# Grammar Correction Model Using Sequence-to-sequence Neural Machine Translation

CS249 Final Project Report

Puchin Chen
404874406
puchinchen@ucla.edu

Cindy Fan Chiang
105025071
hfchiang@ucla.edu

Evelyn Chen
704332587
chenyiying@ucla.edu

## ABSTRACT

We aim to create an automated grammatical error detector and corrector that could ease the pain of English writing and learning. Recent study of sequence-to-sequence learning algorithms could lead to such writing support. Using a a sequence-to-sequence neural machine translation model with attention, we experimented with two types of embedding methods. In addition, we add in error tags as additional information to help the model learn better from the error type. For this project, we create a grammar checker that could take in sentences with grammatical errors and return an edited, grammatically correct sentence. The implementation for this project is in this Github repository [1].

**ACM Reference Format:**
Puchin Chen, Cindy Fan Chiang, and Evelyn Chen. 2018. Grammar Correction Model Using Sequence-to-sequence Neural Machine Translation: CS249 Final Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Machine translation is about converting a text from a language to another. It is one of the original goals for the creation of computers. Classifical machine translation methods involve rules to convert a source language to a target languages. The rules are developed by linguists, who define the rules at the lexical, syntactic, and semantic level. However, these rule-based machine translation methods require the expertise to develop the rules. On top of the man power required for the rule-based methods, there are too many rules and exceptions that need to be defined. This method is soon replaced by a more automated idea, solving problem with math.

Using statistic, statistical machine translation soon outperformed the performance of rule-based machine translation and replaced its place. The use of statistical models help translate the text, choosing a source $S$ to maximize the probability $Pr(S|T)$ of the source given a target $T$. An advantage of statistical machine translation is that it is data driven, as it only requires a corpus of examples. Thus making it free of the need of extensive knowledge about linguistics or handcrafted grammars from human experts. However, since it depends on the target given to calculate the probability, it has a narrow focus on the phrases being translated. This downside of statistical machine translation requires careful tuning of each model, making it time and power consuming to train one model for a specific type of translation.

With the the ability to train directly on source and target in a single system, Neural machine translation became the new focus of machine translation. Improved from the statistical machine translation, neural machine translation does not require pipeline of specialized system. It is end-to-end and requires only one model for the translation. A major part of the framework is the encoder-decoder network [7], a multilayer perceptron neural network model. The encoder-decoder architecture can encode the source text to internal fixed length representation called context vector. However, an disadvantage of the encoder-decoder network is that it can only work with up to a certain length of input. If the given source is too long, the model would fail to predict the best target as output.

To improve on encoder-decoders, attention [2] can be used for the model to place attention on the input sequence as each word of the output sequence is decoded. The encoder-decoder recurrent neural network is currently the state of the out for machine translation. This paper will focus on using sequence-to-sequence neural machine translation using encoder-decoder neural network with attention weights. Two improvements will be made to this model, including adding error tags and using ELMo embeddings.

## 2 BACKGROUND

Our model is adapted from CoNLL 2013 and 2014 shared tasks, focusing on grammar correction. However, grammatical error correction task gained popularity even before it was presented by CoNLL. In 2012, similar Helping Our Own (HOO) shared tasks in 2011 and 2012 gained numerous participants and submissions. Even with such success, the performance of these grammatical error correction methods was not as desired. Many grammar correction tools still rely on expert-defined rules and if-else statements to check and correct essays and sentences. As noted in discussion of the overview paper of CoNLL 2013 [1], lacking of a manually and corrected corpus of English learner texts is one of the main obstacles for grammar correction task because there lacks a way to compare the performance on a common benchmark test. For this reason, CoNLL 2013 released NUCLE dataset to fill the purpose. To ease the way in for the community, the CoNLL 2013 shared task focused on a small list of error types. With an initial success in the shared task, the CoNLL 2014 shared task then required the participating systems to correct all errors in an essay.

