Ctrl + Alt + Captions: Efficient conditional text generation on small corpus

Luca Rinelli Dario Piazza Andrei Schiopu
Polytechnic University of Turin
Turin, Italy
{s269097, s276010, s269306}@studenti.polito.it

https://github.com/lucarinelli/conditional_text_generation

Abstract

In the last years the task of text generation has seen several models achieving great successes, however complications still appear once those models have to be integrated in a product: applications often would benefit from an higher degree of control on the text generated by the model (e.g. chatbots, query answering systems). Moreover these models are constantly growing in number of parameters and thus time and memory complexity, both for training and execution, is increasing resources requirements for natural language processing.

In this paper we describe how we coped with the two problems we presented above, applying the emerging techniques of conditional text generation and knowledge distillation to obtain a model which is both controllable and more efficient.

1. Introduction

Natural language generation (NLG) is one of the most important and challenging tasks in Natural Language Processing (NLP)[14]. The aim of NLG is to produce understandable text in human language starting from an input that may be textual data, numerical data, image data, structured knowledge bases, knowledge graphs or a combination of them.

Common applications of text generations are data-to-text systems that can automatically write reports while performing also data analysis, summarization, chatbots or dialog systems, question answering systems and automated journalism.

1.1. Control the output: Conditional Text Generation

For the applications previously listed, the input text alone is often not enough to produce the desired output and more control on the generated text is needed. Different approaches have been tried to give additional context to outputs, such as a topic, a style, or some keywords. In this paper we will explore a strategy, based on *control codes* inspired by the approach adopted by CTRL[6].

The main idea of text generation is that, given sequences in the form $x = (x_1, ..., x_n)$ where each x_i comes from a fixed set of symbols in a given *dictionary*, the goal of language modeling is to learn $p(x)$. Since x is a sequence, we can factorize this distribution using the chain rule of probability [6]:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{<i}) \tag{1}$$

On the other hand, the concept of Conditional Text Generation focuses on influencing the generation of the sequence by imposing an additional dependency on one or more *control codes* c_i, rather than letting the model generate a sequence based only on the previous words x_i.These special inputs have the goal of adding context, topic or emotion to the generated sentence.

As a result, the probability distribution is similar to the previous one, but it also takes into consideration the control code(s), so it can be expressed as follows:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{<i}, c) \tag{2}$$

where c are the control codes given as input to the model.

Since there are several way to use control codes in the input, we explored two different approaches in our experiments, the *special token* and *separator* ones.

1.2. Models for NLG

Sequence-to-sequence (Seq2Seq) models are now the most commonly used for NLG. They have been around since 2014[10] and there are lot of popular architectures implementing this approach using models based on Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) and lately Transformer.

In particular, the Transformer architecture, introduced in [11], has rapidly become the base for most of the State-of-the-Art for Natural Language Processing[12] models, mitigating problems posed by the previously most common architectures RNN and LSTM.

In RNN, in fact, tokens are processed sequentially, which implies that the model has to maintain a state vector that contains a representation of the data seen after every token. For this reason, to process the n-th token, the model combines the state representing the sentence up to token $n-1$ with the information of the new token, in order to create a new state. Theoretically, the information from one token can propagate arbitrarily far down the sequence, if at every point the state continues to encode contextual information about the token. In practice, however, this mechanism presents one problem: the vanishing (or exploding) gradient. LSTM try to mitigate this problem by adding gates to learn when to keep or let go current or past information. Additionally, from a performance point of view, those type of models are not able to exploit the full power of nowadays GPU. They require sequential processing of the sentences, that are thus processed word by word.

The Transformer architecture introduces the so-called *attention mechanism*, which lets the model access the state at any preceding point along the sequence, according to a learned measure of relevancy, providing relevant information about far-away tokens. This is also better for performance during training since sentences are processed as a whole, and thus the work can be parallelized more efficiently.

Transformer use an encoder-decoder schema, where the encoder maps the input sequence to a fixed-sized vector and the decoder maps the vector to the target sequence. Some models use only a stack of decoders (GPT-2 for example), while others rely on a stack of encoders only (like BERT). Some examples of conditional text generation models based on Transformers are CTRL[6] or PPLM[2]

1.3. Efficiency and sustainability: Knowledge distillation

Models like the ones mentioned in the previous paragraph have important memory and computational power requirements, and recently we have seen model size and complexity growing exponentially, see Figure 1.

