University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Tomasz Garbus
Student no. 370795

Controlled Machine Text Generation of Football Articles

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Przemysław Chojecki
Institute of Mathematics, Polish Academy of Sciences

Co-supervisor:
dr hab. Dominik Ślęzak, prof. UW
University of Warsaw

Warsaw, June 2020

Abstract

Among other benefits of the rapid development in deep learning, language modelling (LM) systems have excelled at producing relatively long text samples that are (almost) indistinguishable from human-written text. This work categorizes conditional text generation systems into three paradigms: generation with placeholders, prompted generation, adversarial/reinforcement learning and provides an overview of each paradigm along with experiments – both machine- and human-judged. Example corpora of football news are used to discuss how a fast, domain-specific named entity recognition (NER) system can be built without much manual labour for English and Polish. The NER module is evaluated on manually labelled texts in both languages. It is then used not only to build fine-tuning sets for the language model, but also to aid its generation procedure, resulting in samples more compliant with provided control codes. Finally, a simple tool EDGAR for prompt-driven generation is presented. Two demos are made for the reader to experiment with and compare the proposed solutions with simply finetuned GPT-2 model.

Keywords

text generation, transformer, control code, named entity recognition, reinforcement learning

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

Computing methodologies-Machine learning
Computing methodologies-Neural networks
Computing methodologies-Natural language generation
Computing methodologies-Information extraction

Tytuł pracy w języku polskim

Maszynowe generowanie wiadomości o piłce nożnej

Contents

Chapter 1

Introduction

In the recent years, massive progress has been made in the domain of machine generation of (mostly visual) content. Human faces generated by Generative Adversarial Networks (GAN, [1])[1] are indistinguishable from real faces. So-called deepfakes, videos of a particular person's face manipulated to perform facial expressions and utter arbitrary words are not only a source of entertainment[2], but also are recognized as one of the major threats of the Internet era[3], with both technological giants and academic environments[4] dedicating resources to counteract them. This surge of constantly improving image generation models is commonly attributed to GANs, although other approaches such as Variational Autoencoders (VAE) have been gaining popularity too.

The revolutionary achievements in visual content generation haven't been directly mirrored in the domain of text generation, due to the nature of textual data – it's discrete, sequential and often very high dimensional, rendering it unsuitable for traditional GAN settings. Great advancements in text generations have been made regardless. Currently, the state-of-the-art results in language modelling are obtained by massive deep learning models based on Transformer ([2]) architecture. The GPT-2 ([3]) model trained by OpenAI has presented the world with machine generated text of almost human-like quality, at the same time starting a public debate on possible implications of powerful language models. Since then, the tendency to upscale the Transformer architecture has started – Nvidia has trained a model with 8.3B[5] parameters ([4]) (whereas OpenAI's model had 1.5B parameters) and Microsoft has built a model with 17B parameters[6]. Most recently (as of June 2020), OpenAI has upscaled its model to 175B parameters ([5]), setting new state-of-the-art results on a number of tasks, such as language modelling, question answering and machine translation.

When approaching a text generation task, clearly the objective is for the text to be indistinguishable from something a human would write. This can be measured using different metrics which aim to simulate the human test as well as possible. This objective has either already been achieved by the massive Transformer models or is quite likely to be achieved in the future. Another goal is to gain control over generated text. This is of course not so clearly defined, since the domain of the corpus and the desired use are what determine what aspects of text one wishes to control. The object of control may be the sentiment, the style, the conveyed information and facts and perhaps several other properties.

This work uses a toy domain of football news to explore the approaches to controllable text generation. While even such restricted domain leaves us with a corpus surprisingly hard to translate to a structured

[1]https://www.thispersondoesnotexist.com/

[2]https://www.youtube.com/watch?v=8OJnkJqkyio

[3]https://www.youtube.com/watch?v=gLoI9hAX9dw

[4] https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html

[5] 1B = 10^9, not to be confused with Polish word *bilion* denoting 10^{12}.

[6] https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

knowledge base (and even more so to do it, at least to some extent, automatically instead of manual labour), the experiments focus on controlling the topic of generated article, where by the topic we understand entities appearing in the article, such as footballers or teams. Several approaches are compared in terms of required computational power, text quality, cohesion and accordance with the provided control codes. An out-of-the-box solution EDGAR is proposed at the end, which can be applied to any domain and corpus, although it only allows control over text's topics, not its logical qualities such as presented facts or events.

Chapter 2

Preliminaries

2.1. Language modelling

A language model is simply defined as a learned probability distribution over texts from a given language, however this definition does not express what is expected from a language model to be practically applicable. Usually the probability distribution is decomposed as a chain of conditional probabilities ([6]):

$$p(x) = \prod_{i=1}^{n} p(s_i | s_1, ..., s_{i-1}) \tag{2.1}$$

where $x = s_1 s_2 ... s_n$ and $s_1, s_2, ...$ are tokens (words, interpunction, acronyms, numbers, EOF markers etc.) belonging to the language. Typically, a natural language generation (NLG) system will, at each step of generation (usually one step means one token, but there exist exceptions) estimate probabilities over the dictionary and output one token. Strategies for selecting the token include drawing from the estimated distribution, random choice from top k tokens and a simple argmax (such generator is then deterministic). Note that language modelling formulation in 2.1 generalizes well to specific tasks; estimating and maximizing conditional probability $p(output|input)$ is the common denominator of machine translation (MT) (*input*: original, *output*: translation), image captioning (*input*: image, *output*: description)[1], text summarization (*input*: original text, *output*: summary) and question answering (*input*: question, *output* : answer).

A simple implementation of a language model is a recurrent neural network (RNN), trained with the Teacher Forcing ([7]) algorithm. At the training time, a sequence is fed to the RNN; at each step the network predicts the next output but regardless of its prediction, a correct token is provided as the next input. While vanilla RNNs are known to suffer from *vanishing gradients* and thus fail to learn long-term dependencies ([8]), variants incorporating a hidden state, such as LSTM ([9]) or GRU ([10]) are used in practice. Architectures with hidden-to-hidden connections are trained with both BPTT (backpropagation through time) and Teacher Forcing ([11, p. 378]).

Such RNN with hidden state can be then applied in an encoder-decoder architecture ([10]) to tasks like MT or conditional generation. The encoder reads the input sequence $(s_1, s_2, .., s_n)$ and produces a *context vector* $z = [z_1, z_2, ..., z_d]^T$ of fixed dimensionality d. The context vector is then fed into the decoder RNN at each timestep of decoding.

[12] show that the fixed-length context vector is the bottleneck in improving the performance of the model, an issue that becomes more severe as the sequence length increases. They improve this architecture

[1] Of course in both examples of MT and image captioning, the input and output languages are different. An image encoding format is not a natural language, but otherwise the problem statement and even evaluation methods are similar as in MT (both are often evaluated against a set of reference answers). Also approaches to those tasks are different at implementation level, at least at the phase of processing the *input*.

by replacing a single context vector per sequence with one context vector per timestep. The encoder is a bi-directional LSTM, producing *forward hidden states* $\overrightarrow{h}_1, \overrightarrow{h}_2, ..., \overrightarrow{h}_{T_x}$ in the forward run and *backward hidden states* $\overleftarrow{h}_1, \overleftarrow{h}_2, ..., \overleftarrow{h}_{T_x}$ in the backward run. The context vector at timestep t is then a concatenation $h_t = [\overrightarrow{h}_t^\top; \overleftarrow{h}_t^\top]^\top$, containing the summaries of both the preceding and following words. The decoder, at each timestep t, computes the current context vector c_t as the weighted average $c_t = \sum_{j=1}^{T_x} \alpha_{tj} h_t$, where α_{tj} are softmaxed scores obtained by all context vectors $h_1, h_2, ..., h_{T_x}$. A score for each context vector is assigned by a feedforward neural network taking as an input current hidden state of the decoder and the context vector.

2.1.1. Transformer

On top of the improvements made in NLG thanks to the attention mechanism a Transformer architecture ([2]) was created – a stack of 6 encoders and 6 decoders. Both encoders and decoders implement a novel *Self-Attention* mechanism, chosen based on three desiderata: a) total computational complexity per layer, b) ability to parallelize the computation, c) the path length between long-range dependencies in the network. Table 2.1 shows that Self-Attention is more attractive than recurrent or convolutional layers both in terms of parallel computation and explicit dependence of tokens at distant positions.

Table 2.1: Table by [2]. n is the sequence length, c is the representation dimension, k is the convolution kernel size.

Layer type	**Complexity per layer**	**Sequential operations**	**Maximum path length**
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(n)$	$O(\log_k(n))$

In Self-Attention layer, three weight matrices are learned: W_Q, W_K, W_V, producing, in result of multiplication, vectors for each input token intuitively thought of as (respectively): *query*, *key*, *value*. Query vector for word i-th q_i is obtained by multiplying x_i by W_Q, where x_i is the representation (e.g. embedding or output from previous layer) of i-th word. Similarly key vector k_i and value vector v_i are obtained. When processing self-attention for i-th word, all other words are scored by computing a scaled dot product of their key vectors with q_i. The scores are then softmaxed and a weighted sum of all value vectors is computed:[2]

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{2.2}$$

where Q, K, V are the matrices query, key and value matrices and d_k is dimensionality of keys. Scaling down by $\sqrt{d_k}$ serves to prevent softmax gradient saturation effect. Transformer extends this self-attention mechanism further to a *Multi-Headed Attention*, meaning that several ($h = 8$ in original work) attention heads run in parallel.

Each encoder layer consists of a self-attention sublayer and a position-wise feedforward network. Each decoder layer consists of a self-attention sublayer, followed by an "encoder-decoder" attention and a feedforward network. In the "encoder-decoder" layers, the queries come from previous decoder layers whereas keys and values – from the last encoder layer.

Other architectural choices in the Transformer model include:

[2]Equation (1) in [2]

a) positional encoding, used to inject the information about word's position in a sequence into the input embedding

b) residual connections around each sublayer inside each encoder and decoder layer

c) regularization: dropout with rate added 0.1 to the output each sublayer and to the sums of embeddings and positional encodings; label smoothing

All experiments with Transformer architecture in this work use pretrained GPT-2 model from `https://github.com/minimaxir/gpt-2-simple`, more specifically GPT-2-medium, with 355M trainable parameters, consisting of 24 stacked decoders, with model dimensionality set to 1024 (dimensionality of base Transformer model is 512).

Interestingly, there exists a tool detecting text generated by GPT-2 ([13]), which, given a text sample, estimates probability of each token using a small (117M) GPT-2 model. Authors show that human-written texts have more randomness (less likely tokens).

2.2. Desiderata

Our goals, expressed in natural language, are twofold:

(1) We would like to obtain control over the message conveyed by generated text. This calls for selecting some property of text that we wish to control and a programmatic way of measuring compliance of generated samples with the selected attribute. For the sake of larger part of this study, texts will be parametrized by sets of occurring entities. It is realistic (although non-trivial) to automatically evaluate samples' compliance with such specifications. The variety of entity names occurring in the football domain also guarantees that the problem is more difficult than multi-class conditional generation.

(2) We wish to achieve (1) without sacrificing quality, originality and other properties of text that make it appealling to a human reader.

For those objectives to be any useful, we must formulate them in a computable way.

2.2.1. F-score

A good candidate for measuring (1) is F-score.

Consider a text T_X conditioned on a set of entities X. Let $Y = \text{NER}(T_X)$, a set of entities occurring in the generated text T_X, where NER is a Named Entity Recognition module.

Precision measure is defined as $\frac{X \cap Y}{Y}$. In other words, it is the ratio of entities mentioned in T_X that were also supplied to the generator in set X, among all entities mentioned in T_X.

Recall is the ratio of those entities in X that were mentioned in T_X. It is defined as $\frac{X \cap Y}{X}$.

F-score is the harmonic mean of the two:

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \tag{2.3}$$

2.2.2. BLEU metric

Bilingual Evaluation Understudy (BLEU) metric, despite being designed for machine translation problem, is also the most widespread metric among other text generation problems, be it conditional (text2text) or unconditional (text2self) generation task.

BLEU is computed for a generated text $T_X = t_1 t_2 ... t_n$ against a set of references $\mathcal{R}_X = R_X^1, R_X^2, ..., R_X^k$, where X is the conditioning value. For instance, in an English-Polish machine translation task, X is a sentence in English, T_X is a Polish translation generated by the model and $R_X^1, R_X^2, ..., R_X^k$ are reference translations.

Let $\#_t(W)$ denote number of occurrences of token t in word W and D be the dictionary – set of all tokens. The BLEU metric is then defined as:

$$\mathrm{BLEU}(T_X; \mathcal{R}_X) = \sum_{t \in D, \#_t(X) > 0} \min\left(1; \frac{\max_{1 \leq i \leq k} \#_t(R_i)}{\#_t(X)}\right) \tag{2.4}$$

In other words, we compute precision of output tokens. For every token, we add 1 to the result if it occurs in some reference text and 0 otherwise. Then we divide the result by the text length $|T_X| = n$. There is only one modification: in order to prevent the generator from repeating the same token over and over, each distinct token can only contribute to the final score as many times as it maximally occurs in some reference text.

Usually, a brevity penalty is also added to BLEU, to prevent the generator from outputting very short unrealistic texts. In this work, such penalty is not used in BLEU-evaluated experiments, since all generated texts are of fixed length.

For the reference sets, random subsets of the training data are selected (in a similar manner that one would extract a development or validation set). There is a property of BLEU:

$$\mathcal{R} \subseteq \mathcal{R}' \Rightarrow \mathrm{BLEU}(T; \mathcal{R}) \leq \mathrm{BLEU}(T; \mathcal{R}') \tag{2.5}$$

that makes it attractive for text2self and general language modelling tasks.

BLEU measure can be also generalized to bigrams, trigrams, and n-grams for any $n \in \mathbb{N}_+$. We will denote such metric BLEU-n.

Comparison of LSTM and GPT-2 on BBC Sport

In order to verify that BLEU is indeed a meaningful metric, it was computed in first 5 variants for two generated corpora: one generated by LSTM, another one by GPT-2. As expected, the Transformer model scored much better on the BLEU metric, especially for longer n-grams. Both models were trained on the same corpus of articles scraped from BBC Sport.

Table 2.2: Evaluation of corpora generated by LSTM and by GPT-2 using BLEU metrics.

metric	**LSTM**	**GPT-2**
BLEU-1	0.992	0.981
BLEU-2	0.909	0.899
BLEU-3	0.617	0.752
BLEU-4	0.327	0.555
BLEU-5	0.172	0.374

2.3. Three paradigms of controlled generation

[14] categorizes training techniques for training RNNLMs (Recurrent Neural Network Language Models) into three paradigms: supervised learning, reinforcement learning and adversarial training. While all three techniques are presented and experimented with in this work, here the division is different. The paradigms

described below are not approaches to training a language model but techniques of gaining control over generated content.

- **Generation with placeholders** is an approach based on replacing all occurrences of certain types of tokens, e.g. entities with placeholders during training and filling those gaps with desired values at generation time.
- **Generation from prompt** is conditional generation from more or less complex "prompts" – from simple control codes such as "positive" or "negative" to sets of entities that should be mentioned in text.
- **Adversarial training and reinforcement learning**. While both adversarial training and RL can be used to train a language model with placeholders or conditioned on a prompt, they are, in theory, more powerful than traditional MLE methods and deserve a separate category. Most such models are first pretrained using simple MLE and then start to optimize another goal. In practice, adversarial and RL architectures for NLG haven't yet matched the results achieved by large Transformer-based models. A probable explanation is that all adversarial architectures proposed so far are computationally heavy and cannot be parallelized.

Chapter 3

Datasets

3.1. Scraper

All data sources were scraped using a dedicated Scraper repository[1]. The `Scraper` class crawls the source website in a BFS order. This procedure is resumable thanks to caching the scraper's state on disk on keyboard interrupt and every 100 articles. To start scraping a data source user needs to subclass `ScraperConfig` and implement its abstract methods:

- `home` – Starting page, provided as an absolute URL.
- `domain` – The domain of the visited portal. It will be used to create absolute URLs from relative URLs by concatenation.
- `state_cache_fname` – Filename of the `ScraperState` cache. It will be placed in `cache` folder.
- `should_follow_link` – Given a link, determines whether scraper should queue it for visit.
- `extract_article` – If the visited web page contains an article, extracts its content and builds an `Article` object. Otherwise returns `None`.
- (optional) `fetch_page` – Function fetching the web page given its URL. For some websites it might be necessary to override it (e.g. to handle Polish characters correctly or sleep 1 second to not get banned).

Such implementation of the Scraper allows for quickly setting up a very simple config for websites like BBC Sport (see `configs/bbcsport.py`[2]), free from paywalls, not bloated with ads or JavaScript. On the other hand, more complex logic can be implemented for slower, script-driven websites (see `configs/orzeczenia.py`[3]). Each Article object is a tuple $(title, date, content)$, where *date* format is not specified, it may also be null. Each article is then stored in the `output_directory/`*date* path. Table 3.1 lists datasets downloaded with purpose of using in this work.

[1] https://github.com/tomaszgarbus/Scraper
[2] https://github.com/tomaszgarbus/Scraper/blob/master/configs/bbcsport.py
[3] https://github.com/tomaszgarbus/Scraper/blob/master/configs/orzeczenia.py

Table 3.1: Comparison of scraped datasets

Dataset	**Language**	**Articles**	**Only football**
bbc.co.uk/sport/football	en	92824	y
football.co.uk	en	2565	y
pilkanozna.pl	pl	121382	y
sport.pl/pilka	pl	57105	y
fakt.pl/sport/pilka-nozna	pl	250	y
sport.interia.pl/	pl	312122	n
pap.pl	pl	9322	n

Since BBC Sport is the only large English scraped resource with football articles, all further experiments (unless conducted on Polish language) use it as the corpus.

3.2. Analysis

Common entity sets

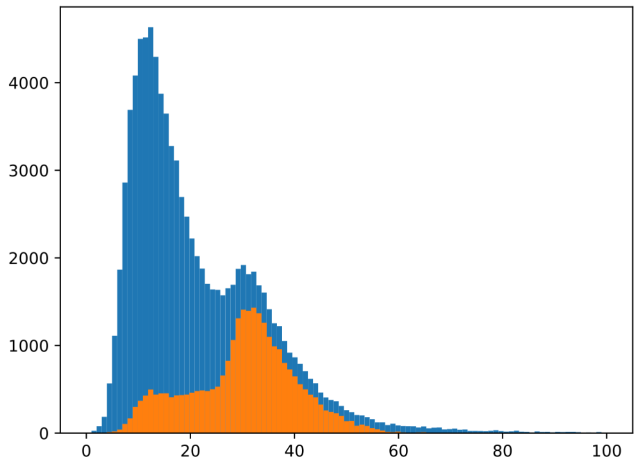
The implementation of named entity recognition (NER) heuristics, described in chapter 4, allowed to build a simple representation of each article, as a set of all named entities of certain types occurring in them. Presented below at Figure 3.1a the list of most common entity subsets in BBC Sport dataset of cardinalities 1-6 computed with Apriori ([15]) algorithm (min support: 0.01, min confidence: 0.2).

Entity set cardinalities distribution

BBC Sport dataset contains 92824 articles, 21407 out of which are match reports. That is, each match report's title matches the pattern `team1 \d:\d team2`, where `\d` is a digit. The article itself describes the match, sometimes an overview of most important events is provided at the bottom. Figure 3.1b shows the distribution of found entity set cardinalities in all BBC Sport articles and only in match reports.

