Comparing deep learning splel corrction approaches for a keyboard inaccurate typing

Kolinko Danylo

Introduction 1

Spell correction applications are mostly aiming at two problems. First is the user experience improvement, and second is the preprocessing of textual data. By this project, I tackled quite a different problem - accessibility. People who have cerebral palsy often cannot properly use the current keyboard. The first automated technique, which can relieve users from manual correction is spell correction, what they had to perform quite often due to cramps or just inaccuracies in the movements.

With a rise of DL was developed several new approaches to this problem. I would like to report results on several of them implemented and modified for keyboard neighboring mistypes:

- semi-character RNN
- character CNN
- seq2seq approach

Dataset 2

I discovered that available datasets are focused on the unigram correction, which is not optimal for the usage of RNN. Among them are WikiEdit Corpus and Birkbeck Spelling Error Dataset. Additionally, they are too small to train on.

So I chose to create my own dataset. For this purpose, I used large corpus that consists of concatenated files from the Gutenberg project - open-source book collection. Then to create target data, I used the nlpaug library, which has a rich collection of augmenters. I decided to create target data with KeyboardAugmenter that simulates keyboard errors.

Preprocessing 3

There was different preprocessing strategies for each of the models.

For the semi-character RNN tokenization was used. Than each word was smart character-wise embedded and fed to the network. For regressive modification, fasttext embeddings were used as target.

For the CharCNN tokenization was used as well, after that each word was one-hot encoded character-wise and padded to the same length.

For the seq2seq model no tokenization was performed, but text wrapped into fixed-length sequences which were encoded then by byte-pair encoding. the agglomerative method to split each word into common sub-words and numerically encode them.

semi-character RNN 4

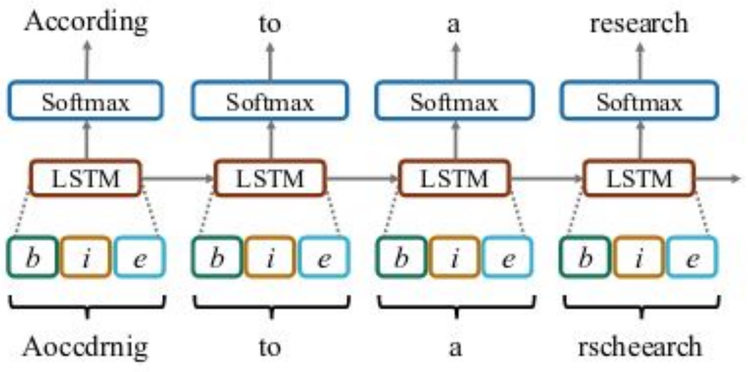
This model was a recreation of the Sakaguchi et al. 2017 paper that introduced a new deep learning approach to the spell correction problem.

Researches suggest that since for humans, spell correction is usually effortless, there might be a more intuitive technique for this problem then building a language and error model with generating candidates as exhaustive search on the n-edit space.

They also suggest that cognitive studies show that spell correction is so easy for humans, particularly in the jumbled letter problem (middle letters are randomly interchanged while first and the last stay at a place). Such as this little piece:

> *Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrodare, the olny iprmoetnt tihng is taht the frist and lsatltteer be at the rghit pclae. The rset can be a toatlmses and you can sitll raed it wouthit porbelm. Tihsis bcuseae the huamn mnid deos not raed ervey lteterby istlef, but the wrod as a wlohe.*

As model authors chose two stacked LSTMs with fully-connected layer on top, then Softmax is used and CrossEntropy as a loss function since the problem formulated as classification word to word.



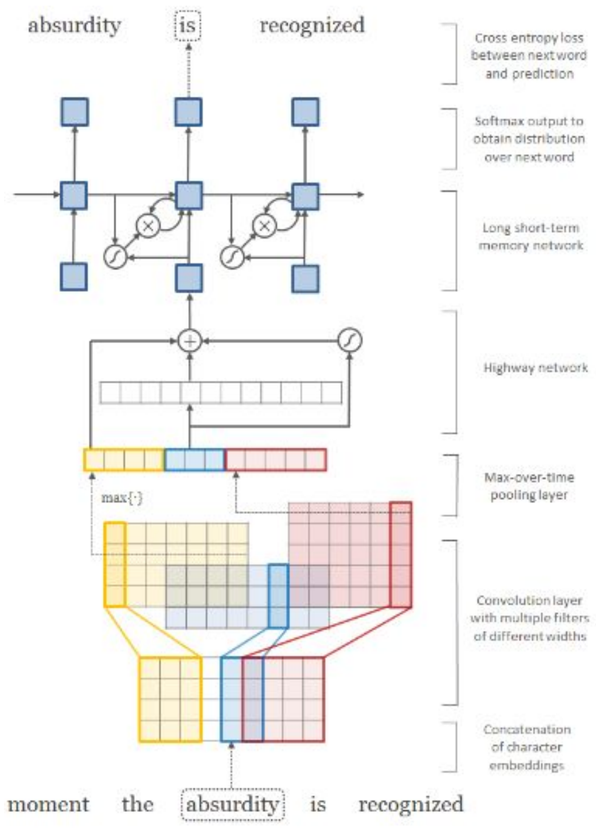
I decided to enrich the experiments a bit. With generated data, I chose to try three different recurrent cells: RNN, LSTM, and GRU. Also, the problem formulated as has one major drawback, since word dictionary can be extensive (40k words in my case) classification problem can be hard to learn.