This sparkled interest[1][2] for more sustainable NLP, concerning mainly efficiency and justifiability. Large models have higher knowledge capacity than smaller ones, but often this capacity might not be fully exploited. This leads to be just a computationally expense to evaluate the full model. This is even more true for small corpus datasets and use cases, when smaller and simpler architecture have proven to be more accurate and efficient[3].

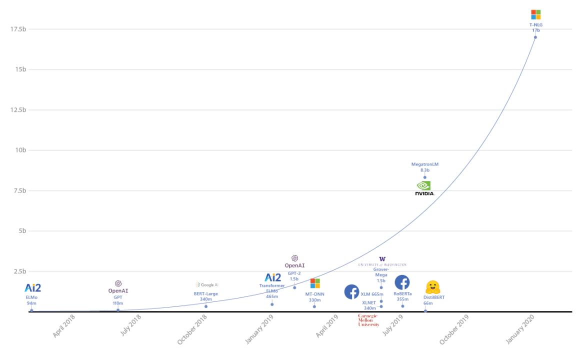

Figure 1. Number of parameters for a selection of models in time, from https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

These considerations prompted us to explore ways to obtain smaller and more efficient models leading us to *knowledge distillation*, that is the process of transferring knowledge from a large model to a smaller one without loss of validity.

It consists of training a smaller network, called the distilled model, on a dataset called transfer set that is obtained using the output of the inference of a larger model. The cross entropy loss function is used between the output of the smaller distilled model (the student) and the output produced by the larger model (the teacher) on the same records.

In this way, the student learns also on the pseudo-likelihoods output of the teacher instead of just on an hard labels. This helps it to learn a more general knowledge representation from the teacher model. A small model which has been distilled should perform much better on test data compared to the same which has been trained in a normal way or just fine tuned on the same training set[5].

2. Data

The dataset we used for our experiments is COCO Captions 2017[1]. It is thought for large-scale object detection, segmentation, and captioning and it is composed by short captions describing images binded to categories and supercategories by labels. We processed captions and labels for each image to obtain a dataset item resembling the following:

`Caption`: ”A bedroom with a bookshelf full of books.”

`Categories`: ”bed”, ”book”, ”chair”

`Supercategories`: ”furniture”, ”indoor”

`References`: ”Bedroom scene with a bookcase, blue comforter and window.”, ”A bedroom

[1]EACL 2021 - Green and Sustainable NLP track https://2021.eacl.org/news/green-and-sustainable-nlp

[2]SustaiNLP 2021 https://sites.google.com/view/sustainlp2021

with a bookshelf full of books.", ... "A bed and a mirror in a small room."

The "references" field is composed of all the captions used to describe the given image. We use these references to compute BLEU metrics described in section 4.

We analysed the training dataset to find out that each image has

- an average of 2.3 supercategories (min 1 max 9), having the average as the 63rd percentile.
- an average of 2.9 categories (min 1, max 18), having the average as the 51st percentile.
- an average of 4.7 supercategories + categories (min 1, max 26), having the average as the 57th percentile.

The goal of the analysis was to understand which strategy to follow for handling the control codes: categories, supercategories or both. We did several experiments for each of the 3 aforementioned strategies, ultimately settling on only supercategories as control codes because it led to a better ability in generalizing the text generation.

We suppose that this is due to the fact that the captions were, on average, 52.47 characters long, and supercategories were more generic and comprehensive (the most of the dataset has less control codes than the average), so the usage of less control codes which were, additionally, more generic, influenced more deeply the training.

The resulting control codes, corresponding to the 12 supercategories are:

- accessory
- animal
- appliance
- vehicle
- electronic
- food
- furniture
- indoor
- kitchen
- outdoor
- person
- sports

Starting from the 473.466 in the training set and 20.014 captions in the validation set, we discarded, respectively, 4.084 and 192 captions because they had no control codes. The resulting dataset has been further filtered by removing 318 and 14 captions, to have exactly 5 captions describing each image, obtaining, at the end, a training set of 469.064 and a validation set of 19.808.

