Abstract

In this paper we evaluate several neural network architectures for error detection in English. For this task, we employ two different error tagsets based on the First Certificate in English (FCE) corpora: a simple binary tagset (correct or incorrect) and an enriched tagset as utilized by the Cambridge Learner’s Corpus (97 tags). The models we experiment with belong to the widely-adopted recurrent neural network family: Long Short-Term Memory networks (LSTMs) and Gated Recurrent Networks (GRUs). Though the LSTMs perform slightly better on the simple tagset and GRUs similarly on the enriched tagset, our results align with past literature in that there does not seem to be a clear difference between the two. We thus conclude that more training data is required as is a tagset that is simultaneously expressive and balanced.

1 Introduction

With an ever-increasing number of people learning and writing in English worldwide, there is a growing demand for natural language processing (NLP) systems that can process a learner’s writing and detect any mistakes present therein. The name of this task is error detection, which has applications in a variety of domains within computational linguistics and NLP. The most popular avenue likely lies in automatic essay scoring, where a writing response to a standardized test prompt (as presented in the GRE, TOEFL, etc.) is scored holistically per the writer’s use of grammar, spelling, style, etc. Another widespread application is grammar and style suggestion, which is concerned with aiding writers (natives and non-natives alike) in recognizing possible mistakes before the compilation of their final document. Other approaches include, but are not limited to, language learning systems, where learners are provided with linguistic feedback about the nature of their writing errors, and machine translation enhancement, in which the output of an automatic translation is corrected for grammaticality before being presented to the end-user (this task is often considered a standalone problem by the name of error correction). (Chodorow et al., 2012)

2 Related Work

Though neural models have been applied to a variety of NLP tasks to achieve state-of-the-art results, such as POS-tagging (Wang et al., 2015) and Named-Entity Recognition (Lample et al., 2016), the first notable extension of this method to the task of error detection was done by Rei and Yannakoudakis (2016). This work presented the performance of several neural network architectures, such as Recurrent Neural Networks (RNNs) (Elman, 1990) and Convolutional Neural Networks (CNNs) (Kalchbrenner and Blunsom, 2013), on the First Certificate in English dataset (FCE) (Yannakoudakis et al., 2011). Out of the surveyed approaches, a Bi-directional Long Short-term Memory network (LSTM) (Hochreiter and Schmidhuber, 1997) achieved the best performance with state-of-the-art results: an f0.5 score of 46.0. However, every method surveyed in this work produced highly competitive results, highlighting the effectiveness of deep learning in error detection.

In a follow up study, Rei et al. (2016) employed character-level representation in training an LSTM on erroneous sentences. The intuition behind this method was that character-level embeddings would compensate for the information lost in training solely on words - where every
unseen token maps to an <UNK> tag by default. This step (with the embeddings trained further on a bidirectional LSTM) allowed for token embeddings to be composed (instead of just generated) before being passed to another, word-level bidirectional LSTM. Ultimately, the character-level approach achieved slightly better results than the LSTM word-level task in Rei and Yannakoudakis (2016), producing another state-of-the-art benchmark. Though our paper does not evaluate a character-level system, it is nonetheless important to acknowledge the potential of this direction in future research.

3 Model

We evaluate four different models for error detection. Every model makes use of an embedding layer of size 100 that is initialized on the input text alone. The first model that we employ is a bidirectional LSTM of size 200, inspired by its performance in Rei and Yannakoudakis (2016). We expand on this model and create a deep LSTM by adding another such layer of identical size. In addition to this, we experiment with Gated Recurrent Units (GRUs) (Cho et al., 2014) in comparison to the LSTMs. Though both infrastructures have been reported to achieve comparable results in past literature (Chung et al., 2014), GRUs have never been employed for the error detection task. Thus, we conduct several experiments with GRUs in order to determine whether the distinction between the two models (particularly the GRU’s lack of a memory unit) accounts for any notable performance difference. The layers are constructed identically to the LSTM layers - a “simple” GRU of size 200 and a deep GRU network of two such layers.

Per (Rei and Yannakoudakis, 2016), we employ intermediary ReLU-activated Dense layers of size 50 in each of the described models in order to reduce the dimensionality of their respective layer outputs. These are also applied before output layer, which is softmax activated. We employ the binary and categorical cross-entropy loss functions to the simple and enriched models, respectively. We use a batch size of 64 sentences and the ADAM algorithm as an optimizer (Kingma and Ba, 2014).

