Understanding Short Texts through Semantic Enrichment and Hashing

Zheng Yu, Haixun Wang, Xuemin Lin, *Senior Member, IEEE*, and Min Wang

Abstract—Clustering short texts (such as news titles) by their meaning is a challenging task. The semantic hashing approach encodes the meaning of a text into a compact binary code. Thus, to tell if two texts have similar meanings, we only need to check if they have similar codes. The encoding is created by a deep neural network, which is trained on texts represented by word-count vectors (bag-of-word representation). Unfortunately, for short texts such as search queries, tweets, or news titles, such representations are insufficient to capture the underlying semantics. To cluster short texts by their meanings, we propose to add more semantic signals to short texts. Specifically, for each term in a short text, we obtain its concepts and co-occurring terms from a probabilistic knowledge base to enrich the short text. Furthermore, we introduce a simplified deep learning network consisting of a 3-layer stacked auto-encoders for semantic hashing. Comprehensive experiments show that, with more semantic signals, our simplified deep learning model is able to capture the semantics of short texts, which enables a variety of applications including short text retrieval, classification, and general purpose text processing.

Index Terms—Short text, semantic enrichment, semantic hashing, deep neural network

1 Introduction

A wide range of applications handle short texts. For example, news recommendation system need to process the news titles which may be not strictly syntactical; in web search, queries consist of a very small number of keywords. Short texts introduce new challenges to many text-related tasks including information retrieval (IR), classification, and clusterings. Unlike long documents, two short texts that have similar meaning do not necessarily share many words. For example, the meanings of "upcoming apple products" and "new iphone and ipad" are closely related, but they share no common words. The lack of sufficient statistical information leads to difficulties in effectively measuring similarity, and as a result, many existing text analytics algorithms do not apply to short texts directly. More importantly, the lack of statistical information also means problems that can be safely ignored when we handle long documents become critical for short texts. Take *polysemy* as an example. The word "apple" gives rise to different meanings in "apple product" and "apple tree". Due to the scarcity of contextual information, these ambiguous words make short texts hard to understand by machines.

In this paper, we propose a novel approach for understanding short texts. Our approach has two components: i) a semantic network based approach for enriching a short text; and ii) a deep neural network (DNN) based approach for revealing the semantics of a short text based on its enriched representation.

1.1 Enriching Short Texts

Many approaches have been proposed to facilitate short text understanding by enriching the short text. For example, we may treat a short text as a search query and enrich it using the results from a search engine (e.g., webpage titles and snippets) [1], [2], [3]. More effectively, a short text can be enriched with explicit semantic information derived from external resources such as WordNet [4], [5], Wikipedia [6], [7], the Open Directory Project (ODP) [8], etc. For example, we may map a short text to a set of Wikipedia concepts (titles of Wikipedia articles), then we enrich it with the corresponding Wikipedia articles and categories.

These methods, however, have their limits. Search-based approaches may work well for so-called head queries, but for tail or unpopular queries, it is very likely that some of the top search results are irrelevant, which means the enriched short text is likely to contain a lot of noise. On the other hand, methods based on external resources are constrained by the coverage of these resources. Take WordNet for example, WordNet does not contain information for proper nouns, which prevents it to understand entities such as "USA" or "IBM." For ordinary words such as "cat", WordNet contains detailed information about its various senses. However, much of the knowledge is of linguistic value, and is rarely evoked in daily usage. For example, the sense of "cat" as *gossip* or *woman* is rarely encountered. Unfortunately, WordNet does not weight senses based on their usage, and these rarely used senses often give rise to misinterpretation of short texts. In summary, without knowing the distribution of the senses, it is difficult to build an inferencing mechanism to choose appropriate senses for a word in a context.

- *Z. Yu is with the East China Normal University, Shanghai, 200062, China. E-mail: zyu.0910@gmail.com.*
- *H. Wang and M. Wang are with the Google Research, Mountain View, CA 94043. E-mail: {haixun, minwang}@google.com.*
- *X. Lin is with the East China Normal University, Shanghai, 200062, China, and the CSE, University of New South Wales, Sydney, N.S.W., Australia. E-mail: lxue@cse.unsw.edu.au.*

Manuscript received 28 Apr. 2015; revised 21 Sept. 2015; accepted 21 Sept. 2015. Date of publication 1 Oct. 2015; date of current version 6 Jan. 2016. Recommended for acceptance by P. G. Ipeirotis.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2015.2485224

TABLE 1
Enrichment Results for "Upcoming Apple Products" and "New iphone and ipad"

short texts	concepts	co-occurring terms
upcoming **apple products**	corporation, company, firm, large company, stock; product;...	mac,iphone, updates, sales, service, forums, online,community;...
new **iphone** and **ipad**	mobile device, device, portable device, apple device, ios device;...	ipod, mac, updates, forums, ipad, ipod touch;...

We argue that in order to understand short texts, we need *conceptual knowledge* and the ability to *conceptualize*, which aims to get the appropriate sense (called concept) for a noun term (a single word or a phrase) under different contexts. For example, given the word "apple," we conceptualize it to concepts such as *fruit* and *company*. Then, given a context, for example, a short text "upcoming apple products", we conceptualize "apple" to the more appropriate concept *company*. In other words, we map a short text to a high dimensional concept space, and the mapping handles disambiguation such that concepts inappropriate for that context are filtered. In this paper, we propose a multiple inferencing mechanism called *conceptualization* to get the most appropriate sense for a term under different contexts. The concept space we employ is provided by Probase [9], [10], [11], which contains millions of fine-grained, interconnected, probabilistic concepts. The concept information is more powerful in capturing the meaning of a short text because it explicitly expresses the semantic.

However, conceptualization alone is still not enough for tasks such as comparing two short texts or classifying short texts. Consider the same two short texts: "upcoming apple products" and "new iphone and ipad". After conceptualization, we get a set of concepts for each short text (see Table 1), but there are still no common terms. To reveal the similarity of their semantics, we further built an inferencing mechanism on Probase to extract certain popular co-occurring terms for each original noun term, and these can be seen as new contexts for that short text. Table 1 also shows the co-occurring terms for the two example short texts. We can see that they share some terms including "mac", "updates", "forums", etc. These common contextual information relates the two short texts in the semantic space.

1.2 Semantic Hashing through a Deep Neural Network

Semantic hashing is a new information retrieval method that hashes texts into compact binary codes using deep neural networks. It can be viewed as a strategy do transform texts from a high dimensional space into a low-dimension binary space, and meanwhile the semantic similarity between texts is preserved by the compact binary codes as much as possible. Therefore, retrieving semantically related texts is efficient: we simply return texts whose codes have small Hamming distances to that of query [12]. Semantic hashing has two main advantages: First, with non-linear transformations in each layer of the deep neural network, the model has great expressive power in capturing the abstract and complex correlations between the words in a text, and hence the meaning of the text; Second, it is able to represent a text by a compact, binary code, which enables fast retrieval.

Salakhutdinov and Hinton [13] first proposed a semantic hashing model based on Restricted Boltzmann Machines (RBMs) [14] for long documents, and the experiments showed that their model achieved comparable accuracy with the traditional methods, including Latent Semantic Analysis (LSA) [15] and TF-IDF [16], [17]. Instead of using RBMs-based model, we construct a deep neural network with a three-layer stacked auto-encoders to perform semantic hashing for short texts. Each auto-encoder has specific learning functions, and we implement a two-stage semi-supervised training strategy, including a hierarchical pre-training and an overall fine-tuning process, to train the model. Compared with the BRMs-based model [13] which has complex transformations during the training process, our network enjoys a simpler and faster training procedure but has a comparative learning ability. Specifically, RBMs are generative models whose gradient computation (used for training) is intractable and requires approximation, whereas the auto-encoders are deterministic and have a simpler training objective – reconstructing the inputs. We show that our DNN model is more efficient than the RBMs-based model.

We carry out extensive experiments on tasks including information retrieval and classification for short texts. We show significant improvements over existing approaches, which confirm that i) concepts and co-occurring terms effectively enrich short texts, and enable better understanding of them; ii) our auto-encoder based DNN model is able to capture the abstract features and complex correlations from the input text such that the learned compact binary codes can be used to represent the meaning of that text.

Contributions. In summary, our main contributions are:

(1) We present a novel mechanism to semantically enrich short texts with both concepts and co-occurring terms, such external knowledges are inferred from a large scale probabilistic knowledge base using our proposed thorough methods;
(2) We construct a more efficient deep neural network based on a three-layer stacked auto-encoders to do semantic hashing for short texts. For each auto-encoder we design a specific and effective learning strategy to capture useful features from input data.
(3) We provide a way to combine knowledge information and deep neural network for text analysis, so that it helps machines better understand short texts.

