Are comedians safe from AI? Exploring GPT2 humour-generation skills.

Susanna Pinotti

Digital Humanities and Digital Knowledge, Dec. 2022

Abstract

This paper illustrates an experiment conducted with Natural Language Generation and GPT2, where I attempted to fine tune the pretrained model with a task-specific dataset, to teach the model to generate funny jokes. The project also explores the challenge of evaluating such task, focusing especially on the qualitative evaluation of the obtained results, adopting an ad hoc procedure for this kind of task, which relies on some text mining techniques such as sentiment analysis and topic modelling to assess the similarity between the training set and the generated text. The results were only partially satisfactory, since humour is quite a special linguistic phenomenon, and further exploration into its mechanisms would be beneficial to a further experiment.

1 Introduction

When proposing the now famous 'Imitation Game', Alan Turing postulated that an intelligent machine would be able to trick human interlocutors into believing they are having to do with another human, in every aspect and observable manner. This reality has recently been made a little more realistic by the advent of deep learning and neural networks, which are now able to produce human like examples of images, sounds, and text. Most recent examples of such astonishing results are image generating networks such as OpenAI's DALL-E, which uses a version of GPT3 modified specifically to generate images from linguistic prompts. GPT3, which was launched in July 2020, generated both great enthusiasm and concerns for the potential risks of achieving such staggering generating results. In the big field of Natural Language Processing (NLP), natural language generation is the task of generating new, unseen text from scratch, starting from a language model which understands the target language (English or any other natural language for which we have enough text) as a set of non-linguistic features representing the information and the meaning carried on by words. Natural Language Generation (from now on, NLG) can be viewed as complementary to many fields of study: from Computational Linguistics to Natural Language Understanding and NLP. NLG existed since the early days of computing, but it is with the deep learning revolution in the latest 15 years that it achieved its first satisfactory results, thanks to transformers architectures such as BERT and GPT-2 (and now GPT-3), which can produce very realistic sentences, undistinguishable from human ones. This project focuses on NLG, specifically, trying to teach GPT-2, OpenAI enormous text generating model, to generate humorous text (jokes, puns etc..), fine tuning it to a task specific corpus. The task can seem quite hard, given that humour is far from having a single, shared, ubiquitous definition linguists agree upon. While "standard" text generation is now considered an achieved task, producing outputs specific to a non-trivial variety of language, such as humour or poetry, can present some difficulties, given that these texts present some very specific mechanisms, which are not always easy for the network to capture. Transfer learning is an easy and powerful process, and it will be interesting to see if the final model proves able to grasp the core structures of humorous text. The first step of

this project was thus to analyse the dataset, in order to be sure that it was suitable for the task and well balanced. Very little pre-processing was needed, since GPT-2 comes with its own tokenizer, and so I jumped directly in the fine-tuning process, training the network with different training arguments, to see which one achieved the best performance overall. Finally, I created a smaller dataset made of generated text only, to compare it with the training set, and see if the model proved able to catch the main words of the topics that emerged in the initial analysis. The paper is divided into a first section, in which I briefly illustrate the functioning of transformers, GPT-2 and transfer learning. Then a small section in which I briefly touch on the linguistics of humour, what is humour and what linguistics theories tried to capture its functioning. Finally, the paper will illustrate the fine-tuning process and the obtained results, with detailed information on the model performances and evaluation strategies, including the comparison among the training dataset and the generated dataset.

All the Colab notebooks and data, as well as a small website comparing Topic Modelling results can be found at this GitHub repository

---

2 Context

2.1 Transformers' architecture

A huge problem in computational linguistics and NLP is to deal with context: the meaning of a sentence is not the mere result of adding the meaning of each individual word, but it is rather the interaction of these words, how they refer to one another and how they are mutually dependent. Words, or tokens, are not isles. Meaning can be conveyed explicitly or implicitly, by omitting already-known words (ellipsis), using pronouns to refer to entities that have already been named, and many other confusing things that can sometimes cause miscomprehension even among humans, let alone machines. For instance, it is pretty easy, if you are human, to understand that the pronoun "*it*" in the sentence "*I accidentally dropped the vase to the floor, making it shatter into a thousand pieces*" refers to the vase that I dropped. Furthermore, some words have some peculiar properties, such as showing up more often in pairs, such as "salt" with "pepper" or "cats" with "dogs". Meaning can be encapsulated by more than one word, which is the case of poli-rhematic words, such as "credit card" or "take part". In order for a machine to be able to grasp such references and peculiarities in natural language, we need some sort of way to take into account not just each word separately, but also their co-text. Recurrent Neural Networks, which are architectures introduced more than 10 years ago, were implemented with a special context-sensitive layer: a hidden state that could output probabilities for the next token in line, having a sense of the preceding context. RNNs have been the standard up until sometime, since they were quite good at understanding short sentences meaning, but they had some difficulties with long sequences. Even more problematic, RNNs are very hard to train since their processes are hard to parallelize and can easily fall into *exploding gradient problems*. RNNs were soon dismissed, at least for NLP tasks, when a new and more performant architecture, the *Transformer*, was presented in a famous paper "Attention is all you need" (Aswani et. Al) [1] by Google Brain researchers. These models scale very well with huge datasets, since they can be parallelized, and more importantly, they introduce big innovations: *the positional encoding* and the *self-attention mechanism* [10]. First, Transformers are what is called encoder-decoder models, or sequence to sequence models. What follows is a schematic representation of the general Transformer architecture as presented in the original paper.

