed by a weighted sum of context vectors where the weights are determined by the attention module. The attention module takes a sequence of context vectors and outputs a weight for each vector. The similarity score between a query and an ad is obtained by the dot product of their distributed representations. Similar query-ad pairs for training are extracted from the click logs. Query-ad pairs that have a click are selected as training samples and for each such sample, a randomly selected set of query-ad pairs (without click) are used as negative samples. Distributed representations are evaluated on click-logs from a product ad search engine with 966K pairs manually labeled by human judges. In the experiments, models that are built from different choices of RNN type (RNN, Bidirectional RNN, LSTM and LSTM-RNN) and pooling strategy (max-pooling, last pooling and attention pooling) are compared. The attention layer provides a significant gain in the AUC (area-under-curve of the receiver operating characteristic) scores when used with RNN and BRNN whereas it performs on a par with last pooling when used with LSTM-based networks. This can be attributed to the power of LSTM units for capturing long-term contextual relations. Besides the evaluation of distributed representations for matching, the attention scores are used to extract a subset of query words. Words that have the highest attention scores are selected to rewrite the query.

8.2 Learn to predict

Grbovic et al [75] present a pair of Learn to predict models in [75, 76]. In [75], Grbovic et al propose `query2vec`, a two-layer architecture, where the upper layer models the temporal context of a query session using a Skip-gram model, and the lower layer models word sequences within a query using `word2vec` CBOW. They also introduce two incremental models: `ad-query2vec`, which incorporates the learning of ad click vectors in the upper layer by inserting them into query sequences after queries that occurred immediately prior to an ad click; and `directed ad-query2vec`, which uses past queries as context for a directed language model in the upper layer. The models are trained using 12 billion sessions collected on Yahoo search and evaluated offline using historical activity logs, where success is measured by the click-through rate of ads served. All three `query2vec` models show improvement over sponsored keyword lists and search retargeting using `word2vec` and query flow graph.

In their subsequent, longer study, Grbovic et al [76] propose a method to train context and content-aware word embeddings. The first proposal is `context2vec`. It treats a search session as a sentence and each query from the session as a word from the sentence. It uses `word2vec`'s Skip-gram model. Queries with similar context will result in similar embeddings. The second model is `content2vec`. This method is similar to Le and Mikolov [103]'s PV in that it uses the query as a paragraph to predict the word in its context. The third model, `context-content2vec`, similar to their earlier `query2vec`, combines `context2vec` and `content2vec` to build a two-layer model which jointly considers the query session context and the query context. The context of the query is defined both by its content and other queries in the same session. The models are trained on Yahoo search logs that contain 12 billion sessions and embeddings for approximately 45 million queries are learned. Learned query embeddings are leveraged for rewriting queries in order to improve search retargeting. The original query is expanded with its k nearest neighbor queries in the query embedding space. The learned model is evaluated on TREC Web Track 2009–2013 queries and an in-house data set from Yahoo. For queries in the TREC data set, the query rewrites obtained by the proposed models are editorially judged. The PV-DM model that only predicts context yields lower editorial grades than the Query Flow Graph (QFG) baseline. Rewrites by *context2vec* and *context-content2vec* embeddings outperform the baseline. The rewrites by the *context-content2vec*$_{ad}$ model, which extends *context-content2vec* by adding the ads and links clicked in the same session to the TTU context, are assigned the highest editorial grades on average.

9 Similar Item Retrieval

9.1 Related document search

9.1.1 Aggregate

Explicit. Kusner et al [101] propose the Word Mover's Distance (WMD) for computing distances between two documents. WMD is defined as the minimum cumulative distance that all words in the first document of the pair need to travel to exactly match the other document of the pair. The distance between two words is computed by the Euclidean distance in the `word2vec` space. Authors evaluate the WMD metric on 8 supervised document datasets: BBC sports articles, tweets labeled with sentiments, recipe procedure descriptions, medical abstracts with different disease groups, academic papers with publisher names, news datasets with different topics, Amazon reviews with different category products. They compare their method with baselines including bag-of-words, TF-IDF, BM25 Okapi, LSI, LDA and so on. Their model outperforms these baselines.

Kim et al [96] propose another version of WMD specific to query-document similarity. The high computational cost of WMD is tackled by mapping queries to documents using a word embedding model trained on a document set. They make several changes to the original WMD methods: changing the weight of term by introducing inverse document frequency, and changing the original dissimilarity measure to cosine similarity. However, they do not provide any comparison vs. WMD as a baseline.

9.1.2 Learn

Learn to match. Semantic hashing is proposed by Salakhutdinov and Hinton [171] to map semantically similar documents near to one another in hashing space, facilitating easy search for similar documents. Multiple layers of Restricted Boltzmann Machines (RBMs) are used to learn the semantic structure of documents. The final layer is used as a hash code that compactly represents the document. The lowest layer is simply word-count data and is modeled by the Poisson distribution. The hidden layers are binary vectors of lower dimensions. The deep generative model is learned by first pre-training the RBMs one layer at a time (from bottom to top). The network is then "unrolled", i.e., the layers are turned upside down and stacked on top of the current network. The final result is an *auto-encoder* that learns a low-dimensional hash code from the word-count vector and uses that hash code to reconstruct the original word-count vector. The auto-encoder is then *fine-tuned* by back-propagation. Results show that semantic hashing is much faster than locality sensitive hashing [6, 51] and can find semantically similar documents in time independent of document collection size. However, semantic hashing difficult optimization procedures and a slow training mechanism, reducing applicability to large-scale tasks [118].

Learn to predict Djuric et al [58] point out use of the HDV model, reviewed in Section 5.1.3, for exploring similar documents in a document collection.

Le and Mikolov [103] assess vectors obtained by averaging PV-DM and PV-DBOW vectors (see Section 3.3.6 for details of PV) on snippet retrieval tasks. The snippet retrieval experiments are performed on a dataset of triplets created using snippets of the top 10 results retrieved by a search engine, for a set of 1 million queries. Each triplet is composed of two relevant snippets for a query and a randomly selected irrelevant snippet from the collection. Cosine similarity between paragraph vectors is shown to be an effective similarity metric for distinguishing similar snippets in such triplets. Dai et al [48] show that paragraph vectors outperform the vector representations obtained by LDA [22], average of word embeddings and tf-idf weighted one-hot vector representations, on a set of document triplets constructed with the same strategy in [103], using Wikipedia and arXiv documents.

9.2 Detecting text reuse

9.2.1 Aggregate

Explicit. The goal of Zhang et al [216] is to efficiently retrieve passages that are semantically similar to a query, making use of hashing methods on word vectors that are learned in advance. Other than the given word vectors, no further deep learning is used. Like Clinchant and Perronnin [42] and Zhou et al [223], they adopt the Fisher Kernel framework to convert variable-size concatenations of word embeddings to fixed length. However, this resulting fixed-length Fisher vector is very high-dimensional and dense, so they test various state-of-the-art hashing methods (e.g., Spectral Hashing [194] and SimHash [35]) for reducing the Fisher vector to a lower-dimensional binary vector. Experiments are conducted on six collections including TIPSTER (Volumes 1–3), ClueWeb09-Cat-B, Tweets2011, SogouT 2.0, Baidu Zhidao, and Sina Weibo, with some sentences manually annotated for semantic similarity. Hashing methods that use Fisher vector representations based on word embeddings achieve higher precision-recall curves than hashing methods without vector representations and have comparable computational efficiency.

9.3 Similar question retrieval

9.3.1 Aggregate

Explicit. Learning of word embeddings coupled with category metadata for CQA is proposed by Zhou et al [223]. They adopt `word2vec`'s Skip-gram model augmented with category metadata from online questions, with category information encoding the attributes of words in the question (see Zhang et al [217] for another example of integrating categorical data with word embeddings). In this way, they group similar words based on their categories. They incorporate the category constraint into the original Skip-gram objective function. After the word embedding is learned, they use Fisher kernel (FK) framework to convert the question into a fixed length vector (similar to Clinchant and Perronnin [42] and Zhang et al [216]). To retrieve similar questions, they use the dot product of FVs to calculate the semantic similarities. For their experiments, Zhou et al [223] train word embeddings on Yahoo! Answers and Baidu Zhidao for English and Chinese, respectively. Results show that the category metadata powered model outperforms all the other baselines not using metadata. Future work might include exploring how to utilize other metadata information, such as user ratings, to train more powerful word embeddings.

9.3.2 Learn

Learn to match. Lei et al [107] address the similar question retrieval in CQA platforms with a framework that applies Learn to match and Learn to generate in two separate stages of training. In the pre-training phase, an encoder-decoder network that generates the title of a question given title, body or merged title-body of the question, is trained. The encoder network is based on gated (non-consecutive) convolutions. Owing to the pre-training step, unsupervised question collection is utilized to tailor task-specific word embeddings and learn a rough semantic compositionality function for the questions. The encoder learned in the first step is used as the SCN for a training Learn to match model on the user-annotated similar question pairs. Evaluation is done on the StackExchange AskUbuntu data set. Negative pairs are constructed by selecting random questions from the collection. For the test set, similar pairs are annotated manually as the user-marked pairs are found to be noisy. In order to evaluate effectiveness, a set of questions retrieved by BM25 are ranked based on similarity scores computed by the learned Learn to match model. Lei et al present a comprehensive set of experiments that analyse both the overall gain by pre-training and the effect of the design choices such as the encoder neural network type, pooling strategies and inclusion of question body. Highest effectiveness scores are achieved by the combination of the gated convolutions, last-pooling and pre-training.

9.4 Recommendation

9.4.1 Aggregate

Explicit Manotumruksa et al [130] models user preferences using word embeddings for the task of context-aware venue recommendations (CAVR). In CAVR, each user can express a set of contextual aspects for their preference. Each aspect has multiple contextual dimension term, with each term having a list of related term. The task then is to rank a list of venues by measuring how well each of venue matches user's context preferences. It develops two approaches to model user-venue preferences and context-venue preferences using word embedding. First, it infers a vector representation for each venue using the comments on the venue, and it models the user-venue preferences using the rated venues' vector representation in the user's profile. Second, it models each dimension of each aspect in the context-venue preferences by identifying several most similar terms of that dimension term.

To evaluate their user-venue preferences model, Manotumruksa et al [130] firstly train word embeddings using Skip-gram model on the venues' comments dataset from Foursquare. Then it calculates the cosine similarity between the vector representation of venue and the user, and use it as a feature in the learning to rank system. It conducts the experiment on TREC 2013 and 2014 Contextual Suggestion tracks, and reports P@5 and MRR. The result shows the system with the proposed features using word embedding outperforms those without using word embedding.

