On Character vs Word Embeddings as Input for English Sentence Classification

James Hammerton(✉), Mercè Vintró, Stelios Kapetanakis, and Michele Sama

Kare [knowledgeware] (formerly Gluru) LTD, Aldwych House 71-91,
Aldwych, London WC2B 4HN, UK
{james,merce,stelios,michele}@karehq.com

Abstract. It has become a common practice to use word embeddings, such as those generated by word2vec or GloVe, as inputs for natural language processing tasks. Such embeddings can aid generalisation by capturing statistical regularities in word usage and by capturing some semantic information. However they require the construction of large dictionaries of high-dimensional vectors from very large amounts of text and have limited ability to handle out-of-vocabulary words or spelling mistakes. Some recent work has demonstrated that text classifiers using character-level input can achieve similar performance to those using word embeddings. Where character input replaces word-level input, it can yield smaller, less computationally intensive models, which helps when models need to be deployed on embedded devices. Character input can also help to address out-of-vocabulary words and/or spelling mistakes. It is thus of interest to know whether using character embeddings in place of word embeddings can be done without harming performance. In this paper, we investigate the use of character embeddings vs word embeddings when classifying short texts such as sentences and questions. We find that the models using character embeddings perform just as well as those using word embeddings whilst being much smaller and taking less time to train. Additionally, we demonstrate that using character embeddings makes the models more robust to spelling errors.

Keywords: Word embeddings · Character embeddings
Long short-term memory networks · Convolutional networks
Text classification · Natural language processing

1 Introduction

In recent years, it has become a common practice to use word embeddings, such as word2vec [1] or GloVe [2], as input to neural networks performing natural language processing (NLP) tasks, such as language modelling, named entity recognition, sentiment analysis or machine translation. This is because such embeddings

Mercè and Stelios have since left the company. Mercè and Stelios can be contacted via merce.ricart@shell.com and s.a.kapetanakis@gmail.com respectively.

K. Arai et al. (Eds.): IntelliSys 2018, AISC 868, pp. 550–566, 2019.
https://doi.org/10.1007/978-3-030-01054-6_40

aid generalisation by capturing statistical regularities in the way words are used, and indeed they can capture many aspects of the meaning of the words. For example, [1] demonstrates that simple vector addition of two embeddings can yield a vector semantically related to both, e.g. 'Vietnam' + 'capital' yielding 'Hanoi' as its closest neighbour and that analogical reasoning can be performed via linear operations on the embeddings. They thus provide a rich representation of words that learners can exploit to perform their tasks effectively.

However such word embeddings require large amounts of textual data to be trained (e.g. [1] used a corpus of 1 billion words when learning embeddings for single words, and then another corpus of 33 billion words when learning embeddings for phrases) and then the construction of large dictionaries of high-dimensional vectors. To handle out of vocabulary (OOV) words or spelling mistakes they require an "unknown" embedding to be used, either randomly generated or generated from rare tokens being replaced by a specialised token. This limits the extent to which OOV words and spelling mistakes can be handled effectively.

Finally, for some NLP tasks people are also finding that they can get good results using character-level input alone (e.g. [3]). There are a few advantages to using character level input that make this option attractive if similar performance can be achieved to that of using word-level input:

- Character embeddings will be based on a fixed set of characters, which will be much smaller than the vocabularies typically used in NLP tasks. Where character input replaces word-level input, this makes the models smaller and more computationally efficient, which is an important consideration when employing them on embedded devices with limited resources.
- Character-level input can help a model deal with OOV words and spelling mistakes, at least where new words with similar spellings to existing words have similar meanings. We will test this point empirically.

In this paper, we explore the use of word embeddings vs character embeddings for two classification tasks using English sentences. The rest of this paper is organised as follows:

- Section 2 sets out the background to the work presented here and the research questions we set out to address.
- Section 3 discusses work related to this paper.
- Section 4 describes the tasks and datasets we use for this work.
- Section 5 describes the approach taken, including the models we use, the preprocessing of the sentences we perform and how the evaluation is carried out.
- Section 6 presents the results of the experiments.
- Section 7 discusses the results presented.
- Section 8 presents our conclusions and discusses some options for future work.