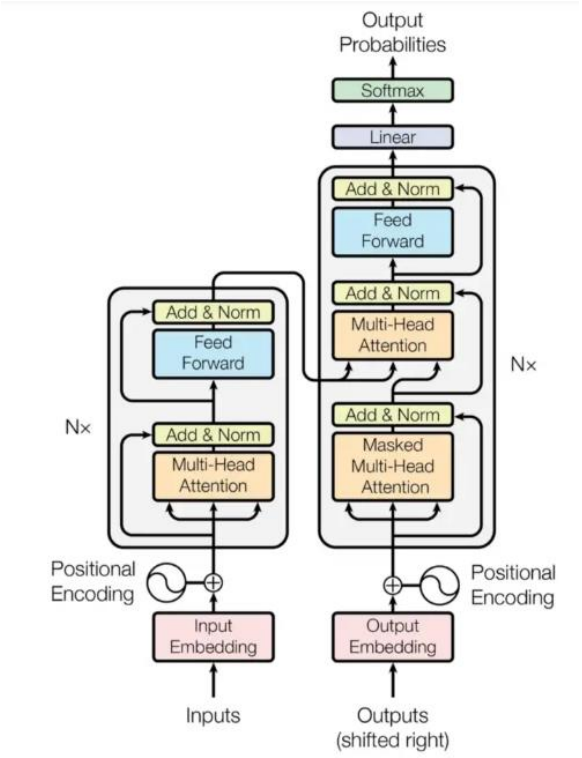

Figure 1.The Transformer architecture, as presented in the original paper

The encoder is the first block, its task is to receive the input sequence and outputting the same sequence in the form of a vector (tensor) which represent the sequence features. For each word in the sentence, the encoder assigns a positional information, which accounts for the words in its co-text, meaning its position with respect to all the other words. This moves the burden of encoding word order away from the structure of the neural network to the data itself. First, each input word of the sentence is turned into a vector using a word embedding algorithm. The sequence passes then through a self-attention layer, which is the context-aware layer of the network. The layer creates three vectors for each embedding: a *Query* vector, a *Key* vector, and a *Value* vector [10]. These vectors are abstraction useful for calculating attention, allowing the model to encode the meaning of the original input sentence. To calculate the attention score of each vector, we take the *dot product* of the query vector and the key vector. To follow that, the vectors are passed to a *Feed Forward Neural Network*. At this point, the encoder has acquired a good understanding of the input sentence and is ready to produce the output to pass to the decoder. The decoder uses the sentence representation (its features), to generate a target sequence of its own. Since Transformers are originally thought to be translators, this usually means the same sentence in another language. The internal functioning of the decoder is quite similar to the encoder one, except for the fact that the attention layer of the decoder cannot use all the words in the sentence, but can only work sequentially, paying attention specifically to the part of the sentence that has already been translated (generated). For instance, if the decoder must generate the third word of the output sentence, it can dispose of the full encoding of the starting sentence but can only predict the current word of the translated sentence. Transformers are language models, meaning that they acquire a general stochastic understanding of the target language, being it English or any other language we have enough written data of. They are trained in a *self-supervised* way, meaning they train themselves to learn one part of the input from another part of the input and auto-generating labels, and thus require an exceptionally large quantity of raw language data. This means that Transformers are generally very large models (few exceptions, like DistilBERT), since performance increases with model size, and training is consequentially very expensive, time and

computation wise. This even implies massive amounts of CO2 emissions, especially for state-of-the-art models such as GPT-2 and BERT, which is why transfer learning and fine-tuning are usually preferrable than training from scratch, when possible.

2.2 GPT-2 and text generation

Sequence to sequence models (encoder-decoder models) can be used to solve task like machine translation or text summarization, but there are tasks for whose we do not actually need the whole structure and the model blocks can be used independently. Models such as BERT, RoBERTA, ALBERT and others are encoder-only models and are best suited for tasks that require the full understanding of the sentence, such as named entity recognition, word or sentence classification and question answering. These models pay a "bi-directional" attention and, at each stage, can access all the words of the sentence. These models are often called "*auto-encoding models*". Models such as GPT, CTRL and TransformerXL are called "*auto-regressive models*" and they use only the decoder part of the transformer architecture, with a masked attention mechanism involved. At each stage of processing, they can only access the word that has already been processed, having to guess the next words in the sequence. These models are most suitable for text generation tasks. Some of the best models that have recently been proposed are OpenAI's GPT-2 [2] and now GPT-3. This paper will deal with GPT-2, since it is the object of the project and freely available on the HuggingFace hub, but the model has been recently potentiated, and GPT-3 has slowly begun to be available to the public, achieving superlative results. GPT-2 was released in 2019, as a direct scale-up of GPT, and was trained on a huge amount of raw text data, specifically up to 8 million of web pages, counting to circa 40GB of total data. With its *1.5 billion parameters*, it is one of the biggest transformer-based models available. The goal of the training was simple, the network should learn to predict the next word in a sentence, given the preceding ones. GPT-2 achieved state of the art performances, being able to generate text that reads astonishingly human-like and has been even deemed possibly susceptible to malicious use, due to its capability of generating fake text [9]. While this is true, it is fairly noticeable that GPT-2 can have problems generating coherent long texts, since it can become repetitive, and, more importantly, it encounters some difficulties in generating specific varieties of language, such as humour, poetry or lyrics, and it needs to be fine-tuned to achieve good results in these tasks. Fine tuning relies on the mechanism of *transfer learning*: this refers to the re-training of pre-trained models to a smaller and more language-variety-specific dataset, to produce outputs that resemble that specific variety. In practice, fine tuning means re-training the entire model on new data using a very low learning rate, incrementally tweaking the pretrained features to the new data. Transfer learning allows to train large networks on small datasets, which is very convenient for language phenomena for which we don't have enough examples to train from scratch.

