Generating fake websites: WikiGen

Michael Longland

A report submitted for the course
COMP3740 - Project Work in Computing
The Australian National University

June 2020
Except where otherwise indicated, this report is my own original work.

Michael Longland
12 June 2020
Acknowledgments

I would sincerely like to thank my supervisors David Liebowitz and Alex Antic for their help and guidance through this project. It’s been fantastic working with you and learning about such an interesting topic.
Abstract

Models for generating natural language have shown significant improvements recently with the introduction of the Transformer architecture, however attention has been focused on only generating single sequences of text. We propose a model we call WikiGen, which incorporates generative models for text and graphs to generate realistic ‘mini-Wikipedias’. WikiGen generates fake Wikipedia articles and a fake link network with similar topology to the real Wikipedia link network, then assigns articles to nodes in an optimal manner. We also show that a language model finetuned on Wikipedia text can generate entirely fake Wikipedia articles that closely resemble real ones. Articles generated by this model are coherent, often entirely novel, and follow the structure of real Wikipedia articles. A demo, along with code and pretrained models, is published on GitHub. ¹

¹https://github.com/longland-m/wikigen
## Contents

**Acknowledgments** vi

**Abstract** vii

1 **Introduction** 1
   1.1 Motivation 1
   1.2 Report Outline 1

2 **Background and Related Work** 3
   2.1 Background 3
   2.2 Related Work 4
      2.2.1 Text Generation 4
      2.2.2 Graph Generation 4

3 **Design and Implementation** 7
   3.1 Data Collection 7
   3.2 Article Generation 9
   3.3 Link Network Generation 10
      3.4.1 Generating a Fake Link Network and Articles 12
      3.4.2 SIF Embedding 13
      3.4.3 Pairwise Cosine Similarities 13
      3.4.4 Article:Node Assignment 14

4 **Article Generation** 15
   4.1 Article Structure 15
   4.2 Consistency and Long-Range References 16
   4.3 References Sections 16

5 **Network Generation** 19
   5.1 Evaluation 19

6 **Fake Mini-Wikipedia Generation** 21
   6.1 SIF Embeddings: Real vs Fake Articles 21
   6.2 SIF Embeddings: Linked vs Non-Linked Articles 23

7 **Discussion and Future Work** 25
   7.1 Discussion 25

---

Final Copy – 12 June 2020
7.2 Future Work .................................................................................. 26

A Example Outputs ........................................................................... 29
A.1 Example Fake Articles ................................................................. 29
A.2 Example Fake Networks ............................................................... 33
A.3 Example Mini-Wikipedia .............................................................. 36

B Independent Study Contract .......................................................... 39

C Artefact Details .............................................................................. 43
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>The distribution of SIF similarities of the introductory sections of 250 fake Wikipedia articles (blue) and 250 randomly selected real articles (red).</td>
<td>22</td>
</tr>
<tr>
<td>6.2</td>
<td>The distribution of SIF similarities for all pairs of introductory sections in the 20,500 Wikipedia article dataset. Blue shows the distribution for articles that have a link between them, red shows the distribution for articles that are not linked.</td>
<td>23</td>
</tr>
<tr>
<td>A.1</td>
<td>Example 1 of a graph from the training data (top) and a similar graph generated by our large GRAN model (bottom).</td>
<td>34</td>
</tr>
<tr>
<td>A.2</td>
<td>Example 2 of a graph from the training data (top) and a similar graph generated by our large GRAN model (bottom).</td>
<td>35</td>
</tr>
<tr>
<td>A.3</td>
<td>Example 1 of a mini-Wikipedia generated by Wikigen. Nodes are labelled with the title of the article assigned to them. Edge labels indicate the SIF similarity between the articles.</td>
<td>36</td>
</tr>
<tr>
<td>A.4</td>
<td>Example 2 of a mini-Wikipedia generated by Wikigen. Nodes are labelled with the title of the article assigned to them. Edge labels indicate the SIF similarity between the articles. The egocentric node is visible on the middle-right, 'Bournemouth Battery'.</td>
<td>37</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Statistics of samples generated by GRAN compared to the training set, for both model sizes. For all metrics, the smaller the better.</td>
<td>19</td>
</tr>
</tbody>
</table>
Introduction

1.1 Motivation

While Transformer models for text generation such as GPT-2 are a very active topic of research, there has been little attention paid to using or finetuning them for generating structured documents and analysis of the structure of the generated documents against the real ones.

Additionally the current Transformer models are unable to be directly applied to the case of generating multiple linked structured documents as well.

The generation of fake Wikipedia-style and HTML documents in particular is a tractable case to study. It’s an interesting and useful problem in its own right, but it is also an important topic in the area of cyber deception. This problem in cyber deception is based around using real documents (from e.g. webpages, or a shared folder) and generating related but fake documents that:

- Don’t leak sensitive info contained in the real documents
- Appear realistic enough to fool an intruder, at least until they are scrutinised

These fake documents would be deployed alongside real documents, to help protect information in the event that an intruder accesses especially sensitive parts of a network. Legitimate users would know which documents are real, and ignore them, while intruders would be forced to waste time guessing which document/s are real. Combining with an intrusion detection system that sets off an alert when the fake documents are accessed adds even more protection.

To our knowledge there has also been no previous work studying generating fake websites, fake Wikipedias, or more generally, generating sets of fake documents linked together with some sort of structure.

1.2 Report Outline

This report is structured as follows. In chapter 2 we discuss the background to this problem and review related work. In chapter 3 we present the design of WikiGen
and the background to the major components, including collecting and building the dataset used for evaluation. Chapters 4 and 5 discuss and evaluate the generation of fake Wikipedia articles and fake Wikipedia link networks respectively. Following this in chapter 6, we analyse the use of the fake Wikipedia articles and link network to build mini-Wikipedias. In chapter 7, we summarise our results and list possibilities for future research. Finally in Appendix A, we show example outputs from the two generative models and example mini-Wikipedias produced with our method.
In this chapter we discuss the background and some related work on the types of generative models used in this paper. Section 2.1 introduces some recent developments in text generation and graph generation, particularly transformer models for text generation. Section 2.2 discusses a number of models for text and graph generation, both simpler traditional models and more complex models which use deep learning.

