|  |
| --- |
| **This Band does not Exist** |
| **A Framework for Generating Objects from a Given Domain** |
| **Naor Haba** and **Niv Sfaradi** |
| Technion - Israel Institute of Technology |
| Emails |
|  |
|  |

Abstract

In this project we present a framework for language generation, aiming to create fake objects from any domain. We assume a domain is represented by a dataset containing objects with several attributes. One can feed the framework with the dataset’s objects, which are then parsed together with special tokens into a certain format. For each object, the new format includes each of its attributes, separated by the special tokens. By using this format, we train a generative language model, capable of learning the interactions between the attributes, which can then be used to generate fake objects from the given domain. To demonstrate the use of our framework, we attempt to solve the problem of bands generation. We present a dataset containing ~300,000 records of bands songs. Each record contains band-name, genre, song-name and lyrics. Therefore, by defining our domain as “bands”, we try to generate bands with these attributes. By using special tokens such as [BAND], [GENRE], etc. we parse a given record to the format of “[BAND] band-name [GENRE] genre-name…”, which is then passed to a generative model. Our framework uses pretrained models such as BERT, GPT2 and T5. Each model is constructed together with a language model head (masked or causal) which allows it to learn the format of the given domain. By using different sampling methods we can generate completely new bands with these models. Moreover, to assure the quality of the generated bands, we include a pipeline of rules that filters out irrelevant generated records. [[1]](#footnote-1)

Introduction

Text generation is a natural language processing field, in which text with specified requirements is being automatically generated. In a generation task, a system is given a dataset of texts as an input, and is required to generate texts based on the given dataset and on certain rules. The text generated by the system is meant to be coherent, as if a human wrote it.

As the use of deep learning for multiple fields grows, it is only natural to see many uses of deep learning concepts in the fields of natural language processing, namely in text generation. Different proposals have been made for solving text generation tasks, and lately the main focus is on using recurrent neural networks (RNNs) and transformers.

In this work, we use transformers to tackle the text generation challenge. We present a generation framework, that given an input (that can be empty) generates fake text, attempting to imitate the domain of the text it trained on. We use transformers (GPT2) to train a generation model, and then use this model to generate text in the given domain. The framework has two main capabilities – generating text given an input, and generating text with no input given.

Related Work

This work is following the work of this-word-does-not-exist [[2]](#footnote-2), which aims to generate dictionary words with definition and examples of use in a sentence. This work is one of many related project aiming to generate objects from a specific domain.

The Data

* 1. The Dataset

The Domain we choose for demonstrating our framework is the music bands domain. We use a dataset that contains songs by multiple bands, with the following columns: Artist, Song, Genre, Language, Lyrics. This dataset consists of ~300,000.

* 1. Data Preparation

After removing non-English records and empty or defected records, we remain with X records.

We transform the data, so each record consists of a single text, separated by special tokens. The special tokens are [ARTIST], [GENRE], [SONG], [LYRICS] and [END], and a typical record has the following format: “[ARTIST] Artist Name [GENRE] Genre Type [SONG] Song Name [LYRICS] Song Lyrics [END]”.

1. Training
   1. The Models

The framework is meant to support generating with input and without input. Therefore we train two models, a forward model and a backward model. Both of the models are based on GPT2.

The forward model is training on generating band-name, genre, song-name and lyrics from scratch, and also on generating each of these objects based on the objects preceding it – genre can be generated based on band-name, song-name can be generated based on band-name and genre, and lyrics can be generated based on band-name, genre and song-name.

The backward model supports generating according to a given input, but not in this order. Based on a given song-name, it generates genre, band-name and lyrics.

* 1. The Training

We tune these models with the following parameters:

* Optimizer – AdamW
* Learning rate – 5e-5
  + Linear scheduler
* Batch size – 8
* Epochs – 3

Training was performed for 20 hours.

1. Generation

We use the tuned GPT2 model to generate new data. Each generated record consists of a band-name, its genre, a song-name and its lyrics. The generation is being performed with sampling (and not in a deterministic way), and is split to two types of generation: generating from scratch and generating given an input.

* 1. Generating from Scratch

We use the model to generate records from scratch.

Each record follows a validation pipeline, to make sure it follows the next rules:

* Band name is unique, that is, does not appear in the original data. This is being validated with a list of forbidden band-names (all band-names from the training data).
* Genre is one of the genres in the training data.
* The lyrics are longer than a given limit.

Generating with a Given Input

The framework supports accepting an input and generating objects with this input, for example – generating lyrics given a band-name, a genre and a song-name.

Here the records also follow a similar validation pipeline, but if the model was given an existing band-name as input, it skips the band-name validation part.

1. Evaluation

As we don’t have a ground truth for comparison, we need to carefully develop an evaluation process. Our main goal is to check if our data resembles real data well enough to confuse a decent classifier.

* 1. Baseline Datasets

We create two baseline datasets, where each record represents a band, consisting of band-name, genre, song-name and lyrics. The band-names, genres and song-names are taken from our generated dataset, and the lyrics are generated using two different methods:

* Sampling random words from the genre’s corpus.
* Sampling words from a tf-idf distribution for each genre. The tf-idf is calculated for each genre by using all of its lyrics from the entire dataset.

The baseline datasets are these:

* A dataset of the random-fake data combined with real data (from the original dataset). Will be referred to as “random-real”.
* A dataset of the tfidf-fake data combined with real data. Will be referred to as tfidf-real.

Each of these datasets is labeled with a True label for the fake data, and a False label for the real data. Since the fake data is labeled as True, we will concentrate on recall evaluation.

