Neural Text Simplification

Davide Mercanti (davide.mercanti@studio.unibo.it)
Riccardo Fava (riccardo.fava6@studio.unibo.it)
Luca Bompani (luca.bompani4@studio.unibo.it)

NLP project + project work (3CFU)

1 Abstract

In our project we have addressed the neural text simplification task, that aims to reduce the linguistic complexity of text, both syntactically and semantically, in order to increase the readability and comprehensibility of content. We decided to address this problem by trying three different approaches, in order to experiment some specific techniques and make a better comparison between their performances and peculiarities.

First of all we built a baseline in order to become familiar with the problem, then we tried to improve our results by implementing more complex architectures. In the end we tried four different approaches for a total of three diverse architectures:

- RNN with attention mechanism
- Transformer model trained using a supervised learning
- Transformer model with a self supervised approach.
- GAN model

For this particular task, several proposals exist in literature for what concerns the English language; however, we decided to rely on datasets based on Italian corpora in order to experiment how neural models can solve the text simplification problem in a language different from the one predominant among existing investigations.

To obtain the Italian gold standard, we merge two available datasets composed of complex–simple pairs of sentences, namely the SIMPITIKI corpus and the PaCCSS-it corpus. For what concerns the self-supervised approach we have utilized also a simple Italian corpus called PAISÀ to enrich the training data with unseen text lines.

The results we obtained in our experimental setup were not so satisfying based on the SARI and BLEU metrics. From our point of view, this is due to several problems that we encountered, but mainly the scarcity and not-so-high quality of the data that we used.

2 Background

Text simplification (TS) aims at making a text more readable by reducing its lexical and structural complexity while preserving its semantics, at least for the key concepts.

A simplified version of a text could benefit users with several reading impairments, such as non-native speakers, people with aphasia, dyslexia, or autism. Simplifying a text automatically could also help improve performance on other language processing tasks, such as parsing, summarization, information extraction.

Neural approaches to the task have gained increasing attention in the NLP community thanks to recent advancements of deep, sequence-to-sequence approaches. However, almost all recent improvements have dealt with English. The main reason is that such data-hungry approaches require large training sets (in the order of hundred thousand instances) and sizable datasets have been developed and made available only for this language.

In fact in order to obtain a large enough gold standard, we decided to merge two available datasets, namely:

- The **SIMPITIKI** corpus, a manually curated corpus with 1,166 complexsimple pairs extracted from Italian Wikipedia and from documents in the administrative domain.
- The **PaCCSS-it** corpus, which contains 63,000 complex-to-simple sentence pairs automatically extracted from the Web.

Instead **PAISÀ** is a large corpus (about 250 millions of tokens) of authentic contemporary Italian texts extracted from the web and automatically cleaned-up.

Most of the early work in the text simplification field involved extractive methods of summarization, taking the sentences from a document that conveyed the most meaning. As research in NLP has exploded

over the past decade, simplification has shifted towards abstractive approaches, actual generation of text. Initially this involved sentence-level simplification through lexical (word-based or phrasal-based) selection and substitution, for example by using the Paraphrase Database.

In recent years, text simplification has evolved to actual generation of new and novel text, thanks to the advent of neural networks and especially Recurrent Neural Networks (RNNs) and Transformers, which allow effective sequence-to-sequence (seq2seq) modeling. Due to the similarities between the two tasks and to the not so widespread development of specific architectures for text simplification, we decided to take inspiration from the neural machine translation (NMT) field. In fact both TS and NMT are text-to-text tasks which involve the computation of a sentence starting from another one. The network architectures which are typically employed follow the Encoder/Decoder pattern. The encoder part squeezes out the structure semantics in order to obtain a compressed and dense representation of the original sentence. The decoder instead utilizes the compressed representation to produce the output of the model. These kinds of models are based on RNNs and, in particular, thanks to the concept of memory they are able to exploit the typical sequential structure of the text. After the invention of Transformers, this type of architecture became the most effective and widespread in most NLP tasks, TS and NMT included. The main feature of Transformers is the attention mechanism, which makes the network able to focus only on small, but important, parts of the data.

Because lack of data to train on remains one of the major problems for TS, most of the recent advancements in research has been to overcome this problem. In addition to creating synthetic data, developing unsupervised techniques for TS has also been of keen interest. One of the most successful architectures that use a different approach with respect to the previously mentioned ones, are the Generative Adversarial Networks (GANs). The core building block is a conditional sequence generative adversarial net which comprises two adversarial sub models: Generator and Discriminator. The Generator aims to generate phrases that are difficult to discriminate against the human-simplified sentences; the Discriminator makes efforts to discriminate the machine-generated sentences from human-simplified ones.

3 System description

3.1 Data handling

Before merging the two corpus to obtain our dataset, we applied some preprocessing steps in order to come up with higher-quality data. First of all we applied a cleaning of the sentences, removing special characters, tags useless for our purposes and misspellings. Then we eliminated the rows with missing values. Lastly we merged and shuffled together the two corpus, coming up with 64,172 couples of complexsimple sentences.

