k-Nearest Neighbor Augmented Neural Networks for Text Classification

Zhiguo Wang[1], Wael Hamza[1], Linfeng Song[2]
[1] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
[2] Department of Computer Science, University of Rochester, Rochester, NY 14627

arXiv:1708.07863v1 [cs.CL] 25 Aug 2017

Abstract

In recent years, many deep-learning based models are proposed for text classification. This kind of models well fits the training set from the statistical point of view. However, it lacks the capacity of utilizing instance-level information from individual instances in the training set. In this work, we propose to enhance neural network models by allowing them to leverage information from k-nearest neighbor (kNN) of the input text. Our model employs a neural network that encodes texts into text embeddings. Moreover, we also utilize k-nearest neighbor of the input text as an *external memory*, and utilize it to capture instance-level information from the training set. The final prediction is made based on features from both the neural network encoder and the kNN memory. Experimental results on several standard benchmark datasets show that our model outperforms the baseline model on all the datasets, and it even beats a very deep neural network model (with 29 layers) in several datasets. Our model also shows superior performance when training instances are scarce, and when the training set is severely unbalanced. Our model also leverages techniques such as semi-supervised training and transfer learning quite well.

Introduction

Text classification is a fundamental task in both natural language processing and machine learning research. Its goal is to assign specific labels to texts written in natural languages. Based on the definition of the specific labels, text classification has many practical applications, e.g., sentiment analysis (Xia et al., 2013) and news categorization (Li et al., 2009).

In recent years, with the renaissance of neural networks, many deep-learning based methods were proposed for text classification tasks (Zhang, Zhao, and LeCun, 2015; Conneau et al., 2016; Joulin et al., 2016; Johnson and Zhang, 2016). Basically, most of these methods construct some kinds of neural networks to encode a text into a distributed text embedding, and then predict the category of the text solely based on it. In the training stage, network parameters are optimized on the training set. In the testing stage, the entire training set can be discarded, and only the trained model is used for prediction. This method has acquired the state-of-the-art performance in many tasks. However, because it abstracts the training set from a statistical point of view, it cannot utilize instance-level information from individual instances in the training set very well. For example, in the news categorization task, the following news is annotated as the "Business" category.

> *Eastman Kodak Company and IBM will work together to develop and manufacture image sensors used in such consumer products as digital still cameras and camera phones .*

A neural network model with the state-of-the-art performance will incorrectly predict it into the "Sci/Tech" category, because, in the training set, up to 1,166 instances about "IBM" are annotated as the "Sci/Tech" category, whereas only 278 instances about "IBM" are annotated as the "Business" category. From the statistical point of view, we cannot blame our model, because the single word "IBM" is a very strong signal for the "'Sci/Tech' category. However, when we look at the 278 instances with the "Business" category, we found a much relevant instance:

> *IBM and Eastman Kodak Company have agreed to jointly develop and manufacture image sensors for mass - market consumer products , such as digital still cameras .*

Therefore, if we can make use of category information of the relevant training instance, we will have a big chance to correct the error.

On the other hand, instance-based (or non-parametric) learning (Aggarwal, 2014) provides us a good method to capture instance-level information. k-nearest neighbor (kNN) classification is the most representative method, where a predication is made for a new test instance only based on its kNN. Quinlan (1993) showed that a better performance can be achieved if combining the model-based learning and the instance-based learning.

Therefore, in this work, we propose to enhance neural network models with information from kNN. Our model still employs a neural network encoder to abstract global information from the entire training set, and to encode texts into text embeddings. Moreover, we also take the kNN of the input text as an *external memory*, and utilize it to capture instance-level information from the training set. Then, the final prediction is

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

made based on features from both the neural network encoder and the kNN memory. Concretely, for each input text, we first find its kNN in the training set. Second, a neural network encoder is utilized to encode both the input text and its kNN into text embeddings. Third, based on the text embeddings of the input text and the kNN, we calculate attention weights for each neighbor. Based on these attention weights, the model calculates an *attentive kNN label distribution* and an *attentive kNN text embedding*. The final prediction is made based on three sources of features: the text embedding of the input text, the *attentive kNN label distribution*, and the *attentive kNN text embedding*. Experimental results on several standard benchmark datasets show that our model outperforms the baseline model on all the datasets, and it even beats a very deep neural network model (with 29 layers) in several datasets. Our model also shows superior performance when training instances are scarce, and when the training set is severely unbalanced. Our model also leverages techniques such as semi-supervised training and transfer learning quite well.

