



AMYA CODERS

107202002

GRAMMAR CORRECTION

Instruction Manual

Author:
Atul Sahay

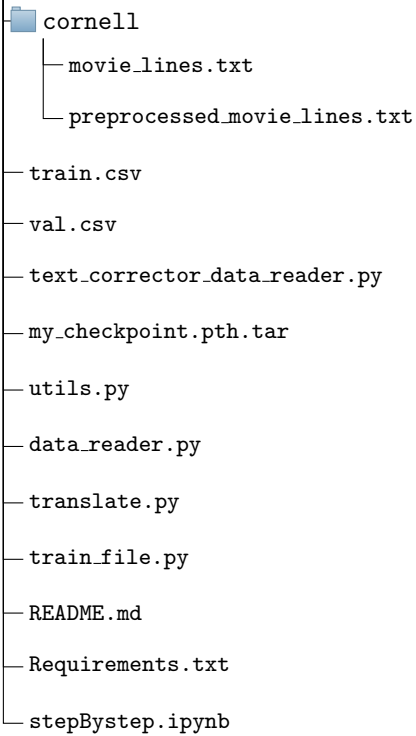
July 31, 2020

Contents

1	Project's Directory Structure	2
2	Requirements	3
2.1	How to install required modules for the project?	3
3	Running the Pretrained Model	4
4	Data Load and Preprocessing	5
4.1	Modified Dataset	5
5	Training The Model	7
5.1	Model Characteristics	7
6	Architectural Details	8
6.1	Introduction	8
6.2	Training	8
6.3	Seq2seq Model	8
6.4	Decoding	9
6.4.1	Biased Decoding	9
7	Experiments and Results	10
7.1	Examples	10

1 Project's Directory Structure

Grammar Correction



```
graph LR; cornell[cornell] --> movie_lines[movie_lines.txt]; cornell --> preprocessed_movie_lines[preprocessed_movie_lines.txt]; train_csv[train.csv]; val_csv[val.csv]; text_corrector_data_reader[text_corrector_data_reader.py]; my_checkpoint_pth_tar[my_checkpoint.pth.tar]; utils_py[utils.py]; data_reader_py[data_reader.py]; translate_py[translate.py]; train_file_py[train_file.py]; README_md[README.md]; Requirements_txt[Requirements.txt]; stepBystep_ipynb[stepBystep.ipynb];
```

The diagram shows a directory structure for a project named 'Grammar Correction'. At the top level, there is a folder named 'cornell' (indicated by a blue folder icon) which contains two files: 'movie_lines.txt' and 'preprocessed_movie_lines.txt'. Below the 'cornell' folder, there is a list of files: 'train.csv', 'val.csv', 'text_corrector_data_reader.py', 'my_checkpoint.pth.tar', 'utils.py', 'data_reader.py', 'translate.py', 'train_file.py', 'README.md', 'Requirements.txt', and 'stepBystep.ipynb'.

2 Requirements

2.1 How to install required modules for the project?

- Search for the **Requirements.txt** file in the project repository.
- Open the terminal and write (be sure you are in the project directory): See Fig: 1