[1] https://github.com/michaelchen110/Grammar-Correction


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In CoNLL 2013 shared task, the common approach to the grammar correction problem is to train a classifier for each error type. Some additional approaches included translation approach and language modeling approach. Since the error types increases in CoNLL 2014 shared task, most teams decided to have the models learn to fix all error types at the same time. As a result, hybrid systems were built to make use of several methods identifying and correcting the errors simultaneously. As the overview [3] indicated, the most popular approach to non-specific error type correction was the Language Model (LM) based approach. All the teams include the LM based approach along with other components in a hybrid system. Another popular approach in CoNLL 2014 is phrase-based statistical machine translation (MT), which differs between teams in terms of tuning. For specific error type, rule-based (RB) was commonly applied to identify specific grammatical error. In our experiment, we focus on the sequence-to-sequence model with attention weights. With the error types provided to the model during training time, we want to allow the model to learn from the additional information and be able to target each error type when predicting.

Lately, a deep contextualized word representation called ELMo [4] gained popularity and attention as it captures both semantics and syntactics meanings of the words. With the deep bidirectional language model (biLM) in the framework, ELMo is pre-trained on a large text corpus, giving it advantages in terms of background information of words. In our experiment, we want to see if replacing the word embedding with ELMo could improve the performance of the leading model from CoNLL 2014.

## 3 DATASET

The dataset we use is from CoNLL-2013 and CoNLL-2014 shared tasks. CoNLL-2013 dataset has annotated dataset with wrong sentence, corrected sentence, position of mistake, and type of error. Specifically, it has five types of errors, including article or delimiter, preposition, noun number, verb form, and subject-verb agreement.

In CoNLL-2013, participating teams obtained a NUCLE training dataset with 1397 essays which has 57,151 sentences with around 1 million word tokens.CoNLL-2014 is similar to CoNLL-2013 that it has an annotated dataset with ungrammatical sentence, grammatically correct sentence, and the types of mistakes. Different from CoNLL-2013, this dataset has more types of errors. It has 28 types of errors.

Since CoNLL shared tasks only released their training data to the participating teams, we used their test data as our training data. The test data was created with 25 non-native speakers of English. 50 essays on two topics were written by these 25 students recruited by NUS. With the 50 essays as source, an English native speaker was responsible for providing error annotation on the test essays. In the test data from CoNLL-2013, there were 1,381 sentencs with 29,207 word tokens. In the test data from CoNLL-2014, there were 1,312 sentences with 30,144 word tokens. We combined the test data from the two shared task as our train data.

## 4 MODELS

### 4.1 Parser

In order to utilize the CoNLL 2013 and 2014 dataset, we developed our own parser to preprocess the data into our desire format. From the CoNLL shared task official website, we obtained the test data from CoNLL 2013 and 2014. In CoNLL 2013, there are two sets of test data, one with only five error types and the other contains all twenty-two error types. In CoNLL 2014, we got two versions of test sets, both with all twenty-two error types. The annotation of the CoNLL test set is similar to a regular html tag. We developed our own tagger parser to create training set with our desire format for our models to learn. For training set, we combined the all-error-type test set from CoNLL 2013 and the two versions of all-error-types from CoNLL 2014. The final training set we obtained from CoNLL are one set of five-error-types training set and one set of all-error-types training set.

The baseline train set we processed from the parser indicates the grammatical error with a deletion tag $< del > \ldots < /del >$ and add in the correction with an insertion tag $< ins > \ldots < /ins >$. The original test data from CoNLL is presented in the form of a sequence of essays with a number of grammatical corrections noted below. The corrections and the essays are documented separately. Our parser insert the corrections into the essays, and only keep the sentences with annotations. The final format of the training set for our model has the format of [original sentence]/t[corrected sentence].

One of the task we aim to experiment on is error type tag. With the hypothesis that adding error type tags would provide the machine more information to learn, we want to see if having the error tags would improve the performance of our model. To add in the error type tag, we extracted the error type from the CoNLL annotation and attach the information with the deletion tag. For instance, if the error type is subject verb agreement, the deletion tag would be $< del >< SVA > eat < /SVA >< /del >$.