2.1 Background

The Transformer architecture has revolutionised language models in the last few years. It was first proposed in 2017 in the paper by [Vaswani et al., 2017]. Transformers greatly improve on previous architectures for NLP tasks, improving parallelisation and overall efficiency, and allowing much larger datasets to be used on with much larger model sizes. Prior to transformers the dominant architecture for language models was recurrent neural network (RNN) based.

Transformers are made up of stacks of encoder and decoder blocks. These blocks are each made up of a feed-forward neural network layer and self-attention layer. Words (or ‘tokens’ as they’re better described here) are passed through the self-attention then the neural network layer of each block. The first layer turns the word into a vector (‘embeds’ it) and from then on the output of each block is used as input to the next one.

The exact encoder and decoder block structures usually differ slightly, and details differ between different models. Both are not strictly necessary. For example, BERT [Devlin et al., 2018] is an encoder-only Transformer.

The success of self-attention in NLP tasks has led to its application being tested in other domains such as graph generation [Liao et al., 2019]. Similarly to NLP, it improves on the state of the art over an RNN-based model.

However the primary focus of these Transformer models has been based around sequences of text. To our knowledge, there has not been any significant attention put
on generating structured documents or websites.

## 2.2 Related Work

### 2.2.1 Text Generation

As mentioned above, BERT is an encoder-only Transformer model introduced in [Devlin et al., 2018]. It’s a bidirectional model, so it is best suited to sequence-to-sequence tasks like translation.

Transformer-XL is another a Transformer model, introduced in [Dai et al., 2019]. It is a decoder Transformer which focuses on long-term context, improving on the original Transformer architecture by incorporating a segment-level recurrence mechanism and an improved positional encoding scheme.

One test done with Transformer-XL was training it just on the WikiText-103 benchmark, constructed for and first introduced in [Merity et al., 2016]. This is a dataset made up of Wikipedia articles rated ‘good’ (by Wikipedia editors) or have been ‘featured’. It was shown to be capable of generating parts of Wikipedia articles. Analysis was done on completion of articles, for example in one experiment the authors gave it the first 500 tokens of a real Wikipedia article as context and told it to generate 500 more. They compared the model output to the text of the real article. The difference between this paper and this test of Transformer-XL is that the authors did not analyse the structure of the generated text, nor did they try to generate entire Wikipedia articles starting from the beginning.

GPT-2 is yet another Transformer model introduced in [Radford et al., 2019]. It is also a decoder Transformer, and is primarily used for generating text. It is autoregressive in nature, generating words one at a time and using that in the context to generate the next word.

### 2.2.2 Graph Generation

**Traditional models.**

Prior to the widespread use of deep models for generating graphs, many simpler methods were relied on. The earliest and most famous graph generative model dates back to 1960 in [Erdős and Rényi, 1960], called the Erdős-Rényi random graph model. This model constructs a graph by adding an edge between every pair of nodes with probability $p$. The ‘small world’ model by [Watts and Strogatz, 1998] generates networks with high clustering and low diameter. This is a property seen in some empirical networks. The Barabási–Albert preferential attachment model [Barabasi and Albert, 1999] generates ‘scale-free’ networks, which have a degree distribution following a power law. These networks have the probability of a node having degree $k$ following $P(k) \sim k^{-\gamma}$. We call $\gamma$ called the degree exponent, and usually $2 < \gamma < 3$. Networks generated by the Barabási–Albert model have degree exponent $= 3$. Many
empirical networks have power law degree distributions, including Wikipedia. In 2017 the estimated degree exponent for the entire English Wikipedia link network was 2.21 [KONECT, 2017].

Exponential random graph models (ERGMs) model graph structure from a given set of sufficient statistics. These statistics are network attributes, such as the mean degree, total number of edges, clustering coefficient, and so on. In practice these sufficient statistics are often based off another observed network.

These traditional models all have significant limitations however. The first 3 all make large assumptions about structural properties of the network, for example the degree distribution or clustering coefficients, and leave little room for flexibility. They only model one or two significant network attributes and leave little variation in others. For example, the Barabási–Albert model generates graphs with a power-law degree distribution but is unable to include any community structure. On the other hand, the Watts-Strogatz model allows customising the mean degree and amount of local clustering, but the degree distribution has very little flexibility, all nodes will generally be of a similar degree. ERGMs allow significantly more flexibility than any of the first 3 models, however they are still limited by the fact they rely on a small number of (usually) hand-crafted sufficient statistics. To be clear, it only learns from these statistics, it doesn’t learn from the entire structure of a given network. These statistics are also insufficient to model more intricate and complex network structures.

Deep models.

Deep learning has recently made its way into the field of graph generation. Deep learning architectures allow for generative models to model significantly more complex graph structures than the traditional models.

GraphRNN [You et al., 2018] is a deep auto-regressive graph generative model which uses recurrent neural networks (RNNs) to model graphs as sequences. It achieved state-of-the-art results on a number of metrics and is able to model arbitrary graphs - it’s not restricted to a single domain or class of graphs. GraphRNN allows use of multiple graphs for training data. It was the first to be able to use multiple graphs, and also have them be large graphs. Other previous deep learning approaches could only train on one graph, or could only train on multiple graphs if they were small (approx. <40 nodes). GraphRNN was tested on networks up to 2000 nodes. However the sequential representation of graphs has a limitation. Nodes that are close in the graph may be far apart in the adjacency matrix and therefore generated far apart. In this case the performance of RNNs may drop as it fails to attend to those nodes as well as it should.

GRAN is another auto-regressive generative model Like the language models described above, it uses the attention mechanism. While it still has the same issue of node ordering possibly affecting graph generation quality, it significantly improves on RNN-based methods. It’s able to generate much larger graphs than GraphRNN. In addition it outperforms GraphRNN on most of the metrics it was tested with.
Because of this clear performance improvement, we use GRAN in this paper.
In this chapter we will explore the background and design of the process of generating fake mini-Wikipedias, and the evaluation of it using a custom-built dataset of Wikipedia articles. Section 3.1 will discuss the data collection and preprocessing for modelling. Section 3.2 will briefly discuss the structure of Wikipedia articles then outline the method and model used to generate fake articles. Section 3.3 will outline the use of subsets of Wikipedia link networks for modelling and generating fake link networks. Finally, section 3.4 will show how the outputs of sections 3.2 and 3.3 are both used to make fake mini-Wikipedias.