user: Explainable-latent-structures\$: **pip3 install -r Requirements.txt**

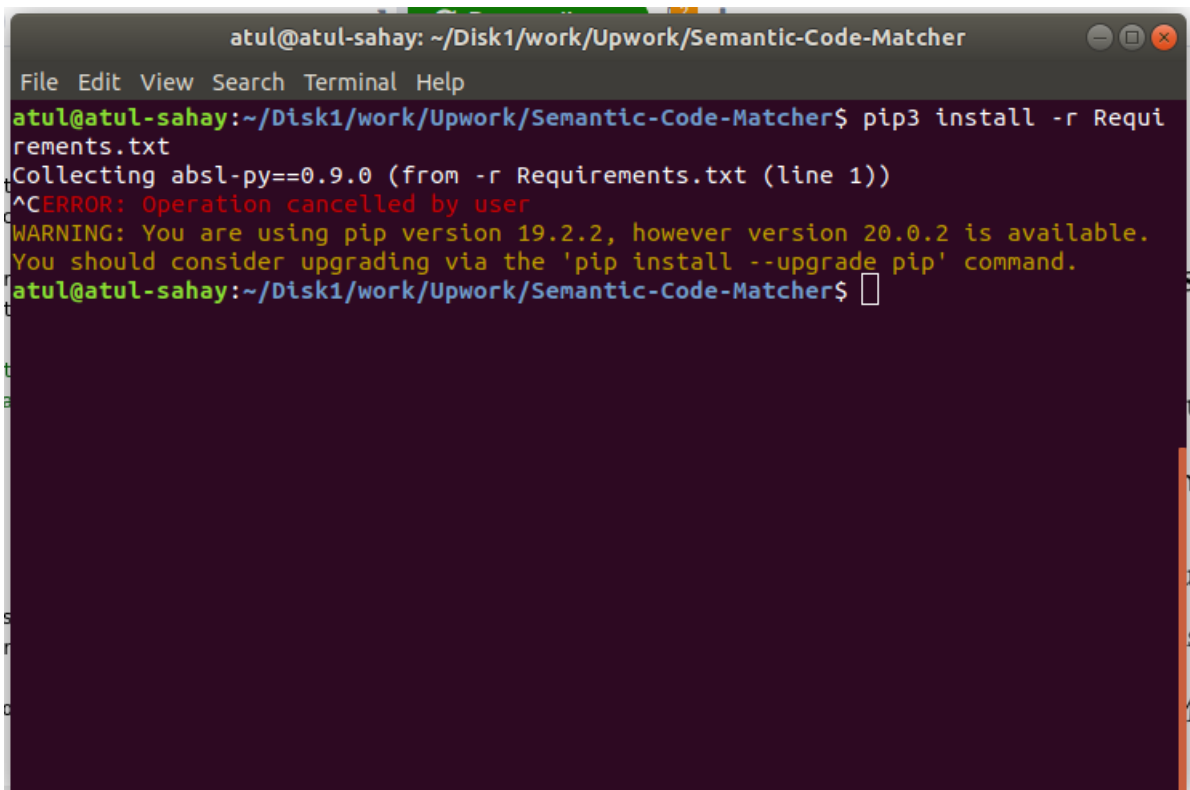
A screenshot of a terminal window titled 'atul@atul-sahay: ~/Disk1/work/Upwork/Semantic-Code-Matcher'. The terminal shows the command 'pip3 install -r Requirements.txt' being executed. The output indicates that 'absl-py==0.9.0' is being collected from the requirements file. However, the installation is cancelled by the user, indicated by a red '^C' and the message 'ERROR: Operation cancelled by user'. A warning message follows, stating that the user is using pip version 19.2.2, while version 20.0.2 is available, and suggesting an upgrade via 'pip install --upgrade pip'. The terminal ends with the prompt 'atul@atul-sahay:~/Disk1/work/Upwork/Semantic-Code-Matcher\$' and a cursor.

Figure 1: Installation of the required modules

3 Running the Pretrained Model

For the sake of running the pretrained model, all you need to is to open the file `translate.py` See Fig : 3. Please make sure you have `my_checkpoint.pth.tar` ready see the section 1 of the project directory

user: Explainable-latent-structures\$: **python3 translate.p**

```

atul@atul-HP-Pavilion-15-Notebook-PC: ~
atul@atul-HP-Pavilion-15-Notebook-PC: ~ 142x38
config.py my_checkpoint.pth.tar seq2seq_attention.py
general_helper.py predict.py train.py
__init__.py prepare_data.py train_script.sh
(venv) ayush@neeti:~/Music/Grammar-Correction$ gdrive upload my_checkpoint.pth.tar
Uploading my_checkpoint.pth.tar
Uploaded 159thEctDVHIV-aDQcJf0NeJaeJ5nRNS at 2.6 MB/s, total 541.7 MB
(venv) ayush@neeti:~/Music/Grammar-Correction/models$ cd ..
(venv) ayush@neeti:~/Music/Grammar-Correction$ ls
cornell_grammar_correction preprocess_seq2seq_model_with_line_align train_file.py
data.csv preprocess_movie_dialogs.py Transformer.ipynb
data_reader.py pycache translate.py
Hmodel.ipynb README.md translate.py
LICENSE Subject Links Requirements.txt utils.py
MACOSX What is deep - stepvstep.ipynb val.csv
models What is deep - stepvstep.ipynb
my_checkpoint.pth.tar text_corrector_data_readers.py
paths.sh collection/blob/machine-learning/more_advanced/seq2seq_attention.py
(venv) ayush@neeti:~/Music/Grammar-Correction$ python3 translate.py
Start: tokenizing the train and validation set
train: 177854
val: 19763
Done with creation of Dataset object
10004
ID | CPU | MEM |
-----
0 | 5% | 4k |
1 | 0% | 0% |
2 | 0% | 0% |
3 | 0% | 0% |
=> Loading checkpoint
bleu score: 39.0071883374451
Enter Sentence: Alex went to market
Translated example sentence:
['alex', 'went', 'to', 'the', 'market', '<eos>']
Enter Sentence: I be there
Translated example sentence:
['I', 'be', 'there', '<eos>']
Enter Sentence: I be there
Translated example sentence:
['I', 'be', 'there', '<eos>']

