Explainable Neural Approaches to Question Classification

Kai Liang*
University of Amsterdam
Amsterdam, the Netherlands
kai.liang@student.uva.nl

Xiaoxiao Wen*
University of Amsterdam
Amsterdam, the Netherlands
xiaoxiao.wen@student.uva.nl

Weitao Luo*
University of Amsterdam
Amsterdam, the Netherlands
weitao.luo@student.uva.nl

Yijie Zhang*
University of Amsterdam
Amsterdam, the Netherlands
yijie.zhang@student.uva.nl

ABSTRACT

We report on our findings in the implementation and comparisons of several text classification architectures, which include both non-neural (i.e. FastText) and neural models (i.e. LSTM and TextCNN). We conclude that TextCNN is superior to LSTM in terms of various evaluation metrics. In addition, we employ unsupervised learning in the neural classification models to extract rationales as justifications for the prediction, which provides better interpretability of models.

KEYWORDS

text classification, deep learning, rationale extraction, unsupervised learning

ACM Reference Format:
Kai Liang, Xiaoxiao Wen, Weitao Luo, and Yijie Zhang. 2019. Explainable Neural Approaches to Question Classification. In *UvA Course '19: Deep Learning for Natural Language Processing, Sept. 2019, Amsterdam, NL.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Text classification is a classical task for natural language processing (NLP) [8]. In this study, we revisit the task of question classification, which is essential for many applications such as question answering [16]. In particular, our focus lies in two parts. Firstly, we implement and train a set of neural classifiers (i.e. LSTM and TextCNN) given full input sequences, and compare with our non-neural baseline (i.e. FastText) with respect to various metrics, including overall accuracy, per-category precision, recall and f1-score. Then, we incorporate unsupervised learning in the neural classifiers by adding a layer of binary latent variables which extracts a subset of tokens (rationales) from the input as features for classification. The rationale extraction model (a.k.a. generator) is optimized jointly with the classifiers, and the gradients are estimated using REINFORCE [15].

*All authors contributed equally to this research.

UvA DLNLP '19, Sept., 2019, Amsterdam, NL

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$0.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Traditional text classifiers mainly rely on machine learning techniques such as nearest neighbours, naive Bayes, decision tree and support-vector machines [16] with hand-crafted features [8]. In recent years, with the emergence of deep learning, many NLP tasks, including text classification, are dominated by complex neural models (e.g. LSTM [4, 5], TextCNN [7], RCNN [8] and BERT [3]). However, such neural models are not flawless, and their applicability is usually limited by their processing speed and interpretability [9].

Previous works tackle the first problem with different approaches, for instance, simple but powerful non-neural models, such as FastText [6], are proposed to increase the processing speed, which is commonly used in industry and we use in this study as a baseline for comparisons to our neural models. For the second problem, both deterministic and generative mechanisms are proposed. Within the deterministic scope, attention-based models [2, 14] are broadly applied to many NLP problems to increase the understanding of the internal workings of neural networks [9]. On the other hand, from the generative perspective, latent variables which are trained via variational inference can control the selection of tokens from the input sequences to feed into the classifier. The extracted tokens expose features for classification, and thus better explicate the neural models.

3 METHODS

3.1 FastText

FastText is a simple and efficient model for text classification, which is proved by previous research that is comparable with deep learning classifiers in terms of evaluation metrics such as accuracy, while a lot faster for both training and evaluation [6]. The architecture of FastText is similar to CBOW [11], except that the task is turned from predicting the middle word to the label of the text sequence. Particularly, the embeddings of the n-gram features of a sequence are first averaged to obtain the hidden vector, which is then fed to a linear classifier. The final prediction output can be computed by the softmax function as common neural classifiers.

3.2 LSTM

Long short-term memory (LSTM) [4] is an improved RNN architecture which is proposed to overcome the gradient vanishing problem of vanilla RNN with feedback connections. It introduces an adaptive gating mechanism [5], which decides how the memory of previous

states and the information of the current input should be combined. LSTM processes a given sequence token by token. In general, an LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. At time step t, the cell state c_t and the hidden state h_t are updated according to the following equations:

$$
\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}\right) \\
f_t &= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}\right) \\
g_t &= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}\right) \\
o_t &= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}\right) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

where i_t, f_t, g_t, o_t are the input, forget, cell and output gates respectively.

