Syntax-based Deep Matching of Short Texts*

Mingxuan Wang[1] **Zhengdong Lu**[2] **Hang Li**[2] **Qun Liu**[3,1]
[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]Noah's Ark Lab, Huawei Technologies
[3]Centre for Next Generation Localisation, Dublin City University

Abstract

Many tasks in natural language processing, ranging from machine translation to question answering, can be reduced to the problem of matching two sentences or more generally two short texts. We propose a new approach to the problem, called Deep Match Tree (DEEPMATCH$_{tree}$), under a general setting. The approach consists of two components, 1) a mining algorithm to discover patterns for matching two short-texts, defined in the product space of dependency trees, and 2) a deep neural network for matching short texts using the mined patterns, as well as a learning algorithm to build the network having a sparse structure. We test our algorithm on the problem of matching a tweet and a response in social media, a hard matching problem proposed in [Wang *et al.*, 2013], and show that DEEPMATCH$_{tree}$ can outperform a number of competitor models including one without using dependency trees and one based on word-embedding, all with large margins.

1 Introduction

Matching is of central importance to natural language processing. In fact, many problems in natural language processing can be formalized as matching between two short-texts, with different matching relations in different applications. For example, in paraphrase identification the relation is synonymy, and in information retrieval it is relevance. In the meantime matching is also a challenging problem, since it requires modeling of the two short-texts as well as their relation. In machine translation, for example, the model needs to determine whether a sentence in the source language has the same meaning as a sentence in the target language. In dialogue, the model needs to judge whether a message is an appropriate response to a given utterance.

Deep neural network can model non-linear and hierarchical relations [Bengio, 2009], and thus is well suited for short-text matching in natural language processing. The very limited work in that thread, makes use of word embedding as the building blocks of matching model. Although embedding-based methods have been proven effective on tasks like question answering [Lu and Li, 2013], paraphrase identification [Socher *et al.*, 2011], and even short text conversation [Lu and Li, 2013; Hu *et al.*, 2014], they are not enough good at handling the subtlety of general short-text matching. Short-texts often represent rich content, their relations are also complicated, and more sophisticated structures are required for comparing the two short-texts. For example, when judging the appropriateness of response "`You should rest more.`" to utterance "`I have to work during the weekend!`", we have to consider the semantic correspondence between "work over the weekend" and "need to rest more", which is hard to be captured by an embedding-based model.

We study the problem of short-text matching in a general setting. Our method, named *Deep Match Tree* (DEEPMATCH$_{tree}$), consists of two sequentially connected components: 1) a mining algorithm to discover rich yet subtle patterns, defined in the product space of dependency trees, from a large corpus of paired short-texts, and 2) a learning algorithm to construct a deep neural network (DNN) for making a matching decision on the two short-texts, on the basis of the mined patterns. The DNN model is specifically trained based on contrastive sampling of negative examples.

Without loss of generality, we focus on the task of matching a response to a given tweet on Weibo, a popular Chinese microblog service, for which a large amount of data is available. This is a hard problem, requiring consideration of complicated correspondence between the structures of two texts. Our experimental results show that DEEPMATCH$_{tree}$ is superior to existing methods on the problem.

Our main contributions are: 1) proposal of an algorithm for mining dependency tree matching patterns on large scale, 2) proposal of an algorithm for learning a deep matching model for using mined matching patterns, and 3) empirical validation of the efficacy and efficiency of the proposed method using large scale real datasets.

2 Direct Product of Graphs (PoG)

We first propose representing the matching of a pair of sentences (in general short-texts) with the direct product between

*this work is done when the first author worked as intern at Noah's Ark Lab, Huawei Technologies.

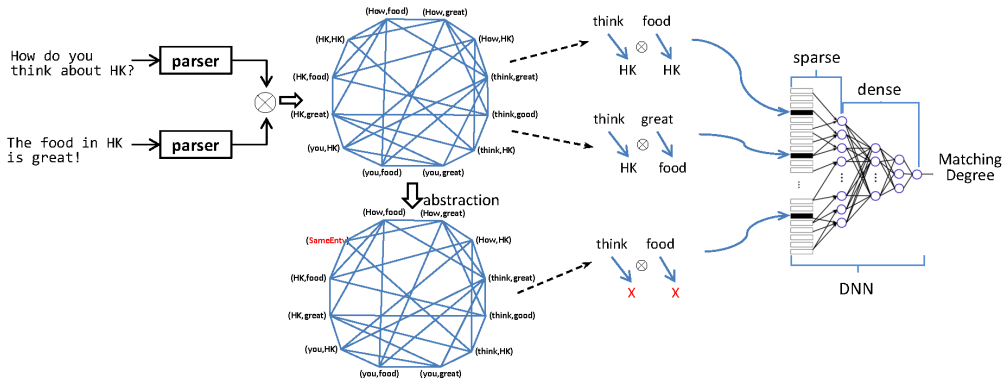

Figure 1: The overall architecture for DEEPMATCH$_{tree}$.