4 Experiments

For each of the four models (LSTM, Deep LSTM, GRU and Deep Gru), we tested the development and the test set on both the simple and enriched data. This means we ran a total of 16 experiments (4x2x2).

5 Data

We employ the CLC-FCE dataset for this project. The dataset is comprised of 1,244 exam scripts, which were submitted by candidates applying for the Cambridge ESOL First Certificate in English (FCE) between the years 2000 and 2001. The dataset comes in two formats - simple and enriched - both of which we experiment with in the paper.

5.1 Simple Data

The simple dataset is binary: every word in the text is tagged as either correct or incorrect. This dataset consists of a total of 528,909 words across 33,673 sentences. Table 1 outlines the distribution of tags in the simple dataset.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct tags</td>
<td>393936</td>
<td>29959</td>
<td>35172</td>
</tr>
<tr>
<td>Incorrect Tags</td>
<td>58897</td>
<td>4640</td>
<td>6305</td>
</tr>
<tr>
<td>Total</td>
<td>452833</td>
<td>34599</td>
<td>41477</td>
</tr>
</tbody>
</table>

5.2 Enriched Data

The enriched data is encoded with much more information: errors are divided into different categories and the correction of the error is provided alongside the original token. We accounted 80% of this data for training, 10% for development and 10% for test. The distribution of correct and incorrect tags in this dataset can be found in table 2.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct tags</td>
<td>330893</td>
<td>41330</td>
<td>42032</td>
</tr>
<tr>
<td>Incorrect Tags</td>
<td>48113</td>
<td>6460</td>
<td>5494</td>
</tr>
<tr>
<td>Total</td>
<td>379006</td>
<td>47790</td>
<td>47526</td>
</tr>
</tbody>
</table>

The categories of the errors are coded following the Cambridge Learner’s Corpus guidelines (Nicholls, 2003). The first letter of the error tag represents the general type of error:

General types of error (first letter of tag):
• F wrong Form used
• M Missing word
• R word or phrase needs Replacing
• U word or phrase is Unnecessary
• D word is wrongly Derived

The second letter of the error tag represents the word class of the erroneous word. In Appendix A the full overview of the error tags can be found.

6 Pre-processing

6.1 Data reformatting

While the simple data did not require any pre-processing, the more complex data called for some formatting. In order to match the simple data, we converted the XML data of the enriched dataset to the CoNLL-U format, where every line contains a tab-delimited word and tag. In some cases, an error tag did not apply to just one word, but to multiple words. In these cases, all words that belonged to the same error were encoded on a separate line with the correct error tag. However, the tag was expanded with the string '_span' in order to capture the fact that the words belong to the same error class. The representation of the sentence "If you have some free time for a you’d better visit Kamakura City" in the processed data can be found below:

if CORRECT
you CORRECT
have CORRECT
some INCORRECT_MQ
free CORRECT
time CORRECT
for INCORRECT_U_span
a INCORRECT_U_span
you’d INCORRECT_R
better INCORRECT_R
visit CORRECT
Kamakura CORRECT
City CORRECT

In this example, the words 'for a' are unnecessary, thus they get the tag 'U'. To represent the fact that they were tagged as part of the same error, '_span' is attached to the tag.

6.2 Data pre-processing

In fitting our data to the models, we lowercased every token and removed all non-alphanumeric characters. The intuition behind this was to keep all errors lexically-based, as issues behind comma and semicolon placement can sometimes be subjective. We experimented with using the pyenchant spellchecking library\(^1\) in order to automatically detect mispelled words and match them with a spelling error tag. However, this step lowered performance significantly and was thus discarded. Lastly, we encoded every word in the vocabulary with a dictionary index and converted the labels to one-hot format. Every sentence was then padded to the length of the longest sentence and passed as input to the models.