2.3 The problem of generating humour

What is humour? What does it take for a text to be funny? These are open questions in linguistics that deserve some kind of definition, let alone clarification, before diving into the fine-tuning part of the experiment. At first, it is necessary to distinguish among "*referential*" and "*verbal*" humour [4]: while the former is purely semantic/pragmatic and independent of the linguistic form, the latter is more strictly linguistic and depends more closely on the signifier. Examples of verbal humour are puns, repetitions (such as alliterations), or ambiguity-based humour. Both "verbal" and "referential" humour, at the end, convey an incongruency at the pragmatic level, since humour is never just linguistically conveyed (no word is humorous on its own, except maybe some onomatopoeia), but contextually constructed. Dated works proposed a simple and general definition of irony: a message is humorous if it conveys the opposite of what it says or write. This classical view of irony and humour

has its weaknesses, as Sperber and Wilson care to point out [5], stating that humour can be understood also in terms of *echoic allusions* in language, and irony highly depends on its recognition among speakers, in accordance with *Relevance Theory*. Other scholars, especially the lexical semanticists such as V. Raskin proposed a very influential theory in the study of humour: the *semantic-script theory* (SSHT)[4]. According to Raskin, a text is funny if (1) it is compatible, fully or in part, with two different scripts and (2) the two scripts the text is compatible with are opposite in a special sense. More generally, the literature deals with the various specific kinds of humour, making subtle distinctions among irony, sarcasm, humour and so on. For the sake of brevity, I will use these terms quite interchangeably since the dataset certainly contains a mix of everything. Besides all the possible consideration linguists can do on the matter, it is worth investigating if Artificial Intelligence can be used to help understand the mechanism of humour, or if it can be trained to grasp what it needs to recognize it and produce it. Some experiments have been conducted, especially with detection tasks where models were trained to label text as humorous or offensive [11] or some experiment on style transfer [3] and they generally achieve good results, suggesting that research on the subject is thriving.

3 Data

The dataset I selected for performing the task of fine-tuning GPT-2 to generate humour can be freely downloaded from Kaggle or used directly with HuggingFace through the Dataset API. The dataset is called “Short Jokes” and it is a .csv file containing approximately 230.000 utterances of jokes extracted from the web (mostly from Reddit.com), of short to medium length (from 10 to 200 characters). The author [here his GitHub] states that he tried to clean the dataset from jokes that could be considered too offensive as best he could, but also reminds that, among so many utterances, he may have missed some, and declines any responsibility. Plus, removing every “dark” joke from the dataset would compromise its representativeness, in the sense that dark humour is, in fact, humour after all. It is worth noting that different people find different things funny and amusing, so much so that something the creator of the dataset personally would consider offensive may be amusing to someone else, and vice versa, making it difficult to remove any offensive joke. The first thing I did when analysing the dataset was to look at the average length of the utterancesl: the average length of the jokes was of 93.05 tokens (with st. dev. 35.25), showing that the length of the sentences is quite variable. Then, after some pre-processing (removal of stop words, lowercasing etc..) I went on at calculating some initial frequency distribution of words, to get a first clue about what could be the most fashionable content of the jokes.

