A Comprehensive Study of State-of-the-Art Word Embedding Algorithms for Natural Language Generation

Adrian Wesek[1]

aw678@kent.ac.uk

Masters by Research in Computer Science

University of Kent
United Kingdom

October 15, 2019

[1] Supervised by Dr Marek Grzes

Contents

1 Abbreviations

- NLP: Natural Language Processing
- NLG: Natural Language Generation
- NLU: Natural Language Understanding
- NLM: Neural Language Modelling
- LM: Language Modelling
- MLM: Masked Language Model
- biLM: bidirectional Language Model
- SCB: Statistical Count-Based
- SG: Skip-Gram
- CBOW: Continuous-Bag-of-Words.
- MT: Machine Translation
- ULMFiT: Universal Language Modelling Fine-tuning
- FNN: Feedforward Neural Network
- CNN: Convolutional Neural Network
- RNN: Recurrent Neural Network
- LSTM: Long Short-Term Memory Networks
- ELMo: Embedding from Language Model
- BERT: Bidirectional Encoder Representations from Transformers

2 Introduction

Static word vector representations, such as word2vec embeddings (Mikolov et al., 2013b), are widely used in the industry to accelerate the performance at all kinds of Natural Language Processing tasks, by numerically mapping the vocabulary. The main limitation of such fixed vector encoding algorithms is that they do not have a way of representing polysemous words, i.e., a word that can have multiple meanings in different contexts. Recent language modelling approaches solve the polysemy issue by using a transformer architecture (Vaswani et al., 2017), that is based on an attention mechanism, to model bidirectional word representations that are context-dependent (Devlin et al., 2019). In contrast to word2vec vectors, the contextual embeddings are dynamically computed as a function of a pre-trained model trained on extensive collections of data. Therefore, the trend of mapping words to the fixed vectors starts to be replaced with language models that produce unique contextual representations on the go. Meaning that, instead of using word vectors at the input layer to represent the text, we now build on top of a language model by adding a layer that we fine-tune on a specific task. Since the recent language models are pre-trained on large datasets, the fine-tuning layer approaches satisfying performance faster, i.e., thanks to the grammatical and conceptual information captured at the pre-training phase. A recent successful pre-trained bidirectional language model (BERT) has proven the power of task-specific fine-tuning by pushing the SuperGLUE[1] benchmarks at eleven Natural Language Understanding tasks. These superior results motivate us to conduct experiments on the interpretability of contextual word embeddings from BERT, as well as on sequence generation capabilities.

As it turns out, by analysing the contextual word vectors, it is possible to perform arithmetic operations on the representations, in order to derive new and conceptually related words. However, in contrast to word2vec, the process is more complicated, and the results may not be as reliable because

[1] Link to SuperGLUE leaderboard: https://gluebenchmark.com/leaderboard

of the embedding dynamic composition nature. Furthermore, it is not clear how to use BERT as a generative model because of the bidirectional training objective function that cannot be reused for output sampling; therefore, a successful strategy remains as an open research question. Consequently, we utilise an alternative model called GPT2 to perform further generative experiments. The GPT2 is based upon the same architecture as BERT, however, trained with a unidirectional objective that makes the output sampling strategy sequential, and therefore, straightforward to sample from. Furthermore, in contrast to BERT, the GPT2 model displays a lower performance at understanding tasks; however, sets strong benchmarks at many generative tasks[2].

Given the ultimate objective of this thesis to explore state-of-the-art models for text generation, we get inspired by the GPT2 benchmark achievements and decide to use it to create a conditional lyrics generative model, i.e., by fine-tuning on lyrics dataset and utilising special tokens to encode the style of a specific genre. After training, the model can style lyrics to a specific genre, as well as can be conditioned on any text sequence. Taking this idea a step further, we train the GPT2 model using lyrics dataset accompanied by the following features: genre, year, author, and song name. Training the model using the data mentioned above and a specific input feeding strategy allows for more fine-grained control over the style of the lyrics. For example, we can produce a blended output based on a country genre, a rock author and the '50 style. Furthermore, given only a subset of conditional metadata, we can generate lyrics, which we can later use to generate any of the missing features.

This work presents the following main contributions:

- Experimentally showing how to fine-tune a pre-trained language model for controllable output generation. This is achieved by an appropriate input construction strategy, as well as an optimal output sampling

[2] Link to benchmarks of many different generative tasks: https://paperswithcode.com/area/natural-language-processing/text-generation

method.

- Showing that a dataset with additional features (metadata) positively influences the quality and diversity of the conditional model outputs.
- Confirm that it is not convenient to generate from BERT in a sequential fashion, given context from left-only, right-only, and both sides.
- Illustrate that manipulating BERT bidirectional token representations using simple arithmetic operations result in new representations that conserve common-sense knowledge.

The thesis has the following structure (see Page 2 for the content table). The background (Section 3) covers the essential knowledge needed to understand the inner-workings of models used for the experiments. In essence, it starts by introducing recent trends and advances in the Natural Language Processing field. Then, it dives into the most fundamental concepts and methods of the field, i.e., what a word embedding is, standard count-based and neural-based approaches of language modelling. Moreover, it covers the recent state-of-the-art language modelling algorithms, e.g. ELMo and BERT, and their composition architectures, that include, LSTM and self-attention. Throughout the whole section, it is shown how the approaches to language modelling change in order to produce embeddings that capture more information about themselves.

Furthermore, Section 4 contains preliminary experiments on the BERT model. The tests start by comparing the performance of two BERT distributions, BERT-Base and BERT-Large, at their training Masked Language Modelling objective. Further experiments focus on the contextual word embedding interpretability. In particular, by inspecting their representations, performing an arithmetic operations, and seeing how the results relate to the way human interpret concepts. Lastly, we conduct experiments on language generation from BERT, using the same objective that was utilised during the pre-training phase.

In Section 5, we cover the methodology, experiments, and results of using a GPT2 model for conditional lyrics generation. In particular, this Section involves experiments on different training dataset sizes, input construction strategies and the most optimal output sampling techniques. Moreover, further experiments on a fine-grained generative model contrast the output qualities based upon weights trained using datasets that vary in data diversity and the dataset size. Also, other experiment compares the output qualities based upon collections generated using different subset of conditional features.

Lastly, Section 6 examines the limitations of the experiments and techniques discussed in this thesis, and expands upon the future work to be conducted.

3 Background

Computers are unable to naturally understand words in the way we humans do, which is because they are fundamentally built to work with numbers. To create a mapping between the natural language and the way computers process things, a mechanism that numerically represents words is required. Differently said, for a system to process, interpret, and disambiguate a textual input, any different word needs a unique and meaningful representation. A standard mechanism for representing a textual input is to use a vector of real numbers - *a Word Embedding.* Similarly to the way a picture is defined by pixels; a word is represented by a vector of values that collectively express a word and the context it is used in. The word embedding should ideally capture both: complex characteristics of word use (grammar and sentiment), and how these uses vary across linguistic context, i.e., to model polysemy.

Word embedding is a crucial component for *Natural Language Processing* **(NLP)** tasks, which significantly contributes to the performance of a wide range of applications by providing fundamental linguistic comprehension. To name a few domain-specific applications on account of word embedding: language translation, sentence classification or text generation. Hence, having an effective mechanism of mapping words to meaningful representations, that capture the context the word is used in, is an essential research area for the natural language processing field today.

Language modelling algorithms have a long and rich history in the NLP field, however, they can be classified into two categories: a *Statistical Count-Based* and the *Neural Language Modelling* approach. At the core of both of these approaches stands a similar *Language Modelling* objective, i.e., to predict a word given the probability of the previous word sequence. The statistical count-based methods, such as the *N-Gram*, involve estimating probabilities via counting appearance frequency (Cavnar, Trenkle et al., 1994). Differently put, the model pre-processes a corpus of text by counting the occurrence frequency of unique n-long sequences (n-grams), and then, using the n-grams' occurrence frequency, estimates the probability of a sequence. The statistical count-based approaches are fast and straightforward. However, some sequences of words can be poorly estimated due to a *Data Sparsity* problem. That is, the chance of same long word sequence to re-appear for the next word prediction, decreases with the length of n-gram, where longer n-grams yield better performance.

Motivated by the fact that words with similar meaning tend to appear in the same context; the neural language modelling approach models the vectors based on the words' context co-occurrence using a neural network (Bengio et al., 2003). This way, a sequence probability can be later modelled based on the individual word vector representations, which solves the data sparsity problem. Mikolov et al. (2013b) proposed two different forms of the vector representation modelling, a *Continuous-Bag-of-Words* and a *Skip-Gram*, which at the time of publication were particularly useful. Both approaches are based on a similar idea of using a shallow feedforward architecture that learns parameters in the training process by predicting n-many context words around an index word. After pre-training the model on a large unsupervised corpus of text, the learnt parameters are used to produce fixed size and dense word embeddings. The produced word embeddings capture grammar and exhibit the property whereby semantically similar words are similar in the vector space. Furthermore, the model's learnt space can be manipulated to find word's relationships by using simple arithmetic operations, e.g., $vec(\text{“}king\text{”})-vec(\text{“}man\text{”})+vec(\text{“}woman\text{”})$ is closest to the $vec(\text{“}queen\text{”})$ than to any other vector (Mikolov et al., 2013a). Despite the great success of accelerating the performance of other neural networks applications, i.e., by using the word embedding at the input layer to represent words, the word vectors still suffer from limitations. The major drawbacks are: inability to represent polysemy; not considering the full contexts of a word; incompetence to represent words that have not been encountered during the training process; and a lack of subword information (*Morphemes*) - words like eventful, eventfully, uneventful and uneventfully should have structurally related embedding in the vector space.

The morphological issue is tackled by FastText, which builds on top of the previous approaches by representing input at the training process in the form of *Character N-Grams* (Bojanowski et al., 2017). This way, the vector space maintains subword structural information, and the model knows how to assign a meaningful representation to the previously unseen words. Furthermore, the limited context problem gets solved by Peters et al. (2018) who used *Long Short-Term Memory* **(LSTM)** neural network architecture and named their model *ELMo*. In more detail, ELMo uses two Long Short-Term Memory networks to scan context from both directions, which produces two outputs for each position in the sequence. By combining the two representations computed in the opposite direction, we derive a *Bidirectional Word Embeddings*. ELMo differs from the previous approaches in the sense that the resulting embeddings are a function of the entire sentence, i.e., the same word can have numerous different internal representations depending on the context it is computed in. By improving the accuracy of multiple benchmark tasks, this work has revealed the importance of *Bidirectional Language Modelling*.

Another impactful work has proposed a *Transformer* network that is solely based on an *Attention Mechanism* (Vaswani et al., 2017). Similarly to ELMo, an attention mechanism is capable of producing word vectors, which are a function of the whole sentence. However, the transformer is naturally more parallel and allows for a much deeper architectural setting than the Long Short-Term Memory network. Furthermore, the transformer has a promising future in replacing the Long Short-Term Memory networks for many tasks since it has shown to maintain long-term dependencies better. One example is the OpenAI GPT model that has applied a unidirectional language modelling objective to the transformer architecture and have improved the benchmarks over ELMo (Radford et al., 2018). The initial transformer network is made up of two parts, an encoder, and a decoder. However, the OpenAI GPT has shown that when using language modelling as a learning function, it is good enough to only use the encoder; it saves computational time. Another work, *ULMFiT* by Howard and Ruder (2018), has proposed a powerful *transfer learning* strategy which aims to *fine-tune* the previously learned (*pre-trained*) model that was trained on a large amount of data. Such an approach benefits applications that have limited learning resources since the pre-trained model already exhibits the fundamental linguistic properties and only has to be fine-tuned to a more specific task. In contrast to word embeddings, fine-tuning does not require building and training a model from scratch. Furthermore, applying ULMFiT to only 100 labelled examples matches the performance of training from scratch on 100 times more data samples, i.e., due to the information captured by the pre-training language model.

Moreover, a different model named *BERT*, at the time of publication has obtained new state-of-the-art results on eleven Natural Language Understanding tasks (Devlin et al., 2019). BERT takes advantage of multiple recent ideas that have emerged in the Natural Language Processing field, namely: OpenAI GPT architecture, ELMo's emphasis on the bidirectional context, and the ULMFiT's fine-tuning strategy. However, to allow BERT for bidirectional learning, the authors have introduced a new *Masked Lan-*

guage Modelling objective. Since the transformer architecture requires to take a whole sentence as an input, the unidirectional objective in the bidirectional fashion would logically not work. That is, the model could attend to all of the words it tries to predict, therefore, resulting in mainly coping the predictions from the input which would not allow for any generalisation. In greater detail, the Masked Language Modelling objective masks 15% of the input sequence that the network has to predict, based on the un-masked context. Nevertheless, inspired by the publication of Dai and Le (2015), BERT uses another objective function to model the learning space jointly with the language modelling objective. The additional inference task requires the network to classify an appropriate continuation of a sentence, given its beginning. The authors claim that the *multi-task learning* improves the network's transfer-learning to tasks requiring a sentence relationship understanding, e.g., question answering and text summarisation. Furthermore, BERT not only provides fundamentals to be fine-tuned to other supervised tasks but also can be used to extract contextualised word vectors.

BERT has proven to be superior over the previous methods for tasks requiring language understanding and has set solid grounds to be widely used in the industry today. However, despite the BERT's strong applicability to tasks requiring knowledge, understanding tasks only stand a subset of problems people want to use such models for. As of today, it is unclear how one could successfully sample from BERT for *Natural Language Generation* tasks because of the new *Masked Language Modelling* objective that is tricky to be re-used for the generation. Thus, the more recent work by Radford et al. (2019) has filled in the gap by training a transformer-based network using a standard language modelling objective, but with increased number of layers, hidden-states, and the training dataset size. The GPT2 model has achieved the state-of-the-art for many generative tasks while being competitive with BERT at understanding tasks. This work has shown the importance of enormously large dataset sizes necessary to improve the performance at transfer-learning tasks.

Language Modelling Algorithms						
Properties: ↓	N-Gram	Word2vec	FastText	ELMo	BERT	GPT2
Word Rep. Type	n-gram	vector	vector	vector	FoaM	FoaM
Context Modelling	✗	partial	partial	full	full	full
Polysemy	✗	✗	✗	✓	✓	✓
Morphemes	✗	✗	✓	✓	✓	✓
Bidirectional	✗	✗	✗	✓	✓	✗
Layers Depth	○ ○ ○	● ○ ○	● ○ ○	● ● ○	● ● ●	● ● ●
Comp. Power	low	low	low	high	high	high

Table 1: A summary of Language Modelling Algorithms' properties. FoaM stands for: the function of a model, and the algorithms with FoaM representation construction property can also produce fixed word embeddings.

3.1 Fundamentals

This section covers the essential background knowledge that is relevant to understand the further explained architecture structures and their use cases. It starts by explaining word embedding, in essence: what it is, what information it captures, and how we can make use of it. Furthermore, the section covers two significant types of training approaches and their hybrid, which gets used for language modelling. Lastly, it discusses how we approach language modelling by setting an objective for an algorithm to capture patterns within the natural language data.

3.1.1 Word Embedding

The input to any neural network model needs a numerical representation in order for the computer to be able to process the input. Intuitively, having a textual input would mean that we need a mechanism which would provide meaningful representations for the respective words. One of the most effective ways of representing words is to use word embeddings. Formally speaking, a word embedding is a mapping between the actual words and their respective dense vector representations. For example, a word could be represented by a vector of length $N = 100$, $[n_0, n_1, ..., n_z]$, where $n_z \in \mathbb{R}$ between $\{-1, +1\}$. Representing words as vectors allow language modelling models to capture grammatical and semantic information by theoretically speaking, permuting the vector values. As a result, vectors with related semantic meaning should have similar internal distribution. For example, consider the following sentences:

- It is really sunny today.
- It was really cloudy yesterday.
- It will be really rainy tomorrow.

The adjective words describing the weather *(sunny, cloudy and rainy)* must have a similar vector distribution because they appear in a similar context, i.e., between the word "really" and the words indicating a particular day. Likewise, the words: *today, yesterday and tomorrow*, must be related since they occur in the context of words indicating the weather. Generally, a word representation can be perceived as a combination of all the contexts it occurred in, for a given textual corpus. Therefore, the more context two words share, the more similar their internal representation will be. Furthermore, an investigation into word embeddings has shown that it is possible to capture interesting linguistic characteristics by manipulating the vector space using linear arithmetic transformations. Meaning, it is possible to use a distance function, i.e., *(cosine similarity)*, to calculate the nearest neighbours for any word. Nevertheless, it is also possible to perform arithmetic operations between two words to derive a new related one. For example, the result of a vector calculation: $vec(\text{“}Poland\text{”}) + vec(\text{“}capital\text{”})$ is close to $vec(\text{“}Warsaw\text{”})$, and $vec(\text{“}King\text{”}) - vec(\text{“}man\text{”}) + vec(\text{“}woman\text{”})$ is closest to the $vec(\text{“}Queen\text{”})$ than to any other vector (Mikolov et al., 2013a).

fWord embedding is a fundamental idea for the *Natural Language Processing* **(NLP)** field due to its essential use at the input layer. It is a common practice to use embeddings to represent a textual input at any other neural network, which, as a result, transfers the common knowledge learned at the word embedding modelling stage to the new model. Furthermore, such a strategy has shown to hugely enhance the accuracy of the neural network models while saving the computational training time. If not the presence of word embeddings, many widely used application today could not provide a reliable and consistent service. To list a few, Google's search and translation services, next word prediction or auto-correction on a mobile device, and spam detection at the email.

However, as it will be explored in greater detail in the further sections, the standard modelling techniques have limitations which leave a footprint on the quality of the final embeddings. One of the primary defects is that such embeddings are context invariant, meaning that they exhibit the same internal distribution no matter the meaningful differences in the surrounding context. To give a better intuition, consider the following two sentences:

- The bank on the other end of the street was robbed.
- We had a picnic on the bank of the river.

Both of the sentences use the word *bank*, but the meaning of the word differs entirely in the two different contexts. This phenomenon is known as *polysemy* and accompanies the word embedding algorithms since the beginning. The standard embedding algorithms struggle to model polysemy since they use a single vector to represent the meaning of the word *bank*. In contrast, the recent algorithms model the word embeddings as a function of the entire sentence and thus, representation of a word changes depending on the words around it. So, the new modelling algorithms would use the above words *robbed* and *river* to disambiguate the meaning of the word *bank*. Furthermore, taking context into account has dramatically increased the performance of many tasks making use of the *contextual word embeddings*.

3.1.2 Model Learning Approaches

The NLP models can benefit from two types of data, structured and unstructured. Depending on the data type, the algorithms require a matching learning strategy, i.e., a mathematical objective function which calculates an error required to optimise the network. At a high level, these learning objectives are split into two groups, *supervised* and *unsupervised.*

The unsupervised approach assumes that a network is given a domain-centred input and is expected to find statistical correlations hidden within the data without any human supervision. In this approach, the network's weights are tuned accordingly to reflect the patterns within the data. A significant advantage of this approach is that the data can remain unlabelled and does not require much human pre-processing. A common belief supported by recent work (Radford et al., 2019) is that using large training sets allows the network to learn more patterns and achieve better generalisation accuracy, and thus, strengthening the significance of unsupervised data learning objectives.

The supervised approach is when each input has its desired output (label) that the network is striving to achieve. For example, consider a binary sentiment classification problem. A network receives sentences as an input that can either have a positive or negative sentiment label. Thus, in this scenario, the network's goal is to recognise whether the input is positive or negative. Returning an output which is other than the expected label, creates an error which is then used to adjust the network's parameters. After *back-propagating* the error, the next time the network receives the same input, it should return an output that is closer to the desired label. In other words, the supervised approach is like having a teacher who points the network to the correct output for each input. Models trained on supervised datasets are generally better since the teacher's signal allows to learn/solve specific problems. However, the creation of a labelled dataset is very time consuming and can lead to an introduction of unwanted human bias into the model.

In contrast, while the supervised approach would classify a sentence positive or negative, its unsupervised counterpart would look at inherent similarities between all the sentences and separate them into groups accordingly. Although both of the learning types sound applicable to this binary classification problem, the unsupervised method often proves to be less accurate on complex data (Patterson and Gibson, 2017).

Luckily for us, there is also a *self-supervised* algorithms' group, which combines the two approaches mentioned above. The self-supervised design benefits from both, using large unlabelled datasets and the teachers signal (label), which is achieved by automating the labelling process on unstructured data. Language models belong to the self-supervised group since their objective function is to predict the next word given the previous sequence of words. Explicitly speaking, self-supervised objectives use the next word in the queue as a label for the current prediction. Section 3.1.3 covers language modelling task to a greater extent. An important take away is that algorithms using this learning objective combines the benefits from unsupervised and supervised approaches and form an intersecting group called self-supervised learning algorithms.

3.1.3 Language Modelling

Most programming languages are deterministic and their expressiveness is limited, in a sense that, a syntactically correct input will always trigger the same predetermined scenario. Capturing linguistic properties in that way is not easy because natural language contains a lot more possible terms, is ambiguous, and changes over time. For example, natural languages have many *homonym* words, i.e., words that have the same spelling but different meanings and origins. In order to capture the linguistic properties, a model needs to make use of a *Language Modelling* **(LM)** learning objective.

In a high level overview, a LM objective is the task of computing a probability of a sequence, $P(W) = P(w_1, w_2, w_3..., w_n)$ and using this probability to predict the upcoming element in the sequence $P(w_4 \mid w_1, w_2, w_3)$. This technique stands at the core of the NLP field due to its extensive application use.

Tasks like text summarisation, question answering, and machine translation require the same predictive objective that the LM provides. Furthermore, LM is also used to model dense vector representations (word embeddings) by encoding grammatical and structural information onto them. Models that take advantage of using word embeddings at the input layer improve the quality of the downstream tasks by providing linguistic information at the input level. Thus, despite some networks using a different objective than LM to learn the parameters, they may still use word embeddings, which is a product of LM and thus, makes LM an even more critical technique for the NLP field.

LM models are classified into two categories, a *Statistical Count-Based* **(SCB)** and a *Neural Language Modelling* **(NLM)** approach. Since SCB approaches are simpler to understand, this section will explore the functioning of the LM objective on behave of the SCB model. The NLM approaches are explored in section 3.3.

The chosen model for the SCB LM illustration is called *n-gram* (Cavnar, Trenkle et al., 1994). The n-gram model introduces different, but very similar, ways of pre-processing a corpus: uni-gram, bi-gram, tri-gram and n-gram, where n is a sequence length. Intuitively, the uni-gram pre-processes a given corpus by identifying all the unique words and counting their occurrence frequency. Whereas the bi-gram looks at all unique word pairs and counts their occurrence frequency. Similarly, the tri-gram looks at all the unique sequences of length 3 and counts their occurrence frequency. In principle, the length of a unique sequence of words can be of any length. However, the longer the sequence, the less likely to appear in the dataset again. Having many unique entries in the look-up table makes it ambiguous, and thus the next word prediction less accurate. After the corpus n-gram pre-processing step, the n-gram model will perform a table look-up in order to retrieve the context co-occurrence frequencies and use it to predict the next word.

The look-up probabilities in the n-gram model are equivalent to a *Markov Chain Probability*, which the LM uses as an underlying composition. The initial Markov Chain Probability computes a joint probability of a sentence by only considering pairs of values at the time. More formally, the Markov Chain Probability rule states:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \longrightarrow P(A \cap B) = P(A \mid B)P(B) \tag{1}$$

We can re-write the equation to represent the next word probabilities:

$$P(w_1, w_2, w_3, ..., w_n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)...P(w_n \mid w_1, ..., w_{n-1})$$
$$P(w_1, w_2, ..., w_n) = \prod_n P(w_n \mid w_1, w_2, ..., w_{n-1}) \tag{2}$$

Now, using Equation 2 to compute the joint probability of the sentence “its the cat that”:

$$P(\text{its}, \text{the}, \text{cat}, \text{that}) = P(\text{its})P(\text{the} \mid \text{its})P(\text{cat} \mid \text{its}, \text{the})P(\text{that} \mid \text{its}, \text{the}, \text{cat}) \tag{3}$$

It is important to note that each of the terms on the right-hand side of the equations is n-gram count probability that gets estimated from the corpus. Thus, in order to predict the next word, the algorithm would need to look-up each part in the conditional probability table. However, note that when the sentences are long, the required n-gram counts are long. This is a problem as it makes the equation unscalable because it is not practical to compute n-grams of every length. For example, consider that we would only pre-process bi-grams and would try to predict $P(\text{cat} \mid \text{a}, \text{happy})$. This would be impossible since we would not have any tri-grams to see how frequently the “a happy cat” sentence has occurred. Hence, to cope with this issue, it is applicable to simplify the equation using the *Markov Assumption*, which states that it is enough to only pick one or a couple of previous words for the history. This assumption means that we can reduce the conditional probability to be fixed and represented by a specific n-gram. Applying the Markov Assumption results in the following formula when using bi-gram:

$$P(w_n \mid w_1, ..., w_{n-1}) \approx P(w_n \mid w_{n-m}, ..., w_{n-1})$$
$$P(\text{that} \mid \text{its}, \text{the}, \text{cat}) \approx P(\text{that} \mid \text{the}, \text{cat}) \tag{4}$$

Abstractly speaking, the n-gram can be compared to a sliding window, which only looks at n most recent words. This also means that the bigger the window size, the more context will be considered to predict the next word. However, as previously mentioned, it should not get too big. Now, consider a 5-gram model and the task of predicting the next two words for the following sentence: "The black cat went out to the garden". In this scenario, the 5-gram model would only consider the last four words (out, to, the, garden) and would look-up all the 5-grams that contain the given four words and would filter out the next word using the following formula:

$$P(w_n \mid w_{n-m}, ..., w_{n-1}) = \frac{count(w_{n-m}, ..., w_{n-1}, w_n)}{count(w_{n-m}, ..., w_{n-1})} \tag{5}$$

The algorithm would choose the next word to come from the 5-gram that has received the highest probability in equation 5. After this step, the sliding window would shift to the words "to, the, garden, pred(t-1)" and the processes would be repeated.

Until now, it has been shown how the count-based probabilistic n-gram model uses the Markov Chain Rule, together with Markov's Assumption to generate next-word predictions. However, this model still faces many serious limitations which come down to the words being represented as context co-occurrence frequencies. This is referred to as the sparsity problem in the literature (Cavnar, Trenkle et al., 1994). For example, consider the following two major problematic cases. When modelling a joint distribution of a sequence/history using a pure n-gram model, there are cases where a word sequence has not been encountered in the corpus; hence the model would not know what to output. Another problem is that word representations are not linguistically informed. Meaning the probabilities themselves do not carry any information about themselves. For example, the words "cat" and a "dog" should have a similar representation as both belong to the subset of animals. The most successful solution to the sparsity problem is to use a neural network with LM objective to model the context co-occurrence frequencies into a single vector of floating numbers.

The more advanced NLM approaches utilise a neural network for LM, with the most common architectural choices being: Feed-forward Neural Networks **(FNN)** and *Recurrent Neural Networks* **(RNN)**, which both naturally solve the data sparsity problem. The examples of how the aforementioned models use the LM objective is explored in sections 3.3.1 (FFN) and 3.3.2 (RNN). Furthermore, the most recent work utilises an idea of *attention-based* architecture with a modified LM objective to produce *contextualised word embedding.* The new embeddings are produced as a function of the whole sentence, meaning that the same word in two different contexts will get a different and more suitable internal representation. The new techniques have proven to be very successful for many deep learning applications and stand ground to be a replacement to the prior methods. The contextual word embedding algorithms are explored in section 3.3, and their internal representations in section 4.

3.2 Neural Network Architectures

3.2.1 RNN: Recurrent Neural Networks

Whenever humans listen to one another speech, they persist the meaning of the previously spoken out words in order to correctly understand the whole sentence. It is not the case that we understand words separately, and whenever we hear a new word, we start thinking from scratch. It is instead the case that we have a memory state, which allows us to persist contextual meaning of a sentence. From the neural network's point of view, consider a task of processing a video clip one frame at a time to capture some events spread across multiple frames. It is unclear how Feedforward Networks could persist the information from different frames in order to collectively recognise an event.

Recurrent Neural Networks addresses this issue of an information persistence by introducing a recurrent loop (see Figure 1).

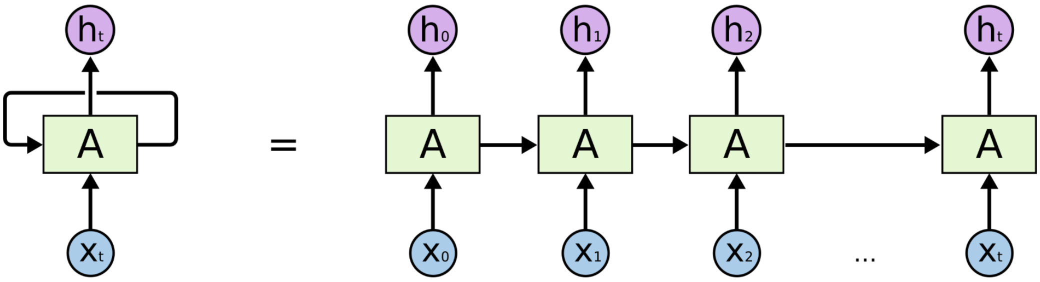


Figure 1: On the left, a schematic representation of a RNN. On the right, an unrolled representation of the left diagram. Each unrolled module represents a state h at some point in time t. From Colah (2015).

What FNNs and RNNs have in common is that both take an input x_t, process it through network A, and produce an output h_t. However, an important distinction is that FNN' output is never fed back into the network, whereas, in RNNs, the future input gets derived from the past outputs. One way of interpreting RNNs is to think of them as multiple instances of the same network, where each module passes its hidden state to the next one in the sequence. This structural composition naturally encodes the order of an input which is very appealing for various applications requiring sequential processing, i.e., text (Sutskever, Vinyals and Le, 2014), video (Ullah et al., 2018), and sound (Parascandolo, Huttunen and Virtanen, 2016).

In theory, RNNs can connect previous information to the present task. In the case of action recognition in video, previous video frames might inform the understanding of the present frame. However, in practice, it is not always the case as RNNs are not so good at keeping long-term dependencies (Vaswani et al., 2017). For example, consider a task of question answering based upon a paragraph. After processing all the words through the RNN, questions based on the most recent sentences would most likely be correct; however, questions on the first sentences would most likely be answered wrong. Such behaviour is natural since the hidden states get overwritten with the newer information - there is no memory control mechanism. This problem was addressed by Hochreiter et al. (2001) and Hochreiter and Schmidhuber (1997), who introduced an *LSTM* network that improves the long-term dependency aspect.

3.2.2 LSTM: Long Short-Term Memory Networks

The RNNs' repetitive internal structure considers the previous knowledge when computing a new output. However, it does not have any mechanism to state the "importance" of the previous knowledge, and thus, a piece of crucial old information often gets overwritten with a new but irrelevant one. The LSTM network is an extension to RNN which introduces a memory controlling mechanism whose objective is to persist the more "important" information (See Figure 2). As a result, the LSTM networks significantly improve on the information long-term dependency aspect and perform well at a broad range of applications. To give an intuitive example, consider predicting next word in the sequence given the following context: "Adrian speaks fluent Polish... Adrian is visiting England. Adrian speaks fluent [...]". Despite the most recent word indicating the nationality being "England", the network would be able to figure out that the correct output is Polish. In contrast, the RNN would most likely predict English, since the starting context would be most likely overridden.

