# LaTeX Author Guidelines for CVPR Proceedings

First Author
Institution1
Institution1 address
firstauthor@i1.org

Second Author
Institution2
First line of institution2 address
secondauthor@i2.org

## Abstract

*The ABSTRACT is to be in fully-justified italicized text, at the top of the left-hand column, below the author and affiliation information. Use the word "Abstract" as the title, in 12-point Times, boldface type, centered relative to the column, initially capitalized. The abstract is to be in 10-point, single-spaced type. Leave two blank lines after the Abstract, then begin the main text. Look at previous CVPR abstracts to get a feel for style and length.*

## 1. Introduction

Please follow the steps outlined below when submitting your manuscript to the IEEE Computer Society Press. This style guide now has several important modifications (for example, you are no longer warned against the use of sticky tape to attach your artwork to the paper), so all authors should read this new version.

## 2. Current methods

In recent years, the field of Natural Language Generation (NLG) has expanded its repertoire of solutions for text generation to include a variety of methods. Models such as RNN's, Bi-LSTM [*], Transformers-based models (e.g. GPT [*]), GAN's, among others, have evolved to produce elaborate pieces of text for different specifics tasks. Each method proposes a different approach to the task at hand.

### 2.1. RNN

Recurrent Neural Networks (RNN) are commonly used in sequence-to-sequence (seq2seq) tasks, such as text generation. Simple RNN's have been adapted to overcome its own limitations, e.g. vanishing or exploding gradients, introducing concepts such as Long Short Term Memory networks (LSTM) or even bidirectional LSTM's (Bi-LSTM). Bi-LSTM have been used in many cases for text generation [*] improving its performance in some cases [*]. These models are still used in recent years for text generation in different languages [*]. Some studies [*] use external word embeddings for the training process which can be seen as pre-trained models for text generation.

In this work, we train a simple RNN with an Embedding layer, one LSTM module and a dense output layer. The task of the RNN is to predict the next word for the text generation. For this net, several hyperparameters can be set such as the number of LSTM layers, whether those layers are bidirectional, pre-trained word embeddings, among many others. The configuration that best minimized the cross-entropy loss was an embedding layer with no pre-trained embeddings, three bidirectional LSTM layers and one dense output layer. With this configuration, 1000 sentences were generated for comparison with alternative methods.

### 2.2. Transformers

With the introduction of [*], new models for text generations task have arisen. [*] introduces an Encoder-Decoder based architecture for machine translation tasks. This architecture was ingeniously modified by [*] and [*] to yield models such as BERT and GPT. Both Generative Pre-trained Transformer (GPT) and Bidirectional Encoder Representations from Transformers (BERT) are based on the idea of transformers and can be finetuned and adapted for a variety of tasks [***]. These models have slowly replaced the RNN approaches. As [*ModernMethodsOfTextGeneration] states: "Transformers disrupted sequence-based deep learning significantly. The two variants of transformer models that we showed, BERT and GPT-2, outperform previous approaches with RNNs".

GPT is widely known as a powerful tool for text generation. Conversely, the BERT model alone has not been accepted as a strong generative model. Some authors [**?**] have proposed workarounds to BERT limitations for text generation.

## 3. BertTextGenerator

BERT is a language representation model designed to achieve state-of-art results in several natural language processing tasks such as question answering and language in-

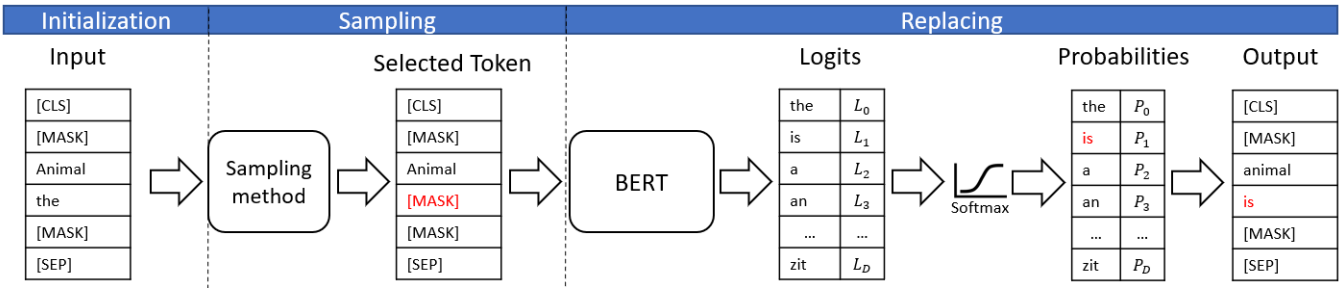


Figure 1. Process to use BERT as a generator.

ference. BERT is trained on masked language modeling objective, in which it predicts a word given its left and right context. Given this behavior, it is possible to deduct that BERT is not the most indicated model for text generation tasks. However, [**?**] have demonstrated why it is possible to use BERT as a traditional language model by means of showing that BERT is a combination of a 'Markov random field language model (MRF-LM), with pseudo Log-Likelihood training'.