In order to better manage the dataset we decided to encode it into a Pandas Dataframe, where each row represents a different couple of complex and simplified sentences. In the case of the PAISÀ dataset we didn't utilize the Dataframes, instead, we relied directly on the Text Dataset utilities of TensorFlow. Then we splitted our data into training, validation and test set.

We designed slightly different versions of text-processing pipelines based on the various versions of model architectures that we implemented for this task. In general we have first tokenized the sentences using the NLTK tokenizer, then we built a vocabulary based on the words present in the corpus. Later we converted each token of the dataset into an index (integer value); in this way we have been able to feed the network with our data.

After this encoding step we computed an embedding matrix in order to allow our networks to start the learning with already trained word embeddings. We extracted the word embeddings from a pretrained FastText italian model, suitably reducing the size of its embeddings from 300 to 200.

These phases were the same for each model. Instead, we adopted a different handling of the OOV terms depending on the different models. For what concerns the baseline, the Transformer (supervised) and the GAN we relied on the FastText utility as it allows us to retrieve the OOV word embedding based on the similarity between character-level n-grams. Regarding the Transformer model trained with a self-supervised approach, while still relying on the FastText utility for OOV words, we introduced a

special token unknown as the size of the vocabulary, using also the PAISÀ corpus, would have been far too massive to allow training.

In general, we have introduced word embeddings for the OOV coming from the whole dataset, including the test set. We are aware that in this way we have inserted some test knowledge within our training dataset, but given the nature of the embedding layer (a retrieval) this fact does not influence our results. So, given the complexity of our original task, we have decided to simplify the OOV problem, which would have to be dealt with differently in a real scenario.

Another difference is that in the text-processing pipeline of the Transformers models we prepended the special token "[start]" and we appended the token "[end]" to each simplified sentence of the dataset. This, because our Transformer architecture is capable of reasoning and generating text one token at time; using these tags allow us and the network to know when we reach the boundaries of the sentences.

In the end before feeding the networks with the processed text, we applied a padding of the sentences in such a way that we can handle the majority of our data (95-99%). This allowed us to perform the training of our models using batches of data. For the GAN architecture, a smaller padding (or truncation) has been used, due to computational limitations.

3.2 Models

3.2.1 Baseline (RNN with attention)

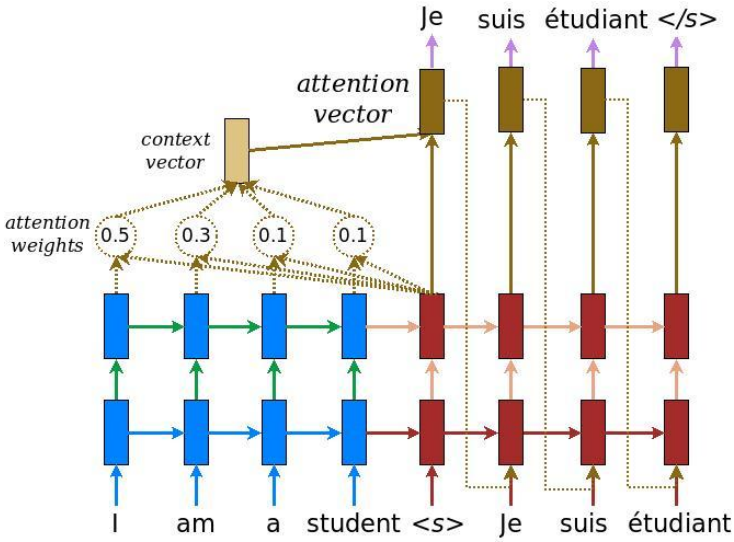
Taking inspiration from the neural machine translation field we decided to implement as baseline a typical sequence-to-sequence Recurrent Neural Network (RNN) with Bahdanau Attention [1].

In particular, the architecture of this model is based on an Encoder/Decoder structure. At each time-step the decoder's output is combined with a weighted sum over the encoded input, to predict the next word.

The encoder is composed by an embedding layer that looks up an embedding vector for each token of the input sentence, and by a GRU layer that processes those vectors sequentially. The outputs are: the processed sequence that will be passed to the attention head and the internal state that will be used to initialize the decoder.

The decoder uses attention to selectively focus on parts of the input sequence. This attention mechanism takes a sequence of vectors as input for each example and returns an "attention" vector for each example.

The decoder aims to generate predictions for the next output token. It is composed of an embedding layer that converts token IDs into vectors, followed by a GRU layer that keeps track of what's been generated so far in order to maintain coherence. Then the GRU output is used as the query to the BahdanauAttention mechanism over the encoder's output, producing the context vector.