```

Figure 2: Command line Grammar Correction results

4 Data Load and Preprocessing

For the ease of the use the preprocessed training and validation file is included in the project directory.

Please ensure that you got the train.csv and val.csv file ready before training and evaluating.

I have used a cornell movie dialogs and modified it to use. This corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts: **Description**

- 220,579 conversational exchanges between 10,292 pairs of movie characters
- Involves 9,035 characters from 617 movies
- In total 304,713 utterances
- movie metadata included:
 - genres
 - release year
 - IMDB rating
 - number of IMDB votes
 - IMDB rating
- character metadata included:
 - gender (for 3,774 characters)
 - position on movie credits (3,321 characters)

4.1 Modified Dataset

Given a sample of text like this, the next step is to generate input-output pairs to be used during training. This is done by:

1. Drawing a sample sentence from the dataset.
2. Setting the input sequence to this sentence after randomly applying certain perturbations.
3. Setting the output sequence to the unperturbed sentence.

where the perturbations applied in step (2) are intended to introduce small grammatical errors which we would like the model to learn to correct. Thus far, these perturbations have been limited to:

- the subtraction of articles (a, an, the)
- the subtraction of the second part of a verb contraction (e.g. “‘ve”, “‘ll”, “‘s”, “‘m”)
- the subtraction of the preposition (‘to’)
- the replacement of a few common homophones with one of their counterparts (e.g. replacing “their” with “there”, “then” with “than”)

For example, given the sample sentence:

And who was the enemy?

the following input-output pair could be generated:

("And who was enemy ?", "And who was the enemy ?")

The rates with which these perturbations are introduced are loosely based on figures taken from the CoNLL 2014 Shared Task on Grammatical Error Correction. In this project, each perturbation is randomly applied in 25% of cases where it could potentially be applied.

5 Training The Model

Change the directory to project parent Directory as **Grammar-Correction**. You are going to use the `train_file.py` for this operation.