Then, to evaluate its context-aware preference model, Manotumruksa et al [130] use cosine similarity between the venue and each of the contextual aspects as a feature in the ranking system. It also incorporates venue-dependent features and user-venue preference features. This experiment on TREC 2015 Contextual Suggestion task shows that the proposed new system outperforms the baseline that does not utilize user information and their contextual preferences, and it also outperforms the top performing system under P@5.

9.4.2 Learn

Learn to match. Gao et al [69] propose using DSSM for both automatic highlighting of relevant keywords in documents and recommendation of alternative relevant documents based upon these keywords. They evaluate their framework based on what they call *interestingness* tasks, derived from Wikipedia anchor text and web traffic logs. They find that feeding DSSM derived features into a supervised classifier for recommendation offers state-of-the-art performance and is more effective than simply computing distance in the DSSM latent space. Future work could incorporate complete user sessions, since prior browsing and interaction history recorded in the session provide useful additional signals for predicting interestingness. This signal might be modeled most easily by using an RNN.

10 Non Textual (or Behavioural)

Borisov et al [25] propose a distributed representation which captures the user information need as sequence of vector states instead of traditional binary event used in a probabilistic graphical model (PGM) based click model [40]. Existing PGM based click models capture a user's behavior as a sequence of observable and latent events. However, one serious limitation of these click models is that the dependencies among the sequence of the events are hand-crafted. The basic idea of Borisov et al's proposed model is as follows: they initialize their vector state with the initial user query and then update the vector states based on the user interaction and the next document. For transitioning from one state to another they define three mapping functions which they learn from the training data using two different neural networks architectures: RNN and LSTM. They perform their experimental analysis using Yandex (a major commercial search engine in Russia) Relevance Prediction dataset[6] on a user click prediction task and a relevance prediction task. As a baseline they use the DBN, DCM, CCM and UBM click models.[7] As a performance metrics they have used perplexity and log-likelihood and for relevance prediction task they have use NDCG at the 1, 3 , 5 and 10 level. Their experimental evidence shows that their proposed model has better predictive power than all those baseline.

As a follow-up to [25], Borisov et al [24] focus on behavioral signals based on times between user actions. The ability to accurately predict (i) click dwell time (i.e., time spent by a user on the landing page of a search result), and (ii) times from submission of a query to the first/last click on the results and to the next query submission (if none of the results will be clicked) allows us to optimize search engines for constructing result pages and suggesting query reformulations that minimize time it takes users to satisfy their information needs. At the heart of the solution proposed by the authors an LSTM is used to capture the contexts in which user actions take place. The proposed context-aware time model is evaluated on four temporal prediction tasks and a ranking task. The results show that the proposed context-aware time model, which makes use of the context in which actions take place, provides a better means to explain times between user actions than existing methods.

Predicting click-through rate (CTR) and conversion rate from categorical inputs such as region, ad slot size, user agent, etc., is important in sponsored search. Zhang et al [217] propose the first neural approach we know of to predict CTR for advertising. The authors develop two deep learning approaches to this problem, a Factorization Machine supported Neural Network (FNN) and Sampling-based Neural Network (SNN). The Factorization Machine is a non-linear model that can efficiently estimate

[6] http://imat-relpred.yandex.ru/en/datasets

[7] Implementations of all of these click models are available at https://github.com/markovi/PyClick

feature interactions of any order even in problems with high sparsity by approximating higher order interaction parameters with a low-rank factorized parameterization. The use of an FM-based bottom layer in the deep network, therefore, naturally solves the problem of high computational complexity of training neural networks with high-dimensional binary inputs. The SNN is augmented either by a sampling-based Restricted Boltzmann Machine (SNN-RBM) or a sampling-based Denoising Auto-Encoder (SNN-DAE). The main challenge is that given many possible values of several categorical fields, converting them into dummy variables results in a very high-dimensional and sparse input space. For example, thirteen categorical fields can become over 900,000 binary inputs in this problem. The FNN and SNN reduce the complexity of using a neural network on such a large input by limiting the connectivity in the first layer and by pre-training by selective sampling, respectively. After pre-training, the weights are *fine-tuned* in a supervised manner using back-propagation. Evaluation focuses on the tuning of SNN-RBM and SNN-DAE models and their comparison against logistic regression, FM and FNN, on the *iPinYou* dataset[8] [115] with click data. Results show that one of the proposed methods performs best, though the baselines are often close behind and twice take second place. The authors also find a diamond-shape architecture is better than increasing, decreasing, or constant hidden-layer sizes and that dropout works better than L2 regularization. Though this method can extract non-linear features, it is only very effective when dealing with advertisements without images. Consequently, further research on multi-modal sponsored search to model images and text would be useful to pursue.

11 Lessons and Reflections

At the end of the early years of neural IR, it is natural to ask the following question: *Is there a set of established guidelines for neural IR? Is it sufficient to aggregate embeddings or do we really need deep neural network models?* In this section, we summarize the lessons and reflections compiled from the reviewed work. Apart from the high-level choice between the Aggregate and Learn methods, there are separate design choices to be made for each category identified in Table 2.

As to a high level view regarding the choice between Aggregate and Learn, no consistent advantage of word embeddings has emerged. Among the models that fall within the *aggregate* category, directly using word embeddings provides consistent gains in [65] but not in [57, 150, 212, 213, 225]. In [225], word embedding similarity achieves comparable effectiveness to mutual information (MI) based term similarity. For query-document similarity, Nalisnick et al [150] point out that utilising relations between the IN and OUT embedding spaces learned by CBOW yields a more effective similarity function for query-document pairs. Diaz et al [57] propose to learn word embeddings from a topically constrained corpora since the word embeddings learned from an unconstrained corpus are found to be too general. Zamani and Croft [212, 213] apply a sigmoid function on the cosine similarity scores in order to increase the discriminative power.

[8] `http://data.computational-advertising.org`

Similar to past development of new modeling techniques in IR, there is a common theme of researchers starting first with bag-of-words models then wanting to move toward modeling longer phrases in their future work. Ganguly et al [65] suggest future work should investigate compositionality of term embeddings. Zuccon et al [225] propose incorporating distributed representations of phrases to better model query term dependencies and compositionality. Zheng and Callan [222] propose direct modeling of bigrams and proximity terms. Zamani and Croft [213] suggest query language models based on mutual-information and more complex language models (bigram, trigram, etc.) could be pursued.

11.1 Reflections on Aggregate

In this section, we present the reflections on the evaluation of neural IR systems that follow the Aggregate approach and rely on word embeddings.

11.1.1 Is there an established guideline for the choices to be made for learning word embeddings?

Use of word embeddings involves the design decisions presented below.

NLM Should one use `word2vec` (CBOW or Skip-gram) [136, 139], GloVe [161], or something else (such as count-based embeddings)? Can embeddings from multiple sets be selected between dynamically or combined together [151, 220]?

Corpora What training data/corpora should be used? Does performance vary much if we simply use off-the-shelf embeddings (e.g., from `word2vec` or GLoVe) vs. re-training embeddings for a target domain, either by *fine-tuning* [133] off-the-shelf embeddings or re-training from scratch? Presumably, larger training data is better, along with in-domain data similar to the test data on which the given system is to be applied, but how does one trade-off greater size of out-of-domain data vs. smaller in-domain data? How might they be best used in combination?

Hyperparameters How should hyper-parameters be set (e.g., dimensionality of embedding space, window size, etc.)?

OOV Words How should one deal with out-of-vocabulary (OOV) terms not found in the word embedding training data?

There is no complete analysis that would help to derive guidelines. We summarize the reported partial observations from the literature.

Choice of NLM. Selection among `word2vec` CBOW or Skip-gram or GloVe appears quite varied. Zuccon et al [225] compare CBOW vs. Skip-gram, finding "no statistical significant differences between the two ..." Kenter and de Rijke [95] use both `word2vec` and GloVe [161] embeddings (both the originally released embeddings as well training their own embeddings) in order to induce features for their machine learning model. They report model effectiveness using the original public embeddings with or without their own additional embeddings, but do not report further ablation studies to understand the relative contribution of different embeddings used. Grbovic et al [75]'s `query2vec` uses a two-level architecture in which the upper layer models the temporal context of query sequences via Skip-gram, while the bottom layer models word sequences within a query using CBOW. However, these choices are not justified, and their later work [76] uses Skip-gram only. AL-Masri et al [3] evaluate both CBOW and Skip-gram `word2vec` embeddings (using default dimensionality and context window settings) but present only Skip-gram results, writing that "there was no big difference in retrieval performance between the two." Zamani and Croft [212, 213] adopt GloVe without explanation. Similarly for `word2vec`, Mitra et al [145] simply adopt CBOW, while others adopt Skip-gram[52, 130, 189, 205, 208, 223]. Zhang and Wallace [219] perform an empirical analysis in the context of using CNNs for short text classification. They found that the "best" embedding to use for initialization depended on the dataset. Motivated by this observation, the authors proposed a method for jointly exploiting multiple sets of embeddings (e.g., one embedding set induced using GloVe on some corpus and another using a `word2vec` variant on a different corpus) [220]. This may also be fruitful for IR tasks, suggesting a potential direction for future work.

Corpora. In the work that we have reviewed, embeddings are either learned from a general-purpose corpus like Wikipedia (*general purpose embeddings*) or a task-specific corpus. For Ad-hoc retrieval, the retrieval corpus itself is considered as a corpus to learn embeddings. Besides, word embeddings are learned from various task-specific corpora of community questions and their metadata [223], journal abstracts and patient records [52] and venues' comments from Foursquare [130]. Vulic and Moens [189] learn bilingual word embeddings from a corpus of pseudo-bilingual documents obtained via a merge-and-shuffle approach on a document level parallel corpora, applying their model to cross-lingual IR.

For ad-hoc retrieval, embeddings learned from Wikipedia (*general purpose embeddings*) and embeddings learned from the retrieval corpus itself *(corpus-specific word embeddings)* are compared in [222, 225]. The authors note that no significant effect of this choice is observed for query-reweighting [222] or in a translation language model for computing term similarities [225]. Zamani and Croft [213] train GloVe on three external corpora and report, "there is no significant differences between the values obtained by employing different corpora for learning the embedding vectors." Similarly, Zheng and Callan [222], regarding their query-reweighting model, write:

> "[the system] performed equally well with all three external corpora; the differences among them were too small and inconsistent to support any conclusion about which is best. However, although no external corpus was best for all datasets ... The corpus-specific word vectors were never best in these experiments ... given the wide range of training data sizes – varying from 250 million words to 100 billion words – it is striking how little correlation there is between search accuracy and the amount of training data."