the dependency trees of them, and then propose treating subgraphs of this product graph as matching patterns.

2.1 Dependency Tree

We represent a sentence with its dependency tree. We choose to do so because a dependency tree tends to expose the "skeleton" of the sentence, revealing both short-distance and long-distance grammatical relations between words [Filippova and Strube, 2008]. For example the dependency tree in Fig.2 contains structures like {Li Na←win→championship} represented as a sub-tree,where the words (boldface) are not necessarily adjacent to each other in the sentence.

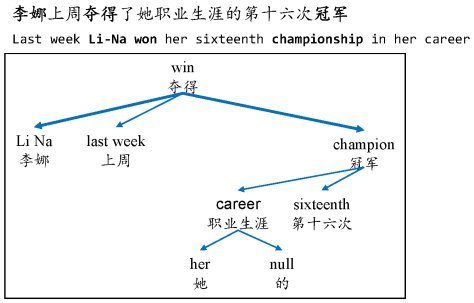

Figure 2: Example of dependency tree, where the main structure of the sentence is represented as the sub-tree in thick edges. The tweet is in Chinese (literal English translation).

2.2 Direct Product of Graphs

The direct product of graphs (PoG) $\mathcal{G}_X = \{V_X, E_X\}$ and $\mathcal{G}_Y = \{V_Y, E_Y\}$, is a graph $\mathcal{G}_{X\times Y}$ [Vishwanathan *et al.*, 2010], with vertices $V_{X\times Y}$ and edges $E_{X\times Y}$

$$\begin{aligned} V_{X\times Y} &= \{(v_i^X, v_{i'}^Y),\ v_i^X \in V_X,\ v_{i'}^Y \in V_Y\} \\ E_{X\times Y} &= \{((v_i^X, v_{i'}^Y)(v_j^X, v_{j'}^Y)),\ (v_i^X, v_j^X) \\ &\quad \in E_X \wedge (v_{i'}^Y, v_{j'}^Y) \in E_Y\} \end{aligned}$$

Given two sentences S_X and S_Y, their interaction relation is represented by the direct product of their dependency trees

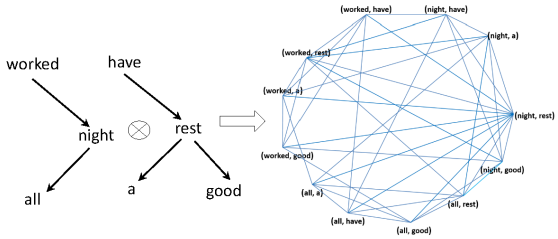

Figure 3: The direct product of two dependency trees.

$\mathcal{G}_{X\times Y}$. For example, two sentences `Worked all night` and `Have a good rest` (with their dependency trees are given by the left panel of Fig.3), and the direct product of the trees is given by the right panel of Fig.3. Note that $\mathcal{G}_{X\times Y}$ is in general a graph even though $\mathcal{G}_X$ and $\mathcal{G}_Y$ are trees.

$\mathcal{G}_{X\times Y}$ directly describes the interaction relation between sentences S_X and S_Y, hosting a rather rich set of structures, both lexical and syntactic, that contribute to the overall matching between the two sentences. Next we make further abstraction of the representation.

2.3 Abstraction

We consider two types of abstraction for vertices in $\mathcal{G}_{X\times Y}$

- **Same Entity:** We replace the vertex (ne_i, ne_i) in $\mathcal{G}_{X\times Y}$ representing the same entity with a general vertex SAMEENTITY. For example for the sentences (`How is the wether in Paris?`) (`Haven't seen such a sunny day in Paris for a while!`), the vertex (`Paris`, `Paris`) after the abstraction will be treated as the same vertex as (`Boston`, `Boston`) after the same type of abstraction. Graph with this type of abstraction is named $\tilde{\mathcal{G}}^e_{X\times Y}$.
- **Similar Word:** We conduct clustering of words based on their word2vectors [Mikolov *et al.*, 2013] using the K-means algorithm. For a vertex (w_i, w_j) in the product graph, if w_i and w_j belong to the same word cluster $\mathcal{C}_k$, then the vertex will be replaced with a new vertex SIMWORD$_k$. Graph with this type of abstraction is named $\tilde{\mathcal{G}}^s_{X\times Y}$.

