A Comparative Study on Word Embeddings in Deep Learning for Text Classification

Congcong Wang
School of Computer Science,
University College Dublin
congcong.wang@ucdconnect.ie

Paul Nulty
School of Computer Science,
University College Dublin
paul.nulty@ucd.ie

David Lillis
School of Computer Science,
University College Dublin
david.lillis@ucd.ie

ABSTRACT

Word embeddings act as an important component of deep models for providing input features in downstream language tasks, such as sequence labelling and text classification. In the last decade, a substantial number of word embedding methods have been proposed for this purpose, mainly falling into the categories of classic and context-based word embeddings. In this paper, we conduct controlled experiments to systematically examine both classic and contextualised word embeddings for the purposes of text classification. To encode a sequence from word representations, we apply two encoders, namely CNN and BiLSTM, in the downstream network architecture. To study the impact of word embeddings on different datasets, we select four benchmarking classification datasets with varying average sample length, comprising both single-label and multi-label classification tasks. The evaluation results with confidence intervals indicate that CNN as the downstream encoder outperforms BiLSTM in most situations, especially for document context-insensitive datasets. This study recommends choosing CNN over BiLSTM for document classification datasets where the context in sequence is not as indicative of class membership as sentence datasets. For word embeddings, concatenation of multiple classic embeddings or increasing their size does not lead to a statistically significant difference in performance despite a slight improvement in some cases. For context-based embeddings, we studied both ELMo and BERT. The results show that BERT overall outperforms ELMo, especially for long document datasets. Compared with classic embeddings, both achieve an improved performance for short datasets while the improvement is not observed in longer datasets.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**.

KEYWORDS

Word Embeddings, Neural Networks, Text Classification

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.
NLPIR 2020, December 18–20, 2020, Seoul, Republic of Korea
© 2020 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-7760-7/20/06.
DOI: https://doi.org/10.1145/3443279.3443304

1 INTRODUCTION

Word embedding is a way to represent a word with fixed-length vectors of continuous real numbers. It maps a word in a vocabulary to a latent vector space where words with similar contexts are in proximity. Through word embedding, a word is converted to a vector that summarises both the word's syntactic and semantic information. Consequently, word embeddings are considered to be particularly suitable to be used as feature representations in neural network models for downstream natural language processing (NLP) tasks, such as text classification [13, 35–37], machine translation [20], sequence learning [2, 24] etc.

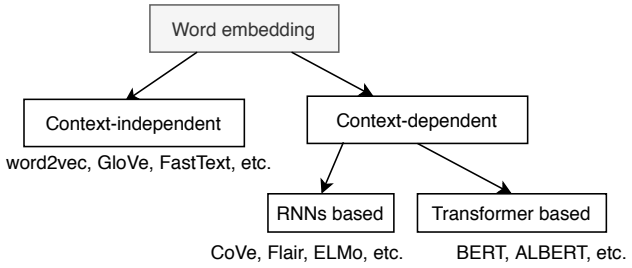

Figure 1: A taxonomy of word embeddings

Figure 1 presents a taxonomy of word embeddings. Broadly speaking, there are two main types of word embeddings that have been developed, namely context-independent and context-dependent embeddings. Context-independent methods are known as "classic" word embeddings, which learn representations through language model (LM) based shallow neural networks or co-occurrence matrix factorisation [34]. The learned representations are characterised by being unique and distinct for each word without considering the word's context. Hence, these are usually pre-trained on general text corpora and are distributed in the form of downloadable files, which may be directly applied to initialise the embedding weights for downstream language tasks. Prominent examples include word2vec [21], GloVe [23] and FastText [6].

In contrast with context-independent word embeddings, context-dependent methods learn different embeddings for the same word that is dependent on the context in which it is used. For example, the polysemy "bank", will have multiple embeddings depending on whether it is used in a river-related context or finance-related one. This feature has brought context-dependent embeddings into the mainstream in recent times. As research has progressed, the learning methods for context-dependent word embedding have

mainly evolved in two categories. One category consists of those approaches based on LM-based Recurrent Neural Networks (RNNs), such as CoVe [20], Flair [2] and ELMo [24]. Alternatively, the recently developed Transformer-based models [28], such as BERT [7] and ALBERT [17], have been demonstrated to learn efficient contextualised word representations. Typically, there are two ways in which contextualised embeddings can be leveraged for downstream tasks. Firstly, they can be used as fixed feature extractors (not trainable along with the downstream parameters) to offer static pre-trained input features of a sequence. Secondly, the pre-trained embeddings are adjusted on a specific downstream dataset for providing input features of a sequence, in a process called "fine-tuning". Through fine-tuning, the Transformer-based models used to represent a sequence have achieved strong state-of-the-art performance for many downstream NLP tasks [7, 17, 33]. Although fixed extractors do not perform quite as well as those that are fine-tuned, they do offer an advantage to the downstream task in that they do not require parameter training and consequently consume less time and memory.

Although many different word embedding methods have been developed, to the best of our knowledge few studies have been undertaken to systematically compare both classic and contextual embeddings in specific downstream language tasks. Our work constrains the specific language task to text classification. We empirically evaluate a wide range of pre-trained embeddings, from the classic word embeddings to more recent contextualised embeddings (used as fixed extractors for efficiency concerns) and report their effect on the performance in different classification domains.

For comparison purposes, each set of pre-trained embeddings is used within our downstream network architecture for a range of document classification tasks. Within this architecture, an encoder is needed for learning a sequence representation from word representations. Each of the pre-trained embeddings is evaluated with both a simple convolutional neural network (CNN) encoder and a bi-directional LSTM (BiLSTM) encoder.