In contrast, Diaz et al [57] highlight that query expansion with word embeddings learned from a topic-constrained collection of documents yields higher effectiveness scores compared to embeddings learned from a general-purpose corpora. Besides, Yang et al [206] considers classification in which one background corpus (used to train word embeddings) is a Spanish Wikipedia Dump which contains over 1 million articles, while another is a collection of 20 million tweets having more than 10 words per tweet. As expected, they find that when the background training text matches the classification text, the performance is improved.

Hyperparameters. Vulic and Moens [189], Yang et al [206], Zheng and Callan [222], Zuccon et al [225] compare different training hyper-parameters such as window size and dimensionality of word embeddings. Zuccon et al [225]'s sensitivity analysis of the various model hyperparameters for inducing word embeddings shows that manipulations of embedding dimensionality, context window size, and model objective (CBOW vs Skip-gram) have no consistent impact on model performance vs. baselines. Vulic and Moens [189] find that while increasing dimensionality provides more semantic expressiveness, the impact on retrieval performance is relatively small. Zheng and Callan [222] find that 100 dimensions work best for estimating term weights, better than 300 and 500. In experiments using the Terrier platform, Yang et al [206] find that for the Twitter election classification task using CNNs, word embeddings with a large context window and dimension size can achieve statistically significant improvements.

OOV Terms. Regarding the handling of OOV terms, the easiest solution is to discard or ignore the OOV terms. For example, Zamani and Croft [213] only consider queries where the embedding vectors of all terms are available. However, in end-to-end systems, where we are jointly estimating (or refining) embeddings alongside other model parameters, it is intuitive to randomly initialize embeddings for OOV words. For instance, in the context of CNNs for text classification, Kim [97] adopted this approach. The intuition behind this is two-fold. First, if the same OOV appears in a pair of texts, queries, or documents being compared, this contributes to the similarity scores between those two. Second, if two different OOV terms appear in the same pair, their dissimilarity will not contribute in the similarity function. However, this does not specifically address accidental misspellings or creative spellings (e.g., "kool") commonly found in social media. One might address this by hashing words to character n-grams (see Section 11.2.6) or character-based modeling more generally (e.g., [46, 56, 218]).

11.1.2 How to use word embeddings?

As mentioned previously, publications in the Implicit category are characterized by integrating word embedding similarity into existing retrieval frameworks. In contrast, publications in the Explicit category build TTU representations based on word embeddings, in an unsupervised way. For the Implicit approach, the term similarity function is a design choice whereas the TTU similarity function stands as a design choice for the Explicit category.

Besides the similarity functions, for the Explicit category, it is crucial to note the choice of the aggregation function that builds the TTU vector from word embeddings. There are some publications that diverge from the simple aggregation method averaging/summing embedding vectors. Clinchant and Perronnin [42], Zhang et al [216], Zhou et al [223] use a Fisher Kernel [90] in order to compute TTU representations. Ganguly et al [66] opine that simply adding vectors of word embedding cannot sufficiently capture the context of longer textual units. Instead, the authors propose a new form of similarity metric based on the assumption that the document can be represented as a mixture of p-dimensional Gaussian probability density functions and each `word2vec` embedding (p-dimensions) is an observed sample. Then, using the EM algorithm, they estimate the probability density function that can be incorporated into the query likelihood language model using linear interpolation.

Word similarity function. Cosine similarity is the choice adopted by almost all of the publications reviewed. However, Zamani and Croft [212, 213] propose using sigmoid and softmax transformations of cosine similarity on the grounds that cosine similarity values are not discriminative enough. Their empirical analysis shows that there are no substantial differences (e.g., two times more) between the similarity of the most similar term and the $1,000$-th similar term to a given term w, while the $1,000$-th word is unlikely to have any semantic similarity with w. Consequently, they propose using monotone mapping functions (e.g., sigmoid or softmax) to transform the cosine similarity scores.

TTU similarity function. Publications under the Explicit category are characterized by building TTU representations based on pre-trained embeddings and computing TTU pair similarity by cosine similarity of the distributed TTU representations. Deviating from this generic approach, Ganguly et al [66] opine that simply adding vectors of word embedding cannot sufficiently capture the context of longer textual units. Instead, the authors propose a new form of similarity metric based on the assumption that the document can be represented as a mixture of p-dimensional Gaussian probability density functions and each `word2vec` embedding (p-dimensions) is an observed sample. Then, using the EM algorithm, they estimate the probability density function which can be incorporated to the query likelihood language model using linear interpolation. Finally, Word Mover's Distance (WMD) [101] is an alternative distance metric defined as the minimum cumulative distance that all words in the first document of the pair need to travel to exactly match the other document of the pair. The distance between two words is computed by the Euclidean distance in the `word2vec` space.

11.2 Reflections on Learn

In this subsection, we present design choices for the models reviewed in the Learn category of the *How* dimension of our taxonomy (Table 2).

11.2.1 How to choose the appropriate category of Learn?

When making a choice for a training objective (a subcategory of Learn), the following three points should be considered: training data, representations for unseen TTUs, and TTU length.

Training data. Learn to match models are trained to maximize the relevance scores of pairs that have been annotated or inferred to be relevant and minimize the scores of irrelevant pairs. In contrast, Learn to predict and Learn to generate models assume that TTUs that co-occur in the same context are similar.

Computing representations for unseen TTUs. In the Learn to autoencode, Learn to generate and Learn to match models, distributed representations of unseen TTUs can be computed by forward propagation through the SCN. In contrast, Learn to predict models are focused on learning embeddings for a fixed collection of TTUs. An additional inference step, which requires including the representations of the entire collection, is required for obtaining the embedding of an unobserved TTU. While PV (the first Learn to predict model) has been adopted in many studies (e.g., Xia et al [200] use PV-DBOW to represent documents) and is often reported as a baseline (e.g., [185]), concerns about reproducibility have also been raised. Kiros et al [98] report results below SVM when re-implementing PV. Kenter and de Rijke [95] note, “it is not clear, algorithmically, how the second step – the inference for new, unseen texts – should be carried out.” Perhaps most significantly, later work co-authored by Mikolov [134][9] has disavowed the original findings of Le and Mikolov [103], writing, “to match the results from Le and Mikolov [103], we followed the [author’s] suggestion ... However, this produces the [reported] result only when the training and test data are not shuffled. Thus, we consider this result to be invalid.”

TTU length. Learn to generate models are designed to generate unseen synthetic textual units. To this end, generating textual units has been shown to be successful for queries [123, 182]. Generating long textual units is not well-studied up to this point, except [116]. The DSSM variants from the Learn to match are shown to work better on document titles instead of the entire document content [78, 175].

11.2.2 Choice of semantic compositionality network: How to represent text beyond single words? Does text granularity influence the choice?

The simplest way to represent longer textual units, such as phrases, sentences, or entire documents, is to sum or average their constituent word embeddings. However such bag-of-words compositions ignore word ordering, and simple averaging treats all words as equally important in composition (though some work has considered weighted operations, e.g., [189]). Ganguly et al [66] opine that:

> “... adding the constituent word vectors ... to obtain the vector representation of the whole document is not likely to be useful, because ... compositionality of the word vectors [only] works well when applied over a relatively small number of words ... [and] does not scale well for a larger unit of text, such as passages or full documents, because of the broad context present within a whole document.”

[9] `https://github.com/mesnilgr/iclr15`

Fortunately, a variety of SCNs, including Feed forward NN, CNN, RNN, LSTM and their variants, have been proposed for inducing representations of longer textual units. However, there is no evidence that any one method consistently outperforms the others, with the performance of each method instead appearing to often depend on the specific task and dataset being studied. We further discuss the use of CNNs and RNNs below.

Convolutional Neural Networks (CNNs). Shen et al [175, 176] propose a Convolutional Latent Semantic Model (CLSM) to encode queries and documents into fix-length vectors, following a popular "convolution+pooling" CNN architecture. The first layer of CLSM is a word hashing layer that can encode words into vectors. CLSM does not utilize word embeddings as input, seemingly distinguishing it from all other works using CNNs. Mitra [143] use CLSMs to encode query reformulations for query prediction, while Mitra and Craswell [144] use CLSMs on query prefix-suffix pairs corpus for query auto-completion. They sample queries from the search query logs and split the query at every possible word boundary to form prefix-suffix pairs.

Severyn and Moschitti [174] and Suggu et al [183] adopt a similar "convolution+pooling" CNN architecture to encode question and answer sentence representations, which serve as features for a second-stage ranking NN. See Mitra et al [145] for further discussion of such two-stage *telescoping* approaches.

Yang et al [205] develop a novel value-shared CNN, and apply it on the query-answer matching matrix to extract the semantic matching between the query and answer. This model can capture the interaction between intermediate terms in the query and answer, rather than only considering the final representation of the query and answer. The motivation behind the value-shared CNN is that semantic matching value regularities between a question and answer is more important than spatial regularities typical in computer vision. Similarly, contemporaneous work by Guo et al [78] notes:

> "Existing interaction-focused models, e.g., ARC-II and MatchPyramid, employ a CNN to learn hierarchical matching patterns over the matching matrix. These models are basically position-aware using convolutional units with a local "receptive field" and learning positional regularities in matching patterns. This may be suitable for the image recognition task, and work well on semantic matching problems due to the global matching requirement (i.e., all the positions are important). However, it may not be suitable for the ad-hoc retrieval task, since such positional regularity may not exist . . . "

Recurrent Neural Networks (RNNs). Sordoni et al [182] build a hierarchical GRU to encode the query and the query session into vector representations. Song et al [180] use an LSTM to model user interests at different time steps and encode them into a vector. Lioma et al [116] create new relevant information using an LSTM that takes the concatenated text of a query and its known relevant documents as input using word embeddings. However, rather than take the whole text of a document, they extract a context window of $\pm n$ terms around every query term occurrence. Yan et al [203] use a bidirectional LSTM followed by a CNN to model the original query, reformulated query, candidate reply, and antecedent post in their human-computer conversation system. Wang and Nyberg [191] use a stacked bidirectional LSTM to sequentially read words from both question and answer sentences, calculating relevance scores for answer sentence selection through mean pooling across all time steps. Section 11.2.3 presents Cohen et al [43]'s comparison of CNNs vs. RNNs by document length.

11.2.3 Text granularity in IR: Does text granularity have an effect on the optimal choice for SCN?

Many studies to date focus on short text matching rather than longer documents more typical of modern IR ad-hoc search. Cohen et al [43] study how the effectiveness of NN models for IR vary as a function of document length (i.e., text granularity). They consider three levels of granularity: (i) *fine*, where documents often contain only a single sentence and relevant passages span only a few words (e.g., question answering); (ii) *medium*, where documents consist of passages with a mean length of 75 characters and relevant information may span multiple sentences (e.g., passage retrieval); and (iii) *coarse*, or typical modern ad-hoc retrieval. For fine granularity, they evaluate models using the TREC QA dataset and find that CNNs outperform RNNs and LSTMs, as their filter lengths are able to effectively capture language dependencies. For medium granularity, they evaluate using the Yahoo Webscope L4 CQA dataset and conclude that LSTM networks outperform CNNs due to their ability to model syntactic and semantic dependencies independent of position in sequence. In contrast, RNNs without LSTM cells do not perform as well, as they tend to "forget" information due to passage length.

