Re-Implementation of Deep Short Text Classification with Knowledge Powered Attention
ECE 57000 Project

Anonymous Authors[1]

Abstract

This paper studies three different papers and re-implement an experiment provided in one of the papers. The three papers examined are "Deep Pyramid Convolutional Neural Networks for Text Classification" (Johnson & Zhang, 2017), "Deep Short Text Classification with Knowledge Powered Attention" (Chen et al., 2019), and finally "Initializing Convolutional Filters with Semantic Features for Text Classification" (Li et al., 2017). The motivation behind studying natural language processing is to fulfill the goal of specializing in this field especially because NLP is becoming extremely predominant in today's society. This paper will be re-implementing the experimentation conducted by Chen et al. by first figuring out how to run the author's implementation, understanding and writing the structure of the STCKA model, the training, and testing algorithms, and then finally conducting an experiment to replace all the linear networks to convolutional networks and examine its effect on predicting the classes of the short text. The results of this experimentation suggests that the linear model is more accurate whereas the convolutional model is more precise and has a better F1-score.

1. Reviews

1.1. Deep Pyramid Convolutional Neural Networks for Text Categorization

1.1.1. Introduction

This paper proposes an alternative method to shallow word-level convolutional neural network (CNN) and deep character-level CNN, to process high volume of documents without a significant increase to the computational complexity as the network goes deeper for accurate text categorization. In other words, the approach introduced in this paper is a low-level complexity convolutional neural network called *Deep Pyramid CNN* or DPCNN. (Johnson & Zhang, 2017)

CNN is used in place of Recurrent Neural Networks (RNN) because although both networks take advantage of word ordering, CNNs can process word orderings in parallel whereas RNN does it sequentially. As a result, CNN is the approach in terms of maintaining a low-complexity network while processing large amounts of data. (Johnson & Zhang, 2017)

1.1.2. Features

The key features that make the DPCNN architecture, a low-complexity text categorization model are the following features:

1. Ability to do downsampling without increasing the number of feature maps. As a result the model reduces the computation time for each convolutional layer by a factor of two. This exponential decrease from layer to layer makes the network a pyramid model. Moreover, downsampling in DCPNN doubles the coverage of the input document (Johnson & Zhang, 2017).
2. Pre-activation to train the model. The pre-activation involves the use of a linear weighting which eases the training of deep networks and has better rates than post-activation (Johnson & Zhang, 2017)
3. Enhancing the text region embeddings with unsupervised embeddings for better accuracy (Johnson & Zhang, 2017)

1.1.3. Experimentation

The experiment to test the validity of the Deep Pyramid Convolutional Neural Network involved gathering a number of training and testing documents from various repositories. The document size could be variable for this experiment, but the size of the network's vocabulary was limited to 30000 words. 10000 documents from the training set will make up the validation set. The parameters, such as the learning rate and regularization parameter, will be determined based on the performance against the validation set. The model is then trained using the training set and tested against the testing set to obtain the error rates and computation time. The error rates and computation times were compared to other models (i.e. shallow CNN with unsupervised embedding, hierarchial attention network, and etc.) (Johnson & Zhang, 2017).

The results of the experimentation demonstrate that the DPCNN method has better accuracy and computation time than the other models it was compared against with.

1.1.4. Final Thoughts

This paper has its strengths and weaknesses. The strengths of this paper are the following:

- Right at the beginning of the paper, the motivation for developing the DPCNN network model is explained. Their motivation was to deepen the current shallow word-level CNN to efficiently process large inputs of data is by deepening it, without increasing the number of feature maps and computation time
- The paper also refers back to the motivation by proving that their DPCNN indeed outperforms other types of CNN (i.e. character-level deep CNN, word-level deep CNN) in terms of accuracy and computation time by using the same training and testing datasets for all the models
- The paper does a good job in explaining why they chose a convolutional neural network approach instead of a recurrent neural network approach. Not many papers justify their chosen neural network model type so this was one of the paper's strengths

This paper also has some weaknesses which are as follows:

- The explanation for what pooling is and how that contributes to downsampling, and the math behind the pre-activation function is not well explained. The authors of this paper have made the assumption that the readers would have read the related readings before reading this paper. This makes it difficult for readers such as myself who are just starting out and have no idea what is meant by shortcut connections with pre-activation for instance
- The paper makes multiple references to other papers to indicate that they are basing or improving upon ideas from these referenced papers. One instance of this is when the paper refers to another paper written by (He et al., 2016) about skipped layers but they didn't elaborate further by defining what this concept meant.