3.1 Data Collection

This project used Wikipedia articles as a simplified version rather than full HTML webpages. We believe these serve well as a simplification as articles are required to have common structure which follows Wikipedia’s rules, and this structure is not overly complex. In addition, data from Wikipedia is easily collected using its API. We collected and curated a custom dataset from Wikipedia for this problem. No known public dataset suited our purposes. The findings from modelling with Wikipedia articles can easily extend to documents with more complex structure in future work, i.e. HTML documents.

Data was collected from the category, subcategory and subsubcategory pages from ‘Military of Australia’, ‘Military of New Zealand’, and ‘Military of Canada’, and the category and subcategory pages from ‘Military of the United Kingdom’ and ‘Military of the United States’. Selection of categories was somewhat arbitrary, the primary reason was that they were the first type of category found that had a sufficiently large number of pages through them and their subcategories, and whose pages generally didn’t stray very far from the central topic. Collected from each page was:

- The text content
- The list of other Wikipedia pages it links to

This was approximately 30,000 articles in total, after duplicates were removed. Nat-
urally, two datasets were made from this, one consisting of the text content and the other the list of links.

As will be described in section 3.2, fake articles generated with GPT-2 are limited to a total length of 1024 words. Articles longer than this (including section headings, references, etc) were removed from the dataset so that the model will learn to generate full articles in a single sample. Similarly, articles shorter than 50 words were also removed as we would prefer the model not generate extremely short samples. This leaves us with 20,500 articles. The dataset has a total of 7,968,934 words, giving an average of just under 400 words per article. It is interesting to note that the distribution of article word counts approximately follows a power law, starting above around 60 words.

Next, each article was prepended with the text == Article Start ==, a line break, then the article title and three more line breaks. They were also appended with <\endofdocument>. This is so there is a clear and unique signal of the beginning and end of articles which the model can learn. All articles were appended together to create a 49MB text file for the first dataset. Note that Wikipedia uses a lightweight markup for article structure:

- == [name] == denotes the start of a section titled [name]
- +++ [name] +++ denotes the start of a subsection titled [name]
- ++++ [name] ++++ denotes the start of a subsubsection titled [name]

This markup style is where inspiration for prepending articles with == Article Start == came from. The <\endofdocument> token is a special token used in the original training data of GPT-2 to separate documents and we have kept it here. This also provides an advantage over datasets such as WikiText-103, used by models such as Transformer-XL (see chapter 2), which has no such token. WikiText-103 wraps article titles in single equals signs, and has no end of text token. If this is not changed during preprocessing, there is no text to use as context which would guarantee starting an article from there. The best possibility would be the text ‘\n= ’, but it is believed this would be inconsistent. In addition, the average article length in WikiText-103 is over 3000 words, which is significantly longer than the samples GPT-2 can generate. Since we focus on article structure in this report, rather than pushing the limits of long-term context in transformers, we ignored this dataset and don’t consider it any further.

Websites can naturally be represented as a directed graph, with nodes representing pages and edges representing a link from one page to another. For the second dataset, the list of links was formatted as an edgelist, specifically, a list of tuples: (from article title, to article title). Many articles outside those in the 20,500 collected are linked to. Entries in the edgelist where an outside article is linked to were removed. We did not collect the text content of those outside articles and as such it would not be useful to use them in our network.

Using the NetworkX Python package, the edgelist was finally converted to a graph.
format. The model that will be used with this data, GRAN, can’t take in extra attributes like node names (i.e. article titles) so these were removed.

### 3.2 Article Generation

For generating fake articles we used GPT-2 [Radford et al., 2019]. Along with the GPT-2 paper, OpenAI released four pretrained model checkpoints, varying by model size. More specifically, the models were trained on the same dataset but differ by the number of parameters and the structure of their transformer blocks. GPT-2 was trained on a large, diverse corpus of English text gathered from websites. Because of the diversity of training data these base models generally generate sentences and/or paragraphs without structure, and certainly without any sort of common structure between samples. This is why finetuning the pretrained models is necessary. Finetuning is also significantly faster and less resource-intensive than training a model from scratch.

The two smallest pretrained GPT-2 models were finetuned on our Wikipedia article dataset introduced in section 3.1. These models have 124M and 345M parameters respectively, and are known as GPT-2 Small and GPT-2 Medium. Finetuning larger models required more resources than were available so these weren’t able to be tested.

We finetuned the pretrained models for 10,000 steps with a learning rate of 0.00002 and a batch size of 2. For both model sizes, training loss stopped decreasing after around 9500 steps.

Following finetuning, both models consistently generated full, but entirely fake, Wikipedia articles. More analysis of generated samples is in chapter 4.

Samples from the models were generated using only ==Article Start== as context, and ended once an <\endoftext> token was generated. The idea is that the model learns to generate articles given this token, with a title first, introductory section, then the rest of the article. Separating different articles with just the <\endoftext> token and using this as context to generate new articles is sufficient in theory. However it was found also including ==Article Start== to begin articles, and using it as context for generating articles instead, yielded better and more consistent performance in practice. The exact reason for this is unknown and remains an open question.

The article samples generated from the 345M model were significantly better than those from the 124M model. They had noticeably more complexity in vocabulary and structure, more consistently generated complete articles (i.e. produced an <\endoftext> token before the 1024 token limit), and articles were more ‘creative’ overall. The only advantages of the 124M model over the 345M model were that finetuning and sampling from the model were around twice as fast, and saved models are smaller, around 0.5GB versus 1.3GB. That the finetuned 345M model performs significantly better than the finetuned 124M model is quite unsurprising.
The 345M model was used for all experiments in the rest of this paper. We use an evaluation dataset of 250 article samples from this finetuned model. Consistent with above, samples were generated using only \texttt{Article Start} as context. To also keep consistent with the training data, samples were discarded if they contained less than 50 words, or did not contain the \texttt{endoftext} token. This was uncommon however, only 6 samples had to be discarded before reaching 250 accepted samples.