For ad-hoc retrieval performance on Robust04, comparing RNNs, CNNs, LSTM, DSSM, CLSM, Vulic and Moens [189]'s approach, and Le and Mikolov [103]'s PV, Cohen et al [43] find that all neural models perform poorly. They note that neural methods often convert documents into fixed-length vectors, which can introduce a bias for either short or long documents. However, they find that the approaches of Vulic and Moens [189] and Le and Mikolov [103] perform well when combined with language modeling approaches that explicitly capture matching information of queries and documents. Similar observation is also reported in [66].

11.2.4 Choice of similarity function: Which function should I choose to measure the similarity between two text snippets?

Similarity here is not strictly limited to linguistic semantics, but more generally includes relevance matching between queries and documents or questions and answers. While cosine similarity is the simplest and the most common approach, a variety of more sophisticated methods have been proposed.

Wang and Nyberg [191] use a stacked bidirectional LSTM to sequentially read words from both question and answer sentences, calculating relevance scores for answer sentence selection through mean pooling across all time steps. Yan et al [203] match sentences by concatenating their vector representations and feeding them into a multi-layer fully-connected neural network, matching a query with the posting and reply in a human computer conversation system. Xia et al [200] propose a neural tensor network (NTN) approach to model document novelty. This model takes a document and a set of other documents as input. The architecture uses a tensor layer, a max-pooling layer, and a linear layer to output a document novelty score. Guo et al [78]'s recent Deep Relevance Matching Model appears to be one of the most successful to date for ad-hoc search over longer document lengths. In terms of textual similarity, they argue that the ad-hoc retrieval task is mainly about relevance matching, different from semantic matching in NLP. They model the interaction between query terms and document terms, building a matching histogram on top of the similarities. They then feed the histogram into a feed forward neural network. They also use a term gating network to model the importance of each query term.

Yang et al [205] propose an attention-based neural matching model (aNMM) for question answering. Similar to Guo et al [78], they first model the interaction between query terms and document terms to build a matching matrix. They then apply a novel value-shared CNN on the matrix. Since not all query terms are equally important, they use a softmax gate function as an attention mechanism in order to learn the importance of each query term when calculating the matching between the query and the answer.

To summarize, simply calculating cosine similarity between two text vector representations might not be the best choice to capture the relevance relation between texts. People develop neural models to learn similarity, and model the interaction between texts in a finer granularity.

11.2.5 Initializing word embeddings in NNs: Should they be tuned or not during the training process?

We observed three approaches for initializing and updating the embeddings during training of Learn models:

– The embedding layer of the models are initialized randomly and learned from scratch during training of the task-specific model.
– The embeddings layer is initialized with pre-trained embeddings and is kept fixed during training of the task-specific model.
– The embedding layer is initialized with pre-trained embeddings and is fine-tuned for the task during training of the task-specific model.

At present, there is no clear evidence showing that either method consistently works best. However, we may note that learning word embeddings is possible only with large corpora; thus in cases where task-specific training data is limited as in TREC Ad-Hoc collections [78] and the TREC-QA data set (see Section 7.1 for the related work), pre-trained word embeddings have been preferred to initialize, in order to shift focus of training towards learning semantic compositionality.

11.2.6 How to cope with large vocabularies?

Training neural networks is computationally expensive and today it is not possible to train a CNN or RNN with a 1M word vocabulary. `word2vec` models, being shallow networks with a single linear layer can be trained with large vocabularies owing to softmax approximation methods such as negative sampling and NCE. In cases where it is crucial to learn a semantic compositionality model with a CNN or RNN architecture, the large vocabulary brings a challenge. *Word hashing*, first introduced in [89] together with the DSSM model and adopted by [143, 144, 175, 176, 207], reduces large vocabularies to a moderate sized vocabulary of 30K of character trigrams. Words are broken down into letter n-grams and then represented as a vector of letter n-grams.

Huang et al [89] provide an empirical analysis where the original vocabulary size of 500K is reduced to only 30K owing to word hashing. While the number of English words can be unlimited, the number of letter n-grams in English is often limited, thus word hashing can resolve the out-of-vocabulary (OOV) problem as well. However, one inherent problem of word hashing is the hashing conflict, which can be serious for a very large corpus. Another set of methods, such as Byte Pair Encoding [173] have been proposed in order to cope with large vocabularies in Neural Machine Translation. Although only word hashing has been used for neural IR to this end, other vocabulary reduction methods should be considered in future work.

12 Conclusion

The purpose of this survey is to offer an introduction to neural models for information retrieval by surveying the state of knowledge up to the end of 2016. To this end we have reviewed and classified existing work in the area. We used a taxonomy in which we recognize different target textual units (TTUs), different types of usage of learned text representations ("*usage*") IR *Task*s, as well as different methods for building representations ("*how*"). Within the latter category we identified two sub-categories: the *aggregate* and *learn* categories. The *aggregate* category includes methods based on pre-computed word embeddings for computing semantic similarity, while the *learn* category covers neural semantic compositionality models.

Within the *aggregate* category we observed two major patterns of exploiting word embeddings. In the *explicit* type of use of embeddings, TTUs are associated with a representation in the word embedding space and semantic similarity of TTUs is computed based on these representations. In the *implicit* type of use, similarity of word embeddings is plugged in as term similarity in an existing statistical language modeling frameworks for retrieval. Several strategies for adapting word embeddings for document retrieval have been introduced, such as topically constraining the document collection, new similarity functions and the inclusion of TF-IDF weights for aggregating word embeddings. This may be understood as an indication that we need to design IR specific objectives for learning distributed representations. Are the training objective and semantic relationships encoded by the embedding vectors useful for the target retrieval task? A future direction would be to identify the types of semantic word relations that matter to semantic matching in web search, across multiple tasks.

We classified the neural semantic compositionality models reviewed in the *learn* category, into four sub-categories *Learn to autoencode, Learn to match, Learn to predict* and *Learn to generate*, considering the training objectives optimized for learning representations. *Learn to match* models are trained using noisy relevance signals from click information in click-through logs whereas the models in the other categories are designed to predict or generate task-specific context of TTUs. The majority of the Learn to match and Learn to predict models are evaluated on datasets extracted from commercial search engine logs. A comparative evaluation of models from different sub-categories on publicly available data sets, is required in order to gain a deeper understanding of semantic compositionality for matching.

Currently, existing Learn to predict/generate context and Learn to generate models mostly rely on temporal context windows. It would be interesting to examine other types of contextual relations in search logs, such as long term search history of users and noisy relevance signals exploited by learn to match models. Another future direction would concern applications of the attention mechanism [10] for designing models that can predict where a user would attend in document, given a query.

Looking forward, we believe there are several key directions where progress is needed. First, we presented the document retrieval, query suggestion and ad retrieval-sponsored search tasks as largely disjoint tasks. However, the models proposed for one task may be useful for another. For instance, the *context-content2vec* model [76] was evaluated only on matching ads to queries yet the distributed query representations could also be evaluated for query suggestion or query auto completion [30]. In particular, there is a need to compare distributed query representations and similarity/likelihood scores produced by the proposed models on query tasks. In some work, the representations were used as features in learning to rank frameworks and there are no clues about the power of these representations in capturing semantics.

More broadly, there is a need for systematic and broad task-based experimental surveys that focus on comparative evaluations of models from different categories, but for the same tasks and under the same experimental conditions, very much like the reliable information access (RIA) workshop that was run in the early 2000s to gain a deeper understanding of query expansion and pseudo relevance feedback [82].

Another observation is that recently introduced generative models—mostly based on recurrent neural networks—can generate unseen (synthetic) textual units. The generated textual units have been evaluated through user studies [116, 182]. For the query suggestion task, generated queries have been found to be useful; and so have word clouds of a synthetic document. The impact of these recent neural models on user satisfaction or retrieval scenarios should be investigated on real scenarios.

Over the years, IR has made tremendous progress by learning from user behavior, either by introducing, e.g., click-based rankers [163] or, more abstractly, by using models that capture behavioral notions such as examination probability and attractiveness of search results through click models [40]. How can such implicit signals be used to train neural models for semantic matching in web search? So far, we have only seen limited examples of the use of click models in training neural models for web search tasks.

Interest in Neural IR has never been greater, spanning both active research and deployment in practice[10] [135]. Neural IR continues to accelerate in quantity of work, sophistication of methods, and practical effectiveness [78]. New methods are being

[10] `https://en.wikipedia.org/wiki/RankBrain`

explored that may be computationally infeasible today (see Diaz et al [57]), but if proven effective, could motivate future optimization work to make them more practically viable (e.g., [92, 154]). NN approaches have come to dominate speech recognition (2011), computer vision (2013), and NLP (2015). Similarly, deep learning will come to dominate information retrieval as well [128].

At the same time, healthy skepticism about Neural IR also remains. The key question in IR today might be most succinctly expressed as: "Will it work?" While NN methods have worked quite well on short texts, effectiveness on longer texts typical of ad-hoc search has been problematic [43, 89], with only very recent evidence to the contrary [78]. Side by side comparisons of lexical vs. neural methods often show at least as many losses as gains for neural methods, with at best an advantage "on average" [187, 188]. In addition, while great strides have been made in computer vision through employing a very large number of *hidden layers* (hence "deep" learning), such deep structures have typically been less effective in NLP and IR than more shallow architectures [158], though again with notable recent exceptions [46]. When Neural IR has led to improvements in ad-hoc search results, improvements appear relatively modest [57, 212] when compared to traditional query expansion techniques for addressing *vocabulary mismatch*, such as pseudo-relevance feedback (PRF). Both Ganguly et al [66] and Diaz et al [57] have noted that *global* word embeddings, trained without reference to user queries, vs. *local* methods like PRF for exploiting query-context, appear limited similarly to the traditional global-local divide seen with existing approaches like topic modeling [209].

As Li [109] put it, "Does IR Need Deep Learning?" Such a seemingly simple question requires careful unpacking. Much of the above discussion assumes Neural IR should deliver new state-of-the-art quality of search results for traditional search tasks. While it may do so, this framing may be far too narrow, as Li [109]'s presentation suggests. The great strength of Neural IR may lie in enabling a new generation of search scenarios and modalities, such as searching via conversational agents [203], multi-modal retrieval [124, 125], knowledge-based search IR [152], or synthesis of relevant material [116]. It may also be that Neural IR will provide greater traction for other future search scenarios not yet considered.