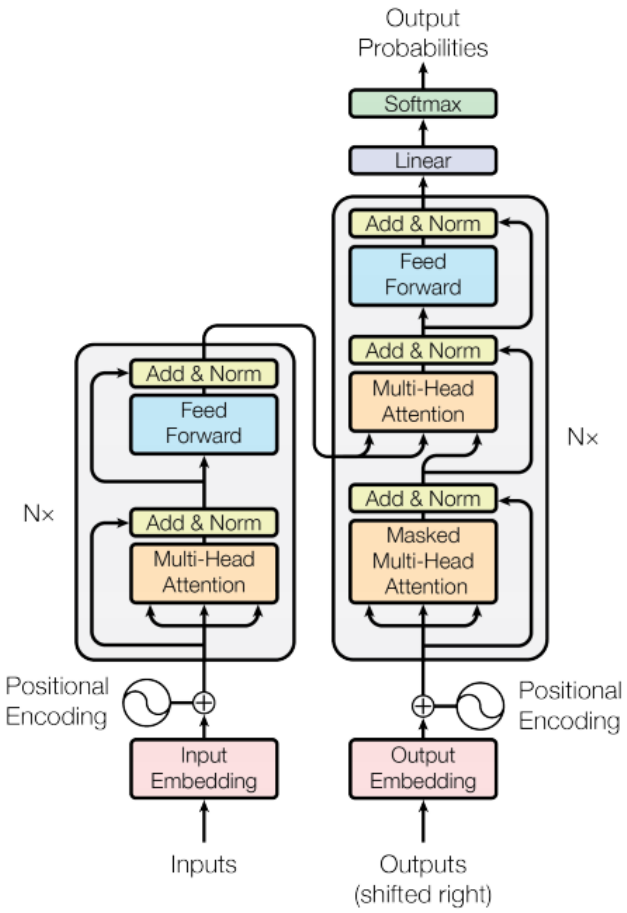


RNN architecture with attention.

Lastly, the decoder combines the GRU output and the context vector to generate the "attention vector", that is used for computing the prediction for the next token thanks to a last fully connected layer.

3.2.2 Transformer (supervised)

Our sequence-to-sequence Transformer [2] consists of an Encoder and a Decoder chained together (see fig. 2). The Transformer differs from the Recurrent Networks models by processing the elements as one-piece, which causes it to lose the information on the order of the elements in the sequence. This information is significant for the next steps; therefore, before both the Encoder and the Decoder is used a function that adds to the embedding vector another part that is called Positional Encoding. This function produces an index that shows the precise words location in the sentence.



Transformer architecture

The Encoder is composed of 2 parts:

- Multi-Head Self-Attention Mechanism: implementation of multi-headed attention as described in the paper "Attention is all you Need" (Vaswani et al., 2017). The principle of self-attention is to check how each word in a sentence is related to the other words.

- Fully-Connected Feed-Forward Network: two fully-connected layers followed by ReLU activation function. It takes each word in the input sentence, processes it to an intermediate representation and compares it with all the other words in the input sentence. The result of those comparisons is an attention score that evaluates the contribution of each word in the sentence to the key word. The attention scores are then used as weights for words representations that are fed to a fully-connected network that generates a new representation for the key word.

The new learned representation will then be passed to the Decoder, together with the target sequence so far (target words 0 to N). The Decoder will then seek to predict the next words in the target sequence (N+1 and beyond).

The Decoder is composed of 3 parts:

- **Masked Multi-Head Self-Attention Mechanism**: Unlike the Encoder, the Decoder uses an addition to the Multi-head attention that is called masking. This operation is intended to prevent exposing posterior information from the decoder. It means that in the training level the Decoder doesnt get access to tokens in the target sentence that will reveal the correct answer and will disrupt the learning procedure. It's a really important part of the Decoder because if we do not use the masking the model will not learn anything and will just repeat the target sentence.
- **Multi-Head Self-Attention Mechanism**
- **Fully-Connected Feed-Forward Network**.

Both the Encoder and the Decoder use a residual connection, followed by an addition of the original input of the sub-layer and by a batch normalization layer. At the end of the Decoder we have a dense layer large as the size of our vocabulary, with a softmax activation function.

This particular architecture has been trained using two different approaches, one supervised and one self-supervised.

3.2.3 GAN

The idea of this approach is to see the simplification task as a generative operation conditioned by the non-simplified input sentence.

Generative Adversarial Networks implement a min-max game between two networks, a discriminator D and a generator G. The discriminator learns from real and generated examples to minimize its loss, while the generator learns to maximize the same loss by generating (realistic) examples that are able to fool D.

Defining E the loss, g and d the parameters of respectively, G and D, we aim to solve:

$$max_{\theta g} min_{\theta d} E(\theta g, \theta d) \tag{1}$$

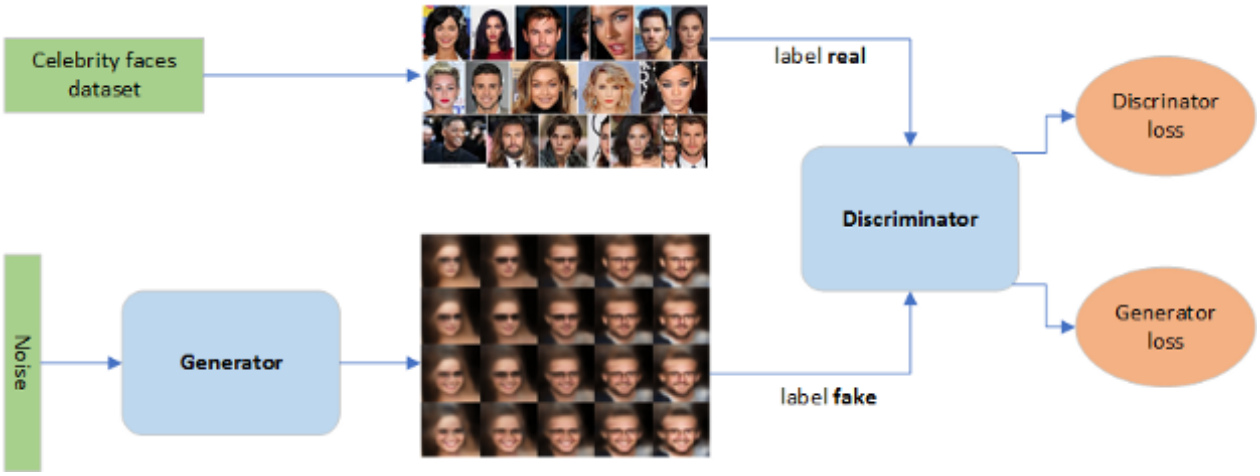
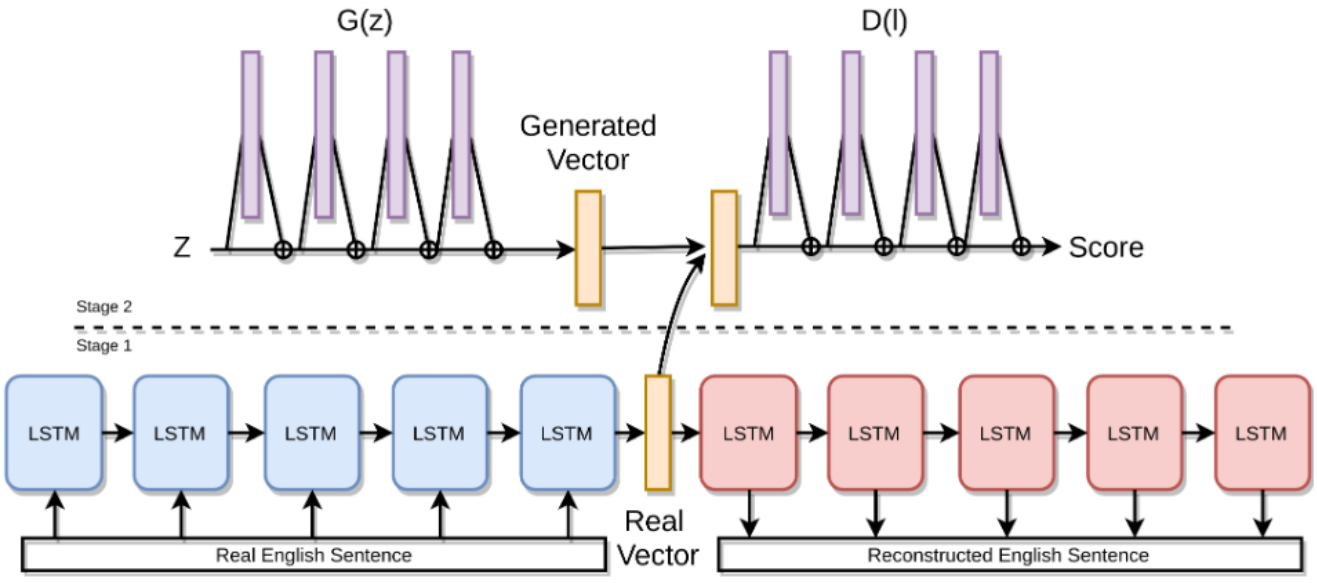


image: https://towardsdatascience.com/generative-adversarial-network-gan-for-dummies-a-step-by-step-tutorial-fdefff170391

