CHARACTER-LEVEL DEEP CONFLATION FOR BUSINESS DATA ANALYTICS

Zhe Gan[†], *P. D. Singh*[*], *Ameet Joshi*[⋆], *Xiaodong He*[*], *Jianshu Chen*[*], *Jianfeng Gao*[*], *and Li Deng*[*]

[†]Department of Electrical and Computer Engineering, Duke University, Durham, NC
[*]Microsoft Research, Redmond, WA
[⋆]Microsoft Corporation, Redmond, WA

ABSTRACT

Connecting different text attributes associated with the same entity (conflation) is important in business data analytics since it could help merge two different tables in a database to provide a more comprehensive profile of an entity. However, the conflation task is challenging because two text strings that describe the same entity could be quite different from each other for reasons such as misspelling. It is therefore critical to develop a conflation model that is able to truly understand the semantic meaning of the strings and match them at the semantic level. To this end, we develop a character-level deep conflation model that encodes the input text strings from character level into finite dimension feature vectors, which are then used to compute the cosine similarity between the text strings. The model is trained in an end-to-end manner using back propagation and stochastic gradient descent to maximize the likelihood of the correct association. Specifically, we propose two variants of the deep conflation model, based on long-short-term memory (LSTM) recurrent neural network (RNN) and convolutional neural network (CNN), respectively. Both models perform well on a real-world business analytics dataset and significantly outperform the baseline bag-of-character (BoC) model.

Index Terms— Deep conflation, character-level model, convolutional neural network, long-short-term memory

1. INTRODUCTION

In business data analytics, different fields and attributes related to the same entities (e.g., same person) are stored in different tables in a database or across different databases. It is important to connect these attributes so that we can get a more comprehensive and richer profile of the entity. This is important because exploiting a more comprehensive profile could lead to better prediction in business data analytics. It is also useful in other applications such as name disambiguation [1].

Specifically, the conflation of data aims to connect two rows from the same or different datasets that contain one or more common fields, when the values of the common fields match within a predefined threshold. For example, in the business data considered in this paper, we aim to detect whether two names refer to the same person or not — see the example in Table 1. Row A and row B represent two name fields from different tables in a database, which is a text string consisting of characters. The strings in the same column of Table 1 represent the names of a same person. There are several reasons for the strings in A and B being different: (*i*) possible misspelling typos; (*ii*) the lack of suffix; (*iii*) the reverse of family names and given names. Due to these variations and imperfection in data entries, plain keyword matching does not work well, and we need a data conflation model in the *semantic* level; that is, the model should be able to identify two different character strings to be associated with a same entity.

Table 1: Example text string pairs in the dataset.

A	emilio yentsch	enrique hafner	javier creswell
B	ydntsch emilip	Mr. halner exrique	Prof. crrxwell javzfr

To address the aforementioned challenges, we propose character-level deep conflation models that take the raw text strings as the input and predict whether two data entries refer to the same entity. The proposed model consists of two parts: (*i*) a deep feature extractor, and (*ii*) a ranker. The feature extractor takes the raw text string at the character level and produce a finite dimension representation of the text. In particular, we constructed two different deep architectures of feature extractors: (*i*) long-short-term-memory (LSTM) recurrent neural network (RNN) [2, 3] and (*ii*) deep convolutional neural network (CNN) [4, 5]. Both deep architectures are able to retain the order information in the input text and extract high-level features from raw data, as shown their great success in different machine learning tasks, including text classification [4, 6], machine translation [7, 8, 9, 10] and information retrieval [11, 12, 13]. Furthermore, extracting the features from the *character*-level is critical in many of the recent success in applying deep learning to natural language processing [14, 15, 16, 17, 18]. As we will show later, our proposed deep conflation model achieves high prediction accuracy in the conflation task for business data, and greatly outperform strong baselines.