### 4.2 Base Model

The base model is a sequence-to-sequence neural machine translation model with attention, adapted from Schmaltz et al [6]. The model does not use any of the rules for detecting error types nor use any individual classifier for human error subsets. In the model, two recurrent neural networks work together to transform one sequence to another. An encoder network condenses an input sequence into a vector while a decoder network unfolds the vector into a new sequence. Character-level convolution network from Zhang et al. [7] is used for as the encoder. Using this as baseline, we hope to reconstruct the structure and improve the model from there. For our baseline, we kept the original framework of the model from Schmaltz, with character-based encoder-decoder.

The sequence-to-sequence encoder is an recurrent neural network (RNN) that outputs some value for every word from the input sentence. For every input word, the encoder outputs a vector and a hidden state. Then, the hidden state is used for the next input word. The diagram for the encoder is shown below.

There are attention weights added to the sequence-to-sequence model. Attention allows decoder to focus on different part of the encoder's output for every step of decoder's output.

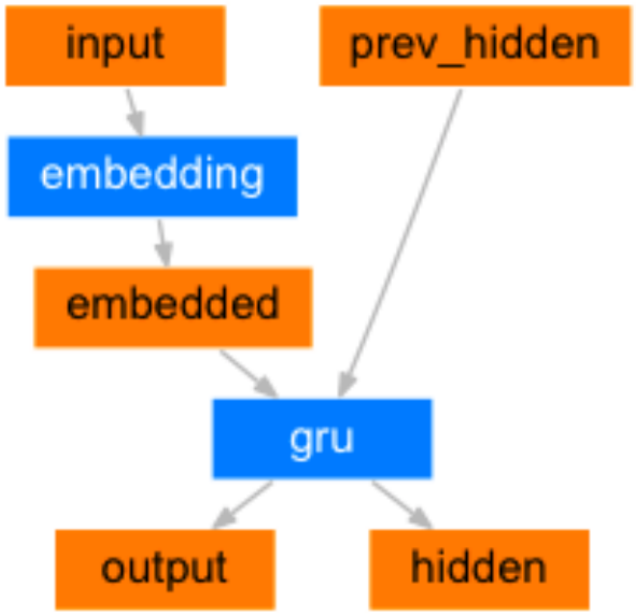


**Figure 1: Encoder**

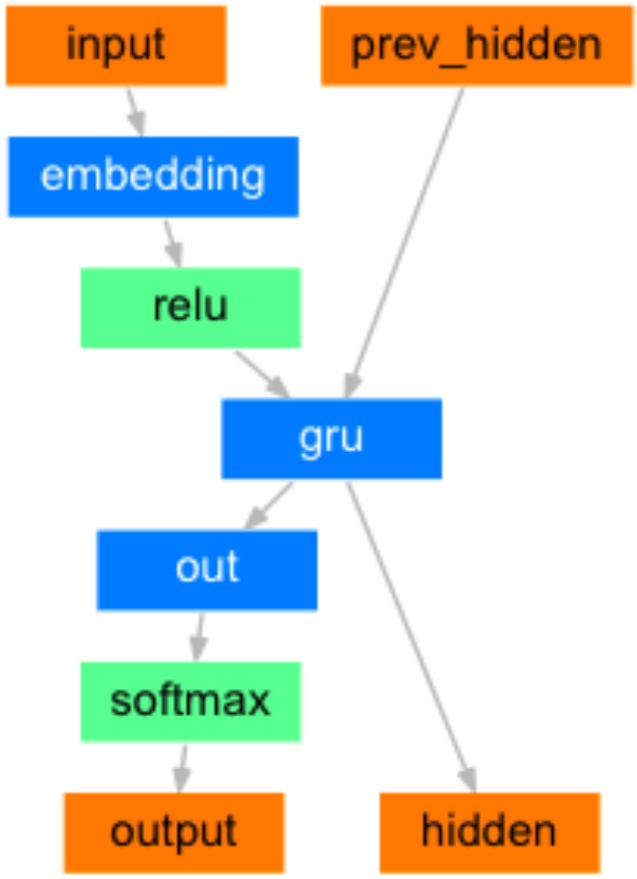


**Figure 2: Decoder**

### 4.3 Proposed Models

There are two proposed models in our study. These two improvements are both based on top of the base model, which is sequence-to-sequence neural machine translation. The first proposed model adds error tags around the edits for the sentence in addition to the insertion and deletion. The second possible improvement to the base model is to use ELMo as the word embeddings.