In addition to the strengths and weaknesses of this paper, I believe that the proposed DPCNN model, although it shows promise for large text samples, might not do so well with short text samples. This is because, for the entirety of the paper, the authors elaborate on not increasing the computation time for large samples of text. As a result, this could be a good question to ask the authors of the paper: “Whether the DPCNN model will still bring about the same level of accuracy and efficient computation time for ambiguous, less informative short text samples”.

1.2. Deep Short Text Classification with Knowledge Powered Attention

1.2.1. Introduction

This paper proposes a method to execute text classification on short text samples which are more ambiguous than paragraphs or document of texts. Current neural networks are capable of analyzing and classifying long text samples but for short text samples there are some complications that need to be taken care of. For this reason, the authors of this paper propose Deep Short Text Classification with Knowledge Powered Attention to take care of these complications. The type of attention used is a Concept towards Short Text (C-ST) attention and Concept towards Concept Set (C-CS) attention to obtain the weightings (Chen et al., 2019).

1.2.2. Features

Current short text classification networks either implement an explicit representation or implicit representation. Explicit representation consists of information regarding n-grams, part of speech (POS) tagging, and syntactic parsing. The explicit model, although is human readable, ignores the context of short text samples and will be incapable of capturing deep semantic information. On the other hand, implicit representations can obtain syntax and semantic information but it lacks in capturing semantic relationships such as *isA* and *isPropertyOf*. As a result, the authors of this paper have decided to integrate both representations into a ”unified deep neural network model” (Chen et al., 2019). The model will obtain knowledge relevant for classification from external Knowledge Bases (KB) to conceptualize semantic relationships and associate each short text to its concepts. (Chen et al., 2019)

Since short texts are ambiguous, not much context is provided, there could be some overlap between classification labels for the same word in the short text. An example, provided by the paper is as follows: “Alice has been using Apple for more than 10 years” (Chen et al., 2019) where Apple will be classified by the concept of fruit and the concept of mobile phone from the Knowledge Base. In addition to this, it is also necessary to take into account of the granularity of the concepts (i.e. entrepreneur vs person) and assign a larger weighting to the granular concept. Both these complexities are addressed by the STCKA. The paper also indicates the use of C-ST and C-CS attention to asses the importance of each concept and combine them via a method of soft switch to obtain the weightings (Chen et al., 2019).

1.2.3. Experimentation

To verify the validity of the STCKA model, an experiment was conducted over four different datasets and the results demonstrated that this model outperformed all the previous CNN models, including that of the CNN model introduced by (Kim, 2014) in classifying short text samples (Chen et al., 2019).

1.2.4. Final Thoughts

This paper has its strengths and weaknesses. The strengths of this paper are the following:

- The paper does a good job explaining the necessity of having a model to accurately classify short texts. It also explains why the current models are not good enough to accomplish this and why classifying short texts is more complex. In other words, the paper explains their problem statement well

- The paper also defines concepts like explicit and implicit representation for instance and explains how such concepts will be integrated into their model to effectively classify short-text samples

- The paper also explains well the mathematical concepts as well as elaborating on what each variable represents

This paper also has some weaknesses which are as follows:

- There is a paragraph in the paper where the authors mention that they will be implementing a bidirectional LSTM similar to another paper they referenced, but they do not go deeper than that. This means, if the readers are to understand this concept, they need to also refer the same paper

- The paper also does not justify why they use a Convolutional Neural Network for the input embedding stage of the model

A question to ask the authors of this paper would be about the computation complexity of their STCKA model as this is not mentioned in the paper.

1.3. Initializing Convolutional Filters with Semantic Features for Text Classification

1.3.1. Introduction

In current convolutional neural network models, embedding layers are initialized by pre-trained word vectors. However, the remaining layers of the network are still randomly initialized. This is a concern because the inappropriate parameters limit the generalization abilities of the networks. This paper addresses this problem by incorporating semantic features into the filters. The method involves helping the filters learning useful n-grams and these n-grams are determined via a combination of Naive Bayes weighting technique and K-means clustering. The centroid of the cluster is used to intialize the filters. (Li et al., 2017)

1.3.2. Features

With this type of procedure, the important n-gram features are extracted at the beginning of the training process. It is advantageous if the CNN model introduced by (Kim, 2014) was combined with the proposed intialization method for three reasons:

1. The semantic features are obtained from only the training data. There is no involvement of external resources (Li et al., 2017)
2. The computation cost is minimal (Li et al., 2017)
3. The intialization method is independent of NLP tasks - it can be applied to any type of NLP task (Li et al., 2017)

1.3.3. Experimentation

The experiment consists of training and testing against seven different datasets and the results are compared to previous CNN models. To intialize the filters, uni, bi, and tri-gram filters were used. The results of the experiments demonstrate the effectiveness of initializing the filters using semantic features. The experiment also demonstrates that this approach has lower computation complexities as well. (Li et al., 2017).

