Data Augmentation for Text Generation From Structured Data

by

Rylen Sampson

B.Sc., Queen's University, 2021

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Rylen Sampson**
Degree:	**Master of Science**
Thesis title:	**Data Augmentation for Text Generation From Structured Data**
Committee:	**Chair:** Angelica Lim Assistant Professor, Computing Science
	Fred Popowich Supervisor Professor, Computing Science
	Anoop Sarkar Committee Member Professor, Computing Science
	Angel Chang Examiner Assistant Professor, Computing Science

Abstract

Data-to-text generation, a subfield of natural language generation, increases the usability of structured data and knowledge bases. However, data-to-text generation datasets are not readily available in most domains and those that exist are arduously small. One solution is to include more data, though usually not a straightforward option. Alternatively, data augmentation consists of strategies which artificially enlarge the training data by incorporating slightly varied copies of the original data in order to diversify a dataset that is otherwise lacking.

This work investigates augmentation as a remedy for training data-to-text generation models on small datasets. Natural language generation metrics are used to assess the quality of the generated text with and without augmentation. Experiments demonstrated that, with augmentation, models achieved equal performance despite the generated text exhibiting different properties. This suggests that data augmentation can be a useful step in training data-to-text generation models with limited data.

Keywords: Data-to-text Generation; Data Augmentation; Natural Language Generation; Language Models

Acknowledgements

There are numerous people I would like to show my appreciation to for helping me in getting to this stage. First, I would like to thank Dr. Fred Popowich for the opportunity to pursue a project of my own interest, the help in bringing this work together, and the additional exposure to problem solving with real-world data.

Foremost, I want to thank my parents for the continuous support over two degrees and the motivation to always put my best foot forward. I will always be thankful for the stress free environment that welcomes me when I return home. I would also like thank my sister for her support as well. The steady stream of puppy pictures never failed to put a smile on my face while I was hard at work.

I would also like to express my gratitude to an amazing group of friends whose support is never unnoticed and made breaks from long periods of work doubly as enjoyable.

Finally, a thank you is in order for Anoop Sarkar and Angel Chang for their insights and time in revising this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis investigates the applicability of existing data augmentation strategies for data-to-text generation and shares results surrounding this idea. I begin by reviewing key concepts pertaining to data-to-text generation, the role of data augmentation in other tasks such as text classification and computer vision (CV), and evaluating text generation. Next, I implement three augmentation strategies that are used to address training data-to-text generation models on small or inadequate datasets. For each strategy I share the implementation details as well as any observed behaviours the model exhibits after being trained on augmented data. The remainder of the thesis assesses the capabilities of two models for data-to-text generation before evaluating the better of the two when different augmentation strategies are incorporated into the training procedure. I discuss any improvements or lack thereof when data augmentation is performed and possible reasons that may shed light on these results. Lastly, I summarize the scope of this thesis and what was learnt prior to suggesting avenues for future work.

The first chapter is an informal introduction to data-to-text generation and motivates the incorporation of data augmentation. Furthermore, I present a collection of hypotheses, contributions, and the overall plan of the thesis.

1.1 Motivation

Everyday, billions of data points are generated and stored for future use. Many methods exist for conducting analysis such as regression, classification, or visualization, but even then an individual will have difficulty providing a succinct summary of the data's content. Weather and sports reporting, electric vehicle charging, and maritime transport are all examples of domains that routinely produce batches of data where summary statistics are insufficient. In these domains, the **why** and **how** are more valuable. For example, a captain of a maritime transport vessel is guaranteed to find more worth in a brief summary of fuel usage from the day's travel and any possible concerns than a list of fuel usage statistics for the day with no accompanying context.

1.2 Problem Statement

The ability to communicate and summarize data from different intervals or distinct situations is sought after as it facilitates domain understanding and decision making. Manually summarizing data of this nature is a repetitive and time-consuming task that also requires domain knowledge. In 2020 when the pandemic was just beginning, the public saw cities quickly switch from written briefs to dashboards for this very reason despite the former being more easily absorbed by non-professionals. Data-to-text generation is the task of producing coherent natural language given non-linguistic input and is an approach for automating the aforementioned process [27]. Traditionally, data-to-text generation models were pipelines that produced text following a series of predetermined transformations [82]. While this guarantees a consistency in the *generated text*, it does not allow for the text to be dynamically changed conditional on the contents *source data*. This results in poor generalization to new domains or even unseen data within the same domain.

In recent years, neural models have shown success on various natural language tasks [30, 47, 7]. These models use hundreds of millions of parameters to encode linguistic information about the data [84]. This high degree of freedom implies a neural model is highly variant in what it can learn and is a leading reason for their popularity. Albeit, high model variance is accompanied by a predisposition to overfitting - otherwise known as the bias-variance trade off [42, 62]. In particular, models with high variance that are trained on small datasets have a tendency to overfit to learned patterns which do not generalize well.

Procuring a sufficiently large dataset for training a data-to-text generation model is challenging as data structures such as tables, databases, or time-series are not generally accompanied by pertinent text in abundance [87]. Example datasets include: WebNLG [26] and E2E [64] which are used in this work, DART [61], ToTTo [67], and RotoWire [101]. WebNLG and E2E share a niche of structured data that is descriptive and relational with the *generated text* conveying this. Commonalities between the two datasets mean conclusions drawn by the models and augmentation strategies used are more thoroughly investigated. As such, this is a reason for the specific focus on these two datasets through this work. Existing datasets are small in comparison to other natural language tasks - usually a magnitude smaller with only tens of thousands instead of hundreds of thousands or even millions of examples. Furthermore, the data-text pairs are limited in linguistic diversity for features such as vocabulary, syntax, and style as they typically follow a loose template for the content included and the terminology used. Given how data scarce this setting is, data-to-text generation models are quick to overfit to the few training examples their exposed to. A classic solution to overfitting is regularization which requires careful hyperparameter tuning, motivating the inclusion of data augmentation instead [9, 34]. Data augmentation is informally defined as strategies that artificially enlarge the size of the dataset by producing slightly modified copies of the original data. Modifying the original training set for data-to-text generation is nontrivial as the semantics of the *target text* are connected to the *source data* and the syntax constrained by dependencies within the *source data*.

1.3 Hypothesis

I hypothesized that the general linguistic information learnt during the pre-training of a language model would result in improved performance on numerous evaluation metrics. Furthermore, when this language model is extended by the inclusion of data augmentation more of an improvement would be observed. Finally, I hypothesized that a language model fine-tuned on augmented data would require less of the original data to be trained as well as the language model without augmentation that was fine-tuned on the full amount of training data.

1.4 Contributions

This work has three primary contributions, although supplementary observations are made in Chapter 4. First, I provide a thorough review of data-to-text generation and the relevant concepts before providing results for the task using two models and two datasets - WebNLG and E2E. Next, I provide results for this task when three data augmentation strategies I implemented are incorporated into the training procedure. To my knowledge these strategies have not been studied for data-to-text generation prior to this work. As well, I compare the ability of each data augmentation strategy for mitigating the loss of information when the amount of training data is drastically reduced.

The methods used in this work are not inherently novel as many have been researched in adjacent fields, but their application to data-to-text generation is. In fact, data augmentation has been extensively covered in other areas which inspires this work and motivates how we evaluate methods for this growing intersection with data-to-text generation. Two of the three augmentation strategies are unexplored for the datasets selected, providing novel observations as well as adding further evidence to existing ones. Additionally, pre-trained language models have made new approaches to data augmentation available - some of which are explored here. Large language models (LLMs) such as the one used here are gaining traction with data-to-text generation and this work elaborates on their applicability. Naturally, LLMs perform well in scenarios where there is limited data but by incorporating data augmentation this success is taken one step further. Novel in regard to unexplored combinations, this work provides a strong basis for future research that is developed from these avenues.

1.5 Organization of Thesis

Chapter 2 begins with a review of natural language generation (NLG) and subsequently data-to-text generation, data augmentation, evaluating text generation, and lastly related work such as motivating examples of data augmentation in CV or other natural language tasks.

Chapter 3 introduces the two datasets that are used throughout the experiments, the implementation of the data augmentation strategies, and the methods for evaluating the experiments.

Chapter 4 presents the results of data-to-text generation with two different models, the inclusion of three data augmentation strategies, and the effect augmentation has when the amount of training data is reduced. Furthermore, these results are discussed in terms of both improvements and shortcomings as well as any possible explanations for confounding observations.

Chapter 5 concludes the thesis through a summary of observations, their significance, and any limitations. Finally, the thesis is wrapped up by touching upon fruitful avenues for future work uncovered in this research.

Chapter 2

Background

2.1 Data-to-text Generation

Natural language generation is generally defined as the construction of models which produce coherent and understandable text given input data of some form. Data-to-text generation builds off this definition, being the task of producing text given non-linguistic input [27]. Although the output of data-to-text generation is always human understandable text, the input is less consistent with examples being tabular data, meaning representations, or structured knowledge bases. Throughout this thesis the ground-truth input and output used for training a data-to-text generation model is referred to as the *source data* and *target text* respectively whereas the text produced by a model during inference in regards to an experiment or test data is denoted *generated text*.

Applications of data-to-text generation include: communicating weather forecasts [29, 83, 95, 79], reporting on financial data [71], wildlife blogging [89], summarizing the meaning behind sensor data [57], and narrating wildlife tracking based on satellite data [72]. While the *source data* format for each of these applications varies and the accompanying text differs in terms of quality and complexity, they all strive to achieve the same goal - imparting information to the user. Ensuring this information is actionable requires a particular quality for the *generated text* which is characterized as fluent, syntactically correct, and faithful to the content of the *source data*. One approach that guarantees a certain level of quality is pipeline models where a data structure is progressively transformed to a standardized response by a series of modular rules.

2.1.1 Pipeline models

Initially, data-to-text generation was conducted using a pipeline architecture that executed the following sub-operations: *content selection*, *content planning*, and *surface realization* [82]. In practical terms that is: *what to say*, *how to say it*, and then *actually saying it*. Figure 2.1 displays the steps and byproduct of each step for a general pipeline architecture. While pipeline architectures provide efficient and robust generation [23], they also require

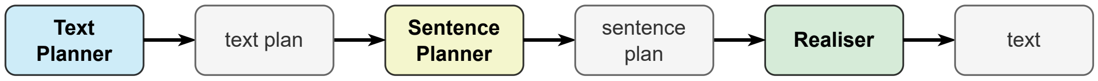

Figure 2.1: General three-step pipeline architecture for data-to-text generation. Coloured segments correspond to the modules or steps while the grey portions pertain to the corresponding output. Figure is from [81, 27] and updated for visibility.

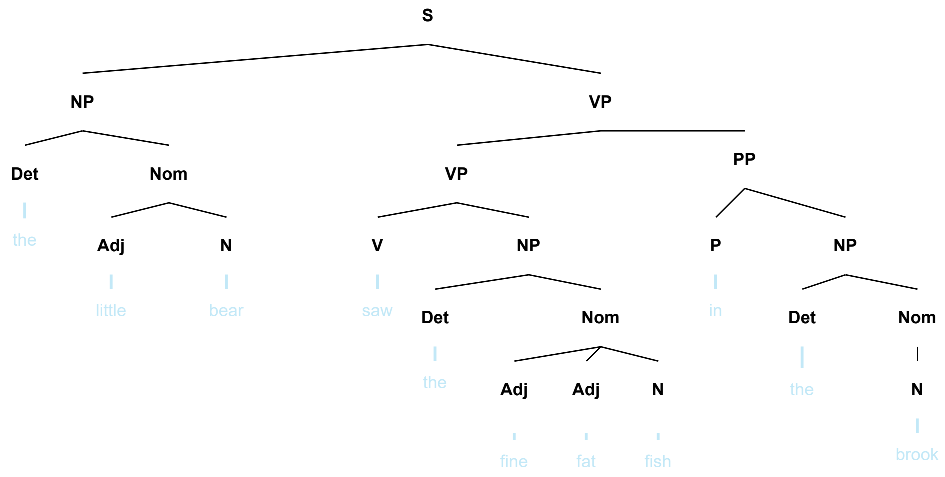

Figure 2.2: A syntax tree for a sample sequence of text that demonstrates the possibility for long-range dependencies that may be difficult for some models to handle.

numerous design choices such as enumerating *target text* possibilities and the rules that decide what template the data is realized as. For small domains where there is minimal variation, these limiting factors are the reason pipeline architectures were popular for data-to-text generation - in short they are cheap and efficient. Now, with the focus of natural language tasks on generalization, models exhibiting linguistic flexibility are preferred.

2.1.2 Neural models

With the explosion of data, both in terms of quantity and quality, small confined datasets are no longer the norm and therefore models capable of learning abstract representations are desirable. The progression of machine learning models has seen a rapid change in perspective regarding neural networks [1] being considered promising to becoming the de facto choice. Now, neural networks regularly achieve state-of-the-art results in various natural language tasks such as text classification, neural machine translation, text summarization, and question-answering [31]. Even neural models require numeric features to learn from and since text data cannot be passed as is, representations of the text have to be derived in order to be usable. For early applications, token occurrence and frequency features such as bag-of-words (BoW) [106] or term frequency-inverse document frequency (TF-IDF) [78] were the predominant text representations. Shortcomings of these text representations include but are not limited to - sparse features, limited to no support for out-of-vocabulary tokens, and important words being assigned insignificant values while stop words are found to be significant. Most notably though is that BoW and TF-IDF have no way of understanding the role of a token for the semantics of a phrase, each token is handled independently. The improvement of text representations which grow more abstract, has been one of the most prominent advances for natural language tasks. In this case, an abstract representation implies that it would have no meaning to a human at a glance compared to BoW where the resulting representation can be understood in terms of token occurrence. Abstract representations are characterized as dense, low-dimensional, and distributed vectors of scalar values. For now we will assume the importance of abstract representations in both data-to-text generation and data augmentation. In Section 2.2.1 this concept will be further defined and explored.

The recurrent neural network (RNN) [85] is a neural network architecture that has demonstrated success on benchmarks for numerous tasks. Their recurrent nature is designed to handle input sequences of variable length which is regularly the case for text data. As the sequence length grows, so does the instability of the network due to passes through the network having to retain all the information in a sole forward-moving signal [68] - this is deftly labeled the *vanishing or exploding gradient problem* [35]. RNNs enhanced with long short-term memory (LSTM) [36] or gated recurrent units (GRU) [17] include mechanisms for early network information to be carried forward or forgotten, skipping steps. Still, sequences of text tend to carry long-range dependencies that are challenging for RNN variations to model as seen in the syntax tree of Figure 2.2. Regardless, the recurrent and sequential nature of LSTMs and GRUs have spawned further model architectures that are capable of generating a sequence given an input sequence. I.e., generating text from structured data.

2.1.3 Encoder-decoder architectures

The encoder-decoder architecture has become integral for solving sequence-to-sequence tasks where the goal is to translate a sequence of some length into a potentially different length sequence [93]. Traditionally, this architecture is implemented using an RNN for both the encoder and decoder networks. The first RNN encodes the input sequence into a vector representation that is subsequently passed to the decoder to generate the resulting sequence [16]. Encoder-decoder models lend themselves well to the task of neural machine translation (NMT) as the task can easily be framed as translating a variable-length sequence of text from a *source language* to a variable-length sequence of text in a *target language* [15]. Fortunately, this perspective for NMT can be extended rather easily to data-to-text generation where the problem formulation is translating a variable-length sequence of *source data* to a variable-length sequence of text [27]. A main drawback of the encoder-decoder architecture is the bottleneck posed by the internal representation that is shared between the two networks as highlighted in Figure 2.3. First off, attempting to consolidate the entire sequence into a vector is limiting and second, there is no mechanism for considering parts of

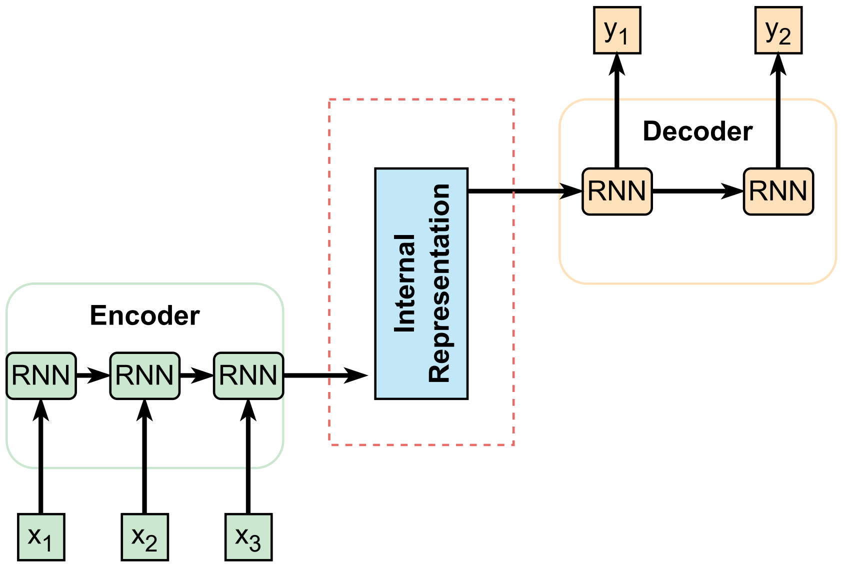

Figure 2.3: A general encoder-decoder architecture based on RNNs. The labeled encoder is highlighted in green and the decoder in orange. Highlighted in a dotted red box is the aspect of this architecture that is often a bottleneck in sequence-to-sequence tasks.

the encoder-fed sequence that needs to be re-aligned for the decoder. Attention is a mechanism incorporated into encoder-decoders that searches for parts of an input sequence that are relevant to generating the next token in the output sequence, removing the challenge of encoding an entire sequence into a fixed-length vector [5]. By giving the decoder a way to choose parts of the input sequence to pay attention to, it can weigh these for easier retrieval when generating the output sequence. This entirely removes the need for the model to learn the alignment between the input and output sequences on its own, allowing it to focus on learning the sequence-to-sequence mapping.