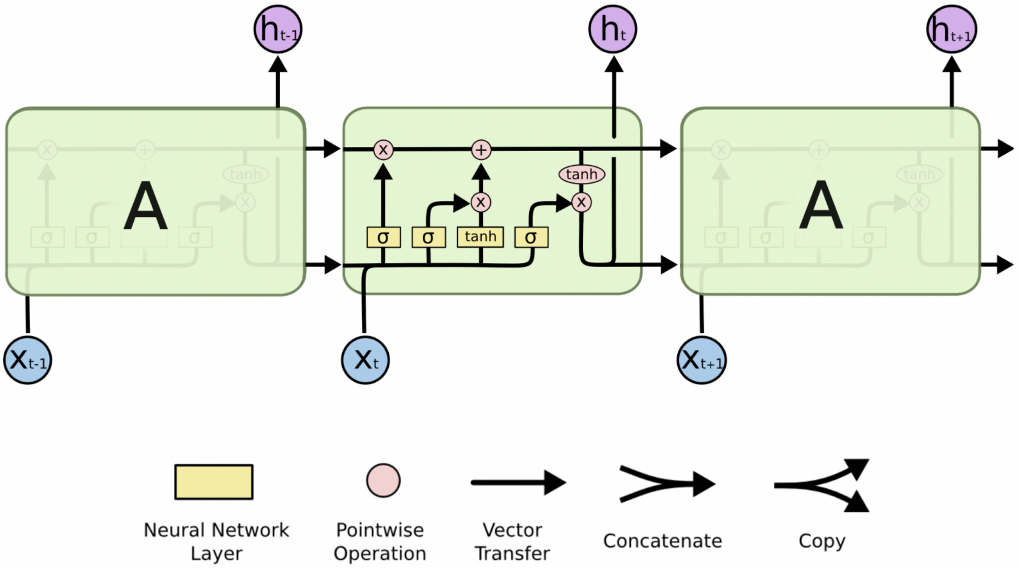


Figure 2: A schematic representation of LSTM's internal structure. From Colah (2015).

The critical concept differentiating the LSTM networks from the RNNs is an internal memory controlling mechanism. The mechanism is made up of a *Cell State* and three *Gating Units*. The cell state can be thought of a relevant "information repository", or simply "memory" that stores the most important information captured throughout the entire input sequence (See Figure 3). Whereas the unit gates are shallow neural networks that collectively control the information flow to the cell state. Each gate plays its part in the whole mechanism. However, collectively at each input step, they create a capability to add and remove "knowledge" from the cell state.

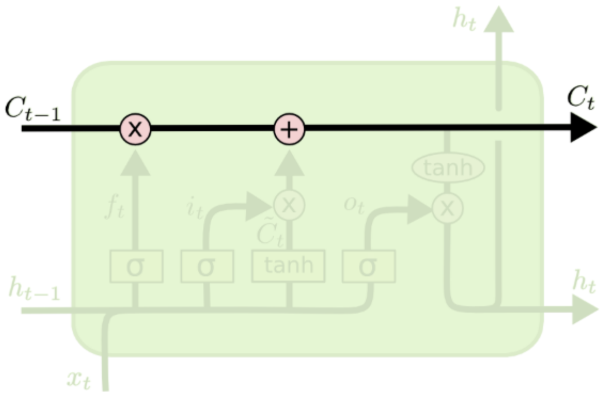


Figure 3: A schematic representation of LSTM cell state. From Colah (2015).

Now, given the core idea, the interactive process of each unit gate will be covered to greater detail by dividing the explanation into four logical steps: forget gate, input gate, cell modification and output gate. Note that, from the technical point of view, the gates are single neural network layers with a non-linear activation function whose outputs get point-wise multiplied in order to pass information to the cell state.

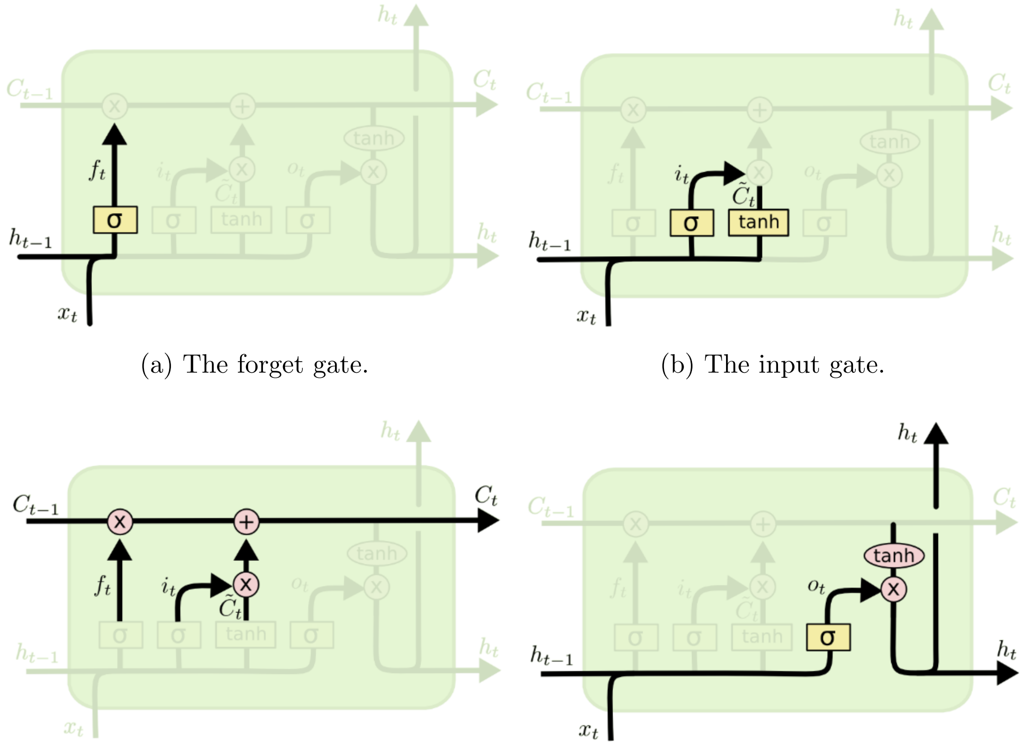
The idea behind the forget gate is to decide what information should be thrown away or kept from the cell state. In more detail, it is done by an element-wise addition of the input with the previous hidden state $(x_t + h_{t-1})$ and then by running it through a sigmoidal activation function. The sigmoid activation function collapses the weighted output of a neural network into a vector of values ranging between 0 and 1. Since anything multiplied by 0 is 0, the values in the resulting array closer to 0 mean to forget and the values closer to 1 mean to keep. (See Figure 4a). To illustrate the forget gate mechanism, consider a language modelling objective, where the task is to predict next word in the sequence given the following sentence: “Adrian is happy because he got a cat for Christmas, however, on the next day the cat run away and Adrian became sad. Adrian feels ...”. In such a scenario, the forget gate should remove the information about Adrian being happy because accordingly to the new information he feels sad. More formally, the process looks as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{6}$$

The intuition behind the input gate is to update the cell state with a new "selected" information. In more detail, the input mechanism is made out of two gating units with separate weight sets. Similarly to the forget gate, the first unit is a sigmoid neural function that outputs a vector with values ranging between 0 and 1, based on a concatenation of the input with previous hidden state. On the other hand, the second input gating unit uses a tanh neural function, which squashed the neural output to lay between -1 and 1. Then, given the tanh output, we multiply it by the sigmoid output. The initiative behind the two units working in collaboration to decide on the "relevant knowledge" is as follows: the tanh values ranging between -1 and 1 decide on new candidate values that we potentially want to place in the cell state. Whereas, the sigmoidal values ranging between 0 and 1, decide on the "degree" of the relevance of the values selected by the tanh output. (See Figure 4b).

$$\begin{gathered} i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \tilde{c} = tanh(W_{\tilde{c}}[h_{t-1}, x_t] + b_{\tilde{c}}) \end{gathered} \tag{7}$$

The previous two steps have shown how LSTM decides what information to forget and how it selects the new information to be stored. Now, using this information, the next major step is to modify the previous cell state C_{t-1} into a new cell state C_t, which gets done in two steps. The first step is to multiply the previous cell state by the forget gate vector $f_t \times c_{t-1}$ to forget the information decided earlier. Then, the second step is to add the input gate vector to the cell state after the forgetting stage, $f_t \times c_{t-1} + i_t \times c_t$ to update the cell state with the new candidate vector (Illustrated in Figure 4c).



(a) The forget gate.

(b) The input gate.

(c) The cell modification.

(d) Output gate.

Figure 4: The information persistence controlling mechanism divided into four major steps. From Colah (2015).

Lastly, we have the output gate. Its role is to decide what the next hidden state should be, which is the actual vector used for next prediction. The hidden state also contains information about the previous outputs since it is made out of the cell state. In fact, it is a filtered version of the cell state. The hidden state production is divided into three parts. The first part is to pass the previous hidden state and the current input into a sigmoid activation function. Secondly, we run the new cell state through a tanh activation. The final step multiplies the output of step two against the output of step one, which derives a new hidden state. See Figure 4d for an illustration and Equation 8 for mathematical support of the process.

$$\begin{aligned} c_t &= f_t \times c_{t-1} + i_t \times \tilde{c}_t \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times tanh(c_t) \end{aligned} \tag{8}$$

To summarise, an LSTM network comes from the RNNs family and extends the basic recurrent loop with an internal memory mechanism. The memory mechanism selects the more relevant information from the whole input sequence and acts as a "knowledge repository" for the inputs. This way, at every time step, the current input knows about the previous in the sequence. The memory unit is called a cell state which gets linear interactions from the unit gates, namely: forget, input, output gate, which collectively control the information flow in and out of the cell state. The internal mechanism provides an information persistence aspect which is beneficial for tasks requiring long-term information dependency, e.g., question answering, text summary, or in general, language generation.

3.2.3 Self-Attention Mechanism

The attention mechanism is motivated on the Human's ability to focus on particular areas of an image, or similarly, the ability to correlate words in a sentence. For example, by looking at an image of a cat in a woody background, we would pay high-resolution attention to the cat while perceiving the woods in a lower settlement. Hence, when seeing something in high resolution, our brains automatically perceive that particular part to be outstanding. Similarly, we can explain the relationship between words in a sentence, i.e., we can point out words which give the most information to the meaning of the whole sentence. From the machine learning perspective, the attention mechanism output can also be interpreted as an importance vector.

The self-attention is a stand-alone mechanism which can be utilised at a variety of architectures and problems. In most use cases, it is applied to the hidden states of RNNs for tasks like language translation (Sutskever, Vinyals and Le, 2014). However, recent work has proven that the attention mechanism can achieve superior results in language translation when applied without any underlying architecture (Vaswani et al., 2017). Note that, there is a variety of attention methods proposed in the literature; however, this section will refer to the self-attention from the publication by Vaswani et al. (2017).

Abstractly speaking, the self-attention mechanism figures out a relationship between elements in a sequence by producing a unique representation of that sequence. The idea of building a relationship between different elements makes the attention mechanism applicable to many problems. For example, it has been successfully applied to the problem involving reading comprehension (Cheng, Dong and Lapata, 2016), abstractive summarisation (Paulus, Xiong and Socher, 2018), textual entailment (Parikh et al., 2016) and more importantly representation learning (Devlin et al., 2019).

To give an intuitive example, consider using an attention mechanism to create *contextual word embeddings*, i.e., embeddings that are a function of the entire sentence. When computing a vector representation for word w_i at a particular position, the presence of all the other words gets reflected in the embedding. However, some of the words in context should be more related to w_i than others; hence, the attention mechanism decides which words are more relevant to the specific word's representation. For example, consider the following sentence: "The car could not run because it was broken". What does the word "it" refers to? For humans, it is a simple answer; however, for algorithms, this is a difficult question. In this scenario, the attention could associate the word "it" with the word "car" in order to reflect the word car in the "it" vector representation.

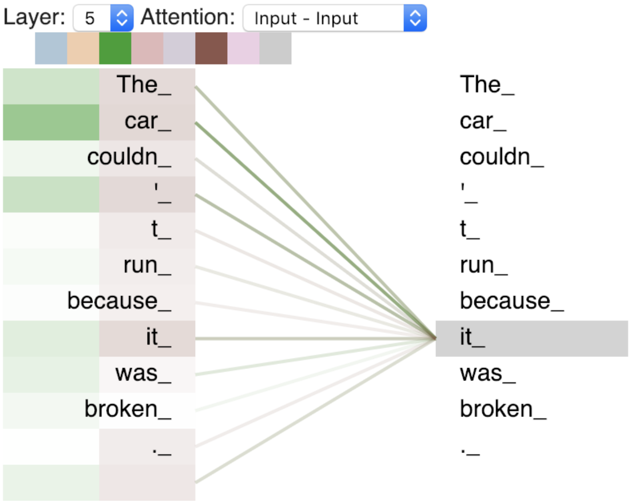


Figure 5: A visualisation of attention weights, which show the relationship of the word "it" to the context of the sentence. From an interactive demo[3]attached to the publication by Vaswani et al. (2018).

Note, from now on, the self-attention mechanism will be explained from the representation learning point of view only. Furthermore, in order to simplify the interpretation, the process will be mostly explained at the level of vector multiplication. Only after, it will be explained in its pure matrix form.

The first step in the process is to create a separate set of three vectors for each input token in the sequence. The three vectors representative of a particular input position are constructed by multiplying the respective input token by three matrices that we optimise during the training process (See Figure 6). The three resulting vectors are called: *Query*, *Key* and *Value* and should reflect the respective input that they are made from. The size of the Query, Key and Value vectors should be relatively small in contrast to the final embeddings since the whole process creates 3 of them for each input token in the sequence. As it is later explained, those vectors are used to create the final contextual word embeddings. Vaswani et al. (2017) have

[3] `https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb`

used the dimension of 64, where their encoded output embedding was of dimension 512. There is no restriction to the dimension of the three vectors; however, it is an architectural trade-off between the performance and the computational time.

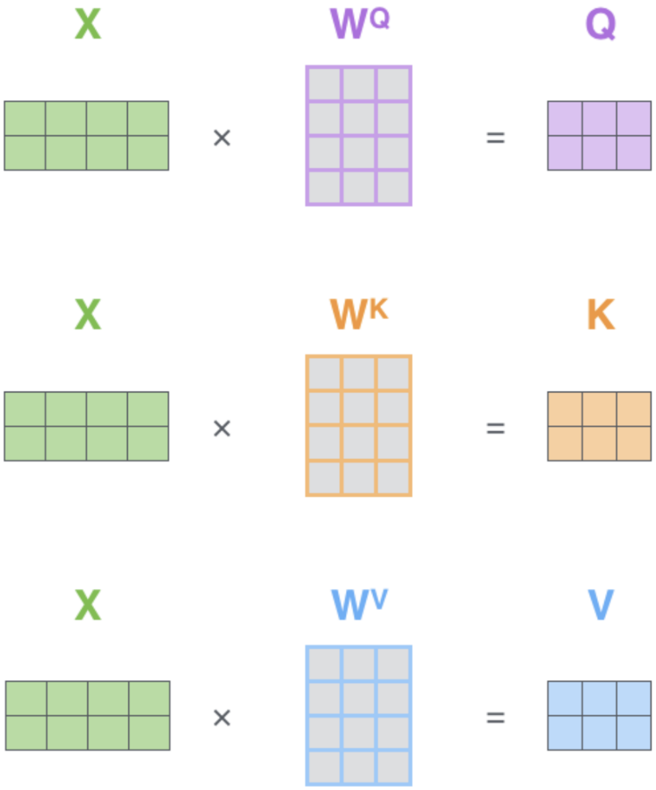


Figure 6: A representation of the Query, Key and Value creation process. The input embeddings x of two words is multiplied by three weight sets $\{W^Q, W^K, W^V\}$, which results in the creation of a query, key and value projection of each word in the input sequence. From Alammar (2018).

Once the vector projections Q, K, V have been obtained, the next step is to calculate a score. Each element of the input sequence needs to be scored against all the other inputs in the sequence. The score will determine how much focus a particular word should get from every single element in the sequence. The scoring is done by taking a dot product of a particular word's query vector with the keys of every other input. For example, if the input sequence is of length three, to compute the scores for the first word, $q_1 \bullet k_1 = s_1, q_1 \bullet k_2 = s_2, q_1 \bullet k_3 = s_3$.

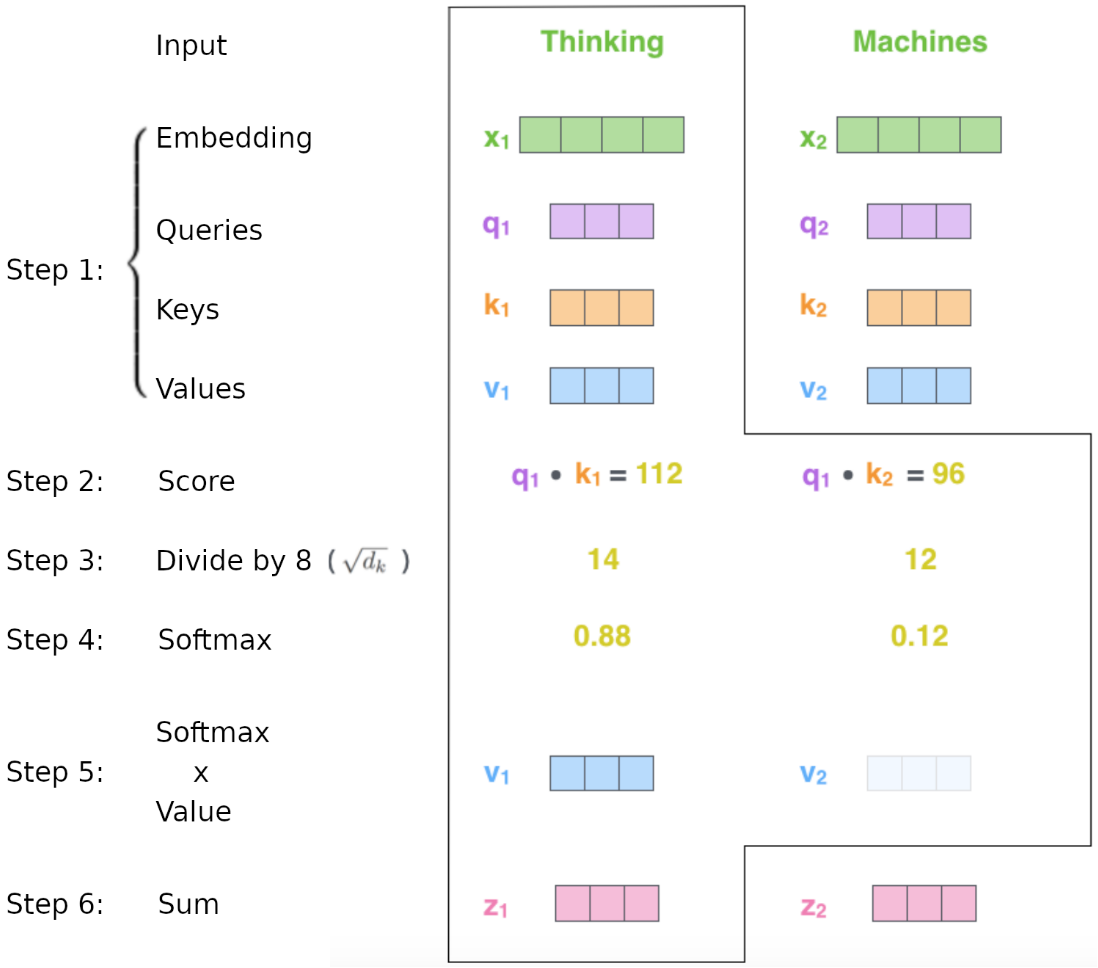


Figure 7: A step by step illustration of the contextual representation building process for a word. From Alammar (2018).

In step three, each score of a particular embedding needs to be divided by the square root of the key vector dimension $\sqrt{64} = 8$, i.e., $\frac{s_1}{\sqrt{64}}, ..., \frac{s_n}{\sqrt{64}}$. The denominator value is not restricted to the use of the square root, however, according to Vaswani et al. (2017) it helps to keep the gradient stable. Moreover, step four is to use a softmax function over the set of values retrieved from the divisions (step three), i.e., $softmax(\frac{s_1}{\sqrt{64}}, ..., \frac{s_n}{\sqrt{64}})$. The softmax function will strengthen the highest result and shrink down the less necessary results in order to reflect the more important context.

The fifth step is to multiply the softmax results by the value vectors. The intuition is that the value vectors that get multiplied by higher scores will be reflected in the embedding more. Note that it is almost always the case that the product of itself scores the highest, as a word needs to prioritise itself in its representation and then consider the other relevant context. Furthermore, in step six, the resulting vectors from step five get summed up into one representation, which is the product of the attention mechanism for the word at the first element in the sequence. The entire process is depicted in Figure 7.

Until now, it has been shown how the attention mechanism calculates a context for one word. Knowing how it works, we can go a step further and explore how it is done for the whole sentence at the matrix multiplication level. Performing the calculation using matrices computes the whole sentence at once, thus, improves the efficiency of calculations.

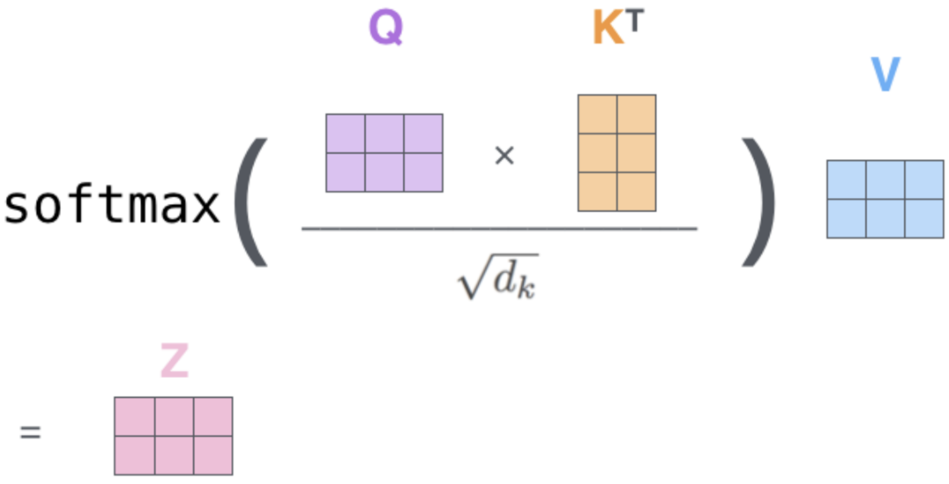


Figure 8: A self-attention formula that collapses steps 2-6, from Figure 7, into a matrix form computation. The matrix formula produces the self-attention output for all the inputs in the sequence, at once. From Alammar (2018).

Figure 6 illustrates the calculation process of the query, key and value projections (step one). Steps 2-6 can be collapsed into one formula which produces the attention output for the whole sequence (See Figure 8).

Vaswani et al. (2017) go a step further and propose a multi-head attention layer. At a high level, the multi-head attention makes use of the self-attention mechanism by computing multiple representations of the same word in the same context and then concatenating the representations to form one. The multi-head attention aims to expand the context of an embedding further.

Recall the query, key and value projection processes. In this step, the set of weights $W_1^{Q,K,V} = \{W^Q, W^K, W^V\}$ decides which parts of the embedding should be projected. Having n-head attention means having $W_n^{Q,K,V}$ sets of weights $\{W^Q, W^K, W^V\}$, where each set is randomly initialised, thus, trained differently. Hence, different heads capture different relevant context for a specific word. Figure 5 illustrated two attention heads and their captured context for the word "it", note that each head captures a different context. Having n heads also results in having Z_n attention outputs for a word, which, as mentioned before, needs to be transformed into a single representation. The transformation process is done by concatenating all the outputs $Z_0, Z_1, ..., Z_n$ and multiplying it by another set of weights W^O, which results in a shrank output of the size of W^O. The multi-head attention is illustrated in Figure 9.

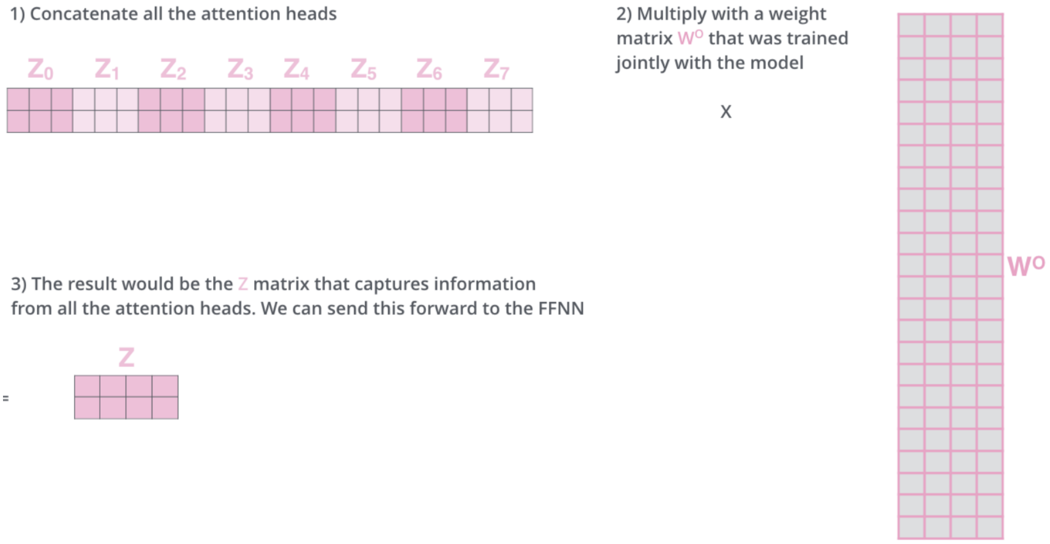

1) Concatenate all the attention heads
Z0
Z1
Z2
Z3
Z4
Z5
Z6
Z7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
Z
=

Figure 9: A schematic representation of the concatenation process involving multi-head attention outputs. From Alammar (2018).

To summarise, the attention is a stand-alone mechanism which can be applied to many architectures for many different tasks. One very successful application is for contextual word representation learning. The attention mechanism perfectly suits such an application since it figures out the relationship between different words in the sequence. In more detail, it projects three different vectors for each token in the sequence (Query, Key and Value) and uses them to calculate the “relevance score” of each element in the sequence. Furthermore, the mechanism creates multiple copies of the process described above, called multi-head attention, where each uses a different set of trainable weights. Such a strategy outputs multiple representations for the same sentence, which are concatenated and then shrunk to one fixed representation by a trained encoder. The resulting matrix is a function of the entire sentence, meaning that, its representation is unique and entirely depended on the tokens in the sentence - changing one input token would change the representation of every other token.

3.3 Context-encoding Algorithms

In this section, we are going to go through the state-of-the-art word embedding algorithms that largely contributed to the NLP field development. In order to understand the differences in word embeddings, it is necessary to know the building mechanism, which defines how the context is captured. Firstly, this section discusses a word2vec embedding modelling approach that uses a shallow feedforward neural network with two different language modelling objectives. Secondly, it is shown how ELMo uses an LSTM network with left-to-right and right-to-left language modelling objectives to combine the two and form a rich bidirectional language model representation that captures context from both sides. Lastly, it is demonstrated how BERT builds on top of recent ideas to construct deep bidirectional language model representations. Due to BERT's superior results over previous methods, BERT has a promising foundation to be a replacement for word2vec in many industrial applications bert.

3.3.1 Word2Vec

Mikolov et al. (2013b) have proposed two effective language modelling methods for learning distributed vector representations (word embeddings), a Skip-gram (SG) and a Continuous-Bag-of-Words (CBOW). Both of these methods are used to map words into vector representations, thus the name word-to-vector (word2vec). The word embeddings obtained from using models trained on large amounts of unlabelled data, capture a syntactic and semantic word relationship. For example, the relationship between "worst" and "best" should be captured by these distributional models as the words have a related meaning through the opposition and often can be found in similar contexts. Refer back to section 3.1.1 for more information on word embeddings and the information they capture.

At the time of the Mikolov et al. (2013b) publication, computational power was a problem; thus, it was harder to train models long enough so that they generalise well. The two major contributions the authors proposed were: a lighter architecture by not using multiple hidden layers, and the consideration of additional context when computing word embeddings. As a result, the models could compute better quality word embeddings from big data in a shorter time, which largely contributed to the models being widely used in the industry today. Furthermore, the authors showed a possibility to manipulate vector space using arithmetic operations and linear distance functions; having two representations it was possible to derive a new and related one. For example, the result of a vector calculation, $vec(\backslash Madrid") - vec(\backslash Spain") + vec(\backslash Poland")$ is closer to $vec(\backslash Warsaw")$ than to any other word vector in the vector space.

The improved word embedding accuracy on downstream tasks is mostly due to specific training strategies, which will be explored now, starting from the CBOW model. Note, that the strategies are used to train the model's weights, which after the training processes are used to produce the word embeddings.

Recall Equation 4 and 5 from section 3.1.3. The LM objective defined in equation 4 can only look back in time as it only uses the previous sequence to predict the next element. LM for word vector modelling does not suffer from this restriction and can also consider the future context. That is an advantage since the more context an objective function is exposed to, the more "knowledge" the produced final embeddings have. Thus, the CBOW model uses both, the n words before and after the pivot word w_n in order to predict the pivot word, (See Equation 12). The model is named a Continuous-Bag-of-Words because it uses continuous context whose order does not get reflected in the embedding; hence, the word "bag". The model's architecture is illustrated in Figure 10.

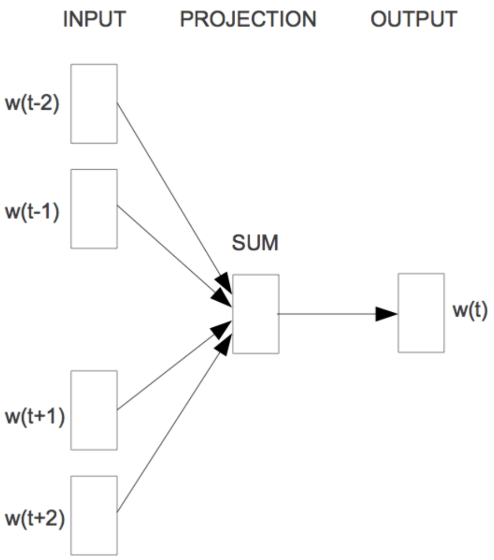


Figure 10: The architecture of a Continuous Bag of Words model proposed by Mikolov et al. (2013b). The input layer represents one-hot vectors of respective words. The word representations are then multiplied by a set of weight connections and their weighted sum is computed. The connection from the projection to the output represents a softmax layer, which produces a probability distribution over all the words in the vocabulary.

$$J_\theta = \frac{1}{N}\sum_{n=1}^{N} \log p(w_n \mid w_{n-m}, ..., w_{n-1}, w_{n+1}, ..., w_{n+m}) \tag{9}$$

The objective of this model is to maximise the average log probability of a target word, given its respective contextual window of size n, for all the words in the vocabulary.

