RNNs vs CNNs for Sentence Classification

Student: Maksym Del; supervisor: Kairit Sirts

Abstract

Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) are widely used architectures nowadays. This project takes as a background previous work on empirical study of these architectures in comparison to different dataset, itself, and each other on sentence classification task and does following: (1) re-implement some models from both papers, (2) adds a new dataset to verify to what extend (some of the) conclusions stated in the papers generalize to a new domain, (3) implements several additional recurrent and mixed architectures, and evaluates them on both new and paper's data, (4) draws some additional notes based on results. We also publish code and data we used[1].

1 Backgournd

This project is based on two papers that present research on CNNs and RNNs for text classification.

The first paper [Kim, 2014] was highly influential work on applying CNNs for sentence classification. It showed that CNNs, that established their place as state-of-the-art in computer vision, can be also used for text data. In particular, they study how well CNNs perform among different sentence classification datasets and compare results to several other models from literature. Moreover, they show how pre-trained word embeddings influence performance of CNNs. They study initialization from pre-trained vectors with fine-tuning these vectors for specific dataset, initialization from pre-trained vectors with fixing these vectors and learning only remaining part of the network, and initialization with random vectors and tuning them. Among other things, results show that in most cases CNNs achieve the highest performance in comparison to some other methods, and that starting from pre-trained vectors and then fine tuning embedding weights systematically give the best performance. Datasets include SST (Stanford Sentiment Treebank).

The second paper [Yin et al, 2017] compares CNNs (in case of starting from random embeddings) with GRU and LSTM recurrent architectures among different Natural Language Processing tasks, including sentence classification. It shows that GRUs achieve the best performance on several datasets, including sentence classification SST dataset.

2 Introduction

Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) are widely used architectures nowadays. As usual feedforward networks they learn representation of the source text by minimizing loss function. Despite the high-level learning mechanism is the same, they learn representations based on different architectures. RNNs are designed to work with sequential data, so that they process information elementwise in a sequence, while CNNs are good and capturing some repetitive patterns by learning filters. We aim to see how different results can be having the same task, but these different architectures.

This project implements 3 CNN models from Kim (2014) (fixed, random, and tunable embeddings) as well as models similar to LSTM and GRU from Yin et al. (2017). Moreover, it adds simple ElmanRNN model, adds architecture advances to RNNs like bidirectional encoding, and averaging across all RNN hidden states (as opposite to taking just last hidden state), and inspects incorporating word characters information to RNNs by learning CNN features on word characters with explicit

[1] https://github.com/maxdel/rnns-vs-cnns

word segmentation. This project also studies the effect of pre-trained word embeddings not only on CNNs, like in work of Kim (2014), but also on RNNs, unlike in work of Yin et al. (2017). Lastly we train and tune our models on KSAI (Kaggle Spooky Authors Identification) dataset and compare results to the SST datasets.

3 Method

This project includes 12 different models. See Table 1 for their relation to the papers we are influenced by.

Model	Relation of the model to papers
boe-baseline	Additional
Elman-RNN-baseline	Additional
Elman-RNN-bi	Additional
Elman-RNN-bi-avg	Additional
LSTM-bi-avg	(Yin at al., 2017)
GRU-bi-avg-rand	(Yin at al., 2017)
GRU-bi-avg-static	Additional
GRU-bi-avg-nonstatic	Additional
GRU-bi-avg-charseg	Additional
CNN-rand	(Kim, 2014)
CNN-static	(Kim, 2014)
CNN-nonstatic	(Kim, 2014)

Table 1: Models implemented

Models with paper citation are derived from the papers we stated in the background. Additional entry means neither Kim (2014) nor Yin at al. (2017) studied this kind of model. CNNs models are almost the same as in original paper, while RNN models are different in 2 ways: (1) they use average among hidden state instead of just last hidden state as a sentence representation, (2) they are bidirectional. We made these changes because they gave us better performance on KSAI dataset we tuned our models at. Next, we describe each model separately. For all the models as pre-trained embeddings we use 100 dimensional glove vectors and as a final layer we use 1-hidden layer feedforward network with 10 hidden units at first layer and C hidden units in output layer, where C equals to the number of classes. Each time we say we project onto classes (labels), we mean pass through this network. Models use tunable glove vectors unless otherwise stated.

We did not try deeper RNNs since there are not enough data to learn more weights.

3.1 Model: boe-baseline

This model starts with pre-trained embeddings, allows to tune them, takes an average of the result dense vector, and projects result sentence representation vector onto classes. See Figure 1 for reference.