The RNN architecture has demonstrated success on various natural language tasks including data-to-text generation and it remains a popular architecture in research [23]. The authors of [74] construct an entity-centric model that handles entities separately when generating text with an LSTM decoder. Additionally, content plans or content ordering accommodates the recurrent nature of an RNN architecture by laying out dependencies to be filled in prior to text generation. Content plans have been constructed at the sentence, paragraph, and document level to improve generating text with RNN decoders [73, 76, 75]. Furthermore, past work on WebNLG has used an LSTM decoder to condition on both a graph encoder and LSTM encoder to improve the text generation quality [107].

Despite the improvement garnered by attention-mechanisms, encoder-decoders built upon RNNs such as those in the aforementioned work remain hindered by their sequential approach to modeling. Any given state within an RNN only ever depends on the last, forcing a strain on the model to remember necessary information about early sequence items over long-ranges as mentioned in the last subsection. Summarizing, the consequences

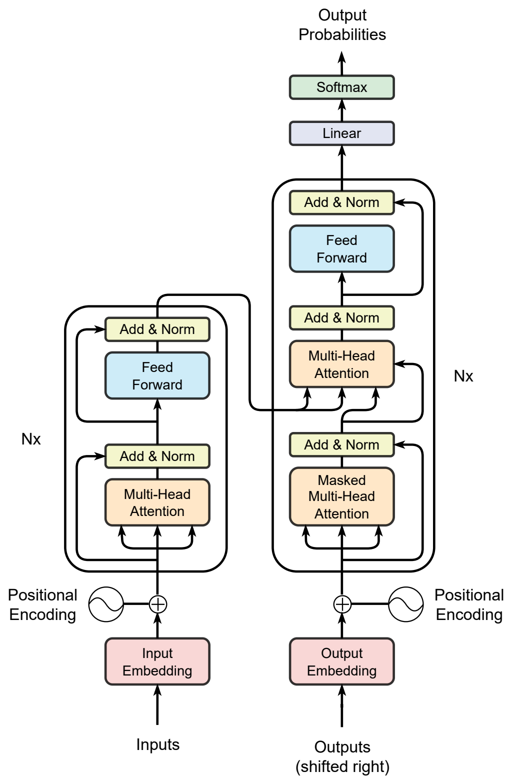

Figure 2.4: The Transformer model introduced in [97]. The left-hand stack is the encoder and the right-hand the decoder. Nx refers to the number of encoder- or decoder-layers the Transformer is initialized with. Transformers take two sequences as input which are embedded into $\mathbb{R}^n$ as input. Figure originally published in [97]

of sequential processing are: that the model cannot be trained in parallel; inference times quickly grow, especially when decoding long sequences; and as only the previous token is being directly taken into account, surrounding tokens or any semblance of sentence structure in the case of text cannot contribute to what is being learnt. In comparison, a model with a parallel architecture and an attention-mechanism that can view the entire encoder-fed sequence could possibly improve on the time it takes to train and its sequence-to-sequence capabilities. In the following subsection, one such model is discussed.

2.1.4 Language models

While language model is a catch-all varying from simple n-gram models to neural networks trained with deep learning approaches, a large language model (LLM) generally pertains to the latter. LLMs have been successful in numerous natural language tasks due to their ability to obtain a general understanding of language and how to generate it [108, 56]. Many LLMs have been proposed in recent years such as the ones used within this work [18, 77, 45, 102]. Although the foundations of these LLMs are similar, their implementation is what sets them apart from one another. This section covers that shared foundation and its relevancy to data-to-text generation. For the remainder of this thesis I will simply use language models in lieu of LLMs to avoid crowding the paper with acronyms.

Many language models leverage the popular architecture the Transformer as it addresses the issues discussed for RNNs [97]. Figure 2.4 shows the architecture diagram for the Transformer and illustrates the following differences between it and an RNN. First, rather than compress an entire sequence into a single vector representation to pass to the decoder, the transformer provides each token with its own dense, low-dimensional, and distributed vector representation. As well, feed-forward neural networks replace the RNNs for processing enabling parallelization. Lastly, a new attention mechanism called self-attention is introduced that is able to handle long-range dependencies within a sequence. It does this by taking into account what is at each position of a sequence to then compute an overall representation for the sequence. This means even dependencies between the first and last token contribute to the representation. I.e., in the sequence "the woman is going away for a long time and she will be missed." self-attention creates an awareness the "she" is in reference to the "woman". But, like an RNN, the Transformer is a suitable architecture for sequence-to-sequence tasks.

Surveys have shown the viability of the Transformer for data-to-text generation and the improvements it offers over RNN-based models [23, 49]. The self-attention mechanism of Transformers allows more to be learnt with less training, opening the possibility for detailed training procedures. Training can differ by the inclusion of data augmentation, curriculum learning, and the use of external data sources as seen in [58]. Similarly, the Transformer can be trained with additional learning objectives to improve aspects of data-to-text generation such as content selection [32] Commonly in data-to-text generation, the *source data* is linearized in some manner that way it is easily handled by encoder-decoder Transformer architectures requiring a sequence. A hierarchical attention mechanism has been explored with the Transformer that first encodes the *source data* based on its constituent units then on its overall structure, removing the loss of information that resides in the structure of the *source data* [80]. Most recently, Transformers have led the way in zero- or few-shot settings for data-to-text generation [39, 14, 11].

Pre-training is the belief that for a *target task*, initializing model weights randomly is subpar to utilizing ones from a model that has already been trained on a similar *source task* [21]. Although random model weights can be adjusted for a particular task, it may be beneficial in both time and results to instead tune model weights from a similar domain or task. Language models are generally not trained on supervised tasks, rather they are trained on a large unlabelled corpora using unsupervised or self-supervised learning to capture model weights for generalized linguistic features (*i.e. syntax and semantics*). While pre-training does not directly solve data-to-text generation, by first learning linguistic information the model is then able to focus more on learning to generate text from structured data. Models required to learn both how to generate linguistically correct text and remain faithful to the content of the *source data* are more likely to perform worse, require more data, more time, and overfit to select examples.

There exists numerous pre-training approaches, listed below are those commonly used or used by the language models in this work.

- **Masked Language Modeling** (MLM): The objective here is to learn the context of a token given the remainder of the sequence. A percentage of the tokens in the original sequence are replaced with a generic token, usually [MASK], effectively hiding the original token from the model. Then, the model tries to reconstruct the original sequence by replacing the generic tokens with the token it hid. This pre-training approach is primarily used by language models consisting of only an encoder transformer and is popular for tasks such as text classification or sentiment analysis.

- **Sentence Order Prediction**: Using self-supervised learning, the model learns to predict the order for a list of sentences. Given a scrambled list, the model reconstructs the original ordering.

- **Replaced Token Detection**: Rather than replacing tokens with a mask, they are replaced with another random token from the vocabulary. The model learns to restore the original tokens, but is not told what tokens were swapped.

- **Contrastive Learning**: The model learns to discern between similar and dissimilar data. Pairs of data are assembled based on similarity to train the model embeddings such that similar pairs are close in the embedding space or far away if they are dissimilar.

- **Span Masking**: Language models pre-trained for natural language generation (NLG) tasks use an alternative approach where instead of a single token being masked, a span of tokens in the sequence is. The model then tries to reconstruct the span in one go. This is useful in downstream tasks where the model can benefit from identifying spans of relevant information in a document. By pre-training the model to predict spans consisting of named entities, it learns to identify and extract these from unseen input. This is just one of many reasons the language model used in this work was selected.

2.2 Natural Language Concepts

The following section will review and discuss the role of relevant natural language processing (NLP) topics for data-to-text generation. While the topics themselves are not novel, they contribute to the novelty of this work and thus their contribution solidly understood. These topics include contextual embeddings, transfer learning, and curriculum learning.

2.2.1 Contextual embeddings

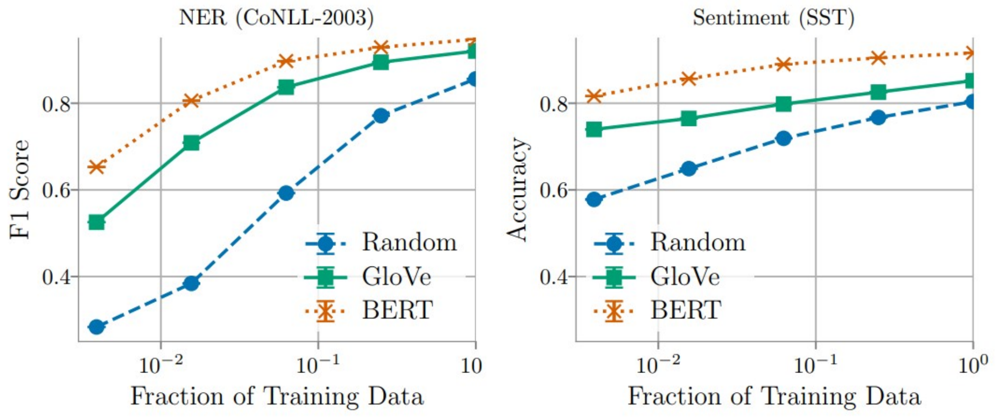

Figure 2.5: Results for contextual embeddings out-performing non-contextual embeddings with varying amounts of data on named-entity recognition (left) and sentiment analysis (right). Figure originally published in [2].

Machines are not capable of handling text as is, requiring it to be transformed into a numeric format before being passed to a model. For example, given a vocabulary of size four the following text sequences are represented as integer vectors:

$$\begin{aligned}\mathcal{V} =& \{apple, banana, grape, orange\}\\ s_1 =& [apple\ orange\ grape]\\ =& [1\ 0\ 1\ 1]\\ s_2 =& [grape\ kiwi\ chestnut\ peach]\\ =& [0\ 0\ 1\ 0]\\ s_3 =& [grape\ pear\ grape\ apple]\\ =& [1\ 0\ 2\ 0]\end{aligned} \tag{2.1}$$

Representing text in this manner is a rudimentary approach based on the number of times a word occurs. Although efficient to implement, it suffers from sparsity and memory issues as the vocabulary $\mathcal{V}$ grows in size. While this captures the composition of a sequence, as the vocabulary scales so does the dimensionality resulting in convoluted or sparse representations. Alternatively, to circumvent an abundance of 0's, the occurrence of tokens can be replaced with frequencies. Even so, these are discrete text representations as they do not exist in $\mathbb{R}^n$, that is these vectors can only take on a fixed set of values. Furthermore, the representations discussed above are global representations meaning each token can only be considered in a single manner. This is a major drawback of global text representations as it

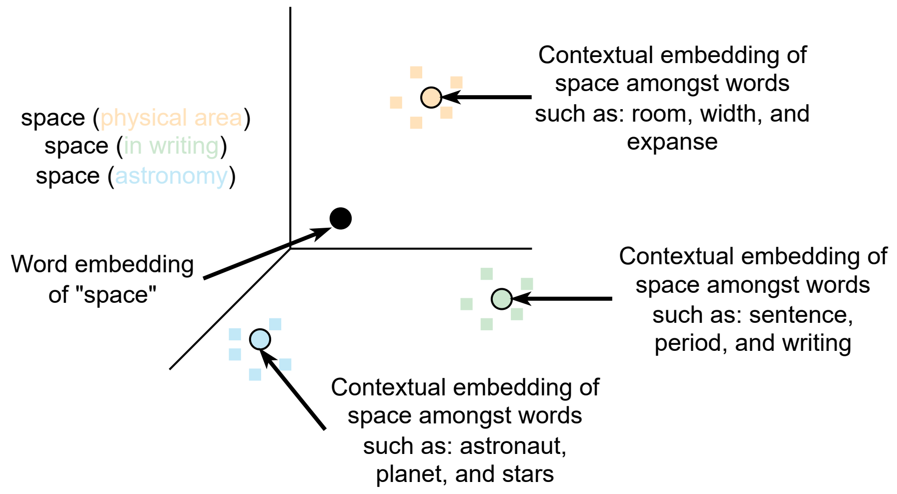
space (physical area)
space (in writing)
space (astronomy)
Word embedding of "space"
Contextual embedding of space amongst words such as: room, width, and expanse
Contextual embedding of space amongst words such as: sentence, period, and writing
Contextual embedding of space amongst words such as: astronaut, planet, and stars

Figure 2.6: An example demonstrating that contextual embeddings provide multiple representations for the same word whereas other approaches only provide a single representation. Here, "space" pertains to a physical area or room, a space in a sentence, and space in reference to what is beyond the planet Earth. In each case, additional words are provided to illustrate the context.

limits a token to a single context when it likely has various, e.g. "The mouse ate the cheese" versus "I moved my computer mouse".

Rather than a rigid representation for a token in the vocabulary, its representation is a function of the all the other tokens in the sequence [70]. In the above example, suppose $r_{token}(sequence)$ is the function for obtaining a token's representation and s_1 and s_2 are the former and latter sequences, then:

$$r_{mouse}(s_1) \neq r_{mouse}(s_2) \tag{2.2}$$

Function computed representations are frequently referred to as contextual embeddings and are characterized by dense, low-dimensional, and distributed vectors as seen below:

$$r_{mouse}(s_1) = \begin{bmatrix} 0.7543 \\ -0.1249 \\ -0.6832 \\ ... \\ 0.4141 \end{bmatrix} \not\approx \begin{bmatrix} -0.2543 \\ 0.9111 \\ 0.2169 \\ ... \\ -0.8765 \end{bmatrix} = r_{mouse}(s_2) \tag{2.3}$$

In addition to multiple possible representations, they are now continuous and exist in $\mathbb{R}^n$ - Figure 2.6 illustrates this. This means the two token representations are comparable using the likes of cosine similarity, such that the words "space" and "planet" would have a high similarity score in the following sentences s_3 = "Space and galaxies interest me" s_4 = "I love

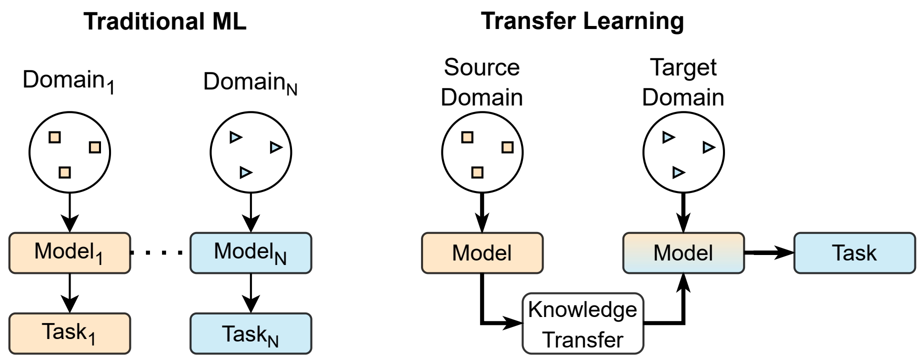

Figure 2.7: Transfer learning visually depicted. Traditional machine learning (left) is described as continuously training new models for novel domains. Transfer learning (right) shows how other domains offer beneficial knowledge for novel domains and that a new model does not always need to be trained from scratch.

viewing planets through my telescope".

$$r_{space}(s_3) = \begin{bmatrix} 0.7543 \\ -0.1249 \\ -0.6832 \\ \dots \\ 0.4141 \end{bmatrix} \sim \begin{bmatrix} 0.6989 \\ -0.1311 \\ -0.6423 \\ \dots \\ 0.5129 \end{bmatrix} = r_{planet}(s_4) \tag{2.4}$$

Two readily-available continuous token representations are word2vec and Glove which were instantiated using a feed forward neural network and matrix factorization respectively [54, 69]. Their main shortcoming is that their vocabularies are fixed due to them being pre-trained. Another approach to learning continuous representations is from language modeling which has been thoroughly covered in previous sections. Language modelling instills semantic and syntactic knowledge into the contextual embeddings through the self-attention mechanism. As these contextual embeddings can easily be extracted from a language model, they are suitable to be transferred to downstream tasks to be used as features or further fine-tuned [18]. The concept of transfer learning is discussed in the next subsection.