Figure 10 best illustrates the word2vec CBOW modelling process. The first layer from the left illustrates the input words in the form of one-hot vectors. Vaguely speaking, one-hot embedding is the easiest method of numerically representing and disambiguating words. In more detail, to construct the one-hot vectors, it is necessary to first, identify all the unique words (vocabulary) in the training corpus[4]. Then, the next step is to create a matrix

[4] All the words. For example, strong and strongly would get two different representa-

of zeros, whose length is the number of unique words. Furthermore, for all of the rows in the matrix, we substitute a unique position with a value of 1. The intuition is that we end up with a matrix of the same size as the vocabulary, where each row contains only one 1, that uniquely identifies a word. Lastly, we create a dictionary between the unique words and the unique vectors, thus, we end up with a unique representation for each unique word in the corpus. More formally, given a corpus C, we identify all its unique words, or its vocabulary, $V \in C$. Then we calculate the length $V_l = len(V)$ and construct a matrix M of size V_l x V_l. Then, for all of the unique words, we map one word to one vector $dict(V_n, M_n)$, where n ranges from 0 to V_l. Note that each word's length is proportional to the vocabulary size, thus, the greater the vocabulary, the more dimensional embeddings.

Furthermore, the projection layer represents the weighted sum of the contextual input, which is denoted $h^s = \sum_{n=1}^{N} w_n h$. Nevertheless, it is also used to create the final word embeddings; after training, we multiply all the one-hot embeddings by the projection layer, and the resulting outputs define the embedding, i.e., $h^s \times V =$ a set of word2vec embeddings. Note that the size of the hidden layer defines the size of the final embeddings, which is usually between 100-1000.

Lastly, the output layer takes the sum of the weighted input h^s of the previous layer and multiplies it by another set of weights v_{w_n}, where, v_{w_n} represent weight connections between the hidden state and the classes representing words in the V. Then, a softmax function, equation 10, is used to produce a probability distribution over the V.

$$p(w_n \mid w_{n-m}, ..., w_{n-1}, w_{n+1}, ..., w_{n+m}) = \frac{exp(h^s v_{w_n})}{\sum_{w_i \in V} exp(h^s v_{w_i})} \tag{10}$$

The inner product $h^s v_{w_n}$ computes the un-normalised log-probability of the word w_n and normalises it by the sum of the log-probability of all words in V. In other words, the softmax uses an exponential function to set all values to be positive and then maximises a probability distribution for each

tions; not considering subword information is a drawback of this method.

word in the vocabulary by dividing it by the sum of all other words. Thus, after softmax, all the values are probabilities which are positive and add up to 1.

In contrast to the CBOW, the SG architecture is the exact opposite. Instead of predicting a target word from the context word, the SG predicts the context words from the centre word. The bigger the window size, the better the quality of the resulting embeddings, however, the computational time increases. Since the more distant surrounding words are usually less related to the target's context, the authors decided to assign smaller weights to the distant words. The architecture is depicted in Figure 11.

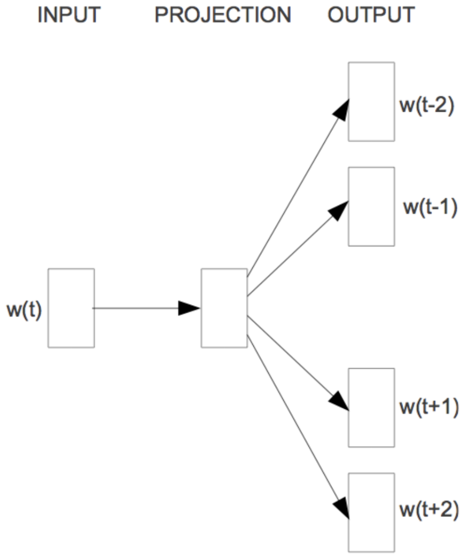


Figure 11: The architecture of a Skip Gram model proposed by (Mikolov et al., 2013b).

Thus, the SG's objective is to average the log probabilities of the surrounding n words to the left and to the right of a target word w_n:

$$J_\theta = \frac{1}{N}\sum_{n=1}^{N} \sum_{-n\leq j\leq,\neq 0} \log P(w_{n+j} \mid w_n) \tag{11}$$

And the softmax is:

$$P(w_{n+j} \mid w_n) = \frac{exp(h^s v_{w_n+j})}{\sum_{w_i \in V} exp(h^s v_{w_i})} \tag{12}$$

To summarise, word2vec are two shallow neural LM approaches that utilise large amounts of unlabelled data in order to optimise its weights; which are later used to create word embeddings that other models benefit from at the input layer. During the training process, the CBOW 's objective maximises the probability of the target word given its context, or differently put, predicts a target word from the sum of all word vectors in its contextual window size. Therefore, after the training process, the model will predict the most probable word for a given context. This approach inherits a predictive limitation which largely depended on the training dataset. To better illustrate the problem, consider the following example. Given the context "yesterday was a really [...] day", the model will most likely predict commonly seen words like "beautiful" or "nice". However, words like "delightful", despite fitting even better into the context, will get less probability since the network's objective models to predict the most probable words. In other words, the "delightful" word probability will get smoothed out because of examples with more frequent words.

In contrast, the SG model rotates the objective and predicts the window of contextual words given a middle target word. In such a setting, the SG model is constrained to understand the word "delightful" as, during the training process, it must predict its contextual window words. Differently put, with SG the word "delightful" will not try to compete with the word "beautiful" in the probability space, but instead, will treat the "delightful" + context pairs as new observations. Therefore, the SG model is better at representing rare words, and thus, is more diverse. However, the CBOW is better at representing more common words, which is beneficial for cases where the word diversity is not a priority.

Furthermore, the SG works better with small amounts of training data because we are using one target word to predict many contextual words, and thus, we create multiple errors given one input; a training objective that creates more error results in faster weights optimisation. On the other hand, the CBOW trains several times faster.

It is essential to understand that there is no better modelling strategy between the two; the choice should entirely depend on the specifics of the problems word2vec-distributed. In order to achieve the best accuracy at a particular task, it is vital to first, fully understand the task, and then to choose an architecture whose trade-offs are more beneficial for the specific task and the available resources.

3.3.2 ELMo: Embedding from Language Model

The embedding from Word2vec LM is limited to a fixed number of contextual words that the model uses to construct the representation, which is a limitation as the true context can vary in length. ELMo, proposed by Peters et al. (2018), solves this problem by benefiting from the sequential characteristics of an LSTM network and using it to encode full sequence while naturally persisting the order of the context. Furthermore, using word2vec to learn word vectors only allows to model one context-dependent representation for each word. This is a problem because in natural language we often have many possible meanings for a word or phrase (polysemy). ELMo, like word2vec, models complex characteristics of word use (grammar and semantics), however, it also models how these characteristics vary across contexts. Hence, ELMo solves the polysemy problem by encoding each word representation in the sequence as a function of the whole sentence, similarly to what self-attention does (see section 3.2.3). This means that ELMo produces different representations of the same word used in a different context. Furthermore, in contrast to word2vec, using one-hot vectors to represent words at the input layer, ELMo uses a convolutional neural network (CNN) over characters (Kim et al., 2016) to construct raw word representations x_k^{LM}. The CNN over characters method assembles words purely from character level representations. This brings an advantage over one-hot vectors because the pure word representation already captures some internal word structure. For example, the words “limit” and “limitless” would be somewhat related because they both contain the same subword. Furthermore, the character-based method is also robust to unknown words that weren’t encountered during the training process as they’re being represented with subword phrases.

ELMo produces word vectors by running the input through learnt functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. Adding the ELMo word vectors to existing models, at the time of publication, has surpassed the state-of-the-art accuracy on six different NLP tasks that include question answering, textual entailment and sentiment analysis (Peters et al., 2018). Now, it will be made clear what the deep bidirectional language model is and how it is used to encode word embedding.

In a high-level overview, the term deep bidirectional language model refers to an architectural structure which makes use of at least two layers. Each layer is constructed with two LSTM networks. One is trained to predict the next element in the sequence given the previous words. Whereas the other uses the LM in a reverse way, i.e., to predict the previous words given the sequence of future words. Thus, the term bidirectional refers to those two LSTM networks using the LM objective function in order to collectively capture context from both directions.

Given a sequence of N words, $(w_1, w_2, ..., w_N)$, the forward language model computes a probability of the sequence by calculating the probability of a word w_k given the history $(w_1, ..., w_{k-1})$:

$$P(w_1, w_2, ..., w_N) = \prod_{k=1}^{N} P(w_k \mid w_1, w_2, ..., w_{k-1}) \tag{13}$$

Now, in order to produce a forward LM output, the sentence representations in the aforementioned form, x_k^{LM}, are then passed through L layers of forward LSTM networks. That is, at each position k, each LSTM layer outputs a context-dependent representation $\overrightarrow{h}_{k,j}^{LM}$ where $j = \{1, ..., L\}$. The top LSTM output, $\overrightarrow{h}_{k,L}^{LM}$, is then run through a softmax layer in order to predict the next word w_{k+1}.

As previously mentioned, ELMo is bidirectional, which means that we also need to predict previous words from the future context. This works similarly to equation 13, however, in a reverse order:

$$P(w_1, w_2, ..., w_N) = \prod_{k=1}^{N} P(w_k \mid w_{k+1}, w_{k+2}, ..., w_N) \tag{14}$$

The backward pass can be implemented in an analogous way to the forward pass and should also produce a representation $\overleftarrow{h}_{k,j}^{LM}$ for each element in the sequence k for L many LSTM layers. Then, following the same procedure as with the forward pass, $\overleftarrow{h}_{k,j}^{LM}$ is processed by the softmax layer in order to predict a word w_k given $(w_{k+1}, ..., w_N)$.

A bidirectional LM that ELMo uses combines the forward and backward LM passes and jointly maximises the log likelihood for the prediction:

$$\begin{aligned} &\textstyle\sum_{k=1}^{N}(log\ P(w_k \mid w_1, ..., w_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) \\ &+ log\ P(w_k \mid w_{k+1}, ..., w_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)) \end{aligned} \tag{15}$$

Both LSTM networks share parameters for input representation construction Θ_x and the softmax layer prediction Θ_s. However, they use separate parameters for LSTM networks going in each direction.

Furthermore, ELMo has introduced a new approach of learning word representations, i.e., a linear combination of the biLM layers. For each word w_k, a L-layers biLM computes a set of $2L + 1$ representations:

$$\begin{aligned} R_k &= \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} \mid j = 1, ..., L\} \\ &= \{h_{k,j}^{LM} \mid j = 0, ..., L\} \end{aligned} \tag{16}$$

Where $h_{k,0}^{LM}$ is the input layer and $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}]$, for each biLSTM layer. Figure 12 illustrate the whole process until now. Furthermore, the authors ran interesting experiments on each of these representations and found that biLM layers encode different types of syntactic and semantic information. Generally, they found that using the first layer's output performs well on part-of-speech tagging tasks indicating that the first layer captures syntax/grammar well. It's also been shown that the last layer's output performs well on supervised word sense disambiguation tasks, indicating that the last layer encodes semantic information. Combining the internal states in this manner allows for very rich word representations.

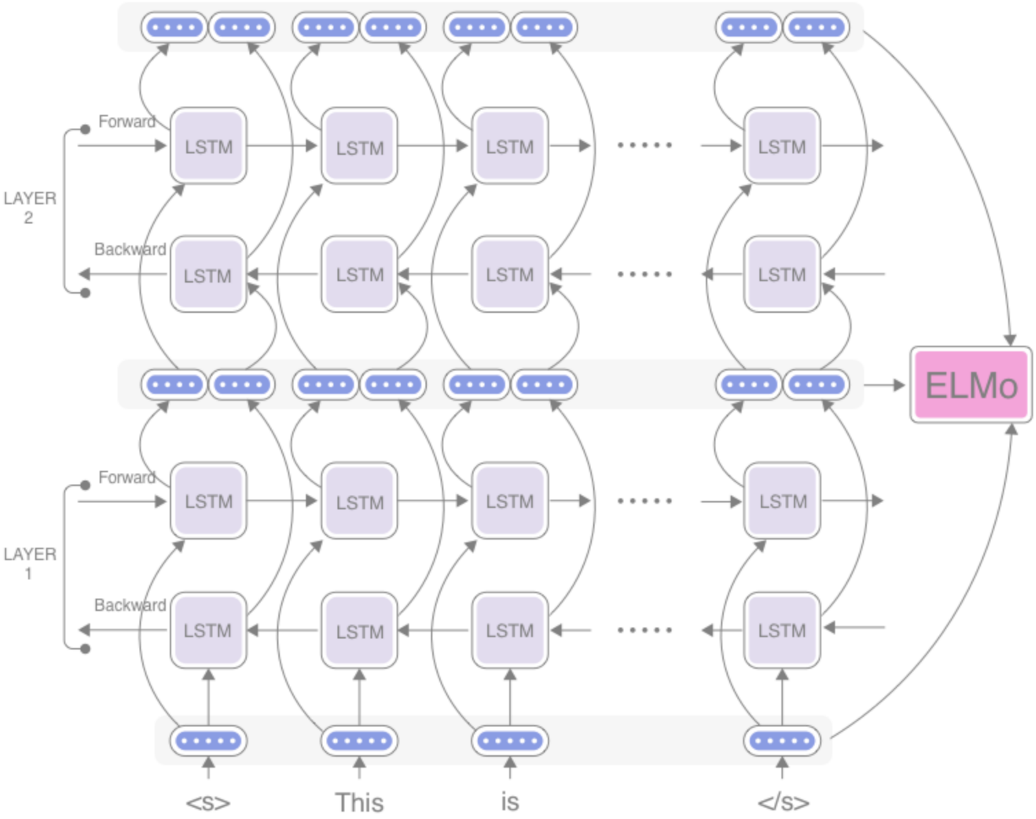


Figure 12: An abstract representation of ELMo's architecture and data flow. From (Hagiwara, 2018).

Now, in order to use the outputs in R_k for downstream tasks, in its simplest case, it is possible to use a particular layer's output as a context-dependent word vector representation, e.g., the top layer's output, $E(R_k) = h_{k,L}^{LM}$. On the other hand, a better way would be to collapse all the outputs into a single vector representation $ELMo_k = E(R_k; \Theta_e)$. This can be done by running all the outputs separately through a task-specific set of weights and then back-propagating the weights on the weighted average error:

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM} \tag{17}$$

Where s^{task} are softmax weights and the scalar parameter γ^{task} helps with scaling the entire ELMo vector.

To summarise, ELMo uses bidirectional LM which is a concatenation of two LMs going in opposite directions. As depicted in Figure 3.3.2, there is one LSTM scanning words from right to left and another scanning from left to right. This means that both of the LSTM networks produce context-depended outputs for each element in the sequence. The outputs then can be concatenated in order to form an intermediate representation of a word. However, ELMo goes a step further and comprises a multi-layer bidirectional language model. This is done by stacking two forward and two backwards passes together where the intermediate representation is fed as an input to the upper layer. This way the intermediate representations get processed further and can represent more abstract semantic concepts. Overall, ELMo produces $2 \times L + 1$ representations, where L stands for number of layers, and uses them to create a task-specific weighted combination of these intermediate word representations.

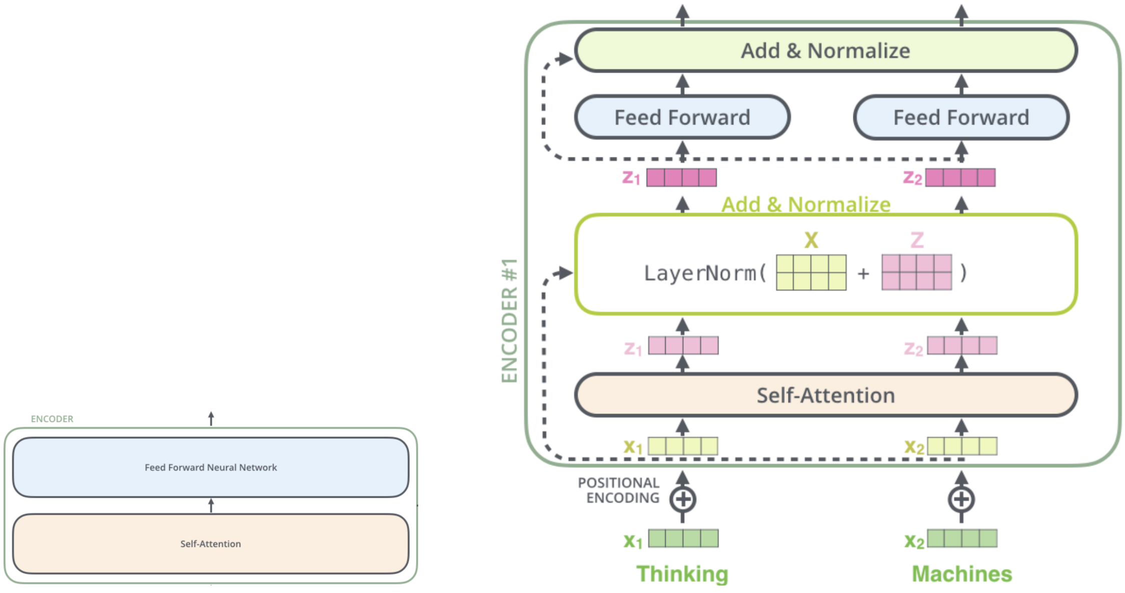
3.3.3 BERT: Bidirectional Encoder Representations from Transformers

BERT (Bidirectional Encoder Representations from Transformer) is an architecture proposed by Devlin et al. (2019). Despite the very recent publication, BERT has received a lot of attention from the community due to its outstanding performance on eleven NLP tasks, which include: sentence classification, token-level classification, inference, and question answering. BERT's key technical innovation is to pre-train a transformer-encoder (TE), an architecture fundamentally based on the self-attention mechanism, using deep bidirectional LM. This means that BERT is trained to produce deep bidirectional representations by jointly conditioning on both, left and right context in all layers. Conditioning on both context sides was previously impossible using one network, however, a new Masked LM (MLM) that BERT has proposed makes it possible.

Now, we will see how BERT works in detail by explaining its architecture structure and its input representation. Further on, it will be explained what the new Masked LM is and how BERT benefits from it in contrast to previous work. Lastly, it will be discussed how we can fine-tune a trained BERT model for transfer learning and how we can extract BERT's contextualised representations for word embeddings.

As previously mentioned, BERT uses a transformer architecture which is fundamentally based on an attention mechanism, thus BERT is able to learn contextual relations between words in a given sentence. The original transformer architecture, proposed by (Vaswani et al., 2017), consists of two parts, encoder and decoder. The encoder processes a textual input and produces an abstract representation, whereas, the decoder is used to produce a prediction for the task. Since BERT's objective is to learn a language model, only the encoder part is necessary.

BERT stacks multiple encoder layers on top of each other, where the output of the previous layer is fed as an input to the layer above. All the encoder layers are identical in structure, however, use different weights, thus each layer up the ladder recomputes the previous layer's output and improves on the overall quality of the final embedding. Furthermore, each encoder layer is made up of two sub-layers, as illustrated in Figure 13a. The first layer consists of a self-attention mechanism, which was discussed in section 3.2.3. Once the contextual representations from the self-attention layer have been produced, the model processes all of them using the same feedforward layer. This way the network can regulate contextual information by making more relevant context stronger. In addition, each of the sub-layers has a residual connection and a layer-normalisation step. As illustrated in Figure 13b, the residual connections add the input to the output of the self-attention module. The resulting vector could increase in the value range, hence, needs to be normalised before it is passed on further. The residual connections help to maintain the inputs' original information, hence allow for stable gradient flow in the multi-layered process.

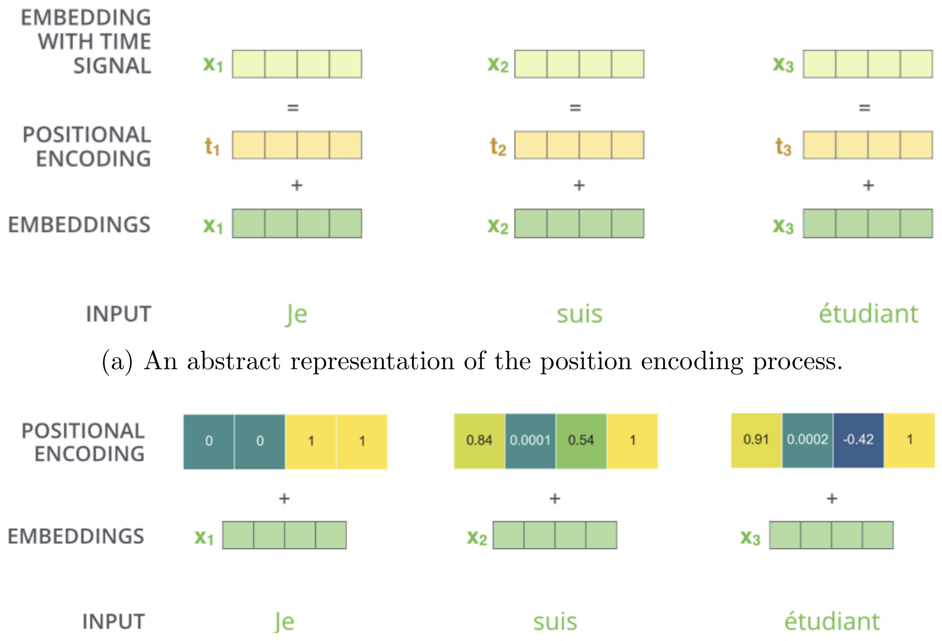


(a) An abstract representation of encoder layer.

(b) A more defined encoder layer representation.

Figure 13: The encoder layer representing inner two layers. From Alammar (2018).

One aspect that hasn't been mentioned yet is the input representation. The input is a sequence of words, which are first embedded into vectors using WordPieces (Wu et al., 2016). This embedding approach provides subword structure information at the input layer and supplies a meaningful representation for words that have not been seen during the training process. Furthermore, as opposed to directional models (LSTM), which sequentially process input in one direction, the TE reads the whole sentence at once. This characteristic allows the self-attention layer to learn the context from both the left and the right direction at the same time. However, unlike the sequential models that naturally encode the input sequence, TE needs a way to represent the input order, thus, uses position encoding. The positional encoding is a vector that is added to each input element (see Figure 14a). These vectors follow a specific pattern that the model learns and uses to determine the distance between different words in the sequence. The intuition is that the self-attention Q, K, and V projections reflect the position pattern, thus, the latter calculated word scores are higher for vectors with somewhat closer vector representations. Figure 14b illustrates the position vectors. Note that the further the words are apart, the larger the value difference at each encoding element. The same encoding process is applied to every input before it is fed as an input to the first layer.



(a) An abstract representation of the position encoding process.

(b) A toy representation of the sequence encoding vectors.

Figure 14: The input layer representation. From Alammar (2018).

It is believed that a deep bidirectional model is more powerful than either a left-to-right model or a shallow concatenation of a left-to-right and right-to-left LM model. Unfortunately, the standard LM can be trained in one direction only, since bidirectional conditioning would indirectly allow each word to "see itself" in a multi-layered context. Thus, BERT authors argue that the current traditional LM techniques restrict the power of the pre-trained representations. For example, Radford et al. (2018) have proposed a model which uses a transformer with a forward LM objective that predicts a word given left context only. Such an approach has shown to be sub-optimal because incorporating context from both directions is crucial for sentence-level tasks like SQuAD question answering (Rajpurkar et al., 2016). Furthermore, recent work by Peters et al. (2018), has shown to use two shallow LSTM networks with a traditional LM objective to separately predict words given context from one direction and then conditioning on a joint representation. This approach provides a bidirectional LM, however, it requires two models to separately compute predictions in one direction, imposing a high computational cost and a parallelism constraint.

To address the aforementioned restrictions, BERT has proposed a new pre-training objective, a Masked Language Model (MLM). Before the input is fed into the network, 15% of the sequence is randomly replaced with a [MASK] token. Then, the model's objective is to predict the original vocabulary ID of the masked tokens based on the non-masked context. To do this, the final hidden vectors corresponding to the masked tokens are fed into the softmax layer, which predicts a word from the vocabulary (similarly to the standard LM). Although this approach allows for bidirectional prediction, it introduces a downside. BERT's loss function only takes into consideration the prediction of masked words, which means that the non-masked words are left out. As a consequence, the model converges slower than the standard LM, which predicts every word. Despite only conditioning 15% of the input, the authors show that MLM yields higher training accuracy earlier in the training process as opposed to the standard LM.

In addition to the MLM, the authors also introduce the "next sentence prediction" task that is jointly pre-trained with the MLM. The sentence prediction is a binary classification problem that is tasked to predict whether or not sentence B is a continuation of sentence A, that the model pair-wise takes as an input. Incorporating this additional training objective is beneficial for many downstream tasks that require sentence relationship understanding, that isn't directly captured by just the LM, e.g., question answering and natural language inference. Aside from the benefits the additional task brings, it is also easy to generate its corresponding labels. The training data is created from a large corpus of text by assigning a 50% chance for the next sentence to be a continuation, as well as, 50% to be a random sentence. Consider the following example, where the [CLS] tag indicates the start of the input and [SEP] separates the sentences:

- Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
  Label = IsNext
- Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight less birds [SEP]
  Label = NotNext

There are two strategies for applying pre-trained language representations to downstream tasks: feature-based and fine-tuning. The feature-based strategy, similarly to ELMo, works by using BERT to create contextualised word embeddings and then learning the task-specific model from scratch based on those extracted representations. Furthermore, the contextualised word embeddings can be added or concatenated with different word embedding. Each layer's output represents the same input, hence we can extract a representation from any intermediate layer, which can be used in an arbitrary way. The authors found that concatenating last four hidden layer representations yields better accuracy on the named-entity recognition task than taking an average of the last four hidden layer outputs. In fact, the concatenation strategy scored 96.1% accuracy, which is 0.3% behind fine-tuning the whole model for the exact same task.

Inspired by transfer learning in the computer vision field and the Universal Language Modelling Fine-tuning (ULMFiT) by Howard and Ruder (2018), the pre-trained BERT model can be fine-tuned to a wide range of tasks with just one additional output layer, to create new state-of-the-art models. Many of those tasks require a labelled dataset, which often are very limited in size since most of them have to be created by a human. By fine-tuning, ULMFiT have shown that only 100 labelled examples match the performance of training from scratch on ×100 more data.

To summarise, BERT is a work inspired by recent advancements in the NLP field, in particular: ELMo's bidirectional language model, transformer architecture that utilised self-attention mechanism to build contextualised representations and ULMFiT that introduced a process for transfer learning. Thanks to the concatenation of those ideas, BERT produced new state-of-the-art results on eleven competitive tasks. In order to allow for bidirectional language learning using the transformer architecture, BERT introduced a new technique called Masked Language Model. The MLM uses a masking tag [MASK] to randomly cover up a section of the input sequence and then uses the uncovered context to predict the masked tokens. Furthermore, BERT also incorporates another supervised task, called Next Sentence Prediction, to jointly train with MLM. The additional task is a binary classification problem, which requires the model to identify whether the right part of an input (sentence B) is a continuation of the left part of the input (sentence A). This way, BERT models sentence relationship knowledge, which is required for many downstream tasks that benefit from abstract-level understanding. In general, BERT can be seen as a universal model that has a strong linguistic knowledge, which can be transferred to any NLP task by adding one classification layer on top. Hence, extracting word vectors or fine-tuning BERT model parameters, provides a promising building block for existing applications to improve.

3.3.4 GPT2: Generative Pre-Training

Generative Pre-Training 2 (GPT2) model is a successor to BERT for generative tasks (Radford et al., 2019). Similarly to BERT, at the core of GPT2 stands the transformer architecture that utilised self-attention to compute the representations of the input. Since BERT internal architecture has been explained in-depth in Section 3.3.3 and the GPT2 is based on the same concepts; the explanation will focus on the architectural differences rather than repeating the whole mechanism. In contrast to BERT, GPT2 uses a standard-LM objective on top of the self-attention representations to learn the language model. The standard-LM objective learns to predict the output given a joint probability of the previous sequence representations (condition), which makes the GPT2 auto-regressive[5] in nature. On the other hand, BERT trades the auto-regression for the ability to incorporate context from both directions, which results in better performance at the NLU tasks. However, the lack of auto-regression makes BERT output sampling not obvious. Therefore, the main concept that sets the GPT2 and BERT apart is the learning objective.

Given that the self-attention mechanism incorporates the bidirectional context to compute a representation for a token at a particular position, the GPT2 uses its modified variant called masked self-attention in order to main the auto-regression. The masked self-attention differs from the original version in that it hides the future context for representation computation at a particular position; therefore, the mechanism only attends to the previous sequence, which accommodates the standard-LM objective used by the GPT2. If one used self-attention representations computed using the bidirectional context with the standard-LM objective, the model would see the words it is trying to predict in the representation of the previous sequence, which would make the training banal that would results in a model not generalising properly. Figure 15 contrasts the two attention mechanisms.

[5] ”Auto-regression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step.”

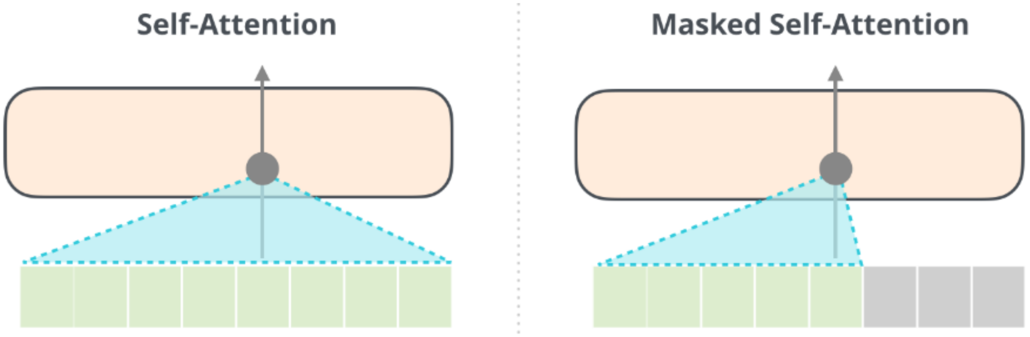


Figure 15: A contrasting illustration of the self-attention mechanism used by BERT (left) and GPT2 (right). The Masked Self-Attention covers the future context when computing a representation for a token, whereas Self-Attention utilised the bidirectional context. Diagram from (Alammar, 2019)

Furthermore, the GPT2 uses a massive dataset of 40GB of text called WebText, to pre-trained the model. At the time of publication, this was by far the most extensive dataset a language model was trained on. The authors scraped the data from Common Crawl website, which is an open-source of diverse textual data from a range of domains. By training the GPT2 model on such a large amount of diverse data, the model facilities well to a range of fine-tuning tasks, e.g., question answering, article generation and text summarisation. Moreover, by improving the state-of-the-art at seven competitive tasks without any task-specific fine-tuning (using pure pre-trained model), the achievement underlines the significance of large training data required to improve the task-specific transferability performance of a pre-trained model (Radford et al., 2019).

The authors have produced four GPT2 model distributions: small, medium, large, and extra-large. What sets the model distributions apart is the performance gained by the higher number of transformer-decoder blocks and hidden state outputs used by the larger distributions. The smallest model stacks 12 transformer-decoder blocks and uses hidden state outputs of 768 neurons (117M computational parameters), whereas the medium model stacks 24 layers and uses 1024 hidden states (345M). The large distribution model is made of 35 layers whose outputs are of size 1280 (762M), and lastly, the extra-large model stacks 48 layers of 1600 hidden states size (1542M computations per input). Note, the performance gained by the extra layers and hidden states is thanks to extra blocks having its own weights, in both, the self-attention and neural network sublayers, that get learnt in the multilayered process.

