Deep Grammar Error Corrector

Zixi Liu
zixiliu@usc.edu

Abstract

Recently, the development of attention mechanism and transformer gives rise to better modeling long term dependency and automatic alignment in human language. Since grammar error correction highly relies on these properties, we apply related techniques to GEC task and compare their performance in different hyperparameter settings. We further explore BPE encoding and embedding sharing to reduce model complexity. Our final model achieve 69.0 GLEU score, surpassing both our baseline model and a statistic machine learning model.

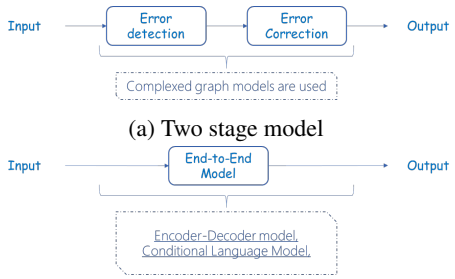
1 Introduction

Correcting erroneous sentences as human beings is one of the key ways leading to artificial intelligence. As a matter of fact, grammar error correction is not only a practical technique but also a generic task for evaluating language models and artificial intelligent algorithms. Specifically, grammar error correction (GEC) aims at correcting various kinds of grammar error corresponding with spelling, punctuation, grammar and word choices.

Traditional machine learning methods has shown good ability in correcting errors strongly related to grammatical errors (Yuan and Felice, 2013), but these models show a poor performance in helping users with better word choices and dealing with ambiguity of human language (Bano, 2015). Since 2014, neural network methods provide great improvement for plenty of NLP tasks, mainly because neural network based word-embedding training and powerful sequence modeling techniques, for the first time in history, provide a compressed representation and long dependency modeling for human language. Neural machine models in machine translation, name entity recognition, summary, natural language inference etc., surpass the performance of traditional methods and some of them even surpass human beings in the evaluation (Ge et al., 2018). With attention mechanism, many works in GEC task have utilized word-by-word attention with various kinds of recurrent neural networks. (Wang et al., 2017) However, the performance of these methods tend to reach ceiling due to low efficient word embedding modeling and memory loss in classical recurrent neural network, i.e. RNN, LSTM, GRU etc. Fortunately, the advent of feed-forward attention-based architectures, such as transformer and convectional Sequence to Sequence (Seq2Seq) model, make breakthroughs for efficient contexts modeling. These methods get rid of recursive nature in RNN like architecture and highly rely on attention mechanism which lead to a tremendous improvement in training speed and substantial reduction in parameter numbers.

In this project, we focus on explore the mechanism and implementation of three popular neural machine techniques (LSTM Seq2Seq(Li et al., 2018; Hochreiter and Schmidhuber, 1997), transformer(Vaswani et al., 2017), GPT-2 model(Radford et al., 2019) fine-tuning) and using these methods for GEC. To enhance our system performance, we further explore sub-word tokenization, embedding sharing methods and pretrained embedding initialization. Through our experiment, we demonstrate that

- 1). deep learning methods can outperform traditional methods under GLEU(Napoles et al., 2015) evaluation,
- 2). attention mechanism can well improve model performance with small increase in parameter numbers,
- 3). under similar parameter settings transformer can outperform LSTM based methods in both training speed and training accuracy,



(a) Two stage model

(b) End-to-end model

Figure 1: Comparison of traditional machine learning formulation and deep learning formulation

- 4). fine-tuned GPT-2 model under conditional language model setting performs better in correcting errors relating to word choice.

In the following sections, we firstly introduce how to formulate GEC task under neural machine based methods. We secondly provide model analysis for the models in this project and thirdly introduce enhancement methods. We finally give our experiment results and conclusion in the last two sections.

2 Problem formulation

In traditional machine learning, GEC task is built as a two-stage model where a potential erroneous sentence is put into an error detection model detecting the possible errors and an error correction model is used to generate possible corrections for each errors. (Figure 1a)Usually in order to obtain a whole sentence context based optimal correction, we need to build a complex graph models to propagate information and search for global optimal result which is very time consuming and need tons of hand-craft architecture design. (Kumar, 2012) In contrast to traditional machine learning methods, deep-learning based GEC system is formulated as an end-to-end system. (Figure 1b) Based on research so far, we conclude following two formulation for GEC task in deep learning context.

2.1 formulate as an Encoder-Decoder Model

Encoder-decoder formulation is the closest formulation to human beings' thinking mode. The possible erroneous sentence is encoded into a single representation or stack of representations corresponding to words or word-pieces in original sentence. Guided by representations from encoder, the decoder generate corrected sentence word by word recursively.

$$\boldsymbol{S} = encoder(S) \tag{1}$$

$$\hat{C}_{opt} = \underset{k=0}{\overset{n}{argmax}} \prod_{k=0}^{n} P(C_k|\boldsymbol{C}_{0\sim(k-1)}, \boldsymbol{S}; \boldsymbol{\theta}) \tag{2}$$

where $\boldsymbol{S}$ is the representations of input potential erroneous sentence S, $\boldsymbol{C}_{0\sim(k-1)}$ are first k words in output sentence C_k is the $k+1^{th}$ word in the output sentence, $\boldsymbol{\theta}$ are model parameters and $\hat{C}_{opt}$ is the optimal corrected sentence.

2.2 formulate as a Conditional Language Model

Conditional language model is a model that predicts the next word recursively based on previous inputs. Since there is no explicit source and target in such model settings, we need to reformat our data and use triggers tokens to inform the model of doing grammar error correction. Here we format our data as below where $======$ is the trigger and $< endoftext >$ is the ending symbol.

$$\text{source sentence} ====== \text{target sentence} < endoftext > \tag{3}$$

$$\hat{T}_{opt} = \underset{k=0}{\overset{n}{argmax}} \prod_{k=0}^{n} P(T_k|\boldsymbol{T}_{0\sim(k-1)}; \boldsymbol{\theta}) \tag{4}$$

where $\boldsymbol{\theta}$ are model parameters and $\hat{T}_{opt}$ is the optimal corrected sentence concatenated by the source sentence with trigger token. Unlike a typical conditional language model, in GEC task, the model only use outputs after trigger token for recursive decoding so as to input all source information into the model.

3 Models

In this section, we introduce three models on GEC task. In section 3.1, we analyze the model architecture of LSTM based encoder-decoder model with and without using attention mechanism. We then provide the transformer architecture in 3.3 and analyze the implementation of transformer in FairSeq(Ott et al., 2019). We finally provide the architecture of GPT-2 small model.

3.1 LSTM based sequence to sequence model

The Seq2Seq model (Sutskever et al., 2014) is a simple and straightforward application of Long Short-Term Memory (LSTM) architecture to solve Seq2Seq language problems. The recurrent neural

network (RNN) is able to process inputs of any length and study the underlined relationship between current state and previous states, while it is not clear how to apply RNN to problems that input and output sequences are in different length. The Seq2Seq can simply address this problem by embedding the whole input sequence as a single vector representation with fixed length. The LSTM units section for input embedding is usually referred as "encoder". One another RNN section, referred as "decoder", is used to feed the input embedding from "encoder" and output the target sequence.

Given the input sentence $\mathbf{x}$ and output sentence $\mathbf{y}$, the model can be formulated as:

$$\begin{aligned} &\mathbf{P}(\mathbf{y}_0, \ldots, \mathbf{y}_{T'} | \mathbf{x}_0, \ldots, \mathbf{x}_T) \\ &= \prod_{t=0}^{T'} \mathbf{P}(\mathbf{y}_t | \mathbf{V}, \mathbf{y}_0, \ldots, \mathbf{y}_{t-1}) \end{aligned} \tag{5}$$

where $\mathbf{x}_t$ and $\mathbf{y}_{t'}$ denote the encoder hidden state at timestep t and decoder hidden state at timestep t'. $\mathbf{V}$ denotes the sentence embedding representation at the final encoder state. The length of input sequence T may be different from the one of output sequence T'. The equation holds in the Seq2Seq model for the reason that each output state $\mathbf{y}_t$ is dependent on the final sentence embedding and previous states of RNN.

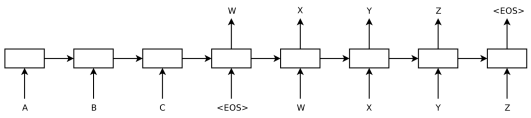


Figure 2: A sample Seq2Seq model with input sentence "A B C" and expected output "W X Y Z". Each sentence ends with a End-Of-Sequence symbol "⟨EOS⟩", which allows to define the model formulation with any input and output length. The encoder embeds the whole input sentence ("A B C ⟨EOS⟩") as a single fixed-length vector representation $\mathbf{V}$. The decoder makes predictions until outputting "⟨EOS⟩".

3.2 Sequence to sequence with attention

Even the Seq2Seq model is easy to interpret and implement, it comes with limitations. The trouble of it is that the decoder only take the information of the last encoder hidden state, which is the single input sequence embedding. In the case of long sentences input, it is very likely for the encoder to lose more information using fixed length embedding representations, which lead to poor performance of the whole system.

Instead of only taking one single embedding from the last state for decoding outputs, the Seq2Seq with attention model (Bahdanau et al., 2014) provides the decoder with information from every encoder hidden state. Figure 3 shows the mechanism of attention. For a certain decoder hidden state $\mathbf{y}_k$, a group of attention scores $\mathbf{e}_{k,0\sim T}$ for each encoder hidden states are calculated as the inner product of them, i.e.,

$$\mathbf{e}_{k,t} = \langle \mathbf{x}_t, \mathbf{y}_k \rangle \tag{6}$$

$$\mathbf{e}_{k,\cdot} = [\langle \mathbf{x}_0, \mathbf{y}_k \rangle, \ldots, \langle \mathbf{x}_T, \mathbf{y}_k \rangle]^T \tag{7}$$

The attention scores are post-processed by softmax activation to yield proper attention (probability) distribution $\alpha_{t,\cdot}$:

$$\alpha_{k,\cdot} = \text{softmax}(\mathbf{e}_{k,\cdot}) \tag{8}$$

Attention outputs are obtained by the weighted sum of encoder hidden states:

$$\mathbf{a}_k = \sum_{i=0}^{T} \alpha_{k,i} \mathbf{x}_i \tag{9}$$

Finally, the current state output of decoder is the concatenation of current state and the attention output:

$$\hat{\mathbf{y}} = [\mathbf{y}_k; \mathbf{a}_k] \tag{10}$$

The attention mechanism allows the decoder searches through the whole sentence for useful information. The attention scores, the inner product of decoder hidden state and all encoder hidden states, measures the correlation between current prediction and each of the words in the input sentence. With higher correlation, the encoder state has higher weights for the attention output. This emulates searching through the input sentence and pays more attention on the highly correlated words. The concatenation of current decoder state and the attention output provides extra information for final prediction.

3.3 Transformer

architecture Transformer is first introduced by Vaswani et al (Vaswani et al., 2017) in 2017 which is feed-forward attention based encoder-decoder model. The model architecture is show in fig. 4. Each layer in encoder contains two sub-layers. The first sub-layer is a multi-head self-attention layer

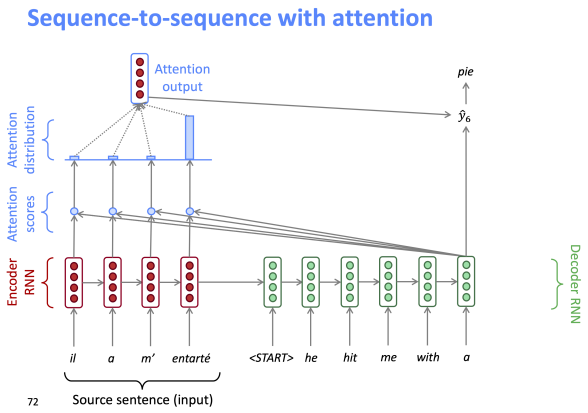


Figure 3: A sample Seq2Seq with attention model decoding the word "pie". With attention mechanism, the prediction pays more attention on the input word "entarte". The output of decoder $\hat{\mathbf{y}}$ is the concatenation of current state $\mathbf{y}_k$ and the attention output $\mathbf{a}_k$.

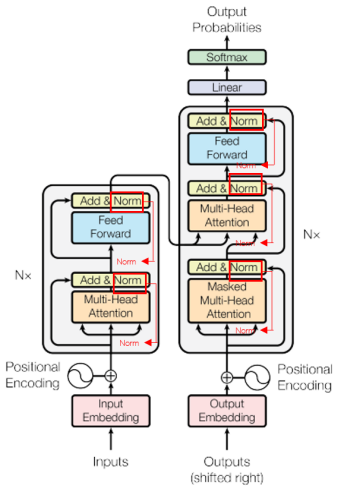


Figure 4: improved transfomer model, move layer normalization in the front of each sub-layer(Vaswani et al., 2017)

where the keys, queries and values are all the same representation matrix coming from last encoder layer or embedding layer. In each head, the attention score and weighted sum are calculated using dot product after projecting key matrix, query matrix and value matrix into a subspace by three different trainable linear projections. In Vaswani et al's implementation, the dimension of these subspace satisfies $Dim_{att} = \frac{1}{h} Dim_{hidden}$, where Dim_{att} is the dimension of subspace and Dim_{hidden} is the dimension of output in each layer of encoder. The representation will then be concatenated and projected again using 2-layer MLP whose the input dimension and output dimension are the same. Each sub-layer also has a residual addition between input and output and layer normalization. The output of encoder is a contextualized representation matrix with size $(\text{sequence length}) \times Dim_{hidden}$ and each row of the matrix is a representation for the corresponding word in the sentence. The representation from last layer in encoder will be put into the second second sub-layer of each layer of decoder, naming cross attention layer. The other part of the decoder is the same with encoder except that a masking is implemented in the first sub-layer to keep recursive property for decoding.

implementation In the FairSeq implementation, there are two improvement of original transformer model. One is to move layer normalization to the front of each sub-layer (show in Figure4). In our experiment, we demonstrate that this implementation improve training speed and robustness (table 1). The other one is using cross attention both in the first and the second sub-layer in the decoder layer. In our experiment, we disable this architecture since it would lead to a small performance degrade.

Table 1: training robustness and layer normalization

Architecture	Val Loss
layer normalization before sub-layer	epoch 1: 6.743 epoch 2: 1.986 epoch 3: 1.224 ...
layer normalization after sub-layer	epoch 1: 9.883 epoch 2: 11.745 epoch 3: 11.445 ...

3.4 GPT-2 model

GPT-2 (Radford et al., 2019) is a large transformer-based language model with 1.5 billion parameters, training on a dataset of 8 million webpage. It shows the capability of generating conditional synthetic text, handling task like question answering, text summery, story generation etc. The architecture of GPT-2 is a stack of decoder layers in transformer (show in Figure 5) and can be view as an auto-regressive decoder in high level.

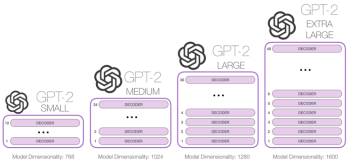


Figure 5: GPT-2 model architecture(Alammar)

4 Parameter Reduction

In order to reduce parameter numbers, we consider to reduce size of embedding layers. Due to large size of vocabulary in natural language, embedding layers usually cover up 40% to 70% of parameters in neural network, but a large size of embedding layer doesn't always provide enhancement in the performance, because a substantial amount of words in human language are rarely used and their embeddings may hardly get effective updates during the training. Fortunately, byte pair encoding (Sennrich et al., 2015) and embedding layer share schema (Press and Wolf, 2016) make it possible to well reduce the vocabulary size.
Byte pair encoding is originally a compression algorithm, in which the most frequent byte pair is mapping to a new byte. Applying this technique to neural machine translation helps well reduce the size of dictionary and in return, reduce the harm of Curse of Dimensionality. Typically speaking, BPE can improve NLP task for 5% on their performance. Embedding sharing is motivated by the fact that GEC task's input and output share almost the same word piece dictionary and language characteristic. Through our experiment, we show that model parameter number can be reduced by20% to 25% with at worse 1.3% degrade in LSTM model and 2% degrade in transformer model. In table 2, we show the vocabulary using different settings and formula 11 shows the relation of embedding layer size and vocabulary size.

$$\text{Embedding layer param} = size(Vocab) \times Dim_{emb} \quad (11)$$

5 Experiment Result

5.1 Implementation Details

Data Set In this project, we training our data using Lang-8 grammar error correction data set (Tajiri et al., 2012) and Conll 2014 shared task dataset (Ng et al., 2014). Lang-8 dataset contains 1136634 grammar error correction sentence pairs and Conll 2014 dataset contains 21835 grammar error correction sentence pairs. Since Lang-8 dataset is collected by online English learners but Conll 2014 dataset is collected by professional correctors in students' paper, Lang-8 dataset is easier than Conll 2014 dataset and contains more noises.

We use these two data set for training LSTM Seq2Seq model and transformer model from scratch. We fine-tune GPT-2 model only on Conll 2014 shared task dataset due to two reasons. First, fine-tuning all two dataset takes 18 hours for one epoch training which is not practical in this course project. Second, we believe GPT-2 model has learning sufficient knowledge of language and it can transfer the knowledge without using too much data.

Software We implement all customized models using FairSeq(Ott et al., 2019) package and fine-tune GPT-2 using Huggingface(Wolf et al., 2019) package. All these two packages are based on PyTorch deep learning framework. The version information is given in table 3. Due to limitation in computation resources, our model is train on three hardware platforms, i.e. Ubuntu 18.04 LTS with RTX2070+RTX2060s, AWS instances with Amazon Linux 2 and P100, Colab with P100. In this three platform we all use python 3.7.5, CUDA 10.1 and keep our software package in same version.

Training hyperparameter setting If not otherwise specified, during training, we use 8000 tokens per batch for training which can avoid truncation when a sentence is too long. We use Adam optimizer with an initial learning rate of 5e-3 and learning rate shrink of 0.5. We use cross-entropy loss function and 0.1 L2-regularization. We train the model for at most 50 epoch and early stop if the loss on validation set doesn't improve after 8 epoch. For LSTM model we use 0.2 dropout rate for all MLP layers and embedding layers. For transformer models, we use 0.1 dropout rate for MLP layers and 0.1 dropout rate for linear layers in attention layers. We fix the layer number to 3 in both encoder and decoder.

5.2 Model Comparison

We keep the same layer number for all customized models and adjust their embedding size and hidden layers size to make different architectures have similar parameter. We build up 9 LSTM models and 4 transformer models to explore how embedding size, hidden size, attention layer and embedding sharing schema influence the model performance and the time for convergence. We list these experiment results and fine-tuning GPT-2 result in table 6.

LSTM model We use 350 embedding size , 350 hidden size and 1 layer for both encoder and decoder LSTM Seq2Seq as our baseline. We then

Table 2: Vocabulary size reduction

BPE encoding	Embedding share	input vocab	output vocab	shared vocab	unknown token percentage
√	-	9944	9592	-	0.00071%
√	√	-	-	15367	0.000739%
-	-	141615	125191	-	0.601%

Software	**Version**
FairSeq	0.9.0
Huggingface	2.8.0
PyTorch	1.4.0

Table 3: Software information

adjust embedding size and hidden size to 273 and use 3-layer for encoder and decoder as our basic architecture. We adjust this basic architecture and build 7 models for comparison. (see in Table 5 row 1-9)

Comparing Model 1 with 2, we can see that our hand-implemented LSTM can achieve similar Val loss as FairSeq package. Also, increasing model layers instead of embedding or hidden size can improve model performance with similar model size. Comparing Model 2 and 4, we can see that embedding sharing can improve performance slightly when not using attention, but from Model 3 and 5, it slightly harm the model performance when using attention. From Model 2-5, we can see embedding sharing reduce 31% of model size. Comparing models using attention and models not using attention, we can see the models using attention doubling the performance. Comparing Model 4 with 6 and 5 with 7, increasing hidden size to 512 brings 28% improvement when models don't use attention and brings 8.4% improvement when models use attention. Comparing Model 8 with 4 and 9 with 5, increasing embedding size to 512 brings 17% improvement when models don't use attention and 1.8% improvement when models use attention.

Transformer We use 256 embebbing size, 256 hidden size and 4 multihead size transformer as our basic architecture. We then adjust basic model to build up 4 transformers for comparison.

Comparing Model 10 with 11, sharing embedding in transformer model can reduce 33% of parameters with 2.1% decreasing in model performance. Comparing Model 10 with 2 and 10 with 3, we can see that transformer model performs twice better than LSTM model without using attention and 12% better than than LSTM model using attention. Comparing Model 12 with 11, we can see that increasing hidden size can bring 3.4% improvement when sharing all embeddings. From Model 12 and 13 we can see that sharing embedding layer only in decoder will lead to performance decreasing.

GPT-2 Fine-tuned GPT-2 model doesn't show good performance compared with customized LSTM Seq2Seq model and transformer model. In our training, we find that GPT-2 model get overfitting after 3 epoch training. Although validation loss of GPT-2 is not competitive, but GPT-2 model does show good performance in using better word choice. See the example in Table 6 row 15.

We finally plot the learning curves of Model 2, 3, 7, 10 and 13 in Figure 6 and Figure 7. The learning curves shows that models using attention mechanism have faster convergence speed that the one without using, i.e. model 2 in the graphs. Transformer models have faster convergence speed than LSTM based models. Further investigation shows that increasing hidden state dimension can help LSTM converge faster, but will slightly decrease convergence speed in transformer model.

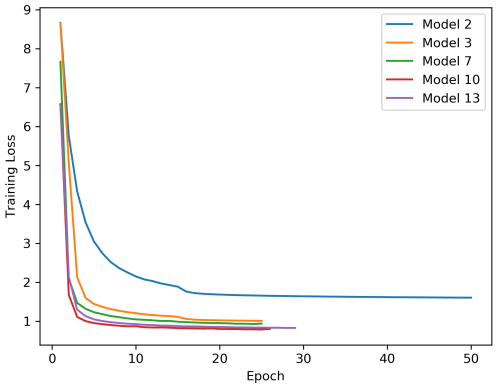


Figure 6: Cross-entropy loss in training set for model 2, 3, 7, 10, 13.

input	If we maximize the effect of Global Warming , all the life form on earth will extinct eventually .
ground truth	If we maximize the effect of Global Warming , all life **forms** on earth will extinct eventually .
GPT-2	If we maximize the effect of Global warming, all life **forms** on earth will eventually **disappear**.

Table 4: GPT-2 model shows better performance in choosing better word choice

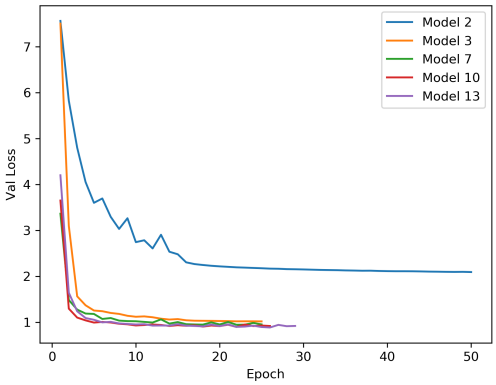


Figure 7: Cross-entropy loss in validation set for model 2, 3, 7, 10, 13.

5.3 Performance Evaluation

We evaluate our models using Conll 2014 shared task test set. GEC task has many evaluation metrics, like M^2 score, BLEU etc. However, these measurements have some shortage. For M^2 score, the score would heavily suffer from reference number and diversity. For BLEU score, the metric may not be able to recognize a grammar error since it search for similar n-gram in the whole reference file. Consider all of these, we decide to use GLEU metric propose by (Press and Wolf, 2016). We compare our 2 of our best model (see 11 in table 7) with our baseline and a model in (Press and Wolf, 2016) which use traditional machine learning method.

Model	**GLEU** on Conll 2014 test set	**GLEU** lang-8 + Conll2014 test set
model 1	0.427082	0.523660
model 13	0.682726	0.690695
NTHU*	0.60	-

Table 7: Evaluation, NTHU(Napoles et al., 2015)

6 Conclusion& Future Work

In this paper, we try out three different architecture and various derivatives of first two networks. We find transformer model shows stronger capability for grammar error correction than LSTM models. Our experiments show that attention mechanism is the key for faster training and better resulting models. We also show that using GPT-2 to solve GEC task as conditional language modeling is not an effective method. Furthermore, using BPE coding and embedding sharing can help LSTM and transformer to well reduce model complexity.

In our future study, we plan to add an encoder module for GPT-2 model or build a decoder module for encoder-like pretrained models, such as BERT(Devlin et al., 2019), RoBERTa(Liu et al., 2019), etc.

References

Jay Alammar. The illustrated gpt-2 (visualizing transformer language models). [EB/OL]. http://jalammar.github.io/illustrated-gpt2/ Accessed August 12, 2019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

Muneera Bano. 2015. Addressing the challenges of requirements ambiguity: A review of empirical literature. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 21–24. IEEE.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Tao Ge, Furu Wei, and Ming Zhou. 2018. Reaching human-level performance in automatic grammatical error correction: An empirical study. *arXiv preprint arXiv:1807.01270*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Vineet Kumar. 2012. *Automatic grammar correction: using PCFGs and whole sentence context*. Ph.D. thesis, UC San Diego.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2015. Ground truth for grammatical error correction metrics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 588–593.

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*, pages 1–14.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. 2012. Tense and aspect error correction for ESL learners using global context. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 198–202, Jeju Island, Korea. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Chuan Wang, Ruobing Li, and Hui Lin. 2017. Deep context model for grammatical error correction. In *SLaTE*, pages 167–171.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zheng Yuan and Mariano Felice. 2013. Constrained grammatical error correction using statistical machine translation. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 52–61.

Table 5: Model architectures and performance comparison

No	Model	Architecture	Num of param	Epoch	Val Loss	Min/epoch
1	Hand-implemented LSTM	Encode Dim 350; Decode Dim 350; Hidden Dim 350; Layer 1; Non-shared Embedding; Attention: no	12659992	35	2.201	1:35 (RTX2070)
2	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 273; Layer 3; Non-shared Embedding; Attention: no	11850148	50	2.091	5:48 (P100)
3	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 273; Layer 3; Non-shared Embedding; Attention: Yes	12073735	25	1.016	6:54 (P100)
4	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 273; Layer 3; shared Embedding; Attention: No	8084076	50	2.042	6:00 (P100)
5	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 273; Layer 3; shared Embedding; Attention: Yes	8307663	25	1.029	6:48 (P100)
6	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 512; Layer 3; shared Embedding; Attention: No	17012633	49	1.457	6:01 (P100 colab)
7	LSTM (FairSeq)	Encode Dim 273; Decode Dim 273; Hidden Dim 512; Layer 3; shared Embedding; Attention: Yes	17799065	25	0.943	6:49 (P100 colab)
8	LSTM (FairSeq)	Encode Dim 512; Decode Dim 512; Hidden Dim 273; Layer 3; shared Embedding; Attention: No	12419292	50	1.692	6:54 (P100 colab)
9	LSTM (FairSeq)	Encode Dim 512; Decode Dim 512; Hidden Dim 273; Layer 3; shared Embedding; Attention: Yes	12642879	50	1.010	6:00 (P100 colab)

Table 6: Model architectures and performance comparison (continuous)

No	Model	Architecture	Num of param	Epoch	Val Loss	Min/epoch
10	Transformer (FairSeq)	Encode Dim 256; Decode Dim 256; FFC Dim 256; Layer 3; Non-shared Embedding	10623488	26	0.895	3:21 (RTX 2070)
11	Transformer (FairSeq)	Encode Dim 256; Decode Dim 256; FFC Dim 512; Layer 3; Non-shared Embedding	11411456	22	0.877	3:35 (RTX 2070)
12	Transformer (FairSeq)	Encode Dim 256; Decode Dim 256; FFC Dim 256; Layer 3; shared Embedding	7100928	21	0.914	3:36 (RTX 2070)
13	Transformer (FairSeq)	Encode Dim 256; Decode Dim 256; FFC Dim 512; Layer 3; shared Embedding	7888896	30	0.883	2:41 (RTX 2070 + RTX 2060S)
14	Transformer (FairSeq)	Encode Dim 256; Decode Dim 256; FFC Dim 512; Layer 3; shared Decoder input-output Embedding	8955904	13	0.93	2:26 (RTX 2070 + RTX 2060S)
15	GPT-2 (FairSeq)	12 Layers Decoder Stack, Embedding Size 768 (Only Trained on CoNLL 2014)	1.5 billion	10	1.11	3:301 (P100)