For both proposed models, a combined data of CoNLL-2013 and CoNLL-2014 is used. There are 28 errors for the combined data. One possible problem is the size of the data set. Since the dataset is small with several types of errors, there could be noises interfere with the model training. In the future, we could also experiment with using data with only 5 error types.

*4.3.1 Proposed Models 1: Adding Error Tags.* The original output of the sequence-to-sequence model is to have corrected sentence with insertion and deletion tags. Here, we add error tags to identify the error type of the annotation. The motivation of this proposed model is to further analyze the types of mistakes in the sentence and further use the error tags in the dataset to improve on grammar correction. The sentence below is an example of the corrected sentence having the additional error tags. The error type is SVA, which is subject verb agreement in this case.

Example:

The models $< del >< SVA > works < /SVA >< /del >< ins > work < /ins >$

*4.3.2 Proposed Models 2: Adding ELMo Embeddings.* The second proposed model is to add ELMo embeddings to the sequence-to-sequence neural machine translation model. ELMo embeddings, from Peters et al. [4], are deep bidirectional language model. It is context dependent and has single layer deep bidirectional language model (biLM). With multi-layer representations, ELMo has certain layers with the potential to perform better than others. More details on ELMo embeddings will be described in the experiment section.

## 5 EXPERIMENTS

For the experiments, we implemented two proposed model as well as the baseline model. We divided the project into two phases, preprocessing and training. In preprocessing, we implemented a CoNLL dataset parser to parse from CoNLL-2013 and CONLL-2014 to our desired format, described in section 4.1. With the data processed, we feed it to ELMo embedding file, which uses Torch 0.3.1 and AllenNLP environments. In training, where sequence-to-sequence attention framework is used, we used Torch 0.4.0 for implementation.

### 5.1 Preprocessing

*5.1.1 Error tag.* With the CoNLL parser, we convert the data from the stand-off annotations, the SGML format, into our desired format explained in section 4.1. The data set is from CoNLL-2013 (5 error types) and CoNLL-2014 (28 error types). We read in the stand-off annotations and directly insert the <ins><del> tags to

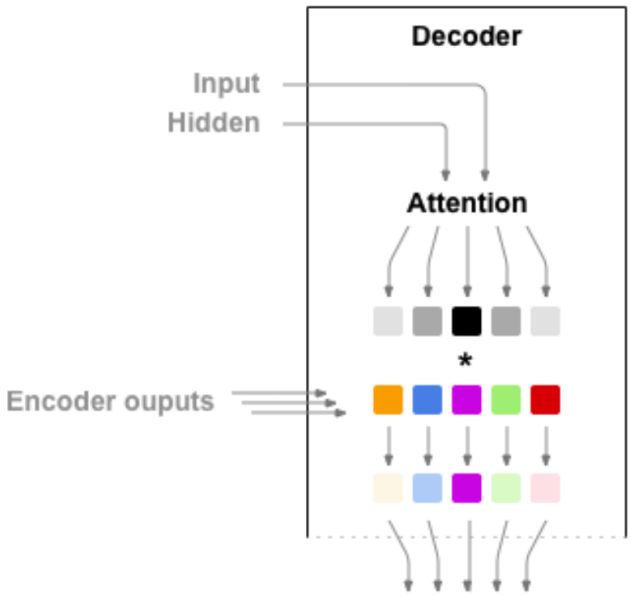


**Figure 3: Attentional Decoder**

sentence with tag for error type like <SVA> or <VForm>. A detailed format of original and preprocessed format are shown below.