Outline. Section 2 briefly introduce the Probase, backpropagation and auto-encoder. Section 3 shows our proposed mechanism to enrich short texts with concepts and co-occurring terms. The architecture and learning process for our DNN model are introduced at length in Section 4. We evaluate our method and show experimental results obtained on information retrieval and classification tasks in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 Preliminaries

In this section, we describe i) Probase, a large-scale semantic network with millions of fine-grained, interconnected, and

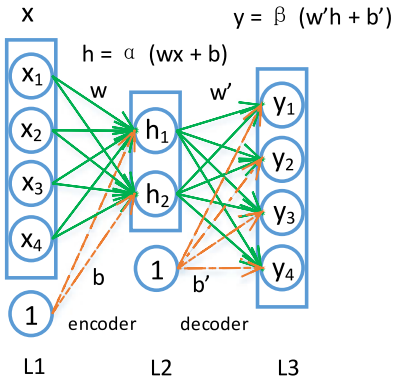

Fig. 1. A three-layer neural network (auto-encoder).

probabilistic concepts, ii) backpropagation, an effective method to train neural network, and iii) auto-encoder, an unsupervised learning algorithm that learns features from unlabeled data.

2.1 Probase

Probase [9] is a large-scale probabilistic semantic network that contains millions of concepts of worldly facts. These concepts are harvested using syntactic patterns (such as the Hearst patterns) from billions of webpages. For each concept, it also finds its instances and attributes. For example, *company* is a concept, and it is connected to instances such as *apple* and *microsoft*. Moreover, Probase scores the concepts and instances, as well as their relationships. For example, for each concept c and each of its instance t, Probase gives $p(t|c)$, the *typicality* of instance t among all instances of concept c, and $p(c|t)$, the *typicality* of concept c among all concepts that have instance t. Furthermore, Probase gives $p(t_1|t_2)$, the co-occurrence probability between the two instances t_1 and t_2. In addition, there are many other probabilities and scores in Probase. These abundant information allows us to build inferencing mechanisms for text analysis and text understanding.

Compared with other knowledge bases such as WordNet, Wikipedia, and ODP, Probase has two advantages. First, the rich concept information enables interpretation at fine levels. For example, given "China, India", the top concepts Probase returns are *country, Asian country*. Given "China, India, Brazil", the top concepts become *developing country, BRIC country, emerging market*. Other knowledge bases do not have a fine-grained concept space, nor an inferencing mechanism for the concept, therefore they can at most map them into the concept of *country*, which is often too general and coarse level for sophisticated text understanding. Second, the probabilistic nature of Probase allows us to build inferencing mechanisms to map words in a context to appropriate fine-grained concepts. This allows us to perform text analytics in the concept space, which contains much richer information than the original short text, which is often sparse, noisy, and ambiguous.

2.2 Backpropagation

Backpropagation [18] is a common method for training artificial neural networks. It is a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions [19]. Fig. 1 shows a simple three-layer neural network, where L_1 is an input layer consisting of four nodes, L_2 is a hidden layer and L_3 is a output layer. The functions α and β are two activation functions acted on L_2 and L_3 respectively. Generally, let i be any node in a layer, and let j be a node that feeds into it (which means there is a directed edge $j \rightarrow i$ from j to i). We use w_{ij} to represent the weight of edge $j \rightarrow i$ and b_i to represent the bias on node i. The value z_i computed on node i is $z_i = f(\Sigma_j w_{ij} z_j + b_i) = f(a_i)$, where f is the activation function and a_i is called the total activation coming into node i.

A neural network aims at finding proper parameters (W, b) in the hypothetic space to make its output y approximate the target t. We define a loss function J with respect to y and t to measure the network's performance. For example, J is a squared loss: $\frac{1}{2}||y - t||^2$.

To train the neural network, we apply an optimization algorithm such as gradient descent [20] to update the parameters to minimize loss J for all training examples.

One iteration of gradient descent updates any parameter w_{ij}, b_i as follows:

$$\begin{aligned} w_{ij} &= w_{ij} - \epsilon \Delta w_{ij}, \\ b_i &= b_i - \epsilon \Delta b_i, \end{aligned} \tag{1}$$

where ϵ is a small positive number (e.g., 0.1) called learning rate, which determines the step size in the gradient descent search. The update Δ is the partial derivative of J with respect to w_{ij} or b_i, i.e., $\Delta w_{ij} = \frac{\partial J}{\partial w_{ij}}$ and $\Delta b_i = \frac{\partial J}{\partial b_i}$.

Backpropagation [18] algorithm gives an efficient way to compute the updates Δw_{ij} and Δb_i. Given one training example (x, t), where x is the input and t is the target. Using the chain rule for derivatives, we further get $\Delta w_{ij} = z_j \delta_i$ and $\Delta b_i = \delta_i$, where $\delta_i = \partial J(x,t)/\partial a_i$, which is the derivative of $J(x,t)$ with respect to the total activation a_i coming into node i, and z_j is just the output value of node j in former layer. Obviously, to get the partial derivatives, the critical step is to compute δ on each node. The principal steps of backpropagation are as follows:

1) Compute $\delta_i = \frac{\partial L(z_i, t_i)}{\partial z_i} f'(a_i)$ for all output nodes i, where $L(z_i, t_i)$ is the local loss for node i, such as $L(z_i, t_i) = \frac{1}{2}(z_i - t_i)^2$;
2) Working backwards, for other nodes j compute δ_j using

$$\delta_j = f'(a_j) \sum_i \delta_i w_{ij}. \tag{2}$$

3) Compute the updates $\Delta w_{ij} = \frac{\partial J(x,t)}{\partial w_{ij}} = \delta_i z_j$ and $\Delta b_i = \frac{\partial J(x,t)}{\partial b_i} = \delta_i$ respectively for every parameter w_{ij} and b_i;

Once scanning all training examples, each parameter is updated by the average value of its partial derivatives w.r.t all the examples. Through backpropagation algorithm, we can iteratively train the neural network until the loss converges or the training process gets to a predefined maximum iteration.

2.3 Auto-Encoder

The auto-encoder [21] is an unsupervised learning algorithm that automatically learns features from unlabeled

data. It is actually a three-layer neural network, and the learning process consists of two main stages, namely the encoder and the decoder (see Fig. 1).

The encoder works as follows: it takes a vector $\mathbf{x} \in \Re^d$ as input and encodes it to a code (hidden feature) $\mathbf{h} \in \Re^{d'}$ through a deterministic mapping, that is

$$\mathbf{h} = \alpha(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where α is the activation function, $\mathbf{W}$ is a $d' \times d$ dimensional weight matrix, and $\mathbf{b}$ is a d' dimensional bias vector.

The decoder needs to reconstruct the input: the code $\mathbf{h}$ is further decoded to a $\mathbf{y}$ which is seen as a prediction of $\mathbf{x}$, that is

$$\mathbf{y} = \beta(\mathbf{W}'h + \mathbf{b}'),$$

where β is a similar activation function, $\mathbf{W}'$ is a $d \times d'$ dimensional weight matrix, and $\mathbf{b}'$ is a d dimensional bias vector. (here, $'$ is not transpose.)

The auto-encoder aims at minimizing the reconstruction error between $\mathbf{x}$ and $\mathbf{y}$, which is measured by a loss function J. For example, if $\mathbf{x}$ and $\mathbf{y}$ are real-valued vectors, we can use the traditional squared loss $J = \frac{1}{2}||\mathbf{y} - \mathbf{x}||^2$ as the loss function. After training, the code $\mathbf{h}$ is believed to contain hidden features that reflect main factors of variations in the data.

3 Enriching Short Texts

We propose a mechanism to semantically enrich short texts using Probase. Given a short text, we first identify the terms that Probase can recognize, then for each term we perform conceptualization to get its appropriate concepts, and further infer the co-occurring terms. We denote this two-stage enrichment mechanism as Concepts-and-Co-occurring Terms (**CACT**). After enrichment, a short text is represented by a set of semantic features and is further denoted as a vector that can be fed to our DNN model to do semantic hashing. We give related definitions as below:

- *Semantic features*—the elementary unit terms that compose (enriched) short texts. The semantic features include original terms, those inferred concepts and co-occurring terms.
- *Semantic feature vocabulary*—a vocabulary that consists of *top-k* (e.g., 3,000) most frequent semantic features obtained from the whole training data set.
- *Semantic feature vector*—a k-dimensional vector that represents a (enriched) short text, each element of the vector is the count of a semantic feature that occurs in the short text.

Next, we introduce how to produce these semantic features for short texts using our proposed enrichment mechanism—CACT. Note that in this paper, we focus on conceptualization and inferring co-occurring terms (do semantic enrichment) for noun phrases. Verbs and adjectives are also important as they can be useful for disambiguation and other tasks. We still remain them as the semantic features for the short text.

3.1 Pre-Processing Short Text

The goal of pre-processing is to break a short text into a set of terms that appear in Probase. That is, in pre-processing we treat the Probase semantic network as a dictionary of terms. Once we identify Probase terms in the short text, we can identify their positions in the semantic network. Then, their rich interconnections to other terms will become available to our analytical tools.