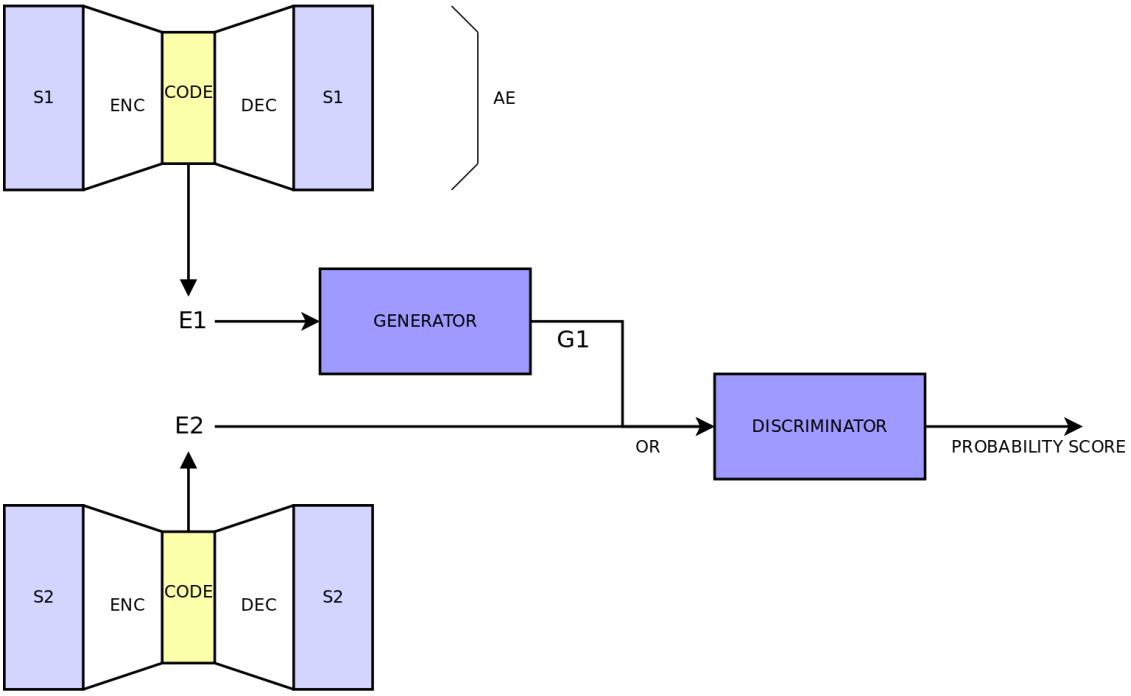
GANs are undoubtedly widespread for image generation and are able to produce noteworthy results for that purpose. Even if their use for text generation may seem obvious, the fact that text is intrinsically discrete poses some problems. A certain text token is usually generated as the result over a (continuous) distribution of the argmax function, that is non-differentiable and therefore hinders backpropagation.

With regard to text sequences generation, Donahue and Rumshisky in "Adversarial Text Generation Without Reinforcement Learning"[3] propose a method to overcome the non-derivability of the argmax function. Their solution consists of initially training an autoencoder (AE) and then rely on the code found, in place of the original input, to perform the generation task. This idea is summarized in the picture:



Autoencoder + GAN, as proposed by [3].

In our scenario, however, we wanted our network to learn the association between complex and simple sentences and, therefore, we used a Conditional GAN (CGAN) able to take as input the code of the complex sentence:



Scheme of our autoencoder + cGAN architecture.

Here, S1 and S2 are respectively the complex and simple sentences; E1 and E2 the corresponding code generated by the AE. After training, to get the simplified sentence one would do: DEC [GEN(ENC(s1))]

As suggested by Donahue and Rumshisky ([3]), both the encoder and the decoder of the AE consist of Long-Short Term Memory (LSTM) networks.

Instead, for the generator and the discriminator we have simply relied on dense networks, although this turned out to be a bad choice.

4 Experimental setup and results

4.1 Environment

The architectures are implemented relying on the Python TensorFlow framework with Keras. Since the models are really huge and heavy to train, we took advantage of Google Colaboratory, a platform well suited to deploy machine learning models, which allows a free use of their GPUs. In particular to train and evaluate our models we used the NVIDIA Tesla T4 GPU, with 16GB of RAM.

4.2 Metrics

- The **bilingual evaluation understudy** (BLEU)[5] has been introduced in the context of machine translation for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machines output and that of a human. In our case its main point is to measure the similarity between human simplified text and its automatic counterpart.

- The **system output against references and against the input sentence** (SARI)[6] is currently the main metric used for evaluating simplification models. Although a lexical simplicity metric, SARI measures the fit of added, deleted, and retained words by comparing the output to several simplified reference sentences.

4.3 Baseline (RNN with attention)

We implemented this architecture taking inspiration from the original paper [1]. From the beginning we had problems related to the high memory consumption of this network. We attempted to solve them in several ways; for example we tried to reduce the number of units of the encoder/decoder, to use a very low batch size and to reduce to the minimum the padding of the input sentences.

To avoid having the network overwhelmed by learning the padding we have introduced a masked loss in order properly don't take it into consideration.

In the end we manage to train this network for only one epoch (due to the too high training time needed) obtaining very poor results.

So, due to the large memory requirements and to the slow training time caused by the inherent sequential nature of the RNN layers, we decided to pass to more efficient architectures like Transformers.

4.4 Transformer (supervised)

As a first step we implemented and set up the model in a standard way according more or less to the original paper [2]. Then on the basis of the previously described metrics, we tried to improve the model performances by tuning the hyper-parameters. In particular the hyper-parameters that we have tuned are: the learning rate, the latent dimension and the number of heads of the layer corresponding to the multi-head attention mechanism.

In the end, after the execution of the hyper-parameters tuning on the validation set, we came up with the best configuration of them (LR=0.0001, Latent dimension = 2048 and Number of heads = 8). However, we did not notice a big difference between the values obtained using the best hyper-parameters and the other configurations. Instead for what concerns the size of the batches more suitable for our available resources was 64.

Then we performed other several tests: varying the dimension of the pretrained embedding from 200 to 300, performing a trial without using the FastText pre-trained embedding (computing them during the training) and switching the optimizer from Adam to RMSprop.

To increase the performances of the model we added a scheduler for the learning rate. With this addition, we noticed that the model did not remain stuck in an oscillatory behavior after some epochs of training. Last, to avoid overfitting, we introduced an early stopping procedure based on the loss of the test set.