Both types of abstraction will enhance the generalization ability of matching pattern mining described next.

2.4 Sub-graphs of PoG as Matching Patterns

With a little abuse of notation, we use $\bar{\mathcal{G}}_{X\times Y} = \{\mathcal{G}_{X\times Y}, \tilde{\mathcal{G}}^e_{X\times Y}, \tilde{\mathcal{G}}^s_{X\times Y}\}$ to denote the PoG for sentence pair (S_X, S_Y) as well as its variants after two types of abstraction. For a sentence pair (S_X, S_Y), any sub-graph in the corresponding $\bar{\mathcal{G}}_{X\times Y}$ describes part of the interaction between the two sentences and therefore can contribute to the matching between the two. For instance, `(weather, sunny)` $\longleftrightarrow$ SAMEENTITY is a sub-graph describing the matching between two sentences in a conversation about weather (see the example two paragraph ago). In general, $\bar{\mathcal{G}}_{X\times Y}$ contains all the meaningful matching patterns for the task.

3 Mining of Matching Patterns

It is the responsibility of a mining algorithm to discover those sub-graphs of $\{\bar{\mathcal{G}}_{X\times Y}\}$ that can work as matching patterns to discriminate matched sentence pairs from mismatched ones, measured in terms of discriminative ability (cf.,[Fan *et al.*, 2008]). Discriminative roughly means it gives some evidence on matching, i.e., it appears in matched pairs more frequently than unmatched pairs. An efficient mining algorithm is vital to the success of this method, when the number of instances is of the order of 10^6 and the number of mined patterns is of the order of 10^7.

3.1 Speeding-up the Mining Process

Fortunately, we can leverage the following fact with respect to sub-graphs in the PoG $\mathcal{G}_{X\times Y}$ (without abstraction).

Proposition 3.1. *Any connected sub-graph $\mathcal{G}^s_{X\times Y}$ in $\mathcal{G}_{X\times Y}$ can uniquely determine a minimal sub-tree in $\mathcal{G}_X$ and a minimal sub-tree $\mathcal{G}_Y$, whose direct product can cover the $\mathcal{G}^s_{X\times Y}$.*

As it implies, the mining of sub-graphs in PoG of trees can be reduced to jointly selecting the sub-trees on two sides. This can not only greatly speed up the mining process, but also avoid finding patterns with duplicate functionality for matching. In the remainder of the paper, we will use $\texttt{sub-tree}_1 \otimes \texttt{sub-tree}_2$ to denote a tree-pair (separated by $\otimes$) mined from the PoG. This however does not apply to the more general case of $\bar{\mathcal{G}}_{X\times Y}$ when some vertices are replaced with non-factorable variants, like SIMWORD$_{123}$, for which we have to introduce some new tricks.

3.2 Mining without Abstraction

The algorithm for mining without abstraction, sketched in Algorithm 1, is to recursively grow the mined sub-graphs while maintaining its discriminative ability. It starts with the simplest pattern (1,1), standing for one-word tree on both X side (tweet) and Y side (response), and grows the mined trees recursively. In each growing step (`LeftExtend()` and `RightExtend()`), the size of sub-trees is increased by one on either X side or the Y side, followed by a filtering step to remove the found pairs with discriminative ability less than a threshold. The growing step is efficient, since we can limit the search for patterns of $(m, n+1)$ from the candidates formed by merging patterns of (m, n). In practice the time for looking-up each sub-tree pair is almost constant with the help of Hashmap. The following table gives some examples of the matching patterns discovered by Algorithm 1.