### 3.3 Link Network Generation

For generating fake link networks we will use GRAN \cite{liao2019gran}. GRAN is a recently published deep graph generative model which generates new graphs similar to those it was trained on one node (or block of nodes) at a time. It incorporates Graph Neural Networks (GNNs) with attention to better capture structural features, especially in larger networks, over methods such as Recurrent Neural Networks (RNNs). GRAN is able to train on multiple graphs, which is a large advantage over many previous and traditional graph generation models which could only use one. This is especially important for this problem because we find that there’s large variability in the local network structure between different subsets of the Wikipedia link network.

Our second dataset in its current form is too large to be useful. We wish to generate much smaller link networks with a range of different possible structural features. We use subgraphs of the full link network as training data for GRAN. The sampling strategy for getting subgraphs is to use order-1 egocentric networks (or ego networks for short). To be specific, these graphs are sampled by randomly selecting a central node, then taking all the nodes it links to, and all the edges that exist between all these nodes.

Egocentric network sampling was chosen for this problem because it allows us to easily take many (usually) non-overlapping samples. The power-law degree distribution in the larger network helps this further, it means we have many lower-degree nodes to select from. Egocentric samples also preserve local network structures, which is desired for our training data.

Two model sizes were tested, a small model which used ego networks with between 15 and 25 nodes, and a large model which used ego networks with between 26 and 100 nodes. We did not use networks over 100 nodes in size as the ego networks above approximately 100 nodes were often found to overlap with each other significantly, which would add unwanted bias to the training data. Central nodes for training networks were selected at random from the full network and added to the training set if the size of its ego network was within the appropriate range (i.e. between 15-25 or 26-100) and it did not already exist in the training data. The small model was trained on 50 networks and the large model on 150 networks. Memory limits prevented a larger dataset from being used for the large model.
Model setup. The configuration for both the small and large models is largely identical. We use a block size of 1 (one node generated each step), a batch size of 1, a learning rate of 0.0001, and run the models for 1000 epochs. We use depth-first search (DFS) node ordering starting at the central node. This means the first column of the adjacency matrix will always be filled with ones (other than the top corner, there’s no self-edges). This consistency helps the model more easily learn to generate only egocentric networks. The small model used embedding and hidden dimensions of 128, 7 GNN layers, and 20 Bernoulli mixture components. The large model used embedding and hidden dimensions of 64, 3 GNN layers, and 15 mixture components. Again, this was due to running out of GPU memory.

The smaller model size doesn’t appear to have any significant negative effect, as we show in chapter 5 it’s still sufficient for generating high quality samples.

Directed networks. We also investigated an extension to GRAN to generate directed networks, but ultimately weren’t happy with the quality and did not use it in this report. This is why training data and generated networks are undirected from here. Directed networks would be much more preferable due to the inherently directed nature of between-page links in Wikipedia. The proposed change was to generating directed samples by generating two graph samples one after the other, to be the upper and lower triangle of the adjacency matrix. This was suggested in the GRAN paper as an extension to make it directed. We disagree with this over-simplification, while the upper and lower triangle of the adjacency matrix for our types of graphs are not independent. Simply running GRAN twice ignores these dependencies entirely. Samples generated did not resemble the training data very well.

We evaluate the generated samples by following the evaluation method used in both in [Liao et al., 2019] and [You et al., 2018]. That is, we use Gaussian kernels with earth mover’s distance in the maximum mean discrepancy (MMD) (see Borgwardt et al. [2006]) over the following graph statistics:

- Degree distribution
- Clustering coefficient distribution
- The number of occurrences of all orbits with 4 nodes
- The spectrum of the normalised graph Laplacian

In addition we compute the proportion of generated graphs that are not egocentric. This is because want to know whether the model has successfully learned this structure, particularly because it is present in all the graphs in the training data.

This analysis is presented in chapter 5.

---

1 We used a GPU with 4GB memory
3.4 Mini-Wikipedia Generation

With fake articles and a fake link structure we can now generate a fake mini-Wikipedia. The main idea here is finding an ‘optimal’ mapping articles to nodes in the link network. The overview of our method is:

1. Generating a network with \( N \) nodes using our GRAN model
2. Generating \( N \) articles with our finetuned GPT-2 model
3. Applying smoothed inverse frequency (SIF) embedding applied to all articles, representing each article \( i \) as a vector \( x_i \in \mathbb{R}^{300} \)
4. Calculating the cosine similarity between all pairs of embedded articles
5. Finding an optimal assignment of articles to nodes with a greedy algorithm to maximises the total similarity score \( S \), defining this as the sum of the cosine similarity between all linked nodes

While we only focus on one method to generate a mini-Wikipedia in this paper, many others are possible and may be explored in future work.

3.4.1 Generating a Fake Link Network and Articles

Steps 1 and 2 are relatively straightforward. Using GRAN, a link network with \( N \) nodes is generated, for some value of \( N \) between 15 and 100 depending on the
model size. Then with GPT-2, $N$ articles are generated, using == Article Start == for the context as usual.

### 3.4.2 SIF Embedding

Next, we look at making embeddings of all the article introductory sections using SIF, introduced by [Arora et al.] [2017]. This is a fast method of sentence embedding we are extending to paragraphs for this problem. While SIF was written for the problem of embedding single sentences, we believe it can be used on text significantly longer than single sentences for this problem with no modifications. We only require a measure of which articles are closer to each other, compared only to the others in the set, not a perfectly accurate similarity measure. That is to say, we primarily care about preserving the relative similarity scores rather than the magnitude, and this is the case for this measure. Although the authors of SIF briefly mention performance drops when embedding longer texts, many manual checks of SIF’s results on article embeddings showed it was perfectly ok. We also only embed the article introductory sections rather than the entire article, since this provides a reasonable summary of all the article content.

The main idea behind SIF is similar to the well-known term frequency–inverse document frequency statistic, in that it assigns less weight to more frequent words - which is where the ‘inverse frequency’ part comes from. SIF uses a pretrained word embedding file, such as one made using word2vec [Mikolov et al., 2013] or GloVe [Pennington et al., 2014] as a base, and then applies a weighting scheme over the texts to generate embeddings for all sentences (or paragraphs) under consideration.