2. CHARACTER-LEVEL DEEP CONFLATION MODELS

We formulate the deep conflation problem as a ranking problem. That is, given a query string from field A, we rank all the target strings in field B, with the hope that the most similar string in B is ranked on the top of the list. The proposed deep conflation model consists of two parts: (*i*) a deep feature extractor; (*ii*) a ranker. Fig. 1 shows the architecture of the deep conflation model. The deep feature extractors transform the input text strings from character-level into finite dimension feature vectors. Then, the cosine similarity is computed between the query string from field A and all the target strings from field B. The cosine similarity value for each pair of text strings measures the semantic relevance between each pair of the text strings, according to which the target strings are ranked. The entire model will be trained in an end-to-end manner so that the deep

Emails: zhe.gan@duke.edu, {prabhs, ameetj, xiaohe, jianshuc, jfgao, deng}@microsoft.com

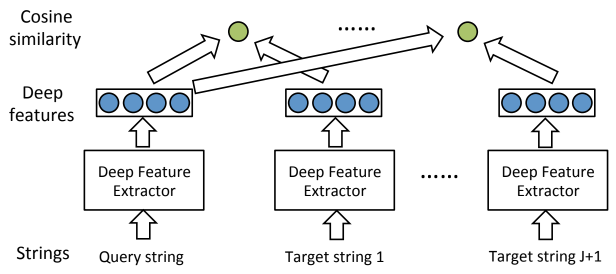

Fig. 1: Character-level deep conflation model.

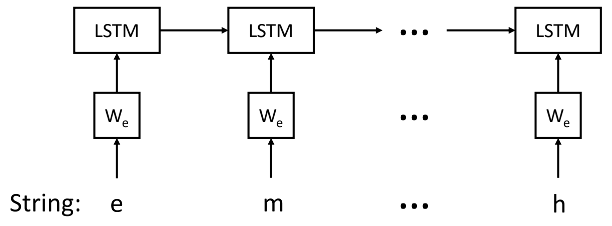

Fig. 2: LSTM based deep feature extractor.

feature extractors are encouraged to learn the proper feature vectors that are measurable by cosine similarity. In the rest of this section, we will explain these two components of the deep conflation model with detail.

2.1. Deep Feature Extractors

The inputs into the system are text strings, which are sequences of characters. Note that the order of the input characters and words is critical to understand the text correctly. For this reason, we propose to use two deep learning models that are able to retain the order information to extract features from the raw input character sequences. The two deep models we use are: (*i*) Recurrent Neural Networks (RNNs); (*ii*) Convolutional Neural Networks (CNNs).

RNN is a nonlinear dynamic system that can be used for sequence modeling. However, during the training of a regular RNN, the components of the gradient vector can grow or decay exponentially over long sequences. This problem with *exploding* or *vanishing* gradients makes it difficult for the regular RNN model to learn long-range dependencies in a sequence [19]. A useful architecture of RNN that overcomes this problem is the Long Short-Term Memory (LSTM) structure. On the other hand, CNN is a deep feedforward neural network that first uses convolutional and max-pooling layers to capture the local and global contextual information of the input sequence, and then uses a fully-connected layer to produce a fixed-length encoding of the sequence. In sequel, we first introduce LSTM, and then CNN.

2.1.1. LSTM feature extractor

The LSTM architecture [2] addresses the problem of learning long-term dependencies by introducing a *memory cell*, that is able to preserve the state over long periods of time. Specifically, each LSTM unit has a cell containing a state $\boldsymbol{c}_t$ at time t. This cell can be viewed as a memory unit. Reading or writing the memory unit is controlled through sigmoid gates: input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{f}_t$, and output gate

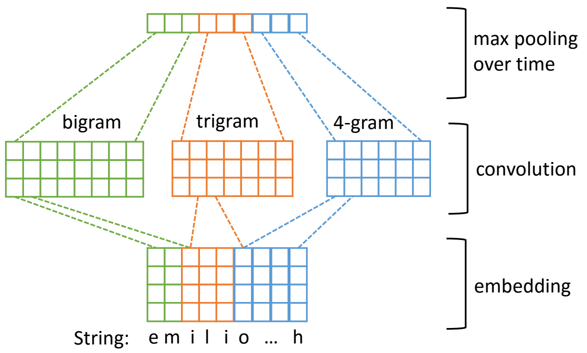

Fig. 3: CNN based deep feature extractor.