Given that the efficacy of deep learning approaches is often driven by "big data", will Neural IR represent yet another fork in the road between industry and academic research, where massive commercial query logs deliver Neural IR's true potential? Or should we frame this more positively as an opportunity for research on generating training material or even simulation, as has previously been pursued for, e.g., learning to rank [117], see, e.g., [9, 20]? There is also an important contrast to note here between supervised scenarios, such as learning to rank vs. unsupervised learning of word embeddings or typical queries (see [143, 144, 182, 187, 188]). LeCun et al [105] wrote, "we expect unsupervised learning to become far more important in the longer term." Just as the rise of the Web drove work on unsupervised and semi-supervised approaches by the sheer volume of unlabeled data it made available, the greatest value of Neural IR may naturally arise where the biggest data is found: continually generated and ever-growing behavioral traces in search logs, as well as ever-growing online content.

While skepticism of Neural IR may well remain for some time, the practical importance of search today, coupled with the potential for significantly new traction offered by this “third wave” of NNs, makes it unlikely that researchers will abandon Neural IR anytime soon without having first exhaustively tested its limits. As such, we expect the pace and interest in Neural IR will only continue to blossom, both in new research and increasing application in practice. Consequently, this first special-issue journal on Neural IR in 2017 will likely attract tremendous interest and is well-poised for timely impact on research and practice.

Acknowledgements We would like to thank Christophe van Gysel from the University of Amsterdam, for his valuable feedback and comments. We would also like to thank our anonymous reviewers for their constructive comments and the guest editors for their advice.

The following additional students at the University of Texas at Austin contributed indirectly to the writing of this literature review: Manu Agarwal, Edward Babbe, Anuparna Banerjee, Jason Cai, Dillon Caryl, Yung-Sheng Chang, Shobhit Chaurasia, Linli Ding, Brian Eggert, Michael Feilbach, Alan Gee, Jeremy Gin, Rahul Huilgol, Miles Hutson, Neha Javalagi, Yan Jiang, Kunal Lad, Yang Liu, Amanda Lucio, Kristen Moor, Daniel Nelson, Geoffrey Potter, Harshal Priyadarshi, Vedhapriya Raman, Eric Roquemore, Juliette Seive, Abhishek Sinha, Ashwini Venkatesh, Yuxuan Wang, and Xu Zhang.

A Acronyms used

AI	artificial intelligence
ASR	automatic speech recognition
BoEW	Bag of Embedded Words
BWESG	Bilingual word Embeddings Skip-Gram
C-DSSM	Convolutional Deep Structured Semantic Models
CBOW	Continuous Bag of Words
CDNN	Convolutional Deep Neural Network
CLSM	Convolutional Latent Semantic Model
CNN	convolutional neural network
CQA	community question answering
DESM	Dual Embedding Space Model
DFR	Divergence From Randomness
DRMM	Deep Relevance Matching Model
DSM	distributional semantic model
DSSM	Deep Structured Semantic Model
EPV	Extended Paragraph Vector
FK	Fisher Kernel
FV	Fisher Vector
GLM	Generalized Language Model
GloVe	Global Vectors
GRU	Gated Recurrent Unit
HAL	Hyperspace Analog to Language
HDV	Hierarchical Document Vector
HRED	Hierarchical Recurrent Encoder Decoder
IR	information retrieval
IS	Importance Sampling
KB	knowledge base
LDA	Latent Dirichlet Allocation
LM	language model
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
LSTM-DSSM	LSTM Deep Structured Semantic Model
LSTM	Long Short Term Memory

MI	mutual information
MT	machine translation
NCE	Noise Contrastive Estimation
NEG	Negative Sampling
NLM	Neural Language Model
NLP	Natural language processing
NLTM	Neural Translation Language Model
NN	neural network
NNLM	Neural Network Language Model
NTN	Neural Tensor Networks
OOV	Out Of Vocabulary
PAMM	Perceptron Algorithm using Measures as Margins
PGM	probabilistic graphical model
PLSA	Probabilistic Latent Semantic Analysis
QA	question answering
PV-DBOW	Paragraph Vector with Distributed Bag of Words
PV-DM	Paragraph Vector with Distributed Memory
PV	Paragraph Vector
QLM	query language model
R-LTR	Relational Learning-to-Rank framework
RBM	Restricted Boltzman Machine
RecNN	recursive neural network
RNN	recurrent neural network
RNNLM	Recurrent Neural Network Language Model
SC	semantic compositionality
SCN	Semantic Compositionality Network
SGNS	Skip-Gram with Negative Sampling
TTU	target textual unit
WMD	Word Mover's Distance

B Resources

In this section, we present pointers to publicly available resources and tools which the reader would benefit for getting started with neural models, distributed representations, and information retrieval experiments with semantic matching.

B.1 Word embeddings

Corpora used. Wikipedia and GigaWord5 are the corpora widely used for learning word embeddings. The latest Wikipedia dump can be obtained from Wikimedia.[11] The GigaWord5 data set is accessible through the LDC.[12] Several authors have learned embeddings from query logs [29, 94, 182].

Pre-trained word embeddings. It is possible to obtain pre-trained GloVe embeddings[13] learned from large corpora, CBOW embeddings trained on large-scale Bing query logs [145][14] and different sets of word embeddings[15] used for evaluating NLTM in [225].

[11] `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2`

[12] `https://catalog.ldc.upenn.edu/LDC2011T07`

[13] `http://nlp.stanford.edu/projects/glove/`

[14] `https://www.microsoft.com/en-us/download/details.aspx?id=52597`

[15] `http://www.zuccon.net/ntlm.html`

Learning word embeddings. The source code for GloVe [161][16] and the models introduced in [108][17] is publicly shared by the authors. Implementations of the Word2Vec and Paragraph Vector models are included in the *gensim* library.[18]

Visualizing word embeddings. The dimensionality reduction technique t-Distributed Stochastic Neighbor Embedding (t-SNE) [127] is commonly used for visualizing word embedding spaces and for trying to understand the structures learned by neural models.[19]

B.2 Test corpora used for retrieval experiments

For retrieval experiments regarding neural models for semantic matching, the full range of CLEF, FIRE and TREC test collections has been used. In Table 3, we present the list of data sets used in experimental frameworks of reviewed work. For each data set, the list of studies that report related experiment results is given in the table.

Table 3: Datasets used in experimental setups. *(Continues)*

English Data	**Study**
20-Newsgroup Corpus	Salakhutdinov and Hinton [171]
Amazon product data [131]	Van Gysel et al [187]
AOL Query Logs	Kanhabua et al [94], Mitra [143], Mitra and Craswell [144], Sordoni et al [182]
Bing Query Logs and WebCrawl	Mitra [143], Mitra and Craswell [144], Mitra et al [145], Nalisnick et al [150]
CLEF03 English Ad-hoc	Clinchant and Perronnin [42]
CLEF 2016 Social Book Search	Amer et al [5]
CLEF Medical Corpora	ALMasri et al [3]
eHealth	Rekabsaz et al [165]
MSN Query Log	Kanhabua et al [94]
MSR Paraphrase Corpus	Kenter and de Rijke [95]
OHSUMED	De Vine et al [52]
PubMed	Balikas and Amini [12]
Reuters Volume I (RCV1-v2)	Salakhutdinov and Hinton [171]
SemEval 2015-2016	DFFN (Suggu et al [183])
Stack Overflow	Boytsov et al [26]
TIPSTER (Volume 1-3)	Zhang et al [216]
TREC 1-2 Ad-hoc (AP 88-89)	Clinchant and Perronnin [42], Zamani and Croft [212], Zamani and Croft [213], Zuccon et al [225]
TREC 1-3 Ad-hoc	Zuccon et al [225], Rekabsaz et al [165]

[16] http://nlp.stanford.edu/projects/glove/
[17] https://bitbucket.org/omerlevy/hyperwords
[18] https://radimrehurek.com/gensim/
[19] https://lvdmaaten.github.io/tsne/

TREC 6-8 Ad-hoc	GLM (Ganguly et al [65]), Lioma et al [116], Rekabsaz et al [164], Roy et al [169], Rekabsaz et al [165], Roy et al [168]
TREC 9-10	Roy et al [168]
TREC 12 Ad-hoc	Diaz et al [57]
TREC 2005 HARD	Rekabsaz et al [164], Rekabsaz et al [165]
TREC 2007-2008 Million Query	Yang et al [204]
TREC 2009-2011 Web	Xia et al [200]
TREC 2009-2013 Web	PV (Grbovic et al [76])
TREC 2010-2012 Web	QEM (Sordoni et al [181])
TREC 2011 Microblog	CDNN (Severyn and Moschitti [174]), Zhang et al [216]
TREC 2012 Microblog	CDNN (Severyn and Moschitti [174])
TREC 2015 Contextual Suggestion	Manotumruksa et al [130]
TREC ClueWeb09-Cat-B	DRMM (Guo et al [78]), QEM (Sordoni et al [181]), Zhang et al [216], Zheng and Callan [222]
TREC DOTGOV	Zuccon et al [225]
TREC Enterprise Track 2005-2008	Van Gysel et al [188]
TREC GOV2	Yang et al [204], Zamani and Croft [212], Zamani and Croft [213], Zheng and Callan [222]
TREC MedTrack	De Vine et al [52], Zuccon et al [225]
TREC QA 8-13	aNMM (Yang et al [205]), BLSTM (Wang and Nyberg [191]), CDNN (Severyn and Moschitti [174]), Yu et al [211], [43]
TREC Robust	GLM (Ganguly et al [65]), Roy et al [169], Roy et al [168]
TREC Robust 2004	Clinchant and Perronnin [42], Diaz et al [57], Guo et al [78], Zamani and Croft [212], Zamani and Croft [213], Zheng and Callan [222]
TREC WSJ87-92	Zuccon et al [225]
TREC WT10G	Roy et al [169], Zheng and Callan [222]
Yahoo! Answers	Boytsov et al [26], Zhou et al [223]
Chinese Data	**Study**
Baidu Tieba	Yan et al [203]
Baidu Zhidao	Yan et al [203], Zhang et al [216], Zhou et al [223]
Douban Forum	Yan et al [203]
Sina Weibo	Yan et al [203], Zhang et al [216]
SogouT 2.0	Zhang et al [216]
Multi-Lingual Data	**Study**
CLEF 2001-2003 Ad-hoc	Vulic and Moens [189]
FIRE 2013	Gupta et al [80]
iPinYou [115]	Zhang et al [217]
UT Expert Retrieval [19]	Van Gysel et al [188]
Yandex	Borisov et al [25]

B.3 Implementing neural SC models

Theano,[20] TensorFlow[21] Torch[22] and PyTorch[23] are libraries that are widely used by the deep learning community for implementing neural network models. These libraries enable construction of neural network models from pre-defined high-level building blocks such as hidden units and layers. It is possible to define neural network models with different choices of architectures, non-linearity functions, etc.