For this problem we use the same word vectors as the SIF authors, the PARAGRAM-SL999 (PSL) word vectors from [Wieting et al., 2015]. This is a relatively small vector file. An alternative considered was the Common Crawl (840B tokens) vectors from Stanford NLP’s GloVe [Pennington et al., 2014] but memory issues meant the smaller file was used instead. For word frequencies we use a document with word frequencies from the Wikipedia corpus as of 2012, which was created and used by the SIF authors in their evaluation.

When performing the embeddings, all paragraphs are fed to SIF in a single step so the word distributions throughout all articles can be captured. We use a weighting parameter of 0.001 for all experiments in this report.

### 3.4.3 Pairwise Cosine Similarities

We next calculate the cosine similarity between all pairs of vectors:

$$similarity = \frac{x_i \cdot x_j}{\|x_i\|_2 \|x_j\|_2}$$

for articles $i$ and $j$. 
3.4.4 Article:Node Assignment

The problem of finding the optimal assignment of articles to nodes to maximise the sum of the cosine similarity across edges can be written as a case of the famous quadratic assignment problem (QAP) [Loiola et al., 2007], which is well known to be NP-hard. We formulate our optimisation problem as:

$$\text{maximise } \text{tr}(MXAX^\top)$$

subject to $X \in \Pi_n$ (3.1)

where $M$ is the matrix of pairwise article SIF similarities, $A$ is the adjacency matrix of the fake link network, and $\Pi_n$ is the set of $n \times n$ permutation matrices.

We will rely on a greedy similarity maximisation heuristic algorithm to find a good assignment of articles to nodes. This algorithm is as follows.

**Greedy similarity maximisation:**

1. Randomly assign articles to nodes, calculate total similarity $S$
2. For each pair of nodes, calculate the new value of $S$ if they were to be swapped
3. Keep the swap which increases $S$ the most
4. Repeat (2)-(3) until no possible swap further increases $S$
5. Repeat (1)-(4) a prespecified number of rounds $R$, and select the assignment corresponding to the highest value of $S$ found

Many other approximation methods are known (see Loiola et al. [2007]) but this greedy algorithm runs quickly enough and performs sufficiently well for this problem, even for the largest link networks with 100 nodes, so no others were tested.
In this chapter we analyse the outputs of our finetuned GPT-2 model. We present some examples of generated articles in Appendix A.1 for reference. Immediately it is clear that the model samples are structurally similar to the real Wikipedia articles it was trained on. In section 4.1 we discuss this structure, especially the ordering of sections, at a high level. We also show that the model generates entirely novel content. In section 4.2 we look at the consistency of text within articles, and in particular the frequent correctness of long-range references. Finally in section 4.3 we focus on the last sections of Wikipedia articles (References and External links) and show the model also learns the specific structure of these.

4.1 Article Structure

We first look at the samples generated at a high level, comparing them to real Wikipedia articles. Refer to appendix A.1 for examples of generated articles.

Articles generated by the model generate the text content for fake Wikipedia articles in full. It corrects starts with a title and introductory section, then follows the normal Wikipedia structure of body sections and subsections, and ends with references sections. The model also learns to generate references and external links sections at the end of the article, and only at the end, before the `<endoftext>` token.

Sections and subsections are designated through the Wikipedia markup described in chapter 3. The model clearly learns this perfectly.

Section and subsection names can be seen to be plausibly related to the article title and introductory section at a glance, and this only becomes more clear as the rest of the article is read. This is important in the context of cyber deception as realistic looking text without realistic looking structure will be noticed much more quickly and easily. For example, sections in an article about a person would be related to life and career-related topics (if not those words exactly), and sections in an article about a type of boat would be related to the design, history, and use.

Articles are frequently entirely novel and ‘imagine’ new people, new events etc.
the occasion that the title generated is the same as an article that does exist in the training data, there will rarely be any information leaking from the real one into the generated one. Note that this was only noticed on a handful of occasions during evaluation and is likely partly due to chance.

Generating an article with the same title multiple times will almost certainly give entirely different topics and structure each time.

4.2 Consistency and Long-Range References

Next we comment on the consistency and long-range reference ability of the model. The model is frequently able to correctly reference facts it stated earlier in an article, such as the date or date range of an event.

Text inside sections and subsections is consistent with both the section heading and the article overall. It is able to stay on topic and generate text relevant to the title, introductory section, and previous sections.

Very interestingly, it learns to generate events in correct orders. Refer to the ‘Military Career’ section in the first example article in appendix A. The paragraph not only refers to the career events for this fictional person in chronological order, the promotions the person receives are in the actual order of seniority for the Canadian Army (albeit with one error).

4.3 References Sections

To finish up, we turn our focus to the references and related sections at the end of Wikipedia articles. We will look at the ‘References’ and ‘External links’ sections that exist in many articles. Note that there also exist related ‘Citations’ or ‘Bibliography’ sections in some articles. These are largely the same as ‘References’ so we do not consider them separately here.

Some examples of these sections are in appendix A.1. We generated these samples differently to previous methods. The context was == References == and == External links == respectively.

The model generates realistic looking book references and URLs in these sections. It is very rare for one of these books or links to actually exist, during testing only two links tested out of over fifty led to real pages. URLs sometimes point to real websites, but have subpages and/or query parameters in them that mean the link does not lead to a valid part of the site. In other circumstances it may invent a new website altogether. Of interest is the fact it has learned to generate links from the Internet Archive, web.archive.org. The links here are essentially made up of two URLs, the base Internet Archive link, and the full link of the archived page.
Another interesting point to make is related to giving the model the context == References == or == External links ==. Specifically, the fact it generates the references and external links correctly from this shows it hasn’t learned that books and URLs come at the end of articles, it’s learned that books and URLs appear in those sections and those sections only. Note that this was expected behaviour, but worth mentioning.
In this chapter we evaluate the link networks generated by GRAN for both the small and large models. In section 5.1 we test the generated graphs against the training sets in a number of local and global graph properties.

5.1 Evaluation

As mentioned in section 3.3, we evaluate the performance of the trained GRAN model using Gaussian kernels with earth mover’s distance in the maximum mean discrepancy (MMD) over distributions of graph statistics. These statistics are the degree distribution, clustering coefficient distribution, the number of occurrences of all orbits with 4 nodes, and the spectrum of the normalised graph Laplacian. In addition we compute the proportion of samples which are not egocentric.