The primary purpose of this work is not to compete with complex state-of-the-art methods. Instead, we take the resource needs of the model into account to present a methodology for selecting suitable low-resource configurations for text classification experiments. The primary contributions of our work are summarised as follows:

- Our work provides insights for choosing word embeddings in different text classification domains using CNN or BiLSTM as the downstream encoders. The experimental methodology is transferable and applicable to other downstream architectures and language tasks.
- We conduct an empirical survey on word embeddings. The survey is considered to be instructive for benchmarking analysis of word embeddings in neural network based text classification.
- We produce a framework that can be used for efficient word embedding selection in neural network models given different text classification datasets (single-label or multi-label).

2 RELATED WORK

This paper mainly focuses on the evaluation of word embeddings. This section introduces related work from methods of evaluating word embeddings, existing studies of evaluating embeddings in downstream tasks.

Word embeddings evaluation is broadly divided into two categories in the literature, namely, intrinsic and extrinsic evaluation [27, 29]. The intrinsic method evaluates word embeddings by examining their effectiveness in providing both syntactic and semantic representations for words. The most common specific tasks for this type of evaluation include word relatedness/similarity or analogy matching [4, 9, 27]. However, our work is more closely related to extrinsic evaluation. It evaluates word embeddings by leveraging them to provide input features in neural-network-based downstream language tasks, such as text classification. To test their effect effectively, one crucial factor in comparison experiments is to keep the embedding layer as the only variant but with others as invariants. This motivates the aims of our work.

Regarding the studies on extrinsic evaluation, Hawani et al., [27] examined some classic word embeddings such as word2vec, Glove, FastText in multiple downstream tasks, including sentiment analysis, text inference, etc. Their results show that random initialisation of the embedding layer is trainable to achieve equivalent performance as initialising the embedding layer with the pre-trained classic embeddings. Naili et al., [22] investigated Latent Semantic Analysis [10], word2vec and GloVe in the task of topic segmentation (TS) with cross-language datasets (Arabic and English). In particular, they examine the word2vec model in depth and explore its impact on TS with different ways of training it. They conclude that word2vec hits a good performance with a careful decision on the training algorithms according to the characteristics of a language-specific dataset. Similar to the comparative study of word embeddings for a downstream task, Dhingra et al., [8] studied the impact that pre-trained word embeddings (word2vec and GloVe) have on reading comprehension (RC) using datasets from different domains. Their work shows that the choice of using pre-trianed embeddings as feature representations largely impact the performance of RC. Kaibi et al., [14] comparatively evaluated three classic word embeddings (Word2vec, Fasttext and Glove) for Twitter sentiment analysis with a number of machine learning algorithms, which concludes FastText combined with a SVM classifier generally outperforms other combinations.

Our work exhibits similarities with the aforementioned studies in one aspect or another. We consider three dimensions in our extrinsic word embeddings evaluation. First, we investigate classic embeddings as well as comparing them to contextualised embeddings in parallel. Second, we select the classification datasets from different domains to explore how differently the downstream model architectures CNN and BiLSTM perform based on dataset characteristics. Last, as inspired by the confidence testing for intrinsic evaluation [27], we report confidence intervals for accuracy as references to evaluate embeddings strictly.

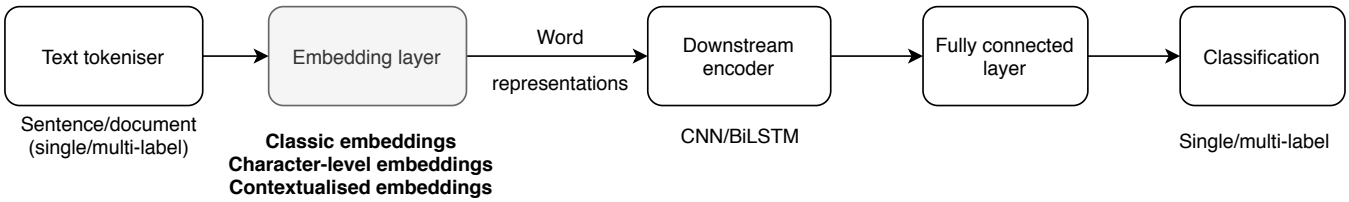

Figure 2: The architecture of leveraging various pre-trained word embeddings for text classification in neural network models.

3 METHODOLOGY

To systematically study word embeddings in neural-network-based downstream models for text classification. Figure 2 presents the overview of our methodology.

First, the text tokeniser splits a text sequence into tokens before it is sent to the embedding layer. To explore how word embeddings have an impact on different datasets, our selected datasets tend to be different in both length and characteristics. Some of the datasets consist primarily of short sentences and some of long documents. Additionally, they are either single-label or multi-label datasets (see Section 4.1 for a detailed description of the datasets). This variety of datasets suits our needs to study the impact of word embeddings on different classification domains.

Having prepared the dataset, next assume a training example s that consists of a sequence of n words: $\langle w_0, w_1, w_2, ..., w_t, ...w_n \rangle$. The embedding layer converts the words with pre-trained word embeddings to word vectors: $\langle x_0, x_1, x_2, ..., x_t, ...x_n \rangle$. In our study, we consider three categories of word embeddings as follows:

Classic embeddings: This type of embedings are pre-trained over very large corpora and shown to capture latent syntactic and semantic features. The following gives a list of popular classic word embeddings.

- **word2vec** [21]. This is a well-known classic word embedding method that applies either of two model architectures to produce word vectors: continuous bag-of-words (CBOW) or skip-gram (SG). Both methods are trained based on a neural prediction-based model. CBOW trains a model that aims to predict a word given its context, while SG does the inverse, to predict the context given the centre word in it.
- **GloVe** [23] is a classic word embedding method that learns efficient word representations by performing training on aggregated global word-word co-occurrence statistics from a corpus. Due to this feature, it is advantageous at capturing language features globally by analysing words' co-occurrences across corpora.
- **FastText** [6] is another classic word embedding method that learns word representations through a neural language model. Unlike GloVe, it embeds words by treating each word as being composed of character n-grams instead of a word whole. This feature enables it not only to learn rare words but also out-of-vocabulary words.