$\boldsymbol{o}_t$. The hidden units $\boldsymbol{h}_t$ are updated as follows:

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_i\boldsymbol{x}_t + \mathbf{U}_i\boldsymbol{h}_{t-1} + \boldsymbol{b}_i)\,, \quad (1)$$
$$\boldsymbol{f}_t = \sigma(\mathbf{W}_f\boldsymbol{x}_t + \mathbf{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{b}_f)\,, \quad (2)$$
$$\boldsymbol{o}_t = \sigma(\mathbf{W}_o\boldsymbol{x}_t + \mathbf{U}_o\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)\,, \quad (3)$$
$$\tilde{\boldsymbol{c}}_t = \tanh(\mathbf{W}_c\boldsymbol{x}_t + \mathbf{U}_c\boldsymbol{h}_{t-1} + \boldsymbol{b}_c)\,, \quad (4)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t\,, \quad (5)$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,, \quad (6)$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function, and $\odot$ represents the element-wise multiply operator. $\mathbf{W}_i$ $\mathbf{W}_f$, $\mathbf{W}_o$, $\mathbf{W}_c$, $\mathbf{U}_i$, $\mathbf{U}_f$, $\mathbf{U}_o$, $\mathbf{U}_c$, $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, and $\boldsymbol{b}_c$ are the free model parameters to be learned from training data.

Given the text string $\boldsymbol{q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_T]$, where $\boldsymbol{q}_t$ is the one-hot vector representation of character at position t and T is the number of characters, we first embed the characters into a vector space via a linear transform $\boldsymbol{x}_t = \mathbf{W}_e\boldsymbol{q}_t$, where $\mathbf{W}_e$ is the embedding matrix. Then for every time step, we feed the embedding vector of characters in the text string to LSTM:

$$\boldsymbol{x}_t = \mathbf{W}_e\boldsymbol{q}_t, t \in \{1, \ldots, T\}\,, \quad (7)$$
$$\boldsymbol{h}_t = \text{LSTM}(\boldsymbol{x}_t), t \in \{1, \ldots, T\}\,, \quad (8)$$

where the operator $\text{LSTM}(\cdot)$ denotes the operations defined in (1)-(6). For example, in Fig. 2, the string `emilio yentsch` is fed into the LSTM. The final hidden vector is taken as the feature vector for the string, i.e., $\boldsymbol{y} = \boldsymbol{h}_T$. We repeat this process for the query text and all the target texts so that we will have $\boldsymbol{y}_Q$ and $\boldsymbol{y}_{D_j}$ $(j = 1, \ldots J+1)$, which will be fed into the ranker to compute cosine similarity (see Sec. 2.2).

In the experiments, we use a bidirectional LSTM to extract sequence features, which consists of two LSTMs that are run in parallel: one on the input sequence and the other on the reverse of the input sequence. At each time step, the hidden state of the bidirectional LSTM is the concatenation of the forward and backward hidden states.

2.1.2. CNN feature extractor

Next, we consider the CNN for string feature extraction. Similar to the LSTM-based model, we first embed characters to vectors $\boldsymbol{x}_t = \mathbf{W}_e\boldsymbol{q}_t$ and then concatenating these vectors:

$$\boldsymbol{x}_{1:T} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]\,. \quad (9)$$

Then we apply convolution operation on the character embedding vectors. We use three different convolution filters, which have the size of two (bigram), three (trigram) and four (4-gram), respectively. These different convolution filters capture the context information of different lengths. The t-th convolution output using window size c is given by

$$\boldsymbol{h}_{c,t} = \tanh(\mathbf{W}_c \boldsymbol{x}_{t:t+c-1} + \boldsymbol{b}_c)\,, \tag{10}$$

where the notation $\boldsymbol{x}_{t:t+c-1}$ denotes the vector that is constructed by concatenating $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+c-1}$. That is, the filter is applied only to window $t : t + c - 1$ of size c. $\mathbf{W}_c$ is the convolution weight and $\boldsymbol{b}_c$ is the bias. The feature map of the filter with convolution size c is defined as

$$\boldsymbol{h}_c = [\boldsymbol{h}_{c,1}, \boldsymbol{h}_{c,2}, \ldots, \boldsymbol{h}_{c,T-c+1}]\,. \tag{11}$$