The smallest and medium model distributions are publicly available; however, in the following experiments, we utilise the smallest distribution because of the computational power limitations.

To summerise, GPT2 is a transformer-based network that utilises an auto-regressive standard-LM objective that allows for output generations in a straight-forward sequential fashion. In order to accommodate the sequential training objective, GPT2 uses a modified variant of the self-attention mechanism, called masked self-attention, that hides the future context at a token representation computation, so that the model can not indirectly see it, in the multi-layers process, at the next word prediction. At the time of publication, GPT2 advances the performance at seven tasks without task-specific fine-tuning, that is thanks to training on a large and diverse training dataset size (40GB), which shows the significance of large amounts of data required for better-performing pre-training model.

3.4 Summary

To conclude, it has been shown how word2vec, ELMo and BERT use different language modelling objectives in order to capture linguistic properties. Word2vec uses a fixed sliding window size around a pivot word in order to capture the context. At the time of publication (2013), it was reasonable to limit the context since the hardware available was much slower. Furthermore, word2vec offered a shallow and efficient neural framework for LM that has reduced the computational time of creating good quality word embeddings. Using word embeddings at the input layer of downstream tasks has provided linguistic information, hence, they became widely used in the industry as a building block for language understanding applications.

ELMo (2018) has addressed the limited context issue by using two LSTM networks, where each sequentially processes the full sentence in one direction while producing an output for each element in the sequence. Then, the unidirectional output representations for each element are concatenated in order to form bidirectional language modelling representations. By considering full sentence context from both directions ELMo has produced superior results over the previous methods for six language understanding tasks, which emphasised the importance of bidirectionality in LM. However, the LSTM's inherent sequential nature restricts the parallelisation within training, which becomes critical at longer sequence lengths. Hence, a shift to a more parallel architecture was required in order to allow for faster computations and deeper architectures.

BERT (late 2018) is built upon recent ideas, which include: a bidirectional LM, transformer architecture and a task-specific fine-tuning of a pre-trained model. The transformer model takes the whole sentence as an input at once and uses a self-attention mechanism to compute word relationships. Furthermore, the transformer network can serve as a complete replacement of the LSTM as it allows for sequence dependency modelling without regard to the distance in the sequence. Nevertheless, due to attention's application flexibility, BERT, in contrast to ELMo, stacks multiple attention layers that collectively improve the contextual representations. Since BERT takes the whole input at once, the unidirectional LM objective would not work as the words could see themselves in the multilayered process, which would make the sequential word prediction pointless. As a solution, BERT has introduced a new language objective, i.e., to cover 15% of the words using a [MASK] tag and then conditioning on the uncovered context to predict the masked words. By using the new bidirectional LM objective in a deeply layered process, at the time of publication, BERT has pushed the benchmarks at multiple tasks.

Generally, since language models benefit from large amounts of unlabelled data to capture grammatical and semantic information, it makes sense to use

BERT as a universal model that can be fine-tuned to a task-specific problem. Fine-tuning, in contrast to word embedding extraction, gives the possibility of biasing the output representations to the specific problem. However, it is also possible to fine-tune a model and then extract the task-specific embeddings. In contrast to word2vec embeddings, BERT and ELMo embeddings are a function of the sentence, meaning that if we provide the same word in two different contexts as inputs, we would get different representations of the same word. Whereas, a word2vec embedding for a word is always the same in a different context. This provides a promising foundation for BERT to become a replacement of word2vec embeddings that are widely used in the industry.

The model performance has been shown to increase with the size of the training dataset (Radford et al., 2019). Despite the fact that the recent architectures have become more efficient, training these models to state-of-the-art quality is still inaccessible to most of the community. For example, GPT2 has been trained on 256 Google cloud TPU v3 cores, for several days, which is equivalent to £40,000[6] (a single training).

[6] Link to HCP price calculator: https://cloud.google.com/products/calculator/

4 Preliminary Experiments

4.1 BERT-Base vs BERT-Large: Comparison

This section compares the performance of two open-sourced BERT model distributions: a smaller version called BERT-Base (BB) and BERT-Large (BL). The BB model stacks 12 transformer layers, where each has 768 hidden neurons and uses 12 multi-head attentions, thus computes 110M parameters per input. Whereas the BL stacks 24 layers, has 1024 hidden states and 16 attention heads, which results in 340M computations. The model, along with the pre-trained weights and pre-trained Masked-LM head, can be obtained from a library called transformers[7]. By comparing the two model distributions, we will explore the new Masked-LM objective in more details as well as see the performance gained by the larger model. Comparing the performance of the two models that considerably differ in size provides a perception for one to decide whether the gains are worth the expenses pulled by the larger model training.

The performance of each model gets tested by the amount of correctly predicted word-tokens using Masked-LM (see Section 3.3.3) as an objective function that the models were pre-trained with. The objective function masks a certain amount of word-tokens that the model has to predict given the unmasked contextual words.

In more detail, the prediction process looks as follows: First, we select a text with a word length smaller or equal to 512. Then we mask n words that we will predict by simply replacing the actual words with a [MASK]. It is essential to keep a reference of the words we are replacing since they will be later used as the labels. Then, the masked input needs to be encoded into token-ids, where each word, including the [MASK] token, has its corresponding token id. Running the encoded input through the model produces a set of hidden states for each word. These hidden states representative of the words are then independently run through the pre-trained Masked-LM head,

[7] Link to huggingface docs: https://huggingface.co/transformers/

which returns a probability distribution over the vocabulary for a particular set of hidden states. Then, in order to derive the prediction for each masked position, we use a softmax over the vocabulary probability distribution. The softmax returns the model's prediction for that particular position, which then gets compared to the corresponding label.

In this experiment, the number of masked tokens gets increased until the maximum coverage of 80% of the sentence length. The experiment is repeated 500 times for each amount of masked words, where at each training iteration (i.e. epoch) the masked words are chosen at random. In order to keep the comparison fair, the models take the same text as an input; however, the word masking selection is random. Figure 16 illustrates the results of the comparison.

When choosing the text for the predictions, it was aimed to find a short descriptive story with an average complexity level. For example, a sequence that does not contain many names, over-complicated words and punctuation like quotes, since such occurrences would make the prediction difficult. The chosen candidate text, *Text A*, comes from the "Beauty and the Beast"[8] narrative and its snippet looks as follows: "Once upon a time, there was a girl named Beauty. She lived with her father and her sisters in a small village. Beauty was a beautiful girl. She was also hard-working. She always helped her father on the farm...". The full story can be viewed in Appendix A.1.

[8] Link to text A orgin website: https://www.scribd.com/document/44455701/Example-of-Descriptive-Text

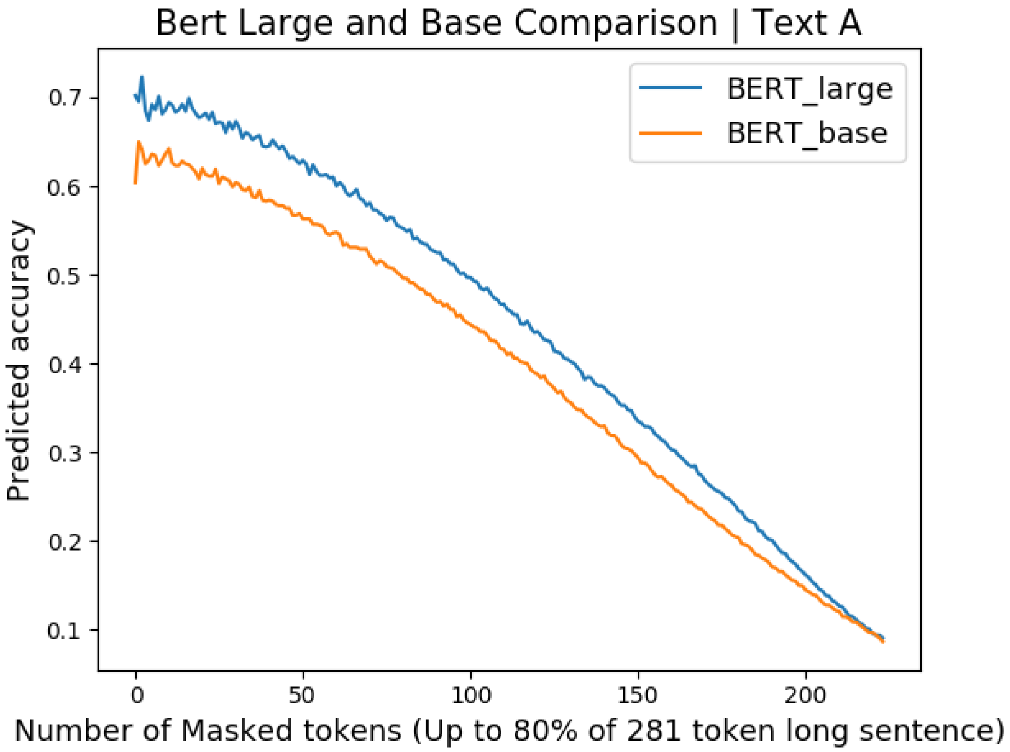
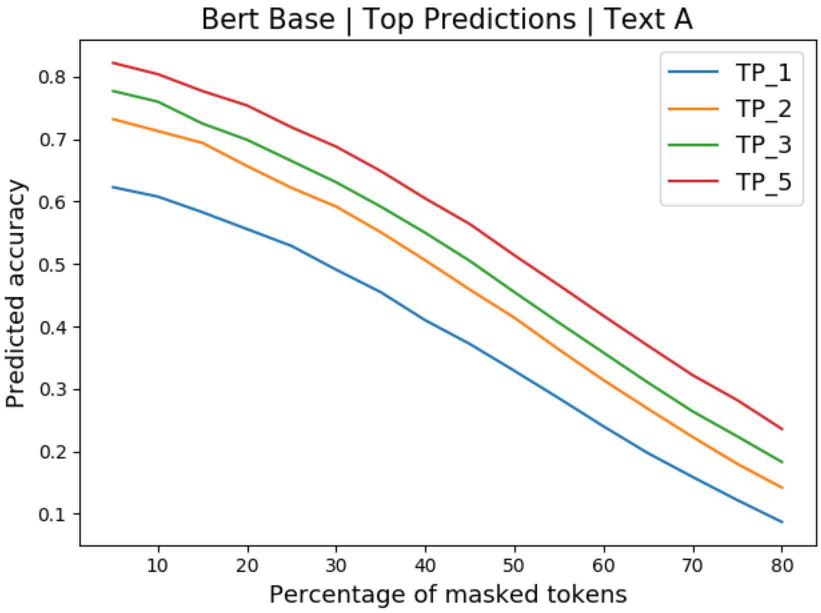


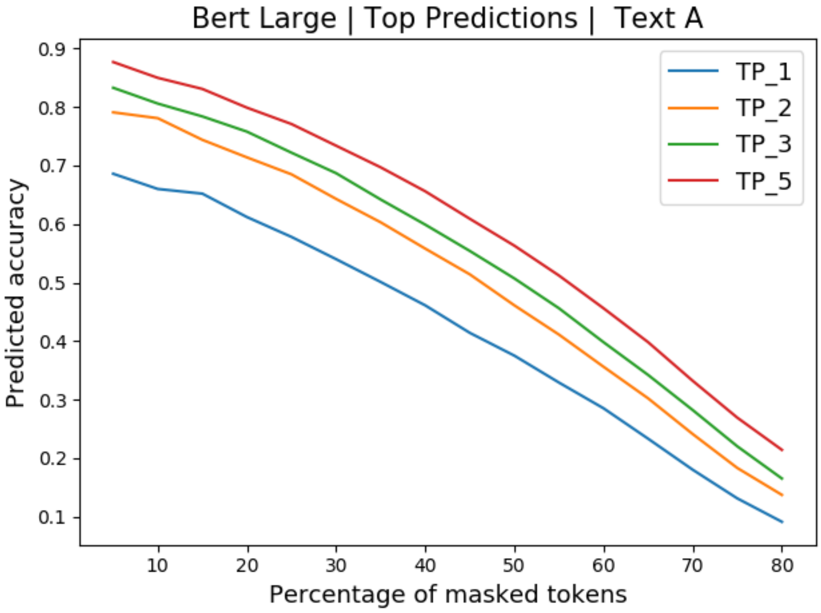
Figure 16: Prediction accuracy from masked language modelling applied to BERT-Large and base model distributions.

Intuitively, the biggest model performs better by approximately 7% for low number of masked tokens. However, as the masked tokens coverage increases, the prediction accuracy of both models converges. This behaviour is natural since as we increase the number of masked tokens the length of surrounding context decreases; thus, the models are more prone to error.

A further investigation into the predicted words shows that some incorrect predictions fit into the sentence just as good as the actual labels. For example, given the masked word "father" and the context"She always helped her [MASK]" the models first predicts "mother" and then "father". In order to allow for such behaviour, in the second experiment, top n predictions are considered. Figure 17 illustrates the top-n prediction accuracy.



(a) BERT-Base.



(b) BERT-Large.

Figure 17: The word prediction accuracy for top 1, 2, 3 and 5 model's predictions, using a Masked-LM objective with an increasing number of masked tokens.

As the results show, by only considering the top 2 predictions, the prediction accuracy increases by approximately 10% for both models. Moreover, the model's first conclusion is often a synonym of the actual label, which also fits into the context. Increasing the top n predictions transitions into a higher prediction accuracy whose absolute improvement tends to shrink with the growing n. Therefore, this experiment shows that when using MLM for tasks like language generation, considering the top few predictions for sentence construction improves the quality of the final output. Moreover, the experiment demonstrates that for tasks like language modelling, covering more than 20% of the input tokens decreases the prediction performance, since the accuracy slope for all experiments starts to get steeper after that point. However, covering tokens until the margin of 20% shows not to harm the prediction accuracy by much and provides fundamentals for experimental investigations.

Interestingly, the BERT paper authors have pre-trained the model by masking 15% of the inputs; however, the results suggest that there is an experimental margin up to 20% that could potentially improve the final prediction accuracy of the pre-trained model. Intuitively, increasing the masking percentage involves a natural trade-off between known information to predict the unknown. The more tokens covered the less information and thus the more laborious task for the model. Trying to solve a harder task at the training time could potentially transition to a better performing model. Furthermore, a higher masking ratio is also a trade-off between the prediction accuracy and the training time since the more tokens covered per input, the more teacher's signal and thus the faster the learning.

If one has enough resources to train a BERT model from scratch, a different Masked-LM strategy could potentially yield an improved performance.

4.2 BERT: Token Representation Interpretability

In this section, we are going to inspect and interpret the BERT's contextual representations, also known as the hidden states of the final layer, in order to state the level of *interpretability* - *"In the context of ML systems, interpretability is the ability to explain or to present in understandable terms to a human"* - Finale Doshi-Velez. The model's output can be considered interpretable when humans can relate and associate it to how we understand and make sense of concepts in the environment we live in. Word2Vec embeddings are interpretable since when manipulating the word's representations we can derive meaningful associations which make sense for us. In particular, this experiment will show whether BERT contextual word representations are as interpretable as Word2Vec word embeddings: i.e., whether performing arithmetic operations on the vector representations would result in new semantically similar vectors that are reasonable for us. For example, whether $king - man + woman$ (KMWQ) results in a vector close in distance to "queen".

In order to state the interpretability, we are going to follow two different approaches, context-free and context-based. The context-free approach experiments will be conducted on BERT representations constructed using single word inputs, whereas the latter will involve experiments with representations of words in context. Lastly, we are going to investigate the influence a single word can have on all of the representations of the contextual words, as well as the impact of different word order.

4.2.1 Context-Free Approach

The first step is to extract the features for the words "king", "man", "woman" and "queen", devoid of any context. The features are extracted from the respective model's last layer from the particular word's index. Note that BERT has been pre-trained to construct contextual word representations and thus heavily relies on the context in output construction; however, in this experiment, we ignore this fact and first experiment with out-of-domain context-free vectors. Furthermore, BERT uses a [CLS] tag to encode the start of the sentence and a [SEP] tag to encode the end; however, these are not included in these tests. That is because these tags are added to every input during the training process, which would introduce a stronger bias towards the representations of words that appeared more frequently. Furthermore, after the particular word's features extraction, a cosine similarity function is used to calculate the distance between two vectors. The cosine function will return a scalar value between 1 and -1, where 1 means vectors are completely the same and -1 completely different. The results are illustrated in Table 2.

Cosine distance between word vectors: no context		
Comparison Between	Base	Large
queen vs queen (Q)	1.0	1.0
king vs queen (KQ)	0.609	-0.079
woman vs queen (WQ)	0.347	0.288
man vs queen (MQ)	0.358	0.824
KMWQ vs queen (QQ)	0.458	-0.476

Table 2: Cosine similarities between pairs of contextual word representations computed without any context.

The comparison *KQ* from BB, puts king quite far in space from the queen, indicating that there is no semantic relationship between the pair. More interestingly, the cosine distance in BL is -0.079, which puts the king on the opposing side of the axis and means that the representations are strongly uncorrelated. Furthermore, we can also see that in both models, the *MQ* similarity score is greater than *WQ*, meaning that the word man has a closer relationship to the queen than the word woman, which is completely opposite to how we interpret this relationship. Lastly, the *QQ* comparison also puts the arithmetic queen far in space from the raw queen representation, which is unsurprising given the not interpretable representations of the previously inspected words. Furthermore, the differences in values between the two models can be explained with the varying hyperparameter setting (capacity) and training data size, and can suggest that the BL model is underfitted.

All of the previous comparisons prove the fact that BERT struggles to construct interpretable representation without context.

4.2.2 Contextual Approach

In this experiment we are going to repeat the same procedure as in the previous test, however, with the difference of target-words being computed in the setting of the following context: *the <target word>went to sleep.* Since BERT has been trained to represent a word given its contextual words, the now added static context should produce more interpretable representations for the target-words. The results are illustrated in Table 3

Cosine distance between word vectors: with context		
Comparison Between	Base	Large
queen vs queen **(Q)**	1.0	1.0
king vs queen **(KQ)**	0.882	0.907
woman vs queen **(WQ)**	0.799	0.836
man vs queen **(MQ)**	0.822	0.826
KMWQ vs queen **(QQ)**	0.798	0.858

Table 3: Cosine similarities between pairs of contextual word representations computed in the context of: *the <target-word>went to sleep.*

By inspecting Table 3, we can see that the new word representations computed in the setting of the context now score higher in the similarity comparison. Such results not only prove that BERT needs context to define words but also that the representations are to some degree interpretable, i.e., the vectors reflect the real-world semantic relationship between the target-words. Taking a closer look into the BB section, the *WQ* scores lower than *MQ*, meaning that the man has a closer relationship to the queen than the woman does. Furthermore, in the case of both models, the arithmetic queen *QQ* is still further in space than any other target-word. Such results suggest that the contextual representations are not convincingly successful in associating words in the context of space manipulation.

Let us see what are the results when we extend the context so that our target-words can be more defined. Would such extent improve the representation of the QQ in such a way that it will be more similar to the queen's representation than any other?

Cosine distance between word vectors: with extended context		
Comparison Between	Base	Large
queen vs queen (Q)	1.0	1.0
king vs queen (KQ)	0.893	0.482
woman vs queen (WQ)	0.840	0.559
man vs queen (MQ)	0.832	0.519
KMWQ vs queen (QQ)	0.906	0.478

Table 4: Cosine similarities between pairs of contextual word representations computed in the context of: *the <target-word>was tired so (s)he went to sleep*.

After adding the new context, the BB finally starts to exhibit favourable behaviour, i.e., WQ is closer than MQ and QQ is closer to the queen than any other vector in the comparison. On the other hand, BL in contrast to BL in Table 3, moves further from the queen for all of the comparisons, thus showing that the extra context can also have an unpredictable and negative effect. The negative effect can be explained with the randomness in the training dataset, i.e., during the training phase, the new words must have appeared more frequently in the context of other words which have modelled its representation to be different. Therefore, adding words with very different representation can drastically change the contextual representations when compared to its representations computed in the setting of partial context. Furthermore, the BL unpredictability may suggest that the BL model is underfitted, since it does not associate concepts as effectively as the BB model.

BERT is a model that uses a Transformer based architecture that is entirely based on an attention mechanism, which allows for bidirectional and deep-in-layers language modelling. Word representations from such a language model are a function of the entire sentence, meaning that the representation of the same word can differ, depending on the contextual words surrounding it. That is an advantageous asset since natural language is complex, and many words are, for example, monomorphic, however: BERT knows how to disambiguate them, thus, its superior performance at word/sentence classification tasks. On the other hand, the complexity of contextuality, in contrast to fixed word embeddings, makes the model space manipulation questionable, i.e., whether it is still possible to find word associations using contextual word embeddings.

The experiments in this section have shown that BERT is also capable of computing context-free representations; however, using these representations to find meaningful associations has proven to be insignificant and unsuccessful. Since BERT has been trained to construct a representation of a word using its context, adding a limited context has shown to improve the target word's representations as it started to exhibit a real-world relationship, just like in the case of word2vec embeddings. Furthermore, for BERT-Base, extending the context has shown to improve the interpretability even further, as, for example, the arithmetic queen vector was the closest to the pure queen vector, which is what we would expect. However, for BERT-Large, the extra context has shown to unexpectedly spread away two representation that we would expect to approach closer in space. Therefore, the BERT-Large irregularities in cosine similarity scores, when compared to BERT-Base, suggest that the model is underfitted, since is unable to provide reasonable associations between the related word's representations. Furthermore, such irregularities in the representations also indicate that BERT output construction can be very unpredictable, as by adding a few extra words to the context, the context words representation can drastically change in a un-interpretable manner. To conclude, contextual representations can provide meaningful associations when inspecting the final hidden state representations; however, they can be unpredictable and thus unreliable. Since contextual embeddings are too dynamic for tasks like model's space manipulation, it is better to use more straightforward and reliable embeddings, like word2vec.

Note that the choice of words plays a big part in this experiment, as so does the similarity metric. On further investigation, we are going to examine the similarity of words based on the sentence level measurement strategy.

4.2.3 Order-based Approach

As BERT is bidirectional in the sequence and layers, the order of the words should matter. In order to test this, we are going to extract representations for the word "queen", used in the same context but with a different order of words. The context is mentioned below:

(a) the queen was tired so she went to sleep *(original)*
(b) the queen was sleep so she went to tired *(two words swapped)*
(c) sleep to went she so tired was queen the *(reverse)*
(d) tired the queen was so to sleep she went *(random)*

Cosine distance between word vectors: with context		
Comparison Between	Base	Large
a vs a (AA)	1.0	1.0
b vs a (BA)	0.852	0.812
c vs a (CA)	0.662	0.409
d vs a (CA)	0.610	0.230

Table 5: Illustrates a cosine similarity between the *queen* vectors in the same context but in different order. The contexts are listed above as a, b, c, d.

Table 5 shows that the order does matter. In fact, the more random or previously unseen the order of the words is, the less the representations for the word "queen".

4.2.4 Word Influence on Contextual Representations

It is unclear how we can precisely measure a difference between two contextual representations since every single word in the sentence impacts the contextual representation of all the other words. One reasonable step in this direction is to include the sentence-level information in the comparison between two word representations. Applying cosine distance metric on the sentence-level, we can compute the similarity between the representations of words in the sentences, i.e., in all-to-all relation for every word. Figure 18 shows the cosine scores computed between two sentences consisting of the same words.

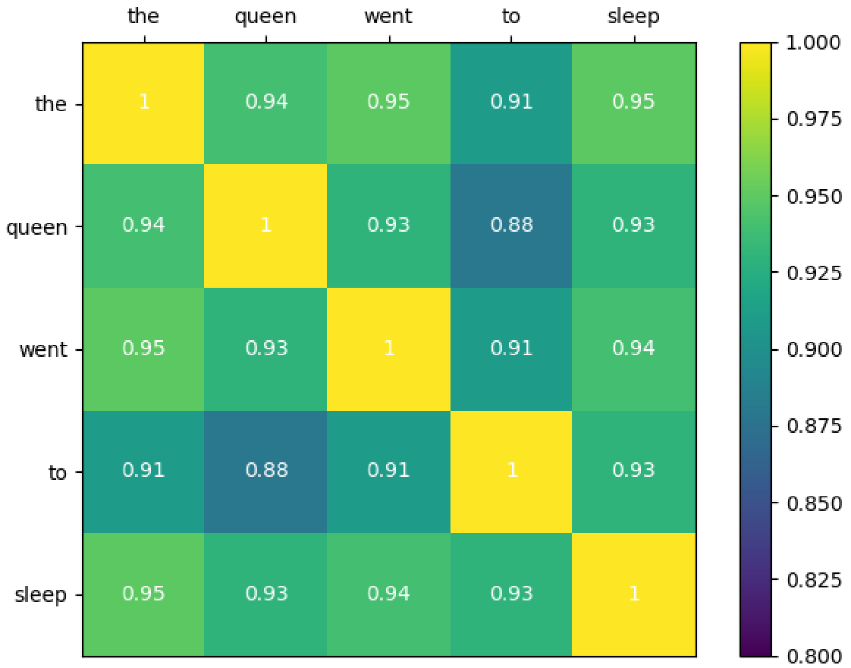


Figure 18: Cosine similarly scores between the pairs of word representations, in all-to-all relation for every word in each sentence, from BERT-Large. The overall average similarity score is: 0.940. The Average Influence Similarity (AIS) score: 1.0 (introduces later)

By looking at the scores, it is notable that the top-down diagonal cosine similarity scores are all equal to 1. This is because they compare the same words computed in the same context. Although the sentences are the same, it is also clear from the heat map that all the other scores depart from 1. This is natural since, e.g., the contextual representation of the word "the", should naturally differ from the word "sleep". This fact indicates that the non-intersecting words cosine scores should not participate in the sentence-level similarity comparison, i.e., averaging such values would falsely downgrade the score. In other words, having the two same sentences, the metric would never produce an expected score of 1. However, using the cosine similarity scores of the intersecting words between the two comparison sentences provides a reasonable ground for a pair-wise contextual representation's similarity measure.

Now, consider all-to-all cosine similarity scores between words in two sentences, that only differ in one position in the sequence (see Figure 19). It is visible that the scores between the same words differ, i.e., the different word has influenced the representation of all the other words, and thus, when computing the cosine similarity, the values depart from 1.

Furthermore, given the two same sentences, which only differ at one particular position in the sequence, we can calculate a more precise similarity of the two differing words by averaging the cosine similarities of the intersecting contextual words. Since there is only one different position between the sentences, the average of the intersecting context words (excluding the differing words representation cosine similarity) also serves as a measure of the influence the differing word has on the representations of the contextual words. In further explanations, the aforementioned sentence-level word comparison metric will be referred to as *Average Influence Similarity* **(AIS)**.

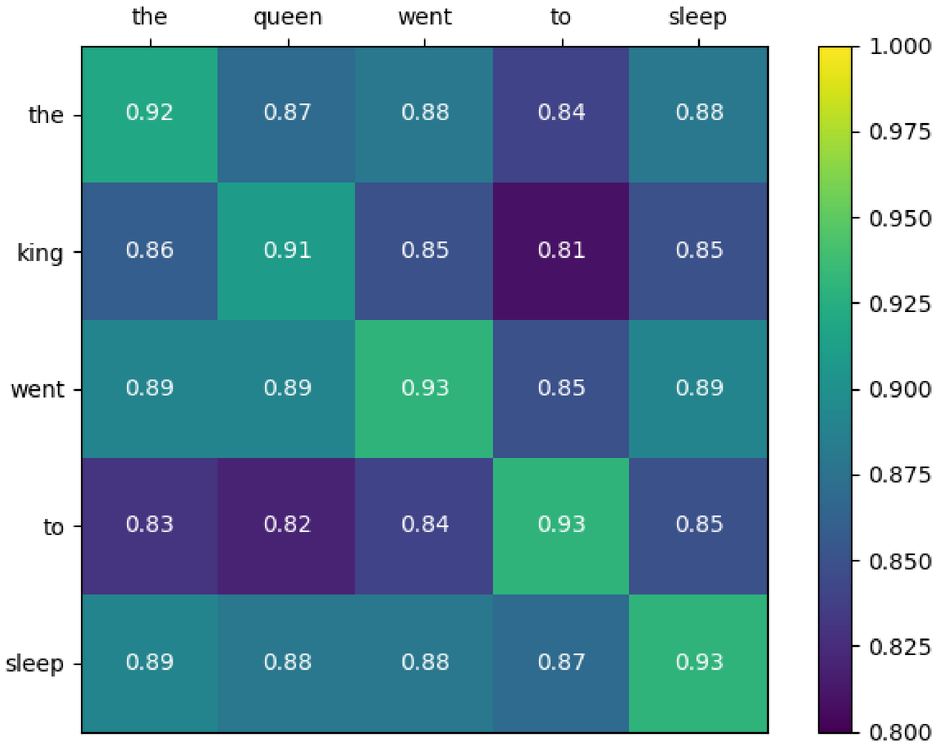


Figure 19: Heat map visualising the cosine similarly scores between the pairs of word representations, in relation all-to-all words in the sentence, from BERT-Large. The overall average similarity is: 0.874. The Average Influence Similarity score is: 0.927

Applying AIS to the sentences displayed in Figure 18 results in an accurate prediction of 1, since the sentences are exactly the same. In contrast to the pure average cosine similarity score, 0.94, this metric is more truthful since one expects the two same sentences to get the maximum similarity score. Furthermore, using AIS to compare the differing words in the sentences visible in Figure 19 results in a score of 0.927. In contrast to the score of 0.874 for average cosine similarity, the AIS seems more reflective since queen and king are commonly perceived as being conceptually close, thus should display a high value. However, it is not possible to stress which score is better because the relationship between the two words is a subject to an opinion.

Despite the AIS metric showing reasonable scores, it suffers from substantial restrictions. It is only possible to compare two word representations at a time, given the same context. This creates a few significant problems. The choice of the context may be biased towards one particular representation. In terms of the other issue, sometimes it may be challenging to find two different words that would fit into the same context. Furthermore, another side problem is that the metric is also computationally expensive since it is required to calculate the cosine similarities between all the intersecting words. Nevertheless, when computing AIS between two the same words in multiple different contexts, the AIS scores would differ, i.e., the score is depended on the context. However, one could compute AIS for two words by approximating the score from multiple scores calculated in different contexts, that could be retrieved by searching a textual corpus.

4.3 Sequence Generation from BERT

Bidirectional Encoder Representations from Transformer (BERT) have been designed to pre-train deep bidirectional representations by jointly conditioning on the left and right context in all layers. The resulting representations capture grammatical and semantical information that is fundamental for any NLP/NLU task. In order to benefit from BERT's language rich information modelling, we simply add an additional layer on top, which we later fine-tune using a task-specific learning objective and data. The knowledge gained at the pre-training step helps the fine-tuning layer reach a satisfiable accuracy on a wide range of NLP tasks using limited dataset sizes. In fact, BERT has shown to achieve state-of-the-art on eleven tasks, which include: sentence classification, token-level classification, inference, and question answering. However, all of the tasks belong to NLU group which models the learning space of the fine-tuning layer into classes. The authors did not conduct any experiments on Natural Language Generation (NLG), which is an equally important group of tasks that serve as fundamentals to solving many problems, e.g., text summarisation and article generation.