2 Background and Research Questions

Prior to this work, we already had a classifier employing word2vec embeddings deployed to make predictions as part of the Gluru task manager application[1]. However we became aware that character models had been applied successfully for some tasks and decided to evaluate whether using character embeddings rather than word embeddings in our production system would make sense. Note that here we're not just interested in the performance of the classifier in terms of how well it would make its predictions but also in terms of the costs of deploying the classifier, e.g. model size, memory usage, prediction times, etc. We thus set out to compare the 2 approaches both in terms of prediction performance and in terms of the computational resources used. We therefore decided to address the following research questions:

- Will using character embeddings yield the same performance as we get with word embeddings?
- What are the training costs (e.g. amount of training data required, time required) of using character embeddings vs word embeddings?
- What are the costs of deploying the models, e.g. memory usage, disk usage, number of parameters, time taken per prediction?
- How resilient are the models to handling spelling errors and OOV words?

By answering these questions, we were then able to make a well informed choice about which model to deploy. These questions will be applicable any time one considers using character vs word embeddings as input for a task, and indeed can be generalised to any choice of two models where processing language is concerned and the possibility of spelling errors or OOV words arises.

3 Related Work

As the amount of available textual information grows, the task of text classification has become fundamental in the field of natural language processing. The body of research concerned with text classification, which involves assigning one of a set of predefined classes to a textual document, is wide and has focused on several of its applications for tasks such as information retrieval, ranking and document classification [4,5]. In our work we focus on short-text classification (i.e. sentences), which has become an important application of text classification for tasks such as sentiment analysis, as well as question answering and dialogue systems.

Several approaches have been proposed for the problem of short-text classification, which leverage traditional machine learning techniques such as Support Vector Machines (SVMs) [6] and Conditional Random Fields (CRFs) [7]. There has also been a body of work concerned with sequential short-text classification [8,9] due to the fact that short texts tend to appear in sequence (e.g. sentences

[1] This application will soon be discontinued.

in a document). However, we limit the scope of our work and only consider short texts in isolation. Recently, models based on neural networks have become very popular due to their good performance. Research on short-text classification has been performed with different neural architectures such as convolutional neural networks (CNNs) [10] and long short-term memory networks (LSTMs) [11]. Many of the neural network approaches treat words as their basic units and make use of word embeddings [12,13], which are fundamental to state-of-the-art NLP. However, there has been a growing interest in developing models at the character level, generating character embeddings using words as a basis [14] or directly from a string of raw characters [3]. This has resulted in performance comparisons between word and character-based models [3]. Our work extends this research to the task of short-text classification and presents further experiments that compare character and word embedding approaches.

4 Tasks and Data Sets

4.1 Benchmark Sentence Classification Problem

Here we used the dataset constructed for our internal sentence classification problem. The task is a binary classification task with a positive class and a negative class.

The dataset consists of sentences extracted from users emails as they appeared in the Gluru task manager application supplemented with some sentences from the emails in the Enron corpus [15]. The sentences were manually labelled.

Table 1 summarises the details of this dataset, reporting the number of sentences in the test and training set alongside the means and standard deviations (given in brackets) of the words per sentence and characters per sentence, plus the numbers of positive and negative examples. Note that during training a random 20% of the training set is used as a validation dataset (e.g. used for deciding best weights or early stopping).

Table 1. Dataset characteristics for benchmark sentence classification problem

	Sentences	Mean (stdev) Words/Sentence	Mean (stdev) Chars/Sentence	Examples	
				Positive	Negative
Training	19,518	14.07 (10.34)	80.82 (64.41)	6,745	12,773
Test	4,876	14.02 (10.02)	80.41 (63.66)	1,683	3,193

4.2 Effect of Reducing Training Set Size on Benchmark Sentence Classification Problem

Here we use the same dataset and task as in Sect. 4.1, but we vary the size of the training set from 5%, 10% and then 20% to 80%, in 20% increments, of the total

size (via random sampling) and see the impact on performance for both character vs word embeddings when evaluating with the entire testing set. The motivation for this is the hypothesis that there might be some size of dataset below which we get differing levels of performance from the two models, and we wanted to cater for the possibility that our dataset was already above the threshold.

4.3 Effect of Introducing Spelling Errors on Benchmark Sentence Classification Problem

In order to assess the robustness of the models to spelling mistakes, we used the same dataset and task as in Sect. 4.1, and we introduced spelling errors during the evaluation phase. We did this by altering X randomly selected characters from each sentence at random, and substituting each character selected with a random choice from the alphanumeric characters or a space. We did this for values of $X = 1$, 4, 8, 12 and 16.