Furthermore, a bi-directional LSTM is utilized in this study to capture both the past and future information at the same time. By applying the bi-directional LSTM, the amount of input information is increased and more sequence contexts are provided for the network. A bi-directional LSTM contains two sub-networks which are for the forward and backward pass respectively:

$$
\begin{aligned}
\overrightarrow{\mathbf{h}_t} &= \overrightarrow{f}\left(\mathbf{x}_t, \overrightarrow{\mathbf{h}_{t-1}}\right) \\
\overleftarrow{\mathbf{h}_t} &= \overleftarrow{f}\left(\mathbf{x}_t, \overleftarrow{\mathbf{h}_{t+1}}\right) \\
\mathbf{h}_t &= \left[\overrightarrow{\mathbf{h}_t} \oplus \overleftarrow{\mathbf{h}_t}\right]
\end{aligned}
$$

where $\oplus$ is the element-wise summation.

3.3 TextCNN

Adapted from traditional CNN models which are generally used for image processing, TextCNN [7] makes minor changes and replicates the huge success on the task of sequence classification. Specifically, TextCNN involves an embedding layer, a convolution layer, a max-pooling layer, a full connection layer and a softmax layer. The major difference of TextCNN from traditional CNN is that the kernels are no longer square matrices (e.g. 3×3), while instead in shape $N\times|V|$, where $|V|$ is an abused notation which stands for the dimensionality of word embeddings. Additionally, by applying max-pooling on the convoluted output, the model ensures fixed-length hidden vectors regardless of the input lengths.

3.4 Rationale Extraction

The goal of rationale extraction is to extract a subset of tokens from the input sequence (a.k.a. *rationales*), which need to be interpretable and sufficient for the prediction as the original sequence [1, 9]. The main components for rationale extraction are the generator network $\mathbf{gen}(\cdot|\phi)$ and the encoder network $\mathbf{enc}(\cdot|\theta)$.

3.4.1 Generator. The generator network $\mathbf{gen}(\cdot|\phi)$ is used to extract rationales from the input sequence, where ϕ denotes the parameters of the generator. $\mathbf{gen}(\cdot|\phi)$ takes a sequence of tokens $\mathbf{x}$ as input and generates the extractive rationale $\mathbf{gen}(\mathbf{x}|\phi)$.

In detail, $\mathbf{gen}(\cdot|\phi)$ works by sampling a binary latent variable $\mathbf{z}$ from a learnable Bernoulli distribution $\mathbf{Bern}(\mathbf{z}|\mathbf{p})$ for every token in $\mathbf{x}$. The parameters $\mathbf{p}$ for the distribution are learned from a pre-generator network as $\mathbf{p} = \mathbf{pregen}(\mathbf{x}|\phi)$. In short, we can define the distribution of the latent variable $\mathbf{z}$ as:

$$p(\mathbf{z}|\mathbf{x},\phi) = \mathbf{Bern}(\mathbf{z}|\mathbf{pregen}(\mathbf{x}|\phi))$$

Considering the different lengths of the input sequences, the pre-generator composes of an embedding layer connected to an LSTM layer [4], followed by an average pooling or a fully connected layer performing on the output vector for each time step with sigmoid activation. By applying element-wise multiplication between the input sequence $\mathbf{x}$ and the latent variable $\mathbf{z}$, we obtain the extracted rationale as:

$$\mathbf{gen}(\mathbf{x}|\phi) = \mathbf{x} \odot \mathbf{z}$$

We note that the selection of each token is conditionally independent given the input sequence $\mathbf{x}$, as shown by:

$$p(\mathbf{z}|\mathbf{x},\phi) = \prod_{t=1}^{l} p(\mathbf{z}_t|\mathbf{x},\phi)$$

where l represents the length of the input sequence $\mathbf{x}$.

3.4.2 Encoder. The encoder network $\mathbf{enc}(\cdot|\theta)$ is used for the classification task given the input sequence or extracted rationale, where θ denotes the parameters for the encoder network. The architecture of the encoder $\mathbf{enc}(\cdot|\theta)$ can potentially be any arbitrary classifier that takes some sequence of tokens $\mathbf{x}$ as input and predicts a probability distribution for the target classes, which in our case are LSTM or TextCNN. The network maps the input $\mathbf{x}$ to the target distribution $p(y|\mathbf{x},\theta)$ (a.k.a. likelihood of the target y given the parameters θ). For classification, the target distribution is considered to be a categorical distribution $\mathbf{Cat}(y|\mathbf{enc}(\mathbf{x},\theta))$.