Size	Itemset	Occurrences
1	Premier League	24641
	Championship	16398
	England	15216
	Manchester United F.C.	12560
	Chelsea F.C.	10860
2	Manchester United F.C.,Premier League	7307
	England,Premier League	6908
	Chelsea F.C.,Premier League	6717
	Manchester City F.C.,Premier League	6225
	Liverpool F.C.,Premier League	6102
3	Chelsea F.C.,Manchester United F.C.,Premier League	3284
	Manchester City F.C.,Manchester United F.C.,Premier League	3203
	Chelsea F.C.,Manchester City F.C.,Premier League	3178
	Liverpool F.C.,Manchester United F.C.,Premier League	2983
	Arsenal F.C.,Chelsea F.C.,Premier League	2927
4	Chelsea F.C.,Manchester City F.C.,Manchester United F.C.,Premier League	2005
	Chelsea F.C.,Liverpool F.C.,Manchester City F.C.,Premier League	1883
	Arsenal F.C.,Chelsea F.C.,Manchester City F.C.,Premier League	1859
	Chelsea F.C.,Liverpool F.C.,Manchester United F.C.,Premier League	1830
	Arsenal F.C.,Chelsea F.C.,Manchester United F.C.,Premier League	1819
5	Arsenal F.C.,Chelsea F.C.,Manchester City F.C.,Manchester United F.C.,Premier League	1372
	Chelsea F.C.,Liverpool F.C.,Manchester City F.C.,Manchester United F.C.,Premier League	1351
	Arsenal F.C.,Chelsea F.C.,Liverpool F.C.,Manchester City F.C.,Premier League	1282
	Arsenal F.C.,Liverpool F.C.,Manchester City F.C.,Manchester United F.C.,Premier League	1254
	Arsenal F.C.,Chelsea F.C.,Liverpool F.C.,Manchester United F.C.,Premier League	1249
6	Arsenal F.C.,Chelsea F.C.,Liverpool F.C.,Manchester City F.C.,Manchester United F.C.,Premier League	1037

(a) Most common itemsets of different sizes in BBC Sport dataset.



(b) Histogram of cardinalities of entity sets detected in all BBC Sport articles. Heights of blue bars correspond to all articles, orange ones only to match reports. Match reports statistically contain more distinct entities since all goals, red cards, substitutions and other significant events are listed at the bottom of the article.

Figure 3.1: BBC Sport entity sets analysis.

Table 3.2: Most common itemsets of different sizes in Sport.pl dataset. Note that the occurrences of *Borussia Monchengladbach* are most likely results of wrong entity resoluion. It's common in Polish articles about Robert Lewandowski that the name *Borussia* occurs before the full name *Borussia Dortmund*. Another examples of wrong resolution found within top 50 common entities are *Real Valladolid* (from *Real*) and *Biggleswade F.C.* (*FC* is listed as its nickname on Wikipedia).

Size	Itemset	Occurrences
1	Polska	66223
	Bayern Monachium	36878
	Robert Lewandowski	35732
	Barcelona	32351
	Real Madryt	29694
	Premier League	27889
	Niemcy	26618
	Legia Warszawa	26098
	Bundesliga	24391
	Juventus	21346
2	Bayern Monachium,Robert Lewandowski	20795
	Polska,Robert Lewandowski	16954
	Bayern Monachium,Bundesliga	15556
	Legia Warszawa,Polska	14953
	Bundesliga,Robert Lewandowski	13932
	Bayern Monachium,Niemcy	13888
	Niemcy,Robert Lewandowski	11588
	Bundesliga,Niemcy	10872
	Niemcy,Polska	10531
	Barcelona,Real Madryt	10401
3	Bayern Monachium,Bundesliga,Robert Lewandowski	11217
	Bayern Monachium,Niemcy,Robert Lewandowski	8814
	Bayern Monachium,Polska,Robert Lewandowski	8120
	Bayern Monachium,Bundesliga,Niemcy	7789
	Bundesliga,Niemcy,Robert Lewandowski	6734
	Bayern Monachium,Borussia Dortmund,Robert Lewandowski	5952
	Bundesliga,Polska,Robert Lewandowski	5779
	Bayern Monachium,Borussia Dortmund,Bundesliga	5584
	Bayern Monachium,Borussia Monchengladbach,Robert Lewandowski	5329
	Borussia Dortmund,Bundesliga,Robert Lewandowski	5192
4	Bayern Monachium,Bundesliga,Niemcy,Robert Lewandowski	5618
	Bayern Monachium,Bundesliga,Polska,Robert Lewandowski	4473
	Bayern Monachium,Borussia Dortmund,Bundesliga,Robert Lewandowski	4217
	Bayern Monachium,Borussia Monchengladbach,Bundesliga,Robert Lewandowski	3689
	Bayern Monachium,Niemcy,Polska,Robert Lewandowski	3425
	Bayern Monachium,Borussia Dortmund,Niemcy,Robert Lewandowski	3076
	Bayern Monachium,Borussia Dortmund,Bundesliga,Niemcy	3075
	Bayern Monachium,Borussia Monchengladbach,Niemcy,Robert Lewandowski	2944
	Bundesliga,Niemcy,Polska,Robert Lewandowski	2889
5	Bayern Monachium,Borussia Dortmund,Bundesliga,Niemcy,Robert Lewandowski	2288
	Bayern Monachium,Bundesliga,Niemcy,Polska,Robert Lewandowski	2271
	Bayern Monachium,Borussia Monchengladbach,Bundesliga,Niemcy,Robert Lewandowski	2123

Chapter 4

Named Entity Recognition

In order to build a language model conditioned on a set of entities, one must be capable of identifying and categorizing entities in real data.

There exist out-of-the box neural NER models (for instance spaCy's EntityRecognizer[1]), however applying them to a specific task requires defining custom entity types and training or fine-tuning the model on an annotated training set. Since all data in the experiments consists of scraped articles from various sources, it does not come with any sort of annotations. Labelling a training set would be a lengthy manual process[2], therefore a more database-driven solution was implemented.

In the experiments, 6 entity categories were selected:

- `player` – currently active footballer, former players do not belong in this category.
- `team` – a football club. It can be ambiguous in an article reporting on a match between two national teams whether e.g. *Germany* should be interpreted as a team or a country.
- `manager` – a club/national team manager/coach/assisting coach. Club owners, financial directors etc. do not belong in this category.
- `league` – any sort of club or international competition; may refer to either a recurring competition (*World Cup*) or a particular instance (*Euro 2016*).
- `country`
- `stadium`

It's easy to observe how `player` and `manager` categories are problematic – in an article from 2005 Zinedine Zidane would be mentioned as a player, in 2020 he would appear as a manager of Real Madrid. An ideal language understanding system should condition the evaluation of provided text on its date. Even more challenges arise if we expect the language understanding system to capture relations between entities, such as `player`'s belonging to a `team`. This and other relations (`team-stadium`, `team-league`, `manager-team`) are also temporally conditioned and should be evaluated as such. Since this work focuses on conditional generation, using NER only as a tool to build a training set, rather than building a comprehensive football knowledge base, a simple approach is taken and entity recognition does not depend on article's date.

[1] https://spacy.io/api/entityrecognizer

[2] It would be interesting to see how many manually labelled articles are actually needed to finetune a NER model to obtain satisfying results.

4.1. Entities database

Entities of the 6 selected types were scraped from `footballsquads.co.uk`, RapidAPI[3], `enfa.co.uk`[4], English Wikipedia pages about England[5], Italy[6], Northern Ireland[7], Scotland[8] and Spain[9] as well as Polish Wikipedia pages about Bundesliga[10], Ekstraklasa[11], Italy[12], French Ligue 1[13] and Spanish La Liga[14]. Numbers of entities fetched from each source are listed in Figure 4.1.

	player	team	manager	league	country	stadium
footballsquads.co.uk	297535	1426	7688	0	0	0
RapidAPI	52379	9703	5124	1319	0	6230
Wikipedia: England	7954	940	926	0	0	856
Wikipedia: Italy	1050	19	17	0	0	16
Wikipedia: Nor. Ireland	607	30	30	0	0	27
Wikipedia: Scotland	1174	41	38	0	0	41
Wikipedia: Spain	747	20	20	0	0	19
enfa.co.uk	3555	0	114	0	0	0
manual/other	1	12	2	61	197	438

(a) English entities

	player	team	manager	league	country	stadium
footballsquads.co.uk	309980	1426	7654	0	0	0
RapidAPI	52379	9703	5124	1319	0	6230
Wikipedia: Bundesliga	667	21	14	0	0	18
Wikipedia: Ekstraklasa	178	60	4	0	0	5
Wikipedia: England	8070	1041	1024	0	0	921
Wikipedia: France	1325	36	29	0	0	31
Wikipedia: Italy	1439	44	28	0	0	28
Wikipedia: La Liga	840	24	21	0	0	21
manual/other	1	2	0	12	198	1

(b) Polish entities

Figure 4.1: Comparison of sources scraped for entities

English variants and filters

In order to increase the chances of detecting an entity in an article, multiple variants of each entity were created. To avoid false matches, filters were applied to remove variants with very high false positive count.

- person (player or manager) name variants – if a person's name consists of two words (e.g. *Firstname Lastname*), a variant *Firstname [Lastname]* is added to database. If it consists of two or more words and the last word is at least 5 letters long, 3 variants are added: *Lastname*, *F Lastname*, *F. Lastname*, where *F* is the first letter of first word. If the name starts with *Sir*, a variant without this title is added. If the person's nationality is known and it is among several chosen countries, a variant such as *the Englishman*, *the Scot*, *the Spaniard* etc. is added.

- team name variants – if the team name contains one of the words *United*, *City*, *Town*, *Athletic*, *Rangers*, *Rovers*, then it's probable the team will be referenced by just this single word (usually in

[3]https://rapidapi.com/api-sports/api/api-football

[4]the `enfa.co.uk` website contains much more data than downloaded, only several teams were hand-picked not to breach the website's terms of use

[5]https://en.wikipedia.org/wiki/List_of_football_clubs_in_England

[6]https://en.wikipedia.org/wiki/2019%E2%80%9320_Serie_A

[7]https://en.wikipedia.org/wiki/List_of_association_football_clubs_in_Northern_Ireland

[8]https://en.wikipedia.org/wiki/List_of_Scottish_Professional_Football_League_clubs

[9]https://en.wikipedia.org/wiki/La_Liga

[10]https://pl.wikipedia.org/wiki/Bundesliga_niemiecka_w_pi%C5%82ce_no%C5%BCnej

[11]https://pl.wikipedia.org/wiki/Ekstraklasa_w_pi%C5%82ce_no%C5%BCnej_(2018/2019)

[12]https://pl.wikipedia.org/wiki/Serie_A

[13]https://pl.wikipedia.org/wiki/Ligue_1

[14]https://pl.wikipedia.org/wiki/Primera_Divisi%C3%B3n

a subsequent mention in an article, e.g. *City* will appear after *Manchester City*). Similarily, the team may be mentioned by the remaining part of the name, e.g. *Manchester* as a shorthand for *Manchester United*. Adequate variants were added to the database. For teams which have *&* in their names, it was replaced with *and* and vice versa. Suffixes such as *F.C.* or *A.F.C.* where added or removed in different variants. Many Wikipedia pages about foootball clubs have a *nicknames* field in their infoboxes. Those nicknames were also scraped and added to database (sometimes several variants of nicknames were created). If the manager's name is known (for instance it was available at the Wikipedia's infobox), then a variant *[manager_name]'s side* is added (for instance a variant of *Tottenham Hotspur* would be *Jose Mourinho's side*).

- stadium name variants – from each name, the following combinations were stored: *[stadium_name]*, *[stadium_name] Arena*, *[stadium_name] Stadium*, *[stadium_name] Park*.

- person name filters – some person name variants had to be filtered out, because they were matched too often in articles, in a context that did not refer to those persons. For instance, it's clear to see why footballer *Tom Corner* was matched too often or players whose last name is the same as a country/month/day of the week. Some erroneous variants were a result of a Wikipedia page where player's names and nationalities columns were swapped in a table and had to be filtered out.

```
['Sunday', 'England', 'Northern Ireland', 'Ireland', 'Portugal', 'N Ireland', 'N. Ireland',
 'France', 'Denmark', 'Spain', 'Sweden', 'Ecuador', 'Netherlands', 'DR Congo', 'D. Congo',
'Congo', 'Mali', 'Croatia', 'Albania', 'Wales', 'London', 'Manager', 'Scotland', 'Nigeria',
 'Jamaica', 'Ghana', 'Zimbabwe', 'August', 'October', '-', 'English', 'Corner', 'March',
     'February', 'German', 'Madrid', 'French', 'Irish', 'until', 'Friday', 'Under']
```

Figure 4.2: The full list of filtered out player names.

Polish variants and filters

Recognizing Polish entity names was a much more difficult task due to declination and shortage of sources to scrape (compared to vast choice of APIs with English names). While many (foreign) names are the same in Polish articles (*Manchester United*), some may have different Polish variants (*Juventus Turyn* – not only is the city name translated, but it's frequently appended to the club name, which is not so common in English sources). Polish person names usually come in two variants – formal (*Jakub*) and informal (*Kuba*)[15]. As for

[15] There was an idea to create name variants by replacing formal names with informal ones, but unfortunately no good name database was found.

declination, there exist some good libraries, such as pystempel[16] or Morfeusz[17], however performance was an issue when trying to process hundreds of thousands of names. Instead, some hand-picked heuristics for matching and replacing word suffixes were used to create multiple variants of each name. As one can notice after reading about the search algorithm below, appending incorrect declinations to the database does not harm the accuracy – it was more important to include the correct declination in the list of variants than it was to not include an erroneous one. Some words were blacklisted from declination, for example *the*, *of*, *City*, *United*. Others were declensed using the patterns listed in Table 4.1. Some word suffix replacement patterns were collected by manual inspection of the corpus, others were found in a book *Nauka o języku dla polonistów*([16]). In a name consisting of multiple words, all words were declensed into several variants and all combinations were added to database – since this procedure is exponential, names with 4 or more words were not declensed at all (but their shorter variants were).

Table 4.1: Declination heuristics – word suffix replacement patterns.

suffix	**replacements**
iga	idze, igi, ige
ski	skiego, skiemu, skim
cki	ckiego, ckiemu, ckim
y	ego, emu, ym
wel	wla, wlowi
ia	ii,ie
ek	ku, ka, kiem, kowi
a	i, ie, y
ko	ki, ce
ka	ki, ce, kiej
lli	llim, lliego, lliemu
n	nie
t	cie
en	nu, nowi, nem
es	su, sa, sem
ec	cu, ca, cowi, cem
ja	i
r	rze
(any consonant)	+owi, +em, +a, +u, +e
(any vowel)	+'owi, +'a
Pogon	Pogoni, Pogonia
Zaglebie	Zaglebia, Zaglebiem, Zaglebiu

The variant creation and filtering rules were similar for Polish entities as they were for English. Several additional filters were added, such as *Canal* (there is a popular TV provider *Canal* in Poland) or *Europa* (which was very often wrongly matched with an *Europa F.C.* club from Gibraltar).

4.2. Search algorithm

There are hundreds of thousands of entities in the database, and millions of different variants in total. A naive implementation of pattern matching is thus too computationally expensive even for very short articles. Instead, a search procedure aided by a trie tree was used.

First, a trie tree is built from the database. The value stored in each leaf is a list of base variants that the string can be resolved to. For instance, `trie['the Blues'] = [('Chelsea F.C.', 'team'), ('Everton F.C', 'team'), ('Burnham F.C.', 'team'), ...]`. A simplified version of the algorithm collecting entities mentioned in an article is presented below:

[16]https://pypi.org/project/pystempel/
[17]http://morfeusz.sgjp.pl/

Algorithm 1 Collect mentioned entities

```
 1: $mentions \leftarrow \emptyset$
 2: $i \leftarrow 0$
 3: while $i < |A|$ do
 4:     $s \leftarrow \epsilon$
 5:     $j \leftarrow i$
 6:     while $A[i:j]$ is a path in $trie$ do
 7:         if $A[i:j]$ ends in a leaf and $A[j+1]$ is not a letter then
 8:             $s \leftarrow A[i:j]$
 9:         end if
10:         $j \leftarrow j+1$
11:     end while
12:     if $s \neq \epsilon$ then
13:         $value \leftarrow \epsilon$
14:         $category \leftarrow \epsilon$
15:         for $v, c$ in $trie[s]$ do
16:             $value \leftarrow v$
17:             $category \leftarrow c$
18:             if $(v, c)$ in $mentions$ then
19:                 break
20:             end if
21:         end for
22:         $mentions \leftarrow mentions \cup \{(value, category)\}$
23:         $i \leftarrow i + |s|$
24:     else
25:         $i \leftarrow i+1$
26:     end if
27: end while
28: return $mentions$
```

The actual implementation is not position-agnostic and takes a callable as a parameter allowing for different handling of entities – building prompts for generation, annotating a dataset, replacing names with placeholders etc. The check in line 18 increases the quality of entity resolution; for instance if the substring *City* is matched and *Norwhich City* was found earlier, then *City* will be resolved to *Norwhich City* and not *Manchester City*.

```
::Andy Halliday extends Middlesbrough deal until 2015::
@1363258836
url:  https://www.bbc.com/sport/football/21783091
Middlesbrough utility player Andy Halliday has signed a two-year contract
extension with the Championship club.  and the former Livingston player has
agreed terms.  Since arriving at the Riverside (Riverside Stadium), under
Gordon Strachan’s management in August 2010, Halliday (Andy Halliday) has
made 32 appearances.  Although he arrived at Middlesbrough as a winger, the
Scot (Andy Halliday) has played at full-back this season.
```

(a) English sample

```
 ::Zinedine Zidane znalazl nastepce Marcelo w Realu Madryt (Real Madryt).
40 milionow euro!  Pilka nozna - Sport.pl::
@2019-05-02 11:52
url:  [...]
Wedlug gazety _Marca (Lavinius Marca)_ Zinedine Zidane_poprosil wladze
klubu sciagniecie do klubu_lewego obroncy Olympique (Olympique Lillois) Lyon
(Olympique Lyonnais), 23-letniego_Ferlanda Mendy (Ferland Mendy)&aposego._W
ostatnich tygodniach w mediach pojawialy sie rozne nazwiska, jak:  Junior
Firpo (Betis (Real Betis)), Alex Grimaldo (Benfica) i David Alaba (Bayern
(Bayern Monachium)), ale to Mendy (Ferland Mendy) najbardziej_przekonuje
trenera Krolewskich.  REKLAMA
```

(b) Polish sample

Figure 4.3: A demo of the entity recognition heuristics – example articles with entities of different categories highlighted with different colors. The values in parentheses are base variants that the mentions were resolved to.

4.3. Evaluation

A test set was prepared to evaluate the NER heuristics. 100 articles from `http://www.sport.pl/pilka` and 100 articles from `https://www.bbc.co.uk/sport/football` chosen by a random number generator were manually annotated with the 6 categories of entities. Figure 4.4 showcases two samples from the test set. During evaluation, each annotated article was translated to a set of tuples $(start, end, category)$[18], where $(start, end)$ is a character-level range in the article. The `Highlighter` class encapsulating the entity recognition logic was asked to produce such tuples from unlabelled article. The two resulting sets – ground truths and predictions – were then compared and F-score, precision and recall metrics were computed. The end metrics were averaged over all articles. Table 4.2 shows the evaluation results on both test sets.