GPU support and automatic differentiation [15] are crucial features required for fast training neural networks. Theano and TensorFlow, enable performing matrix operations efficiently, in parallel, on GPU. Training neural networks with back-propagation requires computation of derivatives of the cost function with respect to every parameter of the network. Automatic differentiation in Theano and TensorFlow relieve the users from the effort on the manual derivation of derivatives of the objective function. These libraries compute the derivatives automatically given the definition of the neural network architecture and the cost function.

B.4 Publicly Available Implementations of Existing Neural IR Models

The deep learning community has a strong tradition of sharing, in some version, the code used to produce the experimental results reported in the field's publications. The information retrieval community is increasingly adopting this attitude too. Some authors of publications on neural IR models have shared their code; see e.g., DRMM [78][24], HRED [182][25], SERT [187, 188][26], CDNN [174][27], DeepMerge [106][28], DeepTR [222][29], Mixed Deep [80][30], NTLM [225][31], aNMM [205][32], KEDRLM [168][33].

References

1. Ai Q, Yang L, Guo J, Croft WB (2016) Analysis of the paragraph vector model for information retrieval. In: Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval, ACM, New York, NY, USA, ICTIR '16, pp 133–142
2. Ai Q, Yang L, Guo J, Croft WB (2016) Improving language estimation with the paragraph vector model for ad-hoc retrieval. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR '16, pp 869–872
3. ALMasri M, Berrut C, Chevallet JP (2016) A comparison of deep learning based query expansion with pseudo-relevance feedback and mutual information. In: European Conference on Information Retrieval, Springer, pp 709–715
4. Amati G, Van Rijsbergen CJ (2002) Probabilistic models of information retrieval based on measuring the divergence from randomness. ACM Transactions on Information Systems (TOIS) 20(4):357–389

[20] http://deeplearning.net/software/theano/
[21] https://www.tensorflow.org/
[22] http://torch.ch/
[23] http://pytorch.org/
[24] http://www.bigdatalab.ac.cn/benchmark/bm/bd?code=DRMM(LCH-IDF)
[25] https://github.com/sordonia/hred-qs
[26] https://github.com/cvangysel/SERT
[27] https://github.com/aseveryn/deep-qa
[28] https://ciir.cs.umass.edu/downloads/DeepMerge/
[29] http://www.cs.cmu.edu/~gzheng/code/TermRecallKit-v2.tar.bz2
[30] http://www.dsic.upv.es/~pgupta/mixed-script-ir
[31] https://github.com/ielab/adcs2015-NTLM
[32] https://github.com/yangliuy/aNMM-CIKM16
[33] https://github.com/gdebasis/kderlm

5. Amer NO, Mulhem P, Gery M (2016) Toward word embedding for personalized information retrieval. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
6. Andoni A, Indyk P (2006) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, pp 459–468
7. Arel I, Rose DC, Karnowski TP (2010) Deep machine learning-a new frontier in artificial intelligence research. IEEE Computational Intelligence Magazine 5(4):13–18
8. Azimi J, Alam A, Zhang R (2015) Ads keyword rewriting using search engine results. In: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, WWW '15 Companion, pp 3–4, DOI 10.1145/2740908.2742739
9. Azzopardi L, de Rijke M, Balog K (2007) Building simulated queries for known-item topics: An analysis using six european languages. In: 30th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval, ACM
10. Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:14090473
11. Bai B, Weston J, Grangier D, Collobert R, Sadamasa K, Qi Y, Chapelle O, Weinberger K (2009) Supervised semantic indexing. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, ACM, pp 187–196
12. Balikas G, Amini MR (2016) An empirical study on large scale text classification with skip-gram embeddings. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
13. Bar-Yossef Z, Kraus N (2011) Context-sensitive query auto-completion. In: Proceedings of the 20th International Conference on World wide Web, ACM, pp 107–116
14. Baroni M, Dinu G, Kruszewski G (2014) Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, pp 238–247
15. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM (2015) Automatic differentiation in machine learning: a survey. arXiv preprint arXiv:150205767
16. Bengio Y (2009) Learning deep architectures for AI. Foundations and Trends in Machine Learning 2(1):1–127
17. Bengio Y, Ducharme R, Vincent P, Janvin C (2003) A neural probabilistic language model. Journal of Machine Learning Research 3:1137–1155
18. Bengio Y, Senécal JS, et al (2003) Quick training of probabilistic neural nets by importance sampling. In: AISTATS
19. Berendsen R, Balog K, Bogers T, van den Bosch A, de Rijke M (2013) On the assessment of expertise profiles. Journal of the American Society for Information Science and Technology 64(10):2024–2044
20. Berendsen R, Tsagkias M, Weerkamp W, de Rijke M (2013) Pseudo test collections for training and tuning microblog rankers. In: SIGIR '13: 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM
21. Berger A, Lafferty J (1999) Information retrieval as statistical translation. In: Proceedings of the 22nd annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 222–229
22. Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. Journal of machine Learning research 3(Jan):993–1022
23. Bordes A, Chopra S, Weston J (2014) Question answering with subgraph embeddings. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, pp 615–620
24. Borisov A, Markov I, de Rijke M, Serdyukov P (2016) A context-aware time model for web search. In: SIGIR 2016: 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 205–214
25. Borisov A, Markov I, de Rijke M, Serdyukov P (2016) A neural click model for web search. In: Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, pp 531–541
26. Boytsov L, Novak D, Malkov Y, Nyberg E (2016) Off the beaten path: Let's replace term-based retrieval with k-nn search. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, pp 1099–1108

27. Broder A, Domingos P, de Freitas N, Guyon I, Malik J, Neville J (2016) Is deep learning the new 42? In: Plenary Panel at the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
28. Bromley J, Bentz JW, Bottou L, Guyon I, LeCun Y, Moore C, Säckinger E, Shah R (1993) Signature verification using a "siamese" time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence 7(04):669–688
29. Cai F, de Rijke M (2016) Learning from homologous queries and semantically related terms for query auto completion. Information Processing & Management 52(4):628–643
30. Cai F, de Rijke M (2016) A survey of query auto completion in information retrieval. Foundations and Trends in Information Retrieval 10(4):273–363
31. Cai F, Liang S, de Rijke M (2014) Time-sensitive personalized query auto-completion. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, pp 1599–1608
32. Carmel D, Zwerdling N, Guy I, Ofek-Koifman S, Har'El N, Ronen I, Uziel E, Yogev S, Chernov S (2009) Personalized social search based on the user's social network. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, ACM, pp 1227–1236
33. Carpineto C, Romano G (2012) A survey of automatic query expansion in information retrieval. ACM Comput Surv 44(1):1:1–1:50
34. Cartright MA, Allan J, Lavrenko V, McGregor A (2010) Fast query expansion using approximations of relevance models. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, ACM, pp 1573–1576
35. Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, pp 380–388
36. Chen W, Grangier D, Auli M (2016) Strategies for training large vocabulary neural language models. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, pp 1975–1985
37. Chirita PA, Firan CS, Nejdl W (2007) Personalized query expansion for the web. In: Proceedings of the 30th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 7–14
38. Cho K (2015) Natural language understanding with distributed representation. arXiv preprint arXiv:151107916
39. Cho K, van Merrienboer B, Gülçehre Ç, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR abs/1406.1078
40. Chuklin A, Markov I, de Rijke M (2015) Click Models for Web Search. Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers
41. Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS Workshop on Deep Learning
42. Clinchant S, Perronnin F (2013) Aggregating continuous word embeddings for information retrieval. In: Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, pp 100–109
43. Cohen D, Ai Q, Croft WB (2016) Adaptability of neural networks on varying granularity IR tasks. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
44. Collobert R, Weston J (2008) A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine learning, ACM, pp 160–167
45. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. Journal of Machine Learning Research 12(Aug):2493–2537
46. Conneau A, Schwenk H, Barrault L, Lecun Y (2016) Very deep convolutional networks for natural language processing. arXiv preprint arXiv:160601781
47. Croft B, Metzler D, Strohman T (2009) Search Engines: Information Retrieval in Practice. Addison-Wesley Publishing Company
48. Dai AM, Olah C, Le QV, Corrado GS (2014) Document embedding with paragraph vectors. In: NIPS Deep Learning Workshop
49. Dang V, Croft BW (2010) Query reformulation using anchor text. In: Proceedings of the third ACM International Conference on Web Search and Data Mining, ACM, pp 41–50
50. Darragh JJ, Witten IH, James ML (1990) The reactive keyboard: A predictive typing aid. Computer 23(11):41–49

51. Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry, ACM, pp 253–262
52. De Vine L, Zuccon G, Koopman B, Sitbon L, Bruza P (2014) Medical semantic similarity with a neural language model. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, pp 1819–1822
53. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. Journal of the American society for information science 41(6):391
54. Deng L (2014) A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Transactions on Signal and Information Processing 3:e2
55. Deng L, Yu D (2013) Deep learning: Methods and applications. Foundations and Trends in Signal Processing 7(3–4):197–387
56. Dhingra B, Zhou Z, Fitzpatrick D, Muehl M, Cohen W (2016) Tweet2vec: Character-based distributed representations for social media. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, pp 269–274
57. Diaz F, Mitra B, Craswell N (2016) Query expansion with locally-trained word embeddings. In: 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, pp 367–377
58. Djuric N, Wu H, Radosavljevic V, Grbovic M, Bhamidipati N (2015) Hierarchical neural language models for joint representation of streaming documents and their content. In: Proceedings of the 24th International Conference on World Wide Web, ACM, New York, NY, USA, WWW '15, pp 248–255
59. Dumais S, Banko M, Brill E, Lin J, Ng A (2002) Web question answering: Is more always better? In: Proceedings of the 25th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 291–298
60. Dyer C (2014) Notes on noise contrastive estimation and negative sampling. CoRR abs/1410.8251
61. Elman JL (1990) Finding structure in time. Cognitive science 14(2):179–211
62. Erhan D, Bengio Y, Courville A, Manzagol PA, Vincent P, Bengio S (2010) Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11(Feb):625–660
63. Faruqui M, Dodge J, Jauhar SK, Dyer C, Hovy E, Smith NA (2014) Retrofitting word vectors to semantic lexicons. arXiv preprint arXiv:14114166
64. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Annals of Statistics pp 1189–1232
65. Ganguly D, Roy D, Mitra M, Jones GJ (2015) Word embedding based generalized language model for information retrieval. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 795–798
66. Ganguly D, Roy D, Mitra M, Jones G (2016) Representing documents and queries as sets of word embedded vectors for information retrieval. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
67. Gao J (2015) Deep learning for web search and natural language processing
68. Gao J, He X, Nie JY (2010) Clickthrough-based translation models for web search: from word models to phrase models. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM), ACM, pp 1139–1148
69. Gao J, Pantel P, Gamon M, He X, Deng L (2014) Modeling interestingness with deep neural networks. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, pp 2–13
70. Goldberg Y (2016) A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research 57:345–420
71. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press
72. Graves A (2012) Neural networks. In: Supervised Sequence Labelling with Recurrent Neural Networks, Springer, pp 15–35
73. Graves A (2013) Generating sequences with recurrent neural networks. arXiv preprint arXiv:13080850
74. Graves A, Wayne G, Danihelka I (2014) Neural turing machines. arXiv preprint arXiv:14105401
75. Grbovic M, Djuric N, Radosavljevic V, Bhamidipati N (2015) Search retargeting using directed query embeddings. In: Proceedings of the 24th International Conference on World Wide Web, ACM, New York, NY, USA, WWW '15 Companion, pp 37–38