4.4 Sentence Classification with Parakweet Lab's Email Intent Dataset

We wanted to assess whether the findings we had with our own internal dataset would carry over to a publicly available data set and we chose the Email Intent Dataset[2] because it was also derived from email data and consisted of English sentences classified into to two classes. This consists of a training set and a test set of sentences drawn from the Enron corpus [15] and labeled as to whether they contain an 'Intent', defined to correspond primarily to the categories 'request' and 'propose' in [16]. In some cases, the creators also applied the positive label to some sentences from the 'commit' category if they contained a datetime.

The dataset details are provided in Table 2[3]. As before, the table reports the number of sentences, means and standards deviations for the word per sentence and characters per sentence and the numbers of positive and negative examples for the training and testing sets. Again during training 20% of the training set is set aside for validation, and here the validation performance was used in some runs for early stopping or determine which epoch produced the best set of weights.

Table 2. Dataset characteristics for Parakweet's email intent detection

	Sentences	Mean (stdev) Words/Sentence	Mean (stdev) Chars/Sentence	Examples	
				Positive	Negative
Training	3,657	16.56 (10.66)	91.64 (62.19)	1,719	1,938
Test	991	16.90 (11.33)	93.14 (68.08)	309	682

[2] https://github.com/ParakweetLabs/EmailIntentDataSet.

[3] The numbers reported here for the size of the training and test sets are what we found in the dataset on direct examination but the Wiki page for the dataset reports different numbers.

As this dataset is quite small to start with, we did not repeat the experiment reducing the size of the training data here.

5 The Approach

5.1 Convolutional Networks for Sentence Classification

We employ convolutional neural networks (CNNs) for the purposes of this paper. In natural language processing, CNNs are increasingly being used as an effective alternative (e.g. [10,12,17]) to the commonly used recurrent neural networks (RNNs) such as (bi)LSTM networks [18,19]. They are often more efficient to implement, requiring only a single forward pass to perform predictions compared to the multiple passes required with RNNs, whilst they do not require backpropagation through time thus simplifying training.

The network architectures we trained thus followed the following template[4]:

- Depending on whether we are using character or word embeddings, the input layer has either $M \times C$ or $N \times W$ units, specifying either the first M characters or the N words of the input sentence with zero padding used where a sentence is shorter than M characters or N words. C and W indicate the size of the character and word vocabularies respectively, with 1 hot vectors indicating which character or word is being presented.
- We then add a single convolution layer over these units using stride 1 and filter length 3, and 250 filters, with max pooling.
- The output from the convolution/maxpooling is then fed into a single dense layer of 100 units.
- Finally a softmax output layer is attached with 1 unit for each output class, in both cases here this means we have 2 output units[5].

When using word embeddings as input, we used the Google News pre-trained word2vec embeddings for English[6]. These are 300 dimensional vectors. The embeddings are refined during training.

When using character embeddings, we allocate 100 dimensions and let the networks learn the embeddings from random initialisation.

5.2 Preprocessing of Text

For both character and word-level input and both datasets, the sentences have already been extracted and we convert the sentences to lower case.

[4] To build and train the networks, we used Python 3.5 and Keras (https://keras.io/) with Tensorflow (https://www.tensorflow.org/) as the backend. All the experiments reported here were performed on an Ubuntu 16.04 Linux server with 64GB main RAM, using a NVIDIA GeForce GTX 1070 GPU with 8GB RAM.

[5] We could use a binary output here instead.

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

For word-level input we then tokenise on whitespace and present up to the first N words to the network, where for each word we look up the embedding from the Google News word2vec embeddings, defaulting to a random embedding for unknown words.

For character level input, we present up to the first M characters of the sentence to the network. To avoid having to store a dictionary mapping each character to its representation to the neural network, we hash each character to one of 968 different values - this number was chosen to be much larger than the vocabulary size in order to avoid collisions.

5.3 Evaluation

In each experiment, we evaluate by presenting the relevant test set and comparing the predictions to the expected results. To account for variations in performance when keeping training algorithm and hyper parameters constant, each experiment involves performing five training runs, each starting from random weight initialisations, and then evaluating the performance of the trained network for each run.

As both tasks are classification tasks, we compute following standard classification metrics for each class:

- the precision, P, where $P = T_P/(T_P + F_P)$
- the recall, R, where $R = T_P/(T_P + F_N)$
- the fscore, $F1$, which is the harmonic mean of the precision and recall, where $F1 = 2PR/(P + R)$

where:

- T_P is the number of true positives, i.e. those examples that are correctly classified as members of the class
- F_P and F_N are the number of false positives and false negatives respectively, i.e. the examples incorrectly identified as members of the class or incorrectly identified as not being members of the class, respectively.