Table 4.2: NER evaluation

language	#entities	F-score	precision	recall
en	4586	0.797617	0.782374	0.829196
pl	2862	0.657026	0.731524	0.614048

There are certain limitations to this evaluation methodology. Entities cannot overlap, so a phrase *former England international*, referring to current Manchester United's first coach (as of 15.03.2020) Michael Carrick, can be labelled as `manager`, but *England* can be labelled as `country`. Moreover, the test set is not annotated with base variants of detected entities (a base variant of *former England international* here would be *Michael Carrick*). It is thus impossible to say whether the `Highlighter` class under test resolves the entities correctly (modulo category).

```
 ::{team:Cliftonville} sign ex-{team:Ballymena} goalkeeper {player:Ryan
Brown}::
@1307556749
url:  https://www.bbc.com/sport/football/13705050
{team:Cliftonville} have signed former {team:Bangor}, {team:Larne} and
{team:Ballymena United} goalkeeper {player:Ryan Brown}.  {player:Brown}
spent last season with {team:Ballymena United} and becomes manager
{manager:Tommy Breslin}'s fifth signing of the summer.
"I'm looking forward to getting started," said former junior international
keeper {player:Brown}.  "{team:Cliftonville} have shown great consistency
over the last few years and have been challenging at the top end of
the table." Other {team:Reds} signing in recent days includes former
{team:Newry} and {team:Portadown} defender {player:John Convery} and
ex-{team:Glentoran} midfielder {player:Peter Steele}.
```

(a) English sample

```
 ::{team:BATE} podbija {league:Ligę Mistrzów} Piłka nożna - Sport.pl::
@2008-10-01 12:54
{team:Juventus} w {league:Lidze Mistrzów}:  {team:Marynarze} w histerii i
stracone punkty
Dla {team:BATE} mecz z {team:Włochami} był ósmym w tej edycji {league:LM}.
{team:Białorusini} zaczynali walkę o fazę grupową, gdy {player:Alessandro
del Piero} i {player:Pavel Nedved} wylegiwali się na karaibskich plażach.
W I rundzie {team:BATE} pokonało islandzkie {team:Valur} (2:0 i 1:0), w
drugiej {team:Anderlecht Bruksela} (2:1 i 2:2), a w trzeciej {team:Lewski
Sofia} (1:0 i 0:0).  {team:Białorusini} są zespołem z najmniejszym
współczynnikiem (1,760) UEFA w fazie grupowej {league:LM}.  Poprzednim była
{team:Artmedia Petržałka} ze współczynnikiem niemal trzykrotnie większym
(4,850).
{team:Słowacy}, gdy grali w fazie grupowej {league:LM} mieli budżet ok.
1,5 mln euro.  Tyle samo ma dziś {team:BATE}.  Większy ma nawet {team:Odra
Wodzisław}.  Albo raczej miała, bo po sukcesach {team:Białorusini} chcą go
powiększyć do 4 mln euro.
```

(b) Polish sample

Figure 4.4: Samples from the NER test set

[18] This is the same format as spaCy NER component uses.

Readers are encouraged to reproduce the NER evaluation experiment in a demo Colab notebook attached to EDGAR repository[19]. It is advised to do it after reading at least chapters 6 and 8. Please note that the results obtained in the Colab notebook are a bit lower than reported above. That is because, in the reported experiment, the piority of categories was carefully selected – if there exist two entities with the same name (not necessarily same base name, but the sets of name variants overlap) but different categories, the following prioritization of categories applied:

1. Country
2. Team
3. Player
4. Manager
5. Stadium
6. League

whereas the order of entities in the example `Highlighter` configuration included in EDGAR repository is random (League, Country, Stadium, Manager, Player, Team). Readers are encouraged, as an exercise, to experiment with the order of entities to improve the final F-score.

[19] `https://colab.research.google.com/drive/1PURy7p7jscwRxjLBnqNaKzxV6hyHPhJe?usp=sharing`

Chapter 5

Paradigm I: Generation with placeholders

5.1. Related work

More dated NLG systems are often based on hand-coded templates or grammars. Facts are extracted from the knowledge base (*what to say*) and a rule-based text generator constructs a sentence (*how to say it*). [17] describes an (at the time of publication) partially implemented computer program that reports on stock market. [18] present a YAG (Yet Another Generator) – an efficient, general-purpose with expressive formal representation language. [19] build a system that generates weather forecasts from numerical data by careful engineering of rules and paying attention to the choice of words; for instance, their study includes statistical research on what is considered *evening* or *late evening* by larger part of population.

[20] use *Boxer* computer program ([21]) to build Discourse Representation Structures ([22]). They use domain general entity tags such as COMPANY, DATE, DATE to transform training sentences to templates. Finally, they use statistical methods to select the best template for a given conceptual unit and fill in the gaps with entities listed in the unit.

[23] propose an approach to neural natural language generation by using neural hidden semi-markov model (HSMM) to learn latent, discrete templates. Their model learns to successfully extract templates from E2E Dataset ([24]) and to semantically split the sequence into multi-word segments.

5.2. Discussion

Languages with declension

English is particularly convenient for template-driven generation. Usually, a sequence from the domain-specific corpus can be split into mutliple interleaving segments: *part of template*, `entity:type`, *part of template*, `entity:type`, *part of template*, ... since the types of entities are often limited to

a) proper nouns

b) nouns in general

c) numerical data

d) verbs from a limited dictionary, (e.g. *increased/decreased* token conditioned on the sign of a certain numerical value)

For the sake of providing concrete data to the user, all we need to know about a template, is the set of entities we need to fill, together with their types. The rest of tokens in the template is only important to the end user, but not to a program providing data to the text generator.

Consider an example from the football domain:

Cristiano Ronaldo's pass was intercepted by Michał Pazdan.

In some knowledge-based system we could create a template:

{player:pass_ origin}'s pass was intercepted by {player:pass_ intercept}.

Consider the same sentence in Polish:

Podanie Cristiano Ronaldo przejęte przez Michała Pazdana.

Clearly, a template:

Podanie {player:pass_ origin} zostało przejęte przez {player:pass_ intercept}.

will not render a grammatically correct sentence when filled with entity names in nominative case. On the other hand, the following template:

{player:pass_ origin} podaje, jednak {player:pass_ intercept} przejął to podanie.

conveys the same fact, yet expects entities in different case.

Enriching templates with synonyms

A tempting techinque for enriching the set of hand-written templates may be replacing some words with synonyms. [25] have used this approach to data augmentation when training a CNN sentence classifier. Concretely, they have used a Thesaurus from LibreOffice, which in turn was obtained from WordNet[1]. The number of words chosen to replace, as well as the index of the chosen synonym (synonyms in WordNet are ordered by decreasing relevance) were determined by a geometric distribution with parameter 0.5. In general, CharCNN variants trained on such augmented data have obtained better results.

Note that applying this method to texts presented to the end user must be done with much more caution and possibly hand-picked filters of words to avoid.

Consider a toy example, a fragment of a sequence from Stanford Sentiment Treebank[2]:

(...) goes on and on to the point of nausea.

Let's review different parametrizations of the geometric distribution determining the number of replaced words (ρ) and the index of selected synonym (ϕ). For each of four pairs of values ρ and ϕ, 5 samples were generated from Thesaurus:

[1] The Thesaurus data is available at `https://raw.githubusercontent.com/LibreOffice/dictionaries/master/en/th_en_US_v2.dat`

[2] https://nlp.stanford.edu/sentiment/treebank.html

(...) goes on and on to the location of disgust.
(...) goes on and on to the convexity of disgust.
(...) goes on and on to the factor of disgust.
(...) goes on and on to the item of sickness.
(...) goes on and on to the detail of disgust.

(a) $\rho = 0.1, \phi = 0.1$

(...) goes on and on to the component of sickness.
(...) goes on and on to the constituent of symptom.
(...) goes on and on to the component of sickness.
(...) goes on and on to the component of sickness.
(...) goes on and on to the component of sickness.

(b) $\rho = 0.1, \phi = 0.9$

(...) goes on and on to the location of nausea.
(...) goes on and on to the convexity of nausea.
(...) goes on and on to the point of disgust.
(...) goes on and on to the item of nausea.
(...) goes on and on to the detail of nausea.

(c) $\rho = 0.9, \phi = 0.1$

(...) goes on and on to the component of nausea.
(...) goes on and on to the constituent of nausea.
(...) goes on and on to the point of sickness.
(...) goes on and on to the component of nausea.
(...) goes on and on to the component of nausea.

(d) $\rho = 0.9, \phi = 0.9$

Using such data augmentation technique may be useful when building a classifier, especially if there are no pre-trained word embeddings. However, most of the samples presented above cannot be presented to a human user with an expectation that they will understand the message.

5.3. Template generation with GPT-2

A powerful language model such as GPT-2 may be a promising tool for generating domain-specific templates, as an alternative for laborious hand-engineering of templates. The language model must be first finetuned on a training set, where controlled variables are replaced with descriptive placeholders. The model then generates large number of texts with placeholders that are to be filled with concrete data, in this case – 6 types of entities.

Setting

GPT-2 medium (355M) model was finetuned on a subset of BBC Sport corpus containing only match reports (see Figure 3.1b). Each article in this sub-corpus has a normalized title format:

```
team1 \d-\d team2
```

where `\d` denotes a digit. Two modifications were made in the training (fine-tuning) set. First, in each article title, the substring `\d-\d` was replaced with one of the three: <, >, ==.[3] For instance, a title

```
Dundee United 0:3 Celtic
```

would become:

```
Dundee United < Celtic
```

Secondly, all detected entities were replaced with placeholders formatted as

```
{[category]:[id]}
```

For instance, the title

```
Dundee United < Celtic
```

[3] This modification was made because previous experiments have shown that prompting GPT-2-medium with exact match score is too hard of a task and the number of goals described in an article would rarely match the final result.

is replaced with:

```
{team:0} < {team:1}
```

Of course, subsequent occurrences of the same entity have the same `id`. GPT-2 model was finetuned for 16000 iterations (about 15-16 hours on a single core of Nvidia Tesla K80). 100 samples were then drawn from the model. Each sample had 300 tokens (shorter than majority of articles in the training set).

5.4. Human evaluation of text cohesion

Samples generated as described in subsection 5.3 were put under human reader evaluation. Two experiments were conducted:

- Longest cohesive prefix, conducted by one person seeking inconsistencies in generated samples.
- Online quiz, where a larger number of judges had to determine whether the presented article is real or not.

Later, in section 6.3 identical experiments were conducted with control-code-driven generation.

5.4.1. Experiment: Longest cohesive prefix

First, all samples were manually analysed[4] by one person and assigned one of the three statuses:

`NO` The text contained (anywhere) a contradiction with the match result stated in the title.

`OK,n:int` The generated text agreed with the title on the matter of who won the game, but there was some internal contradiction in the generated sample. For instance, the first sentence stated that the match ended in a goalless draw, but later a goal was reported. The number n means that first n sentences form a cohesive text, but the $n+1$-th sentence creates a contradiction with something stated earlier. A partial repetition of the previous sentence (a common issue for GPT-2) was also counted as "bad sentence".

`OK` The entire sample was cohesive, did not contain any contradictions or (partially) repeated sentences or other artifacts. Note that the sample didn't necessarily have to be a complete article, since generation was always trimmed at 300 tokens, not at `<|endoftext|>`.

In this experiment, the placeholders in generated articles were not populated.
Below are presented the aggregated results:

Figure 5.2: Longest cohesive prefix experiment results.

NO	OK	OK,1	OK,2	OK,3	OK,4	OK,5
25	0	0	36	11	11	11
OK,6	OK,7	OK,8	OK,9	OK,10	OK,11	OK,12
3	2	1	0	0	0	0

Most common inconsistencies in the generated samples were:

[4] Author has attempted building a classifier which, given a match report sans title, predicts the result of the described match, but results were poor and unpromising.

- contradictions regarding the match score
- unchronological order in reporting of events (one sentence reporting on a goal in last 8 minutes, the next one describing a goal scored in the first half).
- introduction of too many entities, for instance `{team:2}` (which clearly did not play in a match between teams `{team:0}` and `{team:1}`). This particular inconsistency may be a result of imperfect entity resolution during dataset labelling.

5.4.2. Experiment: Online quiz

A certain convenience in evaluating a powerful model such as GPT-2 lies in the fact that a very natural approach to human judgement methodology can be applied: the judge has to predict whether the text sample is real or generated.

An online quiz was implemented as a Django web application. The database contains 100 samples drawn from GPT-2 and 100 samples randomly chosen from BBC Sport sub-corpus of match reports. There are 10 questions in the quiz – randomly chosen samples, with two answers: *Real* and *Generated*, although website visitors were encouraged to take the test more than once, so that they can improve their sensitivity to common patterns in generated articles. After each question the person taking the quiz receives feedback (*[Wrong/Correct]! The text was [real/generated].*). Each sample is trimmed to a random number of sentences between 5 and 10. The titles follow the `team1 ? team2` format where `?` is an inequality sign. Placeholders in generated articles were populated by randomly sampling entity names from English leagues (with uniform distribution).[5] Displayed articles were decorated with a single, (usually) relevant image, obtained live from Google Images via Custom Search API, queried with the article's title.

Results were collected about 7 days after deploying the website. In the meantime, the quiz was shared with other students (from different faculties) and on machine learning and soccer subreddits.

In total, 593 votes were collected, giving almost 3 votes per text sample on average. Votes were well balanced – 278 votes said *REAL*, 315 said *GENERATED*. Figure 5.3 presents aggregated results.

[5] It would be interesting to see how the placeholder population strategy affects the text quality. Perhaps using the same probability distribution as in real corpus would make more realistic articles. Ideally, one should account for relationships between entities when filling the placeholders. For instance, when filling a text fragment `"{team:0} manager {manager:2}"`, the choice of team entity should determine the choice of manager entity. This would require building more complex placeholders, preserving this kind of relationships and was not explored in this work.

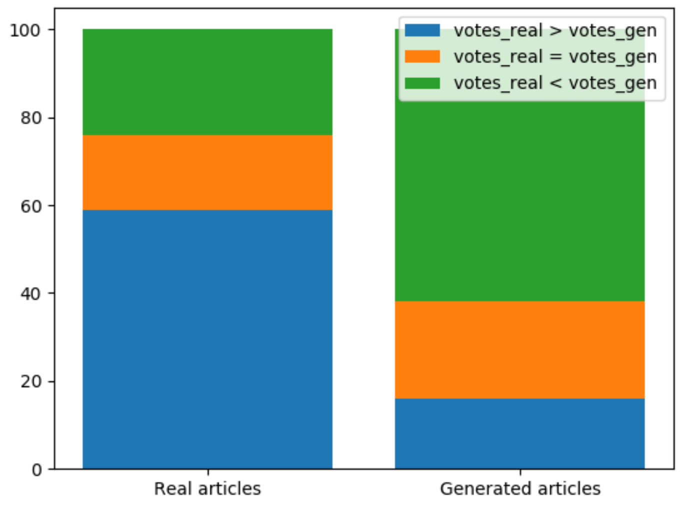
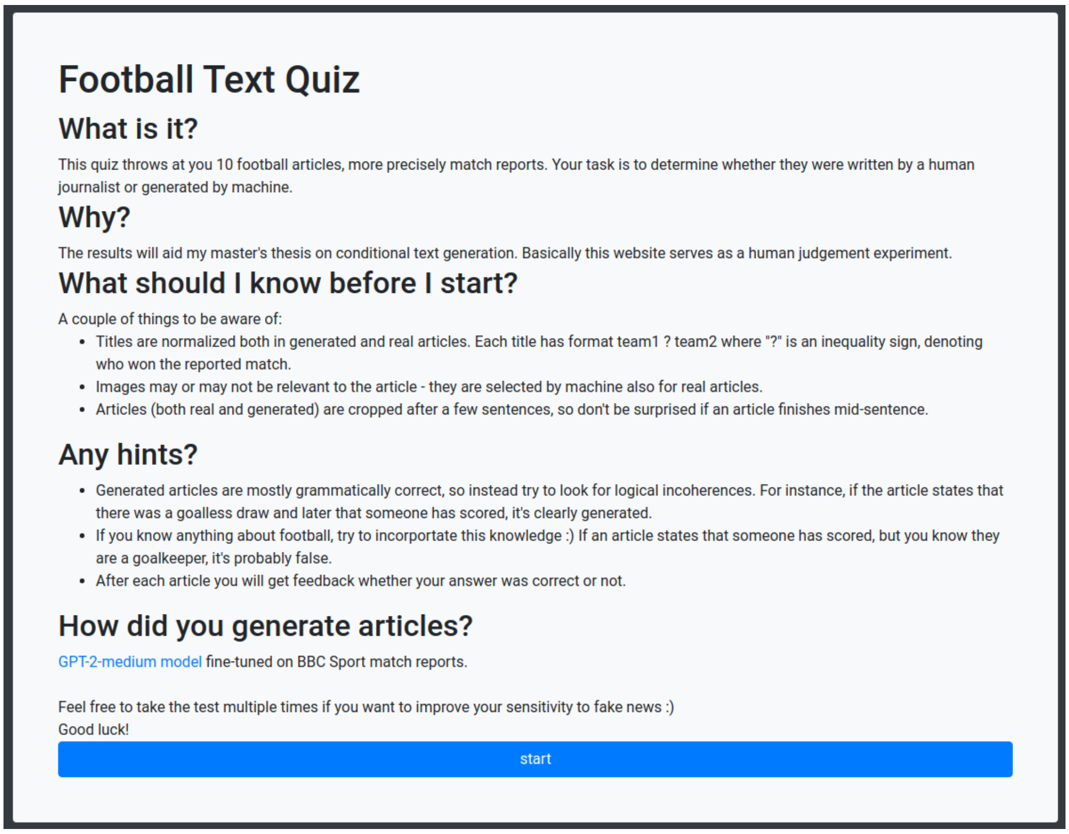


Figure 5.3: Among the human-written articles, 59 were correctly identified as real (i.e. more votes said *REAL* than *GENERATED*), 24 were falsely named generated and in case of remaining 17, there was a tie. 16 of the generated samples have fooled majority of voters, 62 were correctly identified and for the remaining 22, there was a tie.


Football Text Quiz

What is it?

This quiz throws at you 10 football articles, more precisely match reports. Your task is to determine whether they were written by a human journalist or generated by machine.

Why?

The results will aid my master's thesis on conditional text generation. Basically this website serves as a human judgement experiment.

What should I know before I start?

A couple of things to be aware of:

- Titles are normalized both in generated and real articles. Each title has format team1 ? team2 where "?" is an inequality sign, denoting who won the reported match.
- Images may or may not be relevant to the article - they are selected by machine also for real articles.
- Articles (both real and generated) are cropped after a few sentences, so don't be surprised if an article finishes mid-sentence.

Any hints?

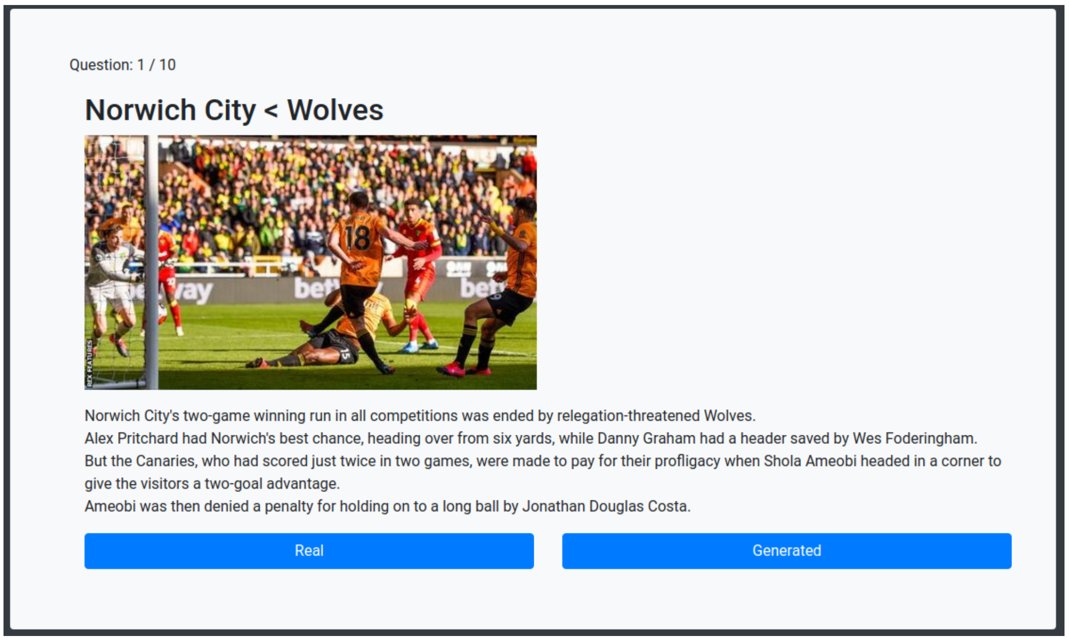
- Generated articles are mostly grammatically correct, so instead try to look for logical incoherences. For instance, if the article states that there was a goalless draw and later that someone has scored, it's clearly generated.
- If you know anything about football, try to incorportate this knowledge :) If an article states that someone has scored, but you know they are a goalkeeper, it's probably false.
- After each article you will get feedback whether your answer was correct or not.