We apply max-pooling over the feature maps of the convolution size c and denote it as

$$\hat{\boldsymbol{h}}_c = \max\{\boldsymbol{h}_{c,1}, \boldsymbol{h}_{c,2}, \ldots, \boldsymbol{h}_{c,T-c+1}\}\,, \tag{12}$$

where the $\max$ is a coordinate-wise max operation. For convolution feature maps of different sizes $c = 2, 3, 4$, we concatenate them to form the feature representation vector of the whole character sequence: $\boldsymbol{h} = [\hat{\boldsymbol{h}}_2, \hat{\boldsymbol{h}}_3, \hat{\boldsymbol{h}}_4]$. Observe that the convolution operations explicitly capture the local (short-term) context information in the character strings, while the max-pooling operation aggregates the information from different local filters into a global representation of the input sequence. These local and global operations enable the model to encode different levels of dependency in the input sequence.

The above vector $\boldsymbol{h}$ is the final feature vector extracted by CNN and will be fed into the ranker, i.e., $\boldsymbol{y} = \boldsymbol{h}$. We repeat this process for the query text and all the target texts so that we will have $\boldsymbol{y}_Q$ and $\boldsymbol{y}_{D_j}$ $(j = 1, \ldots J + 1)$. The above feature extraction process using CNN is illustrated in Fig. 3.

There exist other CNN architectures in the literature [5, 20, 21]. We adopt the CNN model in [4, 22] due to its simplicity and excellent performance on classification. Empirically, we found that it can extract high-quality text string representations for ranking.

2.1.3. Comparison between the two deep feature extractors

Compared with the LSTM feature extractor, a CNN feature extractor may have the following advantages. First, the sparse connectivity of a CNN, which indicates fewer parameters are required, typically improves its statistical efficiency as well as reduces memory requirements. Second, a CNN is able to encode regional (n-gram) information containing rich linguistic patterns. Furthermore, an LSTM encoder might be disproportionately influenced by characters appearing later in the sequence, while the CNN gives largely uniform importance to the signal coming from each of the characters in the sequence. This makes the LSTM excellent at language modeling, but potentially suboptimal at encoding n-gram information placed further back into the sequence.

2.2. Ranker

Now that we have extracted deep feature vectors $\boldsymbol{y}_Q, \boldsymbol{y}_{D_1}, ..., \boldsymbol{y}_{D_{J+1}}$ from the query and candidate strings, we can proceed to compute their semantic relevance scores by computing their corresponding cosine similarity between query Q and each j-th target string D_j. More formally, it is defined as

$$R(Q, D_j) = \frac{\boldsymbol{y}_Q^\top \boldsymbol{y}_{D_j}}{||\boldsymbol{y}_Q|| \cdot ||\boldsymbol{y}_{D_j}||}\,, \tag{13}$$

where D_j denotes the j-th target string. At test time, given a query, the candidates are ranked by this relevance scores.

2.3. Training of the deep conflation model

We now explain how the deep conflation model could be trained in an end-to-end manner. Given that we have the relevance scores between the query string and each of the target string D_j: $R(Q, D_j)$, we define the posterior probability of the correct candidate given the query by the following softmax function

$$P(D^+|Q) = \frac{\exp(\gamma R(Q, D^+))}{\sum_{D' \in \mathbf{D}} \exp(\gamma R(Q, D'))}\,, \tag{14}$$

where D^+ denotes the correct target string (the positive sign denotes that it is a positive sample), γ is a tuning hyper-parameter in the softmax function (to be tuned empirically on a validation set). $\mathbf{D}$ denotes the set of candidate strings to be ranked, which includes the positive sample D^+ and J randomly selected incorrect (negative) candidates $\{D_j^-; j = 1, \ldots, J\}$. The model parameters are learned to maximize the likelihood of the correct candidates given the queries across the training set. That is, we minimize the following loss function

$$L(\boldsymbol{\theta}) = -\log \prod_{(Q,D^+)} P(D^+|Q)\,, \tag{15}$$

where the product is over all training samples, and $\boldsymbol{\theta}$ denotes the parameters (to be learned), including all the model parameters in the deep feature extractors. The above cost function is minimized by back propagation and (mini-batch) stochastic gradient descent.