Algorithm 1: Discriminative Mining of Parse Trees for Parallel Texts

Input: $\mathcal{T}$: tree pairs for original (tweet, response), MaxSize
Output: Set of mined features $\mathcal{F}$;
Initinalize $\mathcal{F} \leftarrow \emptyset$; $\mathcal{M} \leftarrow \emptyset$; $Q \leftarrow [\,]$;
`ENQUEUE` (Q, (1,1));
foreach *node set x, y in $\mathcal{X}, \mathcal{Y}$* **do**
  Append each element of $x \otimes y$ to $\mathcal{F}_{1,1}$;
  Append $x \otimes y$ to $\mathcal{M}_{1,1}$;
$\mathcal{F}_{1,1} \leftarrow$ `DiscriminativeFilter(`$\mathcal{F}_{1,1}$`)`;
while $Q \neq [\,]$ **do**
  $m, n \leftarrow$ `DEQUEUE` (Q);
  if $m+1 <$ *MaxSize* $\wedge$ $(m+1, n)$ *has not been processed* **then**
    $[\mathcal{M}_{m+1,n}, \mathcal{F}_{m+1,n}] \leftarrow$ `LeftExtend(`$\mathcal{M}_{m,n}$`)`;
    `ENQUEUE` (Q, $(m+1, n)$);
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_{m+1,n}$;
  if $n+1 <$ *MaxSize* $\wedge$ $(m, n+1)$ *has not been processed* **then**
    $[\mathcal{M}_{m,n+1}, \mathcal{F}_{m,n+1}] \leftarrow$ `RightExtend(`$\mathcal{M}_{m,n}$`)`;
    `ENQUEUE` (Q, $(m, n+1)$);
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_{m,n+1}$;

Patterns without abstraction
`exam` $\otimes$ `score`
`Information theory` $\otimes$ `Shannon`
`thank`$\rightarrow$`present` $\otimes$ `happy`$\rightarrow$`birthday`
`win`$\rightarrow$`game` $\otimes$ `trying`$\rightarrow$`keep`
`out-of-control`$\rightarrow$`prices` $\otimes$ `regulation`
`work`$\rightarrow$`weekend` $\otimes$ `rest`

3.3 Mining with Abstraction

The algorithm for mining with abstraction is a variant of Algorithm 1. Taking the SameEntity abstraction as example, we first replace each named entity, e.g., `Li Na` (found via a named entity resolution program) with a vertex having the same ID (say, NamedEntity239). The growing step is the same as in Algorithm 1, except that when counting the support (number of instances containing it) of a pattern, it replaces the same entity appearing on both sides with a wildcard, and therefore groups many patterns as the same one. For example, the instances for the following two patterns

`Li Na` $\leftarrow$ `win` $\otimes$ `Li Na` $\leftarrow$ `congratulations`
`Nadal` $\leftarrow$ `win` $\otimes$ `Nadal` $\leftarrow$ `congratulations`

will be counted together for the pattern

$x \leftarrow$ `win` $\otimes$ $x \leftarrow$ `congratulations`

where x stands for the wildcard. The mining with SimilarWord abstraction is similar, only slightly more complicated on deciding when two words can be merged.

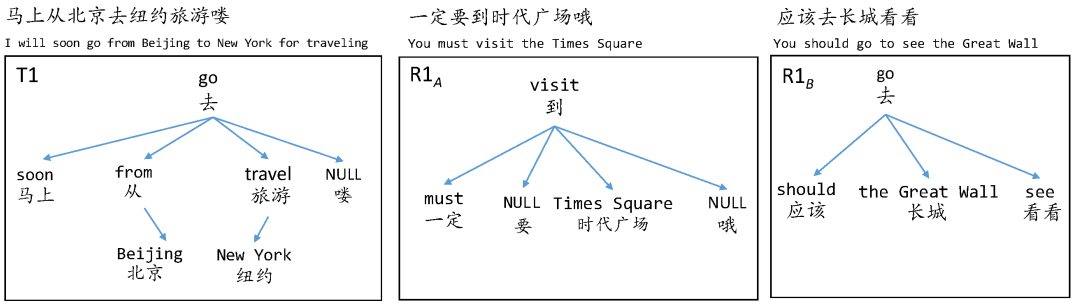

Figure 4: Illustration for dependency trees for tree short sentences.

The following table gives some examples of the matching patterns discovered by our algorithm on the graph with abstraction. Here (x, x') stand for wildcards considered similar enough by the algorithm.