How did you generate articles?

GPT-2-medium model fine-tuned on BBC Sport match reports.

Feel free to take the test multiple times if you want to improve your sensitivity to fake news :)
Good luck!

(a) Introduction and instructions


Question: 1 / 10

Norwich City < Wolves

Norwich City's two-game winning run in all competitions was ended by relegation-threatened Wolves.
Alex Pritchard had Norwich's best chance, heading over from six yards, while Danny Graham had a header saved by Wes Foderingham.
But the Canaries, who had scored just twice in two games, were made to pay for their profligacy when Shola Ameobi headed in a corner to give the visitors a two-goal advantage.
Ameobi was then denied a penalty for holding on to a long ball by Jonathan Douglas Costa.

(b) An example question

Figure 5.4: Screenshots from `https://text-quiz.herokuapp.com/`

Chapter 6

Paradigm II: Prompted generation

6.1. Related work

CTRL

[26] recall the intuitive factorization of the probability distribution approximated by a language model[1]:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{<i}) \tag{6.1}$$

and extend it with a control code:

$$p(x|c) = \prod_{i=1}^{n} p(x_i|x_{<i}, c) \tag{6.2}$$

Similarily, they modify the negative log-likelihood cost function:

$$\mathcal{L}(D) = -\sum_{k=1}^{|D|} \log p_\theta(x_i^k|x_{<i}^k) \tag{6.3}$$

where θ is the parametrization of language model and $D = \{x^1, ..., x^{|D|}\}$ is the dataset, to account for the control code:

$$\mathcal{L}(D) = -\sum_{k=1}^{|D|} \log p_\theta(x_i^k|x_{<i}^k, c^k) \tag{6.4}$$

On top of this intuition, that a language model can model a probability distribution conditioned not only on previous tokens but also on some control code, authors build CTRL – a 1.63 billion-parameter transformer architecture, trained on 140GB of text data, with control codes prepended to each text. Control codes include:

Table 6.1: An overview of control codes fed to the CTRL model.

Control code category	Description	Examples
Style by domain	Specify overall style by pointing to a source domain	`Wikipedia`, `Books`, `Reviews`
More complex codes	Detailed specification of output text	`Reviews Rating: 1.0`
Trigerring specific task	Related to specific tasks such as QA or MT	`Questions Q: What is the capital of India? A:`
Zero-shot code mixing	Mixing of cross-domain control codes that do not occur in training data	Mix of politics subreddit with a French prompt

CTRL achieves phenomenal results from a human perspective. Numerous samples presented in the paper are coherent over multiple lines, compatible with provided parametrization.

[1]Original equations from [26]

One could hypothesise that when it comes to controlled text generation, no generation-time supervision is required and that a traditional likelihood-estimation approach can be successfully generalized to prompts and control codes. CTRL is a strong point in case.

Plug and Play Language Models (PPLM)

[27] propose an alternative to costly domain-specific fine-tuning of a pretrained GPT-2 model. Instead, they attach to the output of the Transformer a discriminator (authors experiment with bag-of-words and linear classifiers), which steers the generation. Having defined a history matrix H_t as key-value pairs from the past: $H_t = [(K_t^{(1)}, V_t^{(1)}), ..., (K_t^{(l)}, V_t^{(l)})]$, where the upper index denotes the layer, and the lower index – timestep, authors summarize the transformer architecture as[2]:

$$o_{t+1}, H_{t+1} = \mathrm{LM}(x_t, H_t) \tag{6.5}$$

where o_{t+1} is a logit vector of vocabulary size, from which a single token is sampled with probability $softmax(W o_{t+1})$ and W is a linear transformation.

At each generation step t, the history H_t is shifted towards a combination of two gradients:

- firstly, history H_t is nudged toward higher output of the attached discriminator.
- secondly, to prevent the generator from sneaky adversarial practices of sacrificing text quality for the sake of fooling the discriminator, H_t is updated toward higher LL of unmodified LM $p(x)$.

Authors construct bag-of-words models for seven topics: Science, Military, Legal, Computers, Space, Politics, Religion. They have also built linear classifiers of text sentiment, distinguishes between two classes, Positive and Negative.

While the PPLM has performed similarily good as CTRL, despite having much less trainable parameters[3], it was also tested on very general control codes (sentiment, general topic etc.). It is difficult to predict how well it would perform under a more fine-grained discriminator.

6.2. Fine-tuning GPT-2

With a set of procedures for annotating an article with the occurring samples (vide chapter 4), it was possible to enhance the training set with relevant prompts.

6.2.1. Prompt format

All prompts (except section 6.3) followed the BNF specification below:

```
<category> ::= "player" | "stadium" | "manager" | "league" | "country" | "team"
<category_entities> ::= "" | <entity> "," <category_entities>
<category_line> :== <category> ":" <category_entities>
<prompt> ::= "" | <category_line> "\n" <prompt>
```

In practice, best results were achieved if several other assumptions were made:

- categories were listed in the same order in all prompts
- empty categories were also listed
- each category was listed exactly once (as opposed to separate line for each entity)

[2] Original equation from [27].
[3] The pretrained GPT-2 model is not updated in PPLM.

6.2.2. Generation procedure

What distinguishes prompted generation implemented in a traditional MLE framework from an adversarial or RL setting at the high level is an assumption that with enough data fed into the model, there is no need for a judge (of course here by judge we mean a function from sequences to real numbers rather than a human linguist) which explicitly evaluates the compliance of output sequence with the expectation (prompt) during training. Such judge can however be useful to select the best sample to be presented to the end user. Following this observation, we want the judge to assist the generation process by selecting the best path every n tokens rather than choosing a single end result. A generation procedure emerges from this intuition:

Algorithm 2 Generate sample

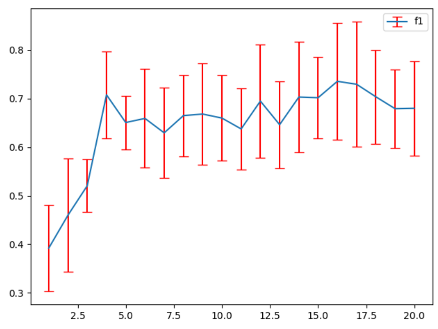
```
Input: $prompt, tokens\_at\_step, num\_steps, nsamples$
1: $entities \leftarrow$ extract_entities($prompt$)
2: $text \leftarrow prompt$
3: for _ in range($num\_steps$) do
4:     $variants \leftarrow$ gpt2.generate($tokens\_at\_step, nsamples$, prefix=$text$)
5:     $variants$.sort(key=$\lambda v$.f_score(extract_entities($v$), $entities$), reverse=$True$)
6:     $text \leftarrow$ concat($text, variants[0]$)
7: end for
8: return $text$
```

Note that this procedure closely resembles a special case of beam search, where only one path is explored at each step. Perhaps a beam search with higher beam width would yield better results, but it was not experimented with (and would be more computationally expensive).

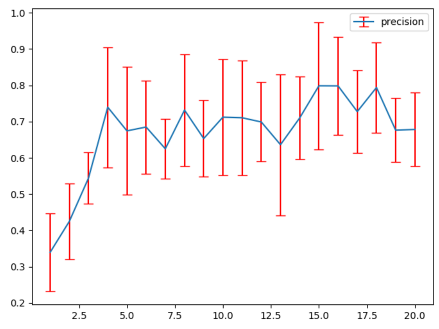
Values $tokens\_at\_step$ and num_steps were set to, respectively, 100 and 4. One can easily notice that the parameter $nsamples$ is a trade-off between computation time and expected quality of produced samples.

Correlation between number of samples at each step and F-measure.

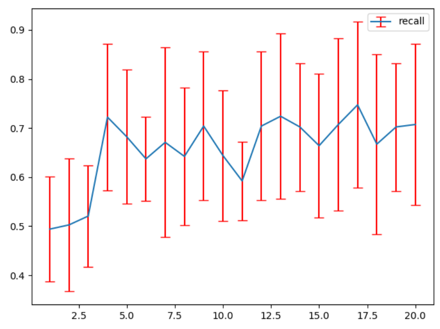
The time required to generate a text increases linearly with the $nsamples$ parameter. A correlation between this parameter and the quality of generated text is not so easy to reason about. To measure it experimentally, generation script was executed for each value of $nsamples$ from 1 to 20 against 6 manually written prompts. Then the 6 scores were averaged for three metrics: precision, recall and their harmonic mean – the F measure.



(a) F1 score



(b) precision



(c) recall

Figure 6.1: Correlation between number of samples at each step of generation and entity-wise quality of generated articles. x-axis – number of samples, y-axis – score in particular metric. The blue line is an averaged score, the red bars span from minimum to maximum obtained among the 6 prompts for a particular $nsamples$.

The results of this experiment suffer from high variance (which seems to be the only explanation why the obtained metrics are not monotonic), but a clear observation is that 5 seems to be a threshold value for $nsamples$, where the plot flattens.

Correlation between prompt likelihood and text quality

Probability distribution similarity of entities in real and generated samples

The approach of describing each text with an itemset of mentioned entities leads us to treating the corpus $\mathcal{C}$ as a database of transactions $\mathcal{D}_\mathcal{C}$, in which some entities are more likely to appear in a given context than others. For instance, *Chelsea F.C.* is more probable to occur in the same row of $\mathcal{D}_\mathcal{C}$ as *Manchester United* than *AS Roma*. It is intuitive to assume that the language model trained on this corpus will also model the probability distribution of itemsets in $\mathcal{D}_\mathcal{C}$.

This assumption was put under test – an unsmoothed probability distribution estimation was computed from entities collected from 100 generated samples. The target probability distribution was computed similarly from all real articles in the corpus. Cosine similarity was chosen as the measure instead of Kullback-Leibler divergence, because the latter requires the actual probability to have no non-zero values for all elements where the target probability has non-zero values[4]. The estimated similarity is presented below at Table 6.2.

Table 6.2: Similarity between entity probability distributions in generated samples and real corpus.

Generative model	Cosine similarity
GPT-2	0.655706
LSTM	0.652653

Prompt likelihood measure and experiments

During prompted generation, a user of such language model may want to draw prompts from a different probability distribution than the one from which $\mathcal{D}_\mathcal{C}$ is sampled (denoted as $\mathcal{P}_\mathcal{C}$). It's not easy to predict

[4] It is of course possible to smooth the probability distribution, but the scale of resulting KL divergence value is then dependent on the level of smoothing. In particular, as the probability $p(e)$ for some element e goes to 0, the KL divergence goes to $+\infty$.

how well will the language model perform prompted with an itemset S such that $\mathcal{P}_{\mathcal{C}}(S)$ is very small. An *exposure bias* is expected to occur. Experiments were made to research the correlation between the prompt likelihood and text quality.

The latter was evaluated using F1 score, to measure how compliant the texts are with provided prompts.

A prompt likelihood is, unfortunately, not so clearly defined. An obvious probability measure would be defined as $p(S) = \frac{\#_S(\mathcal{D})}{|\mathcal{D}|}$, i.e. fraction of rows of $\mathcal{D}$ which are exactly S. This definition is not meaningful enough for itemsets which never occur in $\mathcal{D}$ as they would always be assigned a zero probability. An itemset not present in $\mathcal{D}$, but differing from a real itemset by only one element intuitively seems more "likely" than an itemset consisting of e.g. all hundreds of thousands of entities in database. A new likelihood measure is introduced:

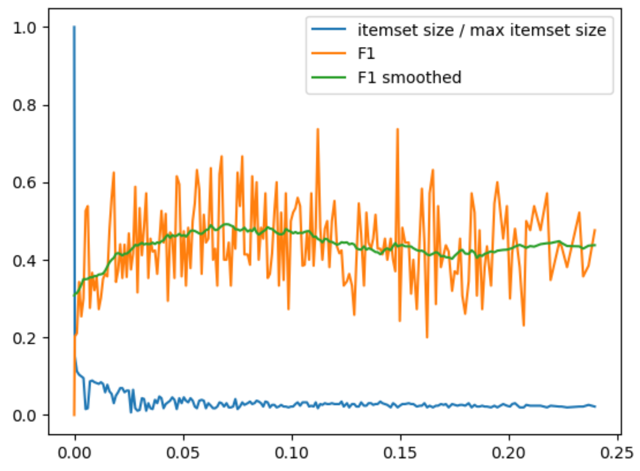
$$\mathcal{PL}'(S) = \texttt{avg}(\{support(s) \mid s \subseteq S, |s| \leq 3\}) \tag{6.6}$$

where *support* is defined as in Apriori algorithm. Such formulation is not enough as it does not consider the size of S. Luckily, the probability distribution of itemset sizes is easy to compute (see 3.1b) and sampled from. The likelihood measure is thus improved:

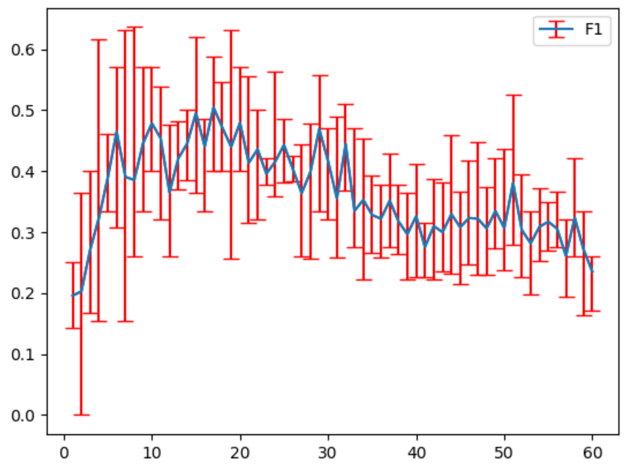
$$\mathcal{PL}(S) = \mathcal{PL}'(S) \cdot p(|S|) \cdot 10^3 \tag{6.7}$$

Note that while the 10^3 constant makes the value range of $\mathcal{PL}$ more pleasant to a human reader, $\mathcal{PL}$ image is not the same as that of a probability function. It is (at least for some database, not necessarily for BBC Sport articles) possible that $\mathcal{PL}(S) > 1$ for some S.

In order to research the correlation between prompt likelihood and text quality a large number of prompts of various likelihoods must be prepared. Instead of building an algorithm for collecting an itemset with fixed likelihood, a large number of itemsets was generated and most were filtered out to obtain a near-uniform distribution.



(a) Correlation between itemset likelihood measure and the F-measure.



(b) x-axis is the itemset size, y-axis is F-measure. Blue line shows the score averaged from 6 samples, red bars span the range from minimum and maximum scores among 6 samples.

Figure 6.2: Correlations between itemset properties and the F-measure.

No clear correlation was observed between the chosen $\mathcal{PL}$ metric and the quality of generated text. In particular, the F1 measure is not increasing as the $\mathcal{PL}$ increases, but peaks around $\mathcal{PL} = 0.06$. This result may simply show that the designed likelihood metric $\mathcal{PL}$ is not representative of what entity sets are really most probable.

On the other hand, there is a visible correlation between F1 score and itemset size, as confirmed by a separate experiment reported on subfigure (b).

6.2.3. EDGAR

The EDGAR repository, described in chapter 8 implements scripts needed to execute a full pipeline of control-code-driven generation. Prompt format from section 6.2.1 is generalized to

```
<category> ::= [a-zA-Z]
<category_entities> ::= "" | <entity> "," <category_entities>
<category_line> :== <category> ":" <category_entities>
<prompt> ::= "" | <category_line> "\n" <prompt>
```

so users can provide their own configurations for the NER component. EDGAR uses algorithm 2 to produce samples more compliant with provided prompts than raw GPT-2 model would do. Users can control parameters *tokens_at_step*, *num_steps*, *nsamples* via command-line or function arguments. All experiments presented above in this chapter are reproducible with little effort using EDGAR scripts on an arbitrary dataset.

6.3. Human evaluation of text cohesion

Human evaluation of prompted texts was conducted in a similar manner as in section 5.4. In both experiments below the prompted generation has performed significantly better than placeholder-driven generation.

Both experiments below have shown that prompted generation outperforms placeholder-driven generation under human judgment.

Setting

Again, a medium GPT-2 model with 36 layers and 355M trainable parameters was finetuned on a subset of BBC Sport corpus with only match reports. Article titles were once again normalized, i.e. exact score was replaced with an (in)equality sign.

We prompt the text generation with match result – one may point out that we did the same in experiments described in subsection 5.4. Note that in the experiments with template generation, only three prompts were possible: `{team:0} < {team:1}`, `{team:0} > {team:1}`, `{team:0} == {team:1}`, reducing it to a 3-class generation task. Here, $3 \cdot |\text{TEAMS}|^2$ prompts are possible.

6.3.1. Experiment: Longest cohesive prefix

The methodology followed the same rules as the experiment described in subsection 5.4.1.
Table 6.3 presents the aggregated results.

Table 6.3: Longest cohesive prefix experiment aggregated results.

NO	OK	OK,1	OK,2	OK,3	OK,4	OK,5
9	40	3	3	5	6	9
OK,6	OK,7	OK,8	OK,9	OK,10	OK,11	OK,12
7	9	1	3	2	2	1

6.3.2. Experiment: Online quiz

The online quiz from subsection 5.4.2 was reused to evaluate samples generated from prompts.

Once again, the quiz was shared on several subreddits (*r/machinelearning*, *r/datascienceproject* and *r/soccer*) as well as Facebook groups for University of Warsaw students. Results were collected after about a week.

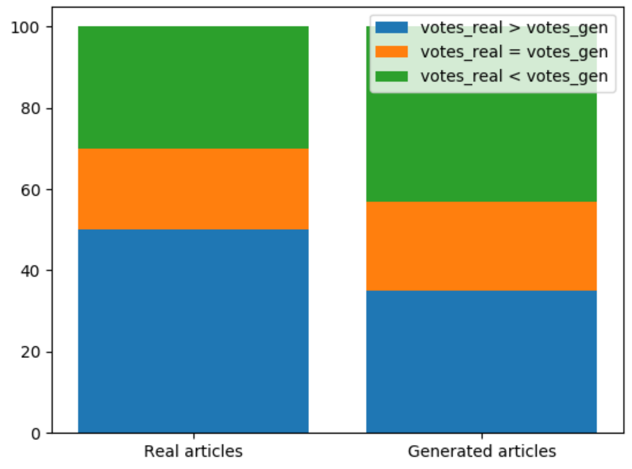
In total, 831 votes were collected, giving a bit over 4 votes per text sample on average. Votes were very well balanced – 415 votes said *REAL*, 416 said *GENERATED*. Figure 6.3a presents aggregated results.

6.3.3. Experiment: Online quiz on football fans

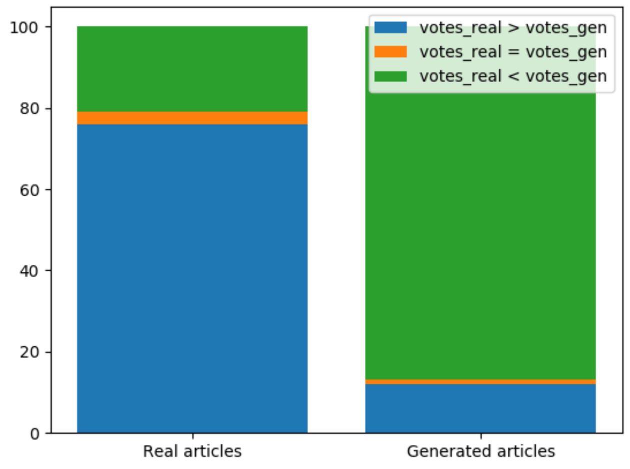
Comparison of results presented in sections 6.3.2 and 5.4.2 shows that samples generated from prompts are more believable to be real. Both experiments were conducted on a mixed audience, with varying degrees of domain knowledge, so most participants had to base their judgement on spotting logical and grammatical errors, or searching for common patterns or artifacts in fake samples.