1.3.4. Final Thoughts

This paper has its strengths and weaknesses. The strength of this paper are the following:

- The paper explains its motivation well by providing some background information and highlighting the issues pertaining to the current text classification models. Their motivation was to eliminate the process of randomizing the weights for some of the layers in the classical CNN model

- The paper provided justification as to why they wanted to implement a Naive Bayes and KNN clustering techniques to acquire the weightings, instead of just making a statement

The paper also has some weaknesses such as making references to several other papers at the beginning to indicate the numerous attempts made to prevent inappropriate parameter settings for CNN (i.e. developing advanced structures, using prior knowledge, using external resources, etc.) but the authors do not elaborate why they believe their semantic features method would outperform these other methods

Furthermore, it would be interesting to evaluate the performance of this method against short text samples. Since this model is dependent on getting valuable n-grams from its training sample, having ambiguity in its input might not lead to an accurate classification. Therefore, this would be a question to ask the authors.

2. Method

2.1. Problem Description

Text classification in general is successful because the computer can understand the context behind the text it is classifying. It is understandably easier to figure out the context behind long paragraphs but how about short sentences? For example, "Jay grew up in Japan". The current models will treat Jay as a new word and will not understand that maybe Jay is a singer for example. Therefore, the context here is a bit more ambiguous for a program to understand so how do we expect the current text classification algorithms to classify short sentences? This is what Deep STCKA architecture attempts to solve.

2.2. Description of the Architecture

2.2.1. Retrieval of Knowledge

This paper examines the re-implementation and slight extension of Deep Short Text Classification with Knowledge Powered Attention, henceforth referred to as the STCKA model. Before diving into the experimentation conducted, it is vital to have a good understanding of the algorithm used to develop this model. The structure of the STCKA model comprises of several parts. The first part involves retrieving relevant knowledge from Knowledge Bases (KB). The knowledge obtained from KB is the semantic relation isA property for the short sentence passed in as an input. In other words, given a short sentence s, this step will return a concept set C relevant to the sentence. This is accomplished by identifying the entities ϵ within s, via the use of an external database called YAGO, and then passing ϵ to another database like CN-Probase to get C. This is a vital step because prior knowledge obtained from databases help decide the class of the short text input. Despite having this as a step, the authors of the paper did not include this procedure in their code as the concepts of each short sentence was hard-coded in their tsv files (Chen et al., 2019).

2.2.2. Input Embedding

In addition, the inputs s and C needed to undergo embedding for the purpose of mapping each word and character in s and C to a high-dimensional vector space. For this paper, the inputs are mapped to pre-trained word vectors from Stanford University's Global Vectors for Word Representation (GloVe) website.

2.2.3. Self-Attention Mechanism

A Bidirectional LSTM was employed to obtain the short text sequence representation q from s because recurrent neural networks (RNN) are good at capturing contextual information of the sequence, thus increasing the "expressive power" of the attention networks. The output of the LSTM will be of dimension $(N, L, D \times H_{out})$ where N is the batch-size, L is the sequence length, D is 2 because it is bidirectional, and H_{out} is the hidden dimension. The output of the BiLSTM is then processed via a scaled dot-product attention formula to learn the word dependence within the sentence and capture its internal structure. The scaled dot-product attention formula is as follows:

$$A = Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{2u}})V \quad (1)$$

where Q, K, V are equal to the output of the Bidirectional LSTM, u is the hidden dimension. Finally, a max-pooling layer is applied in order to choose the highest value on each dimension of vector to capture the most important feature (Chen et al., 2019).