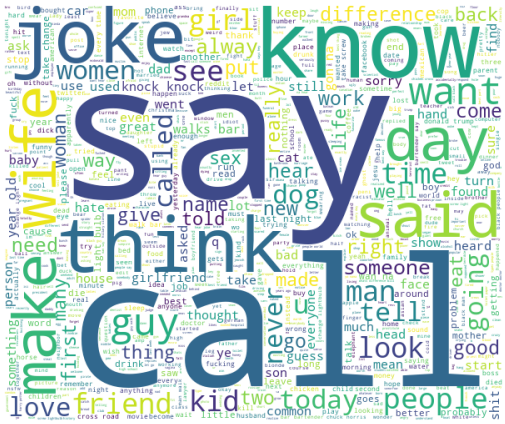

Figure 2.Wordcloud of the dataset

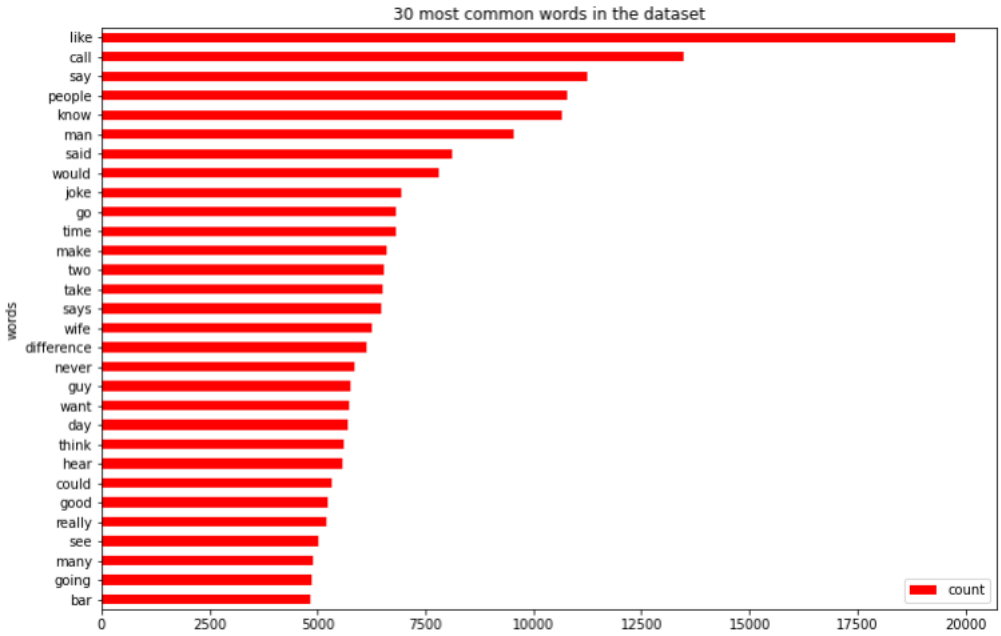

Figure 3.Most frequent words, ordered

By looking at these two charts, it is possible to guess which will be the most common subjects of the jokes in the dataset. Verbs like "call", "say", "joke", "make", "think" or "hear" are, of course, very common verbs on their own, but they are also quite specific of the humorous language, think for example to a classic formula sounding like this : *"What do you call a [obj] that [verb]?"* followed by a punch line. Nouns like "wife" "woman" "man" "bar", instead, are some of the most common objects of such jokes. Some other words that emerged by this first analysis made me think of classic jokes formulations, such as "bar" that may be indicating the presence of the formula "*a man walks into a bar and..[punchline]*", or "difference" like in "*what's the difference between a [obj] and [obj]? [punchline]*". To further analyse the dataset, I applied two more sophisticated techniques of text mining: Topic Modelling, specifically using *Latent Dirichlet Allocation algorithm* (LDA) [6], and Sentiment Analysis. The topic modelling analysis is intended to uncover the underlying topics in the dataset, to have an idea of what to expect once the fine-tuned model is going to generate its own utterances. In the LDA algorithm, text is described as a combination of words, where each word is a discrete probability distribution defining how probable each word is to appear in a particular subject. These probabilities thus represent a document in a concise way. The sentiment analysis is useful for checking, once again, the balance of the dataset in that sense: if there is good balance between positive, neutral and negative sentiment, the model bias towards a specific sentiment will lower. This analysis will also be useful to evaluate the model performances, since a good resemblance between the training set and a generated set of utterances, both in terms of words distribution, sentiment and topics, should be a sign of success. The following are the synthetic results obtained during the topic modelling and the sentiment analysis. The LDA algorithm performed optimally when setting K=4 , since 3 was giving quite general results, and 5 was already presenting some major clusters overlapping.

```
[(0,
  '0.009*"said" + 0.007*"wife" + 0.006*"say" + 0.006*"know" + 0.006*"joke" + '
  '0.006*"dad" + 0.006*"hear" + 0.005*"na" + 0.005*"girl" + 0.005*"would"'),
 (1,
  '0.013*"says" + 0.013*"bar" + 0.011*"call" + 0.010*"two" + 0.010*"walks" + '
  '0.007*"man" + 0.006*"say" + 0.006*"walk" + 0.006*"cross" + '
  '0.006*"difference"'),
 (2,
  '0.020*"call" + 0.020*"like" + 0.012*"man" + 0.010*"black" + 0.008*"people" '
  '+ 0.007*"dog" + 0.006*"say" + 0.005*"know" + 0.005*"white" + 0.005*"woman"'),
 (3,
  '0.012*"like" + 0.010*"take" + 0.009*"people" + 0.007*"many" + 0.007*"know" '
  '+ 0.007*"time" + 0.006*"would" + 0.005*"knock" + 0.005*"think" + '
  '0.005*"never"')]
```

Figure 4. Topics and their dimensions

*For a specific and interactive visualisation of the topic modelling results, refer to **this webpage***

Cluster 0 seems to contain those utterances joking about wives, women, and family life in general. Cluster 1 is indicating the presence of very formulaic jokes, such as those mentioned before. Cluster 2 could contain the most "racial" aspect of jokes as indicated by the importance given to words such as "black", "white" and "people". Cluster 3 is the vaguest one, where jokes that don't specifically belong to any other group would end up in. Both cluster 0 and cluster 2 are susceptible to contain the somewhat darker jokes, and will probably influence the model most, since it has been observed that GPT2 tends to reinforce racial and gender related prejudices and stereotypes [9]. As a last step in this exploratory analysis, I used the *VADER lexicon*, via the NLTK library, to analyse the sentiment scores of the utterances. To each sentence, we assign a compound score, which is given by the sum of its negative, positive, and neutral score, then normalized to fall between the range of -1(very negative) and 1(very positive). If the c.s > 0 we assign the positive label, if c.s. == 0 we assign "neutral", we assign "negative" otherwise.

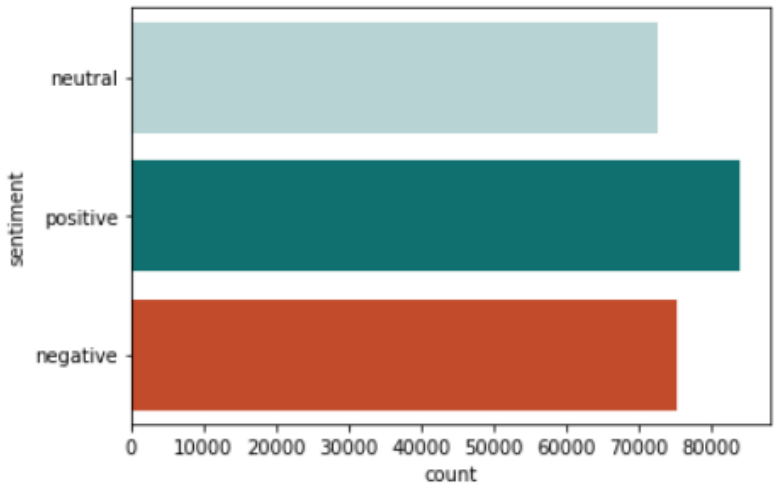

Figure 5. sentiment scores and distribution of sentiment in the dataset

Lastly, this final chart indicates that the dataset is well balanced in terms of its sentiment, being composed of an alike number of positive, negative and neutral jokes. Finally, the dataset was split into a training set and a validation set, using Scikit-Learn library. The ratio between the training and validation set was 80/20, resulting in a training set of circa 185.000 utterances and a validation set of 45.000. The datasets were saved in two separated *.csv* files and loaded using the HuggingFace Dataset library.