To raise the bar for prompted generation, another online quiz experiment was done solely on football fans from *r/football* subreddit.

This time, the quiz drew much more attention. In total, 6258 votes were collected, 2910 stating that a text is real and 3348 that a text is fake. Maximum number of votes per single article was 50 and the minimum – 19. Unsurprisingly, that audience did much better at distinguishing real and fake articles. Figure 6.3b presents aggregated results.



(a) General audience online quiz results. Among the human-written articles, 50 were correctly identified as real (i.e. more votes said *REAL* than *GENERATED*), 30 were falsely named generated and in case of remaining 20, there was a tie. 35 of the generated samples have fooled majority of voters, 43 were correctly identified and for the remaining 22, there was a tie.



(b) *r/football* online quiz results. Among the human-written articles, 76 were correctly identified as real (i.e. more votes said *REAL* than *GENERATED*), 21 were falsely named generated and in case of remaining 3, there was a tie. 12 of the generated samples have fooled majority of voters, 87 were correctly identified and for the remaining 1, there was a tie.

Figure 6.3: Online quiz aggregated results.

Doncaster Rovers > Crawley Town
Doncaster moved into the League One play-off places with victory over Crawley.
The home side came closest to opening the scoring in the first half when Dan Jones' header from a corner hit the crossbar.
Doncaster had the ball in the net when Karleigh Osborne drilled in a shot from 25 yards, but John Marquis' save cleared the danger - before the goal came.
Rovers' third win in four games moves them up to fourth, four points clear of the bottom three.
The goal came when Martin Samuelsen's corner had not been dealt with by the Crawley defence and fired into the net.
Rovers' equaliser came on the stroke of half time, as the ball was knocked down to Osborne and the England Under-21 winger slid the ball into the net from close range.
Crawley's best chance came in the final minute, when Josh Yorwerth's header from a corner was cleared off the line by Marko Marosi.
Match report supplied by the Press Association.
Match ends, Doncaster Rovers 2, Crawley Town 0.
Second Half ends, Doncaster Rovers 2, Crawley Town 0.
Foul by Dean Furman (Doncaster Rovers).
Josh Yorwerth (Crawley Town) wins a free kick in the defensive half.
Substitution, Crawley Town. Dannie Bulman replaces Oliver

Blackburn Rovers == Norwich City
Norwich City missed a chance to move into the top six with a defeat at Championship strugglers Blackburn Rovers.
Keaton Wintle headed in Conor Sammon's cross to give Norwich an early lead.
The Canaries, who looked sure to claim their first away win of the season, went behind when Alex Pritchard's header from a corner was turned into his own net by Darren Ambrose.
And the Canaries were level when Graham Dorrans' low shot from the left side of the area beat keeper Angus Gunn.
Blackburn's best effort came from substitute Chris Taylor, but he could not keep his effort down.
Norwich, who were without captain Russell Martin because of a broken ankle, led when Graham Dorrans' shot from the left side of the area was deflected past goalkeeper Angus Gunn.
The home side's lead lasted just three minutes, though.
Wintle headed home from a corner, but the striker was penalised for handball.

Norwich, who have won just once in six games, did not win their first shot on target, with Taylor's effort blocked by a sliding

Figure 6.4: 2 of the 12 generated articles that have fooled quiz participants into thinking they're human-written. Two articles with highest $\frac{\text{votes\_real}}{\text{votes\_total}}$ ratio were chosen.

Additional notes regarding results:

- Participants often commented that what tipped them off was a wrong relation between entities (eg. a player listed in wrong team). To some extent this may have been caused by the fact that the model was trained on data from over 10 seasons so the team squads are surely inconsistent in training data. However, not all such mistakes can be attributed to the multi-season nature of corpus. For instance, in one generated sample, fooballer Kamil Glik is listed as FC Barcelona player (which was never true in any season of his career).

- Some participants have noted that while the test was easy for them to pass with a high score (such as 8 or 9 out of 10), it was only because they were explicitly searching for artifacts and other properties of machine-generated text and would not necessarily detect a fake article if they had casually read it.

- One of the most significant difficulties for the model was to consistently report on number of goals scored and chronology of the goals. This is why the task was simplified and the model was prompted only with an inequality sign between two teams, and not an exact end result.

- Note that only a random number of first $n \in [2, 3, ..., 10]$ sentences was displayed to a participant, so one should not draw a conclusion that the model is capable of outputting a realistic complete article.

Besides evaluating the model on a more difficult audience, another motivation behind this experiment was to seek correlation between number of sentences presented to the reader and the difficulty in spotting fake texts. At each step of the quiz, a random number of first sentences was presented to the user with the following distribution[5]:

#sentences	2	3	4	5	6	7	8	9	10
p	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.5

Distribution of samples that fooled the voters and those that were correctly classified as fake, for each number n of first n sentences displayed, is reported at Figure 6.5 below.

[5] The same distribution applied to both real and generated samples.

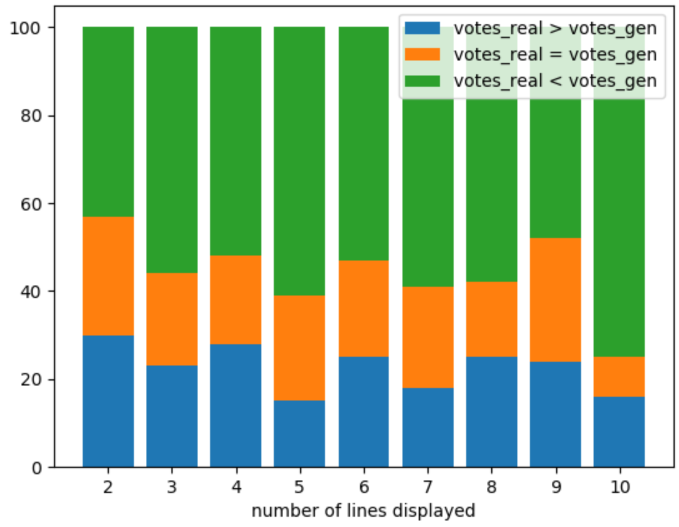


Figure 6.5: Correlation between number of lines displayed to quiz paricipant and votes distribution.

6.4. Experiment: Training a Transformer on Polish corpus

The experiments reported above in this and previous chapter were all conducted on an English corpus. The collected Polish articles, however more numerous and requiring more effort to annotate, were left unused. That is because the GPT-2 model was pretrained exclusively on English corpus and is thus only suitable for fine-tuning on that language.

Benefiting from the TensorFlow Research Cloud (TFRC) programme, which offers free access for 30 days to a limited selection of Google Cloud's Tensor Processing Units (TPUs), an attempt was made to train a Transformer on the Polish sport articles.

In the experiment, Tensor2Tensor library ([28]) was used. Firstly, a problem was defined as a subclass of `Text2TextProblem`, where inputs where entity sets, and targets – sport articles[6]. The maximum subtoken length was set to 8, and the dataset split proportions were left at default values (train:dev = 100:1). Almost all of the 0.5M scraped Polish articles were used: in total 490609 texts from `interia.pl/sport`, `sport.pl/pilka` and `pilkanozna.pl`.

The model was then trained for 250,000 iterations (around a week)[7] on a single TPU v3-8. Results were as underwhelming as they were interesting – an extreme case of overfitting has occured where the model has "memorized" training samples. Whatever the trained model was prompted with – real or hand-written prompt – it would always output a sample that was to be found in the training data.

It's very likely that the model has just failed to generalize due to too small training set. For comparison, [29] have used 58M samples to train an English-to-Czech translator, and [2] have trained English-French translator on 36M sentence pairs and English-German translator on 4.5 milion sentence pairs.

[6] It's worth noting that at first unconditional generation was attempted with `Text2SelfProblem`, but there are known issues in Tensor2Tensor library with handling text problems with no inputs.

[7] For comparison, [2] have trained their big models for 300,000 steps.

Chapter 7

Paradigm III: Adversarial training and reinforcement learning

7.1. Related work

While traditional MLE approaches still dominate (and recently have been rapidly improving thanks to attention mechanism and scaling-up), there were multiple attempts to incorporate adversarial training to text generation[1].

Exposure bias

The common motivation behind this (apart from great success of GAN variants in visual data domain making the architecture very appealing) is a so-called *exposure bias*. On a high level, sequential generative models trained with a Teacher Forcing algorithm learn to select the most probable token, given a certain prefix. However the prefixes fed to the model are drawn from a real data distribution, which is a very limited subspace of all representable inputs. While we can expect the model to perform well in this small subspace, its behavior gets more and more unpredictable as we wander further into the unexplored regions of the data representation space. With every step of generation, a language model accumulates some randomness, decreasing the text quality with every token.

Scheduled sampling

An interesting approach to treating the symptoms of exposure bias was scheduled sampling proposed by [30]. Authors have bridged the gap between teacher-forced training and free-running generation by interleaving Teacher Forcing with BPTT. In other words, input tokens at each step were drawn either from real data (with probability ϵ) or the network's previous output (with probability $1 - \epsilon$). The parameter ϵ was decreasing with each iteration and several scheduling strategies were experimented with, such as linear decay, exponential decay and inverse sigmoid decay. The results were impressive as the model was a winning entry in MSCOCO 2015 image captioning competition. Authors have also experimented with an *always sampling* strategy, where the model was always sampling from itself (i.e. $\epsilon = 0$), but, unsurprisingly, such approach didn't perform well as it was extremely difficult for the model to start learning.

[1] A purely adversarial training method that was not explored here is applying GANs to generate embedding matrix. For instance, a sequence of 32 tokens, each represented as 96-dimensional embedding, can be represented as an RGB 32x32 image, and translated back to text using nearest-neighbor approach for selecting the closest embedding. This approach was discarded since the characteristics of pictures generated by GANs, such as low detail level, repetitivenes and fixed size make it seem unpromising.

Professor Forcing

Professor Forcing ([31]) is arguably the most direct attempt to apply GANs to text generation (although the original work applies it to both discrete and continuous data). Authors show that the exposure bias in recurrent neural networks is observable not only in the discrete output of the network (which humans are best at evaluating), but also in the space occupied by the GRU's hidden states. Using T-SNE visualisation method, authors have shown a discrepancy between hidden states during teacher-forced runs of the network and hidden states in free running mode (i.e. generation).

The intuition behind Professor Forcing architecture is that if exposure bias is unobservable in RNN's hidden states, it will also be remedied in generated sequences. This motivates a GAN setup, where a discriminator is a learned function of generator's hidden states, distinguishing teacher-forced runs from free runs.

Training objectives

Discriminator parameters θ_d and generator parameters θ_g are trained in parallel, combining GAN setup with teacher forcing. The generator RNN minimizes two training objectives[2]:

- the negative log-likelihood objective:

$$NLL(\theta_g) = E_{(x,y)\sim data}[-\log P_{\theta_g}(y|x)] \tag{7.1}$$

- the criterion of fooling the discriminator:

$$C_f(\theta_g|\theta_d) = E_{x\sim data, y\sim P_{\theta_g}(y|x)}[-\log D(B(x,y,\theta_g),\theta_d)] \tag{7.2}$$

  where $B(x,y,\theta_g)$ is a behavior sequence – hidden states and output values of generator's free run and $D(b,\theta_d)$ is the probability, estimated by discriminator, that b was prodced in teacher-forcing mode.

The discriminator is trained as a classifier for behavior sequences:

$$C_d(\theta_d|\theta_g) = E_{(x,y)\sim data}[-\log D(B(x,y,\theta_g),\theta_d) + E_{y\sim P_{\theta_g}(y|x)}[-\log(1-\log D(B(x,y,\theta_g),\theta_d))]] \tag{7.3}$$

Experimental results

Authors report almost state-of-the-art results on character-level language modelling on Penn-Treebank corpus, but they note that no difference was observable between teacher forcing and Professor Forcing on a word-level generation.

On handwriting generation task, Professor Forcing performed better than Teacher Forcing in 76.9% cases, as measured by human evaluation. Similarly, on music synthesis in raw waveforms, it has outperformed Teacher Forcing, achieving an average human score of 2.20 (in a scale 1-3), whereas Teacher Forcing scored only 1.30.

The results are promising, and so is the complexity of the architecture, which scales only linearly as the sequence length grows. It also converges faster than teacher-forced networks and is better at maintaining long-term dependencies. While Professor Forcing architecture was not experimented with in this work, it is worth exploring in the future as it may benefit conditional generation.

[2] Original equations from [31].

SeqGAN

SeqGAN ([32]) adapts a traditional GAN setup, where the discriminator judges output of generator, not its internal state. The osbtacle of discrete data is overcome by expressing the learning procedure in reinforcement learning terms.

The discriminator is a CNN sentence classifier ([33]), implemented as a highway network ([34]) to allow for faster convergence with many layers. The generator RNN consists of LSTM cells, but other variants, such as GRU or even attention mechanism from [12] can be used as well.

First, generator is pre-trained until convergence to optimize the negative log-likelihood (NLL) objective. During aversarial training, generator works to optimize its expected end reward[3]:

$$J(\theta) = \mathbb{E}[R_T|s_0, \theta] = \sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1) \tag{7.4}$$

where:

- R_T is the final reward for the generated sequence
- $Q_{D_\phi}^{G_\theta}$ is the action-value function assuming that the agent follows the policy G_θ

The action-value function is very well defined for complete sequences (it's equivalent to the discriminator output), and for intermediate states it is approximated with Monte Carlo search $MC^{G_\beta}(Y_{1:T}; N)$, where G_β is the rollout policy (which can be modelled by the same generator RNN or a computationally faster approximation). Authors use $N = 16$ Monte Carlo rollouts in their experiments; the same value is used in this work. The gradients of the objective function $J(\theta)$ are then approximated as:

$$\nabla J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{y_t \sim G_\theta(y_t|Y_{1:t-1})}[\nabla_\theta \log G_\theta(y|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)] \tag{7.5}$$

SeqGAN is an appealing architecture for exploring in search of conditional generation techniques, because of its modular design, where very little assumptions are made about each module:

- **generator** can be virtually any neural network with sequential output. The only constraint is that gradients of output probability distribution at each step must be computable.
- **discriminator** is here a convolutional neural network, capable of improving at each step, but any, not necessarily differentiable, function assigning a reward to a complete sequence could be used. A neural net classifier is used in SeqGAN as there is no known metric which could serve as an oracle distinguishing real and fake texts, nevertheless other metrics can be introduced or combined if we want to redefine the training objective (for example to enforce the text sentiment).
- **rollout policy** by default is the same as generator, but faster alternatives can be used if necessary. Again, there is no requirement for this policy to have a computable gradient.

Complexity

Computational complexity is unfortunately a serious limitation of SeqGAN. With N Monte Carlo rollouts, sequence length of M tokens, and batch size B, the total complexity of a single training step (for generator) is $O(NM^2B)$. While the complexity of Self-Attention mechanism in Transformer is also growing quadratically with the sequence length, the Self-Attention is easily parallellized among available processing units, whereas

[3]Original equations from [32]

in SeqGAN, each rollout is sequential. Moreover, Monte Carlo rollouts in SeqGAN incorporate generating a quadratic number (again, $O(NM^2B)$) of tokens, i.e. costly operations of sampling from probability distribution, which does not levarage the GPU.

Several approaches were explored when implementing SeqGAN in this work to approximate the end reward more efficiently. The considered alternatives are described later in this chapter. Unfortunately, none of them performed good enough quality-wise to replace the MC rollouts.

Original results

Another contribution of the SeqGAN paper is proposing synthetic data experiments for evaluating language models. A randomly initialized neural net serves as an *oracle*. In other words, we assume that the oracle is a perfect language model for *some* language. The evaluation metric is then defined as

$$\mathrm{NLL}_{oracle} = -\mathbb{E}_{Y_{1:T} \sim G_\theta} \big[\sum_{t=1}^{T} \log G_{oracle}(y_t | Y_{1:t-1}) \big] \tag{7.6}$$

Using this metric, SeqGAN outperforms 4 compared LSTM training procedues (listed from worst to best performing): random token generation, MLE, scheduled sampling and PG-BLEU (Reinforce algorithm with BLEU reward).

SeqGAN outperforms recurrent network trained with MLE objective on real-life tasks such as Obama speech generation, Chinese poem generation and music generation (sequence of 32 pitches represented as one of the 88 piano keys). The generated samples were evaluated using BLEU metrics, and, in case of poem generation, also human judges. Unfortunately, other methods such as scheduled sampling or PG-BLEU are not evaluated on those tasks.

RankGAN

[35] argue that the binary reward returned by the convolutional discriminator in SeqGAN is not descriptive enough and provides little information for the generator. This conclusion was also one of the observations emerged from experiments conducted for this work. Indeed, the semantics behind values 0 and 1 in reward are defined (*this text is surely real/fake*), but the distribution of values in between depends on discriminator's implementation. For instance, the original discriminator architecture from SeqGAN (10 convolutional layers) resulted in a very polarized discriminator, always outputting values smaller than 0.1 or larger than 0.9. The more shallow the discriminator, the less polarized the outputs were (although there was not much difference in overall acccuracy).

The idea behind RankGAN is that the discriminator, instead of assigning a (soft-)binary reward to a single sequence, ranks a set of sequences. The generator learns to produce sequences which rank high against a set of real reference data.

The ranker R uses learned convolutional filters to map input sequences x into embedded feature vectors y_x. A score of sequence s given reference sequence u is then defined as[4]:

$$\alpha(s|u) = cosine(y_s, y_u) = \frac{y_s \cdot y_u}{\|y_s\| \|y_u\|} \tag{7.7}$$

Sequence s is then ranked against a set of sequences U by distributing their scores using softmax:

$$P(s|u, \mathcal{C}) = \frac{exp(\gamma \alpha(s|u))}{\sum_{s' \in \mathcal{C}'} exp(\gamma \alpha(s'|u))} \tag{7.8}$$

[4] Original equations from [35]

The final reward is then defined as:

$$R_\phi(s|U, \mathcal{C}) = \mathop{\mathbb{E}}_{u \in U}[P(s|U, \mathcal{C})] \tag{7.9}$$

Authors note that the sizes of reference sets U and $\mathcal{C}$ do not have observable impact on convergence time and thus a single reference sample u is used in their experiments. The set $\mathcal{C}$ consists of one real and one generated sequence. The rest of the setup is similar to SeqGAN.

In the synthetic data experiments, RankGAN outperforms MLE, PG-BLEU and SeqGAN[5]:

Method	MLE	PG-BLEU	SeqGAN	RankGAN
NLL	9.038	8.946	8.736	8.247

A common issue in GANs is polarization of the generator's weights leading to repeating the same sample. Once the discriminator learns that this particular sample is generated, the generator modifies it slightly and in result generator and discriminator keep pushing point in representation space infinitely, without improving the generated content quality.

Unfortunately neither [32] nor [35] report on how severe was the mode collapse in their experiments. While it could not have been significant enough to impair the generator's quality – otherwise the proposed architecture would not have scored so highly in experiments – it would be interesting to see how the space occupied by generated samples changes over time.

A hint of how the generated samples distribution differs from real data can be seen on Figure 4 in [36].

GPT-2-FT-RL

OpenAI has experimented with fine-tuning GPT-2 using Reinforcement Learning on human preferences ([37]). For text summarization task, the model has converged after 60k steps. Tasks such as generating a continuation of text carrying a positive sentiment or a physically descriptive text required only 5k steps.

It is worth noting that on the summarization task, human judges have preferred samples which directly copied siginificant phrases from original text[6].

7.2. Evaluation

A perfect method for evaluating language model is human judgement. It cannot be implemented in practice in a large scale due to:

a) high inefficiency and cost of employing human effort

b) subjectivity of human judgement. It's difficult to define on a non-binary scale what does a certain score mean (e.g. when should the judge assign a score 0.6 on a scale [0..1]?). A sensible binary scale, on the other hand, would resemble a Turing test – without prior knowledge about the source of a particular text, the judge needs to guess whether it's generated or real. In practice very few currently existing language models are good enough to possibly obtain non-zero score in Turing test.