This conclusion allows using BERT as a generative model of sentences to either score a sentence or sample a sentence. To generate text with BERT the process is composed by 3 main steps described as follows and illustrated in the Figure 1:

- Initialization: It is necessary to initialize the sequence with a random initial state, this can be either an all-mask sequence, i.e.([MASK], ...,[MASK]), an all-random sequence, or a combination of these, i.e. (50% [MASK] tokens and 50% random tokens).
- Sampling: At each iteration, it is necessary to choose which token of the sequence will be masked, in [**?**] they propose three different methods to carry out the sampling, these methods are described as follows:
  - Sequential: The sequence is masked from left to right one token at a time at each iteration.
  - Parallel: The sequence is completely masked at each iteration.
  - Parallel sequential: The sequence is masked choosing at random one token at each iteration.
- Replacement: Finally, the sequence with the masked token is passed to BERT to produce a tensor of logits $L$ of the length of its vocabulary $v(v \approx 30K)$, the logits are interpreted as a probability distribution of how related are each one of the tokens in the BERT vocabulary with the current sequence, finally the logits are normalized by using a softmax function to retrieve the probability of each token to be inserted on the sequence and based on this probability the new token is selected.

Several experiments have been made to demonstrate the potential of BERT as a text generation algorithm. With the obtained results was possible to conclude that even when the BERT generations are of the worst quality, compared with models like GPT, the text generated by BERT is more diverse and this is a meaningful result [**?**].

### 3.1. Attention mechanism

The attention mechanism is composed by a feed forward neural network that is trained to identify the more relevant components in the context in order to predict a new token.

For example, take the sentence "the animal didn't cross the street because it was too tired", as a human is easy to know that in the sentence the pronoun "it" refers to the subject "the animal" and the verb "was" is the correct conjugation of the verb "To be" for the third person, but for an algorithm is not easy to know these relationships and that is the task of the attention mechanism.

The Figure **??** shows how the attention mechanism create a relationship between the token "it" and several components of the sentence. For example, the first attention head

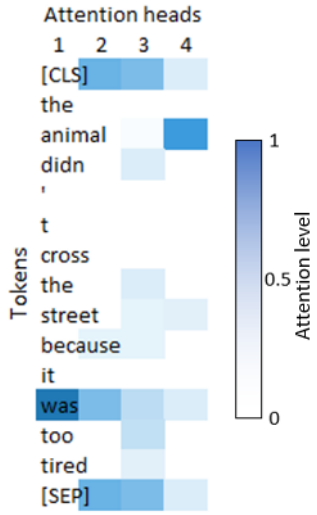


Figure 2. Summarised representation of the functioning of the attention mechanism.

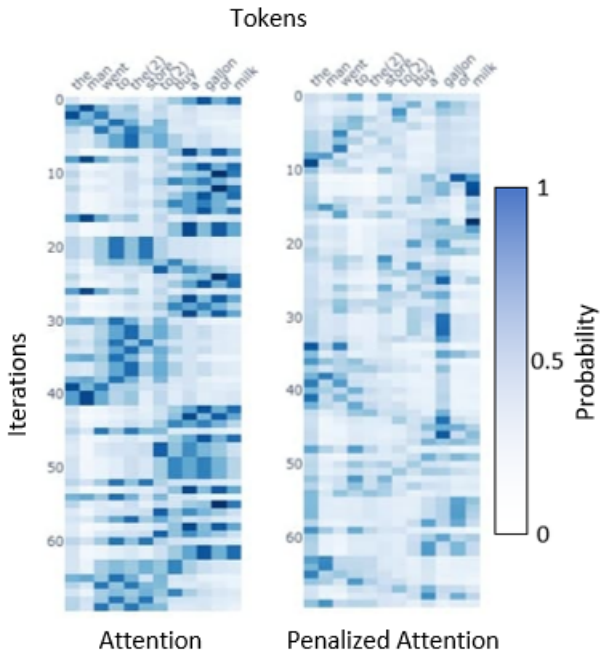


Figure 3. Comparison of the behavior of the proposed sampling methods.

is totally focused on the verb "was", the second and third attention heads are focused on the special tokens "[CLS]" and "[SEP]" that represent the beginning and the end of the sentence respectively, finally the fourth attention head is focused on the word "animal". In this way BERT can understand which parts of the context are more relevant for each one of the tokens and when a token is masked to be replaced on which parts of the context must focused to make the prediction of the new token.