2.2.4. Concept towards Short Text Attention

To reduce the effect of anomalous concepts found in C which would have been introduced as a result of ambiguity in entities or noise in KB, Concept towards Short Text (CST) attention will be used. This attention algorithm measures the semantic similarity between a concept $c \in C$ and q. The formula used to compute CST is as follows:

$$\alpha_i = softmax(w_1^T f(W_1[c_i; q] + b_1)) \quad (2)$$

where α_i is the weight of the attention for the i^{th} concept towards q. The larger the α_i weight, the more semantically similar the i^{th} concept is to q. The function $f()$ is the hyperbolic tangential activation function. $W_1 \in R^{d_a \times (2u+d)}$ is a weight matrix where d_a is the hidden dimension and d is the embedding dimension. $w_1 \in R_a^d$ is a weight vector. b_1 is an offset. Finally, the softmax function is applied to normalize the attention weights for each $c \in C$ (Chen et al., 2019).

2.2.5. Concept towards Concept Set Attention

In addition to CST attention, Concept towards Concept Set (CCS) attention is utilized to take the relative importance of the concepts with respect to the whole concept set. CCS is computed as follows:

$$\beta_i = softmax(w_2^T f(W_2 c_i) + b_2) \tag{3}$$

where β_i is the weight of the attention for the i^{th} concept towards C. The larger the weight, the more important c is. $W_2 \in R^{d_b \times d}$ is a weight matrix and $w_2 \in R^{d_b}$ is a weight vector where $d_b = \frac{d_a}{2}$. b_2 is the offset, and the function $f()$ is the hyperbolic tangential activation function (Chen et al., 2019).

2.2.6. Weighted sum of Concept Vectors

Both α_i and β_i are combined via the following equation:

$$a_i = softmax((\gamma \times \alpha) + (1 - \gamma) \times \beta) \tag{4}$$

where γ is computed as follows:

$$\sigma(w^T[\alpha; \beta] + b) \tag{5}$$

σ is the sigmoid activation function and w and b are parameters (Chen et al., 2019). Despite giving this equation, the authors of this paper made γ a user argument.

All of this leads to a weighted sum of the concept vectors:

$$p = \sum_{i=1}^{m} a_i c_i \tag{6}$$

2.2.7. Final Output Layer

The final part of this structure involves concatenating q and p and is passed into a CNN such that the output dimension is of batch-size and output size.

The output of the entire model is going to be a probability of the class y given short text S, with parameters $\phi : p(y \mid s, \phi)$.

2.3. Training Algorithm

Training the algorithm involves maximizing the log-likelihood with respect to ϕ:

$$\phi \to \sum_{s \in S} \log p(y \mid s, \phi) \tag{7}$$

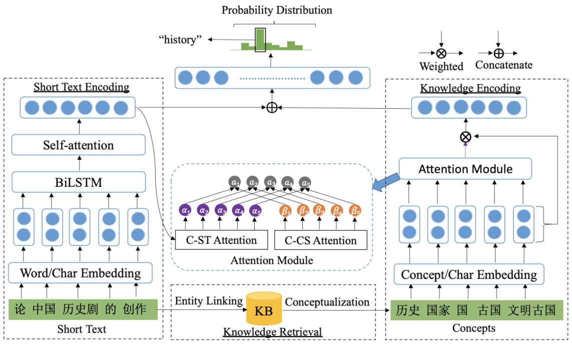


Figure 1. The Structure of STCKA (Chen et al., 2019)

2.4. Implementation

The experimentation began with re-implementing the entirety of the STCKA structure and the training and testing algorithms, then assessing the accuracy of the model on the datasets provided by the author's GitHub (AIRobotZhang, 2019) repository. The re-implementation of the algorithm was accomplished by first reading and understanding each part of the STCKA structure and also examining the diagram illustrated in *Figure 1* to figure out how all the features fit together.

In re-implementing the STCKA structure, the first step was defining all the networks (i.e. embedding, linear, and BiLSTM). When reading about conducting character, word, and concept level embedding on the inputs s and C, we did not possess knowledge on what is meant by embedding so this led to researching what this network meant. According to the pytorch official documentation page, embedding is a "simple lookup table that stores embeddings of a fixed dictionary and size". (Torch-Contributors, 2019). Which means, the embedding network is for mapping the inputs to the pretrained word vectors. The input parameters for the embedding network are size of the dictionary embeddings, which is obtained from the size of the input (either s or C) vocabulary, and the size of each embedding vector which is provided by the user when running the model. Once, the class variable for embedding was defined for both s and C, we confirmed with the author's code and saw that they also appended input embedding weights to the embedding's parameters, after which we included that segment into our code as well.