[32] note that during human evaluation, the judge has full knowledge of the natural language distribution $p_{human}(x)$. Apart from evaluating how well did the neural network learn a *particular* natural language, they

[5]Table via [35]

[6] While this finding was likely disapppointing to the authors, it is interesting in context of the BLEU metric. As shown later in this chapter, BLEU can lead a model to copying n-grams from the corpus it was trained on. As it turns out, human judges are also "imperfect" in that way.

measure its capability to model any language described by an oracle. In their synthetic data experiments, an oracle is defined by a randomly initialized LSTM G_{oracle}. Others ([36], [35]) reuse this idea in their experiments. [38] argue that a randomly generated LSTM gives little control over modelled data complexity (which in case of natural languages is rather high) and instead use CFGs and P-CFGs as oracle language models. Regardless of the approach to synthetic data experiments, both in terms of their reliability and implementation, all mentioned works report on real data experiments as well, most commonly using BLEU metrics. In this work, BLEU metric in its different variants is adopted for evaluation, since it's difficult to reason about textual content of synthetic data. There is one caveat about using BLEU metric – PG-BLEU will always outperform other language models. As shown later in this chapter, it usually achieves its goal by seeking several common n-grams and repeating them in every sample (mode collapse in GAN nomenclature). In order to measure the severity of mode collapse in each experiment, a ratio of distinct words and/or bigrams is reported.

7.3. SeqGAN experiments

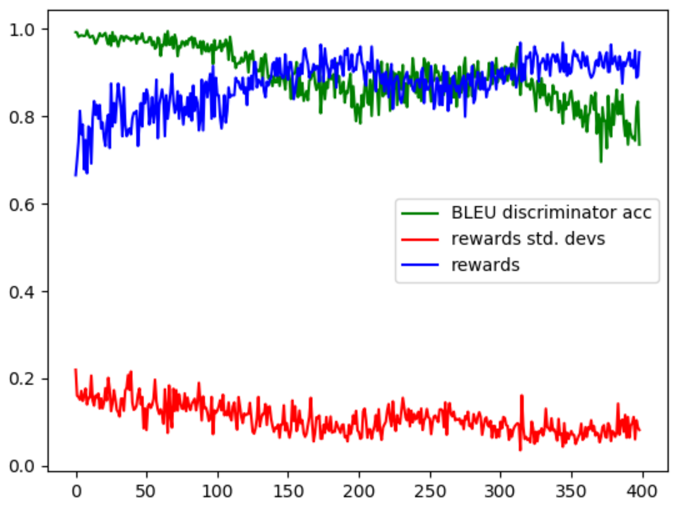
PG-BLEU

A certain appeal of the SeqGAN architecture lies in the fact that the discriminator can be any, not necessarily differentiable, function. The general framework, once implemented can be reused as any policy gradient setting.

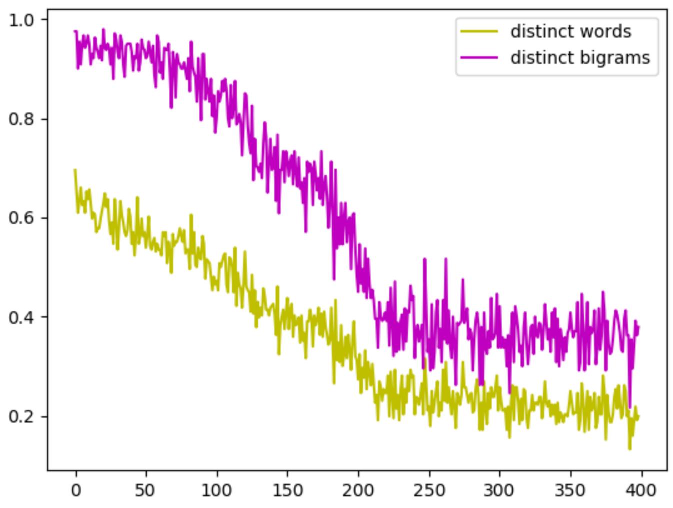
BLEU([39]) is one of the popular metrics to measure text generation quality([36], [40]). While it is rare in NN-based machine learning for the optimized loss function to be the same as the metric used at evaluation time, here we can directly optimize the BLEU metric by presenting the problem in reinforcement learning terms. The setup is similar to [41] – first a recurrent neural network (LSTM) is pretrained in standard Teacher Forcing setting, then policy gradients are used to directly optimize the reward, here defined as the BLEU metric. MIXER ([41]) algorithm is more computationally efficient in that it only draws a single sample from the probability distribution to estimate the expected reward.

In the experiments presented here a constant sequence length of 16 was used so time complexity was not an issue and the number of samples in Monte-Carlo rollouts remained the same – 16.

In the first BLEU experiment, BLEU-3 variant was used. The discriminator was "pretrained" by iterating through all sequences of 16 tokens in the training set and, for each 3-gram (a, b, c), computing the maximum number of its occurrences in a single sequence $m_{abc} = \max_{s \in S_{train}}(\#_{abc}(s))$. The reward for a generated sequence $w = w_0 w_1 ... w_l$ is then computed as $r_w = \sum\limits_{abc \in w} \frac{min(\#_{abc}(w), m_{abc})}{\#_{abc}(w)}$.



(a) BLEU-3 score obtained by generator's samples for the first 400 iterations. To compute the discriminator's accuracy, every sample with BLEU score equal to 1.0 was classified as real, any score lower than 1.0 meant the sample was generated.



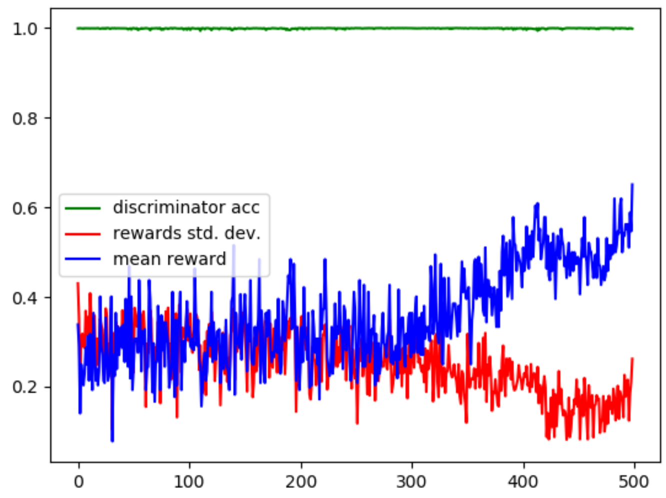
(b) Fraction of distinct words and bigrams in a minibatch dropping.

iter	**BLEU-3**	**sample**
0	0.665	`=dot= Mathieu replaces Aiden O'Brien =dot= Bartley (Southend United) is shown the yellow card =dot= \nSam`
100	0.848	`the former Arsenal boss Jurgen "We have now conceded two and =comma= to add an defender`
200	0.942	`side 2 =comma= Gareth Bale now =comma= the of the of the of the =dot= fans`
300	0.889	`(Derby County) =dot= \nSecond Half begins Salford City 1 =comma= Arsenal 2 =dot= Jack Wilshere (Arsenal)`
400	0.947	`missed his =dot= \nThe European number of the of the of the of the =dot= \nThe`

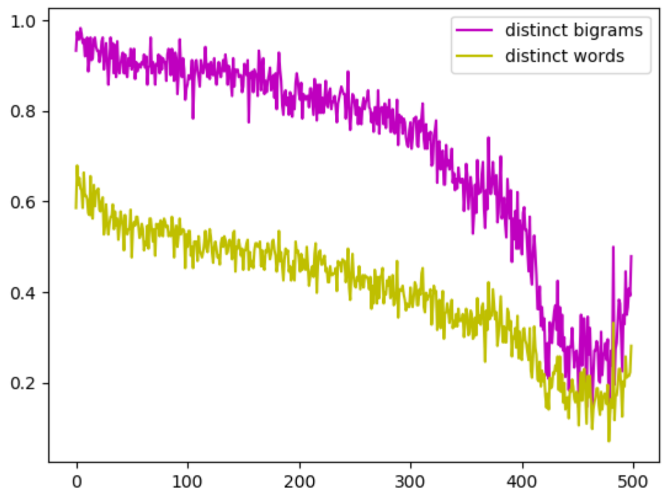
(c) Samples drawn every 100 iterations.

Figure 7.1: Pretrained LSTM generator trained by policy gradient updates to optimize for BLEU-3.

Figure 7.1 shows that even though the generator was indeed improving the BLEU-3 metric, it also started reducing its vocabulary. Unsurprisingly so – a generator finding one perfect sentence and producing it every time is an almost perfect agent in this RL setup (almost because the one n-gram containing the input token may not be present in any of the reference sequences). A possible remedy to this problem is adding a regularizer term, rewarding ratio of distinct words to all words in the mini-batch. An important issue observable in the listed samples is that the generator degenerated towards abusing 3-grams *of the of* and *the of the*, which are incorrect even when used once. The fact that the generator is rewarded for repeating them suggests that there exists a similar sample in the training set. As seen on Figure 7.2, the problem is also present for 5-grams. A likely hypothesis is that the problem starts at the tokenization step. Since no *=unknown=* token was used, article fragments such as: `Coach of the Year:\n National Team of the Year:\n Club of the Year:\n` were squashed to *of the of the of the* (words such as *Coach, National, Club* were unknown to tokenizer because it was case sensitive).



(a) BLEU-5 score obtained by generator's samples for the first 500 iterations.



(b) Fraction of distinct words and bigrams in a minibatch dropping.

iter	BLEU-5	sample
0	0.339	appearance =comma= Yeovil had it clear off when Ward missed a good chance on the counter
125	0.322	after a =dot= \nThe goal came for the only 10 men before Villa came after the
250	0.375	much =comma= and I was not a good season and I'm very =comma= but to see
375	0.417	a of the club =comma= the best way the club =dot= \nThe club was a of
500	0.651	in the of the of the of the of the the club and the outcome =dot=

Figure 7.2: BLEU-5 learning reports.

In the further experiments, the tokenizer's word limit was increased from 5000 to 10000 words, *=unk=* token was introduced for out-of-vocabulary words and the tokenizer was made case-insensitive.

This change turned out to be insufficient. Even though the *=unk=* tokens were rare in the training set (vide Figure 7.3), the RNN suffered from a mode collapse, favoring sequences with multiple occurrences of this token (although the quality was still higher than in previous experiments). Mitigating the mode collapse seems crucial to improving text quality.

In the next experiment, the reward for a minibatch $M = [w_1, w_2, ..., w_k]^T$ ($|w_i| = m$ for all i) was computed as follows:

$$r(M) = [BLEU5(w_1), BLEU5(w_2), ..., BLEU5(w_k)]^T \cdot (1 - \frac{1}{0.6} \cdot |0.6 - \frac{\#\text{distinct words in } M}{k \cdot m}|) \quad (7.10)$$

The 0.6 constant was manually chosen as desired ratio of distinct words in a minibatch based on the plots from previous experiments. Figure 7.4[7] reports on those efforts. Interestingly, the generator after rapidly improving for the first 100 epochs, started slowly decreasing obtained rewards. Enforcing variety in the reward term was not successful and the generator focused on maximizing the BLEU metric instead. A probable hypothesis is that such REINFORCE setting simply fosters the generator's polarization. Since rewards are always positive, the followed path weights in the RNN always get increased. A possible

[7] It's worth noting that the Monte-Carlo rollouts were replaced with only taking one sample in each step. Since the discriminator rewards the number of distinct words in a batch, batches consisting of 16 different rollouts of the same prefix were bound to score poorly. In another experiment the rollouts were randomly distributed among different batches, but the results were similar, while learning speed much lower.

improvement could be subtracting a baseline term (such as average reward so far, or a momentum) to make rewards zero-centered. This is left for potential future work but not researched in more depth here.

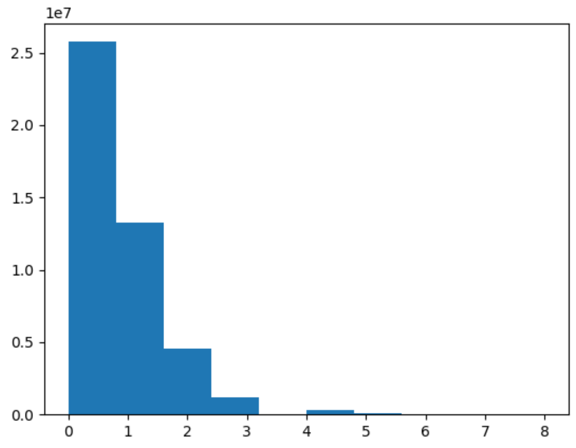
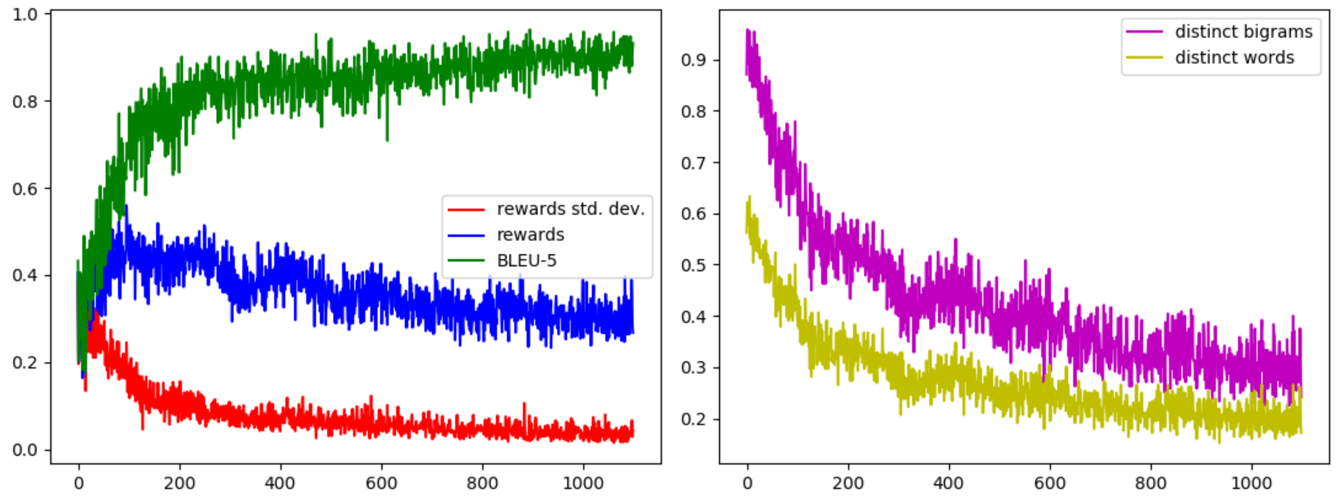


Figure 7.3: Histogram of number of *=unk=* tokens among 45150752 training sequences (each sequence was 16 tokens long).

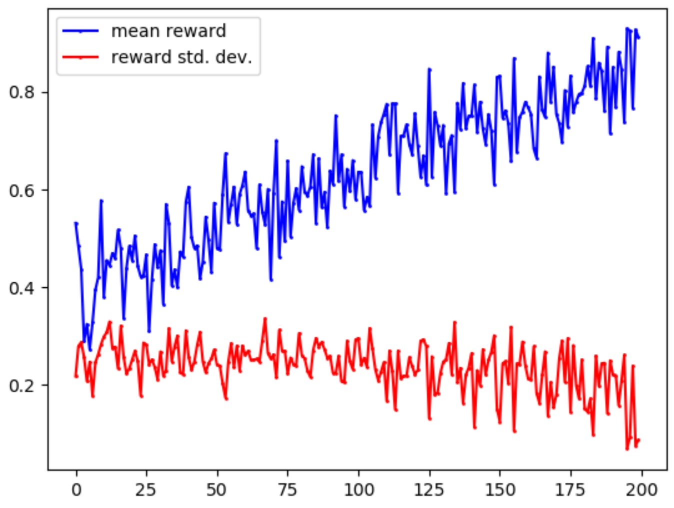


iter	reward	sample
0	0.405	`boyata =unk= has impressed with a small =unk= goal of the season =dot= moyes has spent`
250	0.514	`=unk= caught =unk= cross from the left side of the box =dot= \ncorner =comma= solihull moors`
500	0.386	`=dot= scored in the right side of the box =dot= assisted by =unk= \nfoul by =unk=`
750	0.316	`on the right side of the box =dot= assisted by =unk= \ncorner =comma= cardiff city =dot=`
1000	0.389	`that result of the right side of the box =dot= assisted by =unk= \nfoul by =unk=`

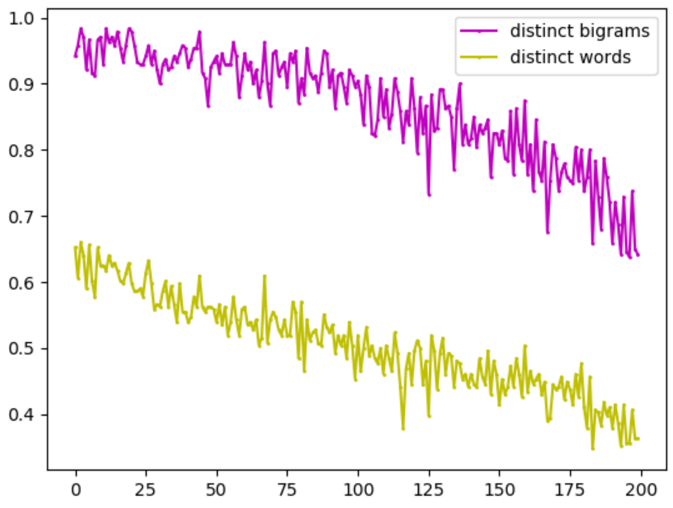
Figure 7.4: Report on PG-BLEU with reward for distinct word ratio. While the samples were of higher quality (based on very subjective human judgement of the author), mode collapse was still severe and lasting on the span of many iteratons (vide samples from iterations 500 and 1000).

With discriminator disabled

In this experiment, the SeqGAN architecture was trained with disabled discriminator. The discriminator was pretrained on 3000 real and 3000 generated texts, but it was not updated during the adversarial phase. This kind of experiment, apart from showcasing the consequences of generator dominating the discriminator served to verify the policy gradient implementation. Figure 7.5 reports the experiment results.



(a) Increasing rewards received by the generator over time.



(b) Decline in number of distinct words/bigrams in generated minibatches over time.

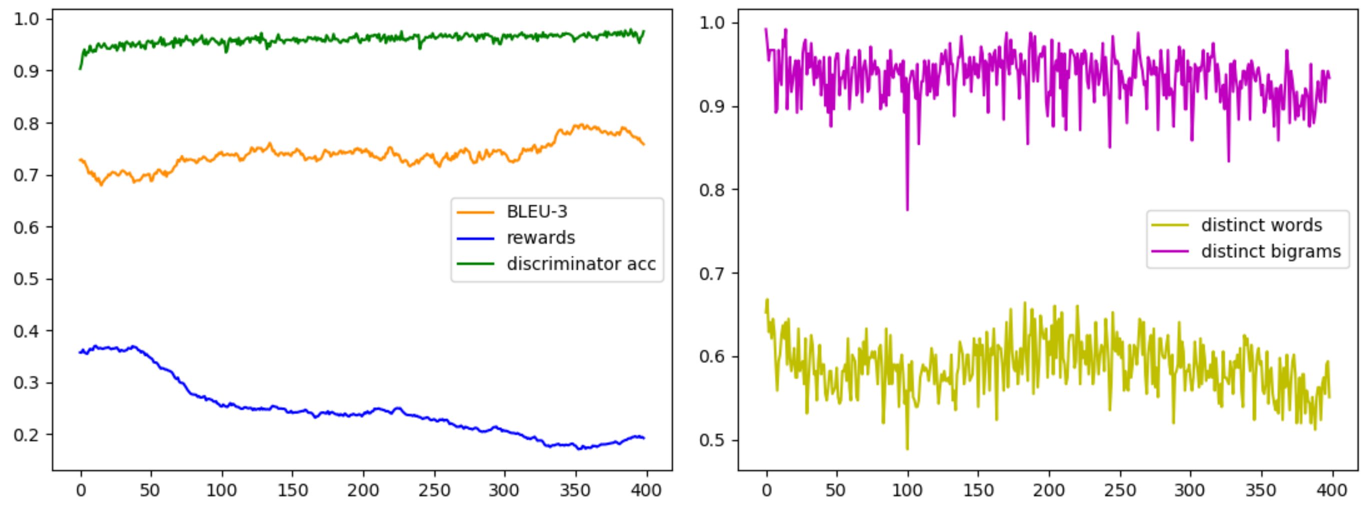
step	sample
1	*with a cross =dot= \nPenalty conceded by Richard Duffy (Oxford United) after a foul in the*
50	*right footed shot from the centre of the box is saved in the centre of the*
100	*of former England loanee is three =dot= "\nIt added: work what to do =dot= are a*
150	*after =dot= \nLast week =comma= 000 =comma= 000 =dot= \n"I have seen =dot= \nThe 25-year-old was*
200	*=dot= =comma= though =comma= =dot= \nIt is when I seen =dot= Town =comma= =comma= =comma= =dot=*

(c) Degeneration of text samples during adversarial training.