3. EXPERIMENTAL RESULTS

3.1. Dataset

We evaluate the performance of our proposed deep conflation model on a corporate proprietary business dataset. Since each string can be considered as a sequence of characters, the vocabulary size is 32 (including one period symbol and one space symbol), which includes the following elements:

DMPSabcdefghijklmnopqrstuvwxyz.

Specifically, the dataset contains $10,000$ pairs of query and the associated correct target string (manually annotated). The average length of the string is 14.47 with standard deviation 2.89. The maximum length of the strings is 26 and the minimum length is 6.

3.2. Setup

We provide the deep conflation results using LSTM and CNN for feature extraction, respectively. Furthermore, we also implement a baseline using Bag-of-Characters (BoC) representation of input text string. This BoC vector is then sent into a two-hidden-layer (fully-connected) feed-forward neural networks. In our experiment, we implement 10-fold cross validation, and in each fold, we randomly select 80% of the samples as training, 10% as validation, and the

Table 2: 10-fold cross validation results using BoC, LSTM and CNN model, respectively. **R@K** denotes Recall@K (higher is better). **Med** r, **Mean** r and **Harmonic Mean** r is the median rank, mean rank and harmonic mean rank, respectively (lower is better).

Model	**R@1**	**R@3**	**R@10**	**Med** r	**Mean** r	**Harmonic Mean** r
Using correct names to query mis-spelled names						
BoC	82.09$\pm$ 1.59	92.30$\pm$ 0.76	96.83$\pm$ 0.36	1.0$\pm$ 0.0	2.380$\pm$ 0.218	1.138$\pm$ 0.009
LSTM	86.66$\pm$ 0.90	95.38$\pm$ 0.53	98.54$\pm$ 0.20	1.0$\pm$ 0.0	1.609$\pm$ 0.092	1.095$\pm$ 0.007
CNN	98.90$\pm$ 0.18	99.97$\pm$ 0.05	100.00$\pm$ 0.00	1.0$\pm$ 0.0	1.012$\pm$ 0.003	1.006$\pm$ 0.001
Using mis-spelled names to query correct names						
BoC	83.56$\pm$ 1.42	93.06$\pm$ 0.80	97.35$\pm$ 0.27	1.0$\pm$ 0.0	2.158$\pm$ 0.128	1.131$\pm$ 0.011
LSTM	87.63$\pm$ 0.92	95.50$\pm$ 0.45	98.67$\pm$ 0.21	1.0$\pm$ 0.0	1.584$\pm$ 0.055	1.088$\pm$ 0.007
CNN	99.25$\pm$ 0.43	99.98$\pm$ 0.06	100.00$\pm$ 0.00	1.0$\pm$ 0.0	1.008$\pm$ 0.005	1.004$\pm$ 0.002

Table 3: Average scores for each of the top four retrieved items.

top 1	**top 2**	**top 3**	**top 4**
0.792$\pm$ 0.086	0.448$\pm$ 0.072	0.397$\pm$0.050	0.371$\pm$0.042

Table 4: An example of the mistakenly retrieved cases.

query string	palmer mehaffey	
ground truth	Mr mehaffep paleer	score
1st result	paleer mehaffep	0.882
2nd result	Mr mehaffep paleer	0.877
3rd result	fendlasyn pdlmer	0.427
4th result	zalwzar sharley	0.420

rest 10% as testing dataset. No specific hyper-parameter tuning is implemented, other than early stopping on the validation set.

For the feed-forward neural network encoder based on the BoC representation, we use two hidden layers, each layer contains 300 hidden units, hence each string is embedded as a 300-dimensional vector. For LSTM and CNN encoder, we first embed each character into a 128-dimensional vector. Based on this, for the bidirectional LSTM encoder, we further use one hidden layer of 128 units for sequence embedding, hence each text string is represented as a 256-dimensional vector. For the CNN encoder, we employ filter windows of sizes {2,3,4} with 100 feature maps each, hence each text string is represented as a 300-dimensional vector.