To let the model learn the meaning of each combinations of control codes, we augment the dataset repeating a caption as many times as the items in the power set of the control codes associated to that caption. To reduce the size of the augmented dataset we limit the length of the tuples of control codes coming from the power set to three.

For example, for the caption shown previously, and considering an experiment which used only supercategories as control codes, the caption was inserted in the dataset four times: the first time with no control codes, the second with control codes `[furniture]`, third one with the control codes `[indoor]`, and lastly with the control codes `[furniture, indoor]`. Since GPT2-based models has been trained on long texts[3], we chose to provide to the model, among the captions with control codes, some unprovided in order to allow the model to produce short sentences even without control codes.

To avoid the need to shuffle the items from the power set tuples we feed the control codes to the model always in alphabetical order.

Performing this augmentation we obtained a training set of 2.567.428 captions and a validation set of 109.060 captions. To reduce training time we take only the first 100.000 and 5000 captions from those sets, after having shuffled the dataset with a defined random seed.

Analyzing the binding between the captions and the aforementioned categories, we noticed the presence of some out liers: there were captions which seemed to be logically disconnected from their categories.

This can influence negatively the perceived performance of the model and is caused by the fact that the COCO dataset is mainly thought as a large-scale object detection dataset, and some captions of the photo may not be directly linked with all the elements which caused the photo to be categorized as it was.

2.1. Tokenization

Even though the model is responsible for taking an input and producing an output, raw words are not suitable to be fed to the model. For this reason, we need some kind of interface to pre-process the inputs and transforming them into a structure that the model can easily manage. The role of the `tokenizer` is, indeed, to take as input a sentence and split it into words (or sub-parts of words) that are called *tokens*.

After this splitting phase, the tokenizer transforms the obtained tokens into integers, that we will refer to as *ids*. In this way, the model will receive as inputs a list of *ids*, and each id will be associated to a word (or part of it) through a data structure similar to a lookup table. For our experiments, we used a Byte-level Byte-Pair-Encoding (Byte-level BPE) tokenizer. Byte-Pair-Encoding means that, at the end of the traning process, the tokenizer will have its own *vocabulary*, which will include all the base symbols (i.e. characters) and their *merges* (i.e. how they are combined in

[3]https://huggingface.co/transformers/model_doc/gpt2.html

the training data).[4]

Example: when processing the caption

"A person is eating."

the model will receive as input (ignoring the control codes which will be explained later on):

[32, 1048, 318, 6600, 13]

where each of those integers corresponds to a word in the embedding space.

An embedding is a low-dimensional space into which you can translate high-dimensional vectors. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space and this make it easier to process large inputs like sparse vectors representing words.

We used a pre-trained tokenizer for pre-trained models and trained one from scratch for non pre-trained ones.

Special tokens are defined to delimit the starting and the ending of a sentence. We used the same for both goals (as the default implementation from hugging-face does): $\langle|$*endoftext*$|\rangle$. Also two separate tokens could be used: $\langle|$*startoftext*$|\rangle$ and $\langle|$*endoftext*$|\rangle$. We used the first strategy for most of the experiments and then switched to the second for others (later indicated as SEP and ST-0 models) to better support empty prompts for generation.

3. Approach

Several variables and procedures characterize an NLP experiment. In our experiments we explored several configurations, concerning mainly the following parameters:

- **Control codes type**: for one class of experiments, we treated the control codes as `special tokens`, meaning that the tokenizer (and also the model) interpreted these words in a different way with respect to *normal* words: each of those words will be associated to a unique id, and, thus, to a unique token. To easily identify these new tokens, we transformed it as follows:

  sports $\rightarrow$ $\langle$*CTRL:sports*$\rangle$

  An example of an input submitted to the model during the training phase:

  $\langle$*CTRL:furniture*$\rangle\langle$*CTRL:person*$\rangle\langle|$*startoftext*$|\rangle$*A few women that are standing around some food*$\langle|$*endoftext*$|\rangle$

