Conditional Text Generation with Salesforce's CTRL

Lukas Canciani Graziani
Politecnico di Torino
278237@studenti.polito.it

Matteo Gally
Politecnico di Torino
277935@studenti.polito.it

Carlo Iurato
Politecnico di Torino
277979@studenti.polito.it

Abstract

In this report we approach the task of conditional text generation through the use of the Conditional Transformer Language Model, CTRL. We have fine tuned a pre-trained model and evaluated the generated sentences with different metrics (BLEU, POS-BLEU, SELF-BLEU), in order to choose the best hyper-parameters combination for both training and generation.

1. Introduction

In the last years, the task of text generation has made significant progress since the first models relying on rules and statistical inference. For this project, we will instead use an approach based on neural networks, which have raised the level of accuracy and variety of generations, taking into account that there are two types of generation:

- Unconditional Text Generation
- Conditional Text Generation

About the former, the main problem is that the generation is mainly based on the content of an input set of examples used for the training phase. This leads to little diversification of the generated text and little adherence to the human expressions. While the latter is influenced by external conditions, such as the context, the topic or the emotion. Today the Conditional Text Generation can be applied to the goal-oriented conversations, chatbots, and Query & Answering (Q&A) systems in order to make the interaction more consistent, smooth and enjoyable. For this project we exploited a predefined model 'CTRL' (provided by Salesforce `https://github.com/salesforce/ctrl`) and we applied it on a dateset of caption from COCO. Our work and other helpful information are available at the link `https://github.com/Carlo13gen/ConditionalTextGeneration`.

2. Tools & Library

For this project we used this tools and libraries:

- pycocotools: Library containing the COCO API [1] to extract the captions from .json files downloaded from COCO website.
- fastBPE[2]: Tool used to translate rare words in more frequent chuncks of characters
- TensorFlow[5]: It is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- gsutil[3]: It is a Python application that lets you access Google Cloud Storage from command line, in our case, we used this tool to download the model of CTRL
- nltk: Natural language toolkit [4], is a suite of libraries and programs for symbolic and statistical natural language processing (NLP), in our case we used this tool to tokenize our sentence and to use our metrics BLEU, SELF-BLEU, POS-BLEU

3. Dataset

For this project we used COCO captions to train model. COCO is a dataset of images and captions, used for object detection, segmentation and image captioning problems. In our case we are interested in the captions we extract from this two file:

- captions train2014.json;
- captions val2014.json;

In such a way we obtain a huge dataset of 414113 captions, we show some examples of these captions below:

1. *"A panoramic photo of a kitchen and dining room."*
2. *"The two people are walking down the beach."*
3. *"A bicycle is parked by a bench at night."*

4. Model

4.1. Language Modeling

Given example sequences of the form $x = (x_1, ..., x_n)$ where each x_i comes from a fixed set of symbols, the goal of language modeling is to learn $p(x)$. Because x is a sequence, it is natural to factorize this distribution using the chain rule of probability [14]:

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

This decomposes language modeling into next-word prediction. Current state-of-the-art methods [15][6] train a neural network with parameters to minimize the negative log-likelihood over a dataset $D = x_1, ..., x^{|D|}$:

$$L(D) = -\sum_{k=1}^{|D|} \log p_{\Theta}(x_m|x_{<i}^{k})$$

Because language models learn $p_{\Theta}(x_i|x_{<i})$, a new $\tilde{x}$ of length m can be generated by sequentially sampling its constituent symbols: $p_{\Theta}(x_0), p_{\Theta}(x_1|\tilde{x}_0), ..., p_{\Theta}(x_m|\tilde{x}_{<m})$.

4.2. Architecture

CTRL is a conditional language model that is always conditioned by a control code c and learns the distribution $p(x|c)$. The distribution can still be decomposed using the chain rule of probability and trained with a loss that takes the control code into account.