Being text simplification a generative task in which we need to do a remapping of the input using the tokens available within the vocabulary, we used as loss function the sparse categorical cross entropy. This seemed to us the best choice given the fact that we are learning the distribution of simplified terms given the input distribution of complex text.ù

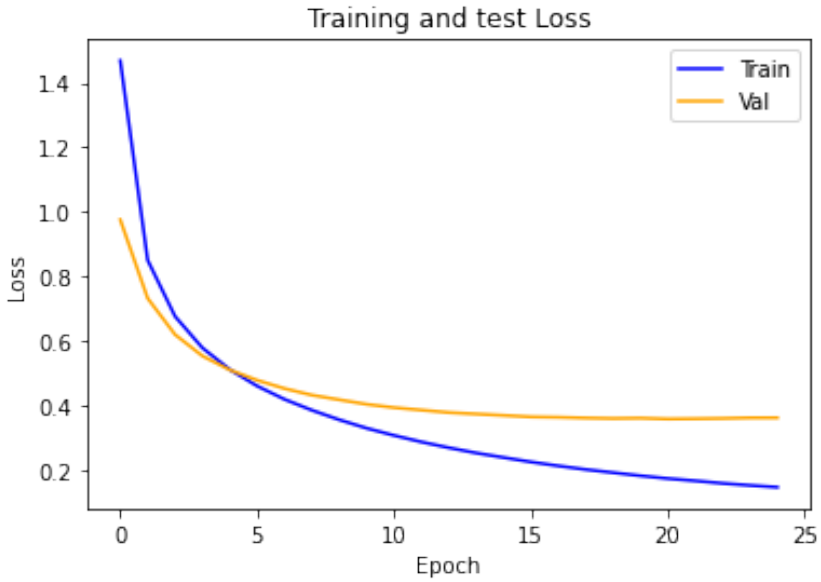
After finding the best configuration of all the previously described hyper-parameters we performed the evaluation on the test set obtaining the following results:

Models	BLEU	SARI
FastText size 200	35.86	19.28
FastText size 300	34.30	19.10
Trained Embeddings	34.23	19.21

BLEU and SARI Metrics for the Supervised Transformer

As we can see we got the best results with the model using the pre-trained word embeddings of size 200. However, theres not a big difference between the models presented in the comparison.

In the following figure instead we can see how the sparse categorical cross entropy loss decreases during the training.



4.5 Transformer (self-supervised)

For the self-supervised approach we have relied once more on the Transformer architecture for its computational efficiency. Yet to try and mitigate the issues presented in the supervised approach we have relied on a self-supervised approach based on a custom loss. In particular, we have tried to adapt a readability metric, the Gulpease Index, to have an automatic measure of the readability of our produced sentence. The metric is based on the number of words and on the number of letters in the sentence, by the formula:

$$89 + (300 - 10 * \text{letters})/\text{words} \tag{2}$$

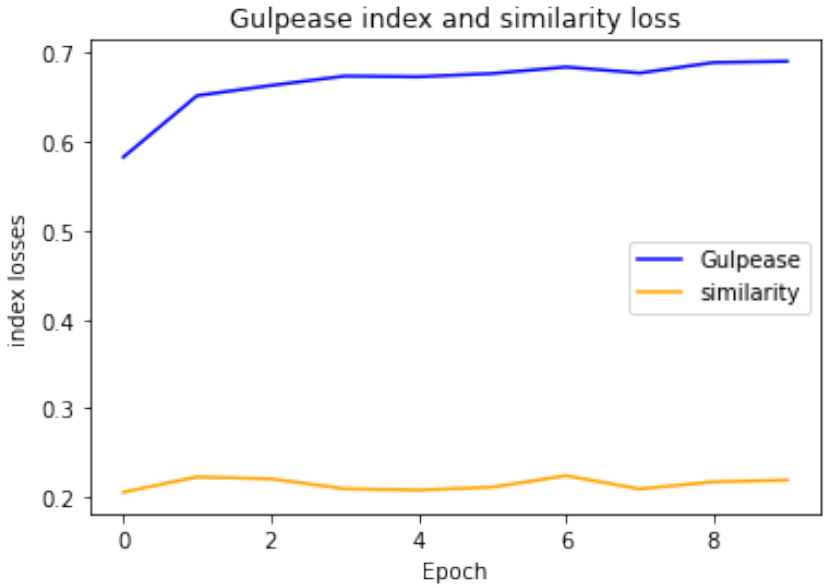
. The typical Gulpease index would not be a differentiable function of our output, as it would require the application of argmaxes, so we have adapted it to work with the probabilistic output produced from

a softmax non-linearity. Therefore, we calculated the length of our sentences by multiplying the length of each word in our vocabulary with the softmax probability distribution obtained from the last layer of our neural network.

A similar approach was employed for the calculation of the total number of words in the sentence by removing from the max sequence length the sum of the probabilities (as they should sum to at most the total number of words) of fake words (i.e. [start],[stop],pad).