4 Fine tuning GPT-2

GPT2 was used via the HuggingFace library, which allows to access various language models, to freely use them and to re-share them with the community for improved interoperability and performances. The HuggingFace hub gives access to each aspect of the *model*, the *tokenizer*, the *pre-trained hyperparameters* and *saved weights*. At first, the datasets were passed through GPT2's own tokenizer, which split each sentence into tokens, where each token roughly corresponds to a single word, or of word particles such as the case of "-ed" or "-ing", that are tokenized with a wildcard sign to indicate that they are part of a specific preceding word (deemed => "deem" + "#ed"). GPT2 also requires to be fed same length utterances, which is almost never the case with such kind of fine-tuning tasks, and thus a padding mechanism must be set into place. Padding adds a specific padding token to guarantee that shorter sequences have the same length as either the largest sequence in a batch or the model's maximum length: in this case, I right-padded the sentences with the standard <|endofsentence|> token. For each row in the final tokenized datasets, the tokenizer returned a vector of two features: ['input_ids', 'attention_mask'], while the column containing the original text was removed, since GPT2 only accepts tensors as input. At this point, after having established access to the chosen model, it was time to start with the fine-tuning. After several attempts, I landed on the hyperparameters settings shown in *[snippet 1]*, that resulted in the best performance overall. The model was initially set with a *residual drop-out* of 0.5, as this usually helps avoid overfitting the data. Residual drop-out allows to randomly remove a section of the network's weights, forcing it to learn to adapt each time. Each training phase took a good amount of time, and since the free version of Google Colab was used, this meant that it has been necessary to stop training and resume from a saved checkpoint, with the clear implication that the hardware made available by the service was not stable during the whole process. The final training session took a total of approximately 8 hours, split into two separate 4 hours sessions. The *batch size* was 8, as default, but the training was sped up by using the *gradient accumulation steps* parameter, which allows to update the network weights only after a fixed number of steps (in this case, 3) instead of after each iteration.

```
training_args = TrainingArguments(
                                        output_dir = '/…/checkpointFinal',
                                        overwrite_output_dir = True,
                                        logging_first_step = True,
                                        num_train_epochs = 5,
                                        evaluation_strategy = 'steps,
                                        per_device_train_batch_size = 8,
                                        per_device_eval_batch_size = 8,
                                        warmup_steps = 200,
                                        save_steps = 1000,
                                        eval_steps = 500,
                                        gradient_accumulation_steps = 3,
                                        weight_decay = 0.3,
                                        learning_rate = 5e-5
                                )
```

Snippet 1. Training Arguments

The chart below shows the training performances with this hyperparameter setting: the loss curve initially decreased, then stabilizing by the end of the training session. On top of the chart, the reader can see the perplexity score achieved by the model.

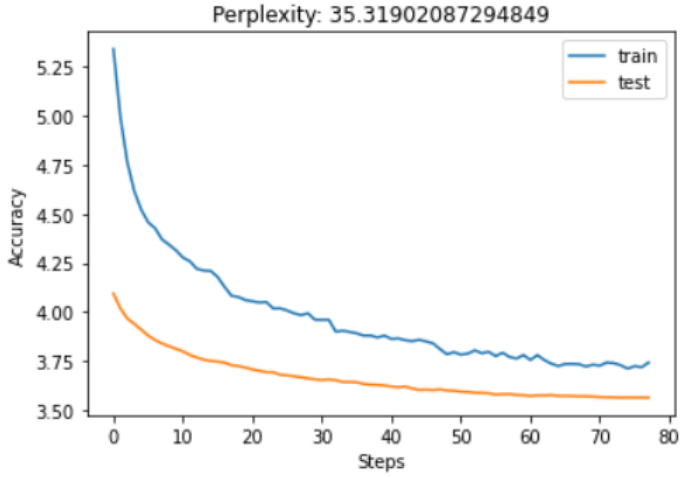

Figure 6. Loss curve

5 Results and Evaluation

Evaluating NLG works can be a difficult problem: there are little to no reference measure specific to the task, since it is hard to find a measure that could assign a score to a completely new generate sentence, since we have nothing really to compare it to, as it is the case with translation or classification tasks. In the literature, some propose to use the BLEU metrics [8], originally conceived for evaluating translations, which is a precision focus metric that calculates n-gram overlapping between the generated text and a reference text, one we wish our model would be able to produce. However, using BLEU is often misleading, simply because this approach is most suitable for translation, since n-gram similarities are indicative of a good translation, but not so much of good sample generation. The spectrum of generable text is very wide, but some sort of comparison among reference and generated text is still a good way to measure, even qualitatively, the performance of the model. Another useful measure for evaluating NLG model efficacy is *Perplexity* [7], which is a measure of the probability for a sentence to be produced by the model. In the field of information theory, perplexity refers to the power of probability distribution to assign probabilities to a sample. In this case, what we want is to control perplexity values on the test set and choose the probability model which assigns high probability to model/generate test set sentences. When the model is performing at its dumbest, the perplexity score corresponds to the size of the vocabulary. A key thing to notice is that the perplexity score does not informs on model's accuracy, it only informs on the model *confidence* in predicting the next word in a sentence. A low perplexity score means that the model is more confident in its choices, which then often correlates with good real-word performance.