In following parts, we start with the description of our model, then evaluate the model on some standard benchmark datasets and different experimental settings. We then talk about related work, and finally conclude this work.

Model

In this section, we propose a model to capture both global and instance-level information from the training set for text classification tasks. To capture global information, we train a neural network encoder to encode texts into an embedding space based on all training instances and their category information. To capture instance-level information, for each input text, we search its kNN from the training set, and then take the kNN as an external memory to augment the neural network.

Figure 1 shows the architecture of our model. The blue flow ① is the typical method for text classification, where an input text is encoded into a text embedding by a neural network "Text Encoder", and then a prediction is made based on the text embedding. The remaining flows are our kNN memory, which employs the attention mechanism to extract some instance-level features for prediction. Formally, given an input text x, its kNN $\{x'_1,...,x'_k,...,x'_K\}$ and their correct labels y and $\{y'_1,...,y'_k,...,y'_K\}$, our task can be formulated as estimating a conditional probability $\Pr(y|x, x'_1, ..., x'_K, y'_1, ..., y'_K)$ based on the training set, and predicting the labels for testing instances by

$$y^* = \arg\max_{y \in \mathcal{A}(y)} \Pr(y|x, x'_1, ..., x'_K, y'_1, ..., y'_K), \quad (1)$$

where $\mathcal{A}(y)$ is a set of all possible labels.

Text Encoder

Text Encoder is a critical component in both typical models and our model. Its task is to encode an input text into a text embedding. Typically, an encoder encodes a text in two steps: (1) *word representation step* represents all words in the text as word embeddings or character embeddings (Zhang, Zhao, and LeCun, 2015); and (2) *sentence representation step* composes the word embedding sequence into a fixed-length text embedding with the Convolutional Neural Networks (CNN) (LeCun et al., 1998) or the Long Short-Term Memory Network (LSTM) (Hochreiter and Schmidhuber, 1997) models. For example, Kim (2014), Kalchbrenner, Grefenstette, and Blunsom (2014), and Wang, Mi, and Ittycheriah (2016b) employed the CNN model to encode texts, Wang, Mi, and Ittycheriah (2016a) utilized the LSTM model to represent texts, and Lai et al. (2015) combined both the CNN and the LSTM.

In this work, we utilize a LSTM network to encode texts. For *word representation step*, inspired by Wang et al. (2016) and Wang, Hamza, and Florian (2017), we construct a d-dimensional vector for each word with two components: a word embedding and a character-composed embedding. The word embedding is a fixed vector for each individual word, which is pre-trained with GloVe (Pennington, Socher, and Manning, 2014) or word2vec (Mikolov et al., 2013). The character-composed embedding is calculated by feeding each character (also represented as a vector) within a word into a LSTM. For *sentence representation step*, we apply a bi-directional LSTM (BiLSTM) to compose the word representation sequence, and then concatenate the two vectors from the last time-step of the BiLSTM (both the forward and the backward directions) as the final text embedding.

kNN Memory

kNN memory is the core component of our model. Its goal is to capture instance-level information for each input text from its kNN. This component includes the following six procedures.

Searching for the kNN This procedure, corresponding to the black flow ② in Figure 1, is to find the kNN of the input text from the training set. In order to efficiently search over the large training set, we employ a traditional information retrieve method to find the kNN. We first build an *inverted index* for all texts in the training set with the open source toolkit Lucene (https://lucene.apache.org/). Then, we take the input text as the query, and utilize the simple and effective BM25 ranking function (Robertson, Zaragoza, and others, 2009) to retrieve the kNN from the *inverted index*.

Encoding for the kNN This procedure encodes all the kNN into text embeddings, which corresponds to the yellow flow ③ in Figure 1. We re-utilize the Text Encoder described above, and apply it to each of the K neighbors individually.