  For a second class of experiments, we treated the control codes as common words, and the only preprocessing applied to each caption consisted in prepending the control codes to the caption itself, separated by a comma and space, before the $\langle|$startoftext$|\rangle$ token which acted as a `separator`. An example of an input submitted to the model during the training phase:

  furniture, person, $\langle|$*startoftext*$|\rangle$*A few women that are standing around some food*$\langle|$*endoftext*$|\rangle$

- **Model size**: we conducted experiments with architectures based on *GPT2 small*(12 layers, 117M parameters), *DistilGPT2*(6 layers, 82M parameters) and *Tiny GPT2*(2 layers, less than 1M parameters).

- **Fine-tune vs train from scratch**: we tried fine-tuning pre-trained models from HuggingFace Models and compared with the same architectures trained from scratch on our captions dataset.

- **Layers freezing**: in pre-trained NLG models, the layers closer to the input are usually the ones involved in more general basic language rules. On the other hand, while layers closer to the output are usually responsible for more high level decisions and are the most important in the fine tuning process, because they contain the information that you want to focus on, while keeping intact the basic language rules already learned by the other layers. With this in mind we compared models with no layer freezing versus ones with different numbers of frozen layers.

- **Distillation**: we compare the fine-tuning of relatively small pre-trained models, *distil-GPT2* and *tiny-GPT2*, with their distillation using as teacher a larger model, in this case our best fine-tuned *GPT2* model.

We started by trying to reproduce the state of the art[5][4][8][13][7] for text generation on COCO captions since we could not find any references for conditional text generation on this dataset. The dataset pre-processing applied for the state of the art models reduced the number of words removing the least frequent ones replacing them with an "unknown" token or removing the corresponding caption altogether. We used instead a less aggressive pre-processing and this had a small impact on evaluation metrics. We experimented with training from scratch or fine tuning a pre-trained model.

We then proceeded with conditional text generation experiments, introducing control codes. Our first implementation of control codes was through special tokens added to the tokenizer. We wanted to be able to use multiple control

[4]A more detailed explanation is available at https://huggingface.co/transformers/tokenizer_summary.html#byte-pair-encoding-bpe

[5]https://paperswithcode.com/sota/text-generation-on-coco-captions

codes in one prompt, so the model has been trained with combinations of control codes as described in section 2.

We performed several experiments fine tuning GPT2 and DistilGPT2 or training them from scratch to decide what to use as control codes: supercategories, categories or supercategories and categories.

We kept comparing fine-tuning against training from scratch, expecting the fine-tuned models to perform better than the ones trained from scratch.

As already mentioned we ended up choosing to use only supercategories as control codes, due to more satisfactory results in text generation tests.

We then tried fine-tuning models while freezing some of the layers closer to the input, hoping to preserve more of the model knowledge coming from its pre-training, to hopefully obtain better results on the validation set.

We tried hyper-parameter search for learning rate and batch size hoping to find the best values to use in experiments to improve our results.

At this point we started exploring knowledge distillation and we did some experiments to compare the performance of models after fine-tuning or distillation.

4. Metrics

To assess the quality of the generated text we used the following metrics:

- **BLEU:** Short for *Bilingual Evaluation Understudy*, it is an algorithm to evaluate the quality of the generated text, comparing it with one or more references. Initially conceived for translation purposes, it uses *n-grams* to evaluate the accuracy of the output of the model with respect to the reference text (all the captions of the image from which the given sentence was derived), meaning that it does not consider only the single words, but also pairs, threes and so on. However, "we obtain the best correlation with monolingual human judgments using a maximum n-gram order of 4"[9]. In our case we used as references to compute BLEU for one caption all the 5 captions available for the same image, including the caption for which the metric is being computed.

- **SELF-BLEU:** Introduced in [15], Self-BLEU is a metric to evaluate the variety of the generated text. It measures BLEU score for each generated sentence by considering other generated sentences as reference and then averages these BLEU scores. A higher Self-BLEU score implies less diversity in the generated text.