We generated an evaluation set for each model as large as the training set, that is, for the small model we evaluate on 50 samples and for the large model we evaluate on 150 samples.

See appendix A.2 for some examples of these generated graphs, and examples from the training data with similar structure.

Table 5.1 shows the statistics for both models. Lower values are better for all statistics. The numbers are very comparable to those in table 1 of the GRAN paper. This table is

<table>
<thead>
<tr>
<th></th>
<th>Small Model</th>
<th>Large Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree</td>
<td>0.0394</td>
<td>0.0139</td>
</tr>
<tr>
<td>Clustering</td>
<td>0.1117</td>
<td>0.0448</td>
</tr>
<tr>
<td>Orbits</td>
<td>0.1201</td>
<td>0.0433</td>
</tr>
<tr>
<td>Spectrum</td>
<td>0.0404</td>
<td>0.0127</td>
</tr>
<tr>
<td>Ego</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Table 5.1: Statistics of samples generated by GRAN compared to the training set, for both model sizes. For all metrics, the smaller the better.
where the authors present these same statistics from GRAN and other models trained on the graph datasets they selected. We also see that the models have generated egocentric networks in every single sample.

One very interesting point to note is that all the statistics for the large model are better than those for the smaller model. Recall that the larger model had a smaller embedding size, fewer GNN layers, and fewer mixture components than the smaller model. We suspect that the numbers would be closer if the smaller model was trained on more graphs. However this was not investigated and the exact reason remains an open question.

Manual inspection shows the samples aren’t identical to any graphs in the training data for either model, clearly it has not overfit at all.

We can conclude from this experiment that GRAN is able to generate suitable fake egocentric link networks.
Fake Mini-Wikipedia Generation

In this chapter we will evaluate and justify our process of mini-Wikipedia generation. We will analyse distributions of similarity scores, first comparing between real and fake articles in section 6.1 then comparing between real articles that are linked and those that aren’t linked in section 6.2.

The primary idea in this chapter is to justify the selection of SIF-similarity maximisation for linking articles. We do this first through comparing the similarity distribution of real and generated articles. If the distributions are sufficiently close, this is evidence that the type of article introductory sections the model generates are semantically similar to the real ones. Following this, we compare the similarity distribution of articles that are linked against that for articles that aren’t linked. A strong correlation between high introductory section SIF similarity and a link existing between the two pages may be used as justification for our method of creating mini-Wikipedias. The desired result of both these steps is, in fact, precisely what we find in our experiments.

6.1 SIF Embeddings: Real vs Fake Articles

For this section we use our 250 generated articles and a randomly selected sample of 250 real articles. SIF embedding of the two sets of articles was done separately.

We see in figure 6.1 that the distributions are essentially identical. Both real and fake article similarities are centred around zero, with shape somewhat resembling that of a normal distribution. We notice a longer tail in the positive values, the highest similarities go close to 1.0 but the lowest only to around -0.6.

It is possible that the bias in the sample of 250 real articles affects the results but this is not believed to be the case. We ran this experiment four additional times with different random article selections to test for this bias. The real article distribution barely changed in any of these runs.
Figure 6.1: The distribution of SIF similarities of the introductory sections of 250 fake Wikipedia articles (blue) and 250 randomly selected real articles (red).
§6.2 SIF Embeddings: Linked vs Non-Linked Articles

Now the second step, to justify the method of linking articles based off their SIF similarities. We used all articles in our 20,500 article dataset in this step. The similarity between all pairs of articles was calculated in a single step and the scores split into two sets afterwards. The first set contains all similarity scores for articles that are linked and the second set contains the scores for articles that are not linked.

We see in figure 6.2 that there is an enormous difference between the similarity distributions of linked and non-linked articles. The mean similarity of linked articles is 0.447 with a median of 0.431. The similarity of non-linked articles is centred around zero, with a mean of 0.011 and median of 0.003.

An interesting point of note is the ‘spike’ in the linked article distribution at exactly 1.00. Investigation of this shows it is due to a single group of articles. These articles are all linked to one another, and have introductory sections that are 2 sentences long and are nearly identical, only differing in a single word, an ID of letters and
numbers. These IDs are out-of-vocabulary for the model, and in the rare case the model encounters an out-of-vocabulary word it assigns no weighting to it. Therefore these articles all have similarities of 1.00. In any case, this does not affect the overall mean or similarity distribution of linked articles significantly. The other point to clarify in this is why the spike does not appear in figure 6.1. This is due to the figure taking only a sample of 250 articles, not all 20,500. If all articles were used it may be noticeable however.

The probability distribution of non-linked articles is far less interesting. It appears very similar to the distribution in figure 6.1 which is largely expected. That is, it is fairly symmetric, resembles the normal distribution, and has no ‘spikes’ or other stand-out features like the linked article distribution.

As mentioned at the beginning of this chapter, we use the results of the two steps described in section 6.1 and this section as justification for our mini-Wikipedia generation method. We see the distribution of similarity scores in sets of real and fake articles are nearly identical. We see there is a very strong correlation between articles having a high similarity score and them being linked. We use this as evidence that maximising SIF similarity is a reasonable method for determining which articles should be linked in fake mini-Wikipedias.
Discussion and Future Work

7.1 Discussion

In this report we investigated the use of multiple deep learning models to create a system to generate fake mini-Wikipedias.

We created two custom datasets, one with Wikipedia text and the other with between-page Wikipedia links. These datasets were curated to best fit with the text generation and network generation models respectively.

We analysed the capability of GPT-2 to generate realistic but fake Wikipedia articles. We found that it is suitable and highly capable in this task, able to generate documents with structure resembling real Wikipedia pages. Especially of note is its capability of learning the different structure in different types of sections, such as generating book titles and URLs in references and external links sections.

Next, we analysed the generation of fake link networks using GRAN. We used egocentric subgraphs of the second dataset we created, the whole between-page link network. We trained GRAN on sets of these egocentric networks and generated samples. The structure of samples is very close to that of the training data. In particular, we see it’s able to generate networks with the multiple types of structures seen in the training data, without being biased towards a single one. Finally, we used the GPT-2 and GRAN models together to produce fake mini-Wikipedias. We used SIF embeddings to determine similarity of articles, and used the sum of this across all edges as a metric to link articles together. Discovering that finding the optimal solution is NP-hard, we used a greedy heuristic algorithm to find a ‘good’ solution.