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

$$L(D) = -\sum_{k=1}^{|D|} \log p_{\Theta}(x_i^k|x_{<i}^k, c^k)$$

The control code c provides a point of control over the generation process. This is true even when sampling x_0, in contrast to the traditional language modeling framework. CTRL learns $p(x_i|x_{<i}, c)$ by training on sequences of raw text prepended with control codes. After minimal preprocessing, a single example sequence containing n tokens is embedded as a sequence of n corresponding vectors in $\mathbb{R}^d$. Each vector is the sum of a learned token embedding and a sinusoidal positional embedding as in the original Transformer architecture [7]. This sequence of vectors is stacked into a matrix $X_0 \in \mathbb{R}^{nxd}$, so that it can be processed by l attention layers. The ith layer consists of two blocks, each of which preserves the model dimension d. The core of the first block is multi-head attention with k heads that uses a causal mask to preclude attending to future tokens:

$$Attention(X, Y, Z) = softmax\left(\frac{mask(XY^T)}{\sqrt{d}}\right)$$

$$MultiHead(X, k) = [h_1; ; h_k]W_o$$

$$where h_j = Attention(XW1_j^1, XW_j^2, XW_j^3)$$

The core of the second block is a feedforward network with ReLU activation [11] that projects inputs to an inner dimension f, with parameters $U \in \mathbb{R}^{dxf}$ and $V \in \mathbb{R}^{fxd}$:

$$FF(X) = max(0, XU)V$$

Each block precedes core functionality with layer normalization [9][13] and follows it with a residual connection [10]. Together, they yield X_{i+1}:

$$\begin{array}{cc} Block1 & Block2 \\ X_1 = LayerNorm(X_i) & H_i = MultiHead(X_i) + X_i \\ H_i = LayerNorm(H_i) & X_{i+1} = FF(H_i) + H_i \end{array}$$

Scores for each token in the vocabulary are computed from the output of the last layer:

$$Scores(X_0) = LayerNorm(Xl)Wvocab$$

During training, these scores are the inputs of a cross-entropy loss function. During generation, the scores corresponding to the final token are normalized with a softmax, yielding a distribution for sampling a new token. [12]

5. Method

5.1. Preprocessing

Since our goal is to train, use and test a conditional text generation model, we focused only on the textual captions contained in the COCO dataset. Once selected and extracted the two files containing captions we merged them in order to obtain one single dataset containing more than 400k captions. From this dataset we extracted the subsets that we needed in order to perform training, generation and evaluation. We used a set of entire captions to train the model. For what concerns the generation we took a different set of sentences, keeping only the first half of each one, in order to use them as a seed. The evaluation is performed using the same set of captions as generation, but this time we stored complete sentences.

5.2. Training and Fine Tuning

We started our training on a pre-trained model provided by Salesforce itself (Figure 1). The model has been trained on data originated from different sources and with different control codes. The pre-trained model could be trained both by adding samples of already existing control codes or by adding a new control code. We considered our dataset as grouped in one single control code denoted a *caption*. The first step to train the Salesforce's model consists in making TensorFlow records, also called TFRecords, starting from raw text sentences.

This conversion takes in account the control code associated to sentences in the provided file.

TFRecords are a compressed storage for efficient training. Data are treated as a single stream of tokens associated to a control code. The stream is then split into contiguous sequences, each with his control code posed as the first token in the sequence.

The training script provided by Salesforce will then use all the TFRecords files placed in the current directory in order to train the provided model.

The training script provided allow to change the value of a parameter called *iterations*, corresponding to the number of steps computed in the training phase. In order to avoid overfitting the documentations suggests to keep the values low, around 1. Since we want to be able to choose the learning rate used during the training, we applied a minor change to script provided by Salesforce, using learning rate as argument and adding the custom value to the optimizer used for the training.

Since the pre-trained model's size is already considerably large and the computation cost increases rapidly as records do, training on a very large dataset, with the resources at our disposal, would be too time expensive. So we have chosen to divide our dataset in 4 smaller datasets. Each of 10000 captions, and for each of them compute the TFRecord file, using for all of them the control code *caption*.