Calculating the neighbor attention This procedure corresponds to the gray flow ④ in Figure 1, and its goal is to calculate similarities (*neighbor attention*) between the input text and each of the K neighbors in the embedding space. Formally, let's denote the text embeddings for the input text x and the k-th neighbor x'_k as $\boldsymbol{h}$ and $\boldsymbol{h}'_k$, which are l-dimensional vectors calculated by the Text Encoder. Theoretically, any similarity metrics will fit here. Inspired from Wang et al. (2016); Wang, Hamza, and Florian (2017), here, we adopt

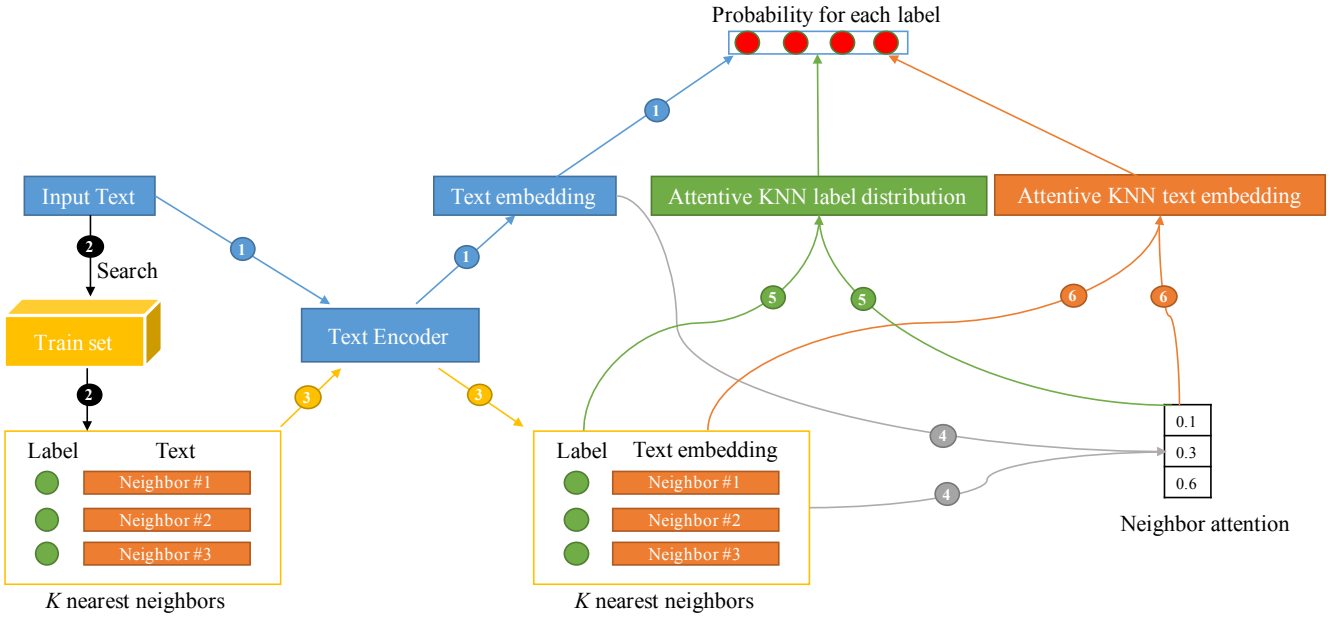

Figure 1: Overall architecture of our model.

the effective *multi-perspective cosine matching* function f_s to compute similarities between two vectors $\boldsymbol{h}$ and $\boldsymbol{h'}_k$:

$$\boldsymbol{s}_k = f_s(\boldsymbol{h}, \boldsymbol{h'}_k; \boldsymbol{W}) \tag{2}$$

where $\boldsymbol{W} \in \Re^{I \times l}$ is a trainable parameter with the shape $I \times l$, I is a hyper-parameter to control the number of perspectives, and the returned value $\boldsymbol{s}_k$ is a I-dimensional vector $\boldsymbol{s}_k = [s_k^1, ..., s_k^i, ..., s_k^I]$. Each element $s_k^i \in \boldsymbol{s}$ is a similarity between $\boldsymbol{h}$ and $\boldsymbol{h'}_k$ from the i-th perspective, and it is calculated by the cosine similarity between two weighted vectors

$$s_k^i = cosine(W_i \circ \boldsymbol{h}, W_i \circ \boldsymbol{h'}_k) \tag{3}$$

where $\circ$ is the element-wise multiplication, and W_i is the i-th row of $\boldsymbol{W}$, which controls the i-th perspective and assigns different weights to different dimensions of the l-dimensional text embedding space.