BERT is pre-trained using Masked-LM that learn to predict masked tokens given the left and right context, however, during the generation phase, it is unclear how the same objective can be used to generate a words sequence from scratch[9]. Furthermore, it is also uncertain how we can use Standard-LM objective to generate from BERT because such an approach imposes a pre-specified generation order that only uses left context. In other words, the bidirectional nature of BERT does not naturally admit sequential sampling. However, given that the dominant approach to language generation is left-to-right, this section will experiment generating from BERT in that manner. The following sampling strategies are also illustrated in Figure 20.

In the following experiments, we are going to test different strategies for generating from BERT. All of the experiments begin with a starting conditional sentence, where for $n \in \{1, ..., T\}$ steps we sequentially append a masking token to the sentence, generate a word for that position, and substitute it into the sentence for the next time step t. After T steps, we end up with a generated sentence based on conditional words. All of the experiments differ in the choice of the masking position and the context used to generate a prediction. In order to predict for a given masked, as in the previous experiments, we softmax over the hidden states of a particular mask.

In the first test, we provide a starting sentence, "Once upon a time, there was a girl named Beauty. She lived with her father and her sisters in a small village. Beauty was a beautiful girl. She was also hard-working. She always helped her", where at each time step we append a masking token at the end of the sentence and thus, use the left-only context to predict the sequence. In this scenario we would expect the model to generate something along the lines "father and sisters on the farm", but it is almost never the case since the model generates tokens which are not coherent, collapse into a repetitive

[9] At least at the time of writing this thesis, it is unclear how we can generate from BERT while achieving a satisfiable performance.

loop and are often a part of punctuation vocabulary, e.g. five full stops in a row.

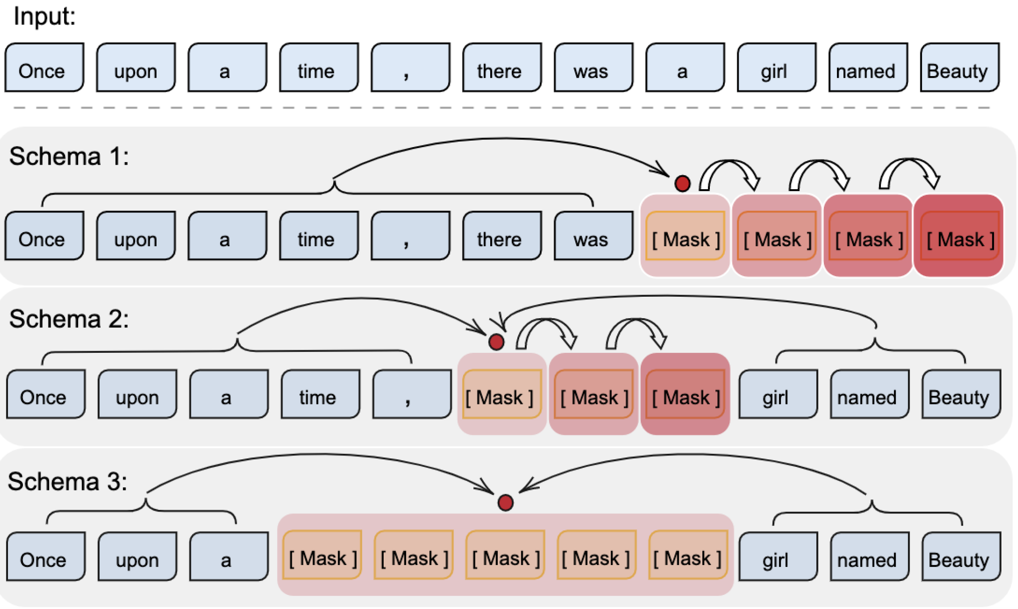


Figure 20: An abstract illustration of three BERT token prediction strategies investigated in this experiment (best visualised in colour). Schema 1: displays a sequential token prediction given left context only. After predicting a word, the word and a new masked token are added to the end of the sequence for next token prediction. Schema 2: context from both directions is reflected in the masked token representations, from which we sequentially predict. Schema 3: using context from both sides, we compute representations for all of the masked tokens at once, from which we later predict (similar process to the Masked-LM objective). Note that the red dot indicates a start of the sequence prediction. The different shades of red refer to different time steps at the prediction.

In the second strategy, we split the conditional sentence into two parts, left and right context. Then we append the masking token to the left context and before running it through the model we join both contexts into one. This way we predict for the masking token which is located in the middle of the sentence and thus we use bidirectional context for the prediction. We also do this for n steps, where at each step we add the masking token to the end of the left context and then substitute the predicted token into that position. The quality of the output is somewhat similar to the previous approach but sometimes predicts words that somewhat fit into the context.

The third and last strategy is to repeat the above experiments, however, with the difference of adding all the masking tokens to the starting sentence at once and then predicting for all the respective positions. Such an approach combined with the second strategy simulates a very similar objective to the one at training. This approach for sentence generation has shown to fall into the same problems as the previous techniques, however, outperforms the previous approaches in terms of the frequency of more suitable output. An interesting observation is that the pre-defined length of the expected number of generations influences the quality of the output. When the expected number of generations fits into the given context, the model can generate a coherent continuation. However, there are two main problems. There is no way knowing and thus pre-defining a number of fit generations, as well as, it is always required to have context on both sides of the prediction word.

Very recently, there have been paper publications tackling the words sequence generation problem using the bidirectional transformer. Stern et al. (2019) proposed an approach which accommodates arbitrary orderings by allowing for tokens to be inserted anywhere in the sequence during decoding. Furthermore, Welleck et al. (2019) has proposed a method which generates a word at an arbitrary position and then recursively generates words to its left and right, while yielding a binary tree. The final output is generated thanks to the imitation learning strategy which learns sequential decision-making policies on the binary tree. Lastly, Gu, Liu and Cho (2019) proposed an extension to the Transformer architecture which models the generation order as a latent variable based on the input information, i.e. at every step, the model sequentially predicts a word as well as its relative position in the input.

All of the papers focus on the same idea - allowing an arbitrary generation order via some sort of insertion mechanism - which spans an important research direction. Most of the recent research in NLP tackling the methodology of processing the inputs have shown to produce high-quality models for most of the possible tasks. However, as this experiment has shown, having a high-quality model is not enough when it is still unclear how to get the most out of it for tasks requiring generation. Further research into generative strategies that are perhaps inspired by the way humans write text could allow for novel interactive applications. Furthermore, considering the poor output performance obtained from BERT points us to using a GPT2, as the fundamental generative model, in further experiments.

5 Conditional Lyrics Generation using GPT2

The focus of this Section is to discuss and analyse technical training details, experimental strategies and results of a successful song lyrics generator **(SLG)** model. The fine-tuned model generates coherent song lyrics of a particular style which is provided as a condition. The conditional generation capabilities are purely achieved with an appropriate fine-tuning strategy, without any architectural changes to the pre-trained model.

In the previous Section, it has been stated that BERT struggles to generate a coherent text due to the lack of a suitable sampling technique that would work well with the Masked-LM objective used during pre-training. As a consequence, using BERT for generative tasks would not provide satisfying results. Instead, a BERT's competitive counterpart, a GPT2 model will be used.

Both language models are pre-trained transformer networks that utilise self-attention to compute contextual representations, which makes them almost identical at the architectural level. Whereas the most significant distinction is the training objective function, which sets the model learning progress as well as, after the training, defines a strategy to generate outputs from it. BERT uses a Masked-LM objective that utilises bidirectional context representations for learning. Whereas the GPT2 uses a unidirectional left-to-right objective function that, in contrast to prior, trades a practical and straightforward approach to sequence generation for slightly lower performance on the downstream tasks. Since the bidirectional representations conditioning raises the model performance, and the left-to-right objective function provides a simple and effective output generation strategy, one may ask: why not to use bidirectional conditioning with a standard learning function for language model modelling. Unfortunately, such an approach is not logically compatible because bidirectional conditioning would allow each word to indirectly "see itself", and the model could trivially predict the target word in a multi-layered context. Moreover, always predicting the next element to the right given the representation of the previous sequence would not reveal full bidirectional potential as much as predicting a word from the middle.

GPT2 pre-trained model is available online as a library package called "transformers" created by Hugging Face[10]. Similarly to BERT, the GPT2 model comes with two sets of pre-trained weights, small and medium. The small weights originate from a model trained with 12 layers, 768 hidden states and 12 multi-attention heads per layer, thus, the maximum of 117M computations per input. Whereas the medium weights come from a model pre-trained with 24 layers, 1024 hidden states, 16 multi-attention heads and thus 345M parameters.

5.1 Methods

This Section details an approach and methods essential to compose a conditional lyrics generation model. The topics include the training dataset details, input feeding strategy, output sampling methods, and the output evaluation metrics. The details discussed in this Section directly relate to the experiments and analysis covered in Section 5.2.

5.1.1 Datasets

The original data[11] used for SLG model training consists of 380,000 lyrics examples, where each example has four additional related features, including a genre. As it is the case with most raw datasets, the lyrics dataset also requires some cleaning. The preparation is an important part of the training process since the quality of the data directly reflects the performance. After the dataset inspection, it can be noted that some input features are blank or contain lyrics in other languages than English; thus, we filter those out. Furthermore, the dataset contains twelve different genres, where each genre has an uneven number of examples in respect to each other. Therefore, we filter out all lyrics but the ones belonging to *metal*, *pop* and *country* because

[10] https://github.com/huggingface/transformers

[11] https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics

only these classes have enough examples to make the new datasets proportional in respect to the classes. Furthermore, we also make sure that each song is in the range of 500-2000 characters, which approximately transitions to 125-500 words.

From the input examples that have passed the above conditions, we create four datasets where each has a different number of inputs. Table 6 summarises the dataset details.

Dataset Sizes		
Dataset Name:	Training Size:	Evaluation Size:
Large Lyrics (LL)	30000	6000
Medium Lyrics (ML)	15000	3000
Small Lyrics (SL)	6000	1200
XSmall Lyrics (XSL)	600	120

Table 6: Displays different dataset sizes used for experiments in Section 5.2. The evaluation datasets do not intersect with the training samples.

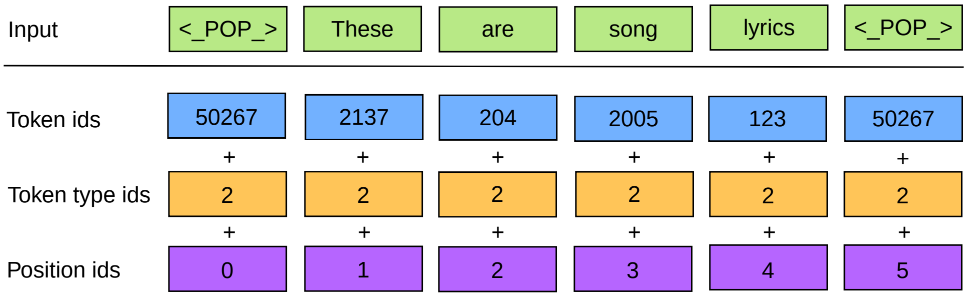
5.1.2 Training: Input Construction Strategy

To construct input for the model, we draw lyrics from the dataset and wrap them with a special token. For example:

<_SPECIAL_> ... lyrics ... <_SPECIAL_>.

A special token is any character composition that is unique to the pre-trained model vocabulary. Technically, the model treats special tokens just like any other words; appends to the vocabulary and represent with a unique reference number. Therefore, it is a programmer choice to decide on a unique special token name that will reflect its purpose. By defining the unique tokens at the beginning of the fine-training, the pre-trained model has no prior knowledge of what they stand for; therefore, the model begins modelling of what the tokens should correlate with. Since the model correlates inputs based on their words occurrence patterns, by placing the special tokens in an appropriate sequence position, we can manipulate the learning space of a model. As a result, after the fine-tuning process, it is possible to condition the model on the special tokens to output lyrics in a particular style. For the experiments in Section 5.2, we define three special tokens, <_METAL_>, <_POP_> and <_COUNTRY_> to represent every genre within the datasets described in Section 5.1.1.

Furthermore, in order to help the model disambiguate between features we want it to learn, in our case genres, we element-wise add additional vectors to the input (Token IDs), i.e., Token Type IDs and Position IDs. The additional vectors can contain any values that mean to help a specific objective generalise. The goal of this conditional generator is to learn the different lyrics styles (mental, country, pop), thus utilise the Token Type IDs to separate the different genres in the following manner. That is, for each different lyrics genre, we fill the Token Type IDs with a different value that is the same for all of the input positions. For example, if the training input is of metal type, we fill the Token Types IDs with ones, and if the lyrics are of a pop genre, we fill with twos, etc. Note that the addition of the Token Type IDs is not compulsory; it is a strategy that we have chosen to experiment with. Moreover, we use the Position IDs to help the model encode the special distance between words of the input. We do it by filling a vector with incremental integer sequence, i.e., [1, 2, 3, ..., input length], which we then element-wise add to the input (Token IDs). By using the positional encoding, the model learns the input length patterns, and thus, learns to predict the unique end token that is used as an indication to cut off further sequence prediction. Figure 21 illustrates an exemplary training input using the above input constriction strategy.



Input	<_POP_>	These	are	song	lyrics	<_POP_>
Token ids	50267	2137	204	2005	123	50267
	+	+	+	+	+	+
Token type ids	2	2	2	2	2	2
	+	+	+	+	+	+
Position ids	0	1	2	3	4	5

Figure 21: Exemplary model training input used for experiments in Section 5.2. The input is wrapped with special tokens that, after the training, are used as a condition that encodes a particular genre style. The Token IDs correspond to input words transformed by a static encoding function (Byte Pair Encoding by Sennrich, Haddow and Birch (2016)). The Token Type IDs are represented with a pre-chosen value that helps the model learn the particular genre type. The Position IDs partition holds pre-chosen spatial distance encoding values that are the same for all genre classes. All of the input partitions are element-wise added before being fed into the model.

Moreover, during the training process, it is also required to provide language modelling labels which are essentially input Token IDs with every element being shifted one position to the right. Lastly, we pad the input partitions with zeroes until the length of the longest input in the dataset, in order for all the inputs to be of the same size in the input batch.

5.1.3 Generation: Output Sampling Strategies

The recent idea of fine-tuning a pre-trained transformer network has resulted in many benchmark advances in natural language understanding and generating tasks. Using a fine-tuned model for the tasks requiring understanding is straightforward since it usually involves processing an input through a model that assigns it into a specific class, which is the required output. However, sampling from a model is not as simple as the classification since it requires an appropriate output sampling technique. The decoding strategies alone drastically affect the quality of the outputs, even when sampled from the same distribution as model weights. Differently put, no matter how excellent the fine-tuned model is, without an appropriate sampling strategy, the output qualities will be poor.

Until recently, the two most common decoding strategies were *greedy-decoding* and *beam-search*. The greedy-decoding is the most basic method which selects the most probable (argmax) next token and repeats the procedure until reaching the end-of-sequence special token. However, always selecting the token with the highest probability distribution introduces a risk of a high-probable word being hidden behind a low-probability token. As mitigation to this problem, at each decoding step, the beam-search maintains a beam of several possible sequences and at the end selects a sentence that is globally maximised. Over the last few years, beam-search has become a standard decoding strategy for almost all generative tasks, which includes both open-ended[12] and non-open-ended[13] tasks. In practice, while it still makes sense to use beam-search for the non-open-ended tasks due to their often tightly scoped probability distribution space, it makes less sense for open-ended tasks since they require more spatial freedom in terms of word choice, where this maximisation-based algorithm do not perform well (Holtzman et al., 2020). As a result, the current influential long sequence generative models have chosen an alternative of a *top-k sampling* to serve as the model sampling strategy (Radford et al., 2019), (Zellers et al., 2019). Since this work has shown to produce state-of-the-art quality outputs, it sets

[12] The open-ended tasks are generally identified by a conditional long sequence generation. An example application includes article generation and text continuation. In such tasks, the output sampling strategy should restrict the range of acceptable output words while leaving a considerable level of freedom for the sentences to be diverse.

[13] The non-open-ended tasks can be mostly defined by an input-output pair, such that the generated output is a close transformation of the input. An example application includes machine translation and text summarisation. At such tasks, the output sampling technique should be tightly scoped by the input condition so that there is not much freedom in the output word selection.

a promising ground for the top-k method to be used in this project. Broadly speaking, the top-k sampling method selects the next token by first, taking top-k most probable words, and then re-scaling their relative probability distributions. More formally, the whole process looks as follows:

Given a sequence of m tokens, $x_1 ... x_m$, as a conditional context, the task is to generate the next n continuation tokens to obtain a complete sequence, $x_1 ... x_{m+n}$. Under the assumption that a standard left-to-right language model for text decomposition is used, we generate token by the token in the following manner:

$$P(x_m : x_{m+n}) = \prod_{i=m+1}^{m+n} P(x_i | x_1 : x_{i-1}) \tag{18}$$

Now, this is where the top-k comes into play. At each decomposition step, given the probability distribution $P(x_i | x_{1:i-1})$ over the model's vocabulary V, we select a set of k highest activations $V^{(k)} \in V$, where k is a pre-chosen hyperparameter. Then, by indexing through the $V^{(k)}$ we take the sum to obtain p':

$$p' = \sum_{x_i \in V^{(k)}} P(x_i | x_{1:i-1}) \tag{19}$$

Then, using the p', we re-scale the original distribution to a new distribution accordingly:

$$P'\left(x_i | x_{1:i-1}\right) = \begin{cases} P\left(x_i | x_{1:i-1}\right) / p' & \text{if } x_i \in V^{(k)} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Using softmax function over the new probability distribution, we obtain the next word.

Furthermore, the recent work by Holtzman et al. (2020) has proven that the distribution of words in texts generated by the beam-search and greedy-decoding differs considerably from the distribution of words in human-written texts. They argue that people optimise text against stating the obvious, making highly predictable text unlikely to occur in practice, therefore, the decoding strategies based on the maximum probability lead to text with unnatural high probability and low variance, which results in an unnaturally looking outputs. This motivation has led to a product of a new *nucleus top-p sampling* strategy, which is based on the idea of randomisation rather than maximisation. The top-p works by sampling from the previous word distribution after having to filter it only to keep the tokens with a *cumulative probability distribution* above the p threshold. Generation using the top-p have shown to produce satisfying quality and diverse outputs which is the reason why it will be used in further experiments.

The top-p will be explained from Equation 18 onwards. Similarly to top-k, the top-p determines a set of tokens to be sampled from. Given a probability distribution $P(x_i|x_{1:i-1})$ over the language model's vocabulary, we select the highest probability tokens whose *cumulative probability mass* exceeds the pre-chosen threshold p:

$$p^{'} = \sum_{x_i \in V^{(p)}} P\left(x_i|x_{1:i-1}\right) >= p \tag{21}$$

In practice, this means that after obtaining the distribution we sort the logits in descending order, apply a softmax function and then on top of that a cumulative sum. Like-wise in Equation 20, the next step is to re-scale the distribution to a new one in order to filer all the tokens whose mass exceeds the threshold p:

$$P'\left(x_i|x_{1:i-1}\right) = \begin{cases} P\left(x_i|x_{1:i-1}\right)/p' & \text{if } x_i \in V^{(p)} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

To compare the two aforementioned methods, both sample from truncated LM distribution, differing on the strategy of where to truncate. The top-k samples from the next token distribution after having to filter it out only to top k tokens, whereas, the top-p, samples from the distribution whose cumulative probability surpassed the p threshold. This makes the top-k strategy constant in terms of the number of words to consider, whereas top-p is dynamic since it picks the number of words whose mass passes a threshold.

Note that the choice of truncation point is an important feature since it defines the model's generative confidence region.

5.1.4 Output Evaluation

Evaluative metrics on generative models' outputs is an active area of research and yet there is no unique metric that would fit all kinds of outputs for a fair comparison across different generative capabilities. A reasonable evaluation metric for generative models are human annotators since we best know what kind of output looks real and within the expectations. However, human judgement can be tricky since most generative outputs are a subject to an interpretation and thus would be difficult to judge fairly, as well as the process is very time-consuming and not reusable.

In this project, we automatically evaluate the models' performance based upon measuring the *quality*, *diversity* and *condition genre reflection* in the generated outputs. The metrics are discussed later in more detail; however, the quality of a model is measured using a *Bilingual Evaluation Understudy* **(BLEU)** score first proposed by Papineni et al. (2002). Whereas for the diversity reflection we use two metrics, a *Self-BLEU* **(SBLEU)** introduced by Zhu et al. (2018) and a *Unique-Ngram Counts* **(UNGC)**, as described in the work of Wang and Cho (2019). Lastly, we train a *genre discriminator* to see whether the condition is reflected in the output of the model.

Quality Metric: The BLEU scoring metric proposed by Papineni et al. (2002) was first designed and introduced in the thought of machine translation **(MT)** evaluation; however, it can also be applied to other tasks' evaluation. As a consequence, the BLEU scoring algorithm will be first explained from the MT point of view in order to understand the trade-offs when used for generated lyrics evaluation. In MT, the closer the translated text (candidate) to a professional human translation (reference), the better the quality.

The first step to derive a BLEU score is to compute an *ngram precision.* This is done by counting the number of matching n-grams between the candidate and the reference and dividing by the total number of n-grams in the candidate. Unfortunately, computing the precision this way introduces a problem at a case where the model over-generates "reasonable" words, resulting in low adequacy but high-precision score, e.g., consider the illustration below where the model generated a matching word multiple times (example inspired on Papineni et al. (2002)).

Candidate: the, the, the, the, the, the, the
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat

Uni-gram Precision: Ref. 1 scores 7/7, Ref. 2 scores 7/7
Modified Uni-gram Precision: Ref. 1 scores 2/7, Ref. 2 scores 1/7

The problem is solved with a *Modified Ngram Precision* **(MNP)** which considers a reference word exhausted when a matching candidate has been found. In order to compute an MNP, one must first count the maximum number of times a specific word occurs in the reference translation (clipped count), and in the case when the number of overlaps for that particular word in the candidate exceeds the clip count, the clip count is used instead for the precision calculation. Mathematically it's computed like: $count\_clip = min(count, max\_ref\_count)$. Note that each candidate can have multiple corresponding target sentences to which, throughout the explanation, it has been referred to as a reference. Furthermore, the modified n-gram precision is a unit evaluation of candidate-to-reference pair; however, what we want is a scalar that describes the MNP over a dataset of candidates. In order to do the prior, the algorithm adds the clipped n-gram counts for all of the candidate sentences and divides by the total number of n-grams in the generated corpus, just like in equation 23 below.

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in \mathcal{C}} Count_{clip}(n - gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram' \in \mathcal{C}'} Count\,(n - gram')} \tag{23}$$

Furthermore, BLEU computes a geometric average of the modified n-gram previsions, p_n, using n-grams up to length N and positive weights w_n which sum up to 1. The geometric mean has been chosen since the greater in length grams, the less probable overlaps; thus the different scores need to be weighted accordingly.

Moreover, from the ML point of view, a perfect translation should neither be too long nor too short and the BLEU strategy detailed until this point is only enforcing it partially. The n-gram precision penalises the spurious words in the candidate that do not appear in any of the reference translations. In addition, the MNP penalises given the word occurs more frequently than in the maximum reference count. However yet, the MNP does not have any explicit mechanism to punish translations much shorter in length, as illustrated in the extreme example below (modified example from Papineni et al. (2002)).

Candidate: of the
Reference 1: It is a guide to action that ensures that the military will forever heed Party commands
Reference 2: It is the guiding principle which guarantees the military forces always being under the command of the Party

Modified Uni-gram Precision: Ref. 1 scores 1/2, Ref. 2 scores 2/2
Modified Bi-gram Precision: Ref. 1 scores 0/1, Ref. 2 scores 1/1

Nevertheless, to resolve such cases as the one illustrated above, the authors came up with a sentence *Brevity Penalty* **(BP)**. With this mechanism, high-scoring candidates must match the reference translations in length, word choice and word order. It works by assigning a sentence BP of 1 in a case when the candidate's length is the same as one of its references. E.g., consider having three reference translations whose lengths are 10, 13 and 17, and a candidate whose length is 13. In such case, the sentence BP would be

1 since the algorithm looks for a "best match length" in all of its references. In order for the penalty not to be too harsh on shorter sentences, the authors decided to compute it at the corpus level. This is done by summing the best match lengths for each candidate sentence in the corpus, r, and dividing by the length of the total words of the target corpus c, where the penalty is a decaying exponential of r/c. Note that the MNP already penalises the translations longer in length thanks to the count clip.

To finalise, BLEU takes a geometric mean of the modified precision scores for all of the n-grams and multiplies the result by an exponential brevity penalty factor. The equation is a follows (Papineni et al., 2002):

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{24}$$

Then,

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{25}$$

Which is more immediately apparent in the log domain,

$$\log \mathrm{BLEU} = \min\left(1 - \frac{r}{c}, 0\right) + \sum_{n=1}^{N} w_n \log p_n \tag{26}$$

Where N=4 and uniform weights $w_n = 1/N$.

In terms of the final BLEU result from MT point of view, a perfect candidate would score 1 if its n-grams would be exactly the same as the n-grams in the reference translation.

By following the intuition of Yu et al. (2017), we compute BLEU between the generated lyrics and the training dataset in order to measure the quality of the generations in contrast to the original data. Note that for the machine translation task perfect quality score is 1; however it is not the case for lyrics generation. This is because lyrics generation task is a fundamentally different and perfect score of 1 would mean that the model is massively overfitted and has mostly copied the true data distribution. However, using

BLEU to evaluate the lyrics is still a true quality reflection that can be used to contrast the different model training settings. Note, using BLEU for lyrics evaluation, the BP should not affect the final score so much since for each generated song we treat all of the training dataset inputs as a reference; thus, it is very probable to find a matching length spawning BP=1. Such behaviour is useful because we expect the generated songs to be of different lengths, and thus we do not want the metric to penalise unless it is an extreme case. Moreover, the metric does not consider synonyms in the n-grams, which is preferable for the machine translation scenario, however, not necessarily essential for lyrics comparison since it would introduce a randomisation factor which would raise the scores.

Furthermore, in the experimental Section 5.2, in order to apply BLEU in a fair environment, for each evaluation model we generate a lyrics collection of size equal to 10% of the current model's training dataset size. We sample from the models by sequentially setting seed from 0 to 100 and generating an equal amount of songs for each genre. Also, the batch size is dynamically adjusted to generate exactly 10% of a particular training dataset size.

Diversity Measurement:

In order to get a diversity measurement in the form of a scalar, we compute Self-BLEU (Zhu et al., 2018), which we will refer to as SBLEU through the thesis. SBLEU, as the name suggests, is a variation of BLEU that evaluates the diversity of the generated data in contrast to itself. Since the underlying mechanism of BLEU compares how similar two sentences are, it can also be used to calculate how one sentence resembles the rest in a generated collection. Particularly, this is done by regarding one generated sentence as a candidate and all the others as a reference, which produces a score for every candidate that is later averaged to create a SBLEU score of the generated dataset. A higher final score implies a smaller diversity.

Moreover, in order to get a more fine-grained diversity metric, we perform a Unique N-gram Count (UNGC) for N that ranges in length from 1 to

4. The UNGC compares the percentage of n-grams that are unique in the whole collection. This is done by first, identifying all n-grams and counting their appearance frequency. Then, counting all the n-grams that have an appearance frequency of one (unique) and dividing by the sum of all n-gram counts. This way the resulting value reflects the proportion of unique n-grams in relation to all the n-grams and their count frequency. Furthermore, the UNGC can be applied to both, the original training dataset (likewise BLEU), and to the corpus of generations (likewise SBLEU). The higher the resulting values for each n-gram, the more diversity. Thus, this metric is somewhat opposite to the BLEU as fewer unique n-grams imply higher BLEU. Note that this metric will never be too close to 1 since the final division denominator includes counts of words whose natural frequency is greater than one, e.g. connective words.

In other words: Given a collection $\{C\}$ we identify n-grams g and their frequency c, $\sum_4^N \sum_{\{C\}}^1 a_n = (g_n, c_n)$ and then we get the unique n-grams by $a_n^{unique} \subset a_n$ where $c_n == 1$. Then, we divide the unique n-grams by the total count $\sum_4^N \frac{unique_n}{\sum_{c_n}^1}$ in order to get a set of probabilities for the collection.

Condition Discriminator:

To obtain the discriminator, we define and train a one layered fully connected network with bias, that takes the GPT2's output in the form of the hidden states and classifies it to a genre class. Having to fine-tune multiple GPT2 models in the different hyperparameter and dataset settings, we can expect each GPT2 to produce a different set of hidden states for the same input text. Therefore it is only fair to train a separate discriminator for each different fine-tuned model because each discriminator learns to classify correctly based on specific model's hidden states. In other words, the discriminator's output is depended on the GPT2's hidden states. We tried using one discriminator to judge the outputs of multiple GPT2 models; however, the performance was poor. In addition, such a strategy would not provide a stable ground for a fair evaluation; one GPT2 hidden state outputs could be more similar to the ones used for discriminator training than others.

Furthermore, in order not to bias the discriminator towards the correct genre class, during the training process, it is crucial to remove the special tokens from the GPT2's input since they will be reflected in the contextual hidden states representation used by the discriminator. This creates a conflict when trying to fine-tune GPT2 model and train the discriminator together since we require one input to be processed twice. As a consequence of double training, we chose a strategy to first fine-tune a GPT2 model on the whole dataset and afterwards train the discriminator using the same dataset. This approach has an advantage of providing a fixed and quality representations for the discriminator that do not reflect the special tokens; however, it prevents from using a joint error to fine-tuning the GPT2 since the model is already trained.

Note, the discriminator is an only one layered networks, which potentially may be too small to classify the GPT2 very dynamically changing representations. However, this may not necessarily be the case since the GPT2 hidden states already encode the input patterns. Therefore, we chose this design implication because of two reasons: lack of computational power, and the curiosity to see what the only one layer can accomplish.

5.2 Experiments & Results

This Section displays results and analysis on three essential aspects of a conditional generative model training, i.e., the dataset size, input construction strategy, and model sampling techniques. Note that the experiments in this Section utilise the methods described in Section 5.1; therefore, in the analysis, it is assumed that the reader is familiar with the covered methodological arrangements.

5.2.1 Fine-tuning on a Variety of Dataset Sizes

In this experiment, we are going to compare and analyse the performance of a GPT2 model fine-tuned on four different dataset sizes in order to find an appropriate balance between the training time and satisfying output quality. Finding such a trade-off is an important aspect of model training since often one has limited computational resources and dataset size. The model's performance is examined by first, generating lyrics collections from weights trained using corresponding datasets, and then, by using BLEU[14], SBLEU and Self-UNGC metrics to measure the quality, diversity and uniqueness of the outputs. The generated collection's size measures 10% of the number of songs in the particular training dataset.

Evaluation of Generated Collections			
Dataset len	BLEU^{Quality}	SBLEU^{Diversity}	Difference
30000 (LL)	42.5%	23.3	19.2%
15000 (ML)	39.5%	20.6	18.6%
6000 (SL)	35.4%	18.4	17.0%
600 (XSL)	20.2%	17.7	2.5%

Table 7: Performance of the GPT2 model fine-tuned using four different dataset sizes with the output sampling setting of (top-p, top-k = 0). The quality and diversity scores are based upon generated collections that make up 10% of the respective training dataset size. The 'difference' column displays a resulting value of the following computation: BLEU score − SBLEU score.