Another element that has been taken into consideration is the mathematical behavior of the index, more specifically while it should be bounded between 0 and 100 (with 0 meaning an extremely hard to read document and 100 a very easy to read document). Yet, there is no mathematical certainty of this property in its formulation. As the text produced by our network, especially during training, is in no way guaranteed to conform to standard text. We have, thus, introduced a term based on the Gulpease index in our loss: $-(70-G)^2$ where G is the Gulpease index. In this way we avoid the possibly strange behavior of the index to influence our training.

To validate this approach we have tested if , in a normal supervised approach, these metrics would improve with the reduction of the loss. The results are presented in figure.



GULPEASE index and Similarity loss during supervised learning.

As we can see, the Gulpease index increases for the duration of the training; this is an expected behavior as an increase in the Gulpease index indicates a more simplified output text. The other indicator, the similarity loss, is used to indicate how much of the semantic information is retained after the simplification. This loss is based on the cosine similarity between the average of the encodings of input and output. Starting from a very high value, which is not visible in the plot due to the averaging within epochs, it quickly goes down to 0.2 stabilizing around the value. This behavior is also expected, as we know that in the dataset utilized the mean cosine similarity has a value of around 0.8 (the similarity loss is defined as 1- cosine similarity).

4.5.1 GAN

In order to shorten the computation, for this experiment we set to 20 the (maximum) sentence length.

The implementation of the AE is just the stacking of a FastText embedding, two LSTM networks and a dense layer to output a probability distribution over the words in the dictionary. It is worth noting that in this way the AE has to learn the correct association between the embedding of a word and the output probability distribution that maps the embedding to the corresponding output unit; in other words it cannot learn a proper identity function.

	Layer (type)	Output Shape	Param #
	input_2 (InputLayer)	[(None, 20)]	0
	embedding_1 (Embedding)	(None, 20, 200)	3595400
ENCODER	lstm_2 (LSTM)	(None, 20, 50)	50200
DECODER	lstm_3 (LSTM)	(None, 20, 50)	20200
	dense_1 (Dense)	(None, 20, 17977)	916827

Total params: 4,582,627
Trainable params: 987,227
Non-trainable params: 3,595,400

Architecture of the Autoencoder.

Since a good learning rhythm was found when the encoder LSTM were instructed to return the whole sequence of outputs computed (one for each timestep), the inner code shape is

$$(max\_sentence\_len,\ inner\_dimension) = (20,\ 50) \tag{3}$$

The autoencoder gave decent results in reproducing sentences; below we show some examples, mainly focused on the errors committed by the model:

"poi si farà."	poi si farà .
ho constatato che davide ha una brutta barba."	ho constatato che davide ha una brutta avevano .
ho constatato che davide ha una barba brutta."	ho constatato che davide ha una avevano brutta .
"Giovanni."	l.d .
"barba di Giovanni."	avevano di discorsi .
"vago è il cammino."	propriamente è il cammino .
"vago."	propriamente .

In order to train the GAN, a custom training loop has been used. Despite all the efforts, at some early point of the training the discriminator and generator loss stop decreasing. The actual values of the loss at the stopping point are different, depending on the learning rate used, but the generator output remained not intelligible. This could be an instance of the known *mode collapsing* problem and more attempts should be made to overcome it.

5 Analysis of results

5.1 Baseline (RNN with attention)

With this architecture we didn't manage to obtain results in terms of metrics due to the problems described in the previous section. In fact, being able to perform only one epoch of training, testing the model we obtained a semantic meaningless sequence of words in relation to the input sentence. Here there is an example of the model prediction with respect to an input sentence to be simplified:

```
Input sentence:     "ma questo a cosa servirebbe ?"
Prediction:         ma una complicazione importante . ma
```

5.2 Transformer (supervised)

For what concerns the Transformer architecture trained with a supervised approach, we manage to obtain quite good results considering the not so high content quality of the dataset used and the general complexity of the text simplification task. The results in terms of metrics are not really comparable with respect to the state of the art models; for example there are networks that are able to archive

scores around 40 for the SARI metric and around 75 with regards to the BLEU metric based on the TurkCorpus dataset. Instead, we can apply a more meaningful comparison with a paper [7] written by a group of Italian researchers that applied text simplification to more or less the same dataset that we used, reaching a SARI score around 50. They applied a huge use of several techniques for augmenting the dataset with training data that we have not at our disposal, and this seems to give them a boost in the results.

Analyzing instead more deeply the prediction of our model, we can say that in most cases, the simplified text generated by the network, consists in a remapping of the original sentence appling small changes at the level of nouns and verbs or simply removing words. Here there are some examples:

```
Input sentence:
"la ringrazio infinitamente per la cortese attenzione e porgo distinti saluti . "
Prediction:  la ringrazio per l'attenzione e le porgo distinti saluti.

Input sentence:     ma non farei di ogni erba un fascio .
Prediction:         non si può fare di ogni erba un fascio.
```

As we can see, the prediction of the model doesnt apply a real and effective text simplification. This is probably due to the fact that the quality of the gold standard is not so high, most of the complex sentences of the dataset are indeed too short to be simplified, and the ground truth doesnt reflect a real simplification of the input. Here there are some examples:

Complex sentence	Simplified sentence
ciao a tutti avrei bisogno di un consiglio .	ho bisogno di un suo consiglio .
possibilmente uno che avesse bisogno dell'aiuto .	ho bisogno di un vostro aiuto .
questa sarebbe una cosa positiva.	questa era una nuova cosa .
quale sarebbe allora la soluzione giusta ?	è questa la soluzione giusta ?
che ne sarebbe della natura della realtà ?	ma di che natura sono queste realtà ?