We set $I = 1$ for the illustration in Figure 1, therefore the neighbor attention is just a vector and each neighbor has only one similarity to the input text. However, for the experiments in following sections, we will utilize multiple perspectives ($I > 1$), and each neighbor could have multiple similarities to the input text.

Calculating the attentive kNN label distribution Based on the neighbor attentions, we calculate the *attentive kNN label distribution* by weighted summing up the label distributions of all kNN (the green flow ⑤ in Figure 1). Formally, let's denote the label distribution of the k-th neighbor x'_k as $\boldsymbol{y'}_k$, which is an one-hot c-dimensional vector for the correct label y'_i, and c is the number of all possible labels in the classification task. Given the label distributions and the neighbor attentions of all kNN, we calculate the i-th perspective of the *attentive kNN label distribution* by

$$\hat{\boldsymbol{y}}_i = \sum_{k=1}^{K} s_k^i * \boldsymbol{y'}_k \tag{4}$$

where $*$ is an operation to multiply the left scalar with each element of the right vector. Then, the final *attentive kNN label distribution* $\hat{\boldsymbol{y}}$ is the concatenation of $\{\hat{\boldsymbol{y}}_1,...,\hat{\boldsymbol{y}}_i,...,\hat{\boldsymbol{y}}_I\}$ from all I perspectives.

Calculating the attentive kNN text embedding Similarly, the *attentive kNN text embedding* is the weighted sum of text embeddings of all kNN (the orange flow ⑥ in Figure 1). Given the text embeddings and neighbor attentions of all kNN, we calculate the i-th perspective of the *attentive kNN text embedding* by

$$\hat{\boldsymbol{h}}_i = \sum_{k=1}^{K} s_k^i * \boldsymbol{h'}_k \tag{5}$$

Then, the final *attentive kNN text embedding* $\hat{\boldsymbol{h}}$ is the concatenation of $\{\hat{\boldsymbol{h}}_1,...,\hat{\boldsymbol{h}}_i,...,\hat{\boldsymbol{h}}_I\}$ from all I perspectives.

Concatenating all features to make the prediction As the final procedure, we concatenate three sources of features (or vectors): the input text embedding $\boldsymbol{h}$, the *attentive kNN label distribution* $\hat{\boldsymbol{y}}$, and the *attentive kNN text embedding* $\hat{\boldsymbol{h}}$. Then, a fully-connected layer with the *softmax* function is applied to make the final prediction.

Experiments

Datasets We evaluate our model on eight publicly available datasets from Zhang, Zhao, and LeCun (2015). Here are the brief descriptions for all datasets.

Dataset	# Classes	Train Samples	Dev Samples	Test Samples
AG's News	4	118,000	2,000	7,600
Sogou News	5	447,500	2,500	60,000
DBPedia	14	553,000	7,000	70,000
Yelp Review Polarity	2	559,000	1,000	38,000
Yelp Review Full	5	647,500	2,500	50,000
Yahoo!Answers	10	1,395,000	5,000	60,000
Amazon Review Full	5	2,997,500	2,500	650,000
Amazon Review Polarity	2	3,599,000	1,000	400,000

Table 1: Statistics of the datasets.