In finding Probase terms in a short text, we need to address the multi-word expression problem. We parse the input short text to find the *longest matching* terms in the Probase dictionary. For example, given "New York Times", the matched instance is *New York Times* itself, although the input contains another term *New York*. Longest matching alone may not always lead to correct result. For example, "april in paris" is the correct longest matching term if it appears with "lyrics". However, in query "april in paris weather", it needs to be broken down into "april" and "paris". Before parsing, we also need to carry out such tasks as word stemming and stopwords removal.[1]

3.2 Conceptualization

We conceptualize each preprocessed term to obtain a set of concepts. In other words, we map a short text to the concept space. Conceptualizing a single term is simply a look-up process, and we output a set of concepts which are ranked by their typicalities (that is, $p(c|t)$). For a short text consisting of two or more terms, which are denoted by $E = \{e_i, i \in 1, \ldots, M\}$, we first map each term to a set of concepts. Considering the true sense of a term is heavily affected by its neighbors, especially for the ambiguous ones, we then need a mechanism to combine the multiple sets of concepts to get the most appropriate sense for each term. In our work, we use multiple mechanisms to do this.

Mechanism 1. Song et al. [10] proposed a simple Naïve Bayesian mechanism to estimate the posterior probabilities of concepts given two or more terms. Effectively, common concepts associated with all of the terms get high posterior probabilities. Formally, the probability of a concept c is:

$$p(c|E) = \frac{p(E|c)p(c)}{p(E)} \propto p(c)\prod_i^M p(e_i|c).$$

For example, given a short text "apple and microsoft," the posterior probabilities of concepts such as *company* and *software company* are much higher than that of *fruit*. This enables us to prune unlikely concepts such as *fruit*. Therefore for those terms in E that have common concepts, we just apply this Naïve Bayesian method to get their most possible concepts.

Mechanism 2. However, we must avoid over generalization for input text that contains multiple concepts. For example, consider the short text "Jobs of apple and Ballmer of microsoft" that contains two main concepts, namely *CEO* and *company*. Naively applying the Bayesian approach as mentioned above means we assume there is only a single concept in the short text. The result is the common concepts of *CEO* and *company*, which are likely to be very vague

1. http://www.ranks.nl/resources/stopwords.html

concepts such as *object*. To solve this problem, we i) group certain terms (e.g., apple and microsoft) together for conceptualization and union the results of different groups (first do the AND operation, then do the OR operation); or ii) conceptualize certain themes (e.g., Jobs and apple) separately and intersect the result from a same theme (first do the OR operation, then do the AND operation). To do this, we first do clustering on the Probase terms based on their co-occurrence relationship, after that, to determine whether two terms are belonged to a same group or theme, we just need to see if they are in a same cluster.

Mechanism 3. Furthermore, we still need to disambiguate the terms even if they are conceptualized in separate groups. For example, consider short text "ipad and apple", which consists of two terms that do not share common concepts (*device*, *fruit*, and *company* are largely independent concepts). If we union their conceptualization results, *fruit* will remain as one of the final concepts. Probase does not explicitly capture the specific semantic relationships between any two terms (e.g., Probase does not know ipad *is a product of* apple). Still, in many cases, the co-occurrence relationship is strong enough for disambiguation. Consider those terms that frequently co-occur with both "ipad" and "apple", the conceptualization results of them do not contain the concept of *fruit*. This enables us to assign high probabilities to *device*-related concepts and *company*-related concepts for "ipad" and "apple" respectively. Alternatively, we use a probabilistic topic model–LDA [22] to model the co-occurrence relationship, as it is able to estimate how words are semantically related based on their co-occurrence statistics. Using the method proposed in [11], we first get a trained topic model C that is used to infer the topic distribution of a given short text s. Let $\vec{s}$ be the sequence of term indices of s, and $\vec{z}$ be the topic assignment vector of $\vec{s}$. Based on model C, the posterior of $\vec{z}$ is inferred by using collapsed Gibbs sampling as follows:

$$p(z_i = k|\vec{s}, C) \propto \left(\sum_w n_{wk} + \alpha\right) \times \frac{C_{s_i k} + n_{s_i k} + \beta}{\sum_w C_{wk} + n_{wk} + N\beta}, \quad (3)$$

where C_{wk} is the number of times term w is assigned to topic k, and n_{wk} is the number of times term w is assigned to topic k of sentence s. N is the size of the vocabulary, and we use α and β to denote the hyper-parameters for document-topic and word-topic distributions respectively. Through this method, we can estimate the posterior probability $p(z_i)$ of each term s_i in the text s.

We then further compute the probability of concept c given an instance term w of s based on the topic distribution $\vec{z}$ as follows:

$$p(c|w, \vec{z}) \propto p(c|w) \sum_k p_{wk} p_{ck},$$
$$p_{wk} = p(z_w = k|\vec{s}, C),$$
$$p_{ck} = \frac{C_{ck} + \beta}{\sum_w C_{wk} + N\beta},$$

where w and c are indices of the instance term and concept term respectively, $p(c|w)$ is the typicality of concept c given the term w, which can be extracted from Probase, p_{wk} is the inferred probability through Equ. (3), and p_{ck} is the probability of concept c given topic k.

Finally, for each term we obtain a set of concepts, each associated with a probability. We select a canonical concept which has the highest probability for that term and regard the rest as member concepts. Table 2 shows the canonical and top member concepts for the word *ipad*, *apple* in the short text "ipad and apple."

TABLE 2
The Conceptualization Result for "ipad and apple"

term	canonical concept	member concepts
ipad	device	mobile device, portable device, tablet device, apple device, etc.
apple	company	corporation, firm, technical company, stock, etc.

3.3 Co-Occurring Terms

To further enrich a short text, we include some external terms that frequently co-occur with the original ones. Distributional hypothesis [23] says "words that occur in the same context tend to have similar meanings", co-occurrences are valuable statistics for understanding the meaning of words. In our work, we regard terms as meaningful concepts in the highly connected semantic network (Probase) instead of plain words, we could tell whether the co-occurring terms are semantically related to the context.

We define the **co-occurrence score** to measure the probability of one term o that co-occurs with a target term t in a short text s. We consider two types of scores: co-occurrence probability and semantic similarity. We believe that a co-occurring term o should not only have a high co-occurrence probability with t (that is, $p(o|t)$), but also has consistent semantic with t under the context s. For example, given a short text "bluetooth mouse", although "cat, dog, keyboard, printer" have high co-occurrence probabilities with "mouse" in Probase, only "keyboard" and "printer" are true co-occurring terms if we take into consideration the semantic(concept) of "mouse" in that text. Therefore, the co-occurrence score between o and t under a short text s can be calculated as follows:

$$S(o|t, s) = \alpha S_{co}(o|t) + (1 - \alpha) S_{se}(o|t, s),$$

where α is a meta parameter that explores the trade-off between the two component scores. $S_{co}(o|t)$ is the co-occurrence probability between instance terms o and t, and it is equal to $p(o|t)$, which is also defined in Probase. $S_{se}(o|t, s)$ measures the semantic similarity between o and t under the text s, since we already know the concept c of term t, we aim to make the co-occurring term o have consistent semantic with c. Formally, $S_{se}(o|t, s)$ is defined as:

$$S_{se}(o|t, s) = \sum_{c_i} p(c_i, c) p(c_i|o).$$

c_i is a concept of term o, and $p(c_i, c)$ is the co-occurrence probability between concepts c_i and c. The co-occurrence probability between concepts can be inferred from the original co-occurrence instance network in Probase, because each concept can be represented as a distribution of

TABLE 3
The Semantic Features of Enriched "ipad and apple"

original terms	concepts	co-occurring terms
ipad, apple	mobile device, portable device, tablet device, device; company, technical company, corporation, firm.	iphone, ipod, accessories, apple, computer, factory, mac, sales, tablet; microsoft, itunes, ibm, google, mac, samsung, products, sony, dell, etc.

instances. Readers can refer to [24] for more details on the construction of concept network.

Through this method, we can infer semantic similar co-occurring terms for each term t that has a concept c under a short text s. Take "mouse" in short text "bluetooth mouse" as an example, we consider its two candidate co-occurring terms "cat" and "keyboard", and get the co-occurrence probabilities $p(cat|mouse) = 0.016$ and $p(keyboard|mouse) = 0.0023$. While given the concept of "mouse" is "brand", the semantic similarities of them are 0.0031 and 0.028 respectively. Therefore, if $\alpha = 0.4$, the final co-occurrence score of "cat" is 0.00826, and the score of "keyboard" is 0.01772. Therefore, "keyboard" (in top-10 ranked list) is far more likely to be co-occurred with "mouse" in the short text than "cat" (outside top-80 ranked list).

After enrichment, a short text is represented by a set of *semantic features*, which include its original terms, the introduced concepts and co-occurring terms (if a short text is not enriched, the semantic features only consists of its original terms). Table 3 shows the three types of semantic features of the enriched short text "ipad and apple". We then select the *top-k* (e.g., 3,000) most frequent semantic features obtained from the whole training set to construct a *semantic features vocabulary*. Based on the vocabulary, each (enriched) short text is represented as a k-dimensional vector, where each dimension corresponds to a feature and the element is the count of that feature appearing in the (enriched) short text.