So I thought of a different method, which is to use as a target, not the word itself but its embedding in the k dimensional space such as fasttext and reformulated classification problem as regression one. This approach can reduce the output vector substantially.

CharCNN 5

This model was a recreation of the Kim et al. 2015 paper.

As model authors chose the following architecture: convolution layer, then subjected to max-pooling layer, then processed with highway network. After this we have some context vector of the word which we can feed into two stacked LSTMs with fully-connected layer on top, then Softmax is used and CrossEntropy as a loss function.

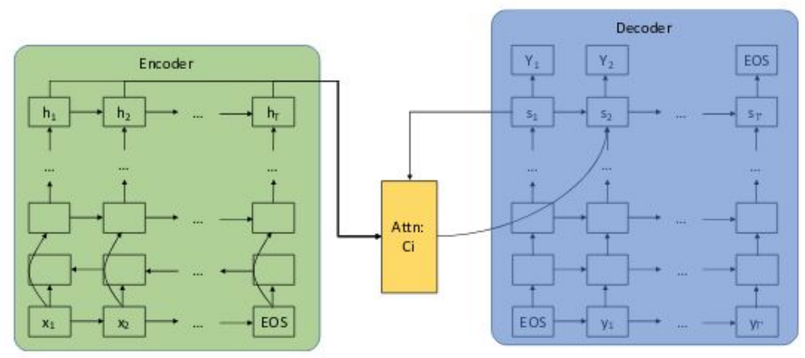


Similar to the memory cells in LSTM networks, highway layers allow for the training of deep networks by adaptively carrying some dimensions of the input directly to the output.

seq2seq 6

This model was a recreation of the Zhou et al. 2019 paper, which reformulated spell correction problem as machine translation one. This is a quite interesting approach, so I decided to give it a try.

This model consists of encoder and decoder in conjunction with an attention block. As target byte-pair encoded words used as well and masked cross-entropy loss used as the loss function.



Interestingly enough, this structure solves two significant problems of the previous architectures. First of all, the ~ 40k classification problem produces too large a target matrix, which may be hard to optimize, this problem is partially solved by BPE, since we can use the fixed size of sub-string dictionary (I used 1k dictionary).

The second problem is that output size may be different from input size, and this is perfectly reasonable for spell checking problems where concatenations of words are common. This problem is solved by autoencoder architecture.

Also, authors introduce the attention mechanism into this model, which I decided to test. So in the experiments I trained model with and without this block and compared them.

Results 7

	Modification	Accuracy
scRNN	classification + RNN	90.5 %
scRNN	classification + LSTM	91 %
scRNN	classification + GRU	92.7%
scRNN	regression + RNN	46.1 %
scRNN	regression + LSTM	53 %
scRNN	regression + GRU	49.8 %
CharCNN	-	72 %
seq2seq	-	7 %
seq2seq	+ attention	97.6 %

Conclusions 8

- Models were tested by sentences, where third of the words were contaminated, and among those words, each character was interchanged, deleted, or inserted a new one with 40% probability, a sequence which can be considered as a good benchmark.
- The results suggest that for keyboard mistypes problem, seq2seq outperforms every other model tremendously. The interesting fact is that the key part is attention block since without it autoencoder performs just at 7% accuracy level.
- I should add that seq2seq models require more time to train, and they are larger and therefore have a bit more computation to produce a result, thus longer reaction time. Nevertheless, the performance is impressive.
- Regarding the CharCNN model, we can see quite mediocre results, without apparent advantages.
- Now, the semi-character RNN as classification performs very well. The differences between the models are not substantial, which is quite interesting since RNN performs at a similar level to LSTM, which suggests that probably long term dependencies do not have a significant influence on this problem.
- Another finding is that proposed regression models perform poorly, probably due to the density of embeddings, which creates difficulties in coming back to word representation.

Applied Sciences Faculty, Ukrainian Catholic University, Lviv, Ukraine (kolinko@ucu.edu.ua)