We report the average and standard deviation of these metrics computed across the 5 runs for each experiment.

6 Experimental Results

6.1 Benchmark Supervised Classification Results

Table 3 summarises the network architectures used for these experiments. The columns are as follows:

- Network - contains labels for the network used, e.g. Char512 indicating character embeddings using the first 512 characters of a sentence and Word89 indicating word embeddings as input using the first 89 words of the sentence. These labels are used in other tables as references to the networks concerned.

Table 3. Networks used for benchmark sentence classification problem

Network	Inputs	Weights	Epoch duration	Size on disk
Char512	512 chars	197,352	6.6s	2.3 MB
Word89	89 words	55,877,752	68.2s	640 MB
Word20	20 words	55,877,752[a]	48.6s	640 MB
Network	Memory usage	Prediction time total (per sentence)		
Char512	3 MB	0.456s (0.094 ms)		
Word89	448 MB	0.366s (0.076 ms)		
Word20	448 MB	0.322s (0.067 ms)		

[a] Because the weights for each input window in a convolution network are shared across all the input windows, altering the number of input words/characters presented does not alter the number of weights.

- Inputs - indicates the number and type of inputs.
- Weights - reports the number of weights in the network.
- Epoch duration - reports the duration in seconds of one complete presentation of the dataset during training.
- Size on disk - reports the size of the saved Keras models.
- Memory usage - reports the estimated memory usage of the models.
- Prediction time - reports the total time and time per sentence for predicting the classifications of the test set.

In all cases, the number of filters was 250, the filter length was 3, hidden layer size was 100 units, and the networks were trained for 5 epochs with batch size 16, using the ADAM optimiser [20] and a learning rate of 0.2. Dropout was also applied to the weights from the final hidden layer to the output during training with probability of 0.2.

As this table shows, both the size of the model (the number of weights) and the runtime are much smaller for the character models than for the word embedding models. However the character embeddings models were slightly slower at doing predictions than the word embeddings model. We hypothesise that the sliding window being run over 512 characters (compared to 89 words) is offsetting other efficiency gains here. The fact that the version of the word model that uses only the first 20 words is faster than the one using 89 words provides some support for this given the 2 models are the same size otherwise.

Table 4 details the results of the experiments. The columns are as follows:

- Network - network label as per Table 3.
- Precision - reports the mean and, in brackets, the standard deviation of the precision across the 5 runs.
- Recall - reports the mean and, in brackets, the standard deviation of the recall across the 5 runs.
- F1 - reports the mean and, in brackets, the standard deviation of the F1 across the 5 runs.

Table 4. Benchmark sentence classification results

Positive class			
Network	Precision (stdev)	Recall (stdev)	F1 (stdev)
Char512	**0.74** (0.05)	**0.81** (0.05)	**0.77** (0.01)
Word89	0.70 (0.04)	0.83 (0.05)	0.76 (0.01)
Word20	0.70 (0.06)	0.79 (0.08)	0.74 (0.02)
Negative class			
Network	Precision (stdev)	Recall (stdev)	F1 (stdev)
Char512	**0.90** (0.02)	**0.85** (0.05)	**0.87** (0.01)
Word89	0.90 (0.02)	0.80 (0.05)	0.85 (0.02)
Word20	0.88 (0.03)	0.82 (0.07)	0.84 (0.03)

For both the positive and negative classes, whilst the model using character inputs obtained the best mean scores for each metric, the standard deviations show that there is no significant difference between using character embeddings vs word embedding when the first 512 characters are used vs the first 89 words of a sentence. Reducing the number of words used to 20 increases the variability of the scores when using word embeddings and caused the average F1 to drop slightly.

6.2 Results for Varying Training Set Size

Table 5 summarises the results for these experiments, note that this was only done for the "Char512" and the "Word89" networks. The columns are:

- Percent - the percentage of the training data used to train the model.
- Positive class F1 - the mean and standard deviation (in brackets) of the F1 for the positive class computed across the 5 runs.
- Negative class F1 - the mean and standard deviation (in brackets) of the F1 for the negative class computed across the 5 runs.

As the table illustrates, the smaller dataset sizes do reduce performance though perhaps not as dramatically as one might expect, until we get down below 20% of the original size. Interestingly, we get only slightly reduced performance for using 40% of the data compared to 100% here. Also the performance of the negative class drops slightly when going from 80% to 100%, though given the variability indicated by the standard deviations this may be more of an apparent effect than a real one.