- AG's News: This is a news categorization dataset. All news articles are obtained from the AG's corpus of news article on the web. Each news belongs to one out of the four labels {*World*, *Sports*, *Business*, *Sci/Tech*}.
- Sogou News: This is a Chinese news categorization dataset. All news articles are collected from SogouCA and SogouCS news corpora (Wang et al., 2008). Each news article belongs to one out of the five categories {*sports*, *finance*, *entertainment*, *automobile*, *technology*}. All Chinese characters have been transformed into *pinyin* (a phonetic romanization of Chinese).
- DBPedia: This dataset is designed for classifying Wikipedia articles into 14 ontology classes from DBpedia. Each instance contains the title and the abstract of a Wikipedia article.
- Yelp Review Polarity/Full: This is a sentiment analysis dataset. All reviews are obtained from the Yelp Dataset Challenge in 2015. Two classification tasks are constructed from this dataset. The first one predicts the number of stars the user has given, and the second one predicts a polarity label by considering stars 1 and 2 as negative, and 3 and 4 as positive.
- Yahoo! Answers dataset: This is a topic classification dataset. The dataset is obtained from Yahoo! Answer Comprehensive Question and Answer version 1.0 dataset. Each instance includes the question title, the question content and the best answer. And each instance belongs to one out of 10 topics.
- Amazon Reviews Polarity/Full: This is another sentiment analysis dataset. All reviews are obtained from the Stanford Network Analysis Project (SNAP). Similar to the Yelp Review dataset, a Full version and a Polarity version of the dataset are constructed.

The original datasets didn't provide the devsets. To avoid tuning model parameters on the test sets, for each dataset, we build a devset by randomly holding out 500 instances for each class from the original training set, and take the remaining instances as our new training set. Table 1 shows the statistics of all the datasets.

Experiment Settings We initialize word embeddings with the 300-dimensional GloVe word vectors pre-trained from the 840B Common Crawl corpus (Pennington, Socher, and Manning, 2014). For the out-of-vocabulary (OOV) words, we initialize their word embeddings randomly. For the character-composed embeddings, we represent each character with a 20-dimensional randomly-initialized vector, and feed characters of each word into a LSTM layer to produce a 50-dimensional vector. We set the hidden size to 100 for our BiLSTM Text Encoder. We train the entire model from end-to-end, and minimize the cross entropy of the training set. We use the ADAM optimizer (Kingma and Ba, 2014) to update parameters, and set the learning rate as 0.0001. During training, we do not update the pre-trained word embeddings. For all experiments, we iterate over the training set for 15 times, and evaluate the model on the devset at the end of each iteration. Then, we pick the model which works the best on the devset as the *final* model, and all the results on the test sets are performed from the *final* models.

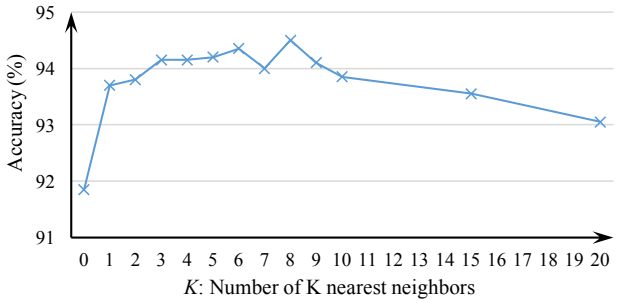

Figure 2: Influence of K nearest neighbors.

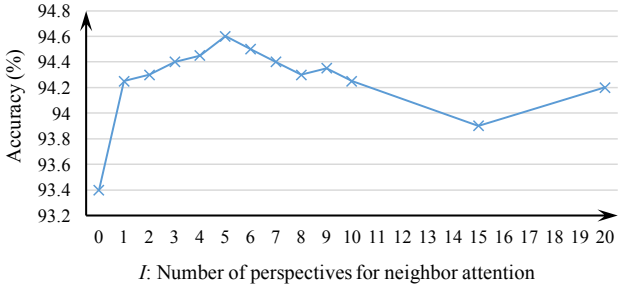

Figure 3: Influence of multi-perspective attentions.

Model ID	Feature Configuration	Accuracy
M1	text-embedding	91.9
M2	attentive-kNN-label	91.5
M3	attentive-kNN-text	92.2
M4	attentive-kNN-label + attentive-kNN-text	93.2
M5	text-embedding + attentive-kNN-label	93.8
M6	text-embedding + attentive-kNN-text	93.6
M7	text-embedding + attentive-kNN-label + attentive-kNN-text	94.6

Table 2: Evaluation of different feature configurations.