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

Figure 7. Perplexity formula

As anticipated in the training loss performance chart above, the perplexity score for the final model was of 35.3. This perplexity value is informing that the model is far from dumb, but faces a certain

level of uncertainty, likely given by the fact that the dataset is highly varied, covering different topics and using a multitude of different words. If the dataset were stricter, for example by containing a fixed linguistic shape such as in poetry, or more topically connotated, such as a dataset of political speeches, for instance, this value could have possibly been lower. As mentioned in the dataset analysis section, while the dataset is well balanced and representative, it is also true that one can joke about so many things, and that word-level variation is expected to be quite high, justifying the model's perplexity score. An NLG model can also be evaluated on its performance scores alone, by assessing whether the training and evaluation loss keep descending during training, meaning that the model is successful at minimizing the error step after step, or in simpler words, that it is adapting its generation capacities to the utterances. Besides considering quantitative measures, NLG tasks are often evaluated qualitatively [8], by visually assessing whether the generated text is coherent and clear, if it "sounds good". To carry out this subjective assessment as rigorously as possible, I generated some jokes switching the generation parameters (temperature, max_new_tokens, repetition penalty) one by one, and then evaluating which one gave the best result in terms of general grammatical correctness, sense-making and how funny it could be considered. *Temperature*, especially, was a key parameter in determining the goodness of the results. In sequence generating models, temperature is a hyperparameter which controls the *randomness* of the model predictions: the lower the temperature, the greedier the model will be, choosing words with higher probability, which usually results in more coherent and correct utterances. If the temperature is high, the model will take some more "liberties" and select words with a wider spectrum of probabilities, which usually resolves into more grammatical mistakes, but also in more "creative" sentences. A trade-off between correctness and creativity is usually preferred, and the default value for temperature is typically 1. When conducting my experiments, I set the values to 0.7, 1.0, 1.3 and 1.6. What I noticed is that the smallest value seemed to perform better at generating somewhat funny jokes, while higher values resulted either in nonsensical phrases and were often prone to use offensive language and not being very funny either. The final parameter settings I landed on can be seen in the code snippet below, together with some examples of the generated jokes.

```
jokes = jokes_gen( "any_prompt",
                   max_new_tokens = 30,
                   temperature = 0.7,
                   num_return_sequences = 500,
                   repetition_penalty = 2.5,
                   do_sample = True
                   )
```

Snippet 2. Generation parameters

Here are some utterances generated by the model with different parameters, with some comments on the process behind their evaluation: first, I considered grammatical correctness, then sense-making and then I evaluated the humoristic expedient.

Generated Text	**Temperature**	**Evaluation**
People like to ask: "What are the symptoms of *Alzheimer's*?" and I reply, *'I can't remember'*	0.7	Correct, makes sense, quite funny. The relation [Alzheimer = not remembering] is correctly exploited to produce the unexpected turnover of the joke
'I have a problem with *procrastination*.....it makes me *lazy*.'	0.7	Correct, makes sense, funny. Redundancy is often used as humorous expedient. [procrastination = being lazy]

'People hate when I say "*I'm gay*"...They think it's nice *to be in the closet*, but it is all about *being straight*. Not sure'	1.0	Correct, makes (kind of) sense, quite funny. Once again, the humour is in the redundancy [being in the closet = faking to be straight = being gay].
'She said she was going to be an *organ donor* when she *dies*. Now, this way no one has any left, why should we keep hearing about her *funeral*?'	1.3	Correct, makes (kind of) sense, not very funny. The relation [death→ organ donor → funeral] is quite clear but not properly exploited
'The president decided to move his campaign headquarters after *Trump* becomes The President, a new slogan the next year - How did he get rid of all those dead *immigrants*.'	1.6	Correct, makes little to no sense, kind of funny, but offensive language or sensitive topic, also not really a joke. Could be an effort of relating [Trump → immigrants].

Table 1. Generated text at different temperature settings

In a final effort to evaluate my work, I generated a new dataset, composed only of generated jokes (with the parameters setting reported above) and tried to apply the same techniques that I used in the exploratory analysis of the training dataset, for comparing the results. I generated 1000 utterances for each of the 10 textual prompts I created, which were informed by the initial analysis. The prompts were: *"A guy", "The president", "My spouse"* [1], *"People", "The difference", "My boss", "My friend", "Why", "She", "I have a problem"*. The result was a dataset of 10.000 jokes, much smaller than the original dataset, but still large enough to allow a sensible comparison. The two datasets appear to have some similarities and some key differences: the charts of the top30 words are quite similar, as well as the topic modelling results, but there is a difference when it comes to sentiment scores. The model tends to generate more polarized utterances, preferring either positive or negative sentences, with a low rate of neutral scores.