Once the embedding networks were defined, the next step was figuring out how to apply BiLSTM on the embedded version of s. Because, we did not posses prior knowledge on how BiLSTM is implemented in code, we re-visited the author's implementation to see how they achieved this. BiLSTM is achieved by a simple one line of code provided by PyTorch, where the parameters are the embedding dimensions for the input dimension and the hidden dimension (provided by user arguments), and bidirectional is set to true because we want the BiLSTM effect. The input dimension is made the embedding dimension because the output of the embedding layer for s has an input size of the embedding dimension. The output of the BiLSTM layer is q.

After the BiLSTM, q is passed into a self-attention algorithm. This was implemented by creating a function within the STCKA structure dedicated for the self-attention mechanism. Q, K, V are set to the output of the BiLSTM layer. The computations in *Equation 1* are conducted in this function and the max-pooled output of the computations is returned. When doing the matrix multiplications, we used torch's in-built bmm function because a lot of permutations were required to make the multiplication valid.

The embedded output of concept set C, has to go through both the CST and CCS attention algorithms. Both CST and CCS have their own dedicated functions in the STCKA class. The CST and CCS functions apply *Equation 2* and *Equation 3* respectively. In the author's implementation, the W_x and w_x where $x \in \{1, 2\}$ were linear neural networks.

And then finally, *Equation 4* (with γ being a user argument) and *Equation 6* were computed in the forward function using the outputs from the previous steps. The result from *Equation 6* was concatenated with q. After finishing this part, based on the diagram, we sincerely believed that the concatenation was the final output of the STCKA model. However, when confirming with the author's code, we came to understand that the authors of the paper passed the concatenation output through the a final linear layer and the output of that layer is the output of the STCKA model.

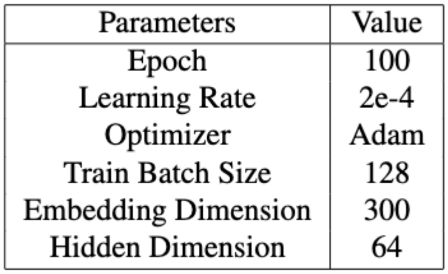
The training algorithm was implemented with the loss function being the CrossEntropyLoss because this function is used to compute the log-likelihood and because it is appropriate for when the output of the model is the probability that the input belongs to a certain class. For the testing algorithm, the only thing that sets the re-implementation apart from the author's implementation is the fact that all the predicted labels and target labels are appended to a list during the loop and the final list was concatenated using the PyTorch built-in function whereas the authors decided to concatenate throughout the loop. The reason for this concatenation was to use Sci-kit learns built in functions to compute the accuracy, precision, recall, and f1-score. Regardless the same objective was achieved.

In other words, throughout the entire re-implementation process, we first read and tried to envision the code structure from the paper, and then made the attempt to write each part of the structure. We then confirmed our version to the author's implementation and if it was slightly different or if we were not too sure, then we would try to understand what they tried to do and then make the necessary adjustments.

Recently, many text classification algorithms are utilizing convolutional neural networks because CNNs are "good at extracting local and position-invariant features" (Ghelani, 2019). As a result, we got curious as to why the authors implemented linear networks throughout the model instead of CNNs when CNNs are proven to be better at text classification. Therefore, we conducted a mini experiment where we converted all linear networks to CNNs while maintaining the same dimensions. This involved several permutations throughout the attention mechanism computations, setting the kernel size to 3, and stride and padding to 1. The objective of this experiment was to see how a CNN would compare to a linear network in terms of runtime, test loss, test accuracy, test precision, test recall, and test f1-score.

3. Experimentation

After re-implementing the STCKA struture, training and testing algorithm, and changing all the fully connected layers to convolutional layers, all that remains is running the program. Figure 2 shows the parameters and user arguments used in the experimentation.



Parameters	Value
Epoch	100
Learning Rate	2e-4
Optimizer	Adam
Train Batch Size	128
Embedding Dimension	300
Hidden Dimension	64

Figure 2. The User Arguments and Parameters

4. Results

The results of the experiment are shown in *Table 1* and *Table 2*. The experiment was conducted over five trials and the average over the five trials is computed for each attribute: runtime (measured in minutes), the testing loss, testing accuracy, testing precision, testing recall, and testing F1-score. The number of epochs for each trial was 100 epochs.