Patterns with abstraction
hope→win→x ⊗ support→x
how about→x ⊗ like→x
gift→x ⊗ happy→x
recommend→x ⊗ x→nice
pretty good→x ⊗ fine→also→x'

3.4 Advantage of Tree Pattern Mining

It is important to note that dependency tree matching patterns can provide better correspondence between the two sentences than word co-occurrences on two sides (an idea first explored in [Lu and Li, 2013]). To illustrate the superiority of using dependency tree matching patterns, suppose that for the tweet T1 in Fig.4 we want to pick a more appropriate one from the two responses ($R1_A$ and $R1_B$). The word-based model tends to assign a high matching score to pair (T1, $R1_B$), due to the pattern {Beijing, travel} ⊗ {Great Wall}, which is however spurious since T1 is about traveling to New York while the word Beijing is a distractor. On the other hand, the tree-based model relies more on patterns like follows

$$\text{travel} \rightarrow \text{New-York} \otimes \text{Times Square}$$
$$\text{travel} \rightarrow \text{Beijing} \otimes \text{Great-Wall}$$

which discriminates between word co-occurrence (e.g., {Beijing, travel}) and dependency-tree pattern (e.g., travel→New York), and gives a higher score to (T1, $R1_A$).

The mining algorithm allows us to find patterns representing deep and long-distance relationship within two short-texts to be matched. The *deep features* therefore provide sophisticated matching structures between two texts. In contrast, the *shallow features* can only give word-level correspondences between words in two texts. The difference is analogous to syntax-based translation model and word-based translation model [Koehn *et al.*, 2003].

4 The Deep Matching Model

The dependency tree matching patterns (or deep features) are then incorporated into a deep neural network for determining the matching degree of a pair of short-texts.

4.1 Model Description

The diagram of our deep matching model is given in Fig.1. When a pair of short-texts is given, we first obtain their dependency trees, form the direct product of them, and then perform abstraction on them (if suitable). After that, we look up the table of dependency tree matching patterns and convert the input text-pair into a binary vector, where an element is one if the corresponding pattern can apply to the input text-pair, otherwise it is zero. The binary vector, which is of $10M$-dimension and is sparse with typically 10∼50 ones in our experiments, is then fed into the deep neural network for the final match decision.

4.2 Learning

The learning of the deep neural network consists of 1) learning of the architecture, and 2) tuning of the parameters.

Architecture Learning

Since there are $10M$ raw features, the number of parameters will be too large if we have the input layer fully connected to the first hidden layer with a reasonable size (say, 1,000 nodes). It is therefore necessary to specify sensible sparse patterns to ensure that the information in the raw features can be well abstracted in the first hidden layer.

It is believed that neural networks are more suited for dense and continuous input, and there is little work on building an appropriate architecture for sparse and discrete input with a demanding size. In this work, we take a simple procedure to ensure each input node is connected to approximately K (referred to as NodeDensity later in the paper) hidden nodes, and the average activations of the hidden nodes (measured as the average times of them connected to hit features) are approximately the same. The underlying belief is that we can preserve as much information as possible when going from the sparse hit patterns to the dense 1,000-D representation.

The Selection of Overall Architecture The overall architecture of the neural network is illustrated in Fig.1. As it shows, we have 1,000 units in the first hidden layer (sigmoid active function), 400 in the second hidden layer, 30 in the third hidden layer, and one in the output layer. Empirical results show that this architecture performs slightly better than a 3-layer one with approximately same number of parameters,

while more hidden layers (say, 5) do not bring any significant further improvement.

4.3 Parameter Learning

We employ a discriminative training strategy with a large margin objective. Suppose that we are given the following triples $(\mathbf{x}, \mathbf{y}^+, \mathbf{y}^-)$ from the oracle, with $\mathbf{x}$ ($\in \mathcal{X}$) matched with $\mathbf{y}^+$ better than with $\mathbf{y}^-$ (both $\in \mathcal{Y}$). We have the following pairwise loss as objective:

$$\mathcal{L}(\mathcal{W}, \mathcal{D}_{trn}) = \sum_{(\mathbf{x}_i, \mathbf{y}_i^+, \mathbf{y}_i^-) \in \mathcal{D}_{trn}} e_{\mathcal{W}}(\mathbf{x}_i, \mathbf{y}_i^+, \mathbf{y}_i^-) + R(\mathcal{W}),$$

where $R(\mathcal{W})$ is the regularization term, and $e_{\mathcal{W}}(\mathbf{x}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$ is the error for triple $(\mathbf{x}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$, given by the following large margin form:

$$e_{\mathcal{W}}(\mathbf{x}_i, \mathbf{y}_i^+, \mathbf{y}_i^-) = \max(0, m + \mathsf{s}(\mathbf{x}_i, \mathbf{y}_i^-) - \mathsf{s}(\mathbf{x}_i, \mathbf{y}_i^+)),$$

with m ($0 < m$) controlling the margin in training. In the experiments, we use $m = 1$, but we find that the results are rather stable with m in a fairly large range.