| Original format | Preprocessed format |
|---|---|
| <DOC nid = "2"><br><TEXT><br><P><br>The original sentence go like this.<br></P><br></TEXT><br><ANNOTATION teacher_id="6"><br><MISTAKE start_par="0"<br>start_off="3" end_par="0"<br>end_off="9"><br><TYPE>ArtOrDet</TYPE><br><CORRECTION>the<br>modern</CORRECTION><br></MISTAKE> | The original sentence go like this. \t The original sentence <del> <SVA> go </SVA> </del> <ins> goes </ins> like this. |

**Table 1: CoNLL Dataset Parsing**

*5.1.2 ELMo embeddings.* For preprocessing of ELMo embeddings, the usage of ELMo embeddings is shown below in the code snippet. ELMo and batch-to-ids are imported from AllenNLP. Options and weight file are fed from Amazon AWS. ELMo has parameters of number of layers and dropout. Here, we specify two layers: In the sentence variable, we have a pair of sentence, where each sentence is a vector of word tokens. "I loves NLP." is the grammatically wrong sentence and "I love NLP." is the corrected sentence. Then, the sentences are converted to character IDs with batch-to-ids. By feeding the newly formed character ID to elmo embeddings, we get a ELMo representation. The layer number can be specified and further used in the model. To get the input and output tensor, we call the embedding with specified layer and get the data from there.

```
from allennlp.modules.elmo import Elmo, batch_to_ids

options_file = "https://s3-us-west-2.amazonaws.com/allennlp/models/
weight_file = "https://s3-us-west-2.amazonaws.com/allennlp/models/e

elmo = Elmo(options_file, weight_file, 2, dropout=0)

sentences = [['I', 'loves', 'NLP', '.'], ['I', 'love', 'NLP', '.']]
character_ids = batch_to_ids(sentences)
elmo_emb = elmo(character_ids)['elmo_representations']

layer = 1
input_tensor, output_tensor = elmo_emb[layer].data
```

**Figure 4: ELMo embedding usage**

Below is a representation of a float tensor for the sentence "I loves NLP.". It is of the size 4x1024. The four comes from the four tokens in the sentence.

One point to note here about ELMo embeddings is that ELMo is context aware. This means that two same words in different sentences will generate different word embeddings. The first diagram above shows the ELMo embeddings for "I loves NLP.", where each row represents each word. The second diagram below show the embeddings for "I love NLP." Even though the last word, "NLP", are the same word, the last row of both embeddings are not the same. This is because ELMo embedding is context dependent, and the embedding of each word depends on other words in the sentence as well.

```
In [7]: # input: I loves NLP .
   ...: input_tensor
Out[7]:

-1.2254e+00 -5.6004e-01 -1.7601e-01  ...
-3.3404e-01 -4.4247e-01 -3.7658e-01  ...
-2.3623e-01 -2.5916e-01  3.3226e-03  ...
-1.1192e+00 -4.8915e-01 -7.5710e-01  ...
[torch.FloatTensor of size 4x1024]
```

**Figure 5: ELMo embedding for "I loves NLP ."**

Since AllenNLP and sequence-to-sequence model use different versions of Torch, we will need to do some adjustment to fit ELMo embedding into the sequence-to-sequence model. To make the changes, we first created the embeddings and save them as elmo embeddings. Then, we read in the elmo embedding and fit it in the sequence-to-sequence model.

```
In [8]: # output: I love NLP .
   ...: output_tensor
Out[8]:

-1.2254e+00 -5.6004e-01 -1.7601e-01  ...
-3.3790e-01 -4.4750e-01 -2.5744e-01  ...
-2.8961e-01 -2.3701e-01  3.7949e-02  ...
-1.0717e+00 -4.3787e-01 -7.6095e-01  ...
[torch.FloatTensor of size 4x1024]
```

**Figure 6: ELMo embedding for "I love NLP ."**

*5.1.3 Training.* The environment and training details are shown below.

| | |
|---|---|
| Environment | • Torch == 0.3.1 for storing ELMo<br>• Torch == 0.4.0 for loading ELMo<br>• CUDA version: 9.2 from lab server |
| Training | • Add <tag> to dictionary at training time<br>• 3000 sentence pairs<br>• 75,000 iterations<br>• Randomly select sentences to train in each iteration |

**Table 2: Training Details**

## 6 RESULTS

### 6.1 Error Tags

Below is a plot of training loss against number of iterations for the first 1,000 iterations. Training loss with and without error tags are compared against each other. The plot shows our model performs slightly better with error tags than without error tags.