Properties of our Model

There are several hyper-parameters in our model. The choices of them may affect the final performance. In this subsection, we conduct some experiments to demonstrate the properties of our model, and select a group of proper hyper-parameters for subsequent experiments. All experiments in this subsection are conducted on the AG's News dataset, and evaluated on the devset.

First, we study the effectiveness of the three sources of features: the input text embedding (text-embedding), the attentive kNN label distribution (attentive-kNN-label), and the attentive kNN text embedding (attentive-kNN-text). We create seven models according to different feature configurations. For all models, we set $K = 20$ for the number of kNN, and $I = 20$ for the number of perspectives in our multi-perspective cosine matching function (Equation (2)). Table 2 shows the corresponding results on the devset, where M1 is our BiLSTM model without using the kNN memory, M2/M3/M4 are the models only utilizing the kNN memory, and M5/M6/M7 are the models leveraging both the text encoder and the kNN memory. From this experiment, we get several interesting observations: (1) when comparing M2 with M1, we find that only utilizing the label information from the kNN can achieve a competitive performance to the typical BiLSTM model; (2) the performance from M3 is on par with M1, which indicates that features solely extracted from the kNN (but without label information) can represent the input text very well; (3) the performance of M4 shows that considering both the label and the text information from the kNN achieves a better performance than the typical BiLSTM model, which shows the effectiveness of our kNN memory; (4) from the results of M5/M6/M7, we can see that combining the kNN memory with our BiLSTM text encoder achieves even better accuracies. The best accuracy is obtained by M7. Therefore, we will use this configuration for the subsequent experiments.

Second, to test the influence of the kNN, we vary K from 1 to 20. Figure 2 shows the accuracy curve, where $K = 0$ corresponds to the performance from our BiLSTM without using the kNN memory. We can see that even with only one neighbor ($K = 1$), our model gets a significant improvement over the BiLSTM model. When increasing the number of neighbors, the performance improves at the beginning, and then drops when K exceeds 8. One possible reason is that the neighbors become noisy when increasing K. In the subsequent experiments, we fix $K = 5$.

Third, we investigate the influence of the hyper-parameter I in our multi-perspective cosine matching function (Equation (2)) by varying I from 1 to 20. We build a baseline by replacing Equation (2) with the vanilla cosine similarity function. Both of the baseline and our model with $I = 1$ calculate a single attention for each neighbor, but the difference is that our model assigns some trainable parameters to each dimension of the embedding space. Figure 3 shows the accuracy curve, where $I = 0$ corresponds to the performance of our baseline. We find that even if utilizing only one perspective ($I = 1$), our model achieves a significant improvement than the baseline. When increasing the number of perspectives, the accuracy improves at the beginning, and then decreases after I is over 5. Therefore, we fix $I = 5$ in the subsequent experiments.

Comparison with the State-of-the-art Models

We construct two models to evaluate on all of the test sets. The first model is the baseline: our BiLSTM model without using the kNN memory (M1 in Table 2). The second model is the BiLSTM model with the kNN memory (M7 in Table 2). Table 3 gives the experimental results. We find that by utilizing the kNN memory, our *BiLSTM with kNN* model outperforms the baseline on all datasets. Among all the state-of-the-art models, the VDCNN model (Conneau et al., 2016) is a very deep network with up to 29 convolutional layers. Our model even beats the VDCNN model on the AG's News, DBpeida and Yahoo! Answers datasets, which shows the effectiveness of our method. Moreover, our kNN memory can be easily adapted into these more complex neural network models.

Evaluation in Other Training Setups

To study the behaviors of our model, we further evaluate it in some other common training setups. All the experiments in this subsection are conducted on the AG's News dataset, and the accuracies are performed on the devset.