4 Model Design

4.1 Problem Definition

Assume there are n training short texts in the dataset, denoted as: $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n\} \in \mathbf{R}^{k \times n}$, where k is the dimensionality of the feature vector, and the row is indexed by position of the semantic feature in the vocabulary. The value x_j^i represents the frequency of feature j occurs in the (enriched) short text i. The objective of our model is to learn optimal binary hashing codes $\mathbf{Y} = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^n\} \in \{0,1\}^{m \times n}$ for the training set $\mathbf{X}$, and a hashing function $f : \mathbf{R}^m \rightarrow \{0,1\}^m$, which maps each short text to its hashing codes with m bits (i.e., $\mathbf{y}^i = f(\mathbf{x}^i)$).

4.2 Model Overview

The learning model we design, as shown in Fig. 2, is a four-layer deep neural network. The input layer is fed by a feature vector of one short text and the output represents the learned binary code for that text. We further propose a two stage semi-supervised approach to train the DNN model to make it capture semantic features from the input

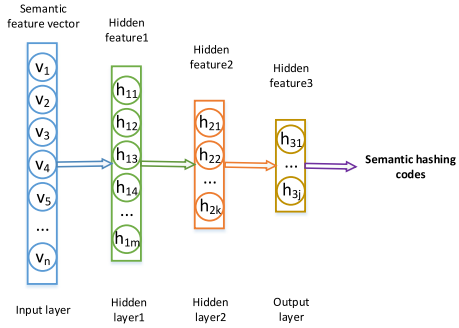

Fig. 2. The architecture of DNN model.

and finally encode those features into the optimal binary hashing codes. Specifically, we first treat the DNN model as a stacked auto-encoders, where each layer (except the input layer) is both a hidden layer of current auto encoder and a input layer of next auto-encoder (as shown in Fig. 3); we then greedily train each auto-encoder with backpropagation method. We call this stage pre-training procedure, and it is totally unsupervised. The philosophy of this design rests in the concept of stacking. The first auto-encoder tends to learn primary features of the raw input data and the following ones further capture more abstract features from its underlying input. After pre-training, the model is further trained as a whole network through a supervised method which aims to predict the label information (e.g., the tag or category) of input data, and we call this stage fine-tuning process.

Once the training process completes, the model is used to do semantic hashing for a given text s. As the raw output is a real valued vector, we further use a threshold to make the output to be binary such that 1) the codes preserve similarity which indicates that semantically similar short texts should be mapped to similar hashing codes within a short Hamming distance, and 2) each bit has an equal probability as positive (the bit is 1) or negative (the bit is 0). For clarity, in the following sections, we introduce the architecture details of our DNN model in terms of the training stages, that is, the pre-training and fine-tuning [25].

4.3 Pre-Training

In pre-training, each auto-encoder is trained as an independent neural network which aims to learn hidden features through reconstructing the input. The more similarly the auto-encoder reconstruct input, the better features the auto-encoder captures. With the final goal of semantic hashing, the three auto-encoders gradually reduce the dimensions of

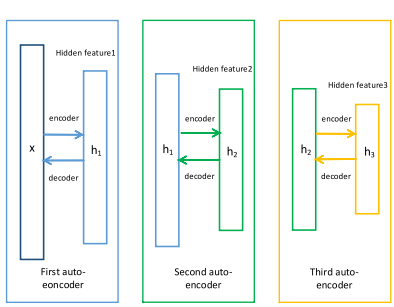

Fig. 3. Recursive pre-training.

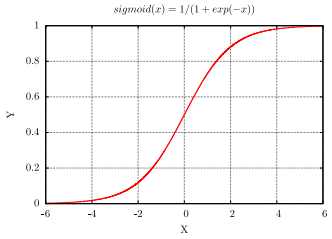

Fig. 4. Sigmoid function.

learned hidden features without losing much useful information. For the first auto-encoder, we directly map the sparse feature vectors to hundred dimensional hidden features, which are dense and sufficient to capture the useful information from the high dimensional sparse input data.

For any auto-encoder, let k, j, i represent a node in the input layer (L_1), a node in the hidden layer (L_2), and a node in the output layer (L_3), respectively (see Fig. 1). First, we use a same non-linear function acted on the hidden layer to learn hidden features. For any node j, the hidden feature learned on it is:

$$h_j = \alpha(b_j + \Sigma_k w_{jk} v_k).$$

Here, we use vector $\mathbf{v}$ to represent the input data of every auto-encoder. Obviously, for the first auto-encoder, $\mathbf{v}$ is equal to one $\mathbf{x}^i \in \mathbf{X}$ (with dimensionality of k); and for the second auto-encoder, $\mathbf{v}$ is the hidden feature $\mathbf{h}$ learned by the first auto-encoder. α is a sigmoid function, which is $y = \alpha(x) = 1/(1 + e^{-x})$. The sigmoid function is a bounded differentiable real function defined for all real input values and has a positive derivative everywhere. As Fig. 4 shows, it maps all real input values to the interval (0, 1) and the derivative with respect to input x is extremely simple if we express it in terms of output y: $dy/dx = y(1 - y)$, which facilitates the computation of partial derivatives in backpropagation [26]. However, for the output layers, they are designed to have different activation functions and to make use of different loss functions according to the objectives of auto-encoders.

4.3.1 The First Auto-Encoder

The first auto-encoder should take one semantic feature vector $\mathbf{x}$ (i.e., $\mathbf{x}^i$) as the input, and we further normalize $\mathbf{x}$ as:

$$v_j = \frac{x_j}{\sum_{n=1}^{k} x_n}.$$

The first auto-encoder aims to learn output $\mathbf{o}$ as the reconstruction of $\mathbf{v}$, and $\mathbf{v}$ can be seen as a probability distribution of semantic features. The normalization ensures that the values across all nodes sum up to 1, which makes learning stable and appropriately deals with texts of different lengths.

We take advantage of softmax as the activation function on the output layer to learn $\mathbf{o}$. Given a training example $\mathbf{v}$, the value o_i computed by softmax on node i is:

$$o_i = \frac{exp(a_i)}{\sum_{m=1}^{n} exp(a_m)},$$

where a_i is the total activation coming into node i, i.e., $a_i = b_i + \sum_j w_{ij} h_j$, and n is the number of nodes in the output layer. Obviously, the value of o_i is between 0 and 1. And thus the output $\mathbf{o}$ can be regarded as a predicted distribution and the target distribution is $\mathbf{v}$. We want to minimize the cross-entropy error between $\mathbf{o}$ and $\mathbf{v}$ at every node i, and the loss function is:

$$J = -\sum_i v_i \log o_i.$$

Then we get $\delta_i = o_i - v_i$, and the updates are:

$$\Delta w_{ij} = h_j \delta_i, \Delta b_i = \delta_i.$$

Furthermore, using backpropagation algorithm, the updates for w_{jk} and b_j are:

$$\Delta w_{jk} = v_k * h_j(1 - h_j) * \sum_i w_{ij} \delta_i,$$
$$\Delta b_j = h_j(1 - h_j) * \sum_i w_{ij} \delta_i.$$

In this pre-training process, we iteratively optimize the auto-encoder's parameters through gradient descent method to minimize loss J until J converges to a minima value, or the training process gets to a predefined maximum number (e.g., 100) of iterations.

4.3.2 The Second Auto-Encoder

Once the training of first auto-encoder is completed, we feed its learned hidden features to the second auto-encoder to do further training. As the input for the second auto-encoder, the values of the first learned hidden features are all between 0 and 1, so we use sigmoid function, which also maps data to internal (0, 1), on the output layer to compute $\mathbf{o}$ as the reconstruction of input vector $\mathbf{v}$. For one node i, the output value o_i computed on node i is:

$$o_i = \alpha\left(b_i + \sum_j w_{ij} h_j\right),$$

where α is the sigmoid function. Because the data in input $\mathbf{v}$ and output $\mathbf{o}$are all real values, we take advantage of simple squared error as loss function J:

$$J = \frac{1}{2}\sum_i (o_i - v_i)^2.$$

We can draw: $\delta_i = o_i(1 - o_i)(o_i - v_i)$, and through backpropagation algorithm, the updates for w_{ij}, b_i, w_{jk} and b_j in this auto-encoder can be efficiently computed.

In the entire deep neural network, the second auto-encoder constitutes a transition layer. It captures more abstract features so as to reduce the feature dimension. Theoretically, we can add more layers of such auto-encoders to build a deeper neural network, but it will increase the training cost and complexity, and it may not bring significant improvement to the expressive power of the deep learning model.