7 Evaluation

Table 3: Baseline accuracy scores

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Most frequent tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple data</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>Complex data</td>
<td>0.84</td>
<td>0.88</td>
</tr>
</tbody>
</table>

We compare the results of our experiments to two Baseline scores. The first is obtained while running a simple structured perception. The second baseline score is the score obtained when tagging every word as correct, because the correct tag is significantly more frequent than the incorrect tag. Table 3 demonstrates that the baseline scores are quite high: due to the skew of the data towards correctness, it is easy to obtain a high accuracy score. However, this does not produce a reliable picture of the performance of the model. Therefore, we decide to calculate F\(_{0.5}\) scores over the erroneous words, since this is the category we are interested in. Since high coverage (represented by recall) is not vital in this task, setting \(\beta\) to 0.5 places more emphasis on precision. We believe this measure will give a better and more reliable insight of the performance of the model than the accuracy score. (Chodorow et al., 2012).

\[
F_{\beta} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{1}
\]

8 Results

The results of the 16 experiments we ran can be found in table 4. In line with (Rei et al., 2016), [http://pythonhosted.org/pyenchant/](http://pythonhosted.org/pyenchant/)
Table 4: Results

<table>
<thead>
<tr>
<th>Simple Data</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.56</td>
<td>0.17</td>
</tr>
<tr>
<td>Deep LSTM</td>
<td>0.48</td>
<td>0.18</td>
</tr>
<tr>
<td>GRU</td>
<td>0.45</td>
<td>0.19</td>
</tr>
<tr>
<td>Deep GRU</td>
<td>0.43</td>
<td>0.23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enriched Data</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.23</td>
<td>0.06</td>
</tr>
<tr>
<td>Deep LSTM</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>GRU</td>
<td>0.21</td>
<td>0.08</td>
</tr>
<tr>
<td>Deep GRU</td>
<td>0.16</td>
<td>0.06</td>
</tr>
</tbody>
</table>

the single LSTM performs better than the deep LSTM on development and test for both the simple and enriched data. This could be due to the fact that adding another hidden layer confuses the relevant features learned by the first LSTM or that the dimensionality reduction by the intermediary dense layer loses some of this information. The same cannot be said for the GRU-based systems, however, as the deep GRU outperforms the simple system on the simple tagset. Generally, the difference in LSTM and GRU’s performance appears very slight, though it can be said that the GRU systems tend to generalize better against the enriched tags while the LSTM fares better with the simple tags. Predictably, all systems perform poorly on the enriched tagset, which is considerably sparse and not normally distributed.

9 Conclusions and Future Work

Our results are disappointing in that none of our systems achieve the state-of-the-art performance described in (Rei and Yannakoudakis, 2016). One potential avenue for improvement in this regard is to train for more than 10 epochs and initialize our embedding layer with pre-trained word embeddings. Furthermore, (Rei and Yannakoudakis, 2016) demonstrated a drastic increase in their systems’ performances when they were trained on more data. This could certainly be extended to our system, given that appropriate data is made available and is properly annotated. In this case, it would be interesting to see whether the GRU systems’ performance on the enriched tagset would improve or worsen.

Our experiments also reveal the need for a proper standardized tagset in training an error detection system. It is not enough to categorize mistakes in a binary fashion, as the class of error remains too abstract for use in any end-application. Conversely, the CLC error tags are far too verbose and too sparsely distributed for any classifier to reasonably train on. An interesting future experiment could be concerned with only extracting the general error types (e.g. F, M, R, U, D) and evaluating a system against them accordingly.
Appendix

A  Error tags in the Complex Data

General types of error (first letter of tag):

- F wrong Form used
- M something missing
- R word or phrase needs Replacing
- U word or phrase is Unnecessary
- D word is wrongly Derived

Word classes (second letter of tag)

- C conjunction
- D Determiner
- J Adjective
- N Noun
- Q Quantifier
- T Preposition
- V Verb
- Y adverb
- P Punctuation error
- A Pronoun

Additional error codes

- Agreement errors (AG + word class)
- Countability errors (C + word class)
- AS incorrect Argument Structure
- CE Compound Error
- CL CoLocation error
- ID IDiom error
- IN Incorrect formation of Noun plural
- IV Incorrect Verb inflection
- L inappropriate register (Label)
- S Spelling error
- SA American Spelling
- SX Spelling confusion error
- TV wrong Tense of Verb
- W incorrect Word order
- X incorrect formation of negative
- CN countability of Noun error
- CQ wrong Quantifier because of noun countability
- CD wrong Determiner because of noun countability

B  Division of Work

Code:

- Processing Complex data Josine
- Perceptron Baseline Artur
- Basline most frequent tag Josine
- LSTM Artur
- Evaluation Artur & Josine

Writing:

- Introduction Artur
- Related work Artur
- Model Artur
- Experiments Josine
- Data Josine
- Pre-processing Artur & Josine
- Results Artur & Josine
- Conclusion & Future work Artur Josine

References