The generation of fake structured documents and fake mini-Wikipedias is a relevant area right now. Penten, a cyber security business based in Canberra, is currently using the WikiGen model and code produced in this project.
7.2 Future Work

In this report we investigated link network and structured document generation to create fake mini-Wikipedias. Future work could extend this to text with more structure and a more complex markup, such as HTML webpages. This would allow us to create a full fake website. Adding to this, these pages could also include fake images or other media embedded in the pages. Including images based off context in the page text could add more credibility to the faked webpages. Using newer language models and analysing the improvements they bring to generated document structure and consistency would be an interesting area to investigate. Many more state-of-the-art have been released since GPT-2, and many of these focus in particular on long term context in the samples.

Simultaneously generating the link network and text content is a highly relevant area for future work. Initially simultaneous generation of the network and article titles would be the aim, and later on extend to the link network and full articles. This initial step was investigated in this project but ultimately discarded due to time constraints. The idea here was to modify the GRAN architecture to incorporate parts of GPT-2, or another language model. This would sequentially generate the network structure and article titles on each node, conditioned not only on the network structure but the titles of neighbouring nodes. If this were successful, the only next step would be to generate the remainder of the articles using the article start token and the article title as context for each.

Comparing the results in this paper to results using a different dataset of Wikipedia articles, especially on an entirely different topic, would be interesting. It is possible that the dataset used here contains biases that mean all results may not necessarily generalise exactly to all other categories of Wikipedia articles.

Finally, analysing the performance of the greedy similarity maximisation algorithm is another point of interest. It would be useful to understand how many runs are required to get a ‘good’ solution (or how close it gets to the globally optimal solution) and how this varies with network size. Additionally, understanding the worst-case behaviour would be beneficial.
Bibliography


Appendix A

Example Outputs

In this appendix we will show some examples of fake generated Wikipedia articles, fake link networks, and how they look when put together to make a fake mini-Wikipedia. In all 3 sections the examples shown here are among the first few we generated, there is little to no cherry-picking.

A.1 Example Fake Articles

Example 1 - full articles:

== Article Start ==

Albert Le Roux

Albert Le Roux (August 14, 1879 – July 16, 1961) was a Canadian historian and political activist. He served in the Canadian Infantry during the First World War.

== Early life and education ==

Le Roux was born in 1879 in Ridgeway, Ontario, the son of Alphonse Le Roux. As a young man, Le Roux attended Mount Allison School in Toronto until his father left the family when Le Roux was about twelve. He was subsequently educated at Osgoode Hall and Mount Allison College.

== Military career ==

Le Roux entered the Canadian Infantry in the Second Canadian Expeditionary Force in 1916. Le Roux was commissioned in the men’s 1st Battalion in January 1917, and was seconded to the 2nd Battalion. Le Roux was promoted to lieutenant in November 1917 and captain in November 1918. In 1917, Le Roux was in command of a battalion in the Italian Campaign.
Le Roux was promoted to captain in May 1919, and was present at the Battle of the Somme when Thomas Hopkinson’s Battalion was sent to the front as reinforcements. Le Roux was also present at the Battle of the Castine, Battle of Saint-André-sur-Richelieu, Battle of the Somme, Battle of the Loire and Battle of Ligny-sur-Seull. He was next awarded the Croix de Guerre with Bars. He was also promoted to brigadier-general in March 1921. Le Roux was promoted to colonel in August 1921. Le Roux served as adjutant general of the Canadian Corps in the First World War. He was also promoted to major general in November 1922.

== Personal life ==

Le Roux lived in Toronto, Ontario, until his death in 1961 at age 61. He was married to Aline Toplinden.

== Article Start ==

HMCS Wellington

HMCS Wellington (pennant J291) was a River-class frigate that served in the Royal Canadian Navy from 1960 to 1963. Built by Wellington Shipyards and Schoellkopf Zeppelin factory in Hamburg, Germany, Wellington was launched in October 1963 as HMCS Wellington. The frigate was commissioned on 6 May 1964.

== Design and description ==

The River-class frigate was designed by William Reed of Smith’s Dock Company of South Bank-on-Tees. Originally called an "eight-gun flare gun", the frigate would have been equipped with a single 32-gun high explosive flare launcher and a single 40-mm smoothbore gun mounted aft. This arrangement would have allowed the ship to carry a greater number of missiles and rockets. The River-class frigate was larger than her predecessor and would have carried more ammunition. A new vertical launch tube and improved cooling and propulsion would have made the ship faster and more maneuverable. Rendering: The ship’s vertical launch tube could be seen through the water.

== Service history ==

After trials, Wellington was ordered into service with the Royal Canadian Navy as HMCS Wellington on 3 July 1964. The frigate was commissioned on 6 May 1964. The following year, the frigate transferred to the Royal Canadian Mounted Police and later to Canadian Forces Base (CFB) Winnipeg and remained there until she was paid off on 22 January 2000, nearly 22 years after she was built.
Example 2 - References section:

== References ==


Example 3 - Two external links sections:

== External links ==


https://navalairmuseum.org/


== External links ==

http://www.AcademyReports.com/SecuritiesDivision.php
http://www.scribd.com/doc/16207070/Andrew-Fascist-German-Nationalism-in-
http://www.youtube.com/watch?v=Vttvt4nCTIY

<|endoftext|>
A.2 Example Fake Networks

See next page for examples. These network examples both come from the large model. In both examples we show a graph from the training data and a graph generated by GRAN that has similar structure.
Figure A.1: Example 1 of a graph from the training data (top) and a similar graph generated by our large GRAN model (bottom)
§A.2  Example Fake Networks

Figure A.2: Example 2 of a graph from the training data (top) and a similar graph generated by our large GRAN model (bottom)
A.3 Example Mini-Wikipedia