Character-level embedding (Char) [16, 35] is commonly used to concatenate with classic word embeddings for providing sub-word features. The character-level embedding for a word is usually output by a CNN or RNN encoder, which is fed with the word's character constitutes as the input. Hence it is not pre-trained but instead is trained along with the downstream model training on task data.

Contextualised embeddings: are known for capturing word semantics in context, in contrast with other embeddings. Below are a list of recently-developed contextualised embeddings that have become mainstream for language tasks.

- **ELMo** [24] is a context-based word embedding method. It learns contextualised word representations based on a neural language model with a character-based encoding layer and two BiLSTM layers. The character-based layer encodes a sequence of characters of a word into the word's representation for the subsequent two BiLSTM layers that leverage hidden states to generate the word's final embedding.
- **BERT** [7] is a relatively new transformer-based language representation model trained on a large cross-domain corpus. Unlike ELMo, which pre-trains representations through bidirectional language models (i.e., simply the combination of left-to-right and right-to-left representations), BERT applies a masked language model to predict words that are randomly masked in a sequence, and this is followed by a next-sentence-prediction task for learning the associations between sentences. Through a process of fine-tuning, BERT has achieved state-of-the art results for a range of NLP tasks. Since it was introduced, many follow-up variants like data enhanced RoBERTa [19] or model optimised Albert [17] have emerged to further advance the state of the art in language representation learning.

As described, each embedding method has its distinctive characteristics, and it is important to know what feature vectors will be offered by either using them individually or in combination. In the workflow for text classification, the embedding layer applies one or more pre-trained embeddings to generate word representations for the downstream encoder, denoted by e. In our method, we apply either CNN or BiLSTM to encode the embedded vectors $\langle x_0, x_1, x_2, ..., x_t, ...x_n \rangle$ into a summary of single sequence representation: i.e. $\mathbf{o} = e(x_0, x_1, ..., x_n)$. After being encoded, $\mathbf{o}$ is forwarded to a fully connected layer denoted by f to output the logits cross all labels: $\mathbf{g} = f(\mathbf{o})$. For single-label classification, the probability of sample s belonging to label l_i, namely $p(l_i|s)$ is estimated by the

softmax function

$$p\left(l_i|\, s\right) = \frac{exp(\mathbf{g_i})}{\sum_{j=0}^{T} exp(\mathbf{g_j})}$$

where T refers to the number of labels. For multi-label classification, the probability is estimated by the sigmoid function

$$p\left(l_i|\, s\right) = sigmoid(\mathbf{g_i})$$

where the label l_i is predicted for the training example s if the estimated probability is larger than 0.5.

4 EXPERIMENT SETUP

The experiment setup includes the selection of datasets, configuration of downstream models, and choice of pre-trained embeddings. The details of these components are given below.

4.1 Datasets

We select four popular benchmarking classification dataset. These datasets vary in size, number of classes, types (single-label or multi-label), and categories (topic or sentiment). Table 1 presents a summary of the datasets and Figure 3 plots sample length distribution for the four datasets.

- The 20NewsGroup [18] dataset consists of around 18,000 newsgroups posts on 20 topics. The dataset is originally split into a training set and test set based on messages posted before and after a specific date. In our experiment, the train set is further split randomly into training and development sets at a ratio of 4:1.
- The Stanford Sentiment Treebank dataset (SST) [25]) contains phrases with fine-grained sentiment labels in movie reviews. This dataset provides standard 5-way (SST-5) or binary (SST-2) classification splits for model training, development and evaluation. We use the standard binary classification splits SST-2 in our experiment.
- arXiv Academic Paper dataset (AAPD) [32] is a multi-label dataset, which comprises paper abstracts extracted in the field of computer science from arXiv[1]. Each document in AAPD is labeled with one or more subjects across 54 subjects in total. In our experiment, we use the train, development, and test splits used in [1].
- Reuters-21578 (Reuters) [3] is another benchmarking multi-label dataset for document classification, which consists of news content extracted from the Reuters newswire in 1987.

[1] https://arxiv.org/

Table 1: Summary of the dataset characteristics.

Name	# Train/dev/test	# Classes	Type	Category	Words/Sample
20NewsGroup	9051 / 2263 / 7532	20	single	Topic	221.26
SST-2	67349 / 872 / 1821	2	single	Sentiment	9.79
AAPD	53840 / 1000 / 1000	54	multi	Topic	167.3
Reuters	5827 / 1943 / 3019	90	multi	Topic	144.3