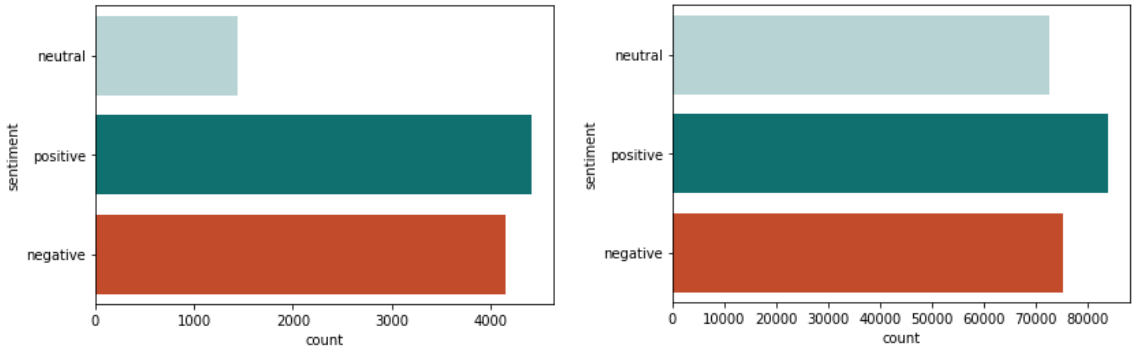

Table 2. comparison between sentiment scores distributions

This difference can be motivated by the fact that, as mentioned, GPT2 tends to enforce polarization in language, since it has been trained mainly on web scraped data, which is knowingly highly polarized, in both negative and positive way. For what concerns the topics comparison, the generated model seems to be partially coherent with the original dataset, when analysed with the same settings (k=4) it resulted into topics that were similarly connotated, as shown in the following picture. Interestingly enough, the clusters appear to be similarly collocated in space, meaning that among-topic distances have been respected during learning, and that there are some similarities in the

[1] Originally, the prompt was "my wife", since in the exploratory analysis the word emerged as fairly common. However, this prompt seemed to generate the ugliest and most sexist utterances, so I preferred to "tone it down" by prompting a more general "my spouse", which partially mitigated the effect, at least for what I was able to see in the experimenting phase. This effect was noticed, although a bit less, with the prompt "The difference", which tends to generate racist sentences. However, since this effect was not as pervasive as the other, I decided to continue to use this prompt.

clusters' internal composition. A detailed and interactive comparison of the topics can be found at [this link.](#) What follows is a screenshotted version of such visualisation.