2.2.2 Transfer learning

Transfer learning is defined as the application of knowledge learnt from a *source task* to a related, but unexplored *target task* [65]. Rather than repeatedly training new models from scratch, a model's knowledge is repurposed for a new task as depicted in Figure 2.7 The success of this ideology has been demonstrated for: text classification [19], spam filtering [51], and digit recognition [8]. In computer vision (CV), a model trained to detect dogs in an image may provide strong initial results for detecting cats in images before further fine-tuning. Transfer learning is omnipresent in CV since the majority of the tasks require knowledge of low- and mid-level features such as edges, shapes, foreground and background.

Similarly for NLG, regardless of the generated text, there is guaranteed to be transferable knowledge between tasks as each requires knowledge of syntax, semantics, pragmatics, and/or phonology. As discussed in the previous section, contextual embeddings are easily shared across tasks and are considered a form of transfer learning as their knowledge is directly usable as features or as a starting point for fine-tuning. This has motivated numerous training regimes for language models in an attempt to learn general linguistic features aside from their supervised tasks [46, 50, 77]. Across these supervised tasks, the internal architecture remains the same with only the output layer changing depending on the task. When fine-tuning downstream, it must be decided whether all the model weights are adjustable or just those in the output layer. If the pre-trained model weights are not frozen, then fine-tuning runs the risk of inflicting mass forgetting and the model's *source task* knowledge is lost. Whereas when the weights are frozen the model has a more difficult time learning the *target task*. The balance between forgetting too much and the speed at which the *target task* is learnt remains a common challenge. The ease at which pre-trained language models are adapted for supervised tasks has resulted in a collection of these models being bundled in the HuggingFace library [102].

In this work, transfer learning will help alleviate the challenge of a small dataset. For example, the weights of a pre-trained language model are used initially to provide a better starting point which in turn reduces the training time. Furthermore, leveraging contextual embeddings from a pre-trained language model are likely better than those that would be obtained from an insufficient amount of data. Transfer learning offers the opportunity to obtain better performance that would not be available otherwise.

2.2.3 Curriculum learning

Language models have demonstrated their ability to generate coherent and contextually relevant text, however training such models is a challenging task due to the complexity and vast amount of data. Curriculum learning, a machine learning paradigm inspired by human experience, suggests a promising approach to improve the training of models. This section will explore the concepts and fundamentals of curriculum learning before discussing the application to language models in particular.

Curriculum learning is a training strategy that carefully considers the order in which training samples are presented to the model. The goal is that an intricately designed order causes the model to more rapidly converge on generalized model parameters. Most simply, the model can begin with simpler or easier examples before progressing in difficulty. This technique imitates how humans learn, beginning small on simple telling examples before approaching complex nuanced examples. The main objective of curriculum learning is to quicken learning and make it more effective by giving the model a thought out learning schedule to build off of. By gradually including more complex examples, the model can hypothetically build on previously learned concepts and more appropriately adjust model weights with small yet concise steps. Whereas randomly shuffling the data results in the model wildly jumping between drastically different training examples that can confuse the learning process. Curriculum learning, like other methods used in this thesis, has shown proven results in other fields such as reinforcement learning, computer vision, and most importantly natural language processing which we extend to data-to-text generation.

One approach is task-based curriculum learning. Here, the training examples are organized based on their difficulty for the given task. In the case of data-to-text generation, the curriculum would start the model off on examples with simple *source data* or shorter *target text* before gradually introducing examples with more intricate connections, syntactic structures, semantic relations, or longer text. Following a task-based curriculum, the model can practice and master on simpler *source data* and *target text* before attempting to address more challenging examples.

In comparison, data-related curriculum learning focuses on the ordering of the training data rather than the task difficulty which can be hard to quantify at times. This approach sorts the training data based on data properties such as frequency, diversity, or similarity. In the case of data-to-text generation, it might be most beneficial to begin with the most frequently occurring *source data* relation before introducing more rare relations. By exposing the model to a wide range of data gradually and pre-meditated, it has a better chance to effectively capture the underlying patterns and generalize better to unseen examples and rarer data.

Curriculum learning offers several benefits for training language models. First, it can improve learning efficiency by structuring the learning schedule. Beginning with simple examples, the model can quickly grasp basic patterns in the data and build upon them. The pre-training of language models accomplishes a similar goal of learning linguistic patterns, so for this work we focus on curriculum learning to improve task knowledge. Secondly, curriculum learning can improve generalization of the model by gradually introducing more complex data. Providing an initial basis of easy-to-learn examples will support the model in later stages of training. Lastly, common issues of language modeling can be ameliorated with curriculum learning such as catastrophic forgetting. When randomly shuffled, models may jump between drastically different examples and easily forgetting what they just learned. A structured learning schedule encourages learning as well as information retention.

This is not to say that curriculum learning is not without its challenges. Structuring effective learning schedules necessitates careful consideration and knowledge of task-specific characteristics, the availability of simple and complex data, and the quality of data. Determining whether to do task- or data-based curriculum learning and the resulting ordering is not trivial.

2.3 Data Augmentation

As mentioned, data augmentation strategies enhance a dataset in terms of size and quality through slight modifications of the original data. Transformations range from simple to complex with one of the most used strategies being to draw random noise from a standard normal distribution and adding this to the existing data in order to reduce overfitting. Given an overarching goal of data augmentation is to reduce overfitting, it tracks that adding random noise has been proven to be synonymous with Tikhonov regularization [9] Although augmentation strategies have grown in complexity, many stem from simple ideas and are adapted for use. While there exists other methods of reducing overfitting, none address the heart of the problem like data augmentation - the dataset. I briefly introduce these methods for the knowledge of the reader as reference to other avenues.

- Dropout [92] is a technique where the values of randomly chosen neurons during training are zeroed. Rather than learn from a subset of heavily weighted neurons, this forces the network to learn more robust activation patterns.

- Batch normalization is another regularization method that normalizes the input between layers to accommodate for the distributional shift [37]. For each batch, the mean is subtracted and it is divided by the standard deviation resulting in a zero mean and unit variance.

- Transfer learning has already been extensively discussed [65]. In short, it is effective for reducing overfitting as models leveraging already trained weights are less susceptible to falling into poor local solutions. These weights are obtained from a model trained on a large generalizable dataset such as the BookCorpus dataset [111]. For this dataset, the model learns low-level features pertaining to story-like language.

- Pre-training was also touched upon in previous topics [20]. Like transfer learning, it involves exposing a model to a large dataset to learn general features. In pre-training, weights are initialized but there remains flexibility in the architecture design unlike transfer learning.

- One- and Zero-shot learning algorithms pose alternative approaches for learning from limited data. An example of one-shot learning uses distance functions to determine if unseen data is similar enough to the few training instances [98]. While zero-shot learning is more extreme, it uses the same concept of identifying classes by finding dissimilarity in the data [91].

In the following subsections, I discuss the origins of data augmentation in CV as the concepts are well-formed and have demonstrated success. Following, I explore how these ideas translate to natural language tasks as well as any challenges that arise in doing so.

For the strategies cover, I mention how they address overfitting and preserve the context or given label of the data.

Data augmentation is akin to the imagination of humans. We are able to take a scenario in our mind and modify it to conceptualize a slightly changed scenario. A car in a different colour, pizza with different toppings, or a corner desk versus a straight desk. These are all scenarios where we understand the underlying context through examples with varying features. Furthermore, as humans experience more examples, we learn more general characteristics. This is what we aim to replicate through incorporating data augmentation.

2.3.1 Motivation from other fields

Vision-based machine learning takes, as input, images and performs computations such as classification, object detection, and more. Models are trained to recognize patterns in the data with more complex architectures capturing increasingly intricate patterns. In images, these patterns may refer to a dog to be detected, colours that lead to the image label, or distinguishing between an apple and an orange in the image. The challenge that arises, and this is the case for all machine learning tasks, is when the test data differs from the training data. The apple on the table covered by a shadow is still an apple, despite the model no longer recognizing it due to the unseen pattern imposed by the shadow. This is the challenge data augmentation aims to overcome through the introduction of modified copies to learn fringe cases [88].

A pressing issue in CV is positional biases. If the object to be detected has never appeared in the bottom right corner then when it does, there is a possibility the object goes undetected. A quick and efficient solution for positional bias is geometric transformations which include rotating, flipping, or scaling images. The caveat is that adding these augmented copies into the dataset does not necessarily preserve the labels associated with the images [4]. For example, in digit recognition, rotating and flipping a 2 is now a 5 but if this label change is not caught then the model is now learning on inaccurate data. Another approach is to crop small portions of an image to encourage the model to recognize new characteristics contributing to the overall label. Cropping was critical in the training of AlexNet on the ImageNet dataset [43]. Alternatively, augmentation can target red, green, and blue (RGB) colour channels instead of physical characteristics [103]. Colorspace transformations are challenging as in many cases the coloring contributes to the image's label, in particular for image sentiment analysis [12].

Based off cropping, a less intuitive approach involves cropping numerous images and stitching these portions to form a new amalgamated image to add to the dataset [94]. Seemingly unintuitive, it is suggested the improvement of mixing images occurs in the low-level features learnt from the increase of edges and angles. In the case of label preservation, the label is assigned randomly from one of the images that was cropped. While this may seem to ruin the image-label relation, instead it is possible the model directly learns features from the cropped portion pertaining to the label helping its general understanding of the label. Alternatively, a simple manipulation of the pixels results in noticeable performance boosts. Noise injection adds a sparse matrix of random values to the existing pixels, slightly deteriorating the quality of the image [60]. This results in the model learning more robust features, reducing overfitting, and as the image is hardly affected the label is preserved. Cropping and noise injection have been explored in tandem with random erasing [109]. Random masking covers a portion of an image with a mask of random values, forcing the model to learn other descriptive characteristics. Research has also shown that random erasing pairs well with other augmentation strategies [53].

Lastly, there exists more tailored augmentation strategies that leverage neural models. I list these for the knowledge of the reader, but due to the nature of this work do not elaborate further. Neural models have been used to search for adversarial augmentations that result in misclassifications [59]; generative modeling to unlock unobserved information from a dataset [10]; and transferring style between images [28], among others.

2.3.2 Application in natural language

Unlike image data, augmenting text is not so straightforward due to its non-numeric nature. For example, flipping an image continues to be valid whereas flipping a sentence likely renders it senseless as reading a sentence from end to beginning typically does not have the same context - if any at all. Text augmentation requires equal or more consideration, but can be facilitated by drawing parallels and inspiration from work in CV. First, I discuss data augmentation strategies applied through various natural language tasks before moving on to examples in data-to-text generation.

With text data, akin to noise injection for image data, random noise can be added by applying sparse transformations on the character level. These transformations include: inserting, deleting, or swapping a percentage of the characters within a sentence [100]. While this muddles the sentence, it essentially forces the model into learning how to reconstruct an output given corrupted input. Moving to the word-level, lexical substitution is an augmentation strategy that aims to replace words in the text with their synonyms [41, 63]. In most cases, the synonym retains the intended context but not all synonyms are created equally and there are times this is not true. More elaborate substitution schemes help reduce this risk, although it does not entirely remove it. Similarly, backtranslation is an approach that makes use of changes at the word-level by accepting incorrect translations as augmented data [86]. Like synonym replacement, this can cause issues with the label as errors can lead to different contexts. E.g., "*Je suis fini*" and "*J'ai fini*" have differing meanings in English. Like mixing images, SWITCHOUT [99] and MIXUP [104] are two text augmentation strategies that provide non-obvious improvements. SWITCHOUT also works at the word-level by randomly replacing words with another from the model's vocabulary. The latter has motivated a collection of data augmentation strategies that interpolates the input and output given two or more examples from the original dataset [22]. In NMT, one can use ciphers to generate augmented copies of the training data which will retain distributional features [38].

Recent research has seen various data augmentation strategies employed for data-to-text generation to date. On a dataset consisting of basketball statistics and game summaries [101], stats were slightly perturbed to ensure the overall outcome remained valid while augmented copies [33]. Another approach has been to incorporate external data as additional copies, deriving source data from text using information parsing techniques [58]. Noise has been injected into the hidden states of a trained model to sample augmented input with minor shifts [40]. Lastly, lexical substitution has been used to create augmented data where the input is non-natural text data [96]

2.4 Evaluation Strategies

Evaluating natural language models, generation in particular, is challenging as there are numerous characteristics one should be interested in - fluency, syntactic diversity, lexical diversity, semantics, correctness, and faithfulness to the input. Existing metrics generally evaluate one or two of these angles, but never all of them. In this section we discuss the various types of metrics as well as what they measure and any shortcomings. Metrics are typically split into string, n-gram, embedding-based, and learning metrics. In this work, I focus on n-gram and embedding-based metrics for evaluating the conducted experiments. I chose these metrics because they are computationally inexpensive, correlate well with human judgement, and allow for comparisons to multiple *target texts*. For simplicity, the text output by the model is referred to as the *generated text* whereas the original text is denoted as the *ground-truth*.

2.4.1 N-gram metrics

N-gram metrics compare the *generated text* to the *ground-truth* by comparing the occurrence of n-grams between the two. Elevated scores on these metrics stem from the model being able to generate the correct vocabulary. This ensures that the *generated text* matches a particular quality given by the *ground-truth*. On the other hand, if the *generated text* includes any deviations from the vocabulary then the score is penalized whether the changed n-grams retain the context or not. While these sorts of metrics promote correctness, they also discourage lexical diversity.

Bilingual Evaluation Understudy (BLEU) score is an n-gram metric that was originally proposed for evaluation machine translation [66]. BLEU score is on a range from 0 to 1 with the former implying the *generated text* has no shared n-grams with the *ground-truth* and the latter indicates a perfect translation. It compares n-grams in the *generated text* to one or more *ground-truth* translations for values of *"n"* in 1 to 4. At its core, BLEU is a precision measure for n-grams. For NMT, n-gram based methods are the standard as there exists only a few correct translations with any other options being considered wrong. In this case, penalizing n-grams not in the *ground-truth* is a positive. Text summarization and data-to-text generation do not have a finite number of correctly *generated texts*, so naturally there will be deviation. Despite this, BLEU is a cheap and efficient way of evaluating natural language models that continues to correlate well with human judgement.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is a metric, similar to BLEU, that consists of varying implementations for different use cases [48]. While BLEU was intended for NMT and is a metric of precision, ROUGE on the other hand focuses on recall as it is intended for summarization tasks. A recall metric scores well when it can reproduce all of the n-grams in the ground-truth, that is given input it generates all encompassing text. ROUGE is comprised of numerous implementations based on different underlying measures such as:

- ROUGE-1: assesses the recall of unigrams (words).
- ROUGE-2: assesses the recall of bigram (pairs of words).
- ROUGE-N: assesses the recall of n-grams (tuples of length "n")
- ROUGE-S: assesses the recall of skip-bigram based co-occurrence statistics. Simply, any pair of words that follow the order in which they appear in the sentence.
- ROUGE-L: assesses the recall of the longest common subsequence (LCS) within the text. This takes into account sentence structure similarity and identifies the longest occurring n-gram.

For the experiments in this work, evaluations are primarily done with ROUGE-L (RL) and discussion of results pertains to the ROUGE-L F1-score if not otherwise specified. Although ROUGE is better designed for summarization tasks, it still suffers from similar issues to BLEU due to its reliance on n-gram comparisons.