- **POS-BLEU:** It takes into consideration also the structure and the syntax in the sentence building. "As its name suggests its main idea is the calculation of the classic BLEU score on the POS tags of the words instead of the words themselves." [6] POS (Part of speech) tags are classes in which words can be grouped (such as noun, verb, adjective, preposition, adverb, etc..) . POS tags are important for finding the word's role in a sentence. References for this metric are handled in the same way as for BLEU.

All the metrics are computed ignoring all special tokens, including control codes.

Note that the validation set used to compute these metrics contains also captions without control-codes, due to the power set strategy adopted in augmenting the dataset described in section 2. This makes the task of conditional text generation harder lowering the value of the metrics.

5. Experiments

5.1. Models

We chosed to use GPT2 like models and conducted experiments with architectures based on *GPT2 small*(12 layers, 117M parameters), *DistilGPT2*(6 layers, 82M parameters) and *Tiny GPT2*(2 layers, less than 1M parameters). We downloaded them already pre-trained from HuggingFace models libraries and eventually ignored the weights if we were training from scratch. HuggingFace used a distillation approach with the supervision of GPT2 (the smallest version of GPT2) on OpenWebTextCorpus[7] to pre-train DistilGPT2 and Tiny GPT2. We have performed several runs (just the recorded ones are more than 120) for many different configurations. For brevity we report here only the results for a selection of models, all of these trained on 10 epochs:

- **ST**: GPT-2 pre-trained and fine-tuned on all 12 layers using special tokens as control codes.

- **ST-F10**: GPT-2 pre-trained and fine-tuned on the last 2 layers (so having 10 layers frozen) using special tokens as control codes.

- **ST-0**: GPT-2 trained from scratch using special tokens as control codes.

- **SEP**: GPT-2 pre-trained and fine-tuned on all 12 layers using serparators as control codes.

- **SEP-F10**: GPT-2 pre-trained and fine-tuned on the last 2 layers (so having 10 layers frozen) using separators as control codes.

- **D-ST**: Distil-GPT-2 pre-trained and fine-tuned on all 6 layers using special tokens as control codes.

[6]Automatic Error Detection and Correction in Neural Machine Translation- Anthi Papadopoulou 2019

[7]https://skylion007.github.io/OpenWebTextCorpus/

- **T-ST**: Tiny-GPT-2 pre-trained and fine-tuned on all 2 layers using special tokens as control codes.

5.2. Implementation

Our IPython notebooks and Python scripts make large use of pytorch and HuggingFace libraries such ad transformers, datasets and tokenizers.

After automatic and manual hyper parameters search we settled on a learning rate of 0.00001 and a batch size of 64.

5.3. Environment

For our experiments, we set up two different environments:

- Google Colab
- Micorsoft Azure

used to test in a faster way our code. Regardless of the GPU allocated, we found in Colab a very suitable environment to verify whether our choices could be valid or not, but since it is thought to be an interactive environment, we moved to Azure to perform longer runs, as it avoids the resources to be preempted.

In order to store the metrics of our models during the training and validation phase, we relied on *WanDB* which provided us experiment tracking, dataset versioning, and model management.

For code versioning, instead, we relied on GitHub where you can find all our work[8].

6. Results

In Table 6 we report the BLEU, SELF-BLEU and POS-BLEU results obtained with the different models.

By observing these statistics, we may notice that the we achieved better performances when using the *special tokens* tokenization method, described in Section 2.1, especially for the BLEU and POS-BLUE metrics.

This means that the *special tokens* models not only are able to formulate sentences that are more similar to the ones provided as input for the evaluation (BLEU), but also the syntax of the generated text looks more coherent (POS-BLEU). On the other hand, the SELF-BLEU metrics are comparable for all the considered models, indicating that the output sentences for all models are relatively similar among themselves.

One interesting point to focus on are the metrics related to the *Distil-GPT2 with special tokens* model (D-ST). In fact, although the model in significantly smaller than GPT2 (6 layers compared to 12), the performance achieved are not only better than the ones obtained using the *SEP* models, but also close to the ones of the ”bigger” GPT2 with special tokens, in fact the performance degradation with respect to GPT2 is less than 5%.

The only outlier in our results is the T-ST, which evidently is too small for this kind of task. We tried to use distillation in place of fine tuning for T-ST but results were not impressive either in this case.