Reducing the size of the training set to 5% has a bigger impact on the performance of the positive class for the character embeddings than the word embeddings, however the mean F1 is still within a standard deviation of the performance of the word embeddings when using only 5% of the data. Otherwise the impact of the reduced dataset sizes looks very similar for the two cases. This

Table 5. Varying size of training data

Word embeddings		
Percent	Positive class F1 (stdev)	Negative class F1 (stdev)
5	0.64 (0.04)	0.83 (0.01)
10	0.68 (0.03)	0.85 (0.01)
20	0.69 (0.05)	0.86 (0.01)
40	0.73 (0.01)	0.87 (0.00)
60	0.75 (0.01)	0.88 (0.01)
80	0.75 (0.02)	0.88 (0.00)
100	0.76 (0.01)	0.85 (0.02)
Best-worst	0.12	0.05
Char embeddings		
Percent	Positive class F1 (stdev)	Negative class F1 (stdev)
5	0.61 (0.06)	0.84 (0.01)
10	0.68 (0.02)	0.85 (0.01)
20	0.70 (0.02)	0.87 (0.00)
40	0.75 (0.02)	0.87 (0.01)
60	0.76 (0.02)	0.87 (0.01)
80	0.76 (0.02)	0.88 (0.00)
100	0.77 (0.01)	0.87 (0.01)
Best-worst	0.16	0.04

suggests that there is not much difference overall in the training data requirements when using character embeddings vs word embeddings, contrary to the hypothesis raised in Sect. 4.2.

6.3 Results for Introducing Spelling Errors on Benchmark Sentence Classification Problem

The results are summarised in Table 6. The 'Perturbations' column indicates how many characters were replaced with a random character as per Sect. 4.3, the remaining columns giving the mean F1 plus standard deviation for the positive and negative classes respectively as in Table 5. Again the perturbations were only done for the "Char512" and "Word89" networks.

Here it can be seen that the performance of both classes becomes progressively worse with the greater number of perturbations, as one would expect. Note however that this effect is only slight for the negative class and is much stronger for the positive class, as indicated by the 'Best-worst' row where the drops for the positive class are 0.53 (word embeddings) and 0.40 (character embeddings).

As one would intuitively expect, the impact on the word embeddings case is larger than in the character embeddings case. The presence of a spelling mistake

Table 6. Effect of varying levels of spelling errors

Word embeddings		
Perturbations	Positive class F1 (stdev)	Negative class F1 (stdev)
0	0.76 (0.01)	0.85 (0.02)
1	0.73 (0.01)	0.87 (0.01)
4	0.60 (0.04)	0.84 (0.00)
8	0.46 (0.03)	0.82 (0.00)
12	0.35 (0.04)	0.81 (0.01)
16	0.23 (0.06)	0.80 (0.01)
Best-worst	0.53	0.05
Char embeddings		
Perturbations	Positive class F1 (stdev)	Negative class F1 (stdev)
0	0.77 (0.01)	0.87 (0.01)
1	0.75 (0.02)	0.88 (0.01)
4	0.67 (0.04)	0.87 (0.01)
8	0.56 (0.06)	0.85 (0.01)
12	0.49 (0.06)	0.84 (0.01)
16	0.37 (0.05)	0.82 (0.01)
Best-worst	0.40	0.06

has a bigger impact here due to the potential for producing an OOV token and thus losing the meaning of the whole word, whereas the use of character embeddings can allow the network to e.g. treat 'like', similarly to 'lije'.

Focusing on the F1s here however misses an interesting phenomenon with the precision and recall. Table 7 summarises the figures (Prec = Precision, Rec = Recall), and shows that the precision of the positive class holds up whilst the recall is reduced, whereas with the negative class the recall holds up and even improves slightly whilst the precision drops, though this is not as big an effect as the drop in the positive class's precision. This effect can be seen both when using word embeddings and when using character embeddings, though as before the drop-off is more pronounced with the word embeddings.

6.4 Results for Classification with Parakweet Lab's Email Intent Dataset

Table 8 summarises the network architectures used here. As with earlier tables, the Network column lists a label for each network architecture and the Inputs column indicates the type and number of inputs to the network. The 'Early stop' and 'Best net' columns indicating whether early stopping was used or whether the best network (based on the validation set loss computed during training) was saved and used in testing. As before the 'Epoch duration' reports the training time in seconds for one complete presentation of the training data.