4.3.3 The Third Auto-Encoder (Denoising)

For the whole learning model, the third auto-encoder works as the last layer, whose hidden features are finally thresholded to be binary codes that represent the meaning of original input text. Therefore, besides reconstructing input data,

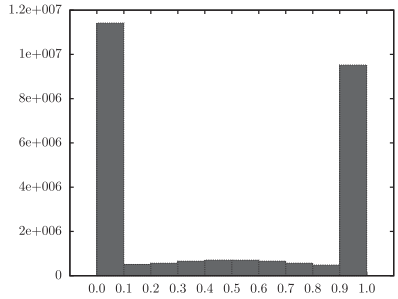

Fig. 5. The distribution of hidden features on third auto-encoder.

the other objective of the third auto-encoder is to make the learned hidden feature h_j to be binary as much as possible (that is, h_j should be close to 0 or close to 1). To enforce this, we add zero mean Gaussian noise to the total activation a_j coming into hidden node j. That is, the corrupted hidden feature h'_j is computed as:

$$h'_j = \alpha(a_j + \eta),$$

where η is the Gaussian noise and α is the sigmoid function.

For the third auto-encoder, in order to remove the added noise and reconstruct the input data, a_j (the total activation coming into hidden node j) must have large absolute values so that the Gaussian noise would have little or even no effect to the sigmoid result. The reason is: As Fig. 4 shows, if a_j has large absolute value, even though we add noise η to a_j, the original output $h_j(h_j = \alpha(a_j))$ of this sigmoid function will be slightly different from h'_j. So, during the training process, the third auto-encoder has to gradually optimizes its parameters to make the absolute values of a_j be large enough to get rid of the noise effect. Therefore, the hidden feature h_j is forced to be close to either 0 or 1. The effect of adding Gaussian noise is demonstrated in Fig. 5, where the hidden values are mostly pushed toward either 0 or 1, but are still good at approximating the input.

The third auto-encoder is able to denoise and capture the underlying features from its input data, so we call it a denoising auto-encoder. For the output layer, we still use sigmoid as the activation function and squared loss as the loss function J. Thus, the update rules for parameters are the same as those in the second auto-encoder.

4.4 Fine-Tuning

After pre-training, all of the three auto-encoders find good regions in the parameters space, but the parameters are not good enough for the whole model, so we combine the three successive encoding parts of these auto-encoders to construct a unified network, and add a prediction layer on top of it (Fig. 6), to further fine-tune the parameters. Specifically, the whole network aims to correctly predict the label information (e.g., tag or category) of input text, and the output **o** is the probability distribution over the labels. We use softmax as the activation function on the prediction layer, and we call the layer "softmax classifier". For any node i, the output o_i is computed as:

$$o_i = \frac{exp(a_i)}{\sum_{m=1}^{n} exp(a_m)},$$

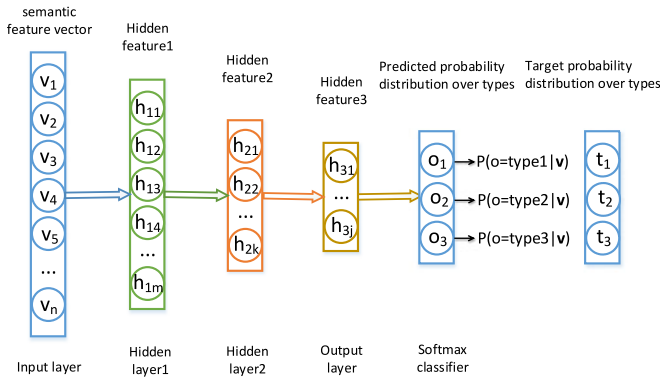

Fig. 6. The fine-tuning architecture of DNN model.

where n is the number of nodes on the prediction layer. The objective of fine-tuning is to minimize the cross-entropy error:

$$J = -\sum_i t_i \log\ o_i.$$

t_i is the target value, and it is set to 1 if its position index corresponds to the input's label. Fig. 6 shows the fine-tuning architecture of our DNN model, where the training data are assumed to have three categories.

For different properties of training data, we have different specific strategies to do the fine-tuning. If the training data is labeled with category or other distinct information, the prediction layer can be used to predict those information of input data, and hence, we fine-tune the model with a supervised learning strategy. While for the data without labels, we make use of the prediction layer to reconstruct again that raw input, which just likes the decoder procedure of an auto-encoder. Through this method, the semantically similar short texts, which have same label information or have similar feature distributions, will be mapped to similar hashing codes in the space.

Finally, given a short text, the output values produced by our DNN model are thresholded to be binary codes, which can be used to represent the semantics of that short text. As each bit of the codes has an equal probability to be positive (1) or negative (0), we set the threshold to be 0.5 since each dimension value is a real number between 0 and 1, and moreover, as shown in Fig. 5, due to the effect of Gaussian noise, the output values obtained from all training data almost have a symmetry distribution with 0.5. At last, to measure the similarity of any two short texts, we can simply feed them into our DNN model and compare the Hamming distance between their compact binary codes.

4.5 Training Details

Before start training the model, we randomize all the parameters, including weights W and bias b, using Gaussian distribution, where the mean and standard deviation is (0, 0.01).

In the optimization of network's parameters, instead of updating the parameters only after scanning all the training examples, we usually perform the update once an example is processed, and we call this method *stochastic gradient descent* (SGD) [27]. In our work, to further improve the

training efficiency, we divide the data into small mini-batches, which consists of a fixed number of training examples (we set the number to 200 and 1,000 for pre-training and fine-tuning respectively), and the parameters are updated by the average derivatives of loss J with respect to each mini-batch. We call this method *Mini-batches gradient descent* [13], and it can significantly speed up the training process because it computes the derivatives for many training examples simultaneously using matrix-matrix multiplies in memory.

Furthermore, we introduce momentum strategy to do updates using Equ. (1). Empirical evidence shows that the use of momentum can be helpful in speeding the convergence and avoiding false local minima. The idea about using a momentum is to stabilize the parameter change by making nonradical revisions using a combination of current gradient decreasing term with a fraction of the previous parameter change. For example, the current update $\Delta w(t)$ for parameter $w(t)$ is:

$$\Delta w(t) = \frac{\partial J}{\partial w(t)} + \lambda \Delta w(t-1),$$

where λ is taken $0 \leq \lambda \leq 0.9$, and t is the index of the current weight change. In both pre-training and fine-tuning, we apply the momentum strategy to facilitate the learning process.

In this work, we use a 3000-600-400-128 DNN architecture (the sizes of four layers are 3,000, 600, 400, and 128 respectively, which also means the vocabulary consists of top-$3{,}000$ most frequent semantic features) to perform semantic hashing. One training process continues until the value of loss function J obtained on the validation data set is converged. Then the output is thresholded to be a 128-dimensional (128D) binary code vector, which allows us to make use of fast bit counting routines to compute the Hamming distance between any two codes.

Please note that the setting of our model's variables, such as the number of layers, and the dimension of one layer, are explored and confirmed according to the performance obtained on the validation data set, and we take into consideration both prediction accuracy and training cost. For example, we observe that higher dimensional layers may slightly improve the retrieval precision, but could increase the training cost heavily.

5 Experiments

To evaluate the effectiveness of enriching short texts and DNN semantic hashing, we carried out two experiments. First, we perform an information retrieval experiment on MNS News data, then we do a classification task on partial Wikipedia data. The experiments clearly demonstrate the benefits of utilizing enrichment and deep neural network separately, while the combination of these two methods in the unified framework generates the best result.

5.1 Description of Data Sets

The MSN News data has six categories: business, sports, entertainment, weather, technology, and food&drink. Each category set consists of 12,000 news data, among which we randomly select 10,000 cases for training, 1,000 cases for validation and 1,000 cases for test. So, there are totally 60,000 training data, 6,000 validation data, and 6,000 test data. The Wikipedia data set consists of 330,000 articles in 110 categories (categorized based on the mapping between Wikipedia articles and Freebase[2] topics). For example, the Wikipedia article titled "The Big Bang Theory" is in the *tv_program* category defined by Freebase. For each category, we collect 2,000 articles as training data, 500 articles as validation data and 500 articles as test data.

We use the titles of MNS news as short texts, because the titles of news usually contain fewer but informative words which may be specific and ambiguous. While for Wikipedia data, compared with a whole document, one sentence has so little statistical information. So, in our experiments, we treat the first sentences of Wikipedia articles as short texts, and perform the classification task on them.

In those two experiments, for every noun term in a short text, we enrich it with the top-5 possible concepts and top-10 possible co-occurring terms. As a result, the representation of a short text is greatly enriched. For example, the average number of semantic features in the original MSN news titles is only 5.5, while it is increased to 64.4 after enrichment.

TABLE 4
The Training Times (s/epoch) Obtained by Mini-Batches and SGD

Method	1st layer	2nd layer	3rd layer	fine-tuning
Mini-batches	226	50	13	185
SGD	2383	589	162	2205

We used a single-threaded program on one machine powered by an Intel Core (TM) i5-2400 3.1-GHz with 8 GB memory, running Windows.