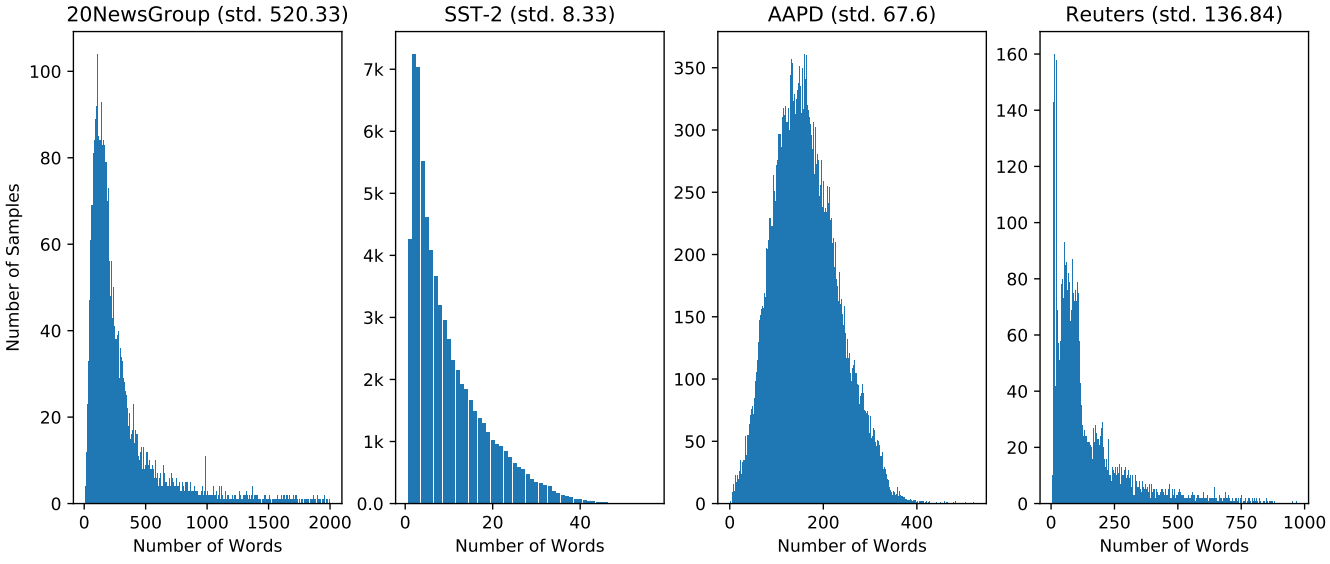

Figure 3: The number of words (sample length) versus the number of samples (sample frequency) of the datasets - 20NewsGroup, SST-2, AAPD and Reuters. For better visualisation, only the samples with number of words less than 2000 and 1000 for 20NewsGroup and Reuters respectively is displayed. std. denotes standard deviation where we can see 20NewsGroup comprises samples with the most varying length among the four datasets.

In this dataset, each sample is labeled with one or more news categories out of 90 in total. We use the standard ModApte splits [3] for our experiments.

4.2 Model Training and Parameters

In our experiments, we adapt two classic deep learning models to the downstream encoder s, namely CNN and BiLSTM. Although there are many other well-developed neural network models, such as C-LSTM [36], bi-attentional classification network [20], etc., we consider CNN and BiLSTM as the starting point due to their wide use and relatively few trainable parameters.

We implement the two encoders using the AllenNLP framework [11]. The configuration details of each encoder are given below.

- LSTM is commonly utilised for sentence-level text classification, due to its ability to understand context-dependent sequences [12]. In our experiment, we use the simple bidirectional LSTM (BiLSTM) [37] as the encoder with single dimensional hidden size being 256.
- CNN was first used by [15] for sentence classification. In our experiment, we use an adapted simple version of the CNN model as the encoder. In order to control the experiment so as to only examine the effect of word embeddings, the CNN's size is determined strictly. We experimented with small, medium and large CNN variations that have the number of trainable parameters around 0.5x, 0.75x, 1.0x as these of BiLSTM respectively. Our pilot study did not indicate performance differences between the three variations and hence the small CNN (number of filters 100, and 3 filter sizes: 2,3,4) has been reported on account of efficiency considerations.

For common hyper-parameters, we train both downstream models with an Adam optimiser (learning rate = 0.001), in 140 training epochs, patience 5 for early stop, based on accuracy evaluated on the development set. In addition, we apply dropout with p between 0.2 and 0.5 across layers of the downstream networks to prevent over-fitting.

4.3 Word Embeddings

Given a neural network model, the word embeddings are used as the feature representations in the emebedding layer.

To keep the embedding layer relatively small, we reduce the vocabulary by ignoring words that appear fewer than three times in total within a dataset for non-character based word embeddings[2]. We experimented using both the reduced vocabulary and full vocabulary on some datasets. As no significant performance differences were seen, the vocabulary reduction schema is applied in our experiment, keeping efficiency in mind. Below is more information on each word embedding choice in our experiment.

- **Baseline**: For the baseline, we simply define a randomly-initialised trainable embedding layer with a dimension of 100 for encoding word inputs. Thus the baseline is not pre-trained but learned from scratch with the rest of the downstream model parameters. Hence, it is considered to study the impact that word embeddings have on performance when learned from scratch or pre-trained.
- **word2vec**: We use the off-the-shelf word2vec embedding pre-trained on Google News dataset, which yields vectors with 300 dimensions.
- **GloVe**: There is a good list of commonly-used pre-trained GloVe embeddings available online[3]. These embeddings vary in size, and are pre-trained on a variety of datasets including Wikipedia, Common Crawl[4], and Twitter. In our experiment, we choose the 50-300-dimension GloVe embedding pre-trained on Wikipedia and Common Crawl (6B and 840B tokens).
- **FastText**: Pre-trained FastText models are available for many languages[5]. Because our experimental datasets are limited to English, we only choose the 300-dimension English version of pre-trained FastText embedding.
- **Character-level embedding (Char)**: In our experiment, we define a CNN encoder for learning character-level word embeddings from scratch. We include the character embedding with the intention of exploring the performance difference when it is concatenated with the classic GloVe embedding as opposed to standalone FastText.
- **ELMo**: Different sizes of pre-trained ELMo models are available online[6]. The original pre-trained model is selected in our experiment considering its size and performance. The linear weighted combination of the 3 layers of ELMo (i.e., character-based output, 1st LSTM output, 2nd LSTM output) are configured with one output representation, as used in the original ELMo paper [24]. Since ELMo requires more memory and time as the input sequence's length increases, we set the maximum length of input sequence to be 1,000. This decision is guided by the sample length distribution of our selected datasets as in Figure 3, which shows that only rare samples are longer than 1,000 words.
- **BERT**: Our experiment includes BERT for studying its effectiveness in text classification when being used as a fixed feature extractor. In our experiment, we use the *bert-base-uncased* version that consists of 12 layers, 12 attention heads and hidden size being 768, totalling 110M parameters. Since it is not used in a trainable way in our experiment, as suggested by [31], we select the second-to-last layer instead of the last layer of BERT as the feature representations (i.e., token embeddings) for the downstream encoders[7]. In addition, as BERT only accepts input sequences with the number of tokens less than 510 [8], we apply a truncation method inspired by [26] and only keep the first 510 tokens of those training samples that are beyond this length.