Metric for Evaluation of Translation with Explicit ORdering (METEOR) is another metric based on comparing subunits of the text. In particular, METEOR is a generalized concept of matching unigrams between the *generated text* and *target text*. Once all unigram matches between the two have been found, a score is computed that captures how well-ordered the matched unigrams are between the *generated text* and *target text*. This is a valuable evaluation metric when the content of the *source data* has a specific ordering within the *target text* that should be enforced.

2.4.2 Embedding-based metrics

BERTScore is an embedding-based evaluation metric that assesses contextual similarity rather than the regurgitation of previously seen n-grams [105]. As discussed in previous sections, the contextual embeddings of pre-trained language models, namely bidirectional encoder representations from transformers (BERT) [18], facilitate comparison of words and sentences using cosine similarity or some other distance metric for $\mathbb{R}^n$. BERTScore, hence the title, compares the contextual embedding for the *generated text* to that of the *ground-truth*. While different sentences ultimately result in non-identical embeddings, cosine similarity is less penalizing in terms of error than precision and recall of occurring n-grams. This means, *generated text* with the same semantics but different wording are still scored well. Conversely, contextual embeddings and why words are located in $\mathbb{R}^n$ as they are is a less transparent evaluation approach. With embedding-based metrics, correctness may be an afterthought, but are useful for evaluating lexical diversity and semantics.

Chapter 3

Methodology

3.1 Datasets

For experiments, the methodology discussed next is evaluated using the WebNLG [25, 26] and E2E [64] datasets. Evaluating with more than one dataset suggests results are generalizable and promotes reproducibility, particularly as both the datasets are publicly available. WebNLG and E2E share numerous parallels across their *source data* and *target text*, making comparisons between the two straightforward. In the datasets the *source data* consists of text structured as semantic triples [90] and abstract meaning representations [6] while the *target text* is summaries of these representations - presenting it as natural language. Although the datasets share various similarities given their formulaic nature, they do differ with respect to size, length of text, and other linguistic characteristics. Table 3.1 displays the number of training samples, validation, and test data for both datasets. I outline these differences as they arise in the discussion of the nuances of the datasets which is to follow.

3.1.1 WebNLG

The WebNLG dataset is crowdsourced from the DBpedia knowledge base which aims to extract structured content from Wikipedia [3]. As mentioned, the *source data* is structured as semantic triples which are comprised of three atomic units: subject, predicate, and object. For a given example in the dataset, the *source data* may pertain to a set of semantic triples with the length of the *target text* being directly correlated to the size of the set. An example is given in the top row of Table 3.2. Each set of triples in the dataset along with their respective summaries are a part of one of the following categories: *Airport, Artist, Astronaut, Athlete, Building, CelestialBody, City, ComicsCharacter, Company, Food, MeanOf-*

Dataset	N_{train}	N_{val}	N_{test}
WebNLG	35,426	4,464	3,934
E2E	42,061	4,672	630

Table 3.1: Number of data in the different splits of either dataset.

Dataset	*Source Data*	*Target Text*
WebNLG	Harold_French | birthYear | 1897	Harold French was born in 1897
E2E	name[Clowns] eatType[pub] near[The Sorrento]	Close to The Sorrento you can find Clowns pub.

Table 3.2: *Source data* and *target text* examples for both WebNLG and E2E.

Transportation, Monument, Politician, SportsTeam, University, and *WrittenWork*. All of the categories appear in both the training and test splits of the dataset. Since the model trains on these categories, we denote it as *in-domain* data. Whereas, the test set includes three additional categories of *Film, Scientist,* and *MusicalWork*. As the model does not see these categories during training, I denote data from these three categories as *out-of-domain* data. WebNLG is supplemented with multiple verbalizations for each set of triples. While there are 35,426 unique training outputs, there are only 13,211 unique training inputs. Furthermore, a dataset consisting of tens of thousands of training examples is considered data scarce in comparison to other datasets making this a good challenge for data augmentation.

3.1.2 E2E

The E2E dataset is crowdsourced from human reviews in the restaurant domain [64]. The *source data* for E2E are abstract meaning representations (AMRs), specifically slot filter representations which convey meaning or semantics through properties. Slot filter representations are popular in describing entities when only a subset of properties from a larger selection apply. E.g. a restaurant may have one or more food types (Italian and Spanish), but certainly not all of them. An example is given in the bottom row of Table 3.2. Given that all the data in E2E pertains to the restaurant domain and any slot filters occurring in the test split only occur in the train split, the E2E dataset has no need for a concept of *in-* and *out-of-domain* data. Similar to WebNLG, E2E consists of less unique inputs than outputs. With only 4,862 unique AMRs and 42,061 unique text summaries, E2E offers a larger disbalance between training data and test data size than WebNLG. Despite this, the authors claim that the descriptive nature of the E2E data results in more lexically diverse data for training. With only 7,000 more approximate training data, E2E is also a candidate dataset for data augmentation.

3.2 Data Augmentation

The data augmentation strategies used in this work are loosely based off motivating work as reviewed in Section 2.3, but is also adjusted for the data-to-text generation datasets present. Each strategy transforms the *target text* in some manner before being added back

Strategy	Sentence
None	Coming from the region of Visayas, in the Philippines, Binignit, is a type of dessert. Which banana as the main ingredient but also has sago in it.
RE	Coming from the region of Visayas, in the Philippines, Binignit, is a **gqcz qp** dessert. Which banana as **rde** main ingredient but also **hfr** sago in **frv**
LS	Coming from the region of Visayas, in the Philippines, Binignit, is a **variety** of dessert. Which banana as its main **component** but also has sago in it.
PP	**Binignit is a type of dessert from the Visayas region, in the Philippines**, which has a banana as the main ingredient, but also has sago in it.

Table 3.3: Sentences generated using the implemented data augmentation strategies. None: no augmentation, RE: random erasing, LS: lexical substitution, PP: paraphrasing. Bolded are examples of the effects each augmentation strategy can have.

to the training data resulting in an enlarged dataset. To ensure comparability between the augmentation strategies, each one transforms the *target text* rather than the *source data*. For other datasets, it is not guaranteed the augmentation strategies used would be applicable depending on the *source data's* structure. As the WebNLG and E2E datasets are limited to thousands of unique training data, the amount of augmented data was restricted such that the cumulative training data does not exceed double the original total. Preliminary experiments demonstrated that an excess of augmented data resulted in poor results, likely as the original training data became overshadowed. Despite this self-imposed restriction, each augmentation strategy is capable of producing numerous augmented copies from a single training data. Paraphrasing is the most limited in this sense as there is a finite number of paraphrased versions for a given *target text*.

In the following subsections I present the high-level details of random erasing, lexical substitution, and paraphrasing. These details include adaptions for data-to-text generation, source material, and an example of how the transformation affects the *target text*. Accompanying the details is a pseudo-code implementation to encourage reproducible results if others choose to do so. The implementations as are, directly correspond to the results in Chapter 4. Furthermore, lexical substitution is accompanied by a dense equation that contributes to the high quality synonyms it produces. This equation is broken down in the corresponding subsection.

3.2.1 Random erasing

Random erasing was briefly touched upon in Section 2.3.1. There, it produced augmented data by randomly erasing part of an image with a mask consisting of random values. Different to cropping, this imposes a grayish rectangle over top the image preventing it from relying on the erased physical characteristics. The adoption for this strategy was motivated by its similarity to inserting, deleting, and swapping random characters. In image data, the random mask consists of random pixels so for text data, the random mask consists of random characters which is just a more concentrated way of doing the previously mentioned token-level transformations. Like the motivating example, I only erase 15% of the *target text*. That is, 15% of the processed tokens are replaced with tokens containing random characters equaling a variable length. For either dataset, this resembles obfuscating a portion of the *target text* as seen in the second row of Table 3.3

A key descriptor of random erasing is its ability to learn through occlusion. Occlusion for our data is considered to be blocking out subsequences within the *target text*. This could be understood as training the model on portions of a training data before it is able to generate the text as fully desired.

Algorithm 1 Random erasing data augmentation algorithm

```
Input:
    $D$ - a dictionary of two lists, source data and tokenized target text
    $p$ - the percent of tokens to randomly erase, 0.15
Output:
    $D^*$, a dictionary of two lists, $|D_l^*| = 2 \cdot |D_l|$
 1: $D^* \Leftarrow D$
 2: for $src, tgt$ pair in $D$ do
 3:     $tgt^* \Leftarrow$ initialize empty array
 4:     for $token$ in $tgt$ do
 5:         $p_1 \Leftarrow Rand(0, 1)$
 6:         if $p_1 \geq p$ then
 7:             $t^* \Leftarrow Rand(|token|, ASCIICharacters)$
 8:             $tgt^*$ append $t^*$
 9:         else
10:             $tgt^*$ append $token$
11:         end if
12:     end for
13:     Append $src, tgt^*$ to $D^*$
14: end for
15: Return $D^*$
```

3.2.2 Lexical Substitution

Lexical substitution is an easy augmentation strategy to conceptualize as it is a skill humans use daily when digesting, communicating, and analyzing information. It is natural for an individual to re-write information they have learnt in their own words by swapping formal for informal language or to convey a topic on a per case basis depending on the audience. There are numerous approaches for generating candidate synonyms to replace words with, but most popular is thesaurus which is the principal behind Word-Net [55]. Look-up methods such as Word-Net are limited by literal synonyms, imposing a hard constraint on how words can be substituted.

The introduction of contextual embeddings offers a soft-substitution method for replacing words with synonyms as seen in Section 2.2.1. One approach for leveraging contextual embeddings to generate synonym candidates is by masking words and predicting the hidden word with a BERT language model [18]. An alternate approach provides slightly more diverse synonyms which is using the contextual embeddings other similarly embedded words in $\mathbb{R}^n$ based on a distance metric. This removes the context of the surrounding *target text* which can constraint the synonym candidates to a small list. To further obfuscate the candidates, applying dropout to a token's embedding incorporates additional variation by shifting the space in $\mathbb{R}^n$ being searched in. For this work, I opt to use the synonym candidate scoring equation shared by [110, 96].

$$Score_{SIM}(t, t'; j) = \sum_{i=0}^{n} \alpha_{i,j} \times \Lambda(\mathbf{h}(t_i), \mathbf{h}(t'_i)) \tag{3.1}$$

In Equation 3.1, t is a sequence composed of words $[w_0, w_1, \ldots, w_n]$ and its counterpart t' is the same sequence, but with the BERT internal embeddings having dropout of 0.2 applied at position j. The modified embedding is used to obtain a candidate c which is substituted into t' such that its sequence resembles $[w_0, w_1, \ldots, c, \ldots w_n]$.

The terms $\mathbf{h}(t_j)$ and $\mathbf{h}(t'_j)$ are the concatenation of BERT's last four hidden layers when the sequences represented by t and t' are passed through the model. Λ represents the cosine similarity between these two stacked vectors. The cosine similarity is then used to scale the first term in the equation. The term $\alpha_{i,j}$ is the average self-attention across all heads in all layers of the attention ranging from the i^{th} to the j^{th} token. If $i > j$ then it is from the j^{th} to the i^{th} token. Finally, this scaled self-attention is summed over all tokens.

The previously cited works demonstrate that high quality synonym candidates are generated when this score is greater than or equal to 0.9. No limit is set for the number of synonyms that can be incorporated into the augmented copy, but only nouns, adjectives, adverbs, and numerals are targeted in this implementation. An example of synonyms generated by this lexical substitution strategy are seen in row three of Table 3.3.

3.2.3 Paraphrasing

Unlike random erasing, paraphrasing has no direct computer vision equivalent. While rule-based approaches for constructing paraphrased copies exist, most commonly it is done using an external model such as a language model. Trained similarly to other pre-trained language models, this approach provides an example-driven albeit computationally expensive way to paraphrase data. On the shorter sentences in both datasets, limited paraphrased copies are available and sometimes not at all. Due to the stochastic nature of the augmentation strategy, I provide precise implementation details to encourage reproducibility and to gauge the impact of this augmentation strategy.

Algorithm 2 Lexical substitution data augmentation algorithm

```
Input:
    $D$ - a dictionary of two lists, source data and tokenized target text
    $p$ - the threshold score for a candidate to be accepted, 0.9
Output:
    $D^*$, a dictionary of two lists, $|D_l^*| = 2 \cdot |D_l|$
 1: $M_{ls} \Leftarrow$ Load a pre-trained BERT model
 2: $D^* \Leftarrow D$
 3: for $src$, $tgt$ pair in $D$ do
 4:     $d \Leftarrow$ empty dictionary
 5:     $candidates(\Leftarrow \text{Dropout}(M_{ls}(tgt)_{embeddings})$
 6:     for $c$ in $candidates$ do
 7:         $s_c \Leftarrow$ Score $c$ with $Score_{SIM}$
 8:         if $s_c \geq p$ then
 9:             Substitute $c$ into $tgt$
10:         end if
11:     end for
12:     Append $src$, $tgt$, to $D^*$
13: end for
14: return $D^*$
```

Strategy	Level	Computational Cost	Language Model?
Random Erasing	Token	Low	No
Lexical Substitution	Token Embedding	Medium	Yes
Paraphrasing	Sequence	High	Yes

Table 3.4: Presents varying characteristics between the implemented data augmentation strategies in this work.

For a given training data, I have the paraphrasing model return **three** paraphrased copies then check to see which is the most diverse from the augmented *target text*. Whichever is most diverse with regards to a word edit score is added to the training data as the augmented copy. If no satisfactory paraphrased version is produced, then the algorithm will skip that training data continuing onwards. The external model used for paraphrasing is based on the bidirectional and auto-regressive transformer (BART) architecture and is available through the HuggingFace library as *"eugenesiow/bart-paraphrase"* [45, 46] An example of paraphrasing augmentation is seen in Table 3.3 in row four. Observe that the ordering of content has changed and contains fewer words.

3.2.4 Augmentation properties

Table 3.4 highlights some of the key differences between the implemented augmentation strategies including, most importantly, the computational cost which is elaborated on in

Algorithm 3 Paraphrasing data augmentation algorithm

```
Input:
    $D$ - a dictionary of two lists, *source data* and tokenized *target text*
Output:
    $D^*$, a dictionary of two lists, $|D_l^*| = 2 \cdot |D_l|$
1:  $M_p \Leftarrow$ Load English paraphrasing model
2:  $D^* \Leftarrow D$
3:  for $src$, $tgt$ pair in $D$ do
4:      $n \Leftarrow \sqrt{||text||} \cdot p$
5:      $t^* \Leftarrow M_p(tgt, 3)$ (generate three paraphased copies of $tgt$)
6:      $s^* \Leftarrow score(t^*)$
7:      if $\exists$ satisfactory $x \in s^*$ then
8:          Append $src, max(s^*)$ to $D^*$
9:      end if
10: end for
11: Return $D^*$
```

Chapter 4. Random erasing which only accesses select tokens within the *target text* requires no external model making it the most inexpensive to implement from a computational standpoint. Although both lexical substitution and paraphrasing require external models, lexical substitution does not require a full pass through the model. Rather the internal embeddings and hidden states are extracted before the *target text* passes through the decoder of the model. Since inference typically is more computationally expensive, lexical substitution sits at a medium between random erasing and paraphrasing in terms of computational cost. Additionally, lexical substitution differs from the other two augmentation strategies in that it manipulates contextual embeddings for the purpose of producing synthetic data. Lastly, as the *target text* requires a full pass through the external model in paraphrasing it achieves the highest computational cost. This is exacerbated by the fact I generate three different paraphrased copies that are scored.

3.3 Data-to-text Generation Models

The data augmentation strategies in this thesis are assessed with the text-to-text transfer transformer (T5) language model [77]. Regardless, this work is still interested in the task of data-to-text generation and thus the previously discussed Transformer is also implemented and evaluated. The Transformer provides the basis for the T5 and offers an interesting comparison both in terms of model complexity, but the effect of pre-training and transfer learning as well. The following subsections loosely discuss common implementation details for both models as well as any particular decisions of this work.

3.3.1 Transformer

As the Transformer is an architecture rather than a model, an instance of it must be put forth in this work. The Transformer model used here follows that of the one used in the experiments of [97]. The finer details of this model will be discussed next and in Chapter 4. Future references to this model, in particular in Chapter 4, will be referred to as the Transformer.