For training, all weights in the CNN and non-recurrent weights in the LSTM are initialized from a uniform distribution in [-0.01,0.01]. Orthogonal initialization is employed on the recurrent matrices in the LSTM [23]. All bias terms are initialized to zero. It is observed that setting a high initial forget gate bias for LSTMs can give slightly better results [24]. Hence, the initial forget gate bias is set to 3 throughout the experiments. Gradients are clipped if the norm of the parameter vector exceeds 5 [9]. The Adam algorithm [25] with learning rate 2×10^{-4} is utilized for optimization. For both the LSTM and CNN models, we use mini-batches of size 100. The hyper-parameter γ is set to 10. The number of negative candidates J is set to 50, which are randomly sampled from the rest of the candidate strings excluding the correct one. All experiments are implemented in Theano [26] on a NVIDIA Tesla K40 GPU. For reference, the training of a CNN model takes around 45 minutes to go through the dataset 20 times.

3.3. Results

Performance is evaluated using Recall@K, which measures the average times a correct item is found within the top-K retrieved results. Results are summarized in Table 2. As can be seen, both of the proposed deep conflation models with LSTM and CNN feature extractors achieve superior performance compared to the BoC baseline. This is not surprising, since sequential order information is utilized in LSTM and CNN. Furthermore, we observe that CNN significantly outperforms LSTM on this task. We hypothesize that this observation is due to the fact that the local (regional) sequential order information (captured by CNN) is more important than the gloabl sequential order information (captured by LSTM) in matching two names. For example, if we reverse the family name and given name of a given query name, LSTM might be more prone to mistakenly classifying these two names to be different, while in fact they refer to the same person.

For further analysis, we checked the CNN results on one predefined train/validation/test splits of the dataset. When CNN is used, for Recall@1, out of 1,000 test samples, only 5 samples are mistakenly retrieved. In Table 4, we show an example of the mistaken case. We can see that the mistakenly retrieved case is quite reasonable. Even humans will make mistakes on these cases. Other four mistakenly retrieved cases are similar and are omitted due to space limit. The average scores for each of the top four retrieved items are given in Table 3. This suggests that, when judging whether two text strings have the same meaning, we can empirically set the threshold to be $(0.792 + 0.448)/2 = 0.62$. That is, when the similarity score between two strings is higher than 0.62, we can safely conclude that they refer to the same entity, and we can then conflate the corresponding two rows accordingly.

4. CONCLUSION

We have proposed a deep conflation model for matching two text fields in business data analytics, with two different variants of feature extractors, namely, long-short-term memory (LSTM) and convolutional neural networks (CNN). The model encodes the input text from raw character-level into finite dimensional feature vectors, which are used for computing the corresponding relevance scores. The model is learned in an end-to-end manner by back propagation and stochastic gradient descent. Since both LSTM and CNN feature extractors retain the order information in the text, the deep conflation model achieve superior performance compared to the bag-of-character (BoC) baseline.

5. REFERENCES

[1] V. I. Torvik and N. R. Smalheiser, "Author name disambiguation in medline," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, pp. 11, 2009.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural computation*, 1997.

[3] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010.

[4] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP*, 2014.

[5] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *ACL*, 2014.

[6] Andrew M Dai and Quoc V Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems*, 2015.

[7] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models.," in *EMNLP*, 2013.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[9] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[10] F. Meng, Z. Lu, M. Wang, H. Li, W. Jiang, and Q. Liu, "Encoding source language with convolutional neural network for machine translation," in *ACL*, 2015.

[11] 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 *CIKM*, 2013.

[12] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *CIKM*, 2014.

[13] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2016.

[14] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush, "Character-aware neural language models," *AAAI*, 2016.

[15] Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso, "Finding function in form: Compositional character models for open vocabulary word representation," *arXiv:1508.02096*, 2015.

[16] Xiang Zhang, Junbo Zhao, and Yann LeCun, "Character-level convolutional networks for text classification," in *NIPS*, 2015.

[17] David Golub and Xiaodong He, "Character-level question answering with attention," *EMNLP*, 2016.

[18] Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio, "A character-level decoder without explicit segmentation for neural machine translation," *arXiv:1603.06147*, 2016.

[19] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "On the difficulty of training recurrent neural networks.," in *ICML*, 2013.

[20] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *NIPS*, 2014.

[21] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," in *NAACL HLT*, 2015.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," in *JMLR*, 2011.

[23] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *ICLR*, 2014.

[24] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv:1504.00941*, 2015.

[25] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[26] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv:1211.5590*, 2012.