#### 3.1.1 Using the attention mechanism to improve the sampling method.

When a token is predicted, it is possible to know which tokens of the context were more relevant to retrieve the prediction based on the attention weights. This information is a very meaningful insight to improve the way in which the sampling method is made.

We propose a new sampling method based on the attention weights returned by BERT when a prediction is done.

**Attention based sampling** First, we design a sampling method totally based on the attention weights, when a token was replaced in the past iteration we retrieve and summarize the attention information corresponded to the replaced token by means of averages inside and between the encoders to get a unique measure of the attention, then we normalize the result to get a probabilistic interpretation and make the new sample based on this probability, in this way we wanted to sample the more related tokens to the one that was replaced to check if this tokens preserve his meaning in the sequence.

However, we found that the attention mechanism creates very strong correlations between few tokens of the sequence, then the sample got stuck always choosing the same tokens among all the iterations. To solve this behavior, we add a penalization to those tokens that were chosen several times, in the next section we describe the new method.

**Penalized attention sampling** This method is very similar to the previous one, however, we include a penalization factor to those tokens that were chosen several times, in that way the sampling method makes a better exploration of the sequence based on the attention information and as a result we get better generations.

below we describe the procedure to develop the sampling method in the iteration $t$:

- Step 1: Retrieve the attention masks of the iteration $t-1$ $(a)$, then get the average value of the attention masks $A_{p,e}$ for each position $p$ inside each one of the encoders $e$.
- Step 2: Get the average value of the attention mask for each position over all the encoders $totalA_p$.
- Step 3: For each position $p$, there is a counter $c$ that register how many times the position $p$ has been selected to be replaced.
- Step 4: The average value of the attention masks $totalA_p$ is divided by the counter $c$ to create a penalization.
- Step 5: Normalize the result of the step 4 to get a probability distribution $P$.
- Step 6: Sample randomly with distribution $P$ the new token to be replaced.

In the step 1 we summarize the information of the attention inside each one of the encoders, then in the step 2 we combine the information of the different encoders to obtain a single measure of the attention, the step 4 is useful to help the method to avoid that the sampling method get stuck sampling always the same positions given its strong relation with the current context, finally the steps 5 and 6 creates a probability distribution based on the penalized attention and sample the new token to be replaced.

Finally, in the Figure **??** it is possible to see how the probability of choose a token evolves during the iterations for each one of the tokens, it is possible to note how the attention method get stuck during several iterations in the same areas of the sequence, for example, during the iterations 10 to 16, the probability was focus on the last 4 tokens

of the sequence. While in the penalized attention is possible to see a more balance probability distribution inside each iteration and the sampling method is not focus in only few parts of the sequence.

## 3.2. Finetuning

BERT is a powerful model pre-trained on a broad set of text composed by the whole Wikipedia and Brown corpora []. One of its main feature is its adaptability to a plethora of different tasks, by means of a fine-tuning on specific domain set. Typical fine-tuning techniques depends on the final task that the user requires BERT to perform. For example, considering classification tasks such as sentiment prediction and text classification, the fine-tuning will be focused on the [CLS] token, a special token used in the pre-training phase for the task of "next sentence prediction" and specifically designed for classification applications.

Text generation is not a usual task for BERT. In this case, the fine-tuning aims to make the model understand the structure of the text, in order to mimic it at inference time. We have implemented a fine-tuning procedure that rely on *Cloze* task present also in the original training []. This consists on replacing a percentage (15% by default) of the tokens for each sentence with a [MASK] token and let BERT predict the blanks. The logits outputted by the model are then used to to compute the cross-entropy loss with respect to the original token that were replaced. This method allows the final user to start from a pretrained model and fine-tune it on a specific text corpus in order then to make it generate text that similar to the original one. This approach also works in the case that a pretrained model is not available in th language selected by the user. In this case, the user can simply start from a cross-language model [...]

### 3.2.1 Generation of text with a specific structure

The default vocabulary of the tokenizer contains around 1000 unused tokens, whose weights are randomly initialized. Typically, these tokens are replaced with domain specific words, so that during the fine-tuning phase the model will be able to learn their representation.

We have extended this idea in order to comprehend also tokens that defines the structure of the text like '\n' '\t'. Note that, simply adding these tokens to the vocabulary or replacing with unused tokens would have no effect. These tokens in fact, are typically replaced even before the tokenization, during the normalization step of the Tokenizer pipeline []. However, depending on the context, these tokens can be important. This can be especially true in the task of text generation.