Although it is hypothesized and expected that T5 will outperform a Transformer, it remains valuable for experimental results to act as a baseline given its popularity. Following the model diagram of 2.4, the embeddings for the input and output of the Transformer are randomly initialized. The encoder stack of the Transformer consists of multi-headed attention, an additive and normalization layer, a feed-forward network of dimension $\mathbf{D} = 256$, and then a final additive and normalization layer. The decoder stack follows a similar structure, with the primary difference is that the attention is accompanied by a mask to prevent it from looking forward into the future. For data-to-text generation this is called a causal mask where each pass of the data reveals the following token one-by-one. A main improvement of this structure over a recurrent neural network (RNN) is that these feed-forward networks operate independently rather than recursively so these sections of the architecture are easily paralleled. All the weights are randomly initialized and any parameters that cannot be randomly initialized are assigned as they were in the paper. This removes any notion of pre-training for the model, emulating the model learning the task anew. Like the T5, the *source data* for the Transformer was converted to a text-to-text format for comparability. The Transformer was trained on data tokenized at the word-level, split by whitespace, and special characters.

3.3.2 T5

The T5 mimics the details of the original paper although the implementation used is from HuggingFace, a library that facilitates the use of language models [77, 102]. As T5 is applicable to a range of natural language tasks, it has to be specified the kind of head the model uses to produce the desired output. For this work a language modeling head is used. Unless specified otherwise, the T5 referred to in this work is the base version available through HuggingFace signified by "t5-base" in the API.

There are a few reasons to discuss on why T5 was chosen as the language model of choice for this work. First, T5 is pre-trained on various language tasks cast as text generation problems meaning in theory it will generalize better to unseen tasks - data-to-text generation. In comparison, BART is more narrowly focused on text reconstruction and summarization tasks as a sequence-to-sequence model [46]. Next, T5 is trained on a massive corpus with text from a multitude of sources which more generally teaches it syntactic structure, semantics, and language. Lastly, there is flexibility when fine-tuning T5 as it employs a unified framework that converts all text-based language problems into a text-to-text format. In this case, so long as the *source data* is able to be represented via text then the T5 will be able to generate corresponding text. Meaning, so long as the *source data* is linearized to a string then T5 is able to model it as a sequence-to-sequence task. For WebNLG and E2E this is simply done by flattening the semantic triple or abstract meaning representation.

T5 is different from the Transformer used in this work in a few ways. First is the pre-training approach used which was span masking that is discussed in Section 2.1.4. Pre-training imparts a beneficial general understanding of language onto the model as discussed prior. Architecturally, the T5 uses a relative positional encoding instead of an absolute positional encoding. Absolute positional encoding operate based on the position of the token given the max sequence length the model is able to handle. Rather, a relative positional encoding compares distance between tokens to accommodate for unknown sequence lengths. Lastly, the T5 model includes layer normalization before each multi-head attention block (orange in Figure 2.4) to prevent overfitting to any of the tasks or data used during pre-training.

To avoid catastrophic forgetting, T5 is initialized with a low learning rate and is only trained for 3 epochs which was found to best when validating different parameters. Furthermore, the weights for the model's encoder are frozen to retain learnt linguistic information and only the decoder weights are tuned to adapt to the unseen task.

In Chapter 4, a T5 *without* pre-training is used to observe the effect prior language knowledge has for data-to-text generation. In truth, this iteration is not exactly without pre-training, rather the model weights are randomly initialized to remove the impact of pre-training. In relevant tables this iteration of the T5 is denoted T5*.

To the best of my knowledge, this is the first time these augmentation strategies have been tested with the T5 on these datasets.

3.4 Summary

In this chapter, I shared the implementation of the data augmentation strategies used in this work as well as how they are incorporated into the training procedure. Additionally, I outlined my experimental set-up for conducting an analysis of data augmentation for data-to-text generation when confronted with scarce data.

The next chapter expands upon this discussion by providing further experimental set-up details. I also share the results for the experiments conducted, highlighting improvements, shortcomings, and other interesting observations.

Chapter 4

Analysis

4.1 Research Questions

As discussed in Section 2.3, data augmentation has been proven successful in many *natural language* tasks as well as computer vision (CV) while for data-to-text generation the overall research is more cursory. These fields all require models utilizing large and balanced datasets for training which, as described, is relatively rare for the field of data-to-text generation due to its nature.

Although the augmentation strategies explored are not original to this work, their combination with T5, application to these datasets, or usage in data-to-text generation are. Despite this, I will deepen the understanding of these strategies for data-to-text generation and show whether they are viable solutions or fall short of the mark. In doing so, I will answer the following research questions throughout this chapter.

The first question is whether pre-trained language models are suitable for the task of data-to-text generation. It is common knowledge that these models have rapidly become state-of-the-art in various *natural language* tasks, but the inconsistent nature of the *source data* does not guarantee "a one size (or model) fits all" approach. I selected a popular pre-trained language model to be fine-tuned on both datasets before assessing its suitability.

Furthermore, does data augmentation provide consistent and observable increases on top of said model for data-to-text generation. While pre-training occurs during an entirely different stage of training a model, its inclusion potentially competes with the improvement data augmentation would garner.

Lastly, does data augmentation provide more substantial improvements when included on datasets with even smaller amounts of training data. If so, is there a threshold at which data augmentation ceases to be impactful.

4.1.1 Training procedure

The training procedure for all models follows roughly the same structure. The data was tokenized using the respective tokenizers before being passed to the models in batches.

As the data augmentation strategies initially produce look-ups that facilitate generating augmented copies then those are incorporated into training the model during run-time. The T5 model was pre-trained on certain tasks using teacher forcing which we continue with as well. For comparability, this is also done during training for the transformer. Both models make use of a cross entropy loss function during training. This means that learning was done using the logits the models produce and not the resulting sequence probabilities.

4.1.2 Validation and testing procedure

Both the WebNLG and E2E datasets come with pre-defined validation datasets meaning cross-validation is not required as a standard evaluation is pre-determined. After a number of training epochs, the model is evaluated on the validation split and scored using loss on the validation split. After completing all epochs, the model checkpoint with the best score is retained for evaluating on the test split of the dataset.

The testing procedure is the same for all models and datasets. Using the test split of the datasets, we generate text from the *source data* and use existing *target text* to score the model with the evaluation metrics discussed in section 2.4.1. To provide further consistency, HuggingFace has a function for standardizing the settings used in the text generation process. All models generate text using beam search decoding due to its improved ability to search for the optimal resulting sequence [24].

It should be noted that the validation split of the datasets are used when selecting the few parameter or hyperparameter values that are changed within the models. These are discussed in the following subsection as well as why the majority of the parameters and hyperparameters remain untouched.

4.1.3 Hyperparameter selection

The focus of this work is the data augmentation strategies, not the models implemented for data-to-text generation. Therefore, the model hyperparameters are left largely untouched to those in the T5 paper [77]. While performance could likely be improved by tuning the hyperparameters extensively, this negates one of the goals of this work which is to see if data augmentation is suitable replacement for such a time-consuming endeavor. Only the number of epochs the model is trained for and the learning rate are adjusted, primarily to avoid catastrophic forgetting or overfitting. Furthermore, additional parameters could have been added to the augmentation strategies and tuned but then it is possible to fall into the same trap as hyperparameter tuning. Instead, the augmentation strategies are similar to their motivating examples in other tasks unless otherwise specified.

The number of epochs and learning rate were uncovered by training the model without any augmented copies, using only the training and validation splits of the dataset to find optimal values. The metric for comparing hyperparameters is METEOR as it is computationally inexpensive and approximates human judgement well. To account for randomness, a configuration is assessed multiple times on the validation split. Lastly, this work assumes that the augmented copies do not deviate sufficiently from the original data distribution to warrant new hyperparameters.

Transformer

Parameters for the transformer model follow common implementations available in various coding libraries and as described in papers. This includes 8 attention heads, 6 layers of *transformer blocks* for both the encoder and decoder, a dimension of 256 for the feed-forward network, and an embedding size of **256** as well. Dropout applied throughout the Transformer is set to 0.1.

To tokenize the data for the Transformer separate tokenizers are required for the *source data* and *target text* due to differences in structure, set-up, special symbols, and sequence length. This holds true for both datasets. As some rare tokens in the *target text* were actually values such as 6600.66 kms, more caution needed to be taken to ensure none of the data was invalidated. Text was split on white spaces or commas, but not periods. Certain special characters were not removed as they were also included from the *source data* such as /, \, and -. Early experiments had shown that given the datasets studied, a subword tokenizer performed worse than the tokenization process outlined above. To better compare with the T5, the subword tokenizer assessed was a byte-pair encoding tokenzier which is used for the language model.

T5

As mentioned, hyperparameters for the T5 model are also selected based on the original paper or as pre-determined by the HuggingFace library. The data is tokenized using the associated tokenizer within the library as well which is a trained SentencePiece [44] tokenizer consisting of sub-word tokens with a vocabulary size of 30,522. The way SentencePiece works is that no text will have any out-of-vocabulary tokens due to the fact that tokenization happens at the sub-word level.

The number of epochs for T5 is kept low to mitigate any catastrophic forgetting of the pre-training and the encoder parameters are frozen as well. Hyperparameter tuning showed that 3-5 epochs was best with the validation performance beginning to decrease after epoch 3 on average.

4.1.4 Incorporating curriculum learning

Often times the complexity of a dataset can inhibit a model's ability to learn amidst the noise. In these scenarios, the solution is not to add more data as this contributes to the problem but simplify the learning process. As discussed in Section 2.2.3, curriculum learning includes approaches to training the model on the data in a meaningful order. Recent work has shown that curriculum learning is not always a viable solution though, in particular for the WebNLG dataset [58]. In fact, the authors of the previous work observed a decrease in performance. The curriculum used for training the model in this work is to pass increasingly complex data. A higher number of semantic triples (WebNLG) or abstract meaning representations (E2E) directly corresponds to a higher complexity, that is both datasets have *source data* that are sets of objects and the larger that set the more complex it is. The training data is then sorted by this complexity with easier examples being presented to the model earlier in the epoch.

Their work used the Transformer architecture and so the same effect should be observable with the Transformer used in this work. Furthermore, I use this curriculum to fine-tune the T5 and see if the effect holds true for this model as well. To confirm whether a curriculum of training on increasingly complex data is worthwhile, I train or fine-tune the models with and without curriculum learning.

4.1.5 Data-to-text generation

The success of the implemented models on the task of data-to-text generation is evaluated by following the procedure outlined in Sections 4.1.1 and 4.1.2. Once trained, the model is used to generate text that is compared to the *target text* of the test data. Here, all variations of ROUGE are considered for a comprehensive experiment but when this experiment is repeated with data augmentation I consider the F1-score of ROUGE sufficient for reporting.

As discussed in 3.1.1, the WebNLG dataset is composed of various categories with a few not being seen until after training is finished. Part of the experiment includes a breakdown of the evaluation metrics per category to observe if T5 is able to generalize to the unseen categories.

4.1.6 Augmentation settings

In section 2.3.1, random erasing was introduced as a data augmentation strategy that is applied to the input image for various CV tasks. While literature points toward data augmentation being more appropriate for the output in sequence-to-sequence tasks, a comparison is worthwhile.

The results when data augmentation is included assume that for each training data, one augmented copy is produced. In some cases, lexical substitution and paraphrasing do not given their implementation and reliance on external models. Data augmentation in performed during the training procedure when a new batch is formed and before it is passed through the model. Between each experiment where data augmentation was used, the implementation and amount in which an example is augmented does not change.

4.1.7 Dataset upscaling

Dataset upscaling refers to increasing the size of a dataset, generally done to improve model performance. This comparison is done with each augmentation strategy as well as a model trained without augmentation. Not only does this experiment evaluate whether augmented data can replace collected data, but it also shows whether a model reaches its performance threshold faster and what amount of training data (N_{train}) does data augmentation work best with. This experiment is conducted with $N_{train} = \text{None}, 1, 5, 500, 1,000, 5,000, 15,000,$ and $25,000$. Conversely, this experiment also informs the dataset size in which augmentations impact begins to fade.

4.1.8 Dataset considerations

As seen in Table 3.2, both datasets come as formatted source data which need to be linearized to be accepted by the Transformer and T5. One of the primary decisions for selecting T5 as the language model used in our experiments is that this conversion of any *input-to-text* task into a *text-to-text* task is rather trivial. This means that despite the *source data* format, the data-to-text generation process is treated the same across tasks and requires minimal if any changes to the architecture. While this provides a standardized approach for handling the data, the WebNLG and E2E datasets contain differences in how it should be processed such as vocabulary, *source data* structure, and length of constituent sequences. Below, I outline some of these differences. The sequence length of the *target text* is measured by the count of characters as different tokenizers generate varying amounts of tokens. For the *source data*, the comprising number of triples or meaning representations is measured as the length. The last dataset characteristic we consider is the vocabulary in both datasets.

WebNLG

As previously defined, the WebNLG *source data* are sets containing a variable amount of semantic triples. Rounded to the nearest integer, the average number of triples used is 3. Despite the *source data* consisting of a few triples, the character count when transformed for the text-to-text nature of the T5 model is an average of 151 characters. The disparity between number of triples and character count is explained by the complexity of the *subject*, *predicate*, and *object* within the triple. For example, one possible *predicate* is "elevationAboveTheSeaLevel". This *predicate* when verbalized is not as simple as a one-word response. When evaluating models I experimented with both retaining compound word as a single token and splitting it into multiple tokens with the latter appearing to have more stable model behaviour. On the other hand, the *target text* in the WebNLG dataset has an average character count of 116 characters. As discussed for the E2E dataset next, the average character count is roughly the same between the two but the vocabulary for WebNLG is much larger. Following a simple strategy of splitting sentences on whitespace and any

Dataset	**Split**	**BLEU**	**R (P)**	**R (R)**	**R (F1)**	**M**
E2E	Train	5.21	39.96	63.39	29.89	30.18
	Dev	5.06	41.14	66.85	30.28	31.34
	Test	24.49	46.29	67.23	35.76	41.47
WebNLG	Train	6.85	44.44	50.45	40.88	10.99
	Dev	7.43	44.05	50.15	40.42	11.31
	Test	4.54	43.32	48.67	40.62	8.53

Table 4.1: BLEU, ROUGE, and METEOR results obtained by a copy-based approach for data-to-text generation. **R**: ROUGE, **(P)**: precision, **(R)**: recall, **(F1)**: F1-score, **M**: METEOR. This approach is useful in approximating the difficulty of the task.

commas, the number of unique words in WebNLG is 13,570. This is the first indicator that n-gram metrics such as BLEU will trend lower on the WebNLG dataset due to the range of possible words to be verbalized. Although metrics are expected to be different between datasets, this helps for framing expectations when discussing results and comparing between the two tasks.

E2E

The E2E dataset has slightly different *source data* which is sets of slot filters. I use the number of slot filters as the input length for E2E. This results in an average input length of 5 which is slightly larger than the WebNLG dataset. Conversely, the average character count for the E2E input is 106. A simple explanation is that E2E uses more slot filters than WebNLG uses triples, but the slot filters are less complex in nature. For example, slot filters found in E2E are *"priceRange"*, *"name"*, *"familyFriendly"*, and more which are easily answered with a single string, float, or boolean value. Despite the difference in *source data*, the *target text* of E2E is roughly the same with an average character count of 109. Although, following the same strategy for identifying E2E's *target text* vocabulary, it results in one that is much smaller with only 4320 unique words. This means by even just regurgitating known words, the model has a better chance at avoiding certain metrics from decreasing.

4.2 Results

This section demonstrates the capabilities of the selected models for the task of data-to-text generation. Furthermore, it highlights the results obtained when these models are supplemented with data augmentation strategies - whether there are improvements or a lack thereof. In particular, these methods are explored with respect to a data scarce environment by investigating what occurs when the amount of training data is reduced. The main approach to investigating these results is through natural language generation metrics as discussed in Section 2.4.1, but in some cases further tables and figures are provided to delve deeper into aspects of the augmented data and how that impacts data-to-text generation.

Model	CL	WebNLG			E2E		
		B	R (F1)	M	B	R (F1)	M
Transformer	no	2.23	23.18	18.44	29.73	48.25	50.79
	yes	1.01	12.94	13.84	25.57	41.92	31.37
T5	no	32.74	57.75	58.79	61.71	64.23	69.48
	yes	31.19	53.48	53.70	61.21	66.19	70.58

Table 4.2: The impact of including a curriculum into the training procedure with both the T5 and Transformer. no: implies training data was randomized yes: curriculum learning was performed.

Model	WebNLG					E2E				
	B	R (P)	R (R)	R (F1)	M	B	R (P)	R (R)	R (F1)	M
Transformer	2.23	21.96	26.65	23.18	18.44	29.73	50.25	48.29	48.25	50.79
T5*	11.16	36.48	20.34	24.33	19.97	45.65	68.43	39.97	49.42	44.58
T5	32.74	63.99	53.91	57.75	58.79	61.71	72.71	58.64	64.23	69.48

Table 4.3: The results of applying a Transformer, T5 without pre-training, and a T5 with pre-training to WebNLG and E2E. * implies that the model was initialized without pre-training.