These results suggest us that, for the considered task of generation of short sentences, employing larger models may lead to considerable overhead in terms of computational load and training time, leaving room for smaller, but equally valid, models.

Besides comparing the metrics of the models, we deem interesting to observe the behaviour of the different models when generating text.

In tables 1, 2, 3, 4 and 5 we report the outputs generated by the models when considering different scenarios.

- By looking at Table 1 we can notice that the generated text is quite different among the various models. Moreover, while the *GPT2*, *Distil-GPT2* and *ST-0* models are perfectly capable of producing a meaningful sentence, the *Tiny-GPT2* output cannot be considered such, despite being free of grammatical errors. This behaviour can also be observed in the other generation results, that we discuss below.

- In Table 2 we can observe the behaviour of the models when providing more context to the input, by using the control codes. One interesting thing to notice is that the output is coherent with the specified context (*sports* and *outdoor*), but the *special tokens*-based models seems to achieve better results (*SEP-10* seems to be missing the *sports* concept).

  As specified in the comment of the table 1, the sentence generated by *Tiny-GPT2* is noticeably lacking of a correct structure, nevertheless it ”understands” the concept of *outdoor* by generating including *streets* among the generated words.

- By examining Table 3 it is possible to analyze the different outputs generated when changing the control codes provided as input to the models.

  Here, the special token *kitchen* is used to condition the generation. As we can notice, the sentences produced are very different from the ones reported in table 2. In fact, it is interesting to see that all the models (apart from *T-ST*) behave to adapt to the context provided as input, and they all produce meaningful sentences, in line with the *kitchen* keyword.

- In Table 4 we reported the sentences produced when *only control codes* are provided to condition the text, while allowing the models to start the sentences free of constraints. It is possible to notice that they seem

[8] https://github.com/lucarinelli/conditional_text_generation

to suffer this particular lack of context. The only two which appear to be capable of generating a meaningful phrase are the *SEP* and the *ST-0*, being the only two trained with the strategy of two different special tokens for the delimitation of text start and text end.

- In Table 5 we tried to submit to the models the beginning of a sentence used in the training procedure, as well as its control codes. We can see how the *ST* and the *D-ST* created a sentence conceptually equivalent to the one used for training. The *SEP-F10* kept the concept of ”standing” while the others are moved quite away from the reference phrase.

Model	Generated sentence
ST	A person in a blue jacket on skis.
ST-F10	A person on a skateboard in mid air.
ST-0	A person is taking a picture of himself
SEP	A person holding a cell phone in their hand.
SEP-F10	A person is riding on a horse in the snow
D-ST	A person riding a skateboard on the street.
T-ST	A person. people sitting to is woman down the of with next man on it

Table 1. Output generated in case the input text is **A person** and no control codes are provided.

Model	Generated sentence
ST	A person on a skateboard doing tricks in the air.
ST-F10	A person is standing on a beach with their surf board.
ST-0	A person flying a kite in the middle of an empty park.
SEP	A person riding a skateboard down the side of an intersection.
SEP-F10	A person standing next to a brick wall.
D-ST	A person riding a skateboard down the street.
T-ST	A person isa. people and table standing on sitting to of street

Table 2. Output generated in case the input text is **A person** and **outdoor, sports** are provided as control codes.

Model	Generated sentence
ST	A person sitting in a kitchen with some items.
ST-F10	A person standing next to a counter holding utensils.
ST-0	A person sitting at a table with several glasses of wine.
SEP	A person holding a hot dog with ketchup.
SEP-F10	A person sitting at a table with food.
D-ST	A person sitting at a table with his laptop.
T-ST	A person a sitting in table on with of is downa

Table 3. Output generated in case the input text is **A person** and **kitchen** is provided as control code.

Model	Generated sentence
ST	Advertisements photo of two people on the beach, one wearing protective gear
ST-F10	A group of people standing next to each other holding glasses.
ST-0	A group of people walking in the rain with their skateboards.
SEP	A baseball player swinging a bat on grass.
SEP-F10	A woman sitting on a bed while holding an open laptop.
D-ST	A young man sitting on a couch holding up Wii controllers.
T-ST	table with sitting group and inA standing people are next of

Table 4. Output generated in case **no input text** is provided and **outdoor, sports** are used as control codes.