[2] This excepts ELMo and Char because they are character-based and also BERT because it uses a pre-defined vocabulary in pre-training so we do not alter this.

[3] https://nlp.stanford.edu/projects/glove/

[4] https://commoncrawl.org/

[5] https://fasttext.cc/

[6] https://allennlp.org/elmo

[7] We additionally experimented from the fourth-to-last (-4) layer to the last layer (-1) using CNN on the SST-2 dataset. The results show -4, -3 and -2 slightly outperform -1.

[8] 510 is 512 subtracting the [CLS] and [SEP] tokens

Table 2: Evaluation results: accuracy and macro-F1 for single-label datasets (20NewsGroup and SST-2). Values in bold are the highest in their column. Where + denotes concatenation, G denotes GloVe (6B tokens), F denotes FastText, and the appended number denotes the embedding dimension. 300s refers to the GloVe embedding with 840B tokens pre-trained on Common Crawl. For example, G-100 means the off-the-shelf pre-trained *glove.6B.100*.

	20NewsGroup				SST-2			
	CNN		BiLSTM		CNN		BiLSTM	
	Accuracy	macro-F1	Accuracy	macro-F1	Accuracy	macro-F1	Accuracy	macro-F1
Baseline	78.00±0.94	77.58±0.94	49.02±1.13	49.27±1.13	81.45±1.79	81.39±1.79	83.58±1.70	83.57±1.70
word2vec	82.67±0.85	82.14±0.87	62.45±1.09	62.43±1.09	82.14±1.76	82.07±1.76	84.09±1.68	84.02±1.68
G-50	79.98±0.90	79.44±0.91	69.03±1.04	68.98±1.04	80.78±1.81	80.75±1.81	83.03±1.72	83.00±1.73
G-100	81.03±0.89	80.28±0.90	72.80±1.00	72.58±1.01	81.47±1.78	81.40±1.79	84.15±1.68	84.07±1.68
G-200	82.95±0.85	82.26±0.86	75.47±0.97	75.16±0.98	82.80±1.73	82.76±1.73	85.04±1.64	85.02±1.64
G-300	82.61±0.86	81.85±0.87	74.24±0.99	73.97±0.99	82.03±1.76	81.98±1.77	84.88±1.65	84.86±1.65
F-300	82.16±0.86	81.57±0.88	62.37±1.09	62.40±1.09	81.44±1.79	81.30±1.79	84.33±1.67	84.31±1.67
F-300+G-300s	**84.39±0.82**	**83.65±0.84**	76.04±0.96	75.77±0.97	83.73±1.70	83.73±1.70	85.76±1.61	85.75±1.61
Char+G-100	82.11±0.87	81.31±0.88	79.31±0.91	78.78±0.92	82.46±1.75	82.41±1.75	84.79±1.65	84.75±1.65
ELMo	78.94±0.92	78.33±0.93	71.86±1.02	71.13±1.02	88.12±1.49	88.11±1.49	89.27±1.42	89.27±1.42
BERT	83.60±0.84	82.88±0.85	**81.25±0.88**	**80.54±0.89**	**90.01±1.38**	**90.00±1.38**	**90.04±1.38**	**90.04±1.38**

Table 3: Evaluation results: accuracy and micro-F1 for multi-label datasets (AAPD and Reuters). Values in bold are the highest in their column and the abbreviations in the first column are equivalent to these in Table 2.

	AAPD				Reuters			
	CNN		BiLSTM		CNN		BiLSTM	
	Accuracy	micro-F1	Accuracy	micro-F1	Accuracy	micro-F1	Accuracy	micro-F1
Baseline	35.30±1.90	66.82±1.88	34.77±1.90	65.26±1.90	74.40±1.40	80.97±1.26	55.11±1.59	60.50±1.57
word2vec	35.93±1.91	68.30±1.85	36.07±1.91	67.85±1.86	80.40±1.27	86.09±1.11	77.61±1.34	82.27±1.22
G-50	34.27±1.89	66.31±1.88	35.27±1.90	67.29±1.87	78.65±1.31	84.48±1.16	77.39±1.34	82.29±1.22
G-100	35.50±1.91	68.04±1.86	36.03±1.91	68.02±1.86	80.45±1.27	85.91±1.11	77.96±1.33	83.16±1.20
G-200	34.87±1.90	67.23±1.87	35.63±1.91	68.02±1.86	80.85±1.26	86.09±1.11	79.96±1.28	84.35±1.16
G-300	34.80±1.90	67.28±1.87	35.57±1.91	68.32±1.85	80.85±1.26	86.45±1.10	80.04±1.28	84.32±1.16
F-300	34.67±1.90	67.37±1.87	34.57±1.89	67.03±1.87	78.68±1.31	84.57±1.16	73.91±1.41	79.17±1.30
F-300+G-300s	35.93±1.91	**69.19±1.84**	**36.07±1.91**	**68.42±1.85**	**81.54±1.24**	**86.87±1.08**	**80.48±1.27**	84.55±1.16
Char+G-100	34.87±1.90	67.43±1.87	35.80±1.91	68.22±1.85	81.32±1.25	86.69±1.09	78.56±1.31	83.84±1.18
ELMo	34.77±1.90	67.25±1.87	31.53±1.85	62.69±1.93	79.13±1.30	84.99±1.14	74.67±1.39	80.39±1.27
BERT	**36.27±1.92**	68.35±1.85	35.77±1.91	67.48±1.87	81.00±1.26	86.23±1.10	79.89±1.28	**84.90±1.15**

5 RESULTS AND DISCUSSION

Based on the experimental components as described, we report the evaluation results that are averaged by repeating each run three times with different random seeds. For all datasets, the results are evaluated using test set accuracy. In addition to accuracy, we also report the evaluation on macro-F1 for single-label datasets and micro-F1 for multi-label datasets. Table 2 and Table 3 present the evaluation results for single-label and multi-label datasets respectively. To gain a general sense of how each score in the tables is comparable to another, we append the confidence intervals (CIs) for each also. We use Wilson Score Interval [30] to compute the confidence interval for each metric at the 95% level. For a cell score c, its interval is calculated by $c = \hat{c} \pm z\sqrt{\frac{\hat{c}(1-\hat{c})}{n}}$, where n is the number of observations evaluated upon (equal to the number of of test samples in our case) and z is the constant (equal 1.96 for 95% CI).