The chart below shows the results for sequence-to-sequence neural machine translation model with and without error tags. Below is one of the translations we got from the input incorrect sentence to the output corrected sentence with error tags. The ungrammatical sentence here is "However, from the ethical point of view." The corrected sentence should be "However, from the ethical point of view," with the difference in the last punctuation mark. In the first column for the result with error tags, the predicted sentence is "However, from the ethical point of view <del> <mec> . <EOS>". This prediction has the deletion tag at the right place and has the correct error tag, <mec> for punctuation. On the right side, the prediction using the baseline model returns "<del> <del> <del> . <EOS>". Here, the deletion of period is correct, but the words in the sentence are not shown in the prediction. Although the model with error tags for this example still lacks the insertion tag and some closing tags like </del> and </mec>, by comparing with the baseline model (without error tags), it is evident that the model with error tags produce better result.

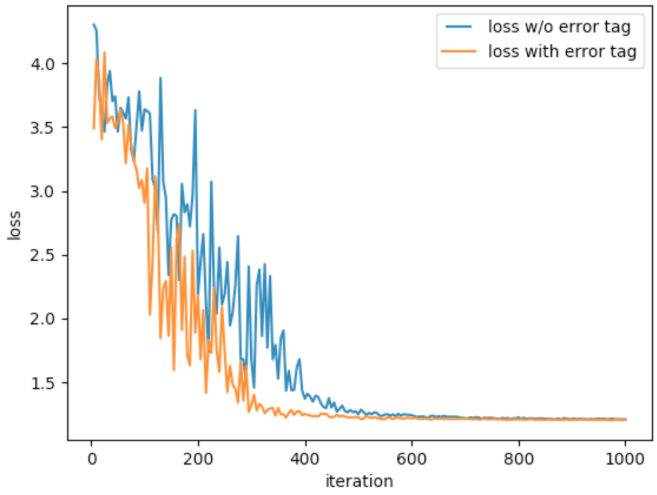


**Figure 7: Training loss w v.s w/o error tag**

| With error tags | Without error tags (baseline) |
|---|---|
| **Input:** However, from the ethical point of view .<br><br>**Predict:** However, from the ethical point of view <del> **<mec>** . <EOS><br><br>**Correct:** However, from the ethical point of view <del> **<mec>** . **</mec>** </del> <ins> , </ins><br><br>**<mec>**: Spelling, punctuation, capitalization, etc. | **Input:** However, from the ethical point of view .<br><br>**Predict:** <del> <del> <del> . <EOS><br><br>**Correct:** However, from the ethical point of view <del> . </del> <ins> , </ins> |

**Table 3: Sample output with additional error tag**

### 6.2 ELMo embeddings

The results for ELMo embeddings are not ideal. One possible reason is that we use the pretrained ELMo. In our dataset, all <tag>s are treated as <UNKNOWN>. Therefore, these tags are likely to become useless noise. We will need to train the embeddings ourselves first in order to resolve this problem. Even if we do so, a remaining concern is the size of the dataset. We think that even if we train the embeddings first ourselves, our dataset may be too small to train a good representation.

We have tried two existed neural machine translation libraries. One is OpenNMT, and the other one is the tutorial of seq2seq for NMT on PyTorch tutorial [5]. Since OpenNMT only provides word2vec and glove embeddings, we spent much time understanding their framework. However, we sadly found that we need to

adjust the whole library in every details in order to fit the model, which is impossible to finish within a short period of time. We have then tried to use NMT code from PyTorch official website. This direction is more promising, but we had wasted too much time on previous library. Although the integration of ELMo is almost finished, we still need a few days to let the code work properly. Therefore, we will try our best to finish them even after the due date for the project. If there is any progress, we will email the updated results to professor.