5.2 Effects of Mini-Batches and Momentum

We first present the efficiency improvements that are obtained by using mini-batches and momentum mechanisms to update network's parameters. We use MSN News as the training data for this experiment, and the number of training examples of each mini-batch in pre-training and fine-tuning is set to 200 and 1,000 respectively. Table 4 shows the time costs achieved by mini-batches method and SGD. Compared with SGD method, mini-batches gradient descent significantly decrease the training time, because it enables us to compute the derivatives for many training examples simultaneously using matrix-matrix multiplies in memory. Fig. 7 shows the addition of momentum helps to speed up the training convergence and greatly decreases the training time.

5.3 Information Retrieval Task

We perform information retrieval task on the MSN news data. We use a title from the test set as a query to retrieve other titles from the same set, and like the measurement used in [13], to decide whether a retrieved title/article is relevant to the query, we simply check if they are in the same category. However, [13] only verifies the effectiveness

2. http://www.freebase.com/

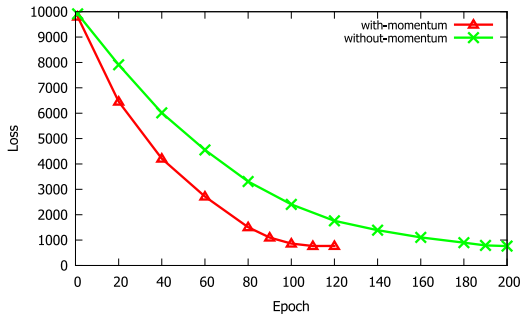

Fig. 7. The loss curves of MSN News training data. We trained the networks through supervised fine-tuning with momentum and without momentum, and momentum is set to 0.9.

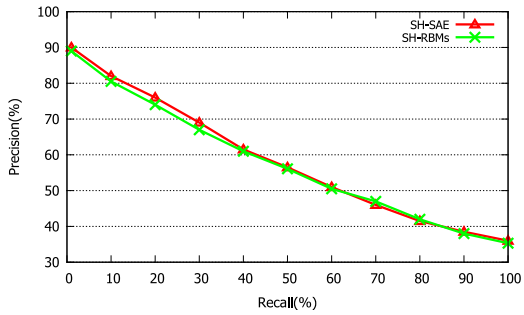

Fig. 8. Precision-recall curves obtained by semantic hashing models based on our stacked auto-encoders and RBMs.

of their semantic hashing model on long documents, our experiments are more challenging. We use the *12-point interpolated average precision* [28] method to get the precision-recall curve to evaluate the retrieval performance. The related definitions are:

$$recall = \frac{number\ of\ retrieved\ relevant\ items}{number\ of\ all\ relevant\ items},$$
$$precision = \frac{number\ of\ retrieved\ relevant\ items}{number\ of\ all\ retrieved\ items}.$$

For each test query, we retrieve the texts (short texts or long documents) ranked by their similarities with that query, then we measure the precisions at 11-points recall levels, which are 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. we then calculate the arithmetic mean of all precisions at each recall level for all test queries and finally get the 11-points precision-recall curve.

5.3.1 Comparison between our DNN and RBMs-Based Model

We first compare our designed DNN model with the model based on RBMs [13]. Both of the two semantic hashing models are trained on enriched titles data to do the IR task. The training time and precision-recall results are illustrated in Table 5 and Fig. 8 respectively.

Basically, the precision-recall results obtained by the two models are very close, while our model is faster than RBMs, which shows that i) our proposed DNN model outperforms the model based on RBMs in terms of training cost, while it enjoys a comparative learning capacity to do semantic hashing, and ii) our proposed semantic enrichment plays a critical role in understanding short text (which will be discussed later in detail). Moreover, the training method can be extended to multi-threads or distributed computation which will significantly reduce the training time.

TABLE 5
The Training Time (s/epoch) of Our Stacked Auto-Encoders (SH-SAE) and RBMs (SH-RBMs)

Model	1st layer	2nd layer	3rd layer	fine-tuning
SH-SAE	226	50	13	185
SH-RBMs	269	67	18	223

We used a single-threaded program on one machine powered by an Intel Core (TM) i5-2400 3.1-GHz with 8 GB memory, running Windows.

5.3.2 Comparison of Retrieval Methods

We denote our proposed method to understand short texts as ***EDNN***, which indicates the combination of CACT enrichment approach and DNN model. For comparison, we use two traditional and popular information retrieval methods—LSA and TF-IDF. Results of [29] show that pLSA and LDA models do not generally outperform LSA and TF-IDF, and [13] also applies the two approaches for comparison, so in this paper we still adopt them as the baseline methods. More specifically, TF-IDF and LSA directly use the original news titles to do the retrieval task. Each news title is represented as a 3,000-dimensional semantic feature vector, which only contains the original terms. To measure any two texts' similarity, we compute the cosine distance between their 3,000-dimensional semantic feature vectors as the metric. We denote the two methods as ***OTFIDF*** and ***OLSA***.

To explore the respective effect of our enrichment method (i.e., CACT) and DNN model in the retrieval task, we further implement two tests: i) we directly apply our DNN model to do semantic hashing on the original titles without enrichment, and the retrieval task is then performed on the learned 128D binary codes. This method is denoted as ***ODNN***. ii) We enrich the news titles using our proposed CACT method, and we then adopt TF-IDF and LSA to do retrieval task on these enriched titles. The two methods are denoted as ***ETFIDF*** and ***ELSA***.

Fig. 9 shows the precision-recall results obtained by our DNN model, TF-IDF and LSA methods on the retrieval of original and enriched MSN news titles. Fig. 9a presents the comparison results of using our proposed unified model (EDNN) and the OTFIDF and OLSA. Fig. 9b shows the result of using different retrieval methods (i.e., DNN, TF-IDF and LSA) when the news titles are not enriched, while Fig. 9c gives the result when the three method performed on enriched news titles, and these two tests aim to show the effectiveness of our DNN model in terms of semantic hashing. Figs. 9d, 9e, and 9f present the comparison results obtained on original and enriched news titles when we use a same retrieval method, and the three tests are able to evaluate the importance of our enrichment method in short text retrieval task.

We draw the following conclusions: First, as shown in Fig. 9a, our proposed method, which combines short

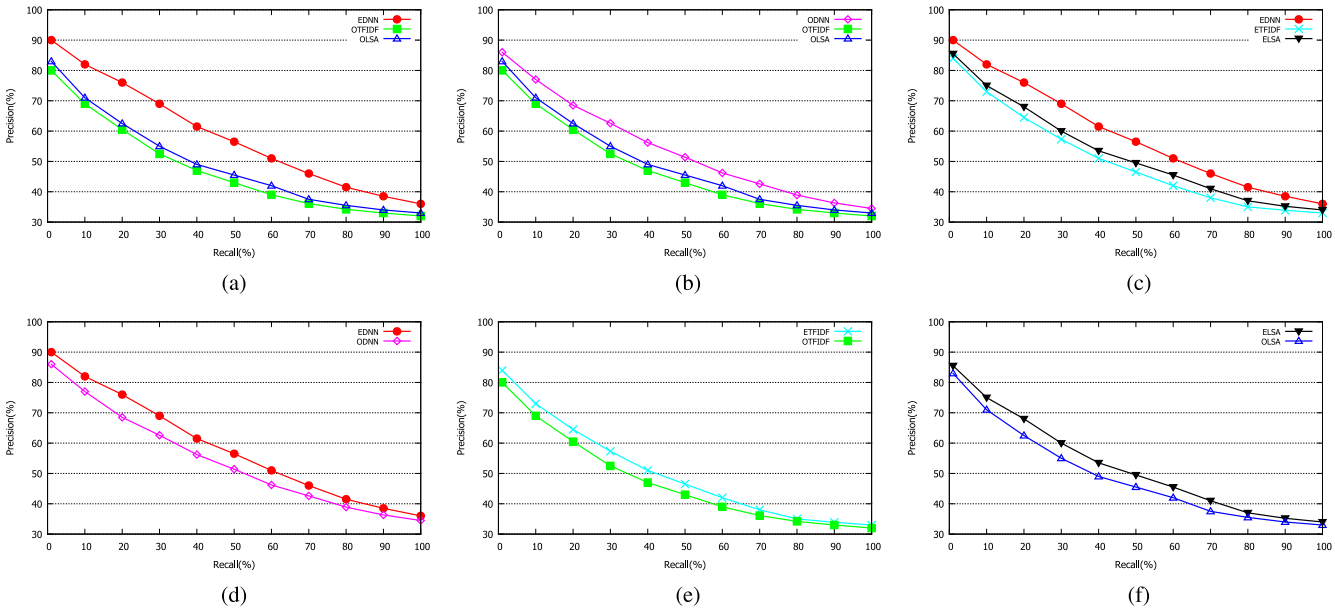

Fig. 9. Precision-recall curves obtained by our DNN, TF-IDF, and LSA on enriched and original MSN news titles.

text enrichment and semantic hashing, significantly outperforms the OTFIDF and OLSA methods. From Figs. 9b and 9c, we can see that no matter whether the titles are enriched or not, using our DNN model to map short texts into compact binary codes enables us to get better retrieval results than using TF-IDF and LSA directly on the short texts. This shows the effectiveness of our semantic hashing model in terms of information retrieval. We argue that our semantic hashing model is able to capture correlationships that beyond text surface between words and encode them into latent features, such that semantically similar short texts would have similar hashing codes. For example, "computer", "internet" and "4g" are learned to be correlated and as strong signals for certain topics, such as technology, and the short texts that contain those words would be mapped to similar binary hashing codes. While it is hard for TF-IDF and LSA methods to capture correlationships between words and encode them into features, because the traditional bag-of-words representation for short texts used in those methods has no such distance similarity property. Furthermore, from Figs. 9d, 9e, and 9f, it can be seen that for each retrieval method, the result obtained on enriched news titles is much better than that obtained on the original news titles without enrichment. This further demonstrates that our CACT enrichment method is able to introduce very useful information that can help the understanding of short texts.