Based on the evaluation results, we summarise several key findings as follows:

5.1 Pre-trained vs non pre-trained

The baseline run (non pre-trained) differentiates itself from the pre-trained ones where the embedding layer is initialised with pre-trained weights before being trained along with the task-specific datasets. When inspecting the baseline performance across the datasets using CNN and BiLSTM, some interesting findings emerge. To conduct a fair comparison, if looking at the baseline and G-100, (both are 100 dimensional), overall we found the performance difference between baseline and the rest is positive but not substantial in the SST-2 and AAPD datasets. However, the positive difference

becomes more pronounced for 20NewsGroup and Reuters. This can possibly be attributed to the characteristics of the datasets in terms of sample variance. It seems that pre-trained embeddings such as G-100 gain a strong advantage over non pre-trained in datasets like 20NewsGroup that contains varying samples of many words. The advantage is less pronounced for short sequence dataset such as SST-2, likely because non pre-trained can easily be trained and catch up to a point where it offers similar features to the pre-trained embeddings. The insight we share from here is that it is better to initialise the embedding layer with pre-trained weights than without in this kind of downstream classification tasks although how much performance gain is largely determined by the dataset characteristics.

5.2 Model architecture selection: CNN vs BiLSTM

In our experiment, we applied CNN-based and BiLSTM-based models for sequence classification. The experimental results do not reveal a general advantage of one over another. Instead, the performance impact of model architectures depends on the dataset characteristics.

First of all, observing the results in Table 2 and Table 3, we see that BiLSTM exhibits a slight advantage over CNN on SST-2. However, CNN overtakes BiLSTM quickly as the document length and variance of the datasets increase. For 20NewsGroup, CNN achieves a significant improvement over BiLSTM. Motivated by these observations, further investigation into the relationship between the dataset sample length and choice of model architecture was conducted. For this experiment, datasets were split into several partitions by sample length. Table 4 presents the statistics of 20NewsGroup and AAPD splits we selected for this experiment. As seen, each of the dataset is split into short, medium and long partitions by length from the original training, development, and test sets accordingly. Next, we select G-100 and ELMo for training and evaluating each split. Table 5 shows the evaluation scores of CNN and BiLSTM on the three splits using G-100 and ELMo in the embedding layer. These results go against our hypothesis that BiLSTM may gain an advantage in short splits. However, Table 5 indicates that CNN outperforms BiLSTM in all circumstances no matter whether the samples are shorter or less distributional and they both generally perform better on long-sample splits than short ones. This bring us to the next stage of qualitative analysis of the impact that dataset traits have on CNN or BiLSTM.

Analytically, the short samples in the datasets are just titles (news titles for Reuters, news group titles for 20NewsGroup, short abstracts for AAPD) instead of context-flow alike sentences, see Table 6. This indicates that there is no context in the sequence to help identify which label the sequence is, which may explain why LSTM does not do well even in short samples of datasets of this kind.

Table 5: Performance of 20NewsGroup and AAPD splits with G-100 and ELMo

	20NewsGroup with G-100			
	CNN		BiLSTM	
	Acc.	Macro-F1	Acc.	Macro-F1
S-short	0.7207	0.6906	0.4598	0.4327
S-medium	0.7837	0.7724	0.5542	0.5437
S-long	0.7878	0.7634	0.5386	0.4948
	20NewsGroup with ELMo			
	CNN		BiLSTM	
	Acc.	Macro-F1	Acc.	Macro-F1
S-short	0.6586	0.6268	0.5622	0.5424
S-medium	0.7474	0.7301	0.6351	0.6336
S-long	0.7584	0.7304	0.7014	0.656
	AAPD with G-100			
	CNN		BiLSTM	
	Acc.	Micro-F1	Acc.	Micro-F1
S-short	0.3058	0.6268	0.3027	0.5997
S-medium	0.3542	0.6642	0.3325	0.6279
S-long	0.3575	0.6877	0.3432	0.6545
	AAPD with ELMo			
	CNN		BiLSTM	
	Acc.	Micro-F1	Acc.	Micro-F1
S-short	0.3052	0.6162	0.2562	0.5623
S-medium	0.3431	0.6599	0.3027	0.597
S-long	0.3525	0.6677	0.3098	0.616

The primary lesson we learn from here is that it is not proper to say LSTM is good at short sequence classification. Instead, we need to choose CNN or LSTM based on the characteristics of datasets. As we see in Table 2, LSTM outperforms CNN in SST-2 due to it is sentence-level sentiment analysis (the sequential context does help decide the sentiment category in this case). However, for document datasets that have similar characteristic like AAPD and 20NewsGroup, LSTM does not add benefit to the performance as opposed to CNN.