For training, we use the generic back-propagation algorithm adapted for the sparse patterns in the first layer. More specifically, when updating the weights in the first layer, we only update the weights associated with the active nodes in the input layer, which faithfully respects the law of back-prop but makes the learning efficiently enough even on a training set with millions of instances. It is easy to see that the number of parameters (at least 4×10^7) is greater than the number of positive instances, and thus some kind of regularization is needed. Here we consider employing both dropout [Hinton *et al.*, 2012] and early stopping [Caruana *et al.*, 2001], which turns out to be important for the success of the model, especially when the number of parameters is over 10^8.

5 Experiments

We report our empirical study of DEEPMATCH$_{tree}$ and compare it to competitors, with a brief analysis and case studies.

5.1 Datasets and Evaluation Metric

The experiments are on two Weibo datasets in two settings.

Original-vs-Random: The first dataset, denoted as DataOrignal, consists of 4.8 million (tweet, response) pairs. For each positive pair (original pair), we randomly select ten responses as negative examples (contrastive sampling of negative examples), rendering 45 million triples. Our evaluation shows that for a given tweet there is $<1\%$ chance that a randomly selected response out of 10 is suitable.

We use 485,282 original (tweet, response) pairs not used in the training for testing. For each pair, we get nine random responses, and testing the performance of each matching model to pick the correct response.

Retrieval-based Conversation: The second dataset, denoted as DataLabeled, consists of 422 tweets and around 30 labeled responses for each tweet [1], as introduced in [Wang *et al.*, 2013] for retrieval-based conversation.

On DataLabeled, we test how different matching models enhance the performance of the retrieval-based conversation model [Wang *et al.*, 2013] on finding a suitable response for a given tweet. It is rather hard, since the negative responses are topically related to the tweet. We use the same retrieval strategy in [Wang *et al.*, 2013], while individually adding the scores of the matching models as a new feature of the ranking function to rank retrieved responses (20$\sim$30 for each tweet).

In both experiments we use precision at one (P@1) [Li, 2011] to measure the accuracy of matching. Basically, for each given tweet T, we calculate the matching scores between T and all candidate responses, and select the one with the highest score. The ranking gets one point iff the selected one is the original (on the Original-vs-Random dataset) or labeled as "good" (on the Retrieval-based Conversation dataset). P@1 measures the chance of getting the selection right averaged over all the tweets in the test set.

5.2 Competitor Methods

- TRANSLATION: We use the translation probability $p(\text{response}|\text{tweet})$ to measure the matching level between the response and tweet[2], which is estimated on a variant of IBM model 1 [Brown *et al.*, 1993] adapted for this task.
- COSSIM: We simply calculate the cosine similarity between two short-texts with their TF-IDF representations. This method is still better than random since a good response tends to share words with the original tweet;
- WORDEMBED: We represent each short-text as the sum of the embedding vectors of the words which it contains. The matching score of two short-texts is calculated using a multi-layer perceptron (MLP) with concatenation of the two vectors as input;
- DEEPMATCH$_{topic}$: We employ the matching model in [Lu and Li, 2013] on the basis of topics and train a neural network with 3 hidden layers and 1,000 hidden nodes in the first hidden layer;
- DEEPMATCH$_{cnn}$: We exploit the matching model proposed in [Hu *et al.*, 2014] represented as a convolutional neural network (CNN).
- LR$_{tree}$: To show the power of mined patterns we also train a logistic regression model taking all the mined patterns as input with the contrastive sampling training strategy. This can be viewed as a shallow version of DEEPMATCH$_{tree}$.

The methods can be roughly categorized into pattern-based methods (COSSIM, TRANSLATION, LR$_{tree}$, & DEEPMATCH$_{tree}$) and embedding-based methods (WORDEMBED, DEEPMATCH, & DEEPMATCH$_{cnn}$), where embedding-based methods represent each word with a vector, based on which the final matching decision is made.

[1] Data: `data.noahlab.com.hk/conversation/`

[2] This performs slightly better than $p(\text{tweet}|\text{response})$.