76. Grbovic M, Djuric N, Radosavljevic V, Silvestri F, Bhamidipati N (2015) Context-and content-aware embeddings for query rewriting in sponsored search. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 383–392
77. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2015) Lstm: A search space odyssey. arXiv preprint arXiv:150304069
78. Guo J, Fan Y, Ai Q, Croft WB (2016) A deep relevance matching model for ad-hoc retrieval. In: The 25th ACM International Conference on Information and Knowledge Management, Indianapolis, United States
79. Guo J, Fan Y, Ai Q, Croft WB (2016) A deep relevance matching model for ad-hoc retrieval. In: CIKM 2016: 25th ACM Conference on Information and Knowledge Management, ACM
80. Gupta P, Bali K, Banchs RE, Choudhury M, Rosso P (2014) Query expansion for mixed-script information retrieval. In: Proceedings of the 37th International ACM SIGIR Conference on Research & development in information retrieval, ACM, pp 677–686
81. Gutmann MU, Hyvärinen A (2012) Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. J Mach Learn Res 13(1):307–361
82. Harman D, Buckley C (2009) Overview of the reliable information access workshop. Information Retrieval 12(6):615–641
83. Harris ZS (1954) Distributional structure. Word 10(2-3):146–162
84. Hill F, Cho K, Korhonen A (2016) Learning distributed representations of sentences from unlabelled data. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, San Diego, California, pp 1367–1377
85. Hinton G, Deng L, Yu D, Dahl GE, Mohamed Ar, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, others (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29(6):82–97
86. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
87. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780
88. Hu B, Lu Z, Li H, Chen Q (2014) Convolutional neural network architectures for matching natural language sentences. In: Advances in Neural Information Processing Systems, pp 2042–2050
89. Huang PS, He X, Gao J, Deng L, Acero A, Heck L (2013) Learning deep structured semantic models for web search using clickthrough data. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge management, ACM, pp 2333–2338
90. Jaakkola TS, Haussler D, et al (1999) Exploiting generative models in discriminative classifiers. Advances in Neural Information Processing Systems pp 487–493
91. Jiang JY, Ke YY, Chien PY, Cheng PJ (2014) Learning user reformulation behavior for query auto-completion. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, ACM, pp 445–454
92. Jurgovsky J, Granitzer M, Seifert C (2016) Evaluating memory efficiency and robustness of word embeddings. In: European Conference on Information Retrieval, Springer, pp 200–211
93. Kalchbrenner N, Grefenstette E, Blunsom P (2014) A convolutional neural network for modelling sentences. arXiv preprint arXiv:14042188
94. Kanhabua N, Ren H, Moeslund TB (2016) Learning dynamic classes of events using stacked multi-layer perceptron networks. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
95. Kenter T, de Rijke M (2015) Short text similarity with word embeddings. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, pp 1411–1420
96. Kim S, Wilbur WJ, Lu Z (2016) Bridging the gap: a semantic similarity measure between queries and documents. arXiv preprint arXiv:160801972
97. Kim Y (2014) Convolutional neural networks for sentence classification. arXiv preprint arXiv:14085882
98. Kiros R, Zhu Y, Salakhutdinov RR, Zemel R, Urtasun R, Torralba A, Fidler S (2015) Skip-thought vectors. In: Advances in Neural Information Processing Systems, pp 3294–3302
99. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp 1097–1105
100. Kumar A, Irsoy O, Su J, Bradbury J, English R, Pierce B, Ondruska P, Gulrajani I, Socher R (2015) Ask me anything: Dynamic memory networks for natural language processing. arXiv preprint arXiv:150607285

101. Kusner MJ, Sun Y, Kolkin NI, Weinberger KQ (2015) From word embeddings to document distances. In: Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), pp 957–966
102. Lavrenko V, Croft WB (2001) Relevance based language models. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR '01, pp 120–127
103. Le QV, Mikolov T (2014) Distributed representations of sentences and documents. In: ICML, vol 14, pp 1188–1196
104. LeCun Y, Bengio Y (1995) Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks 3361(10):1995
105. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
106. Lee CJ, Ai Q, Croft WB, Sheldon D (2015) An optimization framework for merging multiple result lists. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, pp 303–312
107. Lei T, Joshi H, Barzilay R, Jaakkola TS, Tymoshenko K, Moschitti A, Màrquez L (2016) Semi-supervised question retrieval with gated convolutions. In: NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016, pp 1279–1289
108. Levy O, Goldberg Y, Dagan I (2015) Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics 3:211–225
109. Li H (2016) Does IR need deep learning?
110. Li H (2016) Opportunities and challenges in deep learning for information retrieval
111. Li H, Lu Z (2016) Deep Learning for Information Retrieval. In: SIGIR, Pisa, Italy, vol Tutorial at the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp 1203–1206
112. Li H, Xu J (2013) Semantic matching in search. Foundations and Trends in Information Retrieval 7(5):343–469
113. Li J, Luong MT, Jurafsky D, Hovy E (2015) When are tree structures necessary for deep learning of representations? arXiv preprint arXiv:150300185
114. Li X, Guo C, Chu W, Wang YY, Shavlik J (2014) Deep learning powered in-session contextual ranking using clickthrough data. In: In Proc. of NIPS
115. Liao H, Peng L, Liu Z, Shen X (2014) ipinyou global rtb bidding algorithm competition dataset. In: Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ACM, pp 1–6
116. Lioma C, Larsen B, Petersen C, Simonsen JG (2016) Deep learning relevance: Creating relevant information (as opposed to retrieving it). In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
117. Liu TY (2009) Learning to rank for information retrieval. Foundations and Trends in Information Retrieval 3(3):225–331
118. Liu W, Wang J, Ji R, Jiang YG, Chang SF (2012) Supervised hashing with kernels. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, pp 2074–2081
119. Liu X, Croft WB (2004) Cluster-based retrieval using language models. In: Proceedings of the 27th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 186–193
120. Liu X, Gao J, He X, Deng L, Duh K, Wang YY (2015) Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Denver, Colorado, pp 912–921
121. Lu Z, Li H (2013) A deep architecture for matching short texts. In: Advances in Neural Information Processing Systems, pp 1367–1375
122. Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behavior Research Methods, Instruments, & Computers 28(2):203–208
123. Luukkonen P, Koskela M, Floreen P (2016) LSTM-based predictions for proactive information retrieval. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
124. Ma L, Lu Z, Li H (2015) Learning to answer questions from image using convolutional neural network. arXiv preprint arXiv:150600333

125. Ma L, Lu Z, Shang L, Li H (2015) Multimodal convolutional neural networks for matching image and sentence. In: Proceedings of the IEEE International Conference on Computer Vision, pp 2623–2631
126. Ma L, Lu Z, Li H (2016) Learning to answer questions from image using convolutional neural network. In: AAAI Conference on Artificial Intelligence, pp 3567–3573
127. van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. Journal of Machine Learning Research 9(Nov):2579–2605
128. Manning C (2016) Natural Language Inference, Reading Comprehension and Deep Learning
129. Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieval. Cambridge University Press
130. Manotumruksa J, Macdonald C, Ounis I (2016) Modelling user preferences using word embeddings for context-aware venue recommendation. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
131. McAuley J, Pandey R, Leskovec J (2015) Inferring networks of substitutable and complementary products. In: KDD, ACM, pp 785–794
132. McClelland JL, Rumelhart DE, PDP Research Group, et al (1986) Parallel distributed processing. Explorations in the microstructure of cognition 2:216–271
133. Mesnil G, He X, Deng L, Bengio Y (2013) Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: INTERSPEECH, pp 3771–3775
134. Mesnil G, Mikolov T, Ranzato M, Bengio Y (2014) Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. In: International Conference on Learning Representations (ICLR)
135. Metz C (2016) AI is transforming google search. the rest of the web is next. WIRED Magazine
136. Mikolov T, Dean J (2013) Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems
137. Mikolov T, Karafiát M, Burget L, Cernockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: INTERSPEECH, pp 1045–1048
138. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. CoRR abs/1301.3781
139. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781
140. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) Advances in Neural Information Processing Systems 26, Curran Associates, Inc., pp 3111–3119
141. Mikolov T, Yih Wt, Zweig G (2013) Linguistic regularities in continuous space word representations. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Atlanta, Georgia, pp 746–751
142. Mitchell J, Lapata M (2010) Composition in distributional models of semantics. Cognitive Science 34(8):1388–1429, DOI 10.1111/j.1551-6709.2010.01106.x
143. Mitra B (2015) Exploring session context using distributed representations of queries and reformulations. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 3–12
144. Mitra B, Craswell N (2015) Query auto-completion for rare prefixes. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, pp 1755–1758
145. Mitra B, Nalisnick E, Craswell N, Caruana R (2016) A dual embedding space model for document ranking. arXiv preprint arXiv:160201137
146. Mnih A, Teh YW (2012) A fast and simple algorithm for training neural probabilistic language models. arXiv preprint arXiv:12066426
147. Morin F, Bengio Y (2005) Hierarchical probabilistic neural network language model. In: Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005
148. Moshfeghi Y, Triantafillou P, Pollick FE (2016) Understanding information need: An fMRI study. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR '16, pp 335–344

149. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp 807–814
150. Nalisnick E, Mitra B, Craswell N, Caruana R (2016) Improving document ranking with dual word embeddings. In: 25th World Wide Web (WWW) Conference Companion Volume, International World Wide Web Conferences Steering Committee, pp 83–84
151. Neelakantan A, Shankar J, Passos A, McCallum A (2014) Efficient non-parametric estimation of multiple embeddings per word in vector space. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp 1059–1069
152. Nguyen GH, Tamine L, Soulier L, Bricon-Souf N (2016) Toward a deep neural approach for knowledge-based IR. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
153. Onal KD, Altingovde IS, Karagoz P (2015) Utilizing Word Embeddings for Result Diversification in Tweet Search, Springer International Publishing, Cham, pp 366–378. DOI 10.1007/978-3-319-28940-3_29
154. Ordentlich E, Yang L, Feng A, Cnudde P, Grbovic M, Djuric N, Radosavljevic V, Owens G (2016) Network-Efficient Distributed Word2vec Training System for Large Vocabularies. In: The 25th ACM International Conference on Information and Knowledge Management, Indianapolis, United States
155. Palakodety S, Callan J (2014) Query transformations for result merging. In: Proceedings of The Twenty-Third Text REtrieval Conference, TREC 2014, Gaithersburg, Maryland, USA, November 19-21, 2014
156. Palangi H, Deng L, Shen Y, Gao J, He X, Chen J, Song X, Ward R (2014) Semantic modelling with long-short-term memory for information retrieval. CoRR abs/1412.6629
157. Palangi H, Deng L, Shen Y, Gao J, He X, Chen J, Song X, Ward R (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
158. Pang L, Lan Y, Guo J, Xu J, Cheng X (2016) A study of matchpyramid models on ad-hoc retrieval. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
159. Pang L, Lan Y, Guo J, Xu J, Wan S, Cheng X (2016) Text matching as image recognition. In: 30th AAAI Conference on Artificial Intelligence, pp 2793–2799
160. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. ICML (3) 28:1310–1318
161. Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: EMNLP, vol 14, pp 1532–43
162. Ponte JM, Croft WB (1998) A language modeling approach to information retrieval. In: Proceedings of the 21st annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 275–281
163. Radlinski F, Joachims T (2005) Query chains: Learning to rank from implicit feedback. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, ACM, New York, NY, USA, KDD '05, pp 239–248
164. Rekabsaz N, Lupu M, Hanbury A (2016) Uncertainty in neural network word embedding exploration of potential threshold. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
165. Rekabsaz N, Lupu M, Hanbury A, Zuccon G (2016) Generalizing translation models in the probabilistic relevance framework. CIKM'16
166. Robertson S (2004) Understanding inverse document frequency: on theoretical arguments for idf. Journal of documentation 60(5):503–520
167. Rocchio JJ (1971) Relevance feedback in information retrieval. The Smart Retrieval System-Experiments in Automatic Document Processing pp 313–323
168. Roy D, Ganguly D, Mitra M, Jones GJ (2016) Word vector compositionality based relevance feedback using kernel density estimation. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, New York, NY, USA, CIKM '16, pp 1281–1290
169. Roy D, Paul D, Mitra M (2016) Using word embeddings for automatic query expansion. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
170. Rumelhart DE, Hinton GE, Williams RJ (1988) Learning representations by back-propagating errors. Cognitive modeling 5(3):1
171. Salakhutdinov R, Hinton G (2009) Semantic hashing. International Journal of Approximate Reasoning 50(7):969–978
172. Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Networks 61:85–117, DOI 10.1016/j.neunet.2014.09.003

173. Sennrich R, Haddow B, Birch A (2016) Neural machine translation of rare words with subword units. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, pp 1715–1725
174. Severyn A, Moschitti A (2015) Learning to rank short text pairs with convolutional deep neural networks. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 373–382
175. Shen Y, He X, Gao J, Deng L, Mesnil G (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, ACM, pp 101–110
176. Shen Y, He X, Gao J, Deng L, Mesnil G (2014) Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the 23rd International Conference on World Wide Web, ACM, pp 373–374
177. Shokouhi M, Radinsky K (2012) Time-sensitive query auto-completion. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 601–610
178. Smolensky P (1986) Information processing in dynamical systems: Foundations of harmony theory. Tech. rep., DTIC Document
179. Socher R, Chen D, Manning CD, Ng A (2013) Reasoning with neural tensor networks for knowledge base completion. In: Advances in Neural Information Processing Systems, pp 926–934
180. Song Y, Elkahky AM, He X (2016) Multi-rate deep learning for temporal recommendation. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR ’16, pp 909–912
181. Sordoni A, Bengio Y, Nie JY (2014) Learning concept embeddings for query expansion by quantum entropy minimization. In: AAAI, pp 1586–1592
182. Sordoni A, Bengio Y, Vahabi H, Lioma C, Grue Simonsen J, Nie JY (2015) A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM), ACM, pp 553–562
183. Suggu SP, Goutham KN, Chinnakotla MK, Shrivastava M (2016) Deep feature fusion network for answer quality prediction in community question answering. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
184. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp 3104–3112
185. Tai KS, Socher R, Manning CD (2015) Improved semantic representations from tree-structured long short-term memory networks. 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), Association for Computational Linguistics, Beijing, China, pp 1556–1566
186. Turney PD, Pantel P (2010) From frequency to meaning: Vector space models of semantics. J Artif Intell Res (JAIR) 37:141–188, DOI 10.1613/jair.2934
187. Van Gysel C, de Rijke M, Kanoulas E (2016) Learning latent vector spaces for product search. In: CIKM 2016: 25th ACM Conference on Information and Knowledge Management, ACM
188. Van Gysel C, de Rijke M, Worring M (2016) Unsupervised, efficient and semantic expertise retrieval. In: Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, WWW ’16, pp 1069–1079, DOI 10.1145/2872427.2882974
189. Vulic I, Moens MF (2015) Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 363–372
190. Wan J, Wang D, Hoi SCH, Wu P, Zhu J, Zhang Y, Li J (2014) Deep learning for content-based image retrieval: A comprehensive study. In: Proceedings of the 22nd ACM International Conference on Multimedia, ACM, pp 157–166
191. Wang D, Nyberg E (2015) A long short-term memory model for answer sentence selection in question answering. ACL, July
192. Wang M, Smith NA, Mitamura T (2007) What is the jeopardy model? a quasi-synchronous grammar for qa. In: EMNLP-CoNLL, vol 7, pp 22–32

193. Wei X, Croft WB (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, ACM, pp 178–185
194. Weiss Y, Torralba A, Fergus R (2009) Spectral hashing. In: Advances in Neural Information Processing Systems, pp 1753–1760
195. Weston J, Bengio S, Usunier N (2010) Large scale image annotation: learning to rank with joint word-image embeddings. Machine learning 81(1):21–35
196. Weston J, Chopra S, Bordes A (2014) Memory networks. In: International Conference on Learning Representations (ICLR)
197. Wu Q, Burges CJ, Svore KM, Gao J (2010) Adapting boosting for information retrieval measures. Information Retrieval 13(3):254–270
198. Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, et al (2016) Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:160908144
199. Xia L, Xu J, Lan Y, Guo J, Cheng X (2015) Learning maximal marginal relevance model via directly optimizing diversity evaluation measures. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 113–122
200. Xia L, Xu J, Lan Y, Guo J, Cheng X (2016) Modeling document novelty with neural tensor network for search result diversification. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, pp 395–404
201. Xu C, Bai Y, Bian J, Gao B, Wang G, Liu X, Liu TY (2014) Rc-net: A general framework for incorporating knowledge into word representations. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, pp 1219–1228
202. Xu K, Ba J, Kiros R, Cho K, Courville AC, Salakhutdinov R, Zemel RS, Bengio Y (2015) Show, attend and tell: Neural image caption generation with visual attention. CoRR abs/1502.03044
203. Yan R, Song Y, Wu H (2016) Learning to respond with deep neural networks for retrieval-based human-computer conversation system. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 55–64
204. Yang J, Stones R, Wang G, Liu X (2016) Selective term proximity scoring via BP-ANN. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
205. Yang L, Ai Q, Guo J, Croft WB (2016) aNMM: Ranking short answer texts with attention-based neural matching model. In: The 25th ACM International Conference on Information and Knowledge Management, Indianapolis, United States
206. Yang X, Macdonald C, Ounis I (2016) Using Word Embeddings in Twitter Election Classification. In: ACM SIGIR Workshop on Neural Information Retrieval (Neu-IR)
207. Ye X, Qi Z, Massey D (2015) Learning relevance from click data via neural network based similarity models. In: Big Data (Big Data), 2015 IEEE International Conference on, IEEE, pp 801–806
208. Ye X, Shen H, Ma X, Bunescu R, Liu C (2016) From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 404–415
209. Yi X, Allan J (2009) A comparative study of utilizing topic models for information retrieval. In: European Conference on Information Retrieval, Springer, pp 29–41
210. Yu D, Deng L (2011) Deep learning and its applications to signal and information processing. IEEE Signal Processing Magazine 28(1):145–154
211. Yu L, Hermann KM, Blunsom P, Pulman S (2014) Deep learning for answer sentence selection. arXiv preprint arXiv:14121632
212. Zamani H, Croft WB (2016) Embedding-based query language models. In: Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval, pp 147–156
213. Zamani H, Croft WB (2016) Estimating embedding vectors for queries. In: Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval, ACM, pp 123–132
214. Zhai S, Chang Kh, Zhang R, Zhang Z (2016) Attention based recurrent neural networks for online advertising. In: Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, WWW '16 Companion, pp 141–142
215. Zhai S, Chang Kh, Zhang R, Zhang ZM (2016) Deepintent: Learning attentions for online advertising with recurrent neural networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '16, pp 1295–1304

216. Zhang Q, Kang J, Qian J, Huang X (2014) Continuous word embeddings for detecting local text reuses at the semantic level. In: Proceedings of the 37th international ACM SIGIR Conference on Research & development in information retrieval, ACM, pp 797–806
217. Zhang W, Du T, Wang J (2016) Deep learning over multi-field categorical data. In: European Conference on Information Retrieval, Springer, pp 45–57
218. Zhang X, Zhao J, LeCun Y (2015) Character-level convolutional networks for text classification. In: Advances in Neural Information Processing Systems, pp 649–657
219. Zhang Y, Wallace B (2015) A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:151003820
220. Zhang Y, Roller S, Wallace BC (2016) MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, San Diego, California, pp 1522–1527
221. Zhang Y, Lease M, Wallace BC (2017) Exploiting domain knowledge via grouped weight sharing with application to text categorization. arXiv preprint arXiv:170202535
222. Zheng G, Callan J (2015) Learning to reweight terms with distributed representations. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR '15, pp 575–584
223. Zhou G, He T, Zhao J, Hu P (2015) Learning continuous word embedding with metadata for question retrieval in community question answering. In: Proceedings of ACL, pp 250–259
224. Zhu Y, Lan Y, Guo J, Cheng X, Niu S (2014) Learning for search result diversification. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, ACM, pp 293–302
225. Zuccon G, Koopman B, Bruza P, Azzopardi L (2015) Integrating and evaluating neural word embeddings in information retrieval. In: Proceedings of the 20th Australasian Document Computing Symposium, ACM, p 12