To recall the BLEU score meaning, the higher the percentage, the more similar the generated corpus to the training dataset. A vital characteristic of any neural network is for the model to generalise on the data patterns and not copy the dataset (overfit). Intuitively, a BLEU score of a 100% would mean that the network has overfitted the data, which is not what we want. Therefore, the question remains - what is an unknown optimal BLEU value of an open-ended generative task, such as, lyrics generation,

[14] To produce the BLEU score we compare a generated collection to a corresponding model's **training** dataset - not the evaluation dataset.

that would best reflect the aesthetic qualities while not entirely copying the dataset. Differently put, what is an optimal BLEU value that displays a good balance between underfitting and overfitting - good generalisation.

In respect to the BLEU (quality) score alone, Table 7 shows that a dataset size of 600 is not big enough for the model to generalise properly since the score of 20.2% indicates strong data underfitting. However, fine-tuning the model on only 6000 inputs displays almost double quality in contrast to the prior, signifying that this size may be reasonable enough to achieve aesthetically pleasing outputs. Furthermore, increasing the dataset size from 6000 to 15000 gains 4.1%, and from 15000 to 30000 brings only a 3% increase in quality. Revealing that dataset size expansions improve the quality aspect, however, the relative improvement decays with the increase in data size. Meaning that if a large dataset composition is extravagant, one may consider creating a smaller set, e.g., of size 6000, while expecting a slightly smaller performance.

Furthermore, to recall the SBLEU (diversity) meaning, the higher the percentage, the less diverse the generated collection because there are more matching n-grams in relation to each other. When inspecting the SBLEU scores, one obvious correlation is that the model becomes less diverse given a larger dataset size, where the relative diversity loss is constant with the size[15]. Given the diversity scores of the training datasets being at around 30-50%, the correlation of outputs becoming less diverse given more data indicates that the model continues 'fitting' to the training data distribution. Ideally, we do not want the model to copy the data distribution and thus for the diversity score to grow, but we want the model to generalise from the data in such a way that would make the outputs as diverse as possible, i.e., settling somewhere below the training dataset diversity score. However, one may say a high diversity, especially early in training, may not translate to the expected output aesthetics, i.e., the model can generate more out-of-context

[15] At first, the Table 7 may not indicate a constant relative loss in diversity. However, note the irregular increases to the dataset size.

and incoherent sentences which can falsely lower the score. Nevertheless, when manually contrasting the generated lyrics from the weights trained using the dataset size 6000 and 30000 (see Table 8), we can state that the low diversity score truly reflects the good qualities of output, i.e., both lyrics display to be similarly coherent. Therefore, a low SBLEU score should be considered as a striven property of the outputs.

Given that output should have the highest quality and the lowest diversity score, combining the BLEU and SBLEU metrics allows finding a compromise between the training time and the performance. This can be done by fine-tuning a model using a variety of dataset sizes that marginally differ in size in relation to each other. Furthermore, by inspecting the quality and diversity scores of generated collections from the respective weights, it is possible to identify the dataset that produces the biggest proportion between the gain in the quality and the decrease in the diversity, in contrast to the previous dataset.

By analysing Table 7 using the prior hypothesis it is visible that, the more training data, the bigger the 'difference' gets. In other words, the more data we have, the more absolute performance gained. It is notable that after the dataset size of 6000, the relative performance gains decrease. Therefore, given one's limited computational and dataset resources, fine-tuning a GPT2 model using a dataset size of 6000 would produce outputs of reasonable qualities, in terms of the diversity and dataset resemblance. Note, that this experiment is dataset-specific, i.e., we are using a dataset that is composed of lyrics belonging to three different genre classes. Increasing the number of genre classes, or the number of features describing lyrics could result in entirely different scores.

Generated Song Lyrics - Feature-less Approach	
Dataset Size: 6000	Dataset Size: 30000
Condition: Metal	Condition: Metal
You started a journey from the East Nothing have I seen but the ocean The seas will open up for me Can you feel how I feel against your soul My coldest dreams come true I know you took the smile away Blood dried tears and life had left me There have no sadness of the sea i have cried You came right with your smile I stood side by side, open face and face As silent as could be Inside your heart there's nothing to bear I hear your voice smiling in the dark I know you took the smile away Slaves shattered into these sheets Despair did you feel so beautiful That our love would last No tears to tears to be seen again When our dreams will last When our dreams will last	Here's a place in the heart of heroville Where you used to be Walkin' in the parkin' lot It's a crowded place Where I remember the old folks Once ma'am afraid to use that old time playin' song It's a crowded place Where there were those People who were blinded with their eyes Brought me pain and minds who cried And just like they were blind Like the movie star This lonesome entrance slammin' place Wouldn't have the chance to talk it over Thereen retreatin' place That old mistakes were the new Yapp awakened early The Exit Hotel by that old city A girl beneath that old school

Table 8: Generated samples from a GPT2 model trained using feature-less datasets. On the left, song lyrics produced from a model fine-tuned using a dataset of size 6000. On the right, a sample from a model fine-tuned on 30000 inputs. Despite the training size difference, both outputs display aesthetic lyrics qualities. The lyrics were randomly picked without any specific score indication.

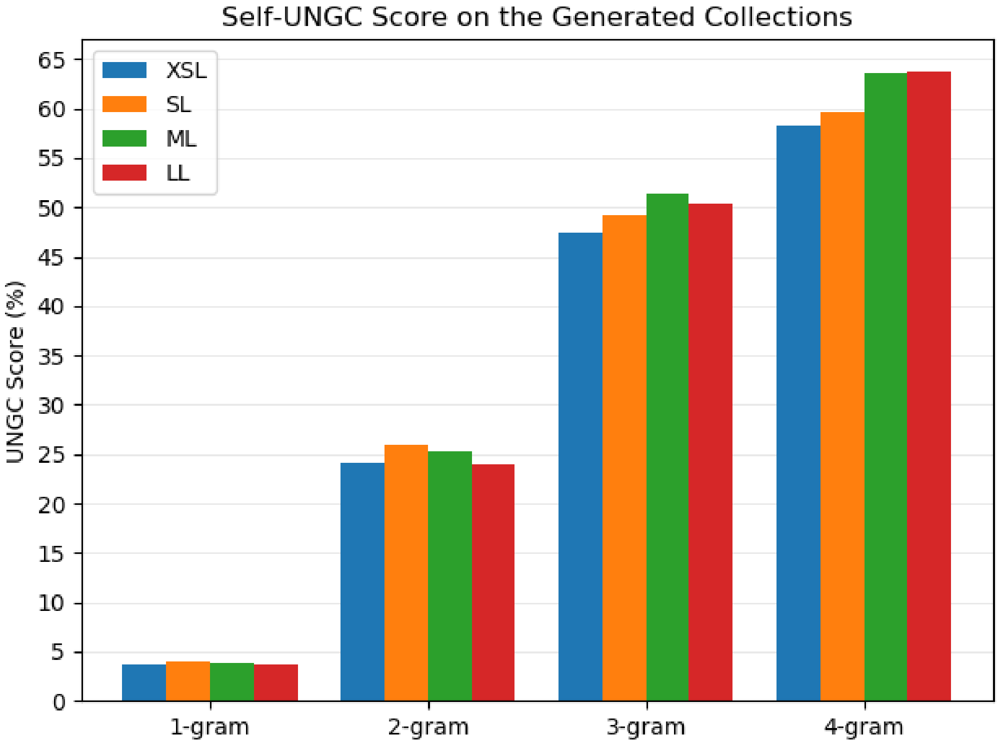


Figure 22: Self Unique Ngram Count scores of generated collections sampled from four different weight distributions that were trained using four datasets varied in size. The scores are produced based upon generated lyrics collections that are of size: 10% of the training dataset.

Now we are going to inspect the output diversity from a different perspective, i.e., using the Self-UNGC metric. A Self-UNGC score represents the percentage of n-grams that are unique in a given collection. A higher percentage means more unique n-grams. Figure 22 illustrates the results.

As it can be observed, the 1-grams and 2-grams for the different collections do not significantly differ, indicating that the generated collections do not vary at the range of word predictions. However, when inspecting 4-grams, it is notable that the larger dataset displays a higher uniqueness at four words sequence composition. The analysis of this metric with respect to the SBLEU ravels that given more data, the outputs generally start to become less diverse; however, short sequences (4-grams) become more unique. While considering the complexity of natural language and its properties uncaptured by the metrics, such correlations may indicate numerous things. For example, given more data inputs, the outputs become less random and out-of-context (less diverse), and more unpredictable and human-like (more uniqueness). This behaviour can be justified with the model generalising on the additional subset of inputs included in the larger dataset, since the extra inputs contain words that have been seen before (lowers diversity), as well as introduce previously unseen words (enlarges the uniqueness). Therefore, given more data, the model more frequently predicts common sentences, however, more often occasionally produces a sequence which is unique. Alternatively, the lower diversity (higher score) could signify that the model collapses its prediction to a smaller subset of words, and thus starts to overfit.

Discriminator Scores		
Dataset size	Genre Classification Acc.	Training Evaluation Acc.
30000 (LL)	46.9%	84.3%
15000 (ML)	50.1%	96.4%
6000 (SL)	46.3%	93.2%
600 (XSL)	37.0%	82.2%

Table 9: On the left, discriminator's genre classification accuracy based upon generated collections of size 10% of the training dataset size. The lyrics collections were generated from GPT2 models trained using the same amount of inputs as the respective discriminators. On the right, the discriminator training evaluation accuracy. All of the discriminators were evaluated using the same evaluation dataset of size 6000.

Now, we will be looking at the discriminator. The discriminator's goal is to classify the generated lyrics into a genre class and check whether it is, in fact, the GPT2's condition. It is important to comprehend that the discriminator's classification output is not entirely depended on the adequacy of the lyrics, but also on the training evaluation error rate. Since in the chosen discriminator training strategy all of the GPT2 models have separate discriminators trained from scratch, we obtain different evaluation scores for different models[16].

When looking at the evaluation and classification test accuracy scores of the training using only 600 examples, we can first suspect that the discriminator has been overfitted; indicative by the high evaluation score of 82.2% and low genre classification accuracy of 37% (bad generalisation). However, when considering the complexity of the experiment, i.e., the GPT2 model, on whose hidden state outputs the discriminator was trained on, was also trained on 600 inputs which as the prior results have shown (see Table 7) did not transition to high-quality outputs. Therefore, the evaluation accuracy most probably indicates that the discriminator is underfitted, and given its not the most optimal state and the badly generated outputs, we get the test accuracy of only 37%. Furthermore, a much higher number of examples - 6000 and 15000 - has brought more satisfying results, however, with an expense of many data examples. However, given the greater amount of inputs, the GPT2 model has learnt to produce better quality outputs (see Table 7) which did not transition into significantly better test[17] accuracy. Therefore, these results indicate that the discriminator is overfitted since it cannot, with a high precision, classify generated lyrics that only slightly differ from the ones in the training dataset. Lastly, the score for LL dataset shows that there were too many different examples for the network to grasp since the two prior results (6K and 15K of inputs) have achieved better accuracy. Therefore, give the twice as large number of examples, the discriminator is

[16] Recall that one discriminator would not be able to fairly evaluate models trained on varying in size datasets because of the randomisation introduced by the extra inputs. Therefore, after training, despite given the same input, the different GPT2 models would produce different hidden state output that the discriminator would classify. As a result, the model with hidden states that would be more-alike the ones the discriminator was trained on, would unfairly score the highest.

[17] Test in this context corresponds to genre classification accuracy.

underfitted.

Generally, when observing the generated lyrics classification scores, it becomes evident that the discriminator has struggled with the classification since most of the accuracy scores are at around 50%, where a random guess would result in the accuracy of 33%. Therefore, the discriminator is not complex enough to capture such dynamic GPT2 representations. In order to improve the reliability of the discriminator, it is necessary to enlarge the structure in terms of the number of layers with a consideration of the training dataset size; the more different training examples we have, the more complex the discriminator should be.

Motivated by the idea of limited dataset and resources for model fine-tuning, in this experiment, we contrasted the performance gains in respect to the increasing data size. In more details, we trained a GPT2 model using four varied in size datasets: 600, 6000, 15000 and 30000. The datasets were proportionally composed of song lyrics belonging to three different genre classes (metal, country and pop). After training, from each obtained weight distributions, we generated lyrics collections of length equal to 10% of the respective training dataset size. Then, we evaluated the performance of these lyrics collections using four measures: BLEU (quality), SBLEU (diversity), Self-UNGC (uniqueness) and the discriminator (genre condition resemblance).

A general observation is that the quality of the outputs positively correlated with the increase in data size; however, the relative improvements gain decayed with the increase in dataset size. Therefore, given one has limited resources and the dataset, the results have shown that it is reasonable to fine-tune a GPT2 model only using 6000 input examples to achieve a reasonably satisfying output quality; the dataset extensions - from 6000 to 15000, and from 15000 to 30000 - only displayed marginal performance gains.

Furthermore, when analysing the SBLEU scores, the outputs display to become less diverse, given more inputs. This property positively correlated with the quality scores, which indicated that the model was learning, and thus, becoming more 'fit' to the dataset distribution. Since the highest quality and the lowest diversity scores (more diversity) are the striven output properties, in this experiment, we show an approach of finding a right balance between the training resources and the output performance qualities, i.e. by identifying a dataset that produces the biggest proportion between the gain in the quality and the decrease in the diversity, in contrast to the previous dataset.

Moreover, the Self-UNGC for 4-grams has shown that the more inputs we trained a model on, the more unique the output sequences become. Therefore, using this specific training approach and the dataset, when increasing the data amount, the outputs improve at the quality and uniqueness, but become less diverse. Given the complexity of natural language, we hypothesise this means that the model predictions become more dataset-alike, occasionally predict the previously unseen tokens learned from the new data, and generally the prediction collapses to the most popular words whose frequency rises given the new data. Lastly, the genre discriminator scores have shown to be just above the guess-level, indicating that the discriminator itself can be unreliable and may not truly measure the genre reflection in the outputs. Displaying that one layer on top of the transformer for simple lyrics classifying task is not big enough, and expansion in the number of layers is required in order to achieve more reliable genre evaluation metric.

5.2.2 Effect of Input Partitions on Model Performance

This Section investigates the influence of input partitions on the model's performance by fine-tuning a model with and without specific partitions, i.e., the input Token IDs **(TIDs)** with and without the Token Type IDs **(TTIDs)** and the Position IDs **(PIDs)**. Recall the input partitions are additional vectors that get added element-wise added to the input in order to manipulate and enhance the learning of a model (see Section 5.1.2). By analysing the additional partitions individually, we reveal the relative performance gains it brings to the optimal model solution, and therefore, can identify a partition that is the most critical to the specific task we are trying to learn.

Identifying partitions with the most significant performance changes provide a foundation for further experimental strategies that mean to develop the model's performance.

In order to make the experiment consistent, for all of the test cases the same ML dataset gets used for training that is made up of 15000 inputs (see Table 6 in Section 5.1.1 for more details). The GPT2 generative performance is evaluated based upon a generated collection from the respective training weights, using the BLEU and SBLEU metrics. Furthermore, the experiment extends the analysis to the genre discriminator network that is built on top of the GPT2 model and trained using its final hidden layer output. Each GPT2 model, fine-tuned using a different input partition approach gets its private discriminator that is trained separately after GPT2 using the same ML dataset. Moreover, after the training process, we evaluate the discriminator's genre classification[18] accuracy on a LL evaluation dataset that makes a total of 6000 samples, and on generated collections consisting of 1500 songs. In order to generate a lyrics collection for a specific setup, a GPT2 model is first fine-tuned using a left-to-right language modelling objective and a particular input partition. Then using its learned weights, we predict the lyrics, token by token - given the previous sequence as a condition with the same input partitions used during the training.

The experimental results will be first explained from the GPT2 model perspective and then from the discriminator point of view.

Table 10 illustrates the quality and diversity scores of generated collections produced from weight distributions trained using a different input construction setting. For simplicity, throughout the analysis, the different input settings will be referred to as rows in the table where: row 1 represents the quality and diversity scores of a full input setting comprising of the Token

[18] For generated lyrics evaluation, the GPT2 genre conditional feature was used as the correct class label for the discriminator. Whereas, in case of the evaluation dataset, the Token Type IDs value was used.

IDs[19], Token Type IDs[20] and Position IDs[21]; row 2 presents scores of the prior but without the Position IDs; row 3 stands for an input without the Token Type IDs; and lastly, row 4 displays scores for the input without the Token Type IDs and Position IDs.

GPT2 Model - Generative Performance		
Input Partitions:	BLEU^{Quality}	SBLEU^{Diversity}
1: TIDs + TTIDs + PIDs	39.5%	20.6%
2: TIDs + TTIDs	26.5%	15.5%
3: TIDs + PIDs	39.8%	20.7%
4: TIDs	19.5%	13.4%

Table 10: Performance of a GPT2 model fine-tuned using different input construction strategies. The two scoring columns correspond to the performance measures, quality (left) and diversity (right), produces based upon 1500 generating lyrics from respective weight distributions.

When comparing row 1 and 3 quality and diversity scores, it is notable that the values are almost the same, i.e., 39.5% and 20.6% to 39.8 and 20.7% respectively. In other words, the score of an input strategy without the TTIDs is almost identical to the one with TTIDs. Meaning that the PIDs bring the full performance gain while the TTIDs do not contribute, in a case when added together with the PIDs. The hypothesis is further confirmed by comparing rows 3 and 2 since the scores of a setting without PIDs depart far from the optimal. Furthermore, when comparing rows 2 and 4, a small performance gain is notable, showing that TTIDs alone do make a small impact.

[19] Correspond to the input words

[20] Each input lyrics of a different genre type get a different constant ineger value that is the same for all the positions. The metal genre lyrics get ones added for every position, pop lyrics get twos, and country lyrics get threes. It means to help the model classify the different lyrics types.

[21] Incremental integer sequence that indicates the model a spacial distance between words of the input.

Interestingly, given the prior analysis, the TTIDs bring no benefit when in the presence of the PIDs; however, alone do make a small contribution. This indicates that during the training process, given an input comprising of a full setup, the GPT2 model puts its full attention to the PIDs only; however, in a case where there are none, the model makes use of the TTIDs to slightly improve the performance. Therefore, such results imply that the PIDs are a critical input partition necessary for an optimal model's generative performance, while the model could do without the TTIDs partition for this particular task.

From now on, the analysis focus on the genre discriminator that is trained on the GPT2 model's last hidden layer outputs. Note, during the training process, in order not to bias the discriminator towards the correct genre class; it is crucial to remove the special tokens from the GPT2's input. Otherwise, the special tokens would be reflected in the contextual hidden state representations used to train the discriminator, and as a result, would make the training objective banal which would transition to poor after training classification performance. See Section 5.1.4 for a more in-depth explanation of the discriminator training strategy. Furthermore, as it was stated at the beginning of this section, the discriminator evaluation consists of two parts: validating using evaluation dataset examples and on lyrics examples generated from the model. For the former, we feed inputs to the GPT2 model using the same partition setup as the one used during this specific model training. However, for the generated lyrics classification, in order to make the task tougher, we remove the additional partitions while providing only the word tokens. See Figure 23 for an overview of this experiment's construction decisions-flow.

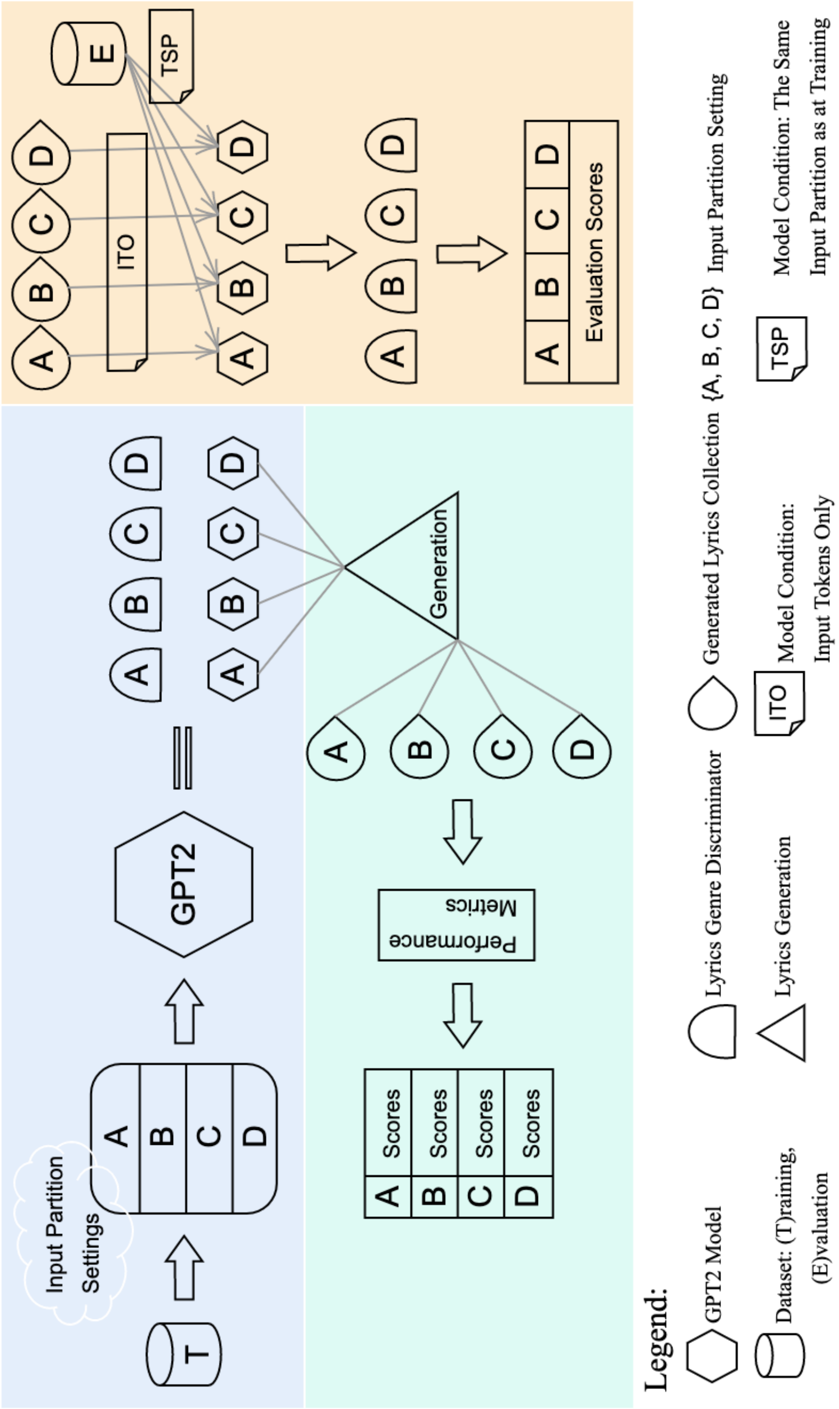

Input Partition Settings
T
A
B
C
D
GPT2
A
B
C
D
Scores
Performance Metrics
Generation
ITO
TSP
E
Evaluation Scores
Legend:
GPT2 Model
Dataset: (T)raining, (E)valuation
Lyrics Genre Discriminator
Lyrics Generation
Generated Lyrics Collection
{A, B, C, D} Input Partition Setting
ITO Model Condition: Input Tokens Only
TSP Model Condition: The Same Input Partition as at Training

Figure 23: A schematic representation of the input partition experimental choices (best visible in colour). The blue background corresponds to the GPT2 model training process. The green area represents lyrics generation and evaluation processes. The orange area displays the discriminator evaluation process.

Furthermore, a keynote to remember throughout the analysis is that the generated lyrics classification scores depend on the GPT2 model's generative performance. Differently put, the discriminator is trained to classify lyrics that are dataset-alike; however, when the GPT2 model generates a song unlike the ones seen in the training dataset, the discriminator has a mean to missclassify. Nevertheless, the prior is also depended on the training evaluation error since the discriminator can also make a mistake. However, given that the training evaluation scores are independent of the GPT2 vulnerability; should display an impartial influence of the particular input partitions, and thus those scores will be analysed first.

Genre Discriminator - Classification Accuracy Scores		
Input Partitions:	Lyrics Class. Acc.	Training Eval. Acc.
1: TIDs + TTIDs + PIDs	50.1%	96.4%
2: TIDs + TTIDs	34.2%	99.7%
3: TIDs + PIDs	58.0%	78.3%
4: TIDs	45.4%	69.7%

Table 11: Classification accuracy scores of a genre discriminator layer that was trained using a GPT2 model's hidden state representations, that were produced from weight distributions learned using the displayed input partition settings. Each row corresponds to a different discriminator, where all get evaluated at two distinct scenarios: after training using 6000 evaluation samples (right), and based upon generated lyrics collections consisting of 1500 samples (left). Note, during the evaluation process, the hidden state representations that were fed into the discriminator were conditioned based upon the corresponding input partition settings; however, in case of the generated lyrics evaluation, the hidden states were conditioned on pure input tokens (TIDs).

Table 11 illustrates discriminator's genre classification accuracy scores. When abstractly examining the training evaluation accuracy, it is notable that the input without the PIDs scores the highest - 99.7%, whereas the second-highest, 96.4%, is the full input partition; therefore, meaning that the position tokens make the classification task harder. Furthermore, when comparing rows 2 and 3, the scores indicate that the TTIDs help the discriminator achieve higher accuracy in contrast to the PIDs, by a considerable amount of 21.4%, thus signifying strong dependence of TTIDs for classification. Lastly, it is notable that all of the additional partitions bring in some value to the discriminator classification as a result of pure TIDs displays the lowest performance out of all setups.

Now, the discriminator training evaluation scores will be analysed with respect to the imperative lyrics classification accuracy in order to see correlations between the two evaluations' scores.

One would expect the highest training evaluation accuracy, displayed in row 2, to correlate with the generated lyrics classification positively; however, this is not the case since the prior displays the lowest score of 34.2%. Given the analysis from the Table 10 indicating a strong GPT2's reliance on the PIDs, in this case of its absence, the GPT2 produces contrasting representations that the discriminator unrecognises, and thus often miss-classifies. This shows the importance of keeping the same input partitions at both, the training and evaluations stages, especially the ones that the GPT2 model mostly relies upon - in order for the discriminator to be useful. Furthermore, when inspecting row 3 displaying an input partition without the TTIDs, we see a low training evaluation accuracy of 78.3% and the highest generated lyrics score of 58.0%. Given the prior hypothesis stating the strong discriminator dependence on the TTIDs, and the firm GPT2 reliance of the PIDs, such behaviour is exhibit because the lack of TTIDs and the presence of PIDs does not heavily hurt the GPT2 representations, and thus, the discriminator can associate the representations with the ones at training. Moreover, given that the lack of TTIDs downgrades the discriminator training performance which positively transitions on the imperative generated lyrics evaluation, it is reasonable to state that, at training, it is advantageous not to bias the discriminator towards the correct class since it does not transition to after training performance. Therefore, providing inputs without TTIDs would make the discriminator rely on the word tokens and their positions more, which would make the training task harder to learn, however, the training evaluation score is more reflective of the actual performance.

To summarise, in this experiment, we fine-tuned a GPT2 model using different input partition setups in order to reveal and compare the generative performance gains of the individual partitions. For the performance comparison, we generated lyrics collections from the corresponding model weights, which we then evaluated using SBLEU (diversity) and BLEU (quality) metrics. Analysis of the evaluation scores had shown that the Token Type IDs and the Position IDs, when separately added to the lyrics tokens, benefit the model performance while the Position IDs are considerably more beneficial. However, when compared both input partitions added to the input words against a setup without just the Token Type IDs - the performance scores were almost identical, showing that the Token Type IDs brings no value when in presence with Position IDs, and thus signifying that the Token Type IDs are not necessarily required for such task. Alternatively, indicating that the Position IDs partition is the most critical for optimal GPT2 model performance.

Furthermore, the experiment extends to an analysis of the influence of input partitions on the lyrics genre discriminator, that is a one layered classification network trained on the GPT2 model's last layer hidden state representations. Each GPT2 model fine-tuned using the specific input partition setting was given its private discriminator, that was trained separately after the GPT2 model using the same dataset. The discriminator classification performance was evaluated at two distinct scenarios; after training, utilising a non-intersecting evaluation dataset; and on generated by the GPT2 song collections. During the evaluation, the hidden state representations that were fed into the discriminator were conditioned based upon the different input partition settings; however, in case of the generated lyrics evaluation, the representations were conditioned on pure lyrics tokens. By analysing the discriminator training evaluation alone, the discriminator showed to heavily rely on the Token Type IDs since when in case of its absence at the condition, the classification accuracy was dropped from 99.7% to 78.3%. Nevertheless, when contrasting the scores of a full input partition setting against a one without Position IDs, we see that the performance of the former dropped by 3.3%, meaning that the Position IDs marginally confuse the discriminator. Additionally, further examination contrasted the training evaluation scores against the imperative generated dataset classification accuracy to inspect correlations between the two evaluations. Input partition combination of the Token IDs and Token Type IDs have shown the biggest negative correlation in respect to the other setting, meaning that the training evaluation performance score did not translate to the after training performance on generated lyrics. Given the prior analysis of the GPT2 depended on the Position IDs, at this case of its absence, the GPT2 has generated lyrics that the discriminator has often miss-classified. Alternatively, given the experimental choice of conditioning on the generated lyrics provided as only token ids, the hidden state representations of the GPT2 model were too different from the ones seen by the discriminator at training. This signifies that when training a model with a specific input partition, after training, it is essential to keep it precisely the same as it was during training - since a lack of partition makes the GPT2 to produce slightly different hidden states which confuse the discriminator.

5.2.3 Getting the Most Out of Top-k & Top-p

An optimal output sampling technique is an essential aspect of any generative model. Without an appropriate sampling strategy, outputs from a prosperous model can display poor quality and not realistically reflect its full potential. See Section 5.1.3 for more details about model sampling.

The most commonly used sampling strategies are top-k and top-p sampling (Holtzman et al., 2020). This experiment aims to find a better performing method of the two and the most optimal hyperparameter setting of it. In this experiment, we train a GPT2 model using a medium-size dataset (15K inputs) and then, using all of the different top-k/p settings, generate 1.5k lyrics for the quality and diversity score measurement. In more detail, for each genre class in the dataset, we generate lyrics from 100 seeds - ranging from 0 to 99 - where for each seed we use a batch of size 5.