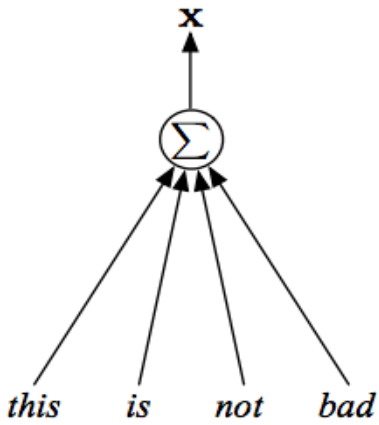

Figure 1: Bag of embeddings encoder
(The image is from oxford practical)

3.2 Model: Elman-RNN-baseline

Simple unidirectional Elman RNN. It reads embeddings word by word, and outputs a vector for each element while passing hidden state forward. Refer to Figure 2 for example. Finally, we take the last hidden state of the network, and project it onto classes.

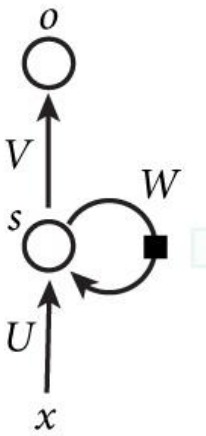

Figure 2: Elman RNN cell
(The image is from WildML blog post)

3.3 Model: Elman-RNN-bi

The same as above, but bidirectional. That is, it reads a word sequence from beginning to the end,

and then from end to the beginning. Encoder outputs for last state are then concatenated and passed next.

3.4 Model: Elman-RNN-bi-avg

The save as above, but here we use the average among all hidden states of the network instead of just the last one. See Figure 3 for a picture of encoder.

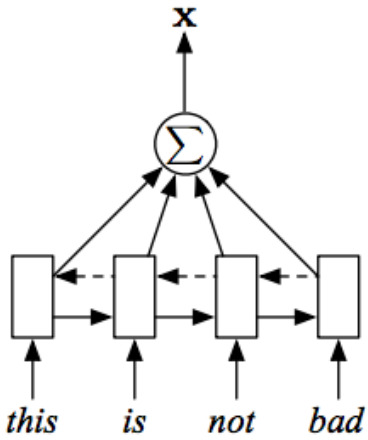

Figure 3: average of hidden states of bidirectional RNN is used as sentence vector representation
(The image is from oxford practical)

3.5 Model: LSTM-bi-avg

The same as above, but LSTM cell used. Refer to Figure 4 for a reminder of LSTM cell, and to the Figure 3 for the architecture.

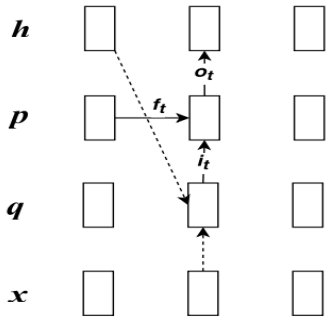

Figure 4: LSTM architecture
(The image is from Yin et al, 2017)

3.6 Model: GRU-bi-avg-rand

The same as above, but starting from random embeddings, and using GRU cell. See Figure 5 for a reminder of GRU cell.

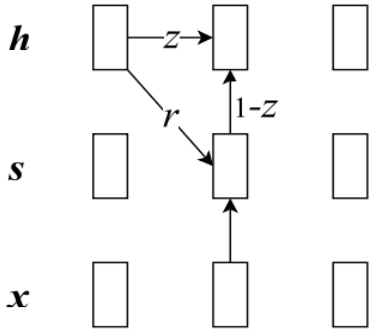

Figure 5: GRU architecture
(The image is from Yin et al, 2017)

3.7 Model: GRU-bi-avg-static

The same as above, but with fixed glove embeddings.

3.8 Model: GRU-bi-avg-nonstatic

The same as above but with tunable glove embeddings.

3.9 Model: GRU-bi-avg-charseg

The same as above, but with additional CNN over character over each word. Then character features for each word are concatenated with word embeddings and are used as word representations.

3.10 Model: CNN-rand

CNN that starts from random embeddings. See figure 6 for a reminder of CNN architecture.

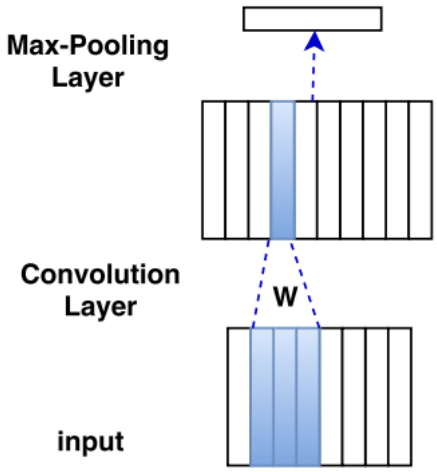

Figure 6: CNN architecture
(The image is from Yin et al, 2017)

3.11 Model: CNN-static

CNN that uses fixed glove vectors. See figure 6 for a reminder of CNN architec-ture.