5.3. Generation and Evaluation

Once the training is completed, the next step is to use the newly updated model to generate captions starting from generation set previously stored. In order to perform the generation starting from a seed file, instead of single seed inserted by prompt, we had to modify slightly the generation script provided by Salesforce. The generation takes into account the seed of the sentence, a control code associated to the desired generation and various parameters in order to generate one or more sentences. In our case the script stores the generated sentences in an external file, later used to compute the performance metrics. The metrics used to evaluate the generation are: BLEU, SELF-BLEU and POS-BLEU. The literature suggests the use of 1000 generated captions to perform the evaluation. After a first evaluation performed on 1000 captions, we reduced the number of generated sentences to 200, since the complexity of the model and the high number of different parameters combination used (243) lead to a too high running time with the resources at our disposal.

6. Experiment and Evaluation

We made several experiments in order to find the best combination for the hyper-parameters we chose to analyze. For the training we decided to observe how the output changes as a function of *learning rate* and *iterations*. For both parameters we chose 3 different values, performing the training over all the 9 possible combinations. For each of the trained models we selected 3 different values respectively for *temperature*, *top-k* and *penalty* to be used during the generation phase. Therefore we have 27 different combinations for each of the trained models and a total of 243 different combinations.

6.1. Metrics

To perform the evaluation of text generated we used the three metrics suggested in the assignment of the project:

- BLEU
- SELF-BLEU
- POS-BLEU

We used their implementations provided by the NLTK library in Python.

All the three metrics are *n-gram overlap metrics*: measures commonly used in NLG (Natural Language Generation) systems, they compute the degree of matching between machine-generated and human-authored texts. [8]

6.1.1 BLEU

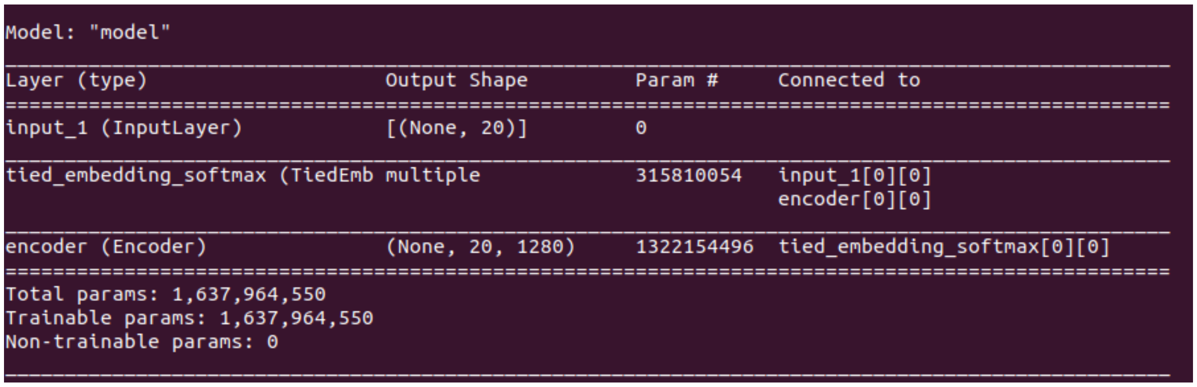
BLEU stands for Bilingual Evaluation Understudy, it is a score of how much similar is a candidate text compared with one or more reference translations. The closer a machine translation is to a human one, the better will be the BLEU score. The score is computed for every candidate caption against all the reference sentences in the evaluation dataset. Unfortunately BLEU metric does not evaluate if the text makes sense so we can have high scores even if our generated sentences have a scarce meaning.