5.3.3 Comparison of Enrichment Approaches

To further explore the effectiveness of using concepts and co-occurring terms in terms of enriching short texts, we compare the CACT method with some other popular enrichment approaches, which are also based on knowledge resources, which are WordNet and Wikipedia. Concretely, in WordNet-based method, we retrieve hypernyms in WordNet for each word as additional features. And in Wikipedia-based method, for a word, we find its categories on Wikipedia (through Wikipedia's Category-Link), then we use anchor texts on the links to enrich it. After enriching the news titles, we apply the DNN model to get the 128-dimensional binary codes representation and then perform the information retrieval task on those binary codes. Here, we further explore the respective contribution of concept and co-occurring terms in the aspect of semantic enrichment for short texts. We denote the approach that enriches news titles only with concepts as ***Concept-DNN***, and the approach that only uses co-occurring terms as ***Cooccur-DNN***. Similarly, the unified model is denoted as ***EDNN***. ***WordNet-DNN*** represents the WordNet-based method, and ***Wiki-DNN*** represents the Wikipedia-based method.

Fig. 10 shows the experiment results. Compared with the WordNet-DNN and Wiki-DNN methods, our EDNN approach gets higher precisions as much as 10 percent at most recall points, and even the Concept-DNN and Cooccur-DNN methods outperform the methods based on WordNet and Wikipedia. While the Concept-DNN performs a little better than Cooccur-DNN. This experiment shows the importance and helpfulness of concept and co-occurring term for short texts, especially the concepts, which greatly facilitate the understanding of short texts. This is because the concepts of a term are able to explicitly express its true meaning under different contexts.

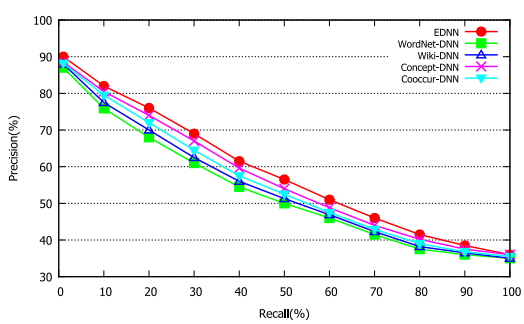

Fig. 10. The results of using different enrichment methods.

TABLE 6
The Retrieval Results of Three Queries Using EDNN and OLSA Methods

Query title	Category	Method	Precisions of top100 retrievals	Enriched semantic features
Jon Bernthal Tapped for Martin Scorsese's "The Wolf of Wall Stree"	Entertainment	EDNN	0.96	(concepts:) artist, movie/film, hollywood legend, feature files, etc; (co-terms:) directed, directors, film, cameron, interview, movies, etc
		OLSA	0.65	
Mickelson inducted into World Golf Hall of Fame	Sport	EDNN	0.96	(concepts:) player, world-class player, sport, competition, outdoor sport, etc; (co-terms:) rank, champ, wins, football, beating, champion, sport, etc
		OLSA	0.71	
Insight: As chip plants get pricey, U.S. risks losing edge	Business	EDNN	0.95	(concepts:) company, plant, firm; million, profile, makers, etc; (co-terms:) manufacturing, state, benefit, business, financial, etc
		OLSA	0.58	

5.3.4 Case Study

We finally analyse some query examples on which EDNN approach works very well while OLSA method performs badly. Table 6 shows the retrieval results of three query news titles. The big difference between the two results reflects the challenges in understanding short texts. Short texts, particularly news titles, usually contain kinds of unpopular but informative terms. Consider the following terms in our query titles, "Jon Bernthal" and "Mickelson" are names of celebrities, "The Wolf of Wall Street" , "World Golf Hall of Fame" and "Insight" all have specific meanings. These special semantics cannot be captured from the surface of short text, let alone some general representations such as bag-of-words. While in our mechanism, we enrich short texts with both explicit concepts and co-occurring terms, these external knowledge could greatly help understanding of short text. See the *added key external knowledge* column in Table 6, most terms in original titles are mapped to their representative concepts. For example, "Jon Bernthal" is mapped to "artist", "The Wolf of Wall Street" is mapped to "film" , and "Mickelson" is mapped to "player" and "world-class player"; furthermore, the introduced co-occurring terms (co-terms) are semantically related to the original titles. For instance, the terms "directed", "director" and "cameron" are closely related to "Jon Bernthal Tapped for Martin Scorsese's 'The Wolf of Wall Street' ".

5.4 Short Text Classification Task

We use support vector machines (SVMs) [30] as the basic model to do classification on the short sentences of Wikipedia. SVMs are very popular supervised learning models that analyze data and recognize patterns. Meyer et al. [31] show that compared with most existing methods, SVMs get mostly good performances on classification and regression tasks. Similarly, in this experiment, we represent each sentence by a semantic feature vector (3,000-dimensional) or the learned semantic hashing code (128-dimensional), and the sentence can be enriched using different methods, including our CACT method, the WordNet-based and method and the Wikipedia-based method. We then make use of a popular SVM implementation, libSVM,[3] to classify the short sentences based on their representations. Table 7 shows the experiment details and results.

For the classification results obtained on the 3,000-dimensional semantic feature vectors, we get the lowest accuracy on original sentences (without enrichment), which is only 29 percent (the result obtained by OS-SVM); while after enriching the sentences using our CACT, WordNet-based and Wikipedia-based methods, the classification accuracies achieved on these enriched sentences are all improved a lot. The CACT-SVM approach gets 47.52 percent accuracy, which is almost 10 percent higher than that achieved by WordNet-SVM and Wiki-SVM. However, if we do the classification on the learned 128-dimensional binary codes of the short sentences, the accuracy can be further improved by nearly 4 percent for each approach, and our unified model—EDNN-SVM still gets the best result. This experiment also demonstrates the respective effectiveness of our proposed enrichment mechanism for short text and DNN model for semantic hashing, and the unified model enables us to better understand the meaning of short texts.

For our classification task, there are up to 110 categories need to be considered. The categories are very diverse and some of them are closely related. For example, the categories of *film.actor*, *award_winner* and *film.music_contributor* are very similar. It is difficult even for humans to classify them

TABLE 7
The Classification Results on Wikipedia Short Sentences

Method	Enriched Method	Input Representation	Accuracy
OS-SVM	Not enriched	3,000D semantic feature vectors	29.00%
CACT-SVM	**CACT**	**3,000D semantic feature vectors**	**47.52%**
WordNet-SVM	WordNet-based	3,000D semantic feature vectors	36.15%
Wiki-SVM	Wikipedia-based	3,000D semantic feature vectors	39.06%
EDNN-SVM	**CACT**	**learned 128D binary code vectors**	**51.35%**
WordNet-SH-SVM	WordNet-based	learned 128D binary code vectors	40.21%
Wiki-SH-SVM	Wikipedia-based	learned 128D binary code vectors	42.76%

3. http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

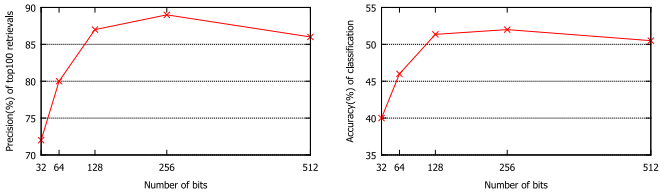

Fig. 11. Results of retrieval and classification based on different dimensional hashing codes.

with background knowledge. For these original sentences, their semantic feature vectors are too sparse to contain sufficient information to identify their categories. For example, in the result obtained by OS-SVM method, most sentences belonged to *film.music_contributor* or *film.actor* or *award_winner* are classified into the category of *people.person*, because their representations contain lots of too general semantic features, such as "american", "know" and "live". However, after enrichment through our method, many more characteristic semantic features such as "singer", "director" and "author" are included, such features make the semantics of these sentences prominent, and further help the classification on these short sentences.