All non-convex models are trained with stochastic gradient descent (SGD) [Le, 2013].We find that their performances are in general quite insensitive to the size of mini-batch.

5.3 Results on Original-vs-Random

In this section we present the results in the orig nal-vs-random setting. For each model, we only report its best performance on the test data, since the large size of test data removes any chance of "accidental cheating". We first study the architecture variations of DEEPMATCH$_{tree}$, and then compare its best setting against the competitors.

Here we compare the performances of DEEPMATCH$_{tree}$ under different settings, more specially, the number of hidden layers (1∼5), NodeDensity (1∼20) and architecture learning (details of results are omitted). In a nutshell, the performance peaks around NodeDensity=10 with architecture learning. With NodeDensity$\geq$ 10, the matching model has over 10^8 parameters, and it needs regularization (e.g., dropout) to prevent overfitting in addition to early stopping. The influence of architecture learning is most salient for a relatively large NodeDensity (say, $>$ 3), while the number of hidden layers stops bringing significant improvement when $\geq$ 3. Generally we found that architectures deeper and larger than the current one does not bring any significant improvement but much slower.

Comparison to Competitor Models

Table 1 compares DEEPMATCH$_{tree}$ to the competitor models. As it shows, our model outperforms all the competitor models with large margins. The contribution of deep architectures is manifested by the differences between the deep architectures and shallow ones with the same mined patterns.

Model	P@1 (1v1)	P@1 (1v9)
COSSIM	0.554	0.377
DEEPMATCH$_{topic}$	0.701	0.330
WORDEMBED	0.774	0.370
TRANSLATION	0.819	0.586
DEEPMATCH$_{cnn}$	0.851	0.496
LR$_{tree}$	0.853	0.652
DEEPMATCH$_{tree}$	**0.889**	**0.708**

Table 1: The results of all models on Original-vs-Random. DeepMatch$_{tree}$ signicantly outperforms all the baselines ($p <$ 0.01 from t-test).

There is a vast gap between pattern-based models and embedding-based models. Although the embedding-based methods can perform fairly well on the one versus one (1v1) setting (0.85+), the performance drops dramatically in the one versus nine (1v9) settings (0.49+), while the pattern-based methods can maintain over 0.55 (dropping from 0.80+) in the same test setting. This contrast suggests that pattern-based methods, with varying coverage in the feature space, are more certain on "matched" positive cases than on negative cases, yielding more reliable ranking results.

5.4 Results on Conversation Data

For each model, we use 5-fold cross validation to choose the hyper-parameter of the ranking model RankSVM and report the best result. Clearly DEEPMATCH$_{tree}$ can greatly improve the performances of retrieving a suitable response from the pool, with significantly better accuracies over the competitor models($p < 0.05$ from t-test). This result is consistent with the result on Original-vs-Random despite the difference in experimental setting.

Model	P@1
BASELINE	0.574
+DEEPMATCH$_{topic}$	0.587
+WORDEMBED	0.579
+TRANSLATION	0.585
+DEEPMATCH$_{tree}$	**0.608**

Table 2: The results on retrieval-based conversation.

5.5 Analysis and Case Study

Deep vs. Shallow Patterns Deep patterns represent information that cannot be adequately modeled by shallow patterns in a deep neural network. Indeed, our study shows that on Original-vs-Random data, P@1 decreases to 0.871 (1v1) and 0.688 (1v9) after removing the deep features. Below is a real case in our experiment. This observation is interesting since feature learning is previously often taken as partially the responsibility of deep learning.

T1 唉，这个周末又要工作
Sigh, have to **work** this **weekend**

$R1_A$ 要注意休息啊
You should **rest** more

$R1_B$ 现在工作不好找啊，准备简历吧
It is hard to find a **job** now, better prepare your **resume**

When trying to find a matched response for T1 , the "deep" pattern $\{\texttt{work} \rightarrow \texttt{weekend} \otimes \texttt{rest}\}$ plays a determining role in picking $R1_A$ over $R1_B$, while shallower patterns as $\{\texttt{work} \otimes \texttt{job}\}$ and $\{\texttt{work} \otimes \texttt{resume}\}$ favor $R1_B$.