6.1.2 SELF-BLEU

SELF-BLEU is a metric which evaluates the diversity of generated text, so it is used to establish how much sentences produced by the model are different between them. An high value of SELF-BLEU indicates that there are a lot of repetitions, while a too low value indicates higher diversity. [8]

6.1.3 POS-BLEU

This third metric is the same of BLEU, but the score is computed on Parts of Speech (POS). All the words inside captions are converted in a POS-tagger which identifies the



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 20)]         0
__________________________________________________________________________________________________
tied_embedding_softmax (TiedEmb multiple             315810054   input_1[0][0]
                                                                 encoder[0][0]
__________________________________________________________________________________________________
encoder (Encoder)               (None, 20, 1280)     1322154496  tied_embedding_softmax[0][0]
==================================================================================================
Total params: 1,637,964,550
Trainable params: 1,637,964,550
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 1. Model image

word: for example the word ”man” will be classified as a noun, while the word ”red” will be classified as an adjective. For this reason POS-BLEU is used to evaluate the quality of generated sentences. [8]

6.2. Training parameters

In this section we will briefly describe the parameters we chose to fine tune our model. For each of the parameters we will provide a graph that shows how the average values of our evaluations metrics changes according to the parameter variations itself.

6.2.1 Learning rate

Learning Rate it is practically the measure of how fast our model is learning. From a theoretical point of view it determines the step size at each iteration while moving toward a minimum of a loss function. A high value for this parameter will make the learning jump over the minimum, while a low value may be inefficient, since the model could not acquire new information (a too low will take too long to converge or get stuck in a local minimum).
We took 3 different values for *learning rate*, as shown in the following image and table, and we computed the average scores using the three different metrics.

Table 1. Mean values for BLEU, SELF-BLEU, POS-BLEU with different Learning Rate.

LR	BLEU	SELF-BLEU	POS-BLEU
0.1	0.539	0.712	0.527
0.01	0.556	0.491	0.544
0.001	0.607	0.412	0.590

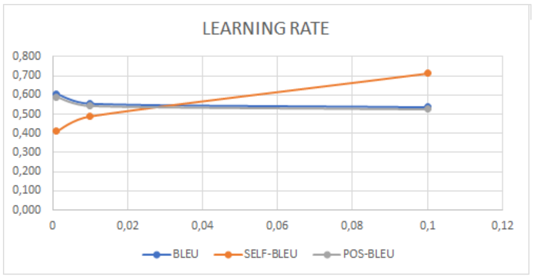


Figure 2. Score variations according to learning rate

6.2.2 Iterations

The parameter *iterations*, also called epoch, defines how many times the entire dataset is passed to the learning algorithm. If we use a high number of iterations we will have a model which perfectly knows all the captions used for training, but which will only be able to reproduce those sentences without building up new texts. On the other side choosing a low value for this parameter, may result in having a model which has not learnt enough to produce high quality sentences.

We took 3 different values for this parameter and we show in the table and image below the mean scores achieved using the three metrics

Table 2. Mean values for BLEU, SELF-BLEU, POS-BLEU with different *iterations* values.

Iterations	BLEU	SELF-BLEU	POS-BLEU
1	0,582	0,511	0,567
5	0,591	0,535	0,576
10	0,507	0,583	0,496

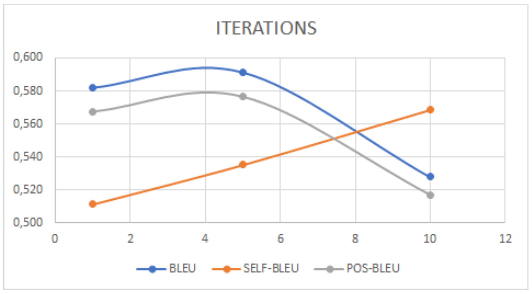


Figure 3. Score values variations according to different *iterations* values

6.3. Generation Parameters

6.3.1 Temperature

The *temperature* parameter determines the probability distribution used to choose the next token among the next *top-k* possible. With T $\to 0$ we have a greedy approach in which the more probable next token has a higher probability to be chosen. On the other side when T $\to +\infty$ the probability is flattened, heading to a uniform probability distribution. Therefore a low value of T will lead to a more deterministic generation, because in this case the peaks in probability distribution are emphasized in such a way that more probable tokens are chosen. While a high value will lead to a more random token choice, which could decrease the quality of the generation. We took 3 different values for *temperature*

Table 3. Mean values for BLEU, SELF-BLEU, POS-BLEU with different temperature values.