Low-Resource Training Setup In the introduction section, we claimed that the kNN memory captures instance-level information from the training set. To verify this claim, we evaluate our model on a low-resource training setup. We construct a low-resource training set by randomly selecting 10% of all instances for each category from the original training set. Then, we train our "BiLSTM" and "BiLSTM

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A	Amz. F.	Amz. P.
BoW[a]	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams[a]	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams-TFIDF[a]	92.4	**97.2**	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN[a]	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN[b]	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN[c]	91.3	96.8	97.7	**95.7**	**64.7**	73.4	**63.0**	**95.7**
fastText-unigram[d]	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
fastText-bigram[d]	92.5	96.8	98.6	**95.7**	63.9	72.3	60.2	94.6
BiLSTM	92.5	94.4	98.9	92.4	59.3	72.5	59.0	94.7
BiLSTM with kNN	**94.2**	96.5	**99.1**	94.5	61.9	**74.4**	60.3	95.3

Table 3: Evaluation on the test sets, and the state-of-the-art models by Zhang, Zhao, and LeCun (2015)[a], Xiao and Cho (2016)[b], Conneau et al. (2016)[c], and Joulin et al. (2016)[d].

Model	Full Setup	Low-Resource Setup	Unbalanced Setup
BiLSTM	91.9	85.2 (-6.7)	48.6 (-43.3)
BiLSTM with kNN	94.6	90.2 (-4.4)	90.6 (-4.0)

Table 4: Evaluation in the rare-resource setup and the unbalanced training setup, where the numbers in brackets show the change of accuracy against the "Full Setup".

with kNN" models, with the same configurations as before, on this low-resource training set. In Table 4, the third column, with the title "Low_Resource Setup", shows the accuracies of our two models. Comparing with the models trained with the full training set ("Full Setup"), our "BiLSTM with kNN" model only drops 4.4 percent, which is lower than the 6.7 percent drop from our "BiLSTM" model. This result shows our kNN memory can capture instance-level information to remedy shortage of the low-resource training set.

Unbalanced Training Setup In this sub-section, we evaluate our model on an unbalanced training set, a scenario not uncommon in text classification. To severely skew the label distribution, we construct an unbalanced training set by randomly selecting 2,000 instances for the *World* category, 4,000 instances for the *Sports* category, 8,000 instances for the *Business* category and 16,000 instances for the *Sci/Tech*. We train both our BiLSTM and proposed models (with the same configuration as before) on this unbalanced training set. The last column of Table 4 shows the accuracies. The BiLSTM model shows a severe (43.3%) degradation of accuracy in the unbalanced setting. On the other hand, our *BiLSTM with kNN* model only drops 4.0% when trained on the unbalanced training set. We believe this is because our model can capture instance-level information from the unbalanced training set.

Semi-supervised Training and Transfer Learning So far, we have verified that the effectiveness of the kNN retrieved from the same training set. *What if we search the kNN from a dataset of a completely different task*? If we consider the text (without the label) information of the kNN from a different task, then this becomes the **semi-supervised training** setup. If we utilize the label information of the kNN from a different task, where the definition of labels are quite different from the task at hand, then this becomes the **transfer learning** setup. To reveal the behavior of our model in these two setups, we take the training set of the DBPedia dataset as the corpus to find the kNN for each text in the AG's News dataset. We construct four models: "BiLSTM" using the "M1" configuration in Table 2, "Semi-supervised Training" using the "M6" configuration, "Transfer Learning (M5)" using the "M5" configuration, and "Transfer Learning (M7)" using the "M7" configuration. Here, we should notice that the "attentive-kNN-text" and the "attentive-kNN-label" features are extracted from the kNN from the DBPedia corpus, which is a different classification task. Table 5 gives the results. We can see that our model achieves improvements from the baseline (BiLSTM) in both the *Semi-supervised Training* and the *Transfer Learning* setups.

Setup	Accuracy
BiLSTM (M1)	91.9
Semi-supervised Training (M6)	92.9
Transfer Learning (M5)	92.7
Transfer Learning (M7)	93.4

Table 5: Evaluation in the semi-supervised training and the transfer learning setups.

Qualitative Analysis

We perform qualitative analysis by looking at some instances incorrectly predicted by the baseline (*BiLSTM*) but get corrected by adding the kNN memory (*BiLSTM with kNN*).

First, for the error illustrated in the introduction section, our model corrected it as expected.