4.2.1 Curriculum learning

Table 4.2 displays the outcome of training a Transformer and T5 with and without curriculum learning. Similar to the aforementioned work, it is observed that curriculum learning results in decreased performance across three different evaluation metrics for both the Transformer and T5 when dealing with the WebNLG dataset. Furthermore, curriculum learning only decreases the performance of the Transformer when trained on the E2E dataset. For the T5 trained on the E2E dataset with curriculum learning, there are slight improvements such as a 1.96 increase in the ROUGE F1-score and 0.9 increase in METEOR. These improvements are insignificant and cannot be confirmed as anything more than randomness based on the experimental settings. Although out of the scope of this work, I suggest further exploration into curriculum learning for data-to-text generation as a research avenue. The results presented moving forward assume that the model has been trained **without** curriculum learning.

4.2.2 Results of data-to-text generation

First and foremost, the question of how difficult of a challenge is data-to-text generation must be answered. A crude method for approximating the difficulty a model will have learning from the data is to construct a copy-based approach as was historically popular with data-to-text generation. For both datasets, any special characters which define a new unit in the *source data* are removed, conjoined words are split apart, and any additional

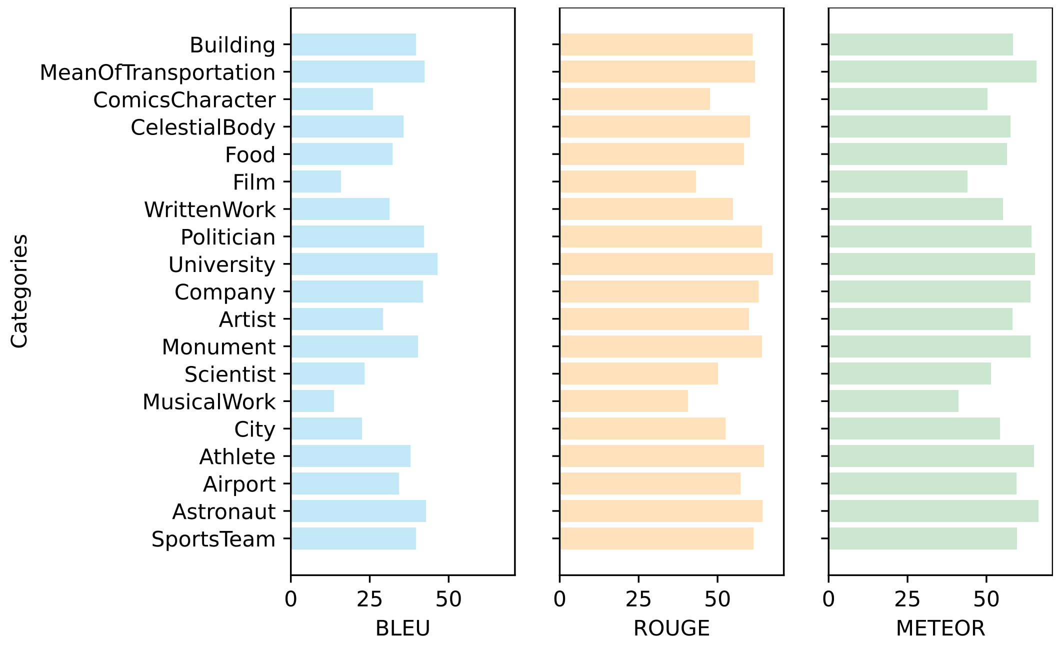

Figure 4.1: A breakdown of BLEU, ROUGE, and METEOR for the WebNLG dataset on the various categories within its test data. *Film, Scientist,* and *MusicalWork* are the categories that are not part of the training data.

tags dropped. Treating this stitched together version of the *source data* as *generated text*, it is then compared to the *target text* with the evaluation metrics mentioned in this work.

Table 4.1 demonstrates the results of the aforementioned copy-based approach for data-to-text generation which provides a brief glimpse into the complexity of the two datasets. For the E2E dataset, simply copying the *source data* to act as the *generated text* demonstrates inconsistencies. While the ROUGE precision and F1-scores are high, the remaining metrics such as BLEU, ROUGE recall, and METEOR are below average in comparison. This can be explained by the fact that much of the *source data* directly appears in the *target text* which would lead to high precision as it is being copied. Consequently, a high ROUGE precision will lead to a higher ROUGE F1-score. A similar effect is noticeable for the WebNLG dataset as well with precision being the best performing ROUGE metric. The BLEU score for WebNLG shows consistency with the test data being the most difficult split naturally. This is not the case for the E2E dataset where the BLEU score is improved four-fold from the train and dev splits. The reasoning for this can be contributed to fewer examples in the test set thus allowing a few successful copied predictions to skew the results towards higher evaluation metrics. Because of this we should expect for the E2E test set to perform better on evaluation metrics, in particular with regards to the WebNLG dataset. For the WebNLG dataset, the test set has a more balanced mixture of regurgitating keywords as well as filler words to create a coherent sentence. This is evident in the balanced ROUGE precision and recall scores, meaning for the WebNLG dataset the most appropriate metric is ROUGE

F1-score. METEOR corresponds to a matching of unigrams between the *generated text* and *target text*. The nature of the E2E dataset is better suited for this evaluation metric than WebNLG as the attributes in the *source data* have less dependencies than the semantic triples in WebNLG. This means the *target text* in E2E has more possible alignments of the *target text* that show up in evaluation.

Table 4.3 displays the results obtained on the task of data-to-text generation with no augmentation, procuring a set of baseline results for comparison moving forward. In addition to the T5 language model, the results for a Transformer model are provided. While the Transformer is not continually used as a baseline in this work, it acts as a comparison for the T5 model and assesses the impact of pre-training. As expected, the WebNLG is the more challenging dataset to train on as noted by the low BLEU score of 2.23 the Transformer achieves. Similarly to BLEU, ROUGE also underperforms in comparison to the copy-based architecture further highlighting the troubles the Transformer experienced learning on the WebNLG dataset. Conversely, the Transformer demonstrates a 9.9 point increase with the METEOR evaluation metric.

The second row of Table 4.3 shows the results of the T5 being initialized without pre-training then trained. It is observed that the architecture change alone prompts improvements of 8.93 on BLEU, 1.15 on ROUGE, and 1.53 on METEOR for WebNLG and improvements of 15.92 on BLEU and 1.17 on ROUGE, but a decrease of 6.21 on METEOR for E2E. A decrease on METEOR appears anomalous and it appears that the more complex structure of the T5 language model provides a general improvement.

In the bottom row of Table 4.3, when comparing the T5 to itself without pre-training and the Transformer on the WebNLG dataset I noted an overall increase for all recorded evaluation metrics. While the Transformer demonstrates mixed success at modeling the dataset, ultimately T5 vastly outperforms it all on fronts - highlighting the efficacy of pre-training. As expected, ROUGE precision is the best performing metric at 63.99 for the T5 on the WebNLG dataset although the METEOR score rises an intriguing amount from both the copy-based model, the Transformer, and now with pre-training. Up 50.26 and 40.35 in METEOR respectively. Though not conclusive, by definition this means some combination of unigram selection and alignment is roughly half that of the *target text*. Similar results are noted for the E2E dataset, albeit the evaluation metrics are elevated all around as predicted from the copy approach used to gauge the datasets. With a high ROUGE precision and METEOR score, 72.71 and 69.48, T5 demonstrates its ability to recall the proper terminology for the E2E dataset coupled with the *generated text* being properly aligned with the *target text*. Figure 4.1 breaks down the performance of a pre-trained T5 model on the different categories that make up the WebNLG dataset.

N_{train}	WebNLG			E2E		
	B	R (F1)	M	B	R (F1)	M
None	1.96	29.51	6.94	7.29	42.27	41.49
1	4.25	34.16	12.80	17.16	37.68	36.80
5	19.15	38.77	29.09	36.57	40.71	36.49
500	30.85	51.25	49.24	54.35	60.10	61.29
1,000	30.72	51.86	50.46	55.91	60.55	62.75
5,000	31.47	56.11	56.32	61.86	65.92	72.37
15,000	32.16	56.95	58.51	63.53	67.50	72.76
25,000	32.51	56.75	58.31	63.21	66.85	72.68

Table 4.4: The performance of a pre-trained T5 when the amount of training data, N_{train} progressively increases from no training data up to 25,000 training data.

4.2.3 Amount of training data

Table 4.3 demonstrated the efficacy of training the T5 on a complete albeit small dataset. One benefit of pre-trained language models is their ability to be fine-tuned to new data with as little as a few examples. Table 4.4 displays the results across various metrics for fine-tuning a T5 model on progressive amounts of data for both the WebNLG and E2E datasets. Evidently, when no fine-tuning occurs and the T5 is initialized with out-of-the-box parameters the performance is not much better than the copy-based model demonstrated in Table 4.1. From no training data to just a single example on the WebNLG dataset roughly improves the performance two-fold on BLEU and METEOR while ROUGE F1-score increases by 4.65. On E2E BLEU increases more than two-fold while the other two metrics marginally decline. As the value of N increases in the leftmost column so do the associated metrics before reaching what appears to be a threshold in improvement for the T5 model. For WebNLG, this threshold appears around N = *5,000* and N = *15,000* for E2E. This corresponds to approximately, 32.0, 56.0, and 58.0 for BLEU, ROUGE and METEOR respectively for WebNLG which aligns with the results obtained by the best performing model in 4.3. For E2E this threshold is defined by 63.0, 67.0, and 72.0 for BLEU, ROUGE, and METEOR respectively which similarly aligns with the information in the aforementioned table.

For WebNLG, BLEU seems to reach its threshold first for $N_{train} = 500$ while ROUGE and METEOR reach it around $N_{train} = 5,000$. With the E2E data the three evaluation metrics all reach their threshold at roughly the same amount of training data, that is when $N_{train} = 5,000$.

4.2.4 Augmenting data-to-text generation

Recall Section 2.3 where the origins of data augmentation were discussed and the motivation for its application to data-to-text generation. In computer vision, data augmentation most often transforms the image which is being analyzed. That is, the *input* or *source data* in

To Augment	WebNLG			E2E		
	B	R (F1)	M	B	R (F1)	M
Source Data	28.38	52.42	53.27	51.25	59.50	63.29
Target Text	31.75	56.58	57.35	61.69	67.89	74.29

Table 4.5: A comparison of the random erasing augmentation strategy being applied to the *source data* and *target text*.

Strategy	WebNLG				E2E			
	B	R (F1)	M	BS	B	R (F1)	M	BS
None	32.74	**57.75**	58.79	92.66	**61.71**	64.23	69.48	96.12
RE	31.75	56.58	57.35	93.52	61.69	**67.89**	**74.29**	**96.24**
LS	27.81	57.38	58.73	93.29	60.20	66.76	73.79	96.13
PP	**32.78**	56.87	**58.95**	**93.74**	60.77	64.75	68.23	96.04

Table 4.6: Presents the results of fine-tuning a pre-trained T5 with the inclusion of data augmentation. None: means no data augmentation occurred, RE: random erasing, LS: lexical substitution, PP: paraphrasing, BS: BERTScore. Bolded values mark the best performance on the corresponding metric.

that case. As briefly mentioned, this raises a flag for data-to-text generation. Opposed to the *source data* always being an image like in computer vision, for data-to-text generation it might be numeric, graphs, or a string. More generally, data-to-text generation is a sequence-to-sequence task which fundamentally relies on a consistent mapping between the *input* and *output* sequence. I used the random erasing augmentation strategy to dictate how I apply the remainder of the augmentation strategies, barring paraphrasing which logistically does not make sense to apply to *source data*.

Table 4.5 highlights the difference between applying an augmentation strategy to the *source data* or *target text*, in particular the random erasing augmentation strategy. Once again reporting BLEU, ROUGE, and METEOR, across all three metrics and both datasets it is evident that in the context of data-to-text generation augmentation is better suited for the *target text*. Though dataset characteristics could be attributed for this difference, the more plausible explanation is that for data-to-text generation and more largely sequence-to-sequence tasks augmentation should be performed on the *target text*. ROUGE and METEOR have been the more telling metrics throughout this analysis and exhibit clear decreases of 4.06 and 4.08 on WebNLG and 10.44 and 11.00 on E2E respectively. The decrease in the model that occurs when random erasing is applied to the *source data* is a result of information necessary to the *target text* being replaced with random characters that then does not appear in the *generated text* because the model has no prior knowledge of it.

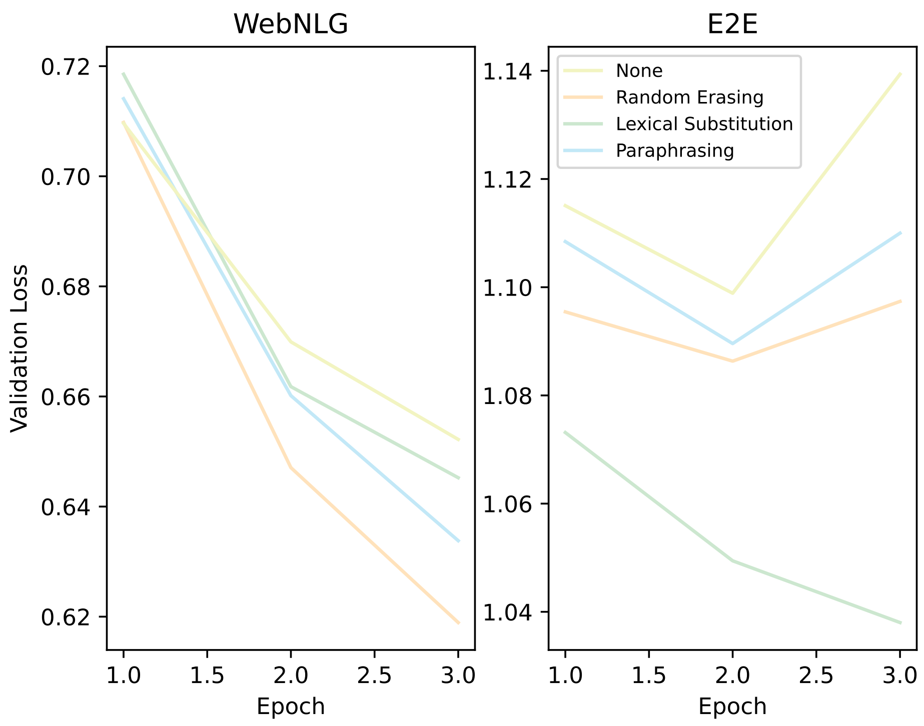

Figure 4.2: Displays the loss over three epochs of a pre-trained T5 without augmentation, with random erasing, lexical substitution, and then paraphrasing. The loss for WebNLG is presented on the left and E2E on the right.

4.2.5 General augmentation

As seen in Table 4.6, definitive improvements are difficult to observe with the exception being random erasing on the E2E dataset. Otherwise, it appears data augmentation may help as much as it may harm. For the provided results, random erasing and paraphrasing provide incremental improvements in comparison to lexical substitution. A common observation across all strategies is that improvement generally occurs in ROUGE and METEOR as opposed to BLEU. For WebNLG, the improvement on any of the metrics does not exceed 0.16 which is hardly convincing of significant improvement. In comparison, observable improvement on the E2E dataset is more significant with ROUGE increasing by 3.66 and 4.81 on METEOR with random erasing.

Despite minimal improvements, all augmentation strategies approximately match the model fine-tuned without augmentation on the BERTScore metric. In fact, for WebNLG the model augmented with paraphrased copies garners a 1.08 improvement while on E2E random erasing garners a less impressive 0.2 increase. Values for BERTScore greater than 90 imply the *generated text* of the augmented models have high degrees of similarity with the *target text* in the embedded representations.

Figure 4.2 displays the validation loss over the three training epochs for a pre-trained T5 model without augmentation, with random erasing, lexical substitution, and finally

N_{train}	$N_{augment}$	WebNLG			E2E		
		B	R (F1)	M	B	R (F1)	M
1	1	2.89	35.10	7.52	22.56	37.77	33.75
5	5	21.69	36.60	24.71	37.79	47.94	46.10
500	500	31.5	52.46	50.31	56.16	58.06	58.27
1,000	1,000	30.53	52.97	50.69	59.19	64.03	70.07
5,000	5,000	31.60	56.02	56.13	61.69	65.09	71.19
15,000	15,000	32.07	56.99	58.72	64.43	67.81	74.83
25,000	25,000	32.53	56.34	58.72	65.79	68.61	75.69

Table 4.7: The performance of a pre-trained T5 with random erasing augmentation when the amount of training data, N_{train}, progressively increases from a single training data up to 25,000 training data. $N_{augment}$ displays the amount of augmented data incorporated. The totals for this column should match that of N_{train} unless an augmented version cannot be generated for the original training data.

paraphrasing. In both cases, the model without augmentation demonstrates worse loss than those with augmentation. Contrary to Table 4.6, for WebNLG random erasing achieves the best loss while lexical substitution does for E2E.