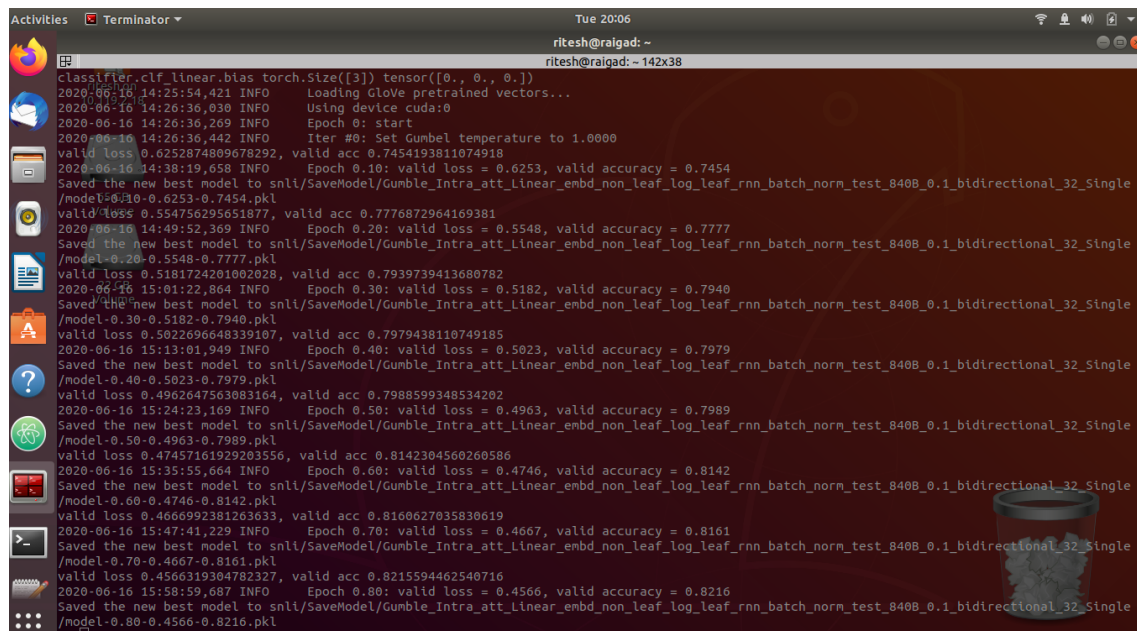
5.1 Model Characteristics

- The whole model is trained on the GPU specification:
 - Name: Geforce RTX 2080 ti
 - Memory: 11 GB
 - Cuda version 10.1
- The whole process takes around 2-3 days for completion

Run the below command for training the model.

```
-- > $ CUDA_LAUNCH_BLOCKING=1 python3 train_file.py
```

Once the training is completed you will have a number of model file "my_checkpoint.pth.tar" which you can use to evaluate.



```

classfier.clf.linear.bias torch.Size([3]) tensor([0., 0., 0.])
2020-06-16 14:25:54,421 INFO Loading GloVe pretrained vectors...
2020-06-16 14:26:36,030 INFO Using device cuda:0
2020-06-16 14:26:36,269 INFO Epoch 0: start
2020-06-16 14:26:36,442 INFO Iter #0: Set Gumbel temperature to 1.0000
valid loss 0.6252874809678292, valid acc 0.7454193811074918
Epoch 0.10: valid loss = 0.6253, valid accuracy = 0.7454
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.10-0.6253-0.7454.pkl
valid loss 0.554756295651877, valid acc 0.7776872964169381
Epoch 0.20: valid loss = 0.5548, valid accuracy = 0.7777
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.20-0.5548-0.7777.pkl
valid loss 0.5181724201002028, valid acc 0.7939739413680782
Epoch 0.30: valid loss = 0.5182, valid accuracy = 0.7940
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.30-0.5182-0.7940.pkl
valid loss 0.5022696648339107, valid acc 0.7979438110749185
Epoch 0.40: valid loss = 0.5023, valid accuracy = 0.7979
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.40-0.5023-0.7979.pkl
valid loss 0.4962647563083164, valid acc 0.7988599348534202
Epoch 0.50: valid loss = 0.4963, valid accuracy = 0.7989
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.50-0.4963-0.7989.pkl
valid loss 0.47457161929203556, valid acc 0.8142304560260586
Epoch 0.60: valid loss = 0.4746, valid accuracy = 0.8142
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.60-0.4746-0.8142.pkl
valid loss 0.4666992381263633, valid acc 0.8106027035830619
Epoch 0.70: valid loss = 0.4667, valid accuracy = 0.8161
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.70-0.4667-0.8161.pkl
valid loss 0.4566319304782327, valid acc 0.8215594462540716
Epoch 0.80: valid loss = 0.4566, valid accuracy = 0.8216
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.80-0.4566-0.8216.pkl
valid loss 0.433951473344444, valid acc 0.833951473344444
Epoch 0.90: valid loss = 0.4339, valid accuracy = 0.8339

```

Figure 3: Training Operation taking place

6 Architectural Details

6.1 Introduction

While context-sensitive spell-check systems (such as AutoCorrect) are able to automatically correct a large number of input errors in instant messaging, email, and SMS messages, they are unable to correct even simple grammatical errors. For example, the message “I’m going to store” would be unaffected by typical autocorrection systems, when the user most likely intendend to communicate “I’m going to the store”.

Inspired by recent advancements in NLP driven by deep learning (such as those in Neural Machine Translation by Ref (1), I decided to try training a neural network to solve this problem. Specifically, I set out to construct sequence-to-sequence models capable of processing a sample of conversational written English and generating a corrected version of that sample

6.2 Training

In order to artificially increase the dataset when training a sequence-to-sequence model, I performed the sampling strategy described above multiple times over the Movie-Dialogs Corpus to arrive at a dataset 2-3x the size of the original corups. Given this augmented dataset, training proceeded in a very similar manner to the seq2seq model. That is, I trained a sequence-to-sequence model consisting of LSTM encoders and decoders bridged via an attention mechanism, as described in Bahdanau et al., 2014 (see Ref (1)).

6.3 Seq2seq Model

Encoder is input with the corrupted sentences and decoder is used to correct those sentences.

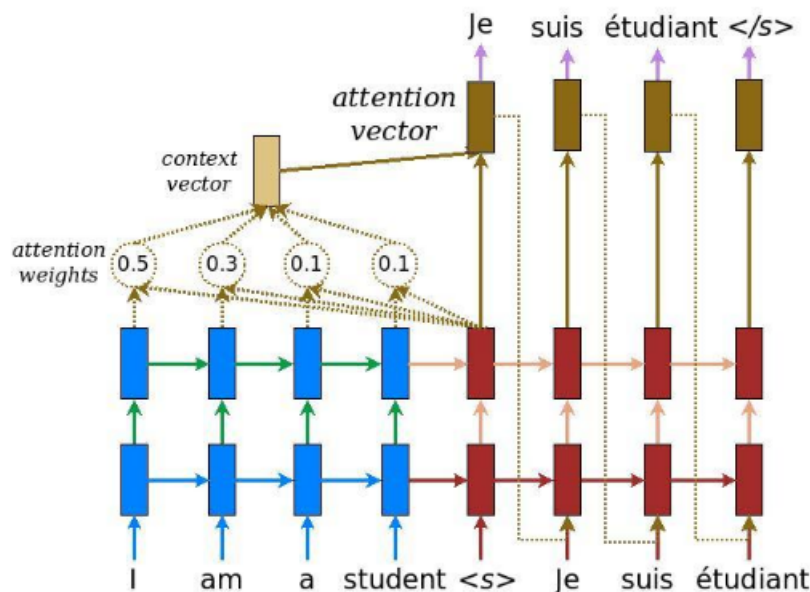


Figure 4: Seq2seqModel

6.4 Decoding

Instead of using the most probable decoding according to the sequence-to-sequence model, this project takes advantage of the unique structure of the problem to impose the prior that all tokens in a decoded sequence should either exist in the input sequence or belong to a set of “corrective” tokens. The “corrective” token set is constructed during training and contains all tokens seen in the target, but not the source, for at least one sample in the training set. The intuition here is that the errors seen during training involve the misuse of a relatively small vocabulary of common words (e.g. “the”, “an”, “their”) and that the model should only be allowed to perform corrections in this domain.

This prior is carried out through a modification to the decoding loop in Pytorch’s seq2seq model in addition to a post-processing step that resolves out-of-vocabulary (OOV) tokens:

6.4.1 Biased Decoding

To restrict the decoding such that it only ever chooses tokens from the input sequence or corrective token set, this project applies a binary mask to the model’s logits prior to extracting the prediction to be fed into the next time step.

This is done by constructing a mask:

```
mask[i] == 1.0 if i in (input or corrective_tokens) else 0.0
```

and then using it during decoding in the following manner:

```
token_probs = tf.softmax(logits)
biased_token_probs = tf.mul(token_probs, mask)
decoded_token = math_ops.argmax(biased_token_probs, 1)
```

Since this mask is applied to the result of a softmax transformation (which guarantees all outputs are positive), we can be sure that only input or corrective tokens are ever selected.

Note that this logic is not used during training, as this would only serve to hide potentially useful signal from the model.

7 Experiments and Results

Below are some anecdotal and aggregate results from experiments using the Deep Text Corrector model with the Cornell Movie-Dialogs Corpus. The dataset consists of 304,713 lines from movie scripts, of which 243,768 lines were used to train the model and 30,474 lines each were used for the validation and testing sets. For the training set, 2 samples were drawn per line in the corpus, as described above. The sets were selected such that no lines from the same movie were present in both the training and testing sets.

The model being evaluated below is a sequence-to-sequence model, with attention, where the encoder and decoder were both 2-layer, 512 hidden unit LSTMs. The model was trained with a vocabulary consisting of the 10,000 most common words seen in the training set.

Model achieved a BLEU score of 0.59878

7.1 Examples

In addition to the encouraging aggregate performance of this model, we can see that its is capable of generalizing beyond the specific language styles present in the Movie-Dialogs corpus by testing it on a few fabricated, grammatically incorrect sentences. Below are a few examples,

```
In [31]: decode("Alex went to market")
Out[31]: 'Alex went to the market'
```

```
In [32]: decode("I be there")
Out[32]: "I 'll be there"
```

```
In [33]: decode("you heard enough . decide . are you going to release me ?")
Out[32]: "you 've heard enough . decide . are you going to release me ?"
```

References

[1] <https://arxiv.org/pdf/1409.0473.pdf>