Table 7. Effect of spelling errors on precision and recall

Word embeddings				
Perturbations	Positive class		Negative class	
	Prec (stdev)	Rec (stdev)	Prec (stdev)	Rec (stdev)
1	0.77 (0.03)	0.70 (0.02)	0.85 (0.01)	0.89 (0.02)
4	0.75 (0.03)	0.51 (0.07)	0.78 (0.02)	0.91 (0.03)
8	0.73 (0.04)	0.34 (0.03)	0.73 (0.01)	0.93 (0.02)
12	0.69 (0.06)	0.24 (0.04)	0.70 (0.01)	0.94 (0.02)
16	0.70 (0.07)	0.14 (0.04)	0.68 (0.01)	0.94 (0.02)
Char embeddings				
Perturbations	Positive class		Negative class	
	Prec (stdev)	Rec (stdev)	Prec (stdev)	Rec (stdev)
1	0.81 (0.04)	0.69 (0.04)	0.85 (0.01)	0.92 (0.03)
4	0.84 (0.04)	0.57 (0.06)	0.81 (0.02)	0.94 (0.02)
8	0.86 (0.02)	0.42 (0.07)	0.76 (0.02)	0.96 (0.01)
12	0.84 (0.04)	0.35 (0.07)	0.74 (0.02)	0.97 (0.02)
16	0.86 (0.04)	0.24 (0.04)	0.71 (0.01)	0.98 (0.01)

Table 8. Networks used for sentence classification with Parakweet's email intent dataset

Network	Inputs	Early stop	Best net	Weights	Epoch duration
Char512	512 chars			197,352	1.4 s
Char512:ES	512 chars	Yes		197,352	1.4 s
Char512:BW	512 chars		Yes	197,352	1.4 s
Word89	89 words			55,877,752	9.2 s
Word89:ES	89 words	Yes		55,877,752	9.2 s
Word89:BW	89 words		Yes	55,877,752	9.2 s

As with the benchmark supervised sentence classification, 250 filters were used, the filter length was 3, the hidden layer consisted of 100 units and the networks were trained for 5 epochs with batch size 16 using the ADAM optimiser with a learning rate of 0.2 and a dropout probability of 0.2. Again we report the mean and standard deviation of the precision, recall and F1 over 5 runs for each configuration.

Note that because this is a much smaller dataset, the epoch durations are shorter as a result, but again the network size and runtimes are smaller for the character embeddings than for the word embeddings.

Table 9 details the results of experiments performed with this dataset. The columns are the same as in Table 4, but reporting for the Intent and Not Intent classes here. Note that the network labels sometimes end with ES (indicating use

Table 9. Results for sentence classification with Parakweet's email intent dataset

Intent			
Network	Precision (stdev)	Recall (stdev)	F1 (stdev)
Char512:ES	**0.83** (0.05)	0.49 (0.14)	0.60 (0.11)
Char512	0.71 (0.04)	0.67 (0.06)	**0.69** (0.02)
Char512:BW	0.61 (0.05)	**0.77** (0.04)	0.68 (0.02)
Word89:ES	0.79 (0.06)	0.59 (0.04)	0.68 (0.01)
Word89:250	0.68 (0.05)	0.67 (0.05)	0.67 (0.01)
Word89:BW	0.64 (0.04)	0.75 (0.04)	**0.69** (0.02)
No intent			
Network	Precision (stdev)	Recall (stdev)	F1 (stdev)
Char512:ES	0.81 (0.03)	**0.95** (0.03)	0.87 (0.01)
Char512	0.86 (0.02)	0.87 (0.03)	0.87 (0.01)
Char512:BW	**0.88** (0.01)	0.77 (0.06)	0.82 (0.03)
Word89:ES	0.83 (0.01)	0.93 (0.03)	**0.88** (0.01)
Word89	0.85 (0.01)	0.86 (0.04)	0.86 (0.01)
Word89:BW	**0.88** (0.02)	0.80 (0.03)	0.84 (0.01)

of early stopping) or BW(indicating use of the best weights seen during training). Early stopping increases the standard deviations whilst reducing the character embeddings mean F1 for the 'Intent' class, whilst using the best weights (assessed via validation loss) boosted recall at the expense of precision for the 'Intent' class and did the reverse for the 'No Intent' class, with the character embeddings F1 reduced in this case.