4.2.6 Augmentation on reduced training data

Similar to Table 4.2.3, I demonstrate the results of fine-tuning a pre-trained T5 on reduced amounts of training data with the exception that the fine-tuning includes augmented data. In particular, it is augmented with the random erasing strategy. The results of this experiment are displayed in Table 4.7 for both the WebNLG and E2E dataset. For the former, there is no immediate evidence that random erasing improves any aspect of the training process when subjected to limited data though it does not noticeably do harm either. Contrarily, there is limited evidence that random erasing pushes the model towards the performance threshold for BLEU on E2E more quickly as every value for N_{train} except for $N_{train} = 5,000$ improves in comparison to the experiment without augmentation. This is coupled with an improvement of 2.26 in BLEU, 1.11 in ROUGE, and 2.93 in METEOR over the previous best further demonstrating the improvement random erasing obtains on E2E.

In the same vein, Table 4.8 displays the results on the same experimental set-up, but now the augmentation strategy used is lexical substitution. Table 4.6 showed that lexical substitution either harmed or exhibited no measurable effect on the performance of a pre-trained T5 for WebNLG and the same conclusion is drawn here in this experiment as well. Table 4.6 showed that lexical substitution either harmed or exhibited no measurable effect on the performance of a pre-trained T5 for WebNLG and the same observation is drawn from Table 4.8. This is because the metrics used rely on comparing generated tokens. Again, the same conclusion is drawn for E2E with the exception that lexical substitution improves

N_{train}	$N_{augment}$	WebNLG			E2E		
		B	R (F1)	M	B	R (F1)	M
1	1	6.13	33.94	17.42	12.15	26.44	23.32
5	5	13.15	35.51	22.92	28.53	46.64	45.30
500	481	29.31	52.25	50.32	51.15	56.78	56.03
1,000	998	28.88	53.46	51.47	55.72	61.13	62.62
5,000	4,976	25.98	56.39	56.44	61.73	65.88	71.34
15,000	15,000	26.74	57.16	58.38	61.05	66.90	73.46
25,000	24,959	26.69	57.11	58.55	61.85	67.57	74.32

Table 4.8: The performance of a pre-trained T5 with lexical substitution augmentation when the amount of training data, N_{train}, progressively increases from a single training data up to 25,000 training data.$N_{augment}$ displays the amount of augmented data incorporated. The totals for this column should match that of N_{train} unless an augmented version cannot be generated for the original training data.

N_{train}	$N_{augment}$	WebNLG			E2E		
		B	R (F1)	M	B	R (F1)	M
1	1	3.71	37.28	9.09	25.82	43.72	41.87
5	4	14.39	37.75	32.42	37.56	37.78	34.26
500	473	33.70	51.46	49.19	56.27	57.89	59.32
1,000	979	31.79	53.94	53.09	56.64	60.55	64.15
5,000	4,961	32.44	55.88	57.16	63.04	65.77	72.89
15,000	14,972	31.72	56.10	57.50	63.36	66.41	72.69
25,000	24,933	32.53	56.67	58.78	63.99	66.54	74.01

Table 4.9: The performance of a pre-trained T5 with paraphrasing augmentation when the amount of training data, N_{train}, progressively increases from a single training data up to 25,000 training data. $N_{augment}$ displays the amount of augmented data incorporated. The totals for this column should match that of N_{train} unless an augmented version cannot be generated for the original training data.

the METEOR score from the previous best by 1.56 which hints at an impact on the training procedure though this is not conclusive.

Lastly, Table 4.9 shows the results for the third augmentation strategy in this experimental set-up - paraphrasing. Again, no obvious pattern is gleaned from this table for WebNLG and only a minor improvement is noted for E2E. Similar to Table 4.8, paraphrasing provides an improvement for E2E with METEOR of 1.23. Once again, this is hardly a significant result and is not conclusive.

4.2.7 Introducing lexical diversity

Table 4.10 highlights that lexical substitution results in a pre-trained T5 producing more unique words throughout the *generated text* than without augmentation. As many of the evaluation metrics employed are computed by comparing words in the *generated text* to

Augmented	WebNLG	E2E
No LS	78,923	13,203
LS	81,822	14,995

Table 4.10: Contains the number of unique tokens that appear in the *generated text* produced by a pre-trained T5 that is fine-tuned without augmentation and one augmented with lexical substitution. No LS: without augmentation, LS: augmented with lexical substitution. The bolded model is the one which produces more unique tokens.

the *target text*, this sheds light on why lexical substitution achieves diminished scores in Table 4.6 with regards to the other augmentation strategies and even the model without augmentation. Regardless, Table 4.6 also demonstrates that the BERTScore value for lexical substitution does not decrease like other metrics suggesting the context and information of the *generated text* does not change.

4.3 Discussion

In this subsection I highlight various interesting or confounding findings observed in the experimental results previously presented. I consider possible explanations for any notable patterns in the results as well as rebuttal for results that conflict with my initial hypotheses. For the following discussion, I include qualitative analysis where pertinent to provide further context. Lastly, I highlight data issues which may work against data augmentation and data-to-text generation.

4.3.1 Data-to-text generation

While both the Transformer and T5 without pre-training provide a good base to build upon, the effect of pre-training is significant and evident across both datasets used in this work. The improvement is most pronounced in WebNLG, the more challenging dataset of the two as seen in Table 4.1. Given what we know from Sections 4.1.8 and 4.1.8 the *target text* is objectively more complex for WebNLG while the *source data* is less granular.

A concern that remains unchecked is that pre-training accounts for the improvements data augmentation would garner on evaluation metrics. Although augmentation may incite benefits beyond the metrics employed.

The Transformer did not perform as strongly on either dataset as I had expected, even declining in both the BLEU and ROUGE metric from the copy-based model which has no mechanism for adding words to the *generated text* that are not in the *source data.*

Although the Transformer architecture is an adaptable architecture that provides a consistent building block for further data-to-text generation models, the easy incorporation of pre-training with language models effectively replaces the demand for customized Transformer models. Pre-training's drastic improvement over the Transformer and T5 without it highlights that generally learning linguistic features alleviates the challenge during training. Instead, the model is able to focus on remaining truthful to the *source data* and ordering it appropriately in the *generated text*. In fact, data augmentation may be better applied to pre-training to further the potential improvement it offers.

Lastly, I did not observe any improvement from curriculum learning when training the Transformer and T5. Despite this being the case for this work, it cannot be generalized to all data-to-text generation applications. The models used and the nature of the datasets likely have an effect on the efficacy of curriculum learning.

4.3.2 Augmentation for data-to-text generation

In both computer vision and text classification data augmentation transforms the input, that is an image or text being passed to the model. For machine translation, a popular augmentation strategy backtranslation transforms the output which is the *target text*. For data-to-text generation it is conceptually intuitive to follow the precedent set by machine translation given that both are sequence-to-sequence tasks, but the inspiration of augmentation strategies from other tasks warranted testing data augmentation to transform the *source data* or *target text*. Table 4.5 shows this comparison where the augmentation strategy used was random erasing. Clearly, augmenting the *target text* is the superior approach, achieving an average improvement of 6.90 on BLEU, 6.27 on ROUGE, and 7.54 on METEOR. This improvement is attributed to data augmentation changing the *source data* which is not reflected in the *target text*, resulting in a mismatch between the information the model thinks the *generated text* should contain and what is contained in the *target text* used for evaluation. While more intricate augmentation strategies exist that leverage both the *source data* and *target text*, in this work the focus remained on strategies affecting the *target text*. Finally, the idea of augmenting the *source data* may be enticing but it is important to recall that between applications the *source data* will vary greatly and augmentation strategies may not always be applicable removing the capability to reproduce results.

Contrary to my hypothesis at the beginning of this work, the three data augmentation strategies studied produce very modest results with minimal improvement to performance as seen in Table 4.6. The exception being that random erasing demonstrates an improvement for E2E on ROUGE and METEOR of 3.66 and 4.81 respectively. I further report BERTScore in Table 4.6 as an evaluation metric which describes the similarity between the embedded representation of the *generated text* and *target text*. While n-gram evaluation metrics such as those reported correlate with human judgement and are important for judging whether the proper terminology is in the *generated text*, that is also one of their pitfalls discussed in Section 2.4.1. BLEU, ROUGE, and METEOR are all penalized when the terminology of the *generated text* does not match that of the *target text* whereas BERTScore does not rely on particular wording rather the context of the text as a whole. For either dataset in Table 4.6 this is the fourth column. Although minimal, for WebNLG all three augmentation strategies improve upon the BERTScore of the pre-trained T5 without augmentation as paraphrasing achieves the best score. For E2E the improvement is even more minor with only random erasing and lexical substitution surpassing the BERTScore of the model without augmentation and the former strategy achieving the best result. The most notable observation pertaining to BERTScore is for WebNLG. When augmenting through lexical substitution the T5 model suffered a 4.93 decrease in BLEU. Despite this notable decline, when augmented with lexical substitution the BERTScore rises by a meager 0.86 which implies that despite different terminology the content contained within the *generated text* is the same. This is observed for E2E as well, just to a lesser degree.

I ran experiments with progressing number of epochs and noted that past 3 or 4 epochs, the augmented models began to perform worse. In these cases, the incorporation of slightly different yet largely the same data caused the model to overfit to the training data. Furthermore overfitting may have been coupled with catastrophic forgetting, a phenomenon pre-trained language models experience when their weights deviate due to too much training. Now, this outlines the setting and circumstances in which data augmentation is useful for pre-trained language models. The question remains, at what point does an augmentation strategy become a hindrance in terms of the time required to train a model. Random erasing was the least computationally expensive augmentation strategy of the three due to not requiring any further models or algorithms for its implementation. To augment a batch with random erasing took 0.00009 seconds for WebNLG and 0.0001 seconds for E2E. Next, lexical substitution took on average 1.06 and 0.88 seconds for WebNLG and E2E respectively. This is explained by the fact that lexical substitution has to sequentially access each token to assess whether a candidate synonym should be included in the augmented copy. Lastly, paraphrasing took on average 2.1 and 1.54 seconds to augment a batch for WebNLG and E2E respectively. Paraphrasing does not require token-level access like lexical substitution, but the external model it uses is more computational complexity hence it taking the longest to augment. On top of model improvements, it is important to take into consideration the time-cost these data augmentation strategies have to weigh which one is best for a given situation.

4.3.3 Random erasing

Random erasing's performance offered the largest boost of the three augmentation strategies, in particular when applied to the E2E dataset. While further experimentation and analysis needs to be done to assert whether random erasing improves performance on WebNLG, it certain that the T5 trained with random erasing is better than without for E2E. That being said, there are indicators that random erasing offers improvements on both. One such observation is in Figure 4.2 where training with random erasing achieves the best loss curve for WebNLG.

As mentioned, random erasing is the least computationally expensive of the three augmentation techniques with a single pass of the *target text* resulting in the augmented copy with subsequences of the characters *erased*. The algorithm for performing random erasing is approximately 10,000 times faster than lexical substitution and 20,000 times faster than paraphrasing. Additionally, the lack of extra models to perform any part of the augmentation means random erasing has the lowest memory cost of the three strategies. While all of the augmentation strategies can be performed in advance, this low memory cost and computational simplicity facilitates its incorporation during the training process.

4.3.4 Lexical substitution

Lexical substitution offers the most inconsistent performance boosts out of the three augmentation strategies. For both WebNLG and E2E, lexical substitution results in BLEU decreasing. Otherwise, the other metrics remain on par with the model trained without augmentation. In fact, for E2E, the lexical substitution augmentation performs better than paraphrasing and the T5 without augmentation by 5.56 and 4.31 respectively on METEOR. Furthermore, Table 4.6 demonstrates that lexical substitution achieves competitive BERTScore values implying the decreas to BLEU might not be all bad and rather hints at an increase in diversity. Lexical substitution also achieves the best loss curve in Figure 4.2 for the E2E dataset.

Lexical substitution was the augmentation strategy that required the most algorithm decisions and parameter setting. While this allows the strategy to be configurable for different datasets, it also requires the most parameter tuning to find satisfactory implementations. Extensive experimenting with the lexical substitution algorithm would likely garner better results though such tuning was not within the scope of this work. In the case of computational concerns, lexical substitution can be made more simple at the detriment of worse performance. This implementation of lexical substitution utilized contextual embeddings to search for synonyms within the neighbourhood of the embedding space for the word to be swapped, but others may make use of Word-Net or some other synonym recommending procedure. To further my implementation of lexical substitution, one could fine-tune the model whose contextual embeddings are extracted from on the current dataset. The issue here is that less diverse synonyms would be provided, more likely resulting in overfitting.

Table 4.10 shows the number of unique tokens in the *generated text* for a model fine-tuned without data augmentation and one fine-tuned on training data that has been augmented with lexical substitution. As seen, lexical substitution results in an increased vocabulary for both datasets. One might argue that the decrease in BLEU displayed in Table 4.6 when using lexical substitution is a result of the model being trained to generate incorrect words yet none of the other metrics show the same decline with some even improving as way of the augmentation. So, if some metrics are saying the augmented model is even better at generating text with correct terminology and the context has not deviated from the *reference text* then there must be an explanation. A difference of 2,899 and 1,792 tokens for WebNLG and E2E respectively between the unaugmented and augmented model is a drastic difference. I conject that the decrease in BLEU for lexical substitution is caused by increased diversity in the tokens used within the *generated text*.

4.3.5 Paraphrasing

Paraphrasing appears to offer a slight yet consistent improvement to the WebNLG dataset as seen in Table 4.6 and a slight yet consistent decline for E2E. Although computationally expensive and the longest time to augment per batch on both datasets, paraphrasing can be easy to implement and holds real-world intuition. The main parameter associated with paraphrasing augmentation is the number of paraphrased copies to generate. Due to the simplicity and length of the *target text* in WebNLG and E2E as well as the size of the datasets, I only produced one augmented copy per training data. Further tuning of when and how many augmented copies are produced could lead to better results. Exploration into configurable paraphrasing models may require less augmented data to observe improved results as well.

4.3.6 Augmentation on reduced training data

At first glance, the results of Tables 4.4, 4.7, 4.8, and 4.9 are almost indistinguishably equal. No consistent pattern is evident with seemingly random improvements and declines in performance occurring for smaller values of N_{train} while larger values appear to reach the same threshold in performance, akin to the best results displayed in Table 4.3. In Section 4.2.6, I highlighted when any of the three augmentation strategies obtained better results or the results for each value of N_{train} converged more quickly than the model without augmentation.

Focusing on an aspect of the latter, I did observe an interesting pattern. For values of $N_{train} = 500, 1,000, 5,000$ there was more consistent improvement over the pre-trained T5 without augmentation results. On these dataset sizes the augmentation strategies performed as follows. Random erasing demonstrated minor improvements, but across both datasets nonetheless. For the previously stated values of N_{train}, on average, random erasing improved BLEU by 0.2, ROUGE by 0.75, and METEOR by 0.37 for WebNLG. For E2E, BLEU improved by 1.65, ROUGE by 0.2, and METEOR by 1.04. As one might expect, lexical substitution provided no improvement on BLEU for either dataset and surprisingly no metric for E2E. On WebNLG, lexical substitution garnered an improvement on ROUGE of 0.96 and on METEOR of only 0.73. Lastly, paraphrasing achieves improvements on average for $N_{train} = 500, 1,000, 5,000$ with the following metrics. For WebNLG on BLEU by 1.63, ROUGE by 0.69, and METEOR by 1.14. For E2E there was only an improvement on BLEU which was by 1.28. While unexpectedly augmentation did not show consistent or meaningful improvements for all values of N_{train}, when focusing on a subset of these values it is observed that there is a range of dataset sizes where augmentation is consistently beneficial. In particular, random erasing which boosted results across all metrics. For applications of data-to-text generation where training data is very scarce, it is likely that data augmentation improves the model whether by incorporating more training data or simply providing enough diversity to avoid overfitting to the few original training data.