5. Evaluation

From examining *Table 1*, we can see that the average runtime, something the authors of the paper did not mention, is approximately 40 minutes for a 100 epochs. In addition to this, the purpose of this paper is to re-implement the author's experimentation and compare the re-implementation results and the author's results. Despite re-implementing the author's code, the accuracy of the re-implementation is significantly better than the author's implementation. This is probably because the author's do not mention the number of epochs they run it for in the paper and in the re-implementation, 100 epochs were chosen because it was a round number that obtained an accuracy of at least 90%. This implies that at some point before reaching 100 epochs, the model would've had a similar accuracy to that of the author's results.

As for the experiment conducted to compare the results of the CNN version of STCKA and Linear version of STCKA, the linear STCKA re-implementation outperformed the CNN version in terms of runtime, loss, accuracy, and recall. However, the CNN version outperformed the linear version in terms of precision and F1-score. The runtime being slower for CNN in comparison to linear makes sense because the fully connected linear network only has to perform an overall matrix multiplication whereas a Conv1D layer with a kernel size 3 would mean that the kernel will have to span across the entire tensor, make the necessary computations to get one cell of the output tensor iteratively, which takes time.

It is understandable why precision and F1-score for the CNN STCKA model is relatively better than the fully connected linear model. First and foremost, CNNs have more parameters than linear networks which means it has the capability of learning minute details better, thus enabling it to efficiently classify text accurately out of the predicted positive. The F1-score is calculated from both the precision and recall scores. Since the CNN version outperforms the linear version in terms of precision, and both versions have relatively similar recall scores, the CNN-STCKA model outperforms the linear version with respect to the F1-score. The linear model has better accuracy because it is the overall matrix multiplication, in other words the model has a grasp of the main idea, whereas the CNN would have a window studying each and every aspect of the text.

6. Discussion

Expanding on the re-implementation conducted, an extension to this paper is to include a function that would retrieve the entity-linkings, from the YAGO databse, and the concept set C, from the CN-Probase database. This way, there would not be a need to hard-code in the concept sets in the tsv data files. In addition, another extension could be to initialize convolutional filters with semantic features for text classification like (Li et al., 2017) suggests and see what effect it has on the accuracy and precision of predicting the classes of short sentences. Furthermore, another extension could be to perhaps integrate the Deep Pyramid Convolutional Neural Networks for Text Classification like (Johnson & Zhang, 2017) suggest in their paper and examine if the runtime of the CNN version of STCKA would decrease.

Table 1. Linear version of STCKA Re-implementation

Trial Number	Runtime (m)	Loss	Accuracy	Precision	Recall	F1-Score
1	45.39	0.3434	0.9243	0.9261	0.9190	0.9221
2	42.49	0.2730	0.9300	0.9263	0.9267	0.9263
3	39.20	0.3289	0.9211	0.9119	0.9215	0.9164
4	35.68	0.2970	0.9304	0.9151	0.9138	0.9143
5	36.14	0.3167	0.9259	0.9267	0.9161	0.9211
Average	39.78	0.3118	0.9263	0.9212	0.9194	0.9200

Table 2. CNN version of STCKA Re-implementation

Trial Number	Runtime (m)	Loss	Accuracy	Precision	Recall	F1-Score
1	54.89	0.3344	0.9243	0.9216	0.9221	0.9215
2	85.36	0.3164	0.9295	0.9277	0.9239	0.9257
3	48.42	0.4161	0.9191	0.9197	0.9083	0.9135
4	56.86	0.2503	0.9308	0.9299	0.9162	0.9222
5	45.52	0.3001	0.9239	0.9171	0.9188	0.9178
Average	58.21	0.3235	0.9255	0.9232	0.9179	0.9201

References

AIRobotZhang. Stcka. https://github.com/AIRobotZhang/STCKA, 2019.

Chen, J., Hu, Y., Liu, J., Xiao, Y., and Jiang, H. Deep short text classification with knowledge powered attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6252–6259, Jul. 2019. doi: 10.1609/aaai.v33i01.33016252. URL https://ojs.aaai.org/index.php/AAAI/article/view/4585.

Ghelani, S. Text classification — rnn's or cnn's? https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

Johnson, R. and Zhang, T. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017. URL https://aclanthology.org/P17-1052.

Kim, Y. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://aclanthology.org/D14-1181.

Li, S., Zhao, Z., Liu, T., Hu, R., and Du, X. Initializing convolutional filters with semantic features for text classification. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1884–1889, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1201. URL https://aclanthology.org/D17-1201.

Torch-Contributors. Embedding. https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html, 2019.