3.4.3 Objective. As rationale extraction involves two separate models that are jointly optimised, the objective as a whole incorporates two aspects, where one is to train the generator to extract short and meaningful rationales from the input $\mathbf{x}$, and the other is to train the encoder to predict the correct label y for the input sequence or rationales.

To train the generator and encoder to generate meaningful rationales, we formulate the task as maximizing the conditional likelihood of the target $p(y|\mathbf{x},\theta) = \mathbf{Cat}(y|\mathbf{enc}(\mathbf{x},\theta))$, which is also conditioned on the latent variable $\mathbf{z}$ as:

$$p(y|\mathbf{x},\theta) = \int_{\mathbf{z}} p(y|\mathbf{x},\mathbf{z},\theta)p(\mathbf{z}|\mathbf{x},\phi)d\mathbf{z} = \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[p(y|\mathbf{x},\mathbf{z},\theta)]$$

By applying Jensen's inequality on the conditional log-likelihood of the target [1], we have:

$$\log p(y|\mathbf{x},\theta) = \log \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[p(y|\mathbf{x},\mathbf{z},\theta)] \geq \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[\log p(y|\mathbf{x},\mathbf{z},\theta)]$$

where we define $\mathcal{E}(\phi,\theta) = \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[\log p(y|\mathbf{x},\mathbf{z},\theta)]$ as the lower bound which we can instead maximize w.r.t. the encoder and generator parameters θ and ϕ respectively. Here $\log p(y|\mathbf{x},\mathbf{z},\theta)$ is the conditional log-likelihood of the target conditioned on the latent variable $\mathbf{z}$, so we know it also follows the Categorical distribution

$\mathbf{Cat}(y|\mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta))$, thus:

$$\begin{aligned}\log p(y|\mathbf{x}, \mathbf{z}, \theta) &= \log \mathbf{Cat}(y|\mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta)) \\ &= \log \prod_{k=1}^{K} \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta)_k^{I[y=k]} \\ &= \sum_{k=1}^{K} I[y = k] \log \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta)_k \\ &= -\mathbf{CrossEnt}(y, \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta))\end{aligned}$$

where $\mathbf{CrossEnt}(p, q)$ denotes the cross entropy between two distributions p and q. Hence, we see that:

$$\begin{aligned}\max \mathcal{E}(\phi, \theta) &= \max \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[-\mathbf{CrossEnt}(y, \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta))] \\ &= \min \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[\mathbf{CrossEnt}(y, \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta))]\end{aligned}$$

Furthermore, to ensure that generated rationales are short and coherent, additional regularizers are added to penalize for the number of selections and discontinuity in the selections [9]. Lastly, we obtain the loss function as:

$$\begin{aligned}\mathcal{L}(y, \mathbf{x}, \mathbf{z}) &= \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[L(y, \mathbf{x}, \mathbf{z})] \\ &= \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[\mathbf{CrossEnt}(y, \mathbf{enc}(\mathbf{x} \odot \mathbf{z}, \theta)) \\ &\quad + \lambda_1 \|\mathbf{z}\| + \lambda_2 \sum_{t=2}^{l} |\mathbf{z}_t - \mathbf{z}_{t-1}|]\end{aligned}$$

In order to train the generator, (mini-batch) stochastic gradient descent is required for minimizing the loss function $\mathcal{L}(y, \mathbf{x}, \mathbf{z})$ w.r.t. ϕ. However, as the loss function $\mathcal{L}(y, \mathbf{x}, \mathbf{z})$ includes the expection w.r.t. $p(\mathbf{z}|\mathbf{x}, \phi)$ which is dependent on the generator parameter ϕ, it requires sampling from $p(\mathbf{z}|\mathbf{x}, \phi)$, while the gradient cannot be back propagated, so REINFORCE is used [9]:

$$\begin{aligned}\frac{\partial \mathcal{L}(y, \mathbf{x}, \mathbf{z})}{\partial \phi} &= \frac{\partial \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[L(y, \mathbf{x}, \mathbf{z})]}{\partial \phi} \\ &= \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[L(y, \mathbf{x}, \mathbf{z}) \frac{\partial \log p(\mathbf{z}|\mathbf{x}, \phi)}{\partial \phi}]\end{aligned}$$