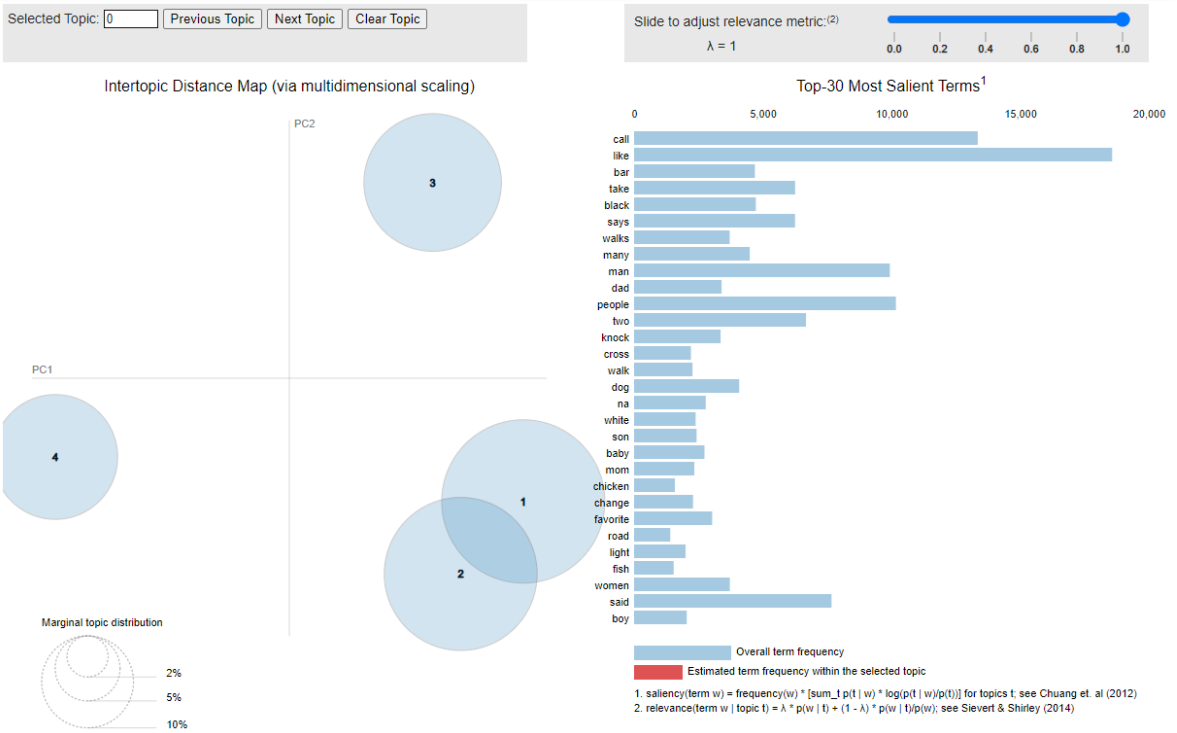

Figure 8. Topic Modelling visualisation of the Training dataset

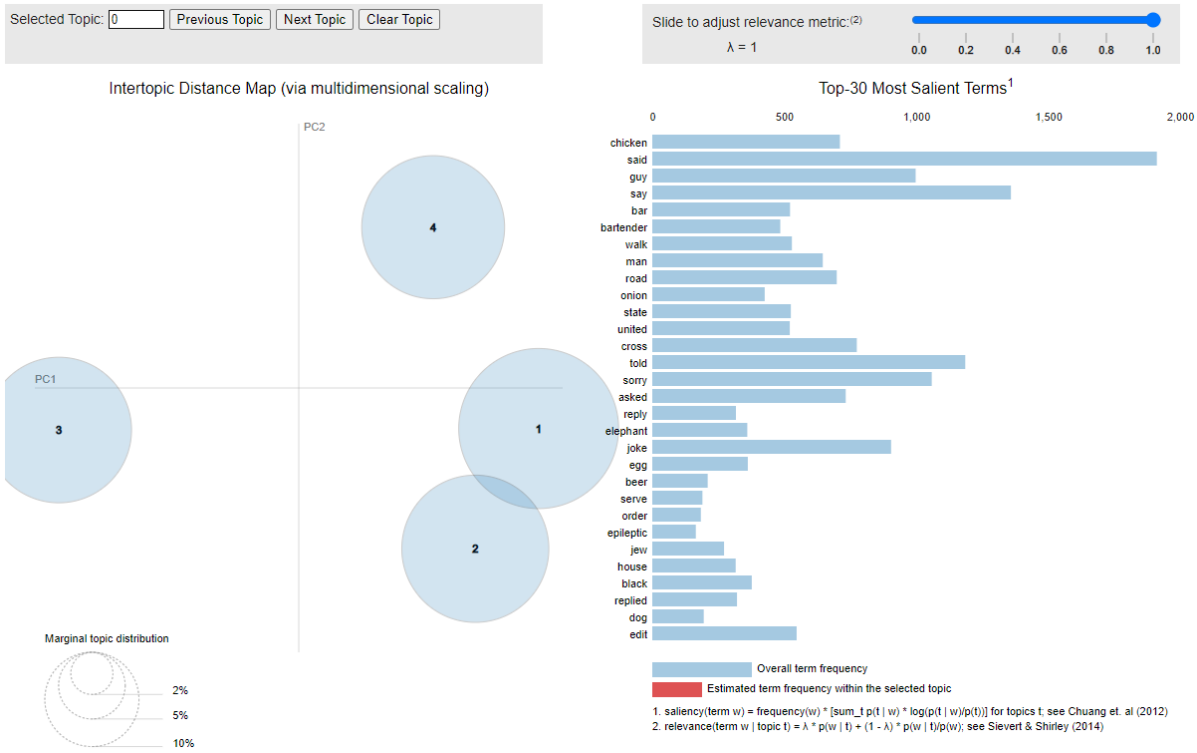

Figure 9. Topic Modelling visualization Generated dataset

6 Conclusion

This experiment was an exploration of GPT2 humour-generation capabilities, with a special interest in studying whether transfer learning could get an intuition of the linguistic mechanisms underlying short jokes. The model was able to produce grammatically correct sentences, that mostly made sense and that were sometimes capturing funny relationships between concepts. The model also proved capable of getting some of the intrinsically linguistic mechanisms of humour, such as redundancy, but failed to capture others, such as elements of surprise, exaggeration, or satire-like subtleties. Most of all, the model quickly caught the concepts that were most represented inside the training dataset and was able to use such relations among frequent words to produce new utterances. As expected, the model was also quick to catch all the sensitive topics and biases underlying the training dataset, so much so that, during the generation process, high temperature values settings generated highly offensive language, which is not reported in the paper, but can be easily imagined+. Low temperature and controlled prompts (for example “my spouse” instead of “my wife”), partially mitigated the sexist, racist and generally biased language, albeit not completely eliminating it. A good part of this project was the evaluation of the model, since I was particularly interested in finding a good way to assess the goodness of the results both quantitatively and qualitatively. Since the problem of NLG task evaluation is somewhat blurred, I evaluated my work from a plethora of perspectives, which resulted, in my opinion, in a good understanding of the model’s strengths and weaknesses. Some future development of this work could involve a more language-specific dataset, which would possibly allow the model to be able to capture more of the strictly linguistic components of humour, besides its topics. Nevertheless, this work proved that experimenting with such articulated and context dependent variety of language can be a challenging and interesting effort in the field of NLG, which calls for future development.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser. “Attention Is All You Need”, Dec 2017 https://arxiv.org/abs/1706.03762v5.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever. “Language Models are Unsupervised Multitask Learners”. 2019. https://openai.com/blog/better-language-models/

[3] O. Weller, N. Fulda, K. Seppi. Can Humour Prediction Dataset be used for Humor Generation? Humorous Headline Generation via Style Transfer. Proceedings of the Second Workshop on Figurative Language Processing, pages 186–191. July 9, 2020. 2020 Association for Computational Linguistics. 10.18653/v1/2020.figlang-1.25. https://aclanthology.org/2020.figlang-1.25

[4] S. Attardo. Humour in Language. Mar. 2017. DOI: 10.1093/acrefore/9780199384655.013.342. https://www.semanticscholar.org/paper/Humor-in-Language-Attardo/62ad5d561eb9df0a73ec316bc44b2bf4526c1863

[5] D. Wilson, D. Sperber. “On verbal irony”. Reprinted from Lingua 87, pp. 53-76, (1992) with permission from Elsevier.

[6] R. Alghamadi, K. Alfalqui. “A Survey of Topic Modeling in Text Mining” in (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 6, No. 1, 2015.

[7] Surge AI. “Evaluating Language Models: An introduction to Perplexity in NLP”. Dec. 15, 2021.https://surge-ai.medium.com/evaluating-language-models-an-introduction-to-perplexity-in-nlp-f6019f7fb914

[8] A. B. Sai, A. K. Mohankumar, M. M. Khapra. “A Survey of Evaluation Metrics Used for NLG Systems. 2020. https://doi.org/10.1145/0000001.0000001

[9] K. Ngo. “Bias in Large Language Models: GPT-2 as a case Study”. Feb 19, 2021 Berkeley.edu. https://blogs.ischool.berkeley.edu/w231/2021/02/24/bias-in-large-language-models-gpt-2-as-a-case-study/

[10] J. Alammar. “The Illustrated Transformer”. June 27, 2018. http://jalammar.github.io/illustrated-transformer/

[11] T. Raha, I. S. Upadhyay, R. Mamidi, V. Varma. “IIITH at SemEval-2021 Task 7: Leveraging transformer-based humourous and offensive text detection architectures using lexical and hurtlex features and task adaptive pretraining”. 2021.