In the end we can say that in this case the real problem is the dataset; in fact having mosts of the complex-simplified sentences that are not aligned following a proper simplification, doesnt allow the network to learn how to accomplish this task in a good manner.

Moreover we can say that without using any kind of other external knowledge the vocabulary of the model is not large enough to apply a good simplification. Anyway in some test sentences, probably coming from the manually annotated SIMPITIKI corpus (high quality), the prediction of the model are not so bad, like in this case:

```
Input sentence: il presente decreto , pubblicato nella gazzetta ufficiale della
repubblica italiana , entra in vigore il giorno seguente alla sua pubblicazione .

Prediction: Questo decreto sarà pubblicato nella gazzetta ufficiale della
repubblica italiana ed entra in vigore il giorno successivo alla sua pubblicazione .
```

Other errors that the model performs are inherent to the replacement of tokens related to numbers, dates or codes that are different from what is present in the starting sentence.

5.3 Transformer (self-supervised)

The self-supervised approach has not produced good results. Predicted sentences tend to have no coherence or to be just full of a single, very short, word repeated over and over. To mitigate these effects we tried to change the relative weight of the 2 parts of our metric, namely to make the similarity loss having more importance with respect to the Gulpease index; the modification produced no effect. We have also tried limiting our cosine loss in a way similar to the Gulpease index, in order to recover the conditions shown in the supervised approach, but neither this approach did produce any results.

Another useful introduction could be based on the passing , in the training phase, of the produced output to the decoder input. This has not been tried due to computational constraints (as it would require about one day per epoch).

To overcome these complexities we believe that more sophisticated methods are necessary for the analysis of the readability of our texts. In particular the training of an autoencoder to obtain the sentence embedding could be particularly beneficial. Another method that could be deployed for the improvement of our model is the use of a vocabulary of common words to evaluate the complexity of the sentences (as done in the READ-IT application [4]).

5.4 GAN

Even if, as anticipated above, the training of the GAN was not successful, we can still do some interesting considerations.

From the error analysis performed on the autoencoder, we can see that some words are consistently mapped wrong irrespective of their position in the sentence. This consideration, jointly with the knowledge of the architecture of the autoencoder, may lead to think that what the autoencoder is actually learning is a PCA-reduced representation of each word, and that the internal code is just a concatenation of these representations. In principle, this would not impede the GANs learning, but this kind of internal state — being close to an identity function and thus failing to capture the meaning of the sentence — would not simplify the function that has to be learned by the GAN, as would be desirable.

For what concerns the GAN, the complexity of finding a Nash equilibrium could depend on the code representation of the complex sentence which is the input of the Generator and is very big with respect to the classical conditioning that can be found in literature for cGAN.

6 Conclusions and Future works

We perfectly know that our work is just a preliminary step for what concerns a truly functional text simplification system; however with the limited resources at our disposal we were not able to properly reach the maximum capabilities of our architectures.

Given this premise, we believe that the main limiting factor for our performances is the quality of the original dataset. This upholds the tenet Garbage-in garbage-out for what concerns the supervised approach. Instead regarding the other two approaches, we have noticed that they are difficult to train due to the inherent peculiarities of the selected task and models.

We strongly believe that the promulgation of better datasets would be of immense benefit for this kind of applications. In particular we think that the corpus would need a higher involvement of more specialized personnel. The use of automatic systems for the creation of datasets, as was for our case, are not very effective as regards the simplification of the text.

Concerning self-supervised approaches we believe that introduction of more sophisticated methods for the evaluation of readability are necessary for obtaining improvements in the execution of this task.

A possible extension that could provide useful insights on the problem is the introduction of a frequency-based approach to evaluate the complexity of words, possibly in conjunction with the method that we tried (solely based on the ratio number of characters/sentence length).

References

1. **Effective Approaches to Attention-based Neural Machine Translation**, Minh-Thang Luong, Hieu Pham, Christopher D. Manning

2. **Attention Is All You Need**, Ashish Vaswani et al.

3. **Adversarial Text Generation Without Reinforcement Learning**, David Donahue, Anna Rumshisky

4. **READ-IT application:** http://www.italianlp.it/demo/read-it/

5. **BLEU: a Method for Automatic Evaluation of Machine Translation** Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu

6. **Optimizing Statistical Machine Translation for Text Simplification**, Wei Xu, Courtney Napoles, Ellie Pavlick et al.

7. **Neural Text Simplification in Low-Resource Conditions. Using Weak Supervision.** Alessio Palmero Aprosio, Sara Tonelli, Mario Turchi et al.