Similarly, to train the encoder, (mini-batch) stochastic gradient descent is performed to minimize the loss function $\mathcal{L}(y, \mathbf{x}, \mathbf{z})$ w.r.t. θ. This is simply achieved by:

$$\frac{\partial \mathcal{L}(y, \mathbf{x}, \mathbf{z})}{\partial \theta} = \mathbb{E}_{p(\mathbf{z}|\mathbf{x},\phi)}[\frac{\partial L(y, \mathbf{x}, \mathbf{z})}{\partial \theta}]$$

4 EXPERIMENTS

4.1 Dataset

We test our model on the Question Classification dataset [10], which contains in total 6 coarse classes: abbreviation, entity, description, human, location and numeric. Although there are more fine-grained classes, we only evaluate the classification results on these coarse ones. Besides, the original dataset is unbalanced on some classes, so we randomly choose 2000 data for each class except for abbreviation, where only 200 are selected for training.

Furthermore, the original dataset originally contains only a training set and a test set. For the purpose of tuning the hyperparameter settings, the training set is split into a training set and a validation set. After balancing the number of data for each class, the first 90% is selected as the final training set and the rest 10% as the validation set.

The implementation of the data pipeline requires the tokens in the dataset to be stemmed by the Porter stemmer [13] during both training and evaluation to have a generalized vocabulary. Also, in order to form mini-batches of the data, in each batch, the length of the input sequences is padded to the longest input sequence in the batch.

4.2 Pre-trained Word Vectors

Due to the limited size of our training dataset, we initialize the word embeddings of all the neural models in this study with Google's pre-trained 300d `word2vec` [12] which is publicly available online[1]. The pre-trained word embeddings are truncated according to the vocabulary of the training and validation set. A uniformly initialized ($\mathcal{U}(-0.05, 0.05)$) embedding is used for the unknown tokens and a zeros embedding is used for the paddings. However, to better adapt to our task and data, we allow fine-tuning for the word embeddings during training. As for the FastText model, the word embeddings are initialized randomly.

4.3 Model Settings

We perform grid search on all models implmented in this study to find sub-optimal hyperparameter settings, and the final settings are listed in Table 2 (FastText), Table 3 (LSTM), Table 4 (TextCNN), and Table 5 ($\mathbf{gen}(\cdot|\phi)$) respectively.

5 RESULTS

5.1 Classification

Results of the three models (i.e. FastText, LSTM and TextCNN) on the classification task with full input sequences are listed in Table 1. We evaluate the model perforamce in terms of overall accuracy, per-category precision, recall and f1-score, which are common statistics for such tasks.

5.2 Rationale Extraction

Given the classification results, TextCNN is selected as the encoder for the rationale extraction experiments. The results of using a fully-connected layer compared with using an average pooling layer in the LSTM in the generator network are listed in Table 6, where the keep rate is a measure of the percentage of original content kept as extractive rationales. In addition, some samples of the extracted rationales are displayed in Table 7 for qualitative evaluation.

6 DISCUSSION

As listed in section 5, the classification results reveal that both of the neural models are superior to the non-neural baseline (i.e. FastText) in terms of the overall accuracy. Also, our TextCNN model achieves the best performance not only in the overall accuracy, but also in the f1-score of 4 classes, which proves its strong model capacity.

As for the results of rationale extraction, it is shown that when the aggregation of the LSTM output is learned with a fully-connected layer, the overall accuracy for classification is decreased by 19.4%, while only 50.3% of tokens are used. On the other hand, when the

[1] https://code.google.com/archive/p/word2vec/

Table 1: Classification results of FastText, LSTM and TextCNN on full input sequences.

Model	Overall Accuracy	Abbreviation			Entity			Description			Human			Location			Numeric		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
FastText	0.860	1.00	0.78	**0.88**	0.83	0.62	0.71	0.79	0.98	0.87	0.87	0.94	**0.90**	0.88	0.86	0.87	0.97	0.88	**0.92**
LSTM	0.862	0.83	0.56	0.67	0.86	0.76	**0.80**	0.81	0.95	0.88	0.88	0.89	0.89	0.82	0.90	0.86	0.98	0.82	0.89
TextCNN	**0.876**	0.83	0.56	0.67	0.93	0.66	0.77	0.82	0.99	**0.90**	0.88	0.92	**0.90**	0.84	0.93	**0.88**	0.96	0.88	**0.92**

Table 2: Hyperparameter settings of FastText.

Parameter	Value
lr	0.1
epoch	70
lr_update_rate	100
embed_dim	100
window_size	5
min_count	1

Table 3: Hyperparameter settings of LSTM.