The best F1s achieved for character embeddings vs word embeddings are either the same (for 'Intent') or within a standard deviation of each other (for 'Not Intent'), again indicating that the two options give similar levels of performance.

7 Discussion

The results presented in the previous section indicate that a model trained using learned character embeddings as input alone can perform as well as a model trained using pretrained word embeddings in some short text classification tasks, and can do so using a much smaller network with far fewer parameters. Whilst more work is needed to determine to what extent this holds up for all text classification tasks, we're not the only ones to have found that character embeddings can be as effective as word embeddings [3]. So why would we get equal performance using word vs character embeddings? Some possible explanations come to mind:

(1) The tasks are so simple that the precise representation of the input does not greatly matter so long as it captures sufficient surface features of the sentence. For both tasks it's not obvious that this would be the case - both tasks intuitively rely on having some understanding of the meaning of a sentence. That said if there are surface/syntactic cues that correlate strongly with the task then the networks may simply be picking those cues up rather than performing the task. If so then it's a form of the explanation in point 2 in this list.
(2) The network is learning a surrogate of the tasks that is based on surface features that are present in both forms of input and which correlates with the tasks. If this is the case, we'd expect that constructing a large scale dataset for each task might be required to get the networks using features beyond the surface cues from the text, at which point we might then see a difference in performance. Moreover it may be that constructing a simple bag of ngrams or bag of words based classifier might perform the task just as well as these networks.
(3) The character embeddings, by capturing information about how characters are used, may be capturing some of the information the word embeddings capture, albeit indirectly. This seems unlikely to us but would be interesting to investigate to see if we can rule it out.

Some further comments:

- The pretrained word embeddings can provide information about the usage and meaning of words derived from a large corpus that you simply would not get from the sentences treated in isolation that we use for both tasks here. On the other hand the character strings also provide an advantage over the word strings in providing information about the morphology of words that you don't get from the word embeddings. These considerations suggest that some combination of word embeddings and character embeddings may boost performance, albeit at the expense of making the models larger again, because each form of input brings information that the other cannot bring. Of course if explanations 1 and 2 above apply then this would not apply.
- The nature of the task may well be important here. If the classification relies on the meaning of each word in the sentence being correctly deduced, we would not expect good performance. For example, we would not expect Named Entity Recognition to work well on character input alone because each word has to be labelled (to indicate if it is part of an entity or not) and thus the meaning of each specific word in the sentence is crucial to the labelling task. Could it be that when assigning a single label to a chunk of text that information from the surface form of the text is sufficient for most such tasks?
- Regarding the character model being more robust to spelling errors and OOV words than the word embedding model, this is intuitive but there are limits here. Handling of OOV words will only work to the extent that words with similar character strings have similar meaning/usage to a relevant in vocabulary word. Similarly, spelling mistakes will only be handled well to the extent

that the misspelled token does not represent another word that the model knows or is sufficiently similar to the correctly spelled word for the model to figure things out. If correct spelling becomes critical to the classification task then it may be better to deal with spelling errors prior to the classification being performed.

8 Conclusions and Future Work

We have presented a set of research questions (in Sect. 2) to answer when evaluating the use of character embeddings vs word embeddings as input to models performing NLP tasks that address both the performance of the classifiers in terms of precision, recall, F1 and in terms of the computational costs of training and deploying such models.

The results of the experiments we presented illustrate a clear win in favour of deploying the character embedding based model. Not only did we confirm that using character embeddings as input can be as effective as word embeddings when making predictions but the models are much smaller, and the computational costs are decisively lower - as confirmed by the fact that our hosting costs for the character model were 200x less than for the word2vec model. Additionally the character embeddings model is clearly more resilient, as evidenced by our experiments on introducing spelling errors to the sentences during evaluation. Also, it has shown that a training dataset of 19k in size can be reduced to 40% of its size with only modest loss of accuracy. However, the character models were slightly slower at generating predictions than the word embedding models.

This work is limited as follows:

- Both of the datasets employed are quite small, and the two tasks, both of which involve classifying sentences in isolation, won't represent the range of possible classification tasks. It may be that other text classification tasks may be more difficult when using character input vs word level input. Also it may be that for some tasks a larger dataset is needed for differences in performance to manifest themselves.
- We use pre-trained word embeddings derived from news articles, where perhaps using pre-trained embeddings derived from a corpus of Gluru users emails or from the Enron corpus might give better results by being better tailored to the domains of the two tasks.
- The comparison is limited to convolutional networks.
- Both tasks involve sentence classification, and it may be that for longer texts the use of character vs word embeddings might make more difference.
- The classification is performed independently of the context in which a sentence occurs.
- No attempt at combining the use of character and word embeddings has been made.