Temperature	BLEU	SELF-BLEU	POS-BLEU
0.1	0,672	0,400	0,653
0.2	0,522	0,628	0,512
0.5	0,507	0,583	0,496

6.3.2 Top-k

While generating a sentence, the probability of each token to be the next is computed and the tokens with higher probability are selected. The parameter named *top-k* determines the number of tokens chosen to be selected as next token for the generation. In the same way as *temperature* a low value will lead to more deterministic sentences based on a lower number of variations given from a starting seed and again

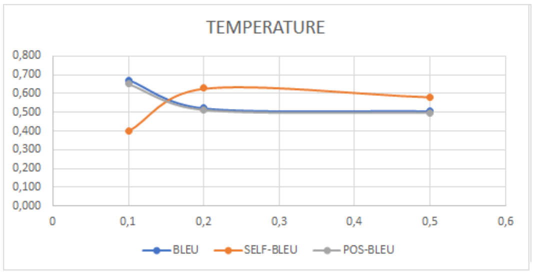


Figure 4. Score values according to different *temperature* values.

a high value will lead to more varied generations but could possible lead to a decrease of quality. We took 3 different values for *top-k*:

Table 4. Mean values for BLEU, SELF-BLEU, POS-BLEU with different *top-k* values.

Top-K	BLEU	SELF-BLEU	POS-BLEU
1	0,590	0,566	0,575
5	0,559	0,531	0,546
10	0,572	0,540	0,573

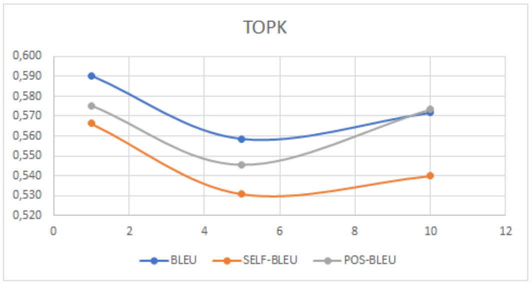


Figure 5. Score values according to different *top-k* values.

6.3.3 Penalty

The *penalty* parameter is used to discount the scores of token already used in the current generation. Higher values of the parameters lead to a greater penalty for repetitions. Setting the value of the *penalty* parameter to 1 does not penalize repetitions. We took 3 different values for *penalty*:

Table 5. Mean values for BLEU, SELF-BLEU, POS-BLEU with different *penalty* values.

Penalty	BLEU	SELF-BLEU	POS-BLEU
1	0,527	0,557	0,514
1.2	0,587	0,539	0,568
2	0,588	0,518	0,568

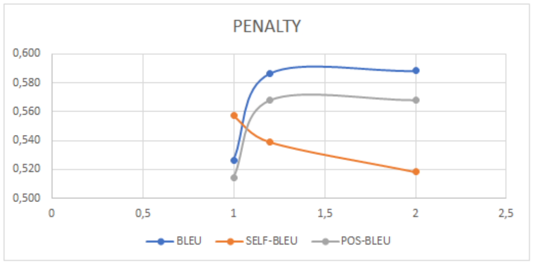


Figure 6. Score values according to different *penalty* values

6.4. Results

In section we analyze the results in terms of scores computed using the metrics we talked about in the previous section. The literature suggested to perform evaluations using 1000 captions, but since we had not enough resources and time, we computed the scores over 200 captions using a total of 243 combinations of hyper-parameters. evaluating the avarage of our metrics we obatined poor results, with an average score of 55%. The specific mean scores for BLEU, SELF-BLEU and POS-BLEU are specified in the Table 1 down below.