Figure A.3: Example 1 of a mini-Wikipedia generated by Wikigen. Nodes are labelled with the title of the article assigned to them. Edge labels indicate the SIF similarity between the articles.
Figure A.4: Example 2 of a mini-Wikipedia generated by Wikigen. Nodes are labelled with the title of the article assigned to them. Edge labels indicate the SIF similarity between the articles. The egocentric node is visible on the middle-right, ‘Bournemouth Battery’.
Appendix B

Independent Study Contract
INDEPENDENT STUDY CONTRACT
SPECIAL TOPICS

Note: Enrolment is subject to approval by the course convenor

SECTION A (Students and Supervisors)

UnilD: U5352303
SURNAME: Longland FIRST NAMES: Michael
TOPIC SUPERVISOR (may be external): David Liebowitz
FORMAL SUPERVISOR (if different, must be an RSCS academic): Alex Antic
COURSE CODE, TITLE AND UNITS: COMP4770, Project Work in Computing 3740, Topics in Computer Science, 6 units

SEMESTER ☒ S1 ☐ S2 YEAR: 2020

TOPIC TITLE:
Modelling and Generating Fake Websites for Cyber Deception

LEARNING OBJECTIVES:
Explore the modelling and generation of realistic HTML documents using machine learning techniques to build models of page and site structures, connections between structures, and document content.

DESCRIPTION:

- Learn structure of document graphs, for example with deep graphs like GraphRNN
- Sample to create novel documents
- Generate text content from topic words or phrases using one of the recent Transformer based language models like GPT-2 or XLNET
- Optionally: Include other fake media, e.g. images or video

Final Copy – 12 June 2020

Research School of Computer Science
ASSESSMENT (evaluated by the Topic Supervisor, unless stated otherwise here)

<table>
<thead>
<tr>
<th>Assessed project components</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>45%</td>
<td>29/5/2020</td>
<td>Examiner</td>
</tr>
<tr>
<td>Artefacts</td>
<td>45%</td>
<td>29/5/2020</td>
<td>Supervisor</td>
</tr>
<tr>
<td>Presentation</td>
<td>10%</td>
<td>29/5/2020</td>
<td>Convener</td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfil the above defined contract:

................................................................. 26/02/20
Signature                                      Date

SECTION B (Supervisor):

I am willing to supervise and support this proposal. I have checked the student's academic record and believe the student can fulfil this contract. If I have nominated an examiner above, I have obtained their consent (via signature below or attached email)

Alex Antic................................................................. 24 Feb 2020.............
Signature                                    Date

Examiner:.................................................................
Name: ................................................................. Signature

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course convenor approval)

................................................................. .................................
Signature                                      Date
Artefact Details

The artefact for this project is a software repository, available at [https://github.com/longland-m/wikigen](https://github.com/longland-m/wikigen). The major files and folders in the repository are:

- **example_run.ipynb**: a demo of the WikiGen process. Running this notebook in full will produce the fake mini-Wikipedia. Written from scratch.
- **requirements.txt**: a list of the packages and versions required to run WikiGen. Written from scratch.
- **setup.sh**: a shell script to download the word vector file used in SIF embeddings. Written from scratch.
- **eval**: a folder containing the sets of generated samples from GPT-2 and both GRAN models used in model evaluation.
- **gpt2**: a folder containing all the necessary code for generating fake articles with GPT-2.
  - **accumulate.py**: accumulates/optimises gradients. From GPT-2 repo, with no changes.
  - **encode.py**: encodes a provided text dataset for use with GPT-2. From GPT-2 repo, with no changes.
  - **encoder.py**: functions for encoding words. Primarily used for encode.py. From GPT-2 repo, with no changes.
  - **interactive_conditional_samples.py**: generates conditional samples (i.e. where context may be provided to start a sample) from a GPT-2 model checkpoint. From GPT-2 repo, with many changes.
  - **load_dataset.py**: loads an encoded dataset. From GPT-2 repo, with no changes.
  - **memory_saving_gradients.py**: optimises gradients when training a model. From GPT-2 repo, with no changes.
  - **model.py**: functions for training/finetuning the Transformer model. From GPT-2 repo, with no changes.
  - **sample.py**: functions used in generating samples. From GPT-2 repo, with no changes.
- **gran**: a folder containing all the necessary code for generating fake link networks with GRAN.
- **gran_sampler.py**: generates graph(s) from a trained GRAN model. Adapted from GRAN repo, with major changes.
- **config**: a folder to store trained GRAN models and their configuration files.
- **dataset**: a folder containing the code for GRAN data functions. From GRAN repo, with minor changes.
- **granmodel**: a folder containing the code for the GRAN mixture Bernoulli model. From GRAN repo, with minor changes.
- **runner**: a folder containing the code for training and testing GRAN models. From GRAN repo, with minor changes.
- **utils**: a folder containing helper functions for different parts of GRAN
  * **arg_helper.py**: helper functions for command line arguments and loading configuration files. From GRAN repo, with minor changes.
  * **data_helper.py**: helper functions for loading and preprocessing graphs. From GRAN repo, with minor changes.
  * **data_parallel.py**: helper functions for data parallelisation. From GRAN repo, with no changes.
  * **dist_helper.py**: helper functions for computing distributions of graph statistics, for testing GRAN models. From GRAN repo, with minor changes.
  * **eval_helper.py**: helper functions for computing the MMD between graph statistics distributions. From GRAN repo, with no changes.
  * **logger.py**: functions for logging when training and testing models. From GRAN repo, with no changes.
  * **orcamodule.cpp**: module for computing the orbits statistic when evaluating GRAN models. From GRAN repo, with no changes.
  * **setup.py**: sets up the orca module for computing orbits. From GRAN repo, with no changes.
  * **train_helper.py**: helper functions for loading and snapshotting models when training. From GRAN repo, with minor changes.
  * **vis_helper.py**: helper functions for visualising graphs generating in testing. From GRAN repo, with minor changes.
- **sif**: a folder containing all the necessary code for using SIF to generate article embeddings, and assigning articles to nodes.
  - **greedy_sim_max.py**: code to run the greedy similarity maximisation algorithm to assign articles to nodes. Written from scratch.
  - **sif_src**: selected code from the SIF repository necessary to generate SIF embeddings of paragraphs and efficiently compute similarities. Very significant modifications were made to the original SIF code for this file.