Model	Generated sentence
ST	A few women in a restaurant standing up and eating.
ST-F10	A few women sitting on a couch with their dogs.
ST-0	A few women and a man in a kitchen preparing food.
SEP	A few women sitting on couches playing a game with Nintendo Wii controllers.
SEP-F10	A few women standing next to each other.
D-ST	A few women sitting at a table with plates of food.
T-ST	A few women a is on tablea are with and of down it next woman

Table 5. Output generated by submitting the text **A few women**. This is the beginning of a sentence the models were trained on: *”A few women that are standing around some food.”*. Control codes used, same as those used for the training: **furniture, person**

Model	BLEU-2	BLEU-3	BLEU-4	SELF-2	SELF-3	SELF-4	POS-2	POS-3	POS-4
ST	37.59	22.39	13.80	92.04	80.90	66.41	77.43	63.46	49.5
ST-F10	37.48	22.33	13.75	92.31	81.61	67.76	77.42	63.40	49.44
ST-0	35.63	20.46	12.06	94.81	85.2	70.79	75.13	60.41	46.01
SEP	33.27	19.9	12.18	92.39	82.69	70.61	67.50	54.27	41.75
SEP-F10	32.86	19.50	12.00	92.17	82.00	69.50	67.31	53.80	41.25
D-ST	37.13	21.91	13.27	92.01	80.85	66.21	76.90	62.22	37.91
T-ST	0.10	0.012	0.003	31.62	21.54	17.78	1.46	1.02	0.86

Table 6. BLEU metrics.

7. Conclusions and ideas for future directions

We presented how we tackled the task of conditional text generation on a small corpus dataset like COCO captions. After analyzing the results we obtained and having also explained the ideas behind distillation, we wanted to prove that the usage of large models (such as CTRL) is not needed for this kind of dataset, and, for this reason, we chose to rely on smaller models like GPT2 and DistilGPT2. Moreover, we are happy that we have been able to train our models so that they are able to process more combination of more control codes in the prompt (differently from the CTRL implementation).

Nevertheless, we are aware that the BLEU metrics presented in Table 6 don't look to promising, especially if we consider that *"scores over 30 generally reflect understandable translations and scores over 50 generally reflect good and fluent translations"*[9]. However, we believe that this lower metrics values can derive from our harder evaluation policy, in fact the sentences generated by the different models are significantly better than what is expected for such metrics.

Concerning future developments we collect here some ideas:

- this work can be an initial step toward a captioning system with images as input
- more exotic GPT2 or Transformer based architectures can be explored to obtain small models to distill
- further effort should be put in tweaking distillation parameters to obtain better results on smaller models
- application of these same techniques to longer text datasets

References

[1] X. Chen, H. Fang, T. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick. Microsoft COCO captions: Data collection and evaluation server. *CoRR*, abs/1504.00325, 2015.

[2] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models: A simple approach to controlled text generation. *CoRR*, abs/1912.02164, 2019.

[3] A. Ezen-Can. A comparison of LSTM and BERT for small corpus. *CoRR*, abs/2009.05451, 2020.

[4] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang. Long text generation via adversarial training with leaked information. *CoRR*, abs/1709.08624, 2017.

[5] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[6] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. Ctrl: A conditional transformer language model for controllable generation, 2019.

[7] K. Lin, D. Li, X. He, Z. Zhang, and M. Sun. Adversarial ranking for language generation. *CoRR*, abs/1705.11001, 2017.

[8] W. Nie, N. Narodytska, and A. Patel. RelGAN: Relational generative adversarial networks for text generation. In *International Conference on Learning Representations*, 2019.

[9] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002.

[10] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[12] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[13] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[14] W. Yu, C. Zhu, Z. Li, Z. Hu, Q. Wang, H. Ji, and M. Jiang. A survey of knowledge-enhanced text generation, 2020.

[15] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu. Texygen: A benchmarking platform for text generation models, 2018.

[9] Evaluating the Output of Machine Translation Systems