5.3 Impact of different classic embeddings

In our experiment, word2vec, GloVe, and FastText are the three classic embeddings for comparative analysis. Taking a closer look

Table 4: Statistics of 20NewsGroup and AAPD splits by length

	20NewsGroup			AAPD		
	# train/dev/test	len_avg.	len_std.	# train/dev/test	len_avg	len_std.
S-short	3017/754/2510	84.64	27.19	15988 / 1000 / 1612	92.23	26.60
S-medium	3017/754/2510	178.21	31.95	15988 / 1000 / 1612	157.78	16.88
S-long	3017/755/2512	587.97	816.64	15990 / 1000 / 1614	240.35	41.06

Table 6: Examples of short samples from Reuter, AAPD, 20NewsGroup that show that the short sequences are not context-indicative of for which label(s) the sequences should fall into. The n-grams in bold are very indicative of which label the short sequences fall into, which is the feature that CNN captures well.

	Short-length sample	Label desc.
Reuter	**CHARTER FEDERAL**, JEFFERSON SAVINGS AGREE TO MERGE.	news topics, e.g., this sample is about politics
AAPD	this article summarizes recent trends in **mobile biometrics**	paper topics, e.g., this sample is about mobile biometrics
20NewsGroup	From: kwgst+@pitt.edu (Mr. Someone) Subject: modem question Article-I.D.: blue.9061 Organization: **pre-EE** Lines: 2	topics, e.g., this sample is about hardware

at word2vec and GloVe variants, Figure 4 plots the performance difference between word2vec and GloVe with CNN and BiLSTM. We can see from the plots that, overall, GloVe has an advantage over the classic word2vec because its 100-dimension vector achieves almost the same performance as the 300-dimension word2vec. For GloVe, the performance increases as the dimensions increase initially, but drops gradually as it continues to increase to 300. When selecting GloVe for a general classification dataset, it is good to use its 100-200 (6B tokens) dimensional pre-trained embeddings.
In addition to word2vec and GloVe, we also conduct experiments for FastText and character-level embeddings. These two types of embeddings share the common feature of capturing subword information. When using the 300-dimensional FastText as the standalone embedding, there is no evidence to indicate that FastText outperforms GloVe. However, it does achieve overall increased scores when it is used in combination with GloVe. As seen in Table 2 and Table 3, the values in bold show that this embedding combination achieves the highest scores for most situations. Two concerns arise for using this combination in real-world applications, however. On one hand, when taking into account the confidence intervals, the increased scores achieved by the combination are not significantly greater than most other standalone word embeddings such as word2vec and G-200. On the other hand, the combination largely increases the embedding size (6 that of G-100, and 3 times that of G-300 and F-300) and thus need to be used carefully despite its marginal increased scores as seen in our experiment.

For character-level embedding (Char), we find overall the performance of its concatenation with the 100 dimensional GloVe (Char+G-100) is marginal or even worse (for AAPD) as compared to the standalone use of GloVe (G-100). As a suggestion, it is better to focus on downstream model selection based on dataset characteristics than to concatenate sub-word pre-trained embeddings for performance gain.

5.4 Classic embeddings vs contexualised embeddings

There are two types of contexualised embeddings (ELMo and BERT) studied in our experiment. The results indicate that for SST-2, ELMo and BERT essentially reach equivalent performance performance

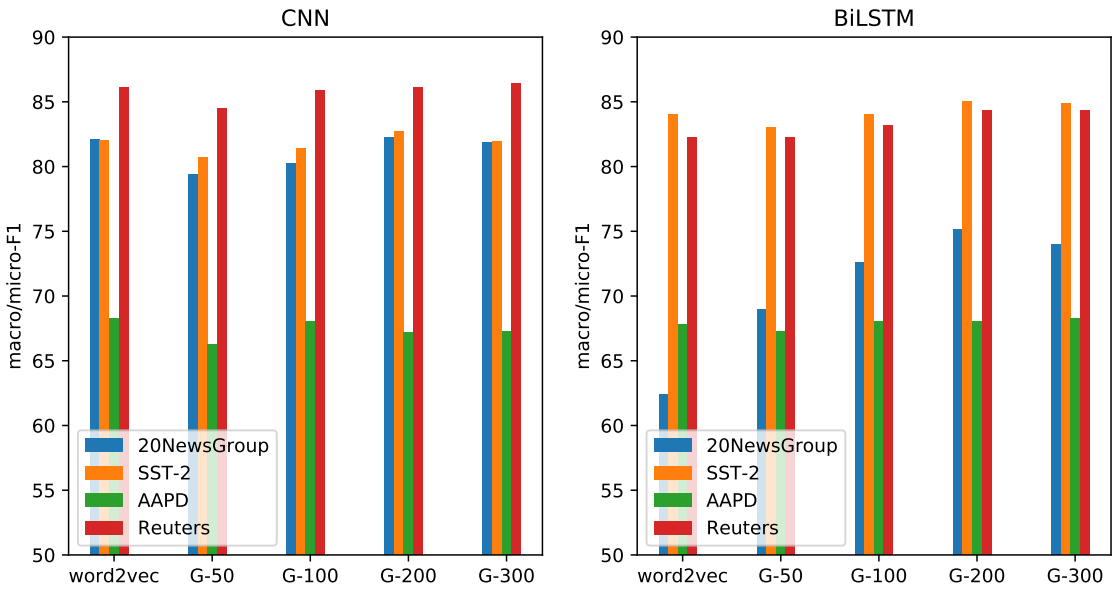

Figure 4: Different embeddings (word2vec and GloVe) versus macro (20NewsGroup and SST-2) and micro (AAPD and Reuters) -F1 with CNN and BiLSTM.

no matter whether CNN or BiLSTM is used. However, BERT's advantage over ELMo becomes more obvious in 20NewsGroup, Reuters and AAPD. Compared with classic word embeddings, ELMo and BERT achieve a strong improvement over classic ones for the sentence-level dataset SST-2. This improvement is not observed in the remaining datasets and even in some cases ELMo harms the performance such as for 20NewsGroup and AAPD.