We have implemented a Formatter class that helps the user to format the text in order to preserve the important tokens. The Formatter replaces a user specified set of tokens that need to be preserved, with some unused-tokens. The replacement happens before the tokenization, in this manner, during the fine-tuning the model will be able to learn the position of these tokens. The unused tokens are then replaced back with the original ones after the generation.

Using this method we were able to make a model learn the famous tercet structure of Dante's Divine Comedy **??**.

### 3.2.2 Sentiment generation

An important aspect of BERT tokenizer are special tokens like [CLS], [MASK] and [SEP]. As the name suggests, these are tokens with specific functions. The tokenizer gave the possibility to define new special tokens. We have taken advantage of this, with the object of define a new method to generate text with a specific sentiment or label.

Considere a domain set $\mathcal{S}$ of sentences each one with an attached label $y \in \mathcal{Y}$, i.e. a text whose label-set are the two possible sentiments $\mathcal{Y} = \{pos, neg\}$. The method consists in three main steps:

- For each sentiment in the label set $y \in \mathcal{Y}$, we define $n$ (3 by default) special tokens $[y_i] \quad for \quad i = 1, \dots n$. In the case of $y = pos$ for example the set of special tokens will be $\{'$[pos-1]$',' $[pos-2]$,' $[pos-3]$'\}$
- Before the fine-tuning each sentence $s \in \mathcal{S}$ is tokenized, and the special tokens are appended at the beginning of the list of tokens.
- The fine-tuning is performed. During this step the model will build a relation among the special tokens of a sentiment and the specific words related to it.
- To generate the text with a given sentiment, the special tokens related to that sentiment are used as seed to create a context on top of which BERT can start generate.

$$\Big[ \underbrace{\text{[CLS] [pos-1] [pos-2] [pos-3]}}_{seed} \text{[MASK] } \dots \Big]$$

This method, even in its simplicity, showed its efficacy in generating positive and negative italian tweets about football. In addition, this method will allow the possibility to consider even more than two labels.

# 4. Experiments and evaluation

Grid search explanation Best model 1000 sents Corpus Bleu, perplexity and rouge Self-Bleu and n-Grams

## 4.1. Italian generation: finetuning testing

In order to test the goodness of the fine-tuning implementation we have performed three experiments. All the experiments have been performed on Italian sets of text and for all the experiments we started from a pretrained italian BERT model, freely downloadable from huggingface [].

| Model | Bleu | | Perplexity | Self-Bleu |
|---|---|---|---|---|
| | WT103 | TBC | | |
| Original [**?**] | 7.06 | 7.8 | 279 | 10.06 |
| Original* | 6.86 | 7.14 | 270.6 | 8.43 |
| Parallel-1 | 6.62 | 7.21 | 287.6 | 6.83 |
| Attention | 8.15 | 6.59 | 386.6 | 11.88 |
| Parallel-0.15 | 8.79 | 7.11 | 1241 | 12.02 |
| Sequential | 5.73 | 7.27 | 374.2 | 7.05 |
| RNN (WT) | 8.54 | 2.9 | - | 15.23 |
| RNN (TBC) | 5.73 | 7.27 | - | 7.45 |

**Motivational quotes** The first experiment aimed at showing the functionality of the fine-tuning method. To do that we have gathered a small set of motivational quotes. The set was constructed scraping from different sources, and the quotes were preprocessed in order to make them all structured in the same way.

> A motivational quote - (anonymous)

**Dante's divine comedy** The second experiment was focused on the structure of the text, for this part we have used Dante's Divine Comedy.

> Nel mezzo del cammin di nostra vita
> mi ritrovai per una selva oscura,
> ché la diritta via era smarrita.

The model was fine-tuned on tercets (groups of three lines) instead of single lines. Moreover, we have replaced '\n' with 'unused1' tokens in order to make BERT learn the structure of the tercets.

> ['Nel', 'mezzo', 'del', 'cammin', 'di', 'nostra',
> 'vita', **'unused1'**, 'mi', 'ritrovai', 'per', ...

**Sentiment generation** The aim of the third experiment was the sentiment generation. In this case we have used a dataset [] composed by num italian tweets. Each tweet has associated a sentiment score for each one of 4 possible labels $\mathcal{Y} = \{\text{POSITIVE}, \text{NEGATIVE}, \text{MIXED}, \text{NEUTRAL}\}$ and a sentiment label corresponding to the sentiment with the highset score. The dataset was highly imbalanced, with respect to the last two classes, so we have considered only the first two. The dataset was preprocessed as explained in 3.2.2; in addition a set of tokens was added in order to deal with '@tags' and 'hashtags'.

## 5. Conclusions