## 7 FUTURE WORK

There are some possible directions to go for future work. First of all, a bigger dataset can be used. CoNLL-2013 and CoNLL-2014 have relatively small dataset. Our study uses data from these shared tasks because they have clearly labeled error types. Currently, there aren't other dataset out there with grammar correction data that have error types. In the future, a possible improvement is to find a better dataset with more data, which hopefully can produce better results.

Since ELMo has many layers of representations. Another possible improvement would be try different layers of ELMo representations and compare the performance on using multiple layers or specific certain layers and see the difference in improvements in adding these different layers of ELMo to the sequence-to-sequence model.

A simple sequence-to-sequence model is used here, but this requires many manual coding and tuning. Another possibility would be using OpenNMT as a base for the experiment. From OpenNMT, LSTM and character model are also used. This can also be used to compare with simple sequence-to-sequence neural machine translation model.

Lastly, this study can be extended by adding a new component of multitask learning to this study. In addition to just translating the ungrammatical sentence to a corrected sentence, we can also classify the error. With multitask learning, we are able to perform two tasks at the same time, which are to predict the output (corrected sentence) and to classify the error type.

## 8 DISCUSSION

| | ERROR TAGS | ELMO EMBEDDINGS |
|---|---|---|
| PRO | Gives more information about the error | ELMo embeddings performs well in many NLP tasks |
| CON | Add noise to the data | Focused on semantics (meaning) rather than syntactic |

**Table 4: Results Comparison**

In this study, we experiment with sequence-to-sequence model by adding error tags and ELMo embeddings. There are advantages and disadvantages of adding error tags and ELMo embeddings, as the chart above summarizes. The motivation of adding error tags in addition to insertion and deletion tags is to provide more information about the error. We hope that by having this information in the neural machine translation, the output will be more accurate. However, the disadvantage of having error tags is that it adds noise to the data, especially when the dataset is small. In this case, the different error tags will get mixed up with the actual grammatically wrong sentence tokens, and the results might not perform as well for small dataset.

ELMo embeddings are experimented because ELMo embeddings performs well in many NLP tasks. However, one possible downside of ELMo embeddings is that it focuses on semantics meaning rather than syntactic. Therefore, while ELMo performs well on many NLP tasks like classification and summarization, it may not perform as ideal with grammar correction.

## 9 CONCLUSIONS

To conclude, in this study we created a sequence-to-sequence neural machine translation with attention for grammar correction. We performed experiments by implementing two possible improvements. One is to add error tags in addition to insertion and deletion tags to the output of the machine translation. The second improvement is to use ELMo embeddings.

Results show that adding error tags helps a little. In addition, ELMo does not perform ideally, and a possible improvement is to have error tags trained on ELMo first before training on the entire sentence.

## REFERENCES


[1] Chenrui Li, Yuanbin Wu, and Man Lan. 2018. Inference on Syntactic and Semantic Structures for Machine Comprehension. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018.* https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16333

[2] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015.* 1412–1421. http://aclweb.org/anthology/D/D15/D15-1166.pdf

[3] Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 Shared Task on Grammatical Error Correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014.* 1–14. http://aclweb.org/anthology/W/W14/W14-1701.pdf

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers).* 2227–2237. https://aclanthology.info/papers/N18-1202/n18-1202

[5] PyTorch. 2017. Translation with a Sequence to Sequence Network and Attention. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

[6] Allen Schmaltz, Yoon Kim, Alexander M. Rush, and Stuart M. Shieber. 2016. Sentence-Level Grammatical Error Identification as Sequence-to-Sequence Correction. In *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications, BEA@NAACL-HLT 2016, June 16, 2016, San Diego, California, USA.* 242–251. http://aclweb.org/anthology/W/W16/W16-0528.pdf

[7] Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel H. Goddard, and Charles A. Sutton. 2018. Sequence-to-Point Learning With Neural Networks for Non-Intrusive Load Monitoring. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018.* https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16623