Analytically, there may be two causes to point out. First, for a document dataset like 20NewsGroup, a truncation method for shortening the input sequence is applied for both ELMo and BERT, due to their restrictions on input length[9]. This means that parts of a document longer than the allowed length are omitted, thus potentially leading to the loss of important information in the document. Second, the two embeddings are non recursively pre-trained only on limited-length chunks of a corpus so it may lack the expression of global semantics of a long document. This motivates a rethink of how a long document should be represented by the contextualised embeddings that perform well on relatively short sequences. Among the work behind this motivation, Transformer-XL [33] may serve as a good reference to the community for document representation learning in the future.

6 CONCLUSION AND FUTURE WORK

This paper introduces a comparative study of word embeddings using CNN and BiLSTM as the downstream encoders for text classification. To make the work as comprehensive as possible, we studied both classic word embeddings (such as word2vec, GloVes and FastText) and two mainstream contextualised embeddings (ELMo and BERT). In addition, we selected four benchmarking classification datasets for exploring how the selection of downstream model and word embeddings affects classification performance on datasets with different characteristics. Finally, we summarised several key findings based on our comparative study. For example, as an overall suggestion based on the experimental results (with confidence intervals), contextualised embeddings are matched well with BiLSTM for SST-2-like classification datasets, while CNN collaborates well with classic embeddings for document datasets like 20NewsGroup, AAPD, and Reuters. Although the work is offered to provide some evidence-based guidance for practitioners selecting word embeddings for text classification in deep models, there is still quite a list of work we seek to explore in the future, as summarised below.

- It remains a research question on how to embed a long document for improving classification performance, especially when using pre-trained contextualised word embeddings, such as ELMo, BERT, XLNet [33], RoBERTa [19], etc.
- One limitation of the study is that the selected datasets are all in English. It would be interesting to conduct a similar study on other languages such as Chinese, or French [5]. Further study can help to explore whether there are any performance alignments concerning embeddings that are pre-trained on different linguistic corpora for text classification.
- It would be interesting to investigate these embeddings on other classification models. This could be beyond neural network models and utilise traditional machine learning methods (ML) such as Naïve Bayes, SVM, etc. The traditional ML approaches are more efficient in terms of training and inference than the deep models used as the downstream encoders. Hence, it may be interesting to explore the tradeoff between the efficiency reward and the likely loss of performance.

[9]This may not true for ELMo with an allowed maximum length 1000 since rare training samples in our selected datasets are longer than that.

REFERENCES

[1] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. DocBERT: BERT for Document Classification. *arXiv preprint arXiv:1904.08398* (2019).

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*. 1638–1649.

[3] Chidanand Apté, Fred Damerau, and Sholom M Weiss. 1994. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems (TOIS)* 12, 3 (1994), 233–251.

[4] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 238–247.

[5] Emily M Bender. 2019. The# BenderRule: On Naming the Languages We Study and Why It Matters. *The Gradient* (2019). https://thegradient.pub/the-benderrule-on-naming-the-languages-we-study-and-why-it-matters/

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Bhuwan Dhingra, Hanxiao Liu, Ruslan Salakhutdinov, and William W Cohen. 2017. A comparative study of word embeddings for reading comprehension. *arXiv preprint arXiv:1703.00993* (2017).

[9] Manaal Faruqui and Chris Dyer. 2014. Community evaluation and exchange of word vectors at wordvectors. org. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 19–24.

[10] Peter W Foltz. 1996. Latent semantic analysis for text-based research. *Behavior Research Methods, Instruments, & Computers* 28, 2 (1996), 197–202.

[11] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. 1–6.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Wei Huang, Enhong Chen, Qi Liu, Yuying Chen, Zai Huang, Yang Liu, Zhou Zhao, Dan Zhang, and Shijin Wang. 2019. Hierarchical Multi-label Text Classification: An Attention-based Recurrent Network Approach. In *Proceedings of the 28th ACM CIKM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, CHINA, Nov 3-7, 2019*. 1051–1060.

[14] Ibrahim Kaibi, Hassan Satori, et al. 2019. A comparative evaluation of word embeddings techniques for twitter sentiment analysis. In *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*. IEEE, 1–4.

[15] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016).

[17] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[18] Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*. Elsevier, 331–339.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[20] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*. 6294–6305.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[22] Marwa Naili, Anja Habacha Chaibi, and Henda Hajjami Ben Ghezala. 2017. Comparative study of word embedding methods in topic segmentation. *Procedia computer science* 112 (2017), 340–349.

[23] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[24] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*. 2227–2237.

[25] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1631–1642.

[26] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to Fine-Tune BERT for Text Classification? *Chinese Computational Linguistics* (2019), 194–206. https://doi.org/10.1007/978-3-030-32381-3_16

[27] Avijit Thawani, Biplav Srivastava, and Anil Singh. 2019. SWOW-8500: Word Association Task for Intrinsic Evaluation of Word Embeddings. In *Proceedings of the 3rd Workshop on Evaluating Vector Space Representations for NLP*. 43–51.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[29] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C-C Jay Kuo. 2019. Evaluating word embedding models: Methods and experimental results. *APSIPA transactions on signal and information processing* 8 (2019).

[30] Edwin B Wilson. 1927. Probable inference, the law of succession, and statistical inference. *J. Amer. Statist. Assoc.* 22, 158 (1927), 209–212.

[31] Han Xiao. 2018. bert-as-service. https://github.com/hanxiao/bert-as-service.

[32] Pengcheng Yang, Xu Sun, Wei Li, Shuming Ma, Wei Wu, and Houfeng Wang. 2018. SGM: Sequence Generation Model for Multi-label Classification. In *Proceedings of the 27th International Conference on Computational Linguistics*. 3915–3926.

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*. 5754–5764.

[34] Lei Zhang, Shuai Wang, and Bing Liu. 2018. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018), e1253.

[35] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.

[36] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1511.08630* (2015).

[37] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 3485–3495.