3.12 Model: CNN-nonstatic

CNN that starts from glove vectors and allows for tuning. See figure 6 for a reminder of CNN architecture.

4 Results

4.1 Data

For our study we use 3 datasets: KSAI (Kaggle Spooky Author Identification) and two variations of SST (Standford Sentiment Treebank): 5 class sentiment prediction and 2 class variant of it. The first one contains sentences from stories by Edgar Allan Poe, Mary Shelley, and HP Lovecraft. The classes are authors themselves. The second one, SST1, is sentiment classification dataset with 5 sentiment classes. The third one SST2, is a binary version of SST1.

SST datasets contain ~10,000 sentences, while KSAI is almost 20,000 rows. 20% of the data is used as a dev set, and 20% for evaluation purpose.

SST1 and SST2 were used in paper by Kim (2014), while SST2 only was used by Yin et al. (2017). KSAI have not been used in the literature before.

4.2 Models performance

We trained all 12 models on datasets described above and measured final scores. The metric is accuracy. We used cross entropy as our loss function, and performed early stopping based on the dev set loss. Hyper-parameters were tuned on the KSAI dataset. Apart from the project code, we also publish full configuration files for all the models with instructions on how to reproduce[2]. Final results are presented in the Table 2.

First, it can be clearly seen that the simplest boe-baseline model performs best on KSAI dataset, while other more advanced architectures perform only as well as or worse. It suggests that for the author identification task the information about concrete words present is crucial, and the information about interactions between these words (which RNNs provide) only makes it harder for network to draw right conclusions. This simple baseline also achieves competitive accuracy on SST2 datasets, but performs weak on SST1. We think that it is just due to lack of hyper-parameters tuning on SST1.

Second, we can confirm results from Kim (2014), that initializing model with pre-trained vectors, and fine-tuning them systematically improves the performance of CNNs. It holds for our KSAI dataset as well. However, we find that this is not always the case for GRUs. An interpretation of this fact is out of the scope of this project so we left it for a future research based discussion. However, we make a note here, that our experiments, as well as experiments by Kim, (2014) are done on small datasets (up to 20,000 sentences). That is why it is not clear how these conclusions extend to tasks on a large scale (e.g. machine translation with millions of sentences).

Third, we verify performance results of Yin at al. (2017), and conclude that GRUs that start from random embeddings perform at least as well as CNNs that start from random emeddings. We, however, verify it on KSAI dataset as well for the pre-trained vectors case, both tunable and fixed. Note, that our RNN models show lower performance on SST datasets, then CNN models by Kim (2014), due to the fact that our RNN models were tuned KSAI instead. Nevertheless, by making this decision we show that models described by Kim, (2014) are robust to the input data.

Third, we can see that character level information gives very little improvement for SST datasets and no improvement on KSAI.

Finally, the Table 2 shows that LSTM outperform GRU only on SST2, while on SST1, and what is more important, on KSAI (where the models were tuned) GRU reliably gives results that outperform the other models. This confirms results by Yin at al. (2017).

Model	Dataset		
	KSAI	SST1	SST2
boe-baseline	**0.82**	0.23	0.82
Elman-RNN-baseline	0.79	0.39	0.82
Elman-RNN-bi	0.81	0.37	0.83
Elman-RNN-bi-avg	**0.82**	0.40	0.84
LSTM-bi-avg	0.78	0.43	0.86
GRU-bi-avg-rand	**0.82**	0.41	0.85
GRU-bi-avg-static	0.76	0.42	0.83
GRU-bi-avg-nonstatic	**0.82**	0.42	0.84
GRU-bi-avg-charseg	**0.82**	0.43	0.85
CNN-rand (Kim, 2014)	0.81	0.45	0.83
CNN-static (Kim, 2014)	0.76	0.46	**0.87**
CNN-nonstatic (Kim, 2014)	**0.82**	**0.48**	**0.87**

Table 2: Models accuracy

[2] https://github.com/maxdel/rnns-vs-cnns/tree/master/experiments_configs

5 Conclusions

By completing this project we implemented 12 different RNN and CCN models, and verified finding of Kim (2014) that CNNs perform well for text classification problems among datasets, and that using tunable glove embedding systematically improves the score. However, we showed that it is not always the case for RNNs, and made a note on dependence on dataset size. We also showed that simple baseline model performs almost as well as RNNs and CNNs, but it probably will not be the case for training corpora. We used a new dataset to see how robust the results are and made the code online.

References

Yin, Wenpeng; Kann, Katharina; Yu, Mo; Schütze, Hinrich, 2017. *Comparative Study of CNN and RNN for Natural Language Processing.* arXiv:1702.01923v1 [cs.CL]

Yoon Kim, 2014. *Convolutional Neural Networks for Sentence Classification.* arXiv:1408.5882v2 [cs.CL]