Figure 7.5: SeqGAN being trained with the discriminator disabled. Before the beginning of the adversarial training, the discriminator had been pretrained to obtain 0.83 accuracy on the validation set. During the adversarial phase, the discriminator's weights were not updated, leading to generator to fixating on several tokens (mostly interpunction markers) which provided high rewards. While this was expected, running a GAN with the discriminator disabled is a useful sanity check that the generator *does* optimize its goal of fooling the discriminator.

Vanilla SeqGAN

First attempts to run the full SeqGAN architecture have resulted in the discriminator instantly taking the lead and outputting 0 rewards for all generated samples. For the purpose of this work, the discriminator network was reduced in size – only every second convolutional layer from original architecture was left. With only 5 layers, the highway architecture was no longer needed.
The network was then trained for 400 iterations with 64 samples in mini-batch. Figure 7.6 reports how the BLEU metric was changing during the training process.



iter	sample
0	williams (oxford united) right footed shot from a difficult angle on the left is blocked =dot=
100	wide from the centre of the box =dot= \ncorner =comma= queen's park =dot= conceded by =unk=
200	(inverness ct) header from the centre of the box misses to the right =dot= \ngoal! bromley
300	to the =unk= tournament =comma= after winning the process and goals we've got to win =dot=
400	=comma= and the =unk= and midfielder had now come off the line and into the box

Figure 7.6: Learning curves with BLEU

Alternatives for rollouts

Three alternative methods of approximating the expected end reward from non-final generation state were explored: appending real data, a special case of Monte Carlo rollouts with $N = 1$ and a *Value Network*.

Value Network (Estimator)

An idea loosely inspired by the Alpha Go architecture was to, instead of performing costly rollouts, implement a network which, for a given state (i.e. prefix of a sequence), estimates the expected end reward. The Estimator network followed the same architecture as Discriminator, but at each layer had twice as many convolution filters. For each Discriminator training batch $B = \{(X_0 : y_0), (X_1 : y_1), ...(X_m, y_m)\}$, where $X_i = x_{i,0}, x_{i,1}, ..., x_{i,n-1}$, the Estimator was trained on a batch

$$\{(x_{i,0}x_{i,1}...x_{i,k}0^{n-1-k}, D(x_{i,0}x_{i,1}...x_{i,n-1}))|i \in [0...m], k \in [0..n-1]\} \tag{7.11}$$

(where $D(X)$ is discriminator's prediction for sequence X) but with a much smaller learning rate. Estimator was both trained and evaluated with MAE (mean absolute error) loss function. Figure 7.7 reports on Estimator's MAE metric over first 100 iterations of adversarial training. For this evaluation, SeqGAN was trained as usual, with MC rollouts, but in addition mean absolute error was measured between end reward from rollout and the end reward predicted by Estimator.

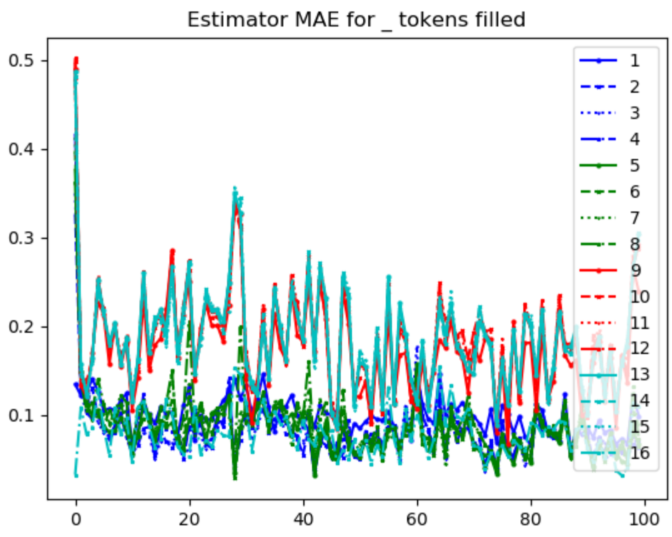


Figure 7.7: Estimator network loss over first 100 adversarial iterations

Estimator has performed well for prefixes shorter or equal than 8 tokens but poorly for longer prefixes. It's therefore not good enough to completely replace the MC rollouts, but some speedup could be achieved by a hybrid solution. Due to unpromising results this was not explored in more depth, but is a potential improvement of the SeqGAN architecture.

Since the Estimator network has not performed as good as expected, further experiments with training SeqGAN using the value network were not conducted.

Appending real data

In this experiment, the computationally expensive rollout policy was replaced with appending real samples from the training dataset to the end of generated sequences. At timestep t, $M - t$ tokens were appended to the t generated ones.

Unlike the two other alternative methods of computing intermediate rewards, appending real data still entails quadratic computational complexity. A constant-factor speedup comes from the fact that now only $O(MN)$ tokens are generated, but still $O(M^2N)$ tokens are fed into the discriminator[8].

Figure 7.8 reports the learning curves from the experiment on two different discriminators – CNN discriminator (trained in an interleaved manner with generator) and BLEU-3 with repetition penalty.

Two natural concerns arise regarding this alternative for rollouts:

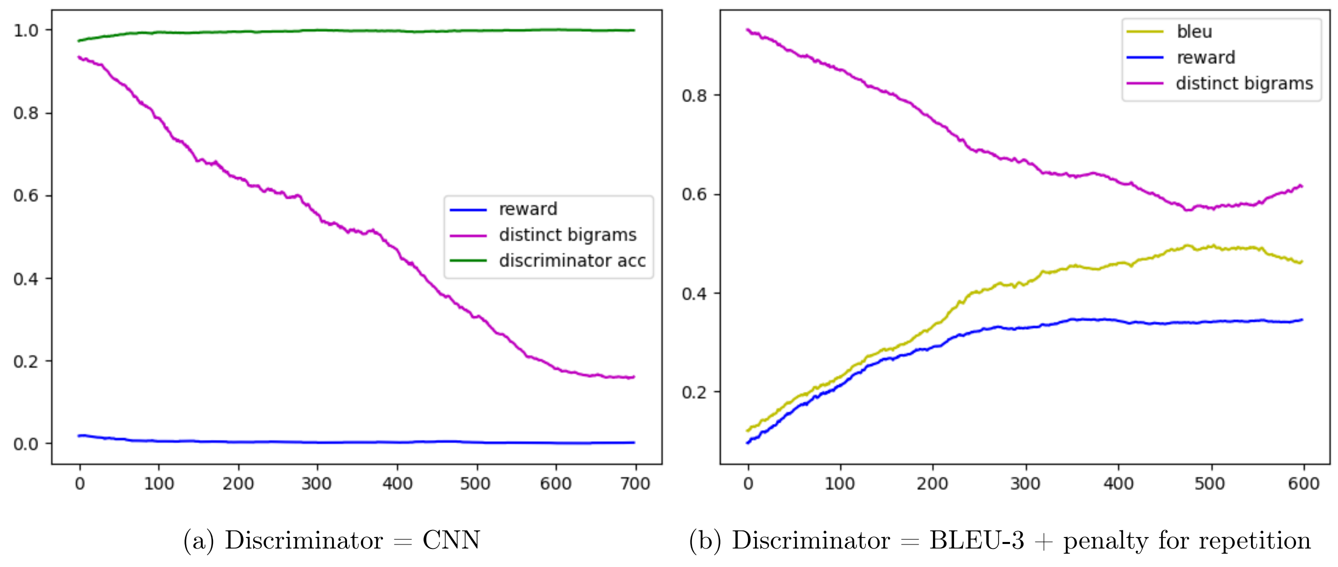
1) The random data appended may (and most likely will) not fit the generated prefix.

2) Intermediate rewards are less meaningful.

Figure 7.9 reports intermediate rewards from the adversarial traninING procedure. Interestingly, the choice of discriminator has made a large difference in the structure of rewards. Unsurprisingly, rewards from earlier steps were higher than for near-complete sequences (an issue that could probably be aided by normalizing rewards at each timestep). An unexpected finding is that, during training with BLEU-3 discriminator, all intermediate rewards stayed on roughly constant level and it was only the final reward (i.e. for complete sequence) that has changed over the course of training procedure. Perhaps this could be explained by the

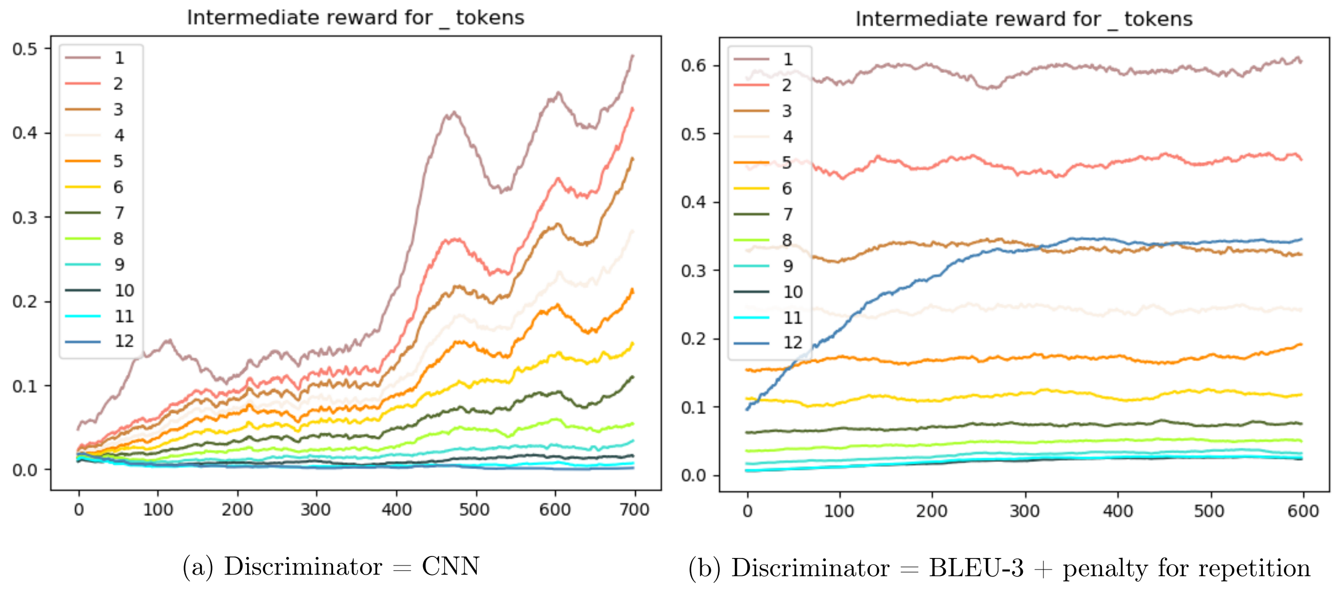
[8] N denotes number of rollouts and M the fixed sequence length.

concern 1) – maybe the generated prefixes glued with real data are "awkward" enough to hinder any growth of the rewards as the generator improves.



(a) Discriminator = CNN (b) Discriminator = BLEU-3 + penalty for repetition

Figure 7.8: Learning curves when appending real data instead of MC rollouts.

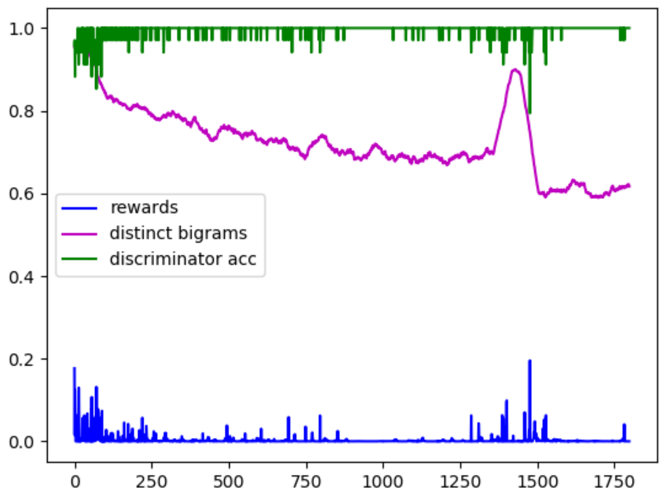


(a) Discriminator = CNN (b) Discriminator = BLEU-3 + penalty for repetition

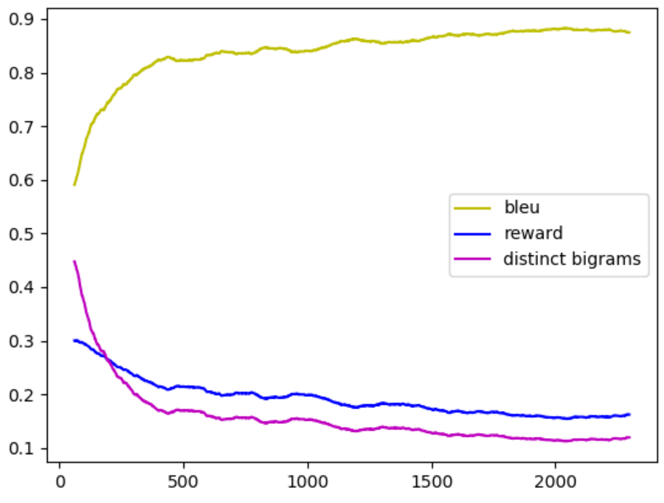
Figure 7.9: Average intermediate rewards at different timesteps. x axis is the training iteration, y – the reward.

$\mathbf{N = 1}$

In all other experiments, $N = 16$ Monte Carlo rollouts are performed at each step. In general, the higher value of N, the less variance is expected in end rewards. While all values $N > 1$ entail a quadratic complexity, with $N = 1$ we can only use the single generated sequence to compute gradients at each generation step. Such intermediate reward is clearly much less meaningful, as it does not provide feedback for each token. Figure 7.10 demonstrates experimental efforts to train SeqGAN and PG-BLEU models with only a single rollout at each step.



(a) Discriminator = CNN



(b) Discriminator = BLEU-3 + penalty for repetition

Figure 7.10: Learning curves with the extreme case of $N = 1$ Monte Carlo rollouts.

Applications for controllable generation

The reinforcement learning setup of architectures such as PG-BLEU, SeqGAN or RankGAN is, at least in theory, very promising for controllable text generation. This framework allows plugging in an arbitrary discriminator $\mathcal{D}$ and optimizing the generator $\mathcal{G}$ towards the highest rewards from $\mathcal{D}$. In particular, the $\mathcal{D}$ can be decomposed as $\mathcal{D} = \alpha_1\mathcal{D}_1\Box\alpha_2\mathcal{D}_2\Box\ldots\Box\alpha_k\mathcal{D}_k$, where $\Box$ is an arbitrary binary operator $\Box : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ and $\alpha_1, \alpha_,..., \alpha_k \in \mathbb{R}$.

Furthermore, both the generator $\mathcal{G}$ and discriminator $\mathcal{D}$ can be conditioned on a control code C that may be different at every step of training. With this much flexibility, we can revisit section 2.2 and directly translate our expectations into the following architecture:

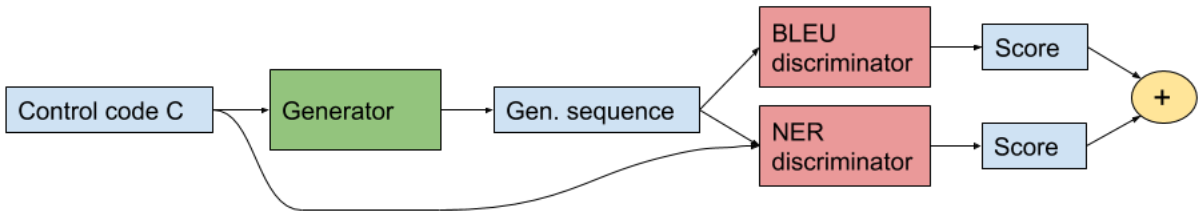


Figure 7.11: Architecture put under test in described experiments.

The control code is a prompt built with techniques presented in chapter 4.

Generator $\mathcal{G}$ is implemented in a seq2seq manner (without attention). First, an unidirectional LSTM $\mathcal{G}_{in}$ layer reads the control code and outputs a context vector v. Next, another LSTM layer $\mathcal{G}_{out}$ conditioned on

v generates a text sample. Following [10], $\mathcal{G}_{out}$ receives the context vector v at each timestep, concatenated with the word embedding of current token[9]. $\mathcal{G}_{out}$ then outputs a sequence that is copied and fed into two discriminators: $\mathcal{D}_{BLEU}$ and $\mathcal{D}_{NER}$.

$\mathcal{D}_{BLEU}$ could in fact be further decomposed into two discriminators, since it computes the BLEU-3 metric and a ratio of distinct bigrams in the batch and returns the arithmetic mean of the two as reward. $\mathcal{D}_{NER}$ is conditioned on control code C. It extracts entities from C as the ground truths set y and entities from generates samples as predictions p. Then it returns the F-score(y,p) as reward. The total reward is then computed $\mathcal{D} = \frac{1}{2}\mathcal{D}_{NER} + \frac{1}{2}\mathcal{D}_{BLEU}$.

Experiments

In the experiments, BBC Sport corpus was used, but this time only headlines were extracted. Generated sequence length was set to $M = 12$ tokens, prompt length to $P = 30$, so that most headlines and most prompts fit into those lengths.

The generator $\mathcal{G} = \mathcal{G}_{in} \circ \mathcal{G}_{out}$ consists of two LSTM networks, each with 128 units. The input and output embedding dimensions were set to 64 (input and output embeddings shared the same trainable weights)[10].

First, $\mathcal{G}$ was pretrained to simply read input sequence and copy it to the output. The goal of this experiment was to verify that the selected architecture is at all capable of encoding and decoding several entities.

During pretraining phase, $\mathcal{G}$ has obtained very high accuracy of output tokens (over 0.98). However, during adversarial training with only one discriminator (NER), the reward was constantly decreasing (vide Figure 7.12).

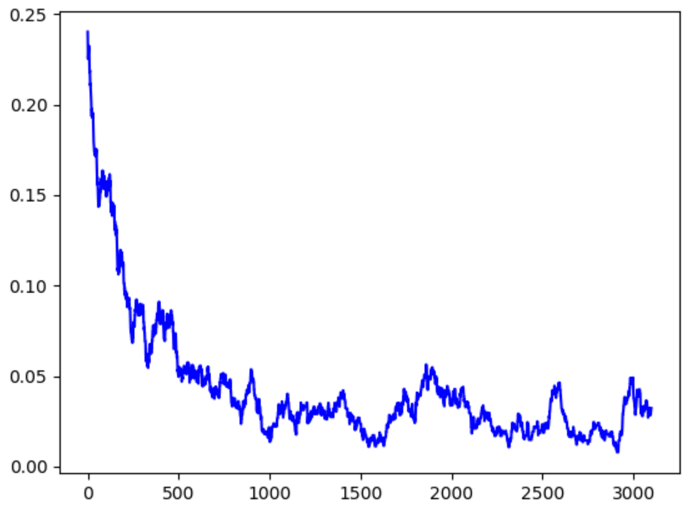


Figure 7.12: Decreasing F-score during RL phase.