Table 6. Mean values for BLEU, SELF-BLEU, POS-BLEU.

Mean BLEU	Mean SELF-BLEU	Mean POS-BLEU
0.538	0.554	0.567

The best hyper-parameters set has an average score of 72% (0.726). The values for parameters are:

- Learning Rate = 0.1
- Iterations = 10
- Temperature = 0.5
- Top-k = 1
- Penalty = 1.2

The single scores for this set of parameters are shown in the table below.

Table 7. Values for BLEU, SELF-BLEU, POS-BLEU using best parameters.

BLEU	SELF-BLEU	POS-BLEU
0.597	0.497	0.589

7. Conclusion

Analyzing the results obtained by our experiments and computing an average between the chosen metrics, after pre-processing the Self-Bleu score, since it desired value is on 0.5, we isolated the best hyper-parameters combination for our task. Even if most of the combinations we used lead to disappointing results in terms of evaluations metrics, some specific combination achieved remarkable results. In particular, some combinations have stood out significantly from the others, like our best case. The result does not surprise us completely, since analyzing the behavior of the individual parameters, as shown in the graphs of the previous chapter, we can see that many of these have been chosen in such a way to maximize the average scores. This are some examples of captions generated during our experiments:

- ”A giraffe is resting by some rocks, and a small dog is sitting next to it.”
- ”A passenger airplane that crashed in the mountains of the Andes in Peru, killing all the passengers”
- ”A family of four living in a small house in the middle of nowhere.”
- ”A motorcycle parked behind a car in a parking lot.”

8. Future works

One big improvement could be given by using 1000 generated sentences to evaluate the performance of the parameter combinations, as suggested by the literature, instead of 200 as we used in our experiments. Our resources, both in time and computational terms, were limited so we tested the generation of 1000 captions for a single combination, which lead to significant changes in terms of scores using our metrics.

Another possible improvement to this work could be, with adequate resources, use the whole Coco dataset to train the model instead of a subset. In this case, having a larger number of samples for the training, it could be possible to differentiate further the control codes of the captions, maybe using the category or the subcategory of the corresponding photo as a additional control code.

Table 8. Values for BLEU, SELF-BLEU, POS-BLEU using same parameters, but generating different number of sentence

NUM	BLEU	SELF-BLEU	POS-BLEU
200	0,589	0,423	0,582
1000	0.399	0.688	0.389

References

[1] Coco api. https://github.com/cocodataset/cocoapi.

[2] fastbpe. https://github.com/glample/fastBPE.

[3] gsutil. https://cloud.google.com/storage/docs/gsutil.

[4] Nltk. https://www.nltk.org/.

[5] Tensorflow. https://www.tensorflow.org/.

[6] R. C. D. L. D. A. Alec Radford, Jeffrey Wu and I. Sutskever. Language models are unsupervised multitask learners.

[7] N. P. J. U. L. J. A. N. G. K. Ashish Vaswani, Noam Shazeer and I. Polosukhin. Advances in neural information processing systems.

[8] J. G. Asli Celikyilmaz, Elizabeth Clark. Evaluation of text generation: A survey. 2021.

[9] R. K. Jimmy Ba and G. E. Hinton. Layer normalization.

[10] S. R. Kaiming He, Xiangyu Zhang and J. Sun. Deep residual learning for image recognition.

[11] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines.

[12] L. R. V. C. X. R. S. Nitish Shirish Keskar, Bryan McCann. Ctrl: A conditional transformer language model for controllable generation.

[13] A. R. Rewon Child, Scott Gray and I. Sutskever. Generating long sequences with sparse transformers.

[14] P. V. Yoshua Bengio, R´ejean Ducharme and C. Jauvin. A neural probabilistic language model. journal of machine learning research.

[15] Y. Y. W. J. C. Q. V. L. Zihang Dai, Zhilin Yang and R. Salakhutdinov. A neural probabilistic language model. journal of machine learning research.