Parameter	Value
lr	0.001
epoch	10
batch_size	64
max_grad_norm	5.0
lstm_layer	2
lstm_direction	2

Table 4: Hyperparameter settings of TextCNN.

Parameter	Value
lr	0.001
epoch	10
batch_size	64
max_grad_norm	5.0
kernel_size	(2, 3, 4)
kernel_num	32
dropout	0.5

Table 5: Hyperparameter settings of the generator network gen$(\cdot|\phi)$.

Parameter	Value
lr	0.001
epoch	50
batch_size	64
max_grad_norm	5.0
lstm_layer	2
lstm_direction	2
λ_1	0.01
λ_2	0.001

aggregation is simply averaging over the LSTM output, the overall accuracy for classification is decreased by 32.2%, while only 27.6% tokens are used. If comparing the ration between the overall accuracy and the keep rate, we find that $\frac{0.706}{0.503} = 1.4 < \frac{0.594}{0.276} = 2.152$, which trivially indicates that using the average pooling layer leads to a better performance in extracting short and meaningful rationales. Furthermore, from the qualitative results, we observe that for both models, the generator network is able to extract keywords that might have the largest contribution to the classification, for instance, 'who' for the class 'Human' and 'state' for the class 'Entity'.

Table 6: Classification results and keep rates of rationales of LSTM-FC (fully-connected) and LSTM-AP (average pooling).

Model	Overall Accuracy	Keep Rate
LSTM-FC	**0.706**	0.503
LSTM-AP	0.594	**0.276**

Table 7: Example of extracted rationales (bold) of LSTM-FC and LSTM-AP.

Model	Extracted Rationales	Class
LSTM-FC	**what is** maryland <unk> **state** bird <unk>	Entity
LSTM-AP	**who** discov <unk> <unk>	Human

7 CONCLUSION

In this study, we firstly implement and compare a set of models (i.e. FastText, LSTM and TextCNN) for the task of question classification. Through extensive experiments, we conclude that both the neural models (i.e. LSTM and TextCNN) outperform the non-neural model (i.e. FastText) in terms of various metrics, and TextCNN achieves the best performance in our dataset. We then combine unsupervised learning with the classification models by adding a layer of binary latent variables to extract rationales from the input sequences for classification, which is trained via a REINFORCE-style algorithm. Our results confirm that the extracted rationales as features are able to reach a similar-level of test performance as the original sequences, which enables better interpretability of our neural models.

REFERENCES

[1] Joost Bastings, Wilker Aziz, and Ivan Titov. 2019. Interpretable Neural Predictions with Differentiable Binary Variables. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2963–2977. https://doi.org/10.18653/v1/P19-1284

[2] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. 2015. ABC-CNN: An Attention Based Convolutional Neural Network for Visual Question Answering. (2015). arXiv:cs.CV/1511.05960

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2018). arXiv:cs.CL/1810.04805

[4] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A Search Space Odyssey. *arXiv e-prints*, Article arXiv:1503.04069 (Mar 2015), arXiv:1503.04069 pages. arXiv:cs.NE/1503.04069

[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[6] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *arXiv e-prints*, Article arXiv:1607.01759 (Jul 2016), arXiv:1607.01759 pages. arXiv:cs.CL/1607.01759

[7] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv e-prints*, Article arXiv:1408.5882 (Aug 2014), arXiv:1408.5882 pages. arXiv:cs.CL/1408.5882

[8] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 2267–2273. http://dl.acm.org/citation.cfm?id=2886521.2886636

[9] Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing Neural Predictions. (2016). arXiv:cs.CL/1606.04155

[10] Xin Li and Dan Roth. 2002. Learning Question Classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. (2013). arXiv:cs.CL/1301.3781

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *arXiv e-prints*, Article arXiv:1310.4546 (Oct 2013), arXiv:1310.4546 pages. arXiv:cs.CL/1310.4546

[13] M. F. Porter. 1997. Readings in Information Retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Chapter An Algorithm for Suffix Stripping, 313–316. http://dl.acm.org/citation.cfm?id=275537.275705

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. (2017). arXiv:cs.CL/1706.03762

[15] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8, 3-4 (May 1992), 229–256. https://doi.org/10.1007/BF00992696

[16] Dell Zhang and Wee Sun Lee. 2003. Question Classification Using Support Vector Machines. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*. ACM, New York, NY, USA, 26–32. https://doi.org/10.1145/860435.860443