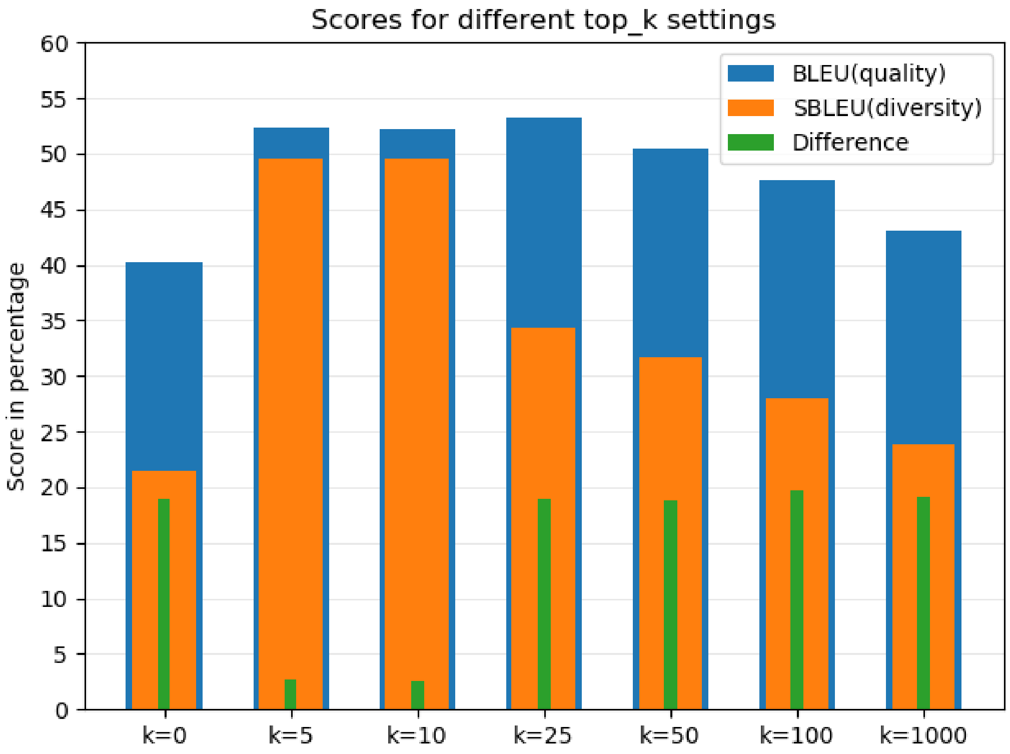


Figure 24: SBLEU and BLEU scores of lyrics collections generated using different top-k hyperparameter settings. The GPT2 model from which the song collections were produced was fine-tuned using the ML dataset that comprises of 15000 inputs. The evaluation is based upon 1500 generated samples.

Recall the SBLEU(diversity) and BLEU(quality) scores. The BLEU score reflects the n-gram count similarity of the generated samples against the training dataset, where the higher the score, the more similar the generated samples to the dataset. Whereas, the SBLEU contrasts a one generated song against all the others, for a given collection. In SBLEU metric, the higher the score, the more similar the outputs; thus, lower score implies greater diversity. Therefore, we strive for the diversity score to be as low as possible, and the quality score to be high. Consequently, this would mean that the model can generate diverse and dataset-like lyrics. Subtracting the diversity score from the quality displays a total performance gain which can highlight the most optimal hyperparameter setting.

Firstly, we are going to inspect top-k scores, which are displayed in Figure 24. By inspecting the bar chart, we can see that one proper performing setting is k=0, with the diversity being 21, quality 40, and the difference between the two 19. Another, also high-performing setting is k=1000, with diversity 24, quality 44, and the difference 19. Generally, we want a setting that has the greatest difference: however, in a case when the two green lines match performance; we need to compromise the other scores in order to define a model that best fits into our use case, i.e., diversity vs quality. One may want a less diverse model, however, with outputs reassembling the dataset more. Alternatively, one may prioritise diversity over quality. In this experiment, we value a balanced setting, and thus, chose k=1000 as the optimum. Note, top-k=0 means no logits filtering: thus, all logits are taken into account. Therefore, having no sample filtering strategy can produce outputs of satisfying quality. Furthermore, k=5 and k=10 display the worst performance, which is apparent by the small difference and high diversity score. When further inspecting the generated outputs from these settings, it is noticeable that they tend to collapse into a repetitive mode (see lyrics samples in Appendix 14).

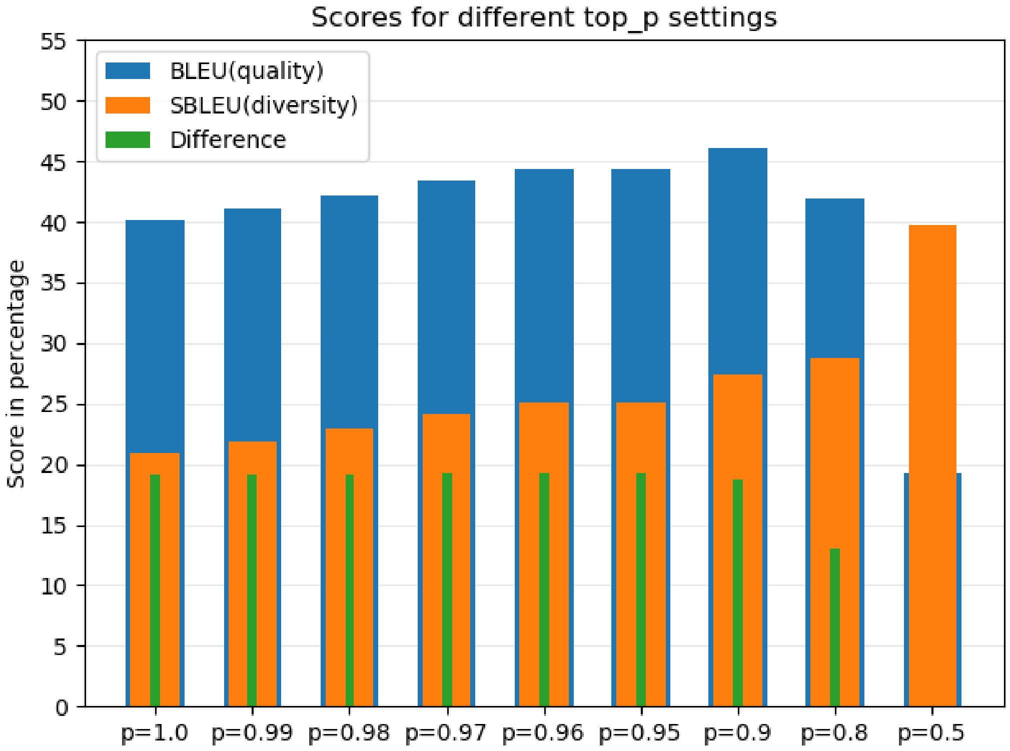


Figure 25: SBLEU and BLEU scores for lyrics collections generated using different top-p hyperparameter settings. The GPT2 model from which the song collections were produced was fine-tuned using the ML dataset that comprises of 15000 inputs. The evaluation is based upon 1500 generated samples.

Moreover, Figure 25 illustrates the top-p results. A commonly visible behaviour is that the results of the threshold p ranging from 1 to 0.9 increase in the quality and diversity scores, given a smaller p. In other words, the decreasing threshold p makes the outputs less diverse; however, more dataset-a-like. When considering the performance gain values (green lines), it is apparent that the model performs similarly for all the settings mentioned above. Therefore, again, the choice of the best setting depends on which compromise is more valuable to the user: diversity or quality. In the choice of the optimal threshold p setting, we went for a middle ground and chose 0.95. This is because this setting displays a reasonably good balance between the quality and diversity, and when inspecting the generated collections manually, they display a coherent and aesthetic text. Furthermore, when decreasing the threshold below 0.8, in contrast to the previous settings, we can see a sudden drop in the performance, which is given by the lower quality and higher diversity score, meaning that the generated words are repetitive and less often subset with the dataset in the n-grams.

Generated Song Lyrics - Feature-less Approach - ML Dataset	
Condition: Metal	Condition: Metal
Top-k = 1000	Top-p = 0.95
I found myself in a place where I turned to walk beneath I've a place where I belong I run and I've all flown And somehow find it anywhere Where there's hope for the sun still shining on But I can't find my way in myself Where the sun still shines on the trees What have I done, what have I done to you? I remember I was smiling in my youth But I don't realize the past behind closed doors The door's wide open that doesn't open Excuse me, don't step inside What have I done, what have I done to you? Life doesn't exist, this is how it used to be I chose to play along I gave my life to the monsters It seemed nothing, nothing at all	These three gifts keep in my heart, They seem quite safe, and they go my way To keep balance between us two, These three gifts keep in my heart, You see, I play with fire, That keeps our both together. There's no time for each one to be born. It seems the other gifts I bring Cannot be bought or sold. If you gave a little sweet tomorrow I would have given you more than I have If things could come my way, And all you want is you to see me, I'm not gonna waste another tear I take off all I can find These three gifts keep in my heart, They seem quite safe, and they go my way To keep balance between us two, These three gifts keep in my heart, You see, I play with fire, That keeps our both together.

Table 12: Lyrics generated from a GPT2 model trained on Dataset of size 15000. On the left, an output produced using top-k=1000 sampling strategy. On the right, lyrics produced utilising top-p = 0.95.

For both sampling strategies, we can see that the settings of top-k = 0 and top-p = 1.0 display a satisfying generative model performance since the evaluated lyrics collections are the most diverse while demonstrating high dataset resemblance score. Therefore, proving that it is possible to generate desirable samples from a model without the top-k and top-p strategy. However, experimenting with the threshold values have shown to trade the model diversity for a greater quality, which may be beneficial for some tasks. In the case of lyrics generation, we want the model to resemble the dataset well so that the outputs look human-written, and thus we chose the k=1000 and p=0.95 as the optimum. Example of lyrics is displayed in Table 12. The k = 1000 and p = 0.95 display a very similar performance in terms of the best BLEU and SBLEU score. Additionally, when inspecting the generated outputs, both of the collections contain high-quality lyrics with coherent and reasonable text. Moreover, for both strategies, the model generates songs with repetitive sentences, which is the required behaviour since most of the songs contain naturally repetitive chorus (see Appendix A.3).

To conclude, sampling from a model can provide satisfiable results without any particular sampling technique available today, since the model showed to generate excellent performance outputs based on a simple maximisation sampling. However, using the top-k/p sampling strategy helps to raise the quality, which is worth testing out, in particular for use cases where the dataset resemblance is a priority. Furthermore, p sampling is more flexible and using any threshold between 1 and 0.9 can be considered 'safe' in terms of the output quality. In contrast, the top-k requires an experimental phrase to find a well-performing top logits threshold.

5.3 Fine-grained Control Over the Style

In this Section, we go a step further and expand the conditional features to include the genre, year, author, and song name. Generating from a model trained using such metadata, we can achieve more fine-grained control over the style of the lyrics. In other words, we can cross-breed the outputs by conditioning on different feature combinations, e.g., we can condition a pop author to produce a country-like song in the style of the '50s. By providing a word sequence as the song name condition, the model often reflects subsets of the conditional tokens in the generation. Furthermore, thanks to the input dropout strategy, conditioning on all features is not compulsory, i.e., we can only condition on any subset of features or even none. Nevertheless, we can also generate any of the conditional features, e.g., we can first generate a song given its name and then regenerate the song name so that it suits the lyrics better.

Training the fine-grain controlled model requires an expanded dataset, that for each input song, contains the appropriate metadata. Having an input described by a number of features it may be unclear how to feed it into the model so that, after training, we get the capability of generating any of the features. In this approach, we set a predefined order of the features, i.e., genre, year, author, song_name and the lyrics and indicate a start and the end of each feature, using special tokens. In particular, conditioning the model on a full context would look as follows (note the missing lyrics end tag):

[s:genre]...[e:genre] [s:year]...[e:year] [s:author]...[e:author]
[s:song_name]...[e:song_name] [s:lyrics]

Using the unique tokens around the different metadata allows the model to map the appropriate input tokens to the corresponding special tokens: so that the starting tags can be used to condition the data seen at training. Intuitively, the end tag is used as an indication to stop generating for a given feature. In terms of the input partitions, for the Token Type IDs, we represent each feature with a constant value which is incremental along with the features. Whilst, the Position IDs remain the same as in the previous experiment.

Furthermore, in order to allow the model to predict given a subset of conditions, the training input requires a "features dropout" function at the training process. In this approach, we first, randomly drop all features but lyrics with the probability of 25%. Further on, in case the condition is unsatisfied, we set a probability of 10% to drop the features individually. This way, for each input from the dataset, there is some chance that the features will be lost and so the model learns to predict given only partial information.

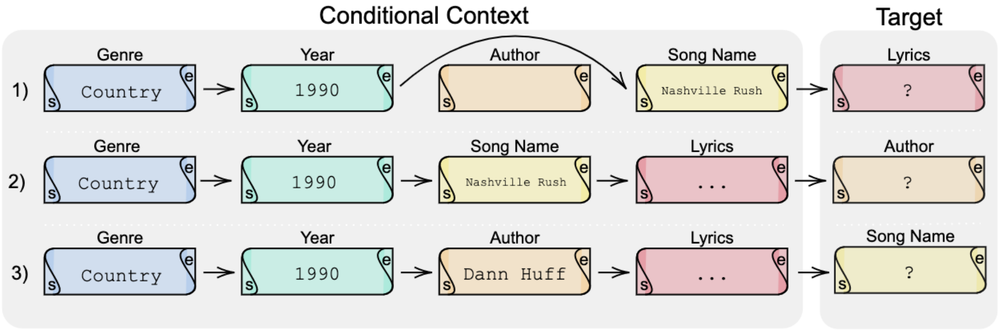


Figure 26: An illustration of styled lyrics generation. 1) Model conditioning using partial context. 2) The model generates an author using the previously generated lyrics. 3) Regeneration of the song name to a one that better suits the new conditional context.

By using the metadata generative modelling approach, the further subsections explore how the dataset size expansions translate on the model performance, as well as how increasing the feature variety in the training data influences the model's generative qualities. Lastly, we investigate how the output qualities change when the model is conditioned on a different subset of the conditional features.

5.3.1 Data Variety & Dataset Size Experiment

The goal of this experiment is to identify performance correlations of the GPT2 model trained using two lyrics data banks that differ in the feature variety, i.e., one bank contains lyrics belonging to 3 genres**(3GEN)**, whereas the other is composed of 10 **(10GEN)**. From the two data pools, we create two groups of incremental in size datasets on which the GPT2 model is separately trained. In particular, the 3GEN datasets are made of metal, pop and country lyrics. Whereas, the 10GEN datasets are composed of the prior and the: electronic, folk, indie, jazz, hip-hop, RB, and rock. In order to make the comparison impartial, each respective dataset size from the two pools contains the same total amount of inputs that are proportionally segmented in regards the number of songs of each genre. By comparing the quality and diversity scores of the generated lyrics, we inspect how the feature variety influences the model performance over an expanding dataset size.

Furthermore, in this experimental investigation, we also contrast the performance of the previously explored feature-less generative model strategy (see Section 5.2.1) against the metadata approach explored in this Section. By comparing the two approaches, we explore the benefits of the metadata in relation to the generative model performance.

Note, the further experimental analysis assumes that one has limited data and computational resources and wants to achieve a well-performing model given the smallest costs. In most cases, the more data provided, the better the outcome; however, this experiment looks for a satisfiable balance between the performance and the resources.

In order to generate a lyrics collection for a given model, we randomly sample from all of the conditions of the respective model's training dataset. Furthermore, all of the generated collections consist of 1000 songs, where each is conditioned on a different seed. The GPT2 hyperparameters that mostly leverage its learning are: learning rate = 0.0000625, warm-up learning rate = 0.0002, batch size = 1, the number of epochs = 4, and the top_p = 0.95.

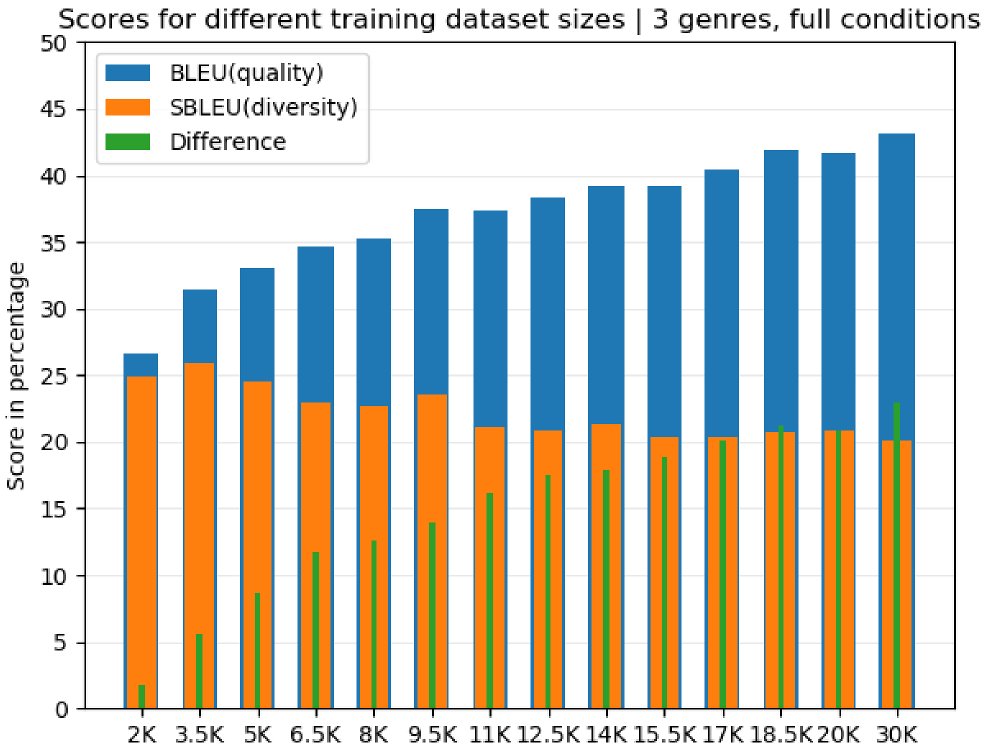


Figure 27: SBLEU and BLEU scores for generated lyrics collections. The models used to produce the evaluated lyrics were trained on varying dataset sizes, which are composed of lyrics belonging to 3 genres that are described by four features, i.e., genre, author, year and song name. In order to generate the outputs, the models were conditioned on all of the features (3GEN(f)) that were randomly selected from the respective datasets.

Figure 27 demonstrates the performance scores of the GPT2 trained on 3GEN datasets. It is important to highlight that when increasing the number of inputs, there is a constant improvement in terms of the diversity and quality: best visualised by the performance gain (green line). However, when reaching about 11K inputs, the diversity starts to become stable indicating that score of 20 might be an optimum given the specific training setup. Whereas the quality carries on improving, indicating that the model's outputs become more dataset-alike, thus the model still fits the data. Additionally, these results give an idea that in order to train a well performing model, it is reasonable to just use a dataset size of around 11K inputs since the output performance displays to be reasonable in contrast to the other training settings that require more resources. Moreover, it is notable that increasing the data size after 20K of inputs, brings a small improvement in the quality, which decays with the increasing number of inputs. This underlines the dataset size importance and constraints, i.e., it is possible to get a good satisfying model with only 11K training examples, however, any small improvement, costs a much greater amount of data than needed for a well-performing model. This is crucial when creating a dataset, especially when the process is expensive and very time consuming, since no one wants to waste time and resources.

When comparing these results to the dataset size experiment on a model trained without features (Section 5.2.1, Table 18), it is notable that the models exhibit the same performance at around 15k inputs, i.e., a score of 20 in SBLEU and 40 in BLEU. However, both models differ in the way they approach the same performance. In terms of the diversity score, the non-features model displays a low diversity score early in training, which given more data inputs, increases over training time; indicating less diversity in the outputs when exposed to more data. Whereas the features model exhibits the exact opposite behaviour, i.e., the diversity positively correlates with the input size. This shows that training a model using a featured dataset helps the model to categorise the lyrics examples in its latent space, and thus, when conditioned on a full set of features, the model can generate from ‘more defined’ and less shared space. Which, as a result, transitions to the outputs being more diverse. Whereas, in case of pure lyrics dataset, the model does not have meta information to separate the lyrics, and thus, when conditioned, more often collapses to a more conventional space, producing less diverse outputs. Furthermore, the fact that given more data the diversity decreases indicates that there are not enough features to properly categories the model space, and thus, the model starts to overfit the data, resulting in less diverse and more similar to dataset outputs.

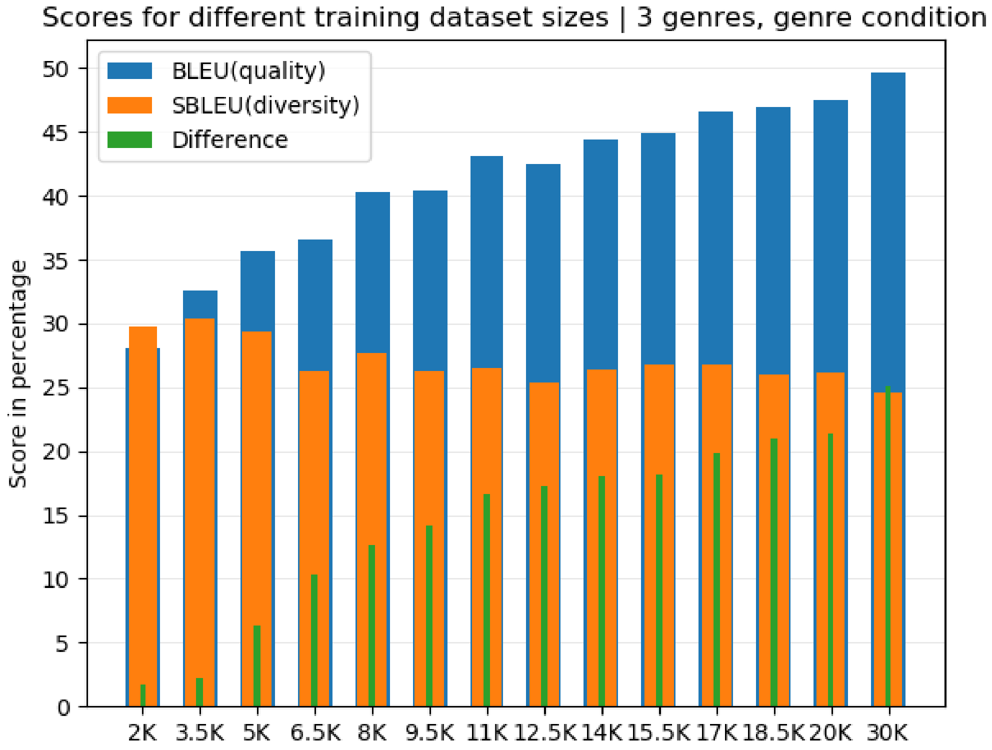


Figure 28: SBLEU and BLEU scores for generated lyrics collections. The models used to produce the evaluated lyrics were trained on varying dataset sizes, which are composed of lyrics belonging to 3 genres that are described by four features (3GEN(f)), i.e., genre, author, year and song name. In order to generate the outputs, the models were conditioned only on the genre feature (3GEN(g)), which was randomly selected from the respective datasets.

Furthermore, Figure 28, illustrates the quality and diversity scores for generated collections conditioned only on the genre feature 3GEN(g). In comparison to 3GEN(f), the outputs improve on both scores in correlation with the increase in dataset size. However, the outputs are less diverse and show higher quality measure, i.e., about 5% increase at both metrics, for all dataset sizes. This is because the outputs are not constrained with as many conditions, and thus, the model predicts the more frequently seen words, which span a higher quality and less diverse outputs (see Appendix ?? for examples).

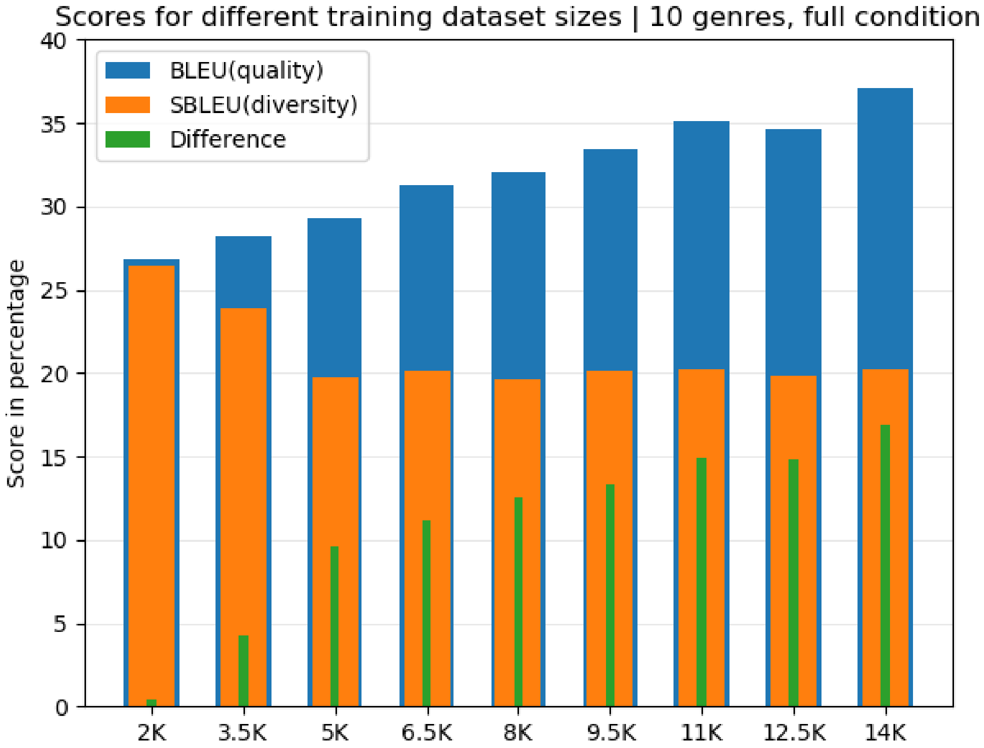


Figure 29: SBLEU and BLEU scores for collections of generated lyrics from models trained on varying dataset sizes. All of the datasets used to train these models were composed of lyrics belonging to 10 genres that are described by four features, i.e., genre, author, year and song name. In order to generate the evaluated collections, the GPT2 model was conditioned on the full set of features (10GEN(f)) randomly selected from corresponding datasets.

Nevertheless, Figure 29, demonstrates the scores of models trained on datasets composed of 10 genres (more various data) and the lyrics generated using all conditions (10GEN(f)). In contrast to 3GEN(f) scores, the outputs' diversity score approaches the optimum of 20 much sooner, i.e., at 5k inputs in contrast to 15.5k inputs. This is because the dataset is in essence more diverse, and thus, the model learns from a more diverse data which transitions on performance. In other words, it contains examples belonging to more genres, which contain songs of a greater range of authors that may potentially use a different subset of vocabulary in their lyrics. Therefore, the model more often predicts a greater subset of vocabulary which transition into the optimal diversity score early in the dataset size. Furthermore, in terms of the quality score, the model has a slightly lower performance of around 2% when compared to 3GEN(f). This is because the data is naturally less similar and thus, it is harder for the model to learn and achieve a higher quality grade.

To conclude, this section compares the performance of a GPT2 model trained on range of dataset sizes, with a varying input's origin, in order to spot correlations between the performance and the increase in dataset size as well as correlations between different data origins. The lyrics contained within the datasets are accompanied by feature descriptions, i.e., genre, year, author and the song name. Furthermore, the experiment involves two sets of datasets on which the model is trained, and the analysis is performed. The first set, (3GEN), defines models trained on datasets whose lyrics only belong to three different genre classes whereas the second set of datasets (10GEN) is composed of 10 genre classes.

When comparing 3GEN and 10GEN outputs generated based on a full set of conditions, the model trained on 10GEN dataset has shown to approach a better diversity score after a smaller number of inputs, i.e. 5k vs 15k, indicating the importance of dataset diversity for more diverse outputs. Furthermore, in terms of the quality score, the models trained on more varied datasets exhibit worse performance (approx. 2%). Indicating that a more diverse dataset is harder for the model to learn.

Moreover, further comparison between models trained on datasets with and without the features has shown the metadata to have a positive influence on the model's performance. That is, when training with features, over an increasing number of training examples, the model produces outputs that become more diverse, while also improving on the quality. Whereas when training without metadata, the produced lyrics also improve on the quality score; however, they become less diverse given more training inputs. This puts the enriched dataset to be advantageous since one wants the model to improve consistently given more data.

6 Discussion

This section covers the main limitations of the presented work, which includes the computational resources, dataset filtering, and evaluation metrics. Furthermore, we discuss undiscovered experimental paths that have the potential to serve as a continuation of this work.

6.1 Limitations

The central limitation of this thesis is related to the accessible computational resources, which has restricted the range of available investigations and its experimental arrangements. An interesting experimental path could have been conducted on the architecture-level of the recent language models, such as the BERT or GPT2. However, given our limited resources, such work was not possible because it requires pre-training from scratch, which is computationally heavy. For an intuitive example, a recent competitive language model was trained on 1024 V100 GPU cards for a few consecutive days, where a fraction of such computational power is not attainable by most of the universities. As a consequence, we have focused on utilising the pre-defined architectures and its pre-trained weights for experiments on task-specific fine-tuning, which is an interesting research direction, however, not as significant and groundbreaking as language model improving. Furthermore, given that the University GPU cards have 16 GB of GRAM, loading a large model distribution was not possible since such weights did not fully fit; thus, we could not operate on the highest performance models.

Furthermore, the choice of the memory-related hyperparameters, such as the batch size which was limited to one, was also restricted due to GRAM requirements. Moreover, all of the experimental results are based upon a single computation. Having to re-compute the same experimental setting a couple of times, and then taking an average of the results, would provide a more reflective validation.

Another limitation of this work is the cleanliness of the data pool used for the training. Data is an essential aspect of any model training since it is what the model learns patterns from and strives to generalise. Despite the data cleaning process of removing blank inputs and lyrics containing special symbols or non-English letters; there are still some songs left within the English alphabet which are written in other languages, since they are difficult to filter out. This is a problem for two main reasons. First, the model learns to generate non-English lyrics occasionally. Second, when such songs make up a part of a generated collection whose diversity level is to be evaluated, the non-English words enrich the evaluation score. Therefore, getting rid of such samples from the dataset would make the model generate only native lyrics which would transition to more fair evaluation scores.

To evaluate the performance of a model, we first generated a lyrics collection (or used a dataset) and then measured its quality, diversity and uniqueness level. All of the mentioned metrics are based upon n-gram counts which, abstractly speaking, measure the similarity between set A and set B. Comparing a generated collection to the training dataset in such fashion provides an indication of the dataset resemblance, therefore the quality. Comparing one generated song against a collection of others, and repeating the same process for all the other songs and then computing an average over all of the results, indicates a diversity score. Whereas, counting all the words appearing only once in a collection and dividing by the total of all the other unique word whose frequency is greater than one, indicates the uniqueness.

Since all of the above evaluation metrics look for matching n-grams between two sets, the resulting scores are size-dependent. In other words, increasing or decreasing the generated collection size, or the dataset size to which the collections are compared results in different score reflection. This is because the more songs make up a set A, the more chance we have to find a subset of A in set B, and vice versa. Therefore, these evaluation metrics are only comparable in a constant experimental setup and are not useful for a global comparison against other people's models since there are no evaluation standards for everyone to follow. Moreover, this work does not implement any other competitive model for a fair comparison using these evaluation metrics. Therefore, despite the results indicating an excellent conditional lyrics generator performance, it is unknown how the model looks at the background of the other.

Nevertheless, these metrics are not always reflective of the actual model performance. For example, consider a poorly trained model that alternately generates coherent lyrics and songs containing non-existing words. Evaluating the diversity and uniqueness of a such generated collection would falsely result in scores indicating higher diversity and uniqueness since the non-existing words would make up more unique n-grams.

6.2 Future Work

Training a conditional lyrics generator requires a dataset of songs accompanied by its origin descriptors (metadata), so that, after the training process, we can use the metadata to condition the style of the output. However, training a model utilising an input feeding strategy that always uses the full set of features to learn the weights confuses the model when conditioned on only partial descriptors. Therefore, in order to introduce the capability of partial conditioning, the training input feeding strategy must occasionally drop out some and all of the features. The dropout probability of the metadata is a modelling choice that affects the output generation performance. For the final generator model used in this thesis, we have used a training input strategy that drops all of the features but the genre, with a probability of 25%, and when this condition is unsatisfied, individually drops the author, year and song name features with a probability of 10%. This way, the model learned to generate lyrics given any combination of the features.