The effect of abstraction The abstraction step helps improve the generalization ability of the matching model, by improving P@1 on Original-vs-Random from 0.876 to 0.889 (1v1) and 0.694 to 0.708 (1v9). This can also be illustrated with the following real example from our experiment,

T2 希望小牛明天的比赛可以取胜
Hope Mavericks can **win** the game tomorrow

$R2_A$ 没想到你也支持小牛
Didn't know you **root for** Mavericks too

$R2_B$ 兄弟们，加油！
Go! Brothers

Suppose that for T2 we want to pick a more appropriate response from candidates $R2_A$ and $R2_B$. The mining algorithm (Algorithm 1) discovers the following pattern after the SameEntity abstraction

$$\texttt{hope} \rightarrow \texttt{win} \rightarrow x \otimes \texttt{support} \rightarrow x$$

where x stands for any named entity. This pattern (and its own sub-patterns) then plays an important role in the later matching model in assigning a higher matching score to (T2, $R2_A$), covering more specific patterns like

$$\texttt{hope}\rightarrow\texttt{win}\rightarrow\texttt{Mavericks} \otimes \texttt{support}\rightarrow\texttt{Mavericks}$$

which are filtered out in the mining step for its small support.

6 Related Work

The proposed model is related to several threads of work in natural language processing and machine learning.

Deep Matching Models There are other works on using deep neural networks for the matching task [Huang *et al.*, 2013; Bordes *et al.*, 2014; Sun *et al.*, 2013; Lu and Li, 2013; Hu *et al.*, 2014], which build upon given or learned representations of objects. In our model, we try to directly mine and learn the representations of matching.

Graph-based Kernel DEEPMATCH$_{tree}$ extends the important notions in conventional graph kernels [Vishwanathan *et al.*, 2010] in two senses. First, our model allows matching of two different subgraphs in two domains (e.g., $\{\texttt{work}\rightarrow \texttt{weekend}\}$ in one domain and $\{\texttt{have}\rightarrow \texttt{rest}\}$ in the other), while graph kernels only consider the common subgraphs on two sides. Second, our model captures the nonlinear and hierarchical relations between different matching patterns, while graph kernels simply add them together, with different weights determined by the types of sub-graphs.

String-Rewriting Kernel DEEPMATCH$_{tree}$ is also related to the string-rewriting kernel (SRK) [Bu *et al.*, 2013] for paraphrase identification, in that SRK also generates many patterns of matching and learns to weigh them in training. The main difference is the matching patterns considered in SRK are exhaustively enumerated (although calculated in a smart way), while ours are discovered via a mining algorithm.

7 Conclusion

We propose a generic model for matching two short-texts, which relies on a tree-mining algorithm to discover a vast amount of matching patterns and a DNN to further perform the task using those patterns. Empirical study on the rather difficult task of tweet and response matching shows that our model can outperform competitor with large margins.

8 Acknowledge

This work is supported in part by China National 973 project 2014CB340301. Qun Liu's work is partially supported by the Science Foundation Ireland (Grant 12/CE/I2267 and 13/RC/2106) as part of the ADAPT Centre at Dublin City University.

References

[Bengio, 2009] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, 2009.

[Bordes *et al.*, 2014] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[Brown *et al.*, 1993] Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[Bu *et al.*, 2013] Fan Bu, Hang Li, and Xiaoyan Zhu. An introduction to string re-writing kernel. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2982–2986, 2013.

[Caruana *et al.*, 2001] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, pages 402–408, 2001.

[Fan *et al.*, 2008] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip Yu, and Olivier Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 230–238. ACM, 2008.

[Filippova and Strube, 2008] Katja Filippova and Michael Strube. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*, INLG '08, pages 25–32, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[Hinton *et al.*, 2012] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[Hu *et al.*, 2014] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2042–2050. Curran Associates, Inc., 2014.

[Huang *et al.*, 2013] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[Koehn *et al.*, 2003] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. NAACL '03, pages 48–54. Association for Computational Linguistics, 2003.

[Le, 2013] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[Li, 2011] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 4(1):1–113, 2011.

[Lu and Li, 2013] Zhengdong Lu and Hang Li. A deep architecture for matching short texts. In Christopher J. C. Burges, Lon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 1367–1375, 2013.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[Socher *et al.*, 2011] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems 24*. 2011.

[Sun *et al.*, 2013] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Hybrid deep learning for face verification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[Vishwanathan *et al.*, 2010] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, 2010.

[Wang *et al.*, 2013] Hao Wang, Zhengdong Lu, Hang Li, and Enhong Chen. A dataset for research on short-text conversations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 935–945, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.