Another example is "*President George W. Bush 's campaign website was inaccessible from outside the United States* ." with the correct label *Sci/Tech*. In the training set, the appearances of the phrase "George W. Bush" in each category are 1,659 (World), 410 (Business), 67 (Sports) and 299 (Sci/Tech), and the frequencies of the word "President" in each category are 4,486 (World), 1,021 (Business), 233 (Sports) and 554 (Sci/Tech). Based on these strong signals, the BiLSTM baseline assigned it with an incorrect label *World*. On the other hand, our *BiLSTM with kNN* corrected it, because there is a very similar neighbor in the training set "*President George W. Bush 's official re-election website was down and inaccessible for hours , in what campaign officials said could be the work of hackers* ." with the label *Sci/Tech*.

Related Work

In recent years, there have been several studies trying to augment neural network models with external memories. Generally, these models utilize the attention mechanism to access useful information outside the model itself (so-called the external memory). For the machine translation task, Bahdanau, Cho, and Bengio (2014) introduced the attention mechanism to access source side encoding information while generating the target sequence. For the question answering task, Weston, Chopra, and Bordes (2014) proposed the memory network to access all supporting sentences before generating the correct answer word. Graves, Wayne, and Danihelka (2014) designed a more complicated "computer-like" external memory to simulate the Turning Machine. Our model also belongs to this group, but with the distinction that we construct the external memory with the K nearest neighbors of the input text, and utilize a multi-perspective attention model.

Vinyals et al. (2016) proposed a matching network for one shot learning task. Their model also classifies the input instance by utilizing a labeled support set as our model does. However, our model differentiates from their model in two ways. First, they assume the labeled support set is given beforehand, whereas our model searches the kNN independently based on the input instance. Second, their model only utilizes the label information of the support set for prediction, whereas our model makes use of information from both the input text and the kNN.

Our model follows an old idea of combining the model-based (or parametric) learning and the instance-based (or non-parametric) learning (Quinlan, 1993). We infuse the old idea with the advanced neural networks and the attention mechanism.

Conclusion

In this work, we enhanced neural networks with a kNN memory for text classification. Our model employs a neural network encoder to abstract information from the entire training set, and utilizes the kNN memory to capture instance-level information. The final prediction is made based on features from both the input text and the kNN. Experimental results on several standard benchmark datasets show that our model outperforms the baseline model on all the datasets, and it even beats a very deep neural network model (with 29 layers) in several datasets. Our model also shows superior performance when training instances are scarce, and when the training data is severely unbalanced. Our model also leverages techniques such as semi-supervised training and transfer learning quite well.

References

Aggarwal, C. C. 2014. Instance-based learning: A survey.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Conneau, A.; Schwenk, H.; Barrault, L.; and Lecun, Y. 2016. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*.

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Johnson, R., and Zhang, T. 2016. Supervised and semi-supervised text categorization using lstm for region embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, 526–534.

Joulin, A.; Grave, E.; Bojanowski, P.; and Mikolov, T. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lai, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, 2267–2273.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Li, S.; Xia, R.; Zong, C.; and Huang, C.-R. 2009. A framework of feature selection methods for text categorization. In *ACL 2009*, 692–700. Association for Computational Linguistics.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Quinlan, J. R. 1993. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, 236–243.

Robertson, S.; Zaragoza, H.; et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval* 3(4):333–389.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 3630–3638.

Wang, C.; Zhang, M.; Ma, S.; and Ru, L. 2008. Automatic online news issue construction in web environment. In *Proceedings of the 17th international conference on World Wide Web*, 457–466. ACM.

Wang, Z.; Mi, H.; Hamza, W.; and Florian, R. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*.

Wang, Z.; Hamza, W.; and Florian, R. 2017. Bilateral multi-perspective matching for natural language sentences. In *IJCAI 2017*.

Wang, Z.; Mi, H.; and Ittycheriah, A. 2016a. Semi-supervised clustering for short text via deep representation learning. In *CoNLL 2016*.

Wang, Z.; Mi, H.; and Ittycheriah, A. 2016b. Sentence similarity learning by lexical decomposition and composition. In *Coling 2016*.

Weston, J.; Chopra, S.; and Bordes, A. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.

Xia, R.; Wang, T.; Hu, X.; Li, S.; and Zong, C. 2013. Dual training and dual prediction for polarity classification. In *ACL 2013*, 521–525.

Xiao, Y., and Cho, K. 2016. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.