As a continuation of this thesis, it would be interesting to conduct an investigation that would look into how the model output performance changes given all of the different possible conditional permutations. Such investigation would display which features, and combination of features, display the best performance. Furthermore, it would have been interesting to analyse the results with respect to different input feeding strategies. Since, for example, given a very high probability of dropping all metadata but the genre, like 70%, would most probably result in a model well handling a genre only conditional scenario. Finding correlations between the conditional feature permutations and different input feeding strategies could help the developers to make a decision on which dropout probability would be best for their particular use case to achieve the most optimal performance.

Furthermore, a typical song is made up of several verses and a repetitive chorus. Enriching the dataset with such lyrics part information and using special tokens to indicate them at the training, would extend the model capabilities to generating a particular part of a song. An appropriate input feeding strategy could even allow for regeneration of the chorus given the verses as a condition, or vice-versa.

Moreover, in the preliminary section, it would be interesting to compare BERT and GPT2 models, however, not by the performance on some auxiliary task, but by the internal representation modelling. In particular, it would be interesting to see how unidirectional (GPT2) and bidirectional (BERT) language model representations differ in a multilayered process, i.e., by investigating the evolution of representations of individual tokens. Such an investigation could characterise how the learning objectives determine the information flow of a model. The examination could start by observing the nature of changes a token undergoes from layer to layer, and then expand onto relationships between tokens, e.g., comparing rare and common words, and short and long in length words. Such research could reveal the hidden data modelling differences between the two commonly used language modelling objective, i.e., the LM and Masked-LM.

7 Conclusion

In this thesis, we carried out a systematic inquiry to discover and examine the generative capabilities of the recent state-of-the-art language modelling approaches, in particular: the pre-trained BERT and GPT2 models. Both of the models utilise a transformer-based architecture, which is based on a self-attention mechanism, to produce contextual representation as a function of the model, which makes them comparable. However, the central fundamental aspect that sets them apart is the Language Modelling objective, i.e., GPT2 uses a standard left-to-right objective, whereas, BERT trains its weights utilising a bidirectional Masked Language Modelling function. In practice, this means that BERT, in contrast to GPT2, exploits self-attention context representations that are computed in both directions in order to learn its weights. At the time of publication, BERT has advanced the NLP field by surpassing the benchmarks at eleven Natural Language Understanding tasks. Given the BERT exceptional performance at tasks requiring language understanding, and, at the time of the model announcement, limited research on the bidirectional modelling, we decided to conduct experiments on the BERT generative capabilities. The experiments have shown that it is not effective to sequentially generate word tokens from BERT, even when providing the conditional context from both directions. Therefore, proving that the Masked-LM objective, that was used in training, does not serve as a re-usable and straightforward approach for text sampling. In further experimentation on BERT, we conduct a series of tests on its bidirectional context representations. In more detail, we investigated whether the model conveys real-world knowledge information by using simple arithmetic operations on the word embeddings and seeing whether the resulting word makes sense for the given scenario. For example, whether the embedding of $king - man + woman$ results in a representation that is the closest to the word *queen*. The analysis has shown that BERT requires context to construct meaningful representations of the individual words, and by manipulating the model space using such representations, depending on the provided context, the model displays a conventional world-knowledge. The tests were conducted on the BERT-Base and BERT-Large weight distributions: BERT-Large often output representations, that in contrast to BERT-Base, did not make sense for the given tests, indicating that BERT-Large model is strongly under-fitted. Generally, the experiment has proven that it is possible to manipulate the model space using the bidirectional context representations: however, the process is very context responsive; thus, it is more reliable to perform such tasks on fixed vector representations, like the ones from word2vec algorithm.

Given that it is unclear how to easily and effectively sample a word sequence from BERT, for further experiments, we operated on its counterpart, a GPT2 model. Since the GPT2 is trained using a left-to-right unidirectional Language Model, we can generate from it in a straightforward sequential fashion (token by token). As a base for the experimental purposes, we create a conditional song lyrics generator by fine-tuning a pre-trained GPT2 model on a lyrics dataset. The experiments are divided into two lyrics generation modelling approaches; a simplified approach where the generated lyrics can only be styles into a specific genre; and a metadata approach that allows for fine-grained control over the style by introducing more conditional features, i.e., genre, year, author and the song name.

In the simpler approach, in order to encode the style of a specific genre, we define a special token for each different song type in the dataset and wrap the training inputs with the corresponding special tokens. Since the unique tokens are introduced at the fine-tuning stage, the pre-trained model has no prior knowledge what they should represent, and therefore starts correlating them with the respective words at the fine-tuning process. As a consequence, after training, we can use the special token to condition the style of the generated output. Furthermore, to help the model establish different lyrics types in its latent learning space, we element-wise add the input partitions, i.e., the Token Type IDs and Position IDs to the Token IDs, that are representative of the input sequence. In particular, for the Token Type IDs, we chose a constant value that is the same for all lyrics of the same genre. Whereas, for the Position IDs, we incrementally increase the value starting from one till the length of the input. The Position IDs help the model identify the end of the lyrics, whereas, the Token Type IDs mean to help differentiate between genres.

By applying the above training strategy, we conduct a series of experiments, which include the dataset size, input partition and the model sampling technique. Each of the empirical subjects is required to be experimentally analysed in order to achieve a well-performing generator. Furthermore, to evaluate the generative model performance at the investigations, we utilise the BLEU (quality), SBLEU (diversity), UNGC (uniqueness) metrics and a one-layered genre discriminator network, on the generated lyrics collections from respective model weights.

Motivated by the idea of limited dataset and resources for model fine-tuning, in the dataset experiment, we contrasted the performance gains with respect to the increasing data size. In more details, we trained a GPT2 model using four varied in size datasets: 600, 6000, 15000 and 30000. A general observation is that the quality of the outputs positively correlated with the increase in data size; however, the relative improvements gain decayed with the increase in dataset length. Therefore, given one has limited resources and the dataset, the results have shown that it is reasonable to fine-tune a GPT2 model only using 6000 input examples to achieve a satisfying output quality; the dataset extensions - from 6000 to 15000, and from 15000 to 30000 - only displayed marginal performance gains. Furthermore, when analysing the SBLEU scores, the model outputs displayed to become less diverse, given more inputs. This property positively correlated with the quality scores, which indicated that the model was learning, and thus, becoming more ‘fit’ to the dataset distribution. Moreover, the UNGC has shown that, given more data, the model outputs become more unique at three and four-word sequences. Therefore, using this specific training approach and the dataset, when increasing the data amount, the outputs improve at the quality and uniqueness but become less diverse. Given the complexity of natural language, we hypothesise this means that the model predictions become: more dataset-alike; occasionally predict the previously unseen tokens learned from the new data; generally, the prediction collapses to the most popular words whose frequency rises given the new data. Lastly, the genre discriminator scores have shown to be just above the guess-level, indicating the discriminator itself can be unreliable and may not truly measure the genre reflection in the outputs. Displaying that one layer on top of the transformer for simple lyrics classifying task is not big enough, and expansion in the number of layers is required in order to achieve more reliable genre evaluation metric.

For the input partition experiment, we fine-tuned a GPT2 model using different input partition setups in order to reveal and compare the generative performance gains of the individual partitions. Analysis of the evaluation scores had shown that the Token Type IDs and the Position IDs, when separately added to the lyrics tokens, benefit the model performance while the Position IDs are considerably more beneficial. However, when compared both partitions added to the word tokens against a setup without just the Token Type IDs - the performance scores were almost identical, showing that the Token Type IDs brings no value when in presence with Position IDs, and thus signifying that the Token Type IDs are not necessarily required for such task. Alternatively, indicating that the Position IDs partition is the most critical for optimal GPT2 model performance. Furthermore, the experiment extends to an analysis of the influence of input partitions on the lyrics genre discriminator, which was trained on the GPT2 model's last layer hidden state representations. The discriminator showed to heavily rely on Token Type IDs since, at a case of its absence, the classification accuracy has drastically dropped. In terms of the Position IDs partition, we saw that it marginally confuses the discriminator. Furthermore, we also found that the experimental strategy of first, fine-tuning a model using a particular partitions setup, and then, after training, conditioning the model on only the Token IDs has shown not to be useful. This is because the hidden state representations of the GPT2 model were too different from the ones seen by the discriminator at training. Therefore, signifying that when training a model with a specific input partition, after training, it is essential to keep it precisely the same as it was during training - since a lack of partition makes the GPT2 to produce slickly different hidden states which confuse the discriminator at the classification.

Lastly, we experimented with two output sampling strategies, called Top-k and Top-p. This investigation aimed to find a better performing method of the two and the most optimal hyperparameter setting of it. A carefully chosen sampling technique in respect to a specific task, as well as its the most optimal setting, is an important aspect of any generative model. Without an appropriate sampling strategy, outputs from a prosperous model can display poor quality and not realistically reflect its full potential. In more detail, we trained a GPT2 model using a medium-size dataset (15K inputs) and then, using different top-k/p settings, generated 1.5K lyrics for the quality and diversity score measurement. The analysis has shown that sampling from a model could do without any particular sampling technique available today since the model displayed to generate excellent performance outputs based on a simple maximisation sampling. However, using the top-k/p sampling strategy helps to raise the quality, which is worth testing out, in particular, for use cases where the dataset resemblance is a priority. Furthermore, p sampling is more flexible and using any threshold between 1 and 0.9 can be considered ‘safe’ in terms of the output quality. In contrast, the top-k requires an experimental phrase to find a well-performing top logits threshold, which when found, is competitive to the optimal top-p.

In the second experimental section, we explore the metadata approach of generator modelling. In this strategy, we go a step further and expand the conditional features to include the genre, year, author, and song name.

Generating from a model trained using such strategy, we can achieve more fine-grained control over the style of the lyrics. In other words, we can cross-breed the outputs by conditioning on different feature combinations, e.g., we can condition a pop author to produce a country-like song in the style of the '50s. Furthermore, conditioning on all features is not compulsory, i.e., we can only condition on any subset of features or even none. Nevertheless, we can also generate any of the conditional features, e.g., we can first generate a song given its name and then regenerate the song name so that it suits the lyrics better.

Training the fine-grain controlled model requires an expanded dataset, that for each input song, contains the appropriate metadata. Having an input described by several features, it may be unclear how to feed it into the model so that, after training, we get the capability of generating any of the features. In this approach, we set a predefined order of the features, i.e., genre, year, author, song name and the lyrics and indicate a start and the end of each feature, using special tokens. In terms of the input partitions, for the Token Type IDs, we represent each feature with a constant value which is incremental along the features. Whilst, the Position IDs remain the same as in the previous experiment, i.e., sequentially increase for each token position.

By using the metadata generative modelling approach, we explore how the dataset size expansions translate on the model performance, as well as, how increasing the feature variety in the training data influences the model's generative qualities over an expanding dataset size. In more detail, we trained a GPT2 model on data bank that contained lyrics belonging to 3 genres (3GEN), and on another set of data composed of 10 (10GEN) genres. When comparing 3GEN and 10GEN outputs generated based on a full set of conditions, the model trained on 10GEN dataset has shown to approach a better diversity score after a smaller number of inputs, i.e. 5k vs 15k, indicating the importance of dataset diversity for more diverse outputs. We also generated lyrics from 10GEN based on the genre field only. In contrast to the full context conditioning, the outputs have shown to be more dataset-alike, however, less diverse. Which is simply because the smaller number of conditions does not restrict the model's predictive space as much as all meta conditioning.

Furthermore, in this experimental investigation, we also contrast the performance of the previously explored feature-less generative model strategy against the metadata approach. By comparing the two approaches, we explore the benefits of the metadata in relation to the generative model performance. The analysis has shown that metadata has a positive influence on the model's performance. That is, when training with features, over an increasing number of training examples, the model produces outputs that become more diverse, while also improving on the quality. Whereas, when training without meta, the produced lyrics also improve on the quality score; however, become less diverse given more training inputs. This puts the training using the metadata datasets to be advantageous since one wants the model to improve consistently, given more data.

Therefore, for a successful lyrics generative model, it is best to train a GPT2 model using a metadata dataset, which introduces a fine-grained control over the output style. Moreover, using metadata brings more advantages. That is, when expanding the dataset, the model consistently improves on the quality and diversity of its outputs, in contrast to the training on featureless data. Furthermore, the more varied the metadata features are, the faster the model approaches the right diversity level early in training. Furthermore, as the experiments have indicated, a good compromise between the training resources and the model performance is to train the model on 11k of data inputs. However, depending on one's available resources, it is best to fine-tune using as much data as possible. In terms of the input partition, it is critical to use Position IDs in addition to the word tokens, since they bring much of the performance and indicate the model when to stop generating for a given feature. Also, after training, it is important to use the same input partition strategy that was used for the training. Concerning the output sampling technique, it is the most advisable to use the recent top-p sampling method since it allows us to have control over the output diversity while generating a coherent and aesthetically-pleasing text. Additionally, the threshold p is flexible, and small changes to it do not negatively impact the model's performance, therefore provides some space for a mistake when looking for an optimum. A GPT2 model trained with respect to the above indications generates lyrics of excellent quality and diversity, whose style can be controlled using several features.

A Appendix

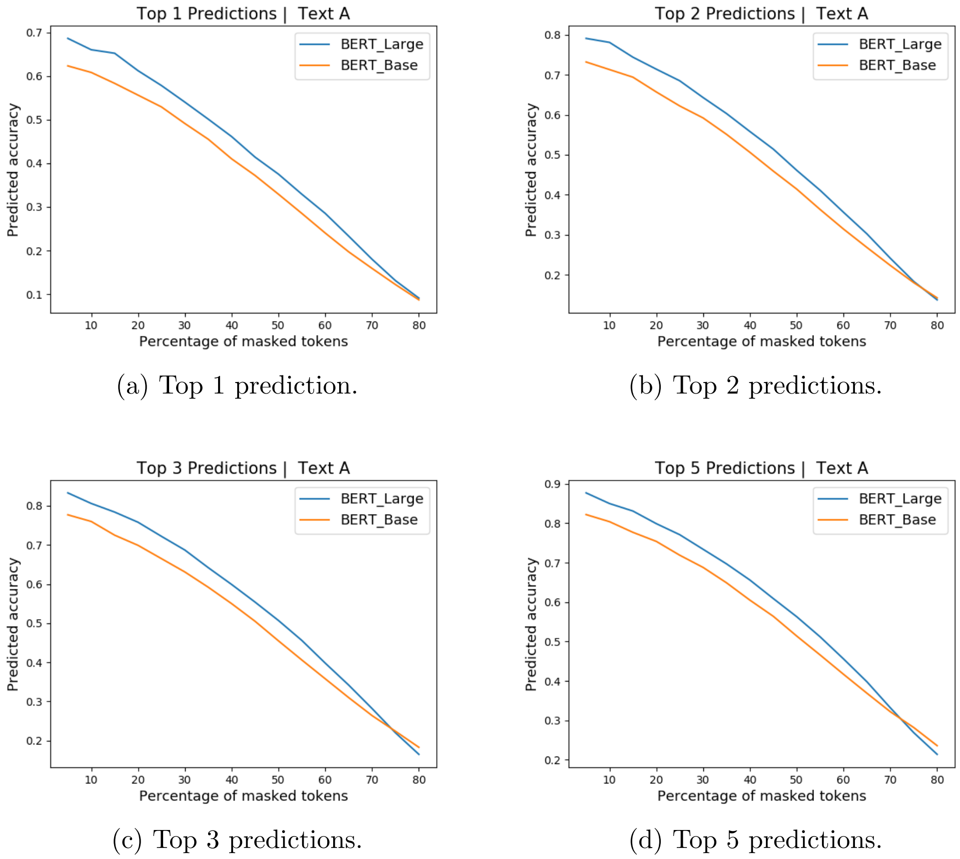
A.1 Example of Descriptive Text

Text A:

"Once upon a time, there was a girl named Beauty. She lived with her father and her sisters in a small village. Beauty was a beautiful girl. She was also hard-working. She always helped her father on the farm. One day, her father set out for the city. He saw an old castle and went in. No-one was in but there was food on the table. Then he walked around the castle. He picked arose from garden for Beauty. Suddenly an angry Beast appeared. He wanted to kill Beauty's father unless Beauty was brought to him. Beauty's father told her daughters what had happened. Beauty's sisters ordered her to see the Beast. Beauty went to see the Beast and had to stay at the castle. She felt scared, lonely and sad. She tried to run away but was stopped by the Beast. The Beast treated Beauty well. Soon, Beauty began to like the Beast. One day, through the Beast's magic mirror, Beauty saw that her father was sick. The Beast allowed her to go home. Her father was happy to see her. One night, Beauty had a dream. A fairly told her that the Beast was sick. Beauty hurried back and saw the Beast dying. She began to cry. Tears fell onto the Beast. Suddenly, the Beast changed into handsome prince. Beauty and the Beast got married and lived happily ever after[22]."

[22] https://www.scribd.com/document/44455701/Example-of-Descriptive-Text

A.2 Top-n Prediction Accuracy



(a) Top 1 prediction.

(b) Top 2 predictions.

(c) Top 3 predictions.

(d) Top 5 predictions.

Figure 30: BERT base and large performance comparison for top n prediction(s).

A.3 Top-k Top-p Experiment - Lyrics Samples

Lyrics Sample from the XL dataset	
Genre	Country
Lyrics	I sobered up in Houston In the bed of my pickup truck My head was hangin' so heavy I could hardly hold it up I got to thinkin' 'bout Sweet little woman I left all alone And that's when I started rollin' *Drinkin' my way back home* *Drinkin' my way back home* *Listenin' to a honky tonk song* *I hope the Devil in my soul don't steer me wrong* *'Cause I'm drinkin' my way back home* I can feel that Texas sun Down on this redneck of mine Every time I pop top I'm getting closer to the Arkansas line I left a trail of Lone Star beers From here to San Antone So, baby, here I come, don't worry *Drinkin' my way back home* *Drinkin' my way back home* *Listenin' to a honky tonk song* *I hope the Devil in my soul don't steer me wrong* *'Cause I'm drinkin' my way back home* Drinkin' my way back home Listenin' to a honky tonk song I hope the Devil in my soul don't steer me wrong 'Cause I'm drinkin' my way back home Here I come, baby

Table 13: A random lyrics sample from the XL training dataset.

Generated Lyrics: Feature-less Approach	
Top-k = 5	Top-p = 0.8
I'm just a girl in the middle of a summer A summer girl in a summer dress A blue summer night and I can't remember her name I can't remember the night we met I was just a girl in a summer dress In summertime I was just a girl in a summer dress In summertime I was just a girl in a summer dress Oh, I'm just a girl in a summer dress Oh I'm just a girl in a summer dress I can't remember the night we met I was just a girl in a summer dress In summertime I was just a girl in a summer dress In summertime I was just a girl in a summer dress In summer	see you with your eyes A beautiful smile And you have the power To say what you want to say I can't stop thinking about you The things you say to me I can't stop thinking about you I can't stop thinking about you The things you say to me I can't stop thinking about you The things you say to me I can't stop thinking about you The things you say to me I can't stop thinking about you I can't stop thinking about you The things you say to me I can't stop thinking about you The things you say to me I can't stop

Table 14: These lyrics present a regularly observed quality achieved with bad top-k/p settings. Note the overwhelming repetitive sentences.

A.4 Lyrics from Metadata Training

Generated Lyrics: 10GEN, 14K inputs		
Conditions		
Genre	Country	Rock
Year	-	-
Author	-	-
Song Name	-	christmas is a birthday
Lyrics	I went to the salesmen’s store I met the bright young man who looked so young He said his name was Johnny He worked hard but then he said, ””Sorry, Johnny, for all my trouble You never made the grade”” He just sold his guitar and he didn’t have any more He never came back again Well Johnny will be gone tomorrow He didn’t go home and he didn’t find a job He put up with poor Johnny and they started a war Then when he was made head of state by the people in the streets Johnny went back to San Antone But Johnny wouldn’t pay the bills and he couldn’t live on Johnny went to your place and he never spoke a word Johnny came home one fine day On the high horse he made his bid for gold So Johnny did his thing and the day is done Well Johnny will be gone tomorrow He didn’t go home and he didn’t find a job He put up with poor Johnny and they started a war Then when he was made head of state by the people in the streets Johnny went back to San Antone But Johnny wouldn’t pay the bills and he couldn’t live on Johnny went to your place and he never spoke a word Johnny came home one fine day On the high horse he made his bid for gold Now when Johnny finds a job He has to look out for the hopeless So he does his best to make the best of his life So he’s making money and running high And when he gets there he’ll realize That he made the best of it all And when he gets there he’ll realize he made the best And when he gets there he’ll realize he made the best Well the first thing that he said to his friend Is that you can have a horse Now if you’re tired of running and want to fly away There’s a horseman named Johnny and he’ll come on your mind And you can have a horse and a horsemen named Johnny Well he’s a long way from San Antone So you can have a horse and a horsemen named Johnny	they caught me making plans the boy said watch this mother***** when you’re feelin’ sad, cryin’ out so loud. startin’ up the fire from where you stand then about half of the flame goes to coal and the boy lies down and complains ””the sun don’t rise any more, so will you.”” then I saw the mess they made while I was layin’ on the snow with the roach that was feeding on it’s carcass I just made up my mind ””look over there, and there’ll be no more shadows”” then up out the chimney went the orange and I saw my shoes covered with snow then got him some Christmas pills then lay him down and told him ””gonna lie down and say you’ve got ’em”” ””gonna lie down and say that it’s all a big lie and here I come from a land where little kids cry and we don’t have to tell eachother anything.”” then he laid there and cried ””I didn’t mean to run you off with my man”” then he laid there and cried ””Gonna lie down and say you’ve got ’em”” (christmas is a birthday)
Metric		
BLEU	37.7%	39.6%
UNGC	1: 0.008, 2: 0.112. 3: 0.401, 4: 0.640	1: 0.016, 2: 0.216, 3: 0.635, 4: 0.840

Table 15: Song lyrics generated from a GPT2 model fine-tuned on lyrics dataset of size 14000, containing 10 different genre classes.

Generated Lyrics: 10GEN, 14K inputs		
Conditions		
Genre	Jazz	Hip-Hop
Year	-	2000
Author	bil miller	golden features
Song Name	from the inside out	her portrait in black
Lyrics	I just hope it's not too late for that Bill miller he's so country From the inside out I'll admit that I do My mama told me back then That one day, I'll go away This old farm town Where corn and peas grow Is where I learned about you In the west side of town Bill miller I hope you'll be home soon In your old pickup truck The boys and girls, they'll all be back by now And baby you'll see They'll work all day and play on the job From the inside out The one thing I hope you'll remember Is when you were just a child Oh darling it's the only way From the inside out Keep your old truck turning round Once it gets used to turning It'll just make it out this way And mama maybe I'll come back someday To stay in your arms Bill miller he'll always be my buddy From the inside out One thing I hope you'll remember Is when you were just a child Oh darling it's the only way	Her features are a perfect line of sight Her voice is clear and clear as the night Her skin is an angel's shield Her arms are warm and wild Her eyes are full of life Her hair is pale and soft Her face is pure and fair When I begin to speak I feel my very first words As she walks into my life A brief glimpse of her smile Lips of ruby light the sky Her eyes are pure and clear Her skin is an angel's shield Her arms are warm and wild Her eyes are full of life Her hair is pale and soft Her face is pure and fair When I begin to speak
Metric		
BLEU	48.5%	31.8%
UNGC	1: 0.006, 2: 0.144. 3: 0.480, 4: 0.780	1: 0.001, 2: 0.103, 3: 0.478, 4: 0.692

Table 16: Song lyrics generated from a GPT2 model fine-tuned on lyrics dataset of size 14000, containing 10 different genre classes.

Generated Lyrics: 3GEN, 30K inputs		
Conditions		
Genre	Metal	Country
Year	-	-
Author	-	-
Song Name	-	cold black days
Lyrics	Gods of war- ideology Thought and power-resist What you are in your own triumph Reform is futile You struggle against your doubts No flesh upon your hands What you were was light Your name is sacred The Saint of prophets You are and is) Behind the rising sun The highest power has risen The spirits of the other side Will guide you against the might Open your eyes-loathe the dead They'll show you the way I will protect you-Holy no spirit in your way I will show you the way I will protect you-Holy no spirit in your way Let you know the time The signs will be your guide We will show you the way No matter how you try to hide What we show is written in the sky The higher that you rise, the harder you'll fall You'll rise, you'll fall The higher that you rise, the harder you'll fall No one can fight the spirit of yesterday Darkness sails over you Taste the wisdom of sain Shall be for ever I need you-Holy no spirit in my way I will protect you-Holy no spirit in my way I will show you the way I will protect you-Holy no spirit in my way	I hear music as I pass the time is tickin' Winter leaves are fallin' But I know as I step into the sunlight I hear a song as I stand and watch the snow fly Walking in the frozen breeze Softly you whisper my name And you weep in remembrance For in dreams I can see your face And I can feel your embrace The warm sound of winter's silent night Slowly walkin' down the street And let the world know the joy you bring For you still had your song In the silence of the street I heard a voice as I passed the time is tickin' Tears are fallin' but I know as I stand by the river The voice that spoke will forever stand There's a place where the sun will shine The warm sound of winter's silent night The warm sound of winter's silent night
Metric		
BLEU	51.8%	55.5%
UNGC	1: 0.019, 2: 0.131. 3: 0.421, 4: 0.644	1: 0.001, 2: 0.074, 3: 0.326, 4: 0.719

Table 17: Song lyrics generated from a GPT2 model fine-tuned on lyrics dataset of size 30000, containing 3 different genre classes.

Generated Lyrics: 3GEN, 30K inputs		
Conditions		
Genre	Pop	Metal
Year	-	1980
Author	belinda carlisle	christina milian
Song Name	straight from hell	out in the open
Lyrics	i never took much time to explain this song and all of the words like falling apart to music. you never knew what to write, and how to play the bass but you know your fiddles are made to make violins so take the beat and break it up for you know there's one way to make it better right down to the elementary and all the things you never thought of before and it doesn't matter how far or how your head struggles if the melody will exist like some notes in harmony with the melody to the beat I never thought much about what it takes to make a man into someone like the finest guitar strings to play the beat it always takes a little to train you to a melody you know so take the beat and break it up for you know there's one way to make it better right down to the elementary and all the things you never thought of before and it doesn't matter how far or how your head struggles if the melody will exist	a highway for thieves and all the victims for there to be no refuge for those who hold stolen souls a long road for those who are betrayed and shunned coming not a light but a shadow obscuring the innocent out on the highway i find myself off at sea in trouble surrounding me but they stay true not a strand of drifting sand that moves with me but a nerve like a poison oak tree stronger than the rain I can see no escape I see if the prey is right beneath the man in the moon he leads his own circle a living out the dream he had staring into the night he brings the light but in the darkness he walks among us gave in to temptation, led to rebellion chasing dreams out on the highway for those who hold stolen souls now it is time to face the day to break the chains that bind us to the road of destray and our souls to make our own way i close my eyes to regain the feeling of a broken world and the day we are born beneath the man in the moon
Metric		
BLEU	55.7%	46.9%
UNGC	1: 0.001, 2: 0.054. 3: 0.324, 4: 0.607	1: 0.005, 2: 0.142, 3: 0.528, 4: 0.850

Table 18: Song lyrics generated from a GPT2 model fine-tuned on lyrics dataset of size 30000, containing 3 different genre classes.

Bibliography

Alammar, J. (2018). *The Illustrated Transformer.* [last accessed on 01/2019].

Alammar, J. (2019). *The Illustrated GPT2.* [last accessed on 10/2019].

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb), pp. 1137–1155.

Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, pp. 135–146.

Cavnar, W. B., Trenkle, J. M. et al. (1994). N-gram-based text categorization. *Ann Arbor Mi*, 48113(2), pp. 161–175.

Cheng, J., Dong, L. and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, vol. 1, Austin, Texas: Association for Computational Linguistics, pp. 551–561.

Colah (2015). *Understanding LSTM Networks.* [Last accessed on 01/2019].

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, vol. 1, pp. 3079–3087.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019). Bert: Pretraining of deep bidirectional transformers for language understanding. *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 1, pp. 4171–4186.

Gu, J., Liu, Q. and Cho, K. (2019). Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7, pp. 661–676.

Hagiwara, M. (2018). *Improving a Sentiment Analyzer using ELMo.* [last accessed on 01/2019].

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), pp. 1735–1780.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Holtzman, A., Buys, J., Du, L., Forbes, M. and Choi, Y. (2020). The curious case of neural text degeneration. In *International Conference on Learning Representations.*

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 328–339.

Kim, Y., Jernite, Y., Sontag, D. and Rush, A. M. (2016). Character-aware neural language models. In *Association for the Advancement of Artificial Intelligence*, vol. 1, pp. 2741–2749.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Conference on Neural Information Processing Systems*, vol. 1, pp. 3111–3119.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013b). Efficient estimation of word representations in vector space. *International Conference on Learning Representations.*

Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. pp. 311–318.

Parascandolo, G., Huttunen, H. and Virtanen, T. (2016). Recurrent neural networks for polyphonic sound event detection in real life recordings.

In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 6440–6444.

Parikh, A. P., Täckström, O., Das, D. and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, vol. 1, Austin, Texas: Association for Computational Linguistics, pp. 2249–2255.

Patterson, J. and Gibson, A. (2017). *Deep Learning: A Practitioner's Approach.* "O'Reilly Media, Inc.".

Paulus, R., Xiong, C. and Socher, R. (2018). A deep reinforced model for abstractive summarization. *International Conference on Learning Representations.*

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, vol. 1, New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237.

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2 amazonaws com/openai-assets/research-covers/languageunsupervised/language understanding paper pdf.*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Rajpurkar, P., Zhang, J., Lopyrev, K. and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016*

Conference on Empirical Methods in Natural Language Processing, Austin, Texas: Association for Computational Linguistics, pp. 2383–2392.

Sennrich, R., Haddow, B. and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725.

Stern, M., Chan, W., Kiros, J. and Uszkoreit, J. (2019). Insertion transformer: Flexible sequence generation via insertion operations. Long Beach, California, USA: PMLR, pp. 5976–5985.

Sutskever, I., Vinyals, O. and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, vol. 1, pp. 3104–3112.

Ullah, A., Ahmad, J., Muhammad, K., Sajjad, M. and Baik, S. W. (2018). Action recognition in video sequences using deep bi-directional lstm with cnn features. *Institute of Electrical and Electronics Engineers Access*, 6, pp. 1155–1166.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, vol. 1, pp. 5998–6008.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., Kaiser, L., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N. and Uszkoreit, J. (2018). Tensor2tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, vol. 1, Association for Machine Translation in the Americas, pp. 193–199.

Wang, A. and Cho, K. (2019). Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:190204094*, 1, pp. 30–36.

Welleck, S., Brantley, K., Iii, H. D. and Cho, K. (2019). Non-monotonic sequential text generation. Long Beach, California, USA: PMLR, pp. 6716–6726.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:160908144*.

Yu, L., Zhang, W., Wang, J. and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient.

Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F. and Choi, Y. (2019). Defending against neural fake news. In *Advances in Neural Information Processing Systems*, vol. 1, pp. 9054–9065.

Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J. and Yu, Y. (2018). Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, ACM, pp. 1097–1100.