* 1. Classification Model

To evaluate our generated data, we use a model for binary classification: We use BORT, a language model suited for classification which is based on BERT, to classify bands to real or fake.

* 1. Training

This model is trained twice – once on the random-real data and once on the tfidf-real data. We then evaluate these model both on the fake data that they trained on (using a held-out validation set), and our generated data.

We trained each model on X rows (split to 0.8X training and 0.2X evaluation), and tested it on X rows as described above.

* 1. Expected Results

Our way of evaluation assumes that the following results mean that our generated data is good:

* High recall on random / tfidf dataset:   
  Meaning the model is capable of learning and of detecting fake data.
* Low recall on our generated dataset:  
  Along with the previous score, this score shows that although the model is capable of detecting fake data, it seems uncapable of distinguishing between our generated data and the real data.

1. Results

**Creating:** To create a new Figure or Table, insert a Text Box where you want it to appear (generally, centered at the top of a column close to where it is referred to) and then fill it in with the Figure (or Table). Highlight and right click to add Caption, with the ACL Caption style (or ACL Caption Long style for multi-line captions), which places 10 pt below and above the caption.

**Numbering:** To update numbering, highlight all the relevant text (*e.g.*, **Ctrl-A + F9**). This will update all the numbering applicable to tables, figures, equations, and headings.

**Cross-referencing:** To add a cross reference to a figure or table:

* Place the mouse pointer at the location where you wish to add the cross-reference.
* Click on the **Insert** menu, (then click **Reference**), and then **Cross-reference** in the **Links** panel.
* In the **Cross-reference** dialog box, click the caption to which you are building the text reference.
* For a figure, under **Reference Type**, click **Figure**.
* Under Insert Reference To, click Only Label and Number, then click OK.
* As much as possible, fonts in figures should conform to the document fonts (this is not the case in the example figure).

This is an example reference to Figure 1.

Hyperlinks

Within-document and external hyperlinks are indicated with Dark Blue text, Color Hex #000099.

References

To create hyperlinks between citations and references, as you insert each full reference in the References section, highlight it and then select Insert, Bookmark. Link back to the reference from its citations in the text by highlight the citation, right clicking, and selecting Insert, Cross-Reference, then selecting the Bookmark you’ve saved. Highlight the citation again to give make it dark blue (included in this theme), if it is not automatically applied. If there are problems saving the hyperlinks when you convert the document to PDF, use an online converter such as <http://go4convert.com>.

Citations

Citations can be created by creating in-document hyperlinks to bookmarks you’ve created. Go to Insert / Hyperlink / This Document / Bookmarks, and select your bookmark.

* 1. Equations

An example equation is shown below:

(1)

To add new equations, authors are encouraged to copy this existing equation line, and then replace with the new equation. The numbering and alignment of equation line elements is automatic. To update equation numbering, press **Ctrl-A + F9**. Note: this will only update the number to the right of the equation; to update numbering within the text you must create a cross-reference.

**Cross-referencing:** To create a cross-reference for an equation:

* Create a bookmark for it.
* Select the number to the right of the equation. Go to **Insert**, **Bookmark** (in the **Links** panel),andthen create a name for your equation. Press **Add** to create the bookmark.
* To refer back, place the mouse pointer at the location where you wish to add the cross reference.
* Go to **Insert, Cross-reference** (in the **Links** panel).In the dialogue box, select **Bookmark** and **Bookmark Text** from each dropdown list. Uncheck **Insert as Hyperlink**, then click **OK**.
* This will make it such that whenever a new equation is added, the references to the equation will be updated when **Ctrl-A + F9** is pressed.
* This an example cross-reference to Equation 1.

Appendices

Appendices, if any, directly follow the text and the

references. Letter them in sequence and provide an informative title: **Appendix A. Title of Appendix**.

1. MS Word STREAM Tools

This Microsoft Word file was updated in 2016 with STREAM Tools, designed for creating well-formatted reports and papers with Microsoft Word (Mamishev, 2010; Mamishev, 2013).

Acknowledgments

An example acknowledgment.

References

Alfred. V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling, volume 1*. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual.* American Psychological Association, Washington, DC.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. [Alternation](http://dl.acm.org/citation.cfm?doid=322234.322243). *Journal of the Association for Computing Machinery*, 28(1):114-133. https://doi.org/10.1145/322234.32224.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503-512.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. [Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing](http://aclweb.org/anthology/P16-1001). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1–11. <https://doi.org/10.18653/v1/P16-1001>.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Mary Harper. 2014. [Learning from 26 languages: Pro- gram management and science in the babel program](http://aclweb.org/anthology/C14-1001). In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin City University and Association for Computational Linguistics, page 1. <http://aclweb.org/anthology/C14-1001>.

Alexander V. Mamishev and Murray Sargent. 2013. *Creating Research and Scientific Documents Using Microsoft Word*. Microsoft Press, Redmond, WA.

Alexander V. Mamishev and Sean D. Williams. 2010. *Technical Writing for Teams: The STREAM Tools Handbook*. Wiley-IEEE Press, Hoboken, NJ.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. *Yara parser: A fast and accurate depen-dency parser*. *Computing Research Repository,* arXiv:1503.06733. Version 2

1. Appendices

Appendices are added after the References section by restarting the header numbering using style “A, B, C”.

1. Supplementary Material

Supplementary material also be included with the Appendices.

1. Our code and data are available at: Add GIT repo [↑](#footnote-ref-1)
2. This Word does not Exist: Add GIT repo? [↑](#footnote-ref-2)