5.5 Analysis of Different Dimensional Semantic Codes

We vary the number of hashing bits (i.e., the size of output layer) learned by our DNN model, and use such different dimensional semantic codes to do the two tasks. Fig. 11a shows the precision of top100 retrievals using the EDNN model, and Fig. 11b shows the accuracy of classification using the EDNN-SVM model. From these comparison results, it can be seen that the retrieval precision and classification accuracy increases quickly when the number of hashing bits increase from 32 to 128, and the results get a little better when the number of bits increase from 128 to 256. However, if the semantic codes become longer than 256, the results turn worse, this is because when using longer hashing bits, the Hamming space becomes increasingly spares such that very few data points fall within the Hamming ball of a radius, and the semantic similarity between related texts becomes weak. So, in our experiments, we choose to use 128-dimensional hashing bits considering its effectiveness and efficiency in computing Hamming distances.

6 Related Work

Many approaches have been proposed to facilitate short text understanding by enriching the short text representations. These approaches can generally be grouped into two directions. One direction is to extend the contextual information, which aims to include more statistical information for text analysis. Intuitively, we can extract the contexts that accompany with short text from a large corpus. For example, if we treat a short text as a query, we may enrich it with search results (e.g., webpage titles and snippets) returned by a search engine. Using titles and snippets to extend web queries, [3] achieves much higher classification accuracy on the query classification task. Through similar method, the accuracy of measuring short text similarity is significant improvement in [1], [2]. Another direction is to enrich short text with explicit semantic information derived from external resources, such as WordNet, Wikipedia and the Open Directory Project. These external resources consist of rich predefined taxonomies and human-labeled information. [5] presents a novel framework to improve the clustering of short texts by incorporating the knowledge from both Wikipedia and WordNet. With the titles of Wikipedia articles as additional features, [6] achieves improvement in the accuracy of short texts clustering. Other researches also show that using Wikipedia based concepts and category information [32], [33] helps text classification and information retrieval. Traditional text retrieval methods, such as TF-IDF [16], [17], LSA [15], LDA [22] and pLSA [34], have made significant achievements in most text-related applications. Recently, Salakhutdinov and Hinton [13] propose a new information retrieval mechanism called semantic hashing. The model is stacked by RBMs and learns to map a document semantic to a compact binary code. Compared with traditional methods, such as TF-IDF and LSA, their semantic hashing model achieves comparable retrieval performance.

7 Conclusions

In this paper, we propose a novel approach for understanding short texts. First, we introduce a mechanism to enrich short texts with concepts and co-occurring terms that are extracted from a probabilistic semantic network, known as Probase. After that, each short text is represented as a 3,000-dimensional semantic feature vector. We then design a more efficient deep learning model, which is stacked by three auto-encoders with specific and effective learning functions, to do semantic hashing on these semantic feature vectors for short texts. A two-stage semi-supervised training strategy is proposed to optimize the model such that it can capture the correlationships and abstract features from short texts. When training is done, the output is thresholded to be a 128-dimensional binary code, which is regarded as a semantic hashing code for that input text. We carry out comprehensive experiments on short text centered tasks including information retrieval and classification. The significant improvements on both tasks show that our enrichment mechanism could effectively enrich short text representations and the proposed auto-encoder based deep learning model is able to encode complex features from input into the compact binary codes.

Acknowledgments

This work is supported by NSFC61232006, ARC DP120104168, ARC DP140103578, and ARC DP150102728. The authors thank the reviewers for their detailed comments.

References

[1] M. Sahami and T. D. Heilman, "A web-based kernel function for measuring the similarity of short text snippets," in *Proc. 15th Int. Conf. World Wide Web*, 2006, pp. 377–386.
[2] W. tau Yih and C. Meek, "Improving similarity measures for short segments of text," in *Proc. 22nd Nat. Conf. Artif. Intell.*, 2007, pp. 1489–1494.

[3] D. Shen, R. Pan, J.-T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang, "Query enrichment for web-query classification," *ACM Trans. Inf. Syst.*, vol. 24, no. 3, pp. 320–352, 2006.
[4] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Cambridge, MA, USA: MIT Press, 1998.
[5] X. Hu, N. Sun, C. Zhang, and T.-S. Chua, "Exploiting internal and external semantics for the clustering of short texts using world knowledge," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, 2009, pp. 919–928.
[6] S. Banerjee, K. Ramanathan, and A. Gupta, "Clustering short texts using wikipedia," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2007, pp. 787–788.
[7] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using Wikipedia-based explicit semantic analysis," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, 2007, pp. 1606–1611.
[8] E. Gabrilovich and S. Markovitch, "Feature generation for text categorization using world knowledge," in *Proc. 19th Int. Joint Conf. Artif. Intell.*, 2005, pp. 1048–1053.
[9] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proc. Int. Conf. Manage. Data*, 2012, pp. 481–492.
[10] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen, "Short text conceptualization using a probabilistic knowledge base," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 2330–2336.
[11] D. Kim, H. Wang, and A. H. Oh, "Context-dependent conceptualization," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, 2013, pp. 2654–2661.
[12] B. Stein, "Principles of hash-based text retrieval," in *Proc. ACM 30th Annu. Int. Conf. Res. Develop. Inf. Retrieval*, 2007, pp. 527–534.
[13] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.
[14] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.
[15] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.
[16] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.
[17] G. Salton. (1991, Aug.). Developments in automatic text retrieval. Science [Online]. 253(5023), pp. 974–980. Available: http://www.jstor.org/stable/2878789
[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
[19] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.
[20] J. A. Anderson and J. Davis, *An Introduction to Neural Networks*. Cambridge, MA, USA: MIT Press, 1995.
[21] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
[22] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.
[23] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.
[24] P. Li, H. Wang, K. Q. Zhu, Z. Wang, and X. Wu, "Computing term similarity by large probabilistic is a knowledge," in *Proc. 22nd ACM Int. Conf. Conf. Inf. Knowl. Manage.*, 2013, pp. 1401–1410.
[25] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, 2006.
[26] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *Proc. Int. Workshop Artif. Neural Net.: From Natural Artif. Neural Comput.*, 1995, pp. 195–201.
[27] L. Bottou, "Stochastic gradient learning in neural networks," in *Proc. Neuro-Nimes. EC2*, 1991.
[28] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ., 2008.
[29] P. V. Gehler, A. Holub, and M. Welling, "The rate adapting poisson model for information retrieval and object recognition," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 337–344.
[30] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[31] D. Meyer, F. Leisch, and K. Hornik, "The support vector machine under test," *Neurocomputing*, vol. 55, nos. 1/2, pp. 169–186, 2003.
[32] X. H. Phan, M. L. Nguyen, and S. Horiguchi, "Learning to classify short and sparse text & web with hidden topics from Large-scale data collections," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 91–100.
[33] O. Egozi, E. Gabrilovich, and S. Markovitch, "Concept-based feature generation and selection for information retrieval," in *Proc. 23rd Nat. Conf. Artif. Intell.*, 2008, pp. 1132–1137.
[34] T. Hofmann, "Probabilistic latent semantic analysis," in *Proc. 15th Conf. Uncertainty Artif. Intell.*, 1999, pp. 289–296.

Zheng Yu received the BE degree in software engineering from East China Normal University in 2011. He is currently working toward the PhD degree in the Software Engineering Institute, East China Normal University, China. He has been an intern on the database team, Microsoft Research, Asia, from November 2012 to November 2013. His current research interests include natural language processing, deep learning, and machine learning.

Haixun Wang received the BS and MS degrees in computer science from Shanghai Jiao Tong University in 1994 and 1996, respectively, and the PhD degree in computer science from the University of California, Los Angeles, in June, 2000. He is a research scientist at Google Research. Before joining Google, he was a senior researcher at Microsoft Research, Asia, where he led the database team. From 2000 to 2009, he was with IBM Research (Watson). He has published more than 190 research papers in referred international journals and conference proceedings. He is serving as an associate editor of *Distributed and Parallel Databases (DAPD)*, the *IEEE Transactions of Knowledge and Data Engineering (TKDE)*, *Knowledge and Information System (KAIS)*, and the *Journal of Computer Science and Technology (JCST)*. His current interests include natural language processing, semantic network, knowledge base, machine learning, and artificial intelligence.

Xuemin Lin received the BSc degree in applied math from Fudan University in 1984. During 1984-1988, he studied for the PhD degree in applied math at Fudan University. He received the PhD degree in computer science from the University of Queensland in 1992. He is a professor in the School of Computer Science and Engineering, University of New South Wales (UNSW). He has been the head in the Database Research Group UNSW since 2002. He is a concurrent professor in the School of Software, East China Normal University. Before joining UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. He is currently an associate editor of the *ACM Transactions on Database Systems*. His current research interests lie in data streams, approximate query processing, spatial data analysis, and graph visualization. He is a senior member of the IEEE.

Min Wang received the BS and MS degrees, both in computer science, from Tsinghua University, Beijing, China, and the PhD degree in computer science from Duke University. She is a research scientist at Google Research. Her previous appointments include HP distinguished technologist and the director in HP Labs China in Beijing and a research staff member and manager in the Unified Data Analytics Department, IBM's Thomas J. Watson Research Center in Hawthorne, New York. Her research interests are in database systems and information management. In 2009, she received the ACM SIGMOD 2009 Test of Time Award for her 1999 SIGMOD paper, "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets."