Grammar Error Correction

Dawid Abucewicz
dawab699@student.liu.se

Stefan Brynielsson
stebr364@student.liu.se

Hugo Cedervall
hugce564@student.liu.se

Viktor Hellqvist
vikhe931@student.liu.se

Hannes Jämtner
hanja189@student.liu.se

Qiming Tang
qimta031@student.liu.se

Mårten Walter
marwa361@student.liu.se

***Abstract*—Grammar error correction is the process of correcting grammatical errors in natural language. Much research exists within the area but most often focuses on English, as there is a lot of data available in a desirable format. This research focuses on how well-known techniques can be applied to a Swedish dataset artificially generated for this research. The Swedish Culturomics Gigaword Corpus was used as a source, and then multiple types of artificial errors were introduced to produce the training examples. Two machine translation methods were evaluated; the pre-trained Swedish BERT model and an LSTM model with pre-trained Swedish embeddings. Both models were trained on grammatically incorrect sentences, and the goal was to transform those into their correct representations. As the LSTM model had converged, adversarial training was also applied to increase the performance even further. There were multiple evaluation metrics, but the BERT model outperformed the LSTM model in all aspects. The adversarial training increased the performance of the LSTM model marginally, probably due to the already poor performance.**

I. Introduction

Natural language processing (NLP) is an emerging area that refers to human language processing with computer algorithms. Grammar Error Correction (GEC) is a field within NLP that concerns detecting and correcting grammatical errors in human language. Much research and many solutions exist for GEC, but often on English datasets with English GEC models. In this project, we aim to test the performance of different promising GEC solutions originally developed for English on Swedish data.

Naturally, the amount of Swedish training data available will be less than what can be found for English. This project will investigate how much Swedish training data is available and how much is required. Also, the format requirements of the training data will be investigated. Some of the research done in English can utilize the available datasets containing sentences with actual grammar errors by humans and the correct counterpart. There are no available datasets containing sentences with incorrect to correct grammar in Swedish. This is the type of dataset required to train any GEC model, so we had to generate this dataset. There are some larger datasets with Swedish texts where the grammar could be considered reasonable and correct. These datasets are used and transformed to the desired dataset format by inserting artificial errors into the sentences. This is challenging as it is hard to reproduce human-like errors artificially. The model results will be highly dependent on the quality of this transformation process, so it must be done thoughtfully.

There are mainly three different methods that have been used to perform GEC successfully; rule-based, classification-based, and machine translation [1]. The Rule-based approach is based on a predefined number of grammatical rules that sentences must follow. This approach is relatively simple to implement, and the performance is rather good for easier grammatical corrections. For more complex cases, it performs worse, as it is not reasonable to generalize the solution to all possible situations. The classification-based approach is a machine learning method that is driven by data to make its corrections compared to the Rule-based method. In this method, all words/replacements are treated as classes, and the goal of the algorithm is to learn which replacement is the best fitting for each word. The last and most recent approach is machine translation, which is built on an encoder-decoder mechanism. The encoder will encode the sentence into a vector, and then the goal of the decoder is to decode this vector to the desired output, which in this case should be the grammatically correct sentence. Most state-of-the-art models within the area are based on this approach, and hence it will also be used in this project.

Within machine translation, there are multiple different techniques one could use. Some of the recent advancements within the NLP area are by the BERT model, which is pre-trained on huge amounts of data. A Swedish BERT version recently was released, which we believe could potentially perform great at the task at hand. Another technique that is often used for various NLP tasks is based on LSTM networks. This is a less complex model, and hopefully, it can achieve pretty good results. By combining LSTM networks with pre-trained Swedish embeddings, the training times might also be reduced. We believe that using adversarial training on the LSTM model could improve the performance even further, as research has shown on English datasets, and as the data in Swedish is somewhat limited. Overall the BERT model

probably has the most potential, but training times might be very long.

II. Method

A. Dataset Generation

There is no dataset containing Swedish grammatical errors. Thus, it had to be created for evaluating the different grammar error correction methods. The dataset called the Swedish Culturomics Gigaword Corpus was used from Språkbanken which was provided by the department of Swedish at university of Gothenburg [2]. This dataset contains one billion Swedish word split into 60 million sentences from 1950 and onward. The dataset contains sentences tagged with their source: news, government, socialmedia, fiction, or science. For the purpose of the project the dataset was preprocessed to only use sentences between the years 1990-2015 and sentences from social media were removed. Additionally sentences with any characters except: {A-Ö,.?} were removed and only sentence with 3-50 words were kept. For the task of grammar error correction a sentence pair of incorrect-, correct-sentences is required, the obtained sentences were used as the correct ones, the incorrect sentences had to be created based on the correct ones in a rule based manner.

1) Grammar Errors: There is no golden rule of how to create a Swedish grammatical error and therefor the group had to be creative when creating different kind of spelling errors. This included swapping two neighbouring words, replacing commonly confused words (e.g. ”en” with ”ett” and ”de” with ”dom” and the opposite). By using Swedish verbs from wikipedia [1], it was possible to swap verb tense and creating an error that occurs when using the wrong verb tense. Furthermore, a vocabulary of 30000 most common words was extracted from the previously downloaded dataset and, so called, *write apart* and *write together* errors were introduced. *Write apart* error implementation, while iterating through the sentence, checked if a word is in the extracted vocabulary and with the probability $p = 0.5$ the word was split into two words if they also were valid words included in the vocabulary. *Write together* implementation checked two consecutive words; if their merged version was included in the vocabulary, the error was injected with the probability $p = 0.5$.

The final dataset consisted of approximately 24 million sentences, out of which 72% contain an error, that is the incorrect and correct sentences differ.

B. BERT

BERT stands for Bidirectional Encoder Representations from Transformers and was introduced by the researchers at Google AI Language [3]. The transformers in BERT includes an encoder and an attention mechanism. The encoder reads the entire input sentence of words at once and the encoder creates predictions on a specific task. The attention mechanism makes it possible for the model to learn dependent relations between words. Traditionally directional models usually reads an input from left to right or right to left, as BERT reads the entire sentence at once, it makes it bidirectional. Instead of knowing the context of a word from the left side or the right side as in a directional model, BERT have the context of word from both sides.

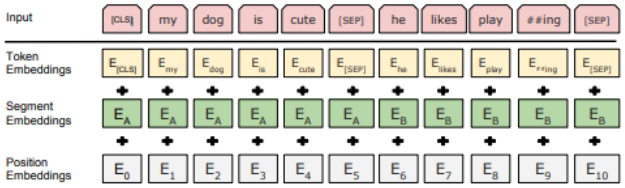

Fig. 1. Description on how the input is processed in BERT [3].

1) Input Process: As the model takes the entire sentence of word as an input, it needs help to determine when a sentence starts and ends. To indicate the start, we use the CLS token which stands for classification and at the end of the sentence there is an SEP token which stands for separation. The token embeddings translate the words to their vocabulary IDs. The segment embeddings is a numeric class which helps to identify the sentences (for example in Figure 1, what belongs to sentence A and sentence B). The position embeddings specify the position for each word in a sentence [3].

2) Model description: In this project a Grammar Error Correction Tag: Not Rewrite (GECToR) system developed by Omelianchuk et al. was used [4]. As the original system was developed for English grammar correction it has been adapted to Swedish language by series of updates as well as contributions to the base code. GECToR is a sequence-to-label system where custom token-level transformations are to be predicted instead of usual sequence-to-sequence where entire sentence is to be predicted. Omelianchuk et al. have shown that such a configuration increases the number of grammar error corrections while decreasing vocabulary size used when training.

In this project a pre-trained BERT model (on 200M sentences consisting of around 3000M tokens coming from Swedish books, news, government publications, wikipedia, and internet forums) from HuggingFace was used [5]. The Swedish BERT is trained with the same parameters as a base model described in Devlin et al. where 110M parameter and whole word masking is used [3]. Word representations of the model are then forwarded to two linear layers with softmax layers. These two linear layers are responsible for detecting the error and tagging the error respectively. When performing error detection, one of four different tags can be predicted; *CORRECT, INCORRECT,* *@@UNKNOWN@@*, *@@PADDING@@*. When performing error tagging, a tag can be chosen among vocabulary that, in this project, consisted of 10000 tags or it can be labeled as *@@UNKNOWN@@*, or *@@PADDING@@*.

3) Preprocessing: In order to train GECToR a data must be preprocessed in a certain way. Source and target, where source are errorful sentences and target are error-free sentences,

[1]https://en.wiktionary.org/wiki/Appendix:Swedish_verbs

must be compared. While comparing those two sets, a file with transformations from source to target is created. There are two types of transformations used in GECToR: *basic transformations* and *g-transformations*.

Basic transformations:

- **$KEEP**: token is preserved from source to target, meaning no changes to the token are made
- **$DELETE**: token from the source is deleted and does not occur in the sequence anymore
- **$APPEND**_$t_1$: an entirely new token t_1 is appended next to some token x_i in the target sequence.
- **$REPLACE**_$t_2$: some token x_i from the source sequence is replaced by token t_2 in the target sequence

G-transformations:

- **$CASE_suffix**: a token casing is changed depending on the *suffix* e.g., *$CASE_LOWER*, *$CASE_CAPITAL_1*.
- **$MERGE**: two consecutive tokens are merged into one token
- **$SPLIT**: a token from source sequence is split into two separate tokens in target sentence
- **$VERB**_$form_1$_$form_2$: a token that is a verb is transformed from $form_1$ is source sequence to $form_2$ in the target sequence, where $form_1$ and $form_2$ are tense tags of that verb.

GECToR needs a list of verbs in order to make use of *$VERB*_$form_1$_$form_2$ tag. From the verbs scraped from Wikipedia a list consisting of 19420 verb transformations of $form_1$_$form_2$:tag_1_tag_2 form was created, where $form_1$ and $form_2$ are verbs and tag_1 and tag_2 are tense representations of these verbs. Possible forms and tags are following:

- infinitive ($tag = VB$)
- present indicative singular ($tag = VBG$)
- past indicative singular ($tag = VBD$)
- past indicative plural 1st and 3rd person ($tag = VBZ$)
- supine ($tag = VBP$)
- past participle ($tag = VBN$)

The list consisting of all possible transformation pairs is saved and used while preprocessing, training, and predicting.

4) Training: For the training a cased pre-trained Swedish BERT model[2] in its base configuration was used. As an already pre-trained model was used, the stage of training was considered as fine-tuning this model as well as training the classifier layers responsible for error detection and error tagging.

The dataset was split 98/2% for training and validation sets. When training, the model was saved to a file after each epoch. The validation set was used to determine what model of those saved models performed best and also to determine if training should be stopped as GECToR was already equipped with the implementation of an early stopping mechanism. There were five models trained on different amount of data in total, and combinations of freezing and unfreezing the pre-trained word embeddings were tested. While freezing the word embeddings, only classification layers were trained, and the training process was two-fold faster. A complete set of training scenarios can be seen in Table I.

[2]https://huggingface.co/KB/bert-base-swedish-cased

TABLE I
SPECIFICATION OF TRAINING FIVE DIFFERENT MODELS

No. of sentences	No. of epochs	Fine-tuned word embeddings
10k	10	YES
1M	6	YES
2M	8	NO
2M	6	YES
4M	2	YES

5) Tagging and Predicting: In order to perform grammar error correction, a file of raw erroful sentences must be fed into GECToR's predictor. For each sentence a set of token transformations, described in subsubsection II-B3 is predicted. The transformations are applied to the sentence and an output file with corrected sentences is produced as an output of the predictor. Prediction of the transformations is paralleled for each sentence; that could be influencing the transformations if the output of the predictor was treated as an input in future iterations. Therefore a default value of five iterations is used when correcting sentences and predicted sentences are built iteratively.

C. LSTM

The Long short-term memory (LSTM) [6] is a type of recurrent neural network (RNN) architecture. RNNs have been successfully used in NLP thanks to their ability to learn long-distance dependencies in input sequences - for example long-distance dependencies between words at different positions in a sentence. Furthermore, the LSTM was designed to solve one of the problems with RNNs that can occur during training: vanishing and exploding gradients when performing backpropagation through time.

LSTMs can be used to implement an encoder-decoder architecture; suitable when working with a sequence to sequence learning problem such as machine translation [7]. A model based on the encoder-decoder architecture takes as input a sequence of variable length (e.g., a sentence) which is fed through the *encoder*. The encoder - consisting of an LSTM of specified number of layers - produces, at each time step, a hidden state and a cell state and takes as input the embedding of the current word and the previous hidden and cell state. When the end of an input sequence is reached, a final hidden and cell state have been produced by the encoder and are used as the initial hidden and cell state of the *decoder* (also LSTM). The decoder also takes the previous hidden and cell state as input at each time step, together with an embedded word. This word is either the correct output word as per the ground

truth output sequence, or the previously predicted output word, depending on if teacher forcing is activated. At each time step, the states produced by the decoder can be sent through a linear layer to produce output words predictions.

1) Beam search: Beam search is a heuristic search algorithm that expands the best x number of nodes (beam width) at each level of the search tree using breadth first-search, while all other nodes are removed from consideration. In the context of a sequence-to-sequence model the encoder-decoder structure produces a probability for each word in the output vocabulary each iteration. Beam search can be used to determine the most likely words and be used to guide the decoding and determine the most likely final sentence.

2) Implementation: A sequence-to-sequence model was implemented based on a model developed by Sutskever et al. [8]. The model implements an encoder-decoder architecture where the encoder and the decoder both consist of multi-layered LSTMs. Beam search was implemented on the output of the model to predict the final sentence.

In Figure 2 an overview of the implemented model can be seen. The input to the model is a vector consisting of integers representing words. The input vector is pre-processed to start with a beginning of sequence token (BOS-token), end with an end of sequence token (EOS-token), and only the 30,000 most common words (vocabulary) are allowed in sentences - otherwise the word is replaced with an unknown token (UNK-token). During training, the model is trained on batches of sentences, requiring the sentences to be of the same length. This is accomplished by sorting the sentences based on length and adding padding tokens (PAD-token) to the end of sentences shorter than the longest sentence in the batch.

During training, the pre-processed sentences are fed to the encoder which consists of an embedding layer and a multi-layer LSTM. The output of the encoder is given to the decoder where the previous predicted word (or BOS-token, if the first iteration) is given to the LSTM and the next word is predicted. Teacher forcing is used with a probability of 0.5 for each word in the sentence, meaning that the previous token is either the correct previous word or the predicted one. The output of the LSTM is fed through a linear layer and for each index of the output sentence the probability of each word in the vocabulary is predicted. This is then, combined with the true correct sentence, used to update the weights of the model using an optimizer.

When using the model, for example during evaluation, beam search is used to choose the best sentence. Sentences are scored by multiplying the probabilities of each word in the sentence based on previous words and dividing with the length of the sentence.

In the embedding layers of the model, frozen pre-trained weights [3] were used in order to speed up the training time.

Additionally there are a number of hyperparameters that were kept static during all of the experiments:

- Source/target vocabulary size: 30,000
- Embedding dimension: 300, pre-trained and frozen
- Batch size: 80
- Hidden dimension: 512
- Learning rate: 1e-3

[3]https://www.ida.liu.se/divisions/hcs/nlplab/swectors/

D. Generative Adversarial Training

Generative Adversarial training is a class of machine learning where two models are trained simultaneously, a generator and a discriminator. The models are "pitted" against each other where the discriminator attempts to determine if a sample is from the generator or from the data set. The generator is then updated based on if the discriminator successfully could determine if the data points were from the generator or not.

The LSTM model, described in subsection II-C, is used as the generator model and a new model is designed and implemented for the discriminator. The implementation of the disicriminator and the combined training was based on the implementation by Raheja and Alikaniotis [9], that showed significant improvements on generative models for GEC using adversarial training. The rest of this section will detail how data was generated for training, how the discriminator was implemented, and how the combined training was implemented.

1) Discriminator Model: The discriminator is a binary classification model that attempts to predict if a "correct" sentence is from the dataset, and therefor actually correct, or if it is from the generator. As input to the discriminator the model takes a pair of sentences, one incorrect sentence and one correct sentence. While usually the discriminator in a GAN setup only takes only one input, the generated or real data value, in this case the discriminator takes both sentences [10]. This is both to make the task easier and to make it possible for the discriminator to separate a generated sentence that follows the distribution of our correct sentence data but does not properly represent the information in the incorrect sentence and a correct sentence from data. The output is a single value representing the probability that the correct sentence is real. The discriminator model consists of a GRU layer with hidden size 128 , a feed-forward (FF) network with one hidden layer of size 128 with ReLU as activation function, and a Sigmoid activation function for the output value to get probabilities out. On a forward pass both the incorrect sentence and correct sentence are sent through the same GRU layer separately, and the last hidden states are concatenated and then sent through the FF network. The discriminator was pre-trained before being used for the combined training. The training was done with mini-batching, binary cross entropy as the loss function, and Adam [11] as the optimizer. After each epoch the validation loss and the accuracy on the validation data was used to evaluate the model.

2) Dataset for the discriminator: The discriminator is pre-trained before the combined training. And in order to pre-train the discriminator, the discriminator needs a dataset with both sentences from the dataset and sentences generated by the generator. So the first task is to generate data based on ground truth data from subsection II-A and a pre-trained generator from subsection II-C. For each sentences pair $<$*incorrect*

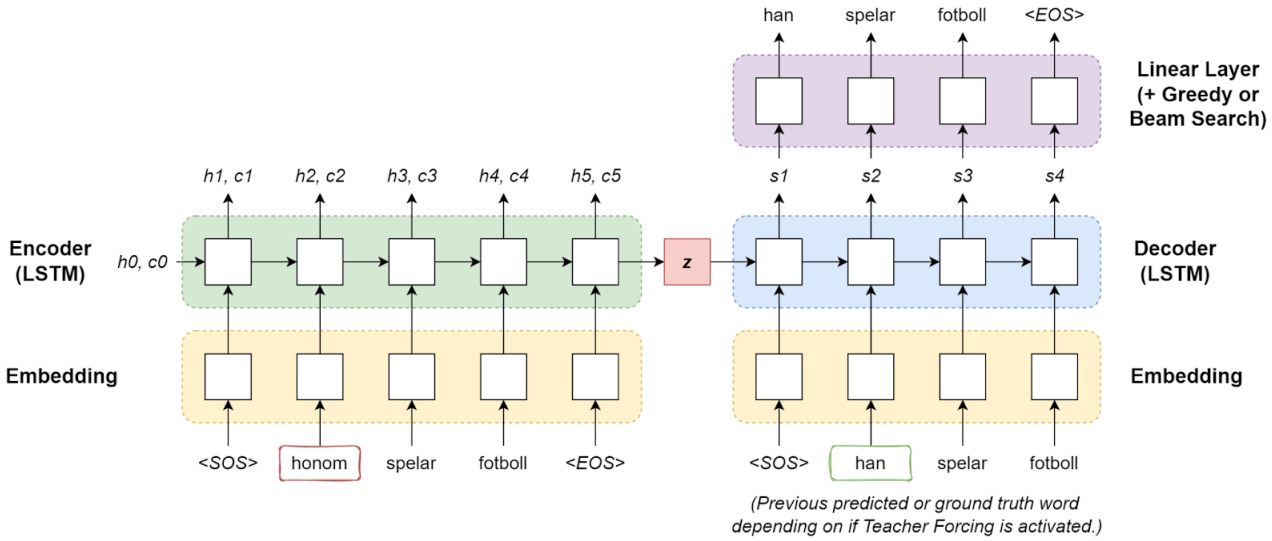

Fig. 2. LSTM-model overview

sentence, correct sentence> in ground truth data, the *incorrect sentence* is fed into the generator. The generator generate a corrected sentence, with greedy decoding without teacher forcing, that is used in the dataset. Note that the pre-trained generator used to create the dataset is exactly the same as the generator used in the combined training. For each pair of sentences in ground truth data, two pairs of training data sentences will be generated for discriminator training:

1) <*incorrect sentence, corrected sentence*> with target 0.
2) <*incorrect sentence, corrected sentence*> with target 1.

The target for each sentence pair indicates whether the *correct sentence* is from ground truth or the generator.

3) Combined Training: The combined training procedure between the generator and discriminator is often volatile [9]. In order to stabilize this training, both models need to be pre-trained before the combined adversarial training. Firstly, the generator model must be pre-trained on the ground truth data; in our case, the LSTM model must be trained on the incorrect to correct sentences, as described in subsection II-C. This training must be conducted until convergence in order for the combined training to be more stable, faster and to ensure convergence. The pre-training of the discriminator, on the other hand, is not as straightforward. The discriminator should be pre-trained on a generated dataset, as describe in subsubsection II-D2. The combined training is heavily dependent on the relation between the performances of the two models, and as the discriminator often converges faster than the generator, the optimal performance relationship is pretty hard to predict beforehand. Raheja and Alikaniotis [9] conducted several combined training with discriminator models with accuracies between 60 % and 90 % and found the most optimal initial accuracy of the discriminator to be 75 %, but anywhere in between 65 % and 85 % showed good performances as well. At convergence of the combined training, the discriminator should not be able to distinguish between ground truth data and data generated by the generator, yielding an accuracy of around 50 % [12].

The actual combined training is based on using the generator to create a correct sentence from an incorrect sample, and the discriminator will be used to predict the probability of the current sample being from the generator, which is used to calculate the loss for the generator [9]. If the probability from the discriminator is high, the error for the generator will be amplified, and if the probability is instead low, the loss will be minimized. This way, the generator will learn to trick the discriminator, hopefully improving its general performance on the GEC task. More details can be seen in Algorithm 1.

E. Evaluation

To evaluate the performance of all the models, precision (see Equation 1), recall (see Equation 2), f-score (see Equation 3), and accuracy (see Equation 4) were implemented and used to compare the models:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F_{score} = \frac{(1 + \beta^2) \cdot precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

To be able to calculate above metrics, notions of *true positives (TP)*, *true negatives (TN)*, *false positives (FP)*, and *false negatives (FN)* predictions had to be defined. Definitions of those are the following:

- TP: correction was performed and predicted sentence is the same as ground truth sentence

Algorithm 1 Adversarial Training

$G, D \leftarrow pretrain(G), pretrain(D)$
while not converged **do**
  $\# \; train \; generator$
  $y', y'_{prob} \leftarrow G(x_i, teacher\_force = 0)$
  $prob\_generated \leftarrow D(x_i, y')$
  if $random() < 0.4$ **then**
    $reward \leftarrow log_e(prob\_generated)$
    $reward \leftarrow reward - AvgRewLast100Runs$
    $G_{loss} \leftarrow log_e(y'_{prob}) * rewards$
  else
    $y', y'_{prob} \leftarrow G(x_i, teacher\_force = 1)$
    $G_{loss} \leftarrow CrossEntropy(y', y)$

  $\# \; train \; discriminator$
  $prob\_y\_generated \leftarrow D(x_i, y)$
  $all\_probs \leftarrow cat(prob\_generated, prob\_y\_generated)$
  $targets \leftarrow cat(0, 1)$
  $D_{loss} \leftarrow BinaryCrossEntropy(all\_probs, targets)$

  $\# \; update \; weights$
  $G_\theta \leftarrow G_\theta - \alpha * G_{loss}$
  $D_\theta \leftarrow D_\theta - \alpha * D_{loss}$

- TN: correction was not performed and predicted sentence is the same as ground truth sentence
- FP: correction was performed and predicted sentence is not the same as ground truth sentence
- FN: correction was not performed and predicted sentence is not the same as ground truth sentence

In this project only F-score for $\beta = 0.5$ and $beta = 1$ was used.

Furthermore, to know how a sentence length influences the model performance, the above evaluation metrics were calculated separately for sentences with: 0-10 words, 10-20 words, 20-30 words, 30-40 words, 40-50 words, as well as for all the sentences in total to see the overall scores.

III. Results

A. BERT

The BERT models were trained on different amount of sentences, in table II it is noticeable that the model trained on 4 million (M) sentences had the best overall performance. However, in shorter sentences with length 0-10 the model trained on 1M was the best. In total there were 5 models and 1 of these was trained with frozen embeddings. After seeing no gain from frozen embeddings, the others was trained with unfrozen embeddings. One can remark that all the models performed poorly on correcting long sentences, this will be discussed further in the discussion section.

B. LSTM

Three different LSTM models were trained where different amount of training data and number of layers in the LSTMS were used:

Fig. 3. Validation and training loss of lstm models

- **LSTM_1:** 1 million sentences used during training 2 layers in each LSTM
- **LSTM_2:** 1 million sentences used during training 4 layers in each LSTM
- **LSTM_3:** 2 million sentences used during training 2 layers in each LSTM

The result of the training can be seen in Figure 3. The result of the evaluation can be seen in Table III. In addition, the effect of using different beam width were evaluated and can be seen in Table IV.

C. Generative Adversarial

As previously mentioned, before the combined training the discriminator was pre-trained to a certain accuracy. A pre-trained accuracy of 70% was selected for the combined training. The combined training converged after only two epochs and both the discriminator and the generator validation loss started increasing quickly. Especially the discriminator loss doubled from the second to third epoch. The resulting generator had a validation loss almost 10% lower than before the combined training. This decrease however is not represented in the evaluation metrics, in Table V we can see the metrics for the generator model after combined training and the results are identical before and after combined training.

IV. Discussion

A. Dataset Generation

The dataset contains synthetic data and therefore the models should not reflect normal grammar errors. The fact that synthetic data were used also means that models trained on it can not be compared with models trained on real datasets since the error distribution and the types of errors is not comparable. When generating errors, there is no context of what have been change and multiple changes can happened to the same sentence and word. An example of this that were found, is the following: “uppenbar” → “uppen bar” → “uppen bär”. In the first split, the generator splits the word and in the second step it changes the verb form on the second word. As one can notice, it’s hard for the average person to notice this error as

TABLE II
RESULTS FROM BERT MODELS WHERE THE TOP DESCRIBES SENTENCE LENGTH AND * DENOTES MODELS TRAINED WITH FROZEN WORD EMBEDDINGS

	[0-10]					[10-20]					[20-30]				
% Sentences	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
10 k	0,52	0,745	0,553	0,612	0,699	0,322	0,719	0,362	0,445	0,409	0,187	0.737	0.22	0.298	0.218
1 M	**0,591**	**0,93**	**0,637**	**0,723**	**0,749**	0,472	0,967	0,526	0,634	0,539	0,375	0,985	0,428	0,543	0,397
2 M	0,565	0,932	0,614	0,703	0,728	0,472	0,967	0,526	0,635	0,539	0,386	0,984	0,44	0,555	0,408
2 M*	0.442	0.508	0.454	0.473	0.633	0.285	0.511	0.313	0.366	0.36	0.152	0.524	0.177	0.236	0.179
4 M	0,582	0,927	0,629	0,715	0,739	**0,476**	0,966	**0,53**	**0,638**	0,539	**0,39**	0,985	**0,444**	**0,559**	**0,411**

	[30-40]					[40-50]					Total				
% Sentences	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
10 k	0,1	0.729	0.121	0.176	0.11	0.055	0.676	0.067	0.102	0.06	0.335	0.731	0.376	0.459	0.478
1 M	0,275	0,991	0,321	0,339	0,281	0,204	0,996	**0,339**	0,243	0,206	0,447	0,957	0,53	0,637	**0,587**
2 M	0,291	0,992	0,339	0,45	0,298	0,207	0,994	0,246	0,343	0,209	0,473	**0,958**	0,527	0,634	0,581
2 M*	0.076	0.511	0.092	0.132	0.084	0.043	0.478	0.053	0.079	0.046	0.282	0.511	0.31	0.363	0.425
4 M	**0,304**	**0,993**	**0,353**	**0,465**	**0,31**	**0,215**	0,996	0,255	**0,354**	**0,217**	**0,481**	0,956	**0,534**	**0,64**	0,586

TABLE III
RESULTS FROM LSTM MODELS WITH GREEDY SEARCH DECODING, WHERE THE TOP DESCRIBES SENTENCE LENGTH, NOTE THAT DIFFERENT LENGTH FOR SENTENCES WERE NOT EVALUATED FOR LSTM_1

	[0-10]					[10-20]					[20-30]				
Model	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
LSTM_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LSTM_2	0.131	0.943	0.159	0.231	0.280	0.004	0.978	0.005	0.008	0.005	0.000	0.000	0.000	0.000	0.000
LSTM_3	0.140	0.942	0.169	0.244	0.295	0.000	0.953	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

	[30-40]					[40-50]					Total				
Model	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
LSTM_1	-	-	-	-	-	-	-	-	-	-	0.041	0.938	0.051	0.079	0.102
LSTM_2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.045	0.944	0.056	0.086	0.107
LSTM_3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.046	0.942	0.057	0.087	0.110

it's not a normal grammar error. The are no limit on how many changes a sentence can contain, thus it's more likely for long sentences to contain many errors. Which can be noticeable in the result since many of the models have lower score on long sentences compared to short. Another problem to note is that when splitting the data into training- and test-dataset, no statistical evaluation were conducted to verify that they have the same properties.

B. BERT

When evaluating the BERT models, there were a lot of FNs which necessarily does not mean that the corrected sentences were wrong. As there are many ways to correct a grammar error and this project only checked if it was corrected to the original sentence. As expected, the model trained on 10k performed poorly. This is probably due to the fact of too little data and the performance comes mostly from the fact that the model is pre-trained. After doing experiments with the different models, it is observable that when using frozen embedding (see 2M* in the table), the gain over computational cost for performance is poorly and clearly not worth it in this case. Although the models with more training data performed better, as can be seen in table II, using more data gave diminishing returns on performance compared to the increase in computation time. When having short sentences, there is no need for a model trained on a lot of data. However, when a sentence becomes longer it is better to use a model trained more data. The group did not have any measure on the dataset and therefor it is hard to tell whether BERT becomes better with more training or if it is the fact that the data contains more long sentences after a certain amount of data. With that said, the dataset is not sorted on sentence length so one needs to have in mind of randomness when evaluating different models. BERT performed good overall and the fact that all errors that have been generated in the scope of GECToR makes it a little bit unfair against the other methods. After having no knowledge about creating grammar errors, a lot of inspiration was taking from their paper. Which makes the fact that BERT should performed good an obvious reason.

C. LSTM

As can bee seen in Table III, the LSTM model performed quite poorly with the best configuration only obtaining a total accuracy of 0.110. There are multiple factors that could have contributed to these results. Firstly, training the different model configurations took quite a bit of time (approximately 61 hours) which made hyperparameter tuning impractical. For example, the hyperparameter *vocabulary size* was limited to 30,000. Had the vocabulary size been larger, the number of predicted sentences containing UNK-tokens could have been decreased, which would have led to fewer sentences being automatically evaluated as incorrect irregardless of the model's ability to correct grammar. This is potentially influenced further by the dataset which among other things contains

TABLE IV
RESULTS FROM LSTM MODELS DEPENDING ON BEAM WIDTH FOR LSTM_3

Beam width	P	R	$F_{0.5}$	F_1	A
1	0.046	0.942	0.057	0.087	0.110
2	0.047	0.942	0.058	0.089	0.111
4	0.047	0.942	0.058	0.089	0.112

TABLE V
RESULTS FROM LSTM MODEL AFTER COMBINED TRAINING WITH GREEDY SEARCH DECODING WHERE THE TOP DESCRIBES SENTENCE LENGTH.

	[0-10]					[10-20]					[20-30]				
Model	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
Best Adversarial	0.122	0.936	0.148	0.216	0.269	0.003	0.985	0.004	0.006	0.004	0.000	0.000	0.000	0.000	0.000
	[30-40]					[40-50]					Total				
Model	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A	P	R	$F_{0.5}$	F_1	A
Best Adversarial	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.041	0.938	0.051	0.079	0.102

news articles' text which often contains uncommon words like names and places. Secondly, since pre-trained embeddings were used, a word in the vocabulary without a pre-trained embedding was initialized randomly. And since the embeddings were frozen, the random initialization values were kept throughout training and likely had a negative effect on the performance.

All model configurations perform considerably better on short sentences than long. This might be due to the LSTM not being able to preserve the context as the sentence gets longer. It might also be due to a sentence being more likely to contain errors and UNK-tokens the longer it is and therefore also harder to correct.

Increasing the beam width during the beam search improved the performance of the models (as can be seen in Table IV), although the increase was quite small. This might be due to the overall bad performance of the model and the binary nature of the evaluation correction; with a larger beam width, the model may predict a more correct sentence but not a completely correct one and therefore not increase the performance.

D. Generative Adversarial Training

When first implementing the discriminator model the same hyperparameters were used as Raheja and Alikaniotis [9] used in their discriminator. When pre-training however this showed to be a too complex model and even after multiple epochs of training the discriminator had not improved at all, so the complexity had to be lowered. Since the generator model did not achieve results comparable with that of Raheja and Alikaniotis model, the lower complexity discriminator was still able to achieve good results and after only a few epochs the discriminator achieved as high accuracy as needed. So the lower complexity did not limit the discriminator; however, we expect that our discriminator would not preform well given a better generator, e.g., the BERT model.

As previously mentioned, the combined training is quite volatile. If the discriminator is too good/bad compared to the generator when starting training, the training will not be stable, and if one of the models improve a lot quicker during the combined training the training will also not be stable. To get stable training there are multiple parameters that can be changed, the learning rate of the generator and discriminator model can be set individually and we can tune how often teacher forcing MLE is used to update the generator to try and stabilize the training. So, there are multiple parameters that can be changed to get the best results, or even to get stable training at all. Because of this grid search or random search would be suitable to find a good set of parameters. However, the computational-, and time limitations during the project meant that we could not optimize the parameters, and this most likely heavily affected the results of the combined training. It is also especially hard to estimate good parameters because they are very dependent on the performance and complexity of both the discriminator model and the generator model. Comparing parameters with e.g. Raheja and Alikaniotis is not very useful because our generator have a lot worse performance and our discriminator has a lot lower complexity.

It is also worth mentioning that the combined training is used as a fine-tuning training, since the generator model is already trained to convergence before the combined training. And it might be the case that the performance of the LSTM model is not good enough for the combined training to be efficient, and that a better model whose output better predicts the target output is needed for the combined training to work. However, without properly testing parameters for the combined training it is hard to say what is stopping the combined training for improving the generator.

V. CONCLUSION

In conclusion when comparing the performance of the different models it can been seen that the BERT model is considerably better than the LSTM model, this might indicate that a model based on error tagging is more suited for grammar error correction than a sequence to sequence based model since it avoids the problem of out of vocabulary errors. Using adversarial training did not result in improvements of the LSTM model in the metrics but if the LSTM model would have been better the adversarial training might have

have yielded better results. It is noteworthy that a model trained from scratch (e.g. the LSTM-model) means longer training times is required then for a pre-trained model (e.g. the BERT model), in a context where computational and/or time resources is limited it is therefore more suitable to use a pre-trained model.

It can also be concluded that all the models is better at correcting short sequences which might be due to longer sequences having more errors as well as an increased opportunity of multiple errors happening to the same words, completely changing the meaning of a sentence.

Synthetic data has been used during the project which means the results is not comparable with existing related research since the types of errors and their distribution might not match with real ones.

In the future it might be interesting to investigate the difference in performance when using synthetic data and real data for training. Another interesting future research direction could be to evaluate models on a per error type basis.

REFERENCES

[1] Nora Madi and Hend S. Al-Khalifa. Grammatical error checking systems: A review of approaches and emerging directions. In *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*, pages 142–147, 2018.

[2] University of Gothenburg Department of Swedish. The swedish culturomics gigaword corpus.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. Gector – grammatical error correction: Tag, not rewrite, 2020.

[5] bert-base-swedish-cased. https://huggingface.co/KB/bert-base-swedish-cased. Accessed: 2020-12-15.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. Chapter 9.6. Encoder-Decoder Architecture.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[9] Vipul Raheja and Dimitris Alikaniotis. Adversarial Grammatical Error Correction. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3075–3087, Online, November 2020. Association for Computational Linguistics.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.