Despite this unpromising result, one more experiment was conducted. $\mathcal{G}_{in}$ was again fed the control code – a set of entities – but $\mathcal{G}_{out}$ was trained to output a headline in natural language. During adversarial training, three metrics were tracked: BLEU, F-score and final reward (defined as $\frac{1}{2}\mathcal{D}_{BLEU} + \frac{1}{2}\mathcal{D}_{NER}$).

[9] In the first experiments, instead of providing v at each timestep to $\mathcal{G}_{out}$, the hidden state of $\mathcal{G}_{out}$ was simply initialized to v. Unsurprisingly this aproach was ineffective, since the recurrent net was responsible not only for updating its hidden state, but also remembering v.

[10] Experimens with larger dimensions were not performed due to limitations of the GPU memory.

Results were disappointing (see Figure 7.13) as the F-score was again decreasing over time. This is not very surprising – the network learns to condition output on the set of input entities, in other words it learns a set of rules *if entity E occurs in the input, it must also occur in the output.* Intuitively, each rule is learned separately for each entity. Since the velocity of providing new samples to the network is much lower during adversarial/reinforcement learning phase than it is during much faster MLE pretraining, the metrics go down. What is indeed surprising and much harder to explain is why the BLEU metric was also decreasing in this experiment, since all previous experiments have shown that it is relatively easy to optimize for the generator.

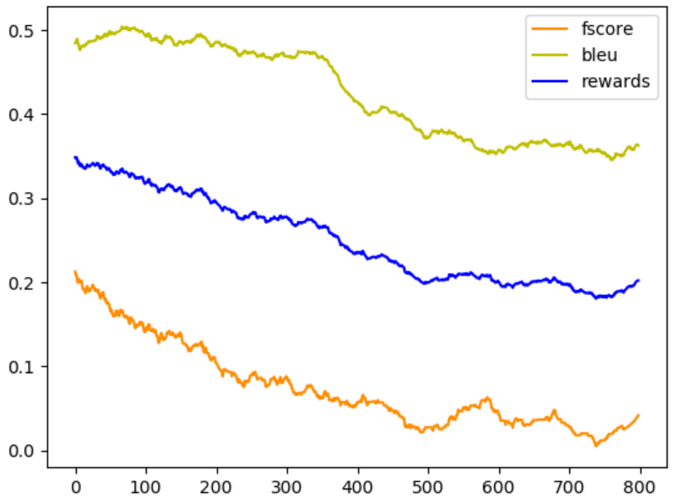


Figure 7.13: Metrics measured during the RL phase.

In theory, the freedom offered by the adversarial/reinforcement settings to optimize any arbitrary objective could provide many opportunities for controllable generation. Experimental results hint that traditional MLE setups for training are, if not more effective, at least easier to make use of.

Chapter 8

EDGAR: Out-of-the-box solution

From the experimental repository[1], a set of key functionalities was extracted to build an out-of-the-box solution for prompted generation – a tool named EDGAR (**E**ntity **D**riven **G**PT-2-based **A**utomatic w**R**iter) hosted at `https://github.com/tomaszgarbus/edgar`.

EDGAR consists of 3 main components:

- clone of the gpt-2-simple repository
- NER module described in chapter 4
- a set of scripts binding the two together

There are also two demo Colab notebooks:

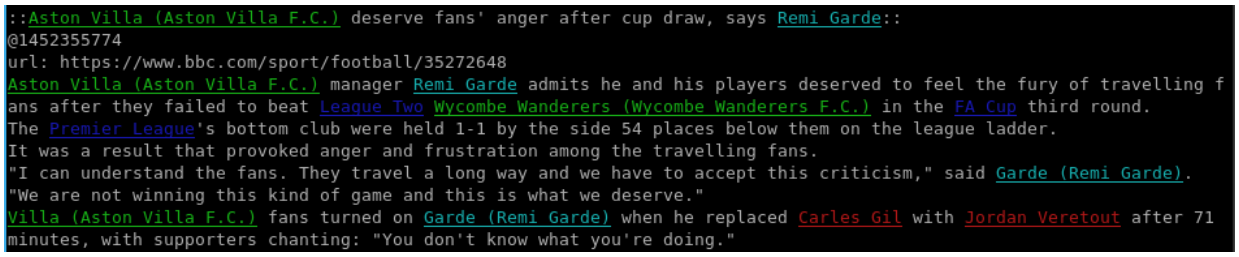
- E2E demo Colab notebook demonstrating how to use EDGAR on E2E dataset ([24]) to increase generated text compliance with control codes (~ 0.94 F-score versus ~ 0.85 obtained by raw GPT-2 model)
- football demo Colab notebook, which uses the provided football config Json. In this notebook, EDGAR achieves ~ 0.52 F-score while raw GPT-2 scores only ~ 0.25 (although the model is highly underfitted for the sake of reasonable cells execution time).

8.1. Usage

highlight_demo.py

A demo script for the Highlighter class. User is expected to provide an input path and a Json config file. The script outputs the samples from input path in random order, highlighting the entities:

[1] `https://gitlab.com/tomasz.garbus1/experimental`
This repository contains most of the work described here (except for the online quizes and Scraper tool).



```
::Aston Villa (Aston Villa F.C.) deserve fans' anger after cup draw, says Remi Garde::
@1452355774
url: https://www.bbc.com/sport/football/35272648
Aston Villa (Aston Villa F.C.) manager Remi Garde admits he and his players deserved to feel the fury of travelling f
ans after they failed to beat League Two Wycombe Wanderers (Wycombe Wanderers F.C.) in the FA Cup third round.
The Premier League's bottom club were held 1-1 by the side 54 places below them on the league ladder.
It was a result that provoked anger and frustration among the travelling fans.
"I can understand the fans. They travel a long way and we have to accept this criticism," said Garde (Remi Garde).
"We are not winning this kind of game and this is what we deserve."
Villa (Aston Villa F.C.) fans turned on Garde (Remi Garde) when he replaced Carles Gil with Jordan Veretout after 71
minutes, with supporters chanting: "You don't know what you're doing."
```

Figure 8.1: Output of the `highlight_demo.py` script.

```
usage: highlight_demo.py [-h] -i I -c C

optional arguments:
    -h, --help  show this help message and exit
    -i I        input file or directory
    -c C        config json path
```

build_prompts.py

Creates an annotated dataset.

```
usage: build_prompts.py [-h] -i I -o O -c C [--limit LIMIT]

optional arguments:
    -h, --help     show this help message and exit
    -i I           Dataset input file directory.
    -o O           Prompted dataset output directory.
    -c C           Highlighter config file path.
    --limit LIMIT  Limit of output files. Use this parameter if you want the
                    script to run for a shorter time.
```

finetune.py

Finetunes GPT-2 model on the annotated dataset.

```
usage: finetune.py [-h] -i I [--num_steps NUM_STEPS]
                   [--model_name {124M,355M,774M,1558M}] [--multi-gpu]

optional arguments:
    -h, --help             show this help message and exit
    -i I                   Input corpus directory.
    --num_steps NUM_STEPS
    --model_name {124M,355M,774M,1558M}
            Name of the finetuned model to use. Note that most
            likely you don’t have enough computing power to use
            1558M or 774M.
    --multi-gpu            Whether or not to use multiple GPU cores. This may be
            useful if you have an Tesla K80.
```

generate.py

Generates one or more samples from the provided control codes.

```
usage: Generate samples from finetuned model. [-h] -i I -c C -p P -o O
    [--steps STEPS]
    [--step_length STEP_LENGTH]
    [--samples_per_step SAMPLES_PER_STEP]
    [--multi-gpu] [--limit LIMIT]

optional arguments:
    -h, --help            show this help message and exit
    -i I                  Input directory you have used to train the model.
    -c C                  Highlighter config. Refer to ner/schema.py for the
    JSON schema definition.
    -p P                  Prompts source directory.
    -o O                  Output directory for samples.
    --steps STEPS         Number of generation steps.
    --step_length STEP_LENGTH
    Number of tokens generated at each step.
    --samples_per_step SAMPLES_PER_STEP
    Number of samples to select from at each step.
    --multi-gpu           Whether or not to use multiple GPU cores. This may be
    useful if you have an Tesla K80.
    --limit LIMIT         Maximum number of samples to generate.
```

8.2. Example use: E2E dataset

E2E dataset was selected for an EDGAR demo since it comes with manual annotations and it's fairly easy to build configuration file for. This dataset is also a good example that even if the data is already annotated, the NER module of EDGAR can still be benefited from during generation.

An example use of EDGAR (on E2E dataset, following the Colab demo) looks as follows:

Build a configuration JSON

EDGAR is configured by providing a JSON file following the schema:

```
schema = {
"type": "object",
"required": ["categories"],
"properties": {
    "categories": {
        "type": "array",
        "description": "Categories of entities",
        "items": {
            "type": "object",
            "required": ["name", "entities"],
            "properties": {
                "name": {
                    "type": "string",
                    "description": "Category name."
                },
                "entities": {
                    "type": "array",
                    "description": "Entities belonging to the category.",
                    "items": {
                        "required": ["name", "variants"],
                            "name": {
                                "type": "str",
                                "description": "Base variant of the entity name."
                            },
                            "variants": {
                                "type": "array",
                                "description": "Other variants of the entity name.",
                                "items": {
                                    "type": "str"
                                }
                            }
                        }
                    }
                }
            }
        },
        "case_sensitive": {
            "type": "boolean",
            "description": "Case-sensitive entity name matching.",
            "default": True
        }
    }
}
```

An example configuration file is provided at configs/football.json. The user must hand-pick a set of categories of entities that they want to condition the generation on. On E2E dataset, this step is particularly simple, because all texts are already annotated with a set of properties: `area, familyFriendly, food, eatType, priceRange, name, near, customer rating`.

Optional: Evaluate the Highlighter

`Highlighter` is a central class to the NER component. It executes the algorithm 1 to detect entities in texts and builds prompts. A helper function `evaluate` in evaluate_highlighter.py prints to standard output three metrics for each category and for all categories altogether: precision, recall and F-score.

Annotate the dataset

Next we annotate the dataset by prepending a prompt to each text. An example execution of the `build_prompts` script:

```
$ python build_prompts.py -i trainset_nlr.txt -c configs/e2e_v2.json -o train_data -limit 1000
```

where:

- `trainset_nlr.txt` is the input path. The input path may either be a single file (then each line is a separate text) or a (potentially multi-level) directory (then each file is a separate text).
- `configs/e2e_vs.json` is the config JSON produced for E2E dataset.
- `train_data` is the output directory.
- `--limit 1000` is used to set the maximum number of output files. For a very large dataset it makes little sense to wait for a long time to get it annotated, if we do not plan on fine-tuning the model for that many steps.

Finetune GPT-2

After building prompts we are ready to finetune the model:

```
$ python finetune.py -i train_data -num_steps 1000
```

Generate samples

Samples can be generated either by a call to the `generate` function from generate.py or by running a script:

```
$ python generate.py -i train_data -o gen_samples -c configs/e2e_v2.json -p dev_prompts -limit
        10 -steps 5 -step_length 6 -samples_per_step 5
```

The parameters in the above command are:

- `-i train_data` is the path to an input directory that the model was finetuned on. This value is used to retrieve the correct model checkpoint (as there may be potentially multiple cached models).
- `-o gen_samples` is the output directory.
- `-c configs/e2e_v2.json` is, again, the config json.
- `-p dev_prompts` is the input directory with prompts. One sample will be generated for each file in `dev_prompts` (unless `--limit` is provided).
- `--limit 10` sets the maximum number of samples to generate. If provided, prompts are selected randomly with uniform distribution.
- `--steps 5` sets the number of generation steps (vide algorithm 2)
- `--step_length 6` sets then number of tokens to output in each generation step.
- `--samples_per_step 5` sets the number of variants generated in each step, from which the one with highest F-score is selected.

Chapter 9

Conclusions

Controlled text generation is an area of rapid development in deep learning. Most efforts made to gain control over generated content can be categorized into three main paradigms:

- Template-driven generation, where templates can be either hand-engineered or machine-generated.
- Control codes, an intuitive extension of the language modelling problem, where the token probability distribution is conditioned not only on previous tokens, but also on the control code. In fact, any language modelling system capable of generating a continuation of text, could be trained for conditional generation by simply pretraining on a dataset prepended by control codes.
- Adversarial training and reinforcement learning techniques or combination of the two.

Each paradigm could be explored in more detail before any definitive statements are made about one's superiority over another. However, the presented experiments point to the following conclusions:

- Control codes are a quick way to build a text generation system that produces realistic-looking samples, even more so if one uses a pretrained language model.
- Generation of templates requires domain-specific knowledge. In the online quiz experiment, template-based samples were found to be much easier to distinguish from human-written articles than the ones produced with only control codes.
- The REINFORCE algorithm gives much more freedom in specifying the goal for our system to optimize. There are however some downsides:
  - Not all objectives are equally easy to optimize. If the objective is defined by a discriminator that keeps on learning, even a positive reward may be hard to obtain.
  - The RL setup can act like an evil genie who exploits loopholes in one's wish. For instance, if BLEU score is being optimized, the generator may find that a rare sequence of consecutive `<unk>` tokens scores high and repeat it over and over.
  - Most reported results from RL or adversarial based architectures are on very short texts. To author's knowledge, the only RL-based system generating long samples of high quality is GPT-2 finetuned on human responses ([37]).

We must also recognize that the toy domain of football articles comes with its own characteristics:

- relative repetitiveness and limited number of facts to be said, compared to e.g. politics

- large number of distinct entities, relations between which change over time

that are not representative of all possible domains.

Readers are encouraged to try out the control-code-driven generation using EDGAR repository on various domains.

Bibliography

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *ArXiv*, abs/1406.2661, 2014.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[3] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[4] Mohammad Shoeybi, Mostofa Ali Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053, 2019.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL `https://arxiv.org/abs/2005.14165`.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003. URL `http://jmlr.org/papers/v3/bengio03a.html`.

[7] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270. URL `https://doi.org/10.1162/neco.1989.1.2.270`.

[8] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181. URL `https://doi.org/10.1109/72.279181`.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[10] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings*

of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL `https://doi.org/10.3115/v1/d14-1179`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.0473`.

[13] Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. GLTR: statistical detection and visualization of generated text. In Marta R. Costa-jussà and Enrique Alfonseca, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28 - August 2, 2019, Volume 3: System Demonstrations*, pages 111–116. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-3019. URL `https://doi.org/10.18653/v1/p19-3019`.

[14] Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. Neural text generation: Past, present and beyond. *CoRR*, abs/1803.07133, 2018. URL `http://arxiv.org/abs/1803.07133`.

[15] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. URL `http://dl.acm.org/citation.cfm?id=645920.672836`.

[16] Danuta Bartol-Jarosińska, Stanisław Dubisz, Jerzy Podracki, Józef Porayski-Pomsta, and Elżbieta Sękowska. *Nauka o języku dla polonistów.* Książka i Wiedza, Warszawa, 1998.

[17] Karen Kukich. Design of a knowledge-based report generator. In Mitchell P. Marcus, editor, *21st Annual Meeting of the Association for Computational Linguistics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, June 15-17, 1983*, pages 145–150. ACL, 1983. doi: 10.3115/981311.981340. URL `https://www.aclweb.org/anthology/P83-1022/`.

[18] Susan Weber McRoy, Songsak Channarukul, and Syed S. Ali. YAG: A template-based generator for real-time systems. In Michael Elhadad, editor, *INLG 2000 - Proceedings of the First International Natural Language Generation Conference, June 12-16, 2000, Mitzpe Ramon, Israel*, pages 264–267. The Association for Computer Linguistics, 2000. doi: 10.3115/1118253.1118293. URL `https://www.aclweb.org/anthology/W00-1437/`.

[19] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. Choosing words in computer-generated weather forecasts. *Artif. Intell.*, 167(1-2):137–169, 2005. doi: 10.1016/j.artint.2005.06.006. URL `https://doi.org/10.1016/j.artint.2005.06.006`.

[20] Blake Howald, Ravikumar Kondadadi, and Frank Schilder. Domain adaptable semantic clustering in statistical NLG. In Katrin Erk and Alexander Koller, editors, *Proceedings of the 10th International Conference on Computational Semantics, IWCS 2013, March 19-22, 2013, University of Potsdam, Potsdam, Germany*, pages 143–154. The Association for Computer Linguistics, 2013. URL `https://www.aclweb.org/anthology/W13-0113/`.

[21] Johan Bos. Wide-coverage semantic analysis with boxer. In Johan Bos and Rodolfo Delmonte, editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings, Venice, Italy, September 22-24,*

2008. Association for Computational Linguistics, 2008. URL https://www.aclweb.org/anthology/W08-2222/.

[22] Hans Kamp and Uwe Reyle. *From Discourse to Logic - Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, volume 42 of *Studies in linguistics and philosophy*. Springer, 1993. ISBN 978-0-7923-1028-0. doi: 10.1007/978-94-017-1616-1. URL https://doi.org/10.1007/978-94-017-1616-1.

[23] Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. Learning neural templates for text generation. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3174–3187. Association for Computational Linguistics, 2018. doi: 10.18653/v1/d18-1356. URL https://doi.org/10.18653/v1/d18-1356.

[24] Jekaterina Novikova, Ondrej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Saarbrücken, Germany, 2017. URL https://arxiv.org/abs/1706.09254. arXiv:1706.09254.

[25] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015. URL http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.

[26] Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. CTRL - A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint arXiv:1909.05858*, 2019.

[27] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1edEyBKDS.

[28] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018. URL http://arxiv.org/abs/1803.07416.

[29] Martin Popel and Ondrej Bojar. Training tips for the transformer model. *Prague Bull. Math. Linguistics*, 110:43–70, 2018. URL http://ufal.mff.cuni.cz/pbml/110/art-popel-bojar.pdf.

[30] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1171–1179, 2015. URL http://papers.nips.cc/paper/5956-scheduled-sampling-for-sequence-prediction-with-recurrent-neural-networks.

[31] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *CoRR*, abs/1610.09038, 2016. URL `http://arxiv.org/abs/1610.09038`.

[32] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL `http://arxiv.org/abs/1609.05473`.

[33] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751. ACL, 2014. doi: 10.3115/v1/d14-1181. URL `https://doi.org/10.3115/v1/d14-1181`.

[34] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

[35] Kevin Lin, Dianqi Li, Xiaodong He, Ming-Ting Sun, and Zhengyou Zhang. Adversarial ranking for language generation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3155–3165, 2017. URL `http://papers.nips.cc/paper/6908-adversarial-ranking-for-language-generation`.

[36] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In *AAAI*, 2017.

[37] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *CoRR*, abs/1909.08593, 2019. URL `http://arxiv.org/abs/1909.08593`.

[38] Sandeep Subramanian, Sai Rajeswar, Francis Dutil, Chris Pal, and Aaron C. Courville. Adversarial generation of natural language. In Phil Blunsom, Antoine Bordes, Kyunghyun Cho, Shay B. Cohen, Chris Dyer, Edward Grefenstette, Karl Moritz Hermann, Laura Rimell, Jason Weston, and Scott Yih, editors, *Proceedings of the 2nd Workshop on Representation Learning for NLP, Rep4NLP@ACL 2017, Vancouver, Canada, August 3, 2017*, pages 241–251. Association for Computational Linguistics, 2017. doi: 10.18653/v1/w17-2629. URL `https://doi.org/10.18653/v1/w17-2629`.

[39] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL, 2002. URL `https://www.aclweb.org/anthology/P02-1040/`.

[40] William Fedus, Ian J. Goodfellow, and Andrew M. Dai. Maskgan: Better text generation via filling in the _______. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=ByOExmWAb`.

[41] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1511.06732`.