These limitations, and the discussion in Sect. 7, suggest the following lines of followup work:

- Investigate using alternative embeddings to the Google News embeddings, including using embeddings derived from a corpus of emails.
- Investigate the extent to which the tasks can be performed based on e.g. bag of words or bag of (character or word) n-grams, to help determine to what extent the tasks can be resolved via surface features.
- Extend the comparison to use LSTM networks. [18] and (bi-directional) LSTM networks [19] to see if the sequential nature of processing in such recurrent networks makes any difference to the issue.
- Extend the comparison to other text classification tasks and larger data sets, such as the Twitter Sentiment Analysis dataset[7] and the IMDB movie review dataset[8]. These datasets are both much larger datasets and the latter involves processing long multi-paragraph text rather than sentences.
- Investigate whether we can combine the use of character and word embeddings effectively to improve performance.
- Investigate whether using other sub-word level input, such as character n-gram embeddings, can be an effective strategy.

Acknowledgment. The authors would like to thank their colleagues at Gluru for their support during this work.

References

1. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS 2013, pp. 1–9, October 2013. http://arxiv.org/abs/1310.4546
2. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543 (2014). http://www.aclweb.org/anthology/D14-1162
3. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28 (NIPS 2015), pp. 649–657. Curran Associates, Inc. (2015). http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf
4. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**(6), 391 (1990)
5. Pang, B., Lee, L., et al.: Opinion mining and sentiment analysis. Found. Trends® Inf. Retrieval **2**(1–2), 1–135 (2008)
6. Silva, J., Coheur, L., Mendes, A.C., Wichert, A.: From symbolic to sub-symbolic information in question classification. Artif. Intell. Rev. **35**(2), 137–154 (2011)
7. Nakagawa, T., Inui, K., Kurohashi, S.: Dependency tree-based sentiment classification using CRFs with hidden variables. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 786–794. Association for Computational Linguistics (2010)

[7] http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/.

[8] http://ai.stanford.edu/~amaas/data/sentiment/.

8. Lendvai, P., Geertzen, J.: Token-based chunking of turn-internal dialogue act sequences. In: Proceedings of the 8th SIGDIAL Workshop on Discourse and Dialogue, pp. 174–181 (2007)
9. Lee, J.Y., Dernoncourt, F.: Sequential short-text classification with recurrent and convolutional neural networks. In: Proceedings of NAACL-HLT 2016 (2016). https://arxiv.org/abs/1603.03827
10. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) (2014). http://emnlp2014.org/papers/pdf/EMNLP2014181.pdf
11. Socher, R., Huval, B., Manning, C.D., Ng, A.Y.: Semantic compositionality through recursive matrix-vector spaces. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language processing and Computational Natural Language Learning, pp. 1201–1211. Association for Computational Linguistics (2012)
12. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (Almost) from Scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
13. Kalchbrenner, N., Grefenstette, E., Blunsom, P.: A convolutional neural network for modelling sentences. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, June 2014. http://goo.gl/EsQCuC
14. Santos, C.D., Zadrozny, B.: Learning character-level representations for part-of-speech tagging. In: Proceedings of the 31st International Conference on Machine Learning (ICML 2014), pp. 1818–1826 (2014)
15. Klimt, B., Yang, Y.: The enron corpus: a new dataset for email classification research. In: Boulicaut, J.F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) Machine Learning: ECML 2004. Lecture Notes in Computer Science, vol. 3201, pp. 217–226. Springer, Heidelberg (2004) https://doi.org/10.1007/978-3-540-30115-8_22
16. Cohen, W.W., Carvalho, V.R., Mitchell, T.: Learning to classify email into "Speech Acts". In: EMNLP 2004, vol. 4, pp. 309–316 (2004). http://acl.ldc.upenn.edu/acl2004/emnlp/pdf/Cohen.pdf
17. Conneau, A., Schwenk, H., Barrault, L., Lecun, Y.: Very deep convolutional networks for natural language processing. In: Proceedings of EACL 2017 (2017). https://arxiv.org/abs/1606.01781
18. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1–32 (1997)
19. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(56), 602–610 (2005). http://www.sciencedirect.com/science/article/pii/S0893608005001206
20. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: 3rd International Conference for Learning Representations (2015). http://arxiv.org/abs/1412.6980