4.4 Comparison to Augmentation for Natural Language Processing

The improvement to results provided by data augmentation in this thesis are distinctly less profound than in CV and even other natural language tasks such as text classification and machine translation. For example, the authors of [13] and [100] consistently observe improved performance from token-level augmentation such random erasing on supervised learning tasks for natural language processing (NLP). Though, it is stated that on more difficult tasks this improvement tends to drop.

I believe that augmentation for data-to-text generation in its current state is less effective than it is for counterpart tasks due to its inherent complexity. For text classification, the input is always text regardless of the domain it is extracted from. Whereas the *source data* for data-to-text generation can have vastly different structures, for example, a sequence of numeric values cannot be treated the same as a collection of edges and nodes from a graph. In fact, the data between WebNLG and E2E could be tackled in different ways but for comparability I chose to use the T5 model as it could handle both. I argue this point of view because: a) the number of applicable is almost certainly less than the possible verbalizations for a given *source data* and b) minor transformations to text may not result in a label change, but if we consider the difference between "It is raining" and "It is downpouring" in the context of summarizing weather data there is a huge difference.

4.5 Considerations

As is the case with any machine learning work, there are a myriad of considerations to take into account. In a perfect world, biased data would not exist but as the authors of [52] note this is not the case and most often has social implications.

In practice, datasets can be screened prior to use for bias and the problematic data removed. The nature in which augmented data is included, typically with a factor of randomness, its possible that data augmentation introduces biases into the data as well. Not all biases have such an extreme implication, for example augmentation can simply affect the disposition of text. Consider data augmentation via paraphrasing. In my implementation this utilizes an external model that was trained on other data with its own style and vocabulary. Passing the text from WebNLG and E2E might result in paraphrased data that utilizes different words or even sentence structure. Similarly, the lexical substitution implementation utilizes BERT's contextual embeddings which were trained on data separate from either of the datasets in this work. Substituted words might not be contextually relevant in this scenario, be overly formal, or any other number of issues. This could explain why of the three augmentation strategies, random erasing is the most consistent despite being the least computationally expensive and from an implementation point of view the simplest.

4.6 Summary

In the Analysis, I assessed a pre-trained language model's ability to be adapted for data-to-text generation and the benefits it provides. I further evaluated how three data augmentation strategies complemented this model for the task at hand and the effects or lack thereof that were exhibited. From this, I constructed an understanding of how these augmentation strategies affect evaluation metrics such as BLEU, ROUGE, METEOR, and BERTScore. Furthermore, I analyzed whether augmentation played a role in improving the model's performance when encountering small or incomplete datasets.

In the remaining chapter I summarize the conclusions of this work on data-to-text generation and data augmentation, provide final remarks, and outline possible avenues at this intersection of topics that future work could pursue.

Chapter 5

Conclusions

5.1 Summary

This thesis demonstrated the T5's capability for data-to-text generation and its ability to be improved upon with data augmentation. The three augmentation strategies used in this work were random erasing, lexical substitution, and paraphrasing. Although the results are modest, random erasing showed the most consistent improvement on evaluation metrics while retaining the original context of the *generated text* followed by paraphrasing. Across experiments, random erasing performed better than the model with no data augmentation on both WebNLG and E2E at times. Paraphrasing performed well, but was computationally expensive whereas random erasing was the least expensive in terms of time and memory. Lexical substitution was not only computationally expensive, but also performed worse than the model trained without augmentation. Summarizing, 32.78 is the best BLEU score achieved on WebNLG which occurs when using all of the training data and doubling it with paraphrased augmented copies while 65.79 is the maximum score for E2E and is obtained when only 25,000 of the training data is used and then doubled using random erasing.

As hypothesized, data-to-text generation does not benefit from augmenting the *source data*. That is, the applicable data augmentation strategy studied for its effect on the *source data* versus the *target text* did not. Augmentation that transforms the content of the *source data* theoretically can diminish the *correctness* of the corresponding *target text*.

With the exception of ROUGE for WebNLG and BLEU for E2E, improvements were achieved via augmentation on both datasets. In general, WebNLG was viewed as the more difficult task and this was expressed with lower results than obtained for E2E. Augmentation on E2E achieved greater margins of improvement than on WebNLG, reinforcing the discrepancy between the two tasks. As expected, a more difficult task benefits less from data augmentation although improvements can still be obtained.

Evidently when trained on less data the model's performance worsens. Incorporating data augmentation, there is not a clear observable relationship between the amount of training data and an improvement on evaluation metrics. For datasets of size 500, 1,000, and 5,000 it appears that random erasing and paraphrasing marginally improve model performance, but further datasets are likely required to confidently conclude whether this is true.

Lexical substitution was a computationally expensive augmentation strategy and produced the least consistent improvements - at times doing more harm than good. Yet, it is likely that the drastic nature in which it substituted words within the text resulted in the most diverse augmented data. In particular, this had ill effects on BLEU for both datasets. Lexical substitution did result in 2,899 and 1,792 more unique tokens in the *generated text* for WebNLG and E2E respectively. This expanded vocabulary by the augmented model is not erroneous terminology either, just different wording for the same context. From a results standpoint, lexical substitution falls short of random erasing and paraphrasing, but appears to add diversity to the model's vocabulary. Better evaluation approaches may help conclude augmentation strategies that go beyond evaluation metrics, influencing the characteristics of a model's *generated text*.

5.2 Future Work

As discussed through Chapter 4, a major decision of data augmentation is deciding how much augmented data should be included in the training process. While figures in Sections 4.2.3 and 4.2.6 hint at what this optimal value may be, they are only conclusive for WebNLG and E2E and loosely so. Treating the amount of augmented data as a hyperparameter to be tuned is equally as time-consuming. The amount of augmentation a dataset can withstand before it loses its integrity and ground-truth distribution depends on factors such as its size, the complexity, the model applied, and the strategy itself. For example, a small dataset can only incorporate so much augmented data before a model will begin to learn the shifted distribution. Future work investigating functions, metrics, or approaches that approximate how much augmented data a dataset can withstand before being corrupted would be highly valuable in facilitating reproducible data augmentation research.

In this work, I demonstrated the performance of two models the Transformer and T5 language model on data-to-text generation and the impact of data augmentation by incorporating three different strategies. Only more can be learnt about the intersection of data-to-text generation and data augmentation if more models, datasets, and strategies are experimentally evaluated with. Data-to-text generation is less popular than other natural language tasks and so wide-spanning research such as this will continue to provide more coverage and a further understanding of the task. Another avenue of data-to-text generation are encoder-decoder models with external mechanisms for addressing usual issues in the *generated text*. Porting data augmentation to these models could open up more accessible research than the computationally expensive language models. Data-to-text generation is sorely lacking a comprehensive survey of data augmentation for the task in comparison to its counterparts such as text classification or computer vision.

In Section 4.2.7, Table 4.10 demonstrated that by augmenting with lexical substitution the *generated text* produced upwards of thousands new tokens than without the strategy. Effects such as these go unseen within the results as lexical substitution did not perform worse on metrics such as ROUGE and METEOR. This begs the question if augmentation does and can do more than just prevent overfitting. If lexical substitution increases the number of tokens produced then what impact does paraphrasing have on the characteristics of the text. A valuable avenue for future research could be exploring whether augmentation can be used to impose desired characteristics onto the *generated text* such as a change in writing style. In doing so, the model could be prevented from overfitting while simultaneously generating text in a different manner.

The nature of data-to-text generation and the training data required means tailored data augmentation strategies such as those that exist for text classification or computer vision do not exist. Rather than adopting strategies presented for other tasks, embedding-based augmentation strategies should be researched. This would prevent erroneous data that invalidates the *source data* or *target text* from being produced by augmentation strategies as it would work on a lower level. Research on these augmentation strategies is growing more popular in other tasks such as neural machine translation, another sequence-to-sequence task [38]. Future research may wish to follow their lead.

Lastly, data-to-text generation lacks real-world datasets that are publicly available. This work has shown that a combination of language models and data augmentation obtains satisfactory performance on data-to-text generation datasets even when there are as few as thousands of training data. Requiring such few training data means these methods can be used to address real-world data with less and less domain specific tailoring. Challenges associated with real-world data would then become the focus of research and how to handle more nuanced relationships between the *source data* and *target text*. Models and data augmentation strategies can continue to be developed and evaluated against benchmark datasets such as those used in this work, but without diversity in experimentation many issues may continue to go unnoticed. Despite requiring collaboration between natural language generation researchers and domain experts, there are numerous domains as mentioned in Chapter 1 where data-to-text generation could dutifully be applied and lend a profound impact such as conservation monitoring, business decision-making, and weather reporting.

Bibliography

[1] James A Anderson. *An introduction to neural networks.* MIT press, 1995.

[2] Simran Arora, Avner May, Jian Zhang, and Christopher Ré. Contextual embeddings: When are they worth it? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2650–2663, Online, July 2020. Association for Computational Linguistics.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, pages 722–735. Springer, 2007.

[4] Hessam Bagherinezhad, Maxwell Horton, Mohammad Rastegari, and Ali Farhadi. Label refinery: Improving imagenet classification through label progression. *arXiv preprint arXiv:1805.02641*, 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186, 2013.

[7] Yonatan Belinkov and James Glass. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019.

[8] Steffen Bickel. Ecml-pkdd discovery challenge 2006 overview. In *ECML-PKDD Discovery Challenge Workshop*, pages 1–9, 2006.

[9] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[10] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and Daniel Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv preprint arXiv:1810.10863*, 2018.

[11] Ernie Chang, Xiaoyu Shen, Dawei Zhu, Vera Demberg, and Hui Su. Neural data-to-text generation with lm-based text augmentation. *arXiv preprint arXiv:2102.03556*, 2021.

[12] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[13] Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. An Empirical Survey of Data Augmentation for Limited Data Learning in NLP. *Transactions of the Association for Computational Linguistics*, 11:191–211, 03 2023.

[14] Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. Kgpt: Knowledge-grounded pre-training for data-to-text generation. *arXiv preprint arXiv:2010.02307*, 2020.

[15] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[16] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Chuong B Do and Andrew Y Ng. Transfer learning for text classification. *Advances in neural information processing systems*, 18, 2005.

[20] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[21] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160. PMLR, 2009.

[22] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

[23] Thiago Castro Ferreira, Chris van der Lee, Emiel Van Miltenburg, and Emiel Krahmer. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. *arXiv preprint arXiv:1908.09022*, 2019.

[24] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[25] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.

[26] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, 2017.

[27] Albert Gatt and Emiel Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.

[28] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[29] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53, 1994.

[30] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[31] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1):1–309, 2017.

[32] Li Gong, Josep M Crego, and Jean Senellart. Enhanced transformer model for data-to-text generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 148–156, 2019.

[33] Hiroaki Hayashi, Yusuke Oda, Alexandra Birch, Ioannis Konstas, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Katsuhito Sudoh. Findings of the third workshop on neural generation and translation. *arXiv preprint arXiv:1910.13299*, 2019.

[34] Alex Hernández-García and Peter König. Data augmentation instead of explicit regularization. *arXiv preprint arXiv:1806.03852*, 2018.

[35] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[37] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[38] Nishant Kambhatla, Logan Born, and Anoop Sarkar. Cipherdaug: Ciphertext based data augmentation for neural machine translation, 2022.

[39] Zdeněk Kasner and Ondřej Dušek. Neural pipeline for zero-shot data-to-text generation. *arXiv preprint arXiv:2203.16279*, 2022.

[40] Chris Kedzie and Kathleen McKeown. A good sample is hard to find: Noise injection sampling and self-training for neural language generation models. *arXiv preprint arXiv:1911.03373*, 2019.

[41] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201*, 2018.

[42] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83. Citeseer, 1996.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[44] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[45] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[46] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[47] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[48] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[49] Yupian Lin, Tong Ruan, Jingping Liu, and Haofen Wang. A survey on neural data-to-text generation. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2023.

[50] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[51] Durjoy Sen Maitra, Ujjwal Bhattacharya, and Swapan K Parui. Cnn based common approach to handwritten character recognition of multiple scripts. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 1021–1025. IEEE, 2015.

[52] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35, 2021.

[53] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[54] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[55] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[56] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[57] Martin Molina, Amanda Stent, and Enrique Parodi. Generating automated news to explain the meaning of sensor data. In *Advances in Intelligent Data Analysis X: 10th International Symposium, IDA 2011, Porto, Portugal, October 29-31, 2011. Proceedings 10*, pages 282–293. Springer, 2011.

[58] Sebastien Montella, Betty Fabre, Tanguy Urvoy, Johannes Heinecke, and Lina Rojas-Barahona. Denoising pre-training and data augmentation strategies for enhanced RDF verbalization with transformers. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 89–99, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics.

[59] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[60] Francisco J Moreno-Barea, Fiammetta Strazzera, José M Jerez, Daniel Urda, and Leonardo Franco. Forward noise adjustment scheme for data augmentation. In *2018 IEEE symposium series on computational intelligence (SSCI)*, pages 728–734. IEEE, 2018.

[61] Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. Dart: Open-domain structured data record to text generation. *arXiv preprint arXiv:2007.02871*, 2020.

[62] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, 2018.

[63] Yuyang Nie, Yuanhe Tian, Xiang Wan, Yan Song, and Bo Dai. Named entity recognition for social media texts with semantic augmentation. *arXiv preprint arXiv:2010.15458*, 2020.

[64] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*, 2017.

[65] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[66] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[67] Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. Totto: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*, 2020.

[68] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[69] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[70] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[71] Vassilis Plachouras, Charese Smiley, Hiroko Bretz, Ola Taylor, Jochen L Leidner, Dezhao Song, and Frank Schilder. Interacting with financial data using natural language. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1121–1124, 2016.

[72] Kapila Ponnamperuma, Advaith Siddharthan, Cheng Zeng, Chris Mellish, and René Van Der Wal. Tag2blog: Narrative generation from satellite tag data. In *Proceedings of the 51st annual meeting of the association for computational linguistics: System demonstrations*, pages 169–174, 2013.

[73] Ratish Puduppully, Li Dong, and Mirella Lapata. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6908–6915, 2019.

[74] Ratish Puduppully, Li Dong, and Mirella Lapata. Data-to-text generation with entity modeling. *arXiv preprint arXiv:1906.03221*, 2019.

[75] Ratish Puduppully, Yao Fu, and Mirella Lapata. Data-to-text generation with variational sequential planning. *Transactions of the Association for Computational Linguistics*, 10:697–715, 2022.

[76] Ratish Puduppully and Mirella Lapata. Data-to-text generation with macro planning. *Transactions of the Association for Computational Linguistics*, 9:510–527, 2021.

[77] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[78] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[79] Alejandro Ramos-Soto, Alberto Jose Bugarin, Senén Barro, and Juan Taboada. Linguistic descriptions for automatic generation of textual short-term weather forecasts on real prediction data. *IEEE Transactions on Fuzzy Systems*, 23(1):44–57, 2014.

[80] Clément Rebuffel, Laure Soulier, Geoffrey Scoutheeten, and Patrick Gallinari. A hierarchical model for data-to-text generation. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part I 42*, pages 65–80. Springer, 2020.

[81] Ehud Reiter. Pipelines and size constraints. *Computational Linguistics*, 26(2):251–259, 2000.

[82] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[83] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167(1-2):137–169, 2005.

[84] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

[85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[86] Rico Sennrich, Barry Haddow, and Alexandra Birch. Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*, 2016.

[87] Anastasia Shimorina, Elena Khasanova, and Claire Gardent. Creating a corpus for russian data-to-text generation using neural machine translation and post-editing. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 44–49, 2019.

[88] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[89] Advaith Siddharthan, Matthew J Green, Kees van Deemter, Chris Mellish, and Rene Van Der Wal. Blogging birds: Generating narratives about reintroduced species to promote public engagement. In *INLG 2012 Proceedings of the Seventh International Natural Language Generation Conference*, pages 120–124, 2012.

[90] Michael Sintek and Stefan Decker. Triple—a query, inference, and transformation language for the semantic web. In *International semantic web conference*, pages 364–378. Springer, 2002.

[91] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. *Advances in neural information processing systems*, 26, 2013.

[92] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[93] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[94] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, 2019.

[95] Ross Turner, Somayajulu Sripada, Ehud Reiter, and Ian P Davy. Selecting the content of textual descriptions of geographically located events in spatio-temporal weather data. In *Applications and Innovations in Intelligent Systems XV: Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 75–88. Springer, 2008.

[96] Chris van der Lee, Thiago Castro Ferreira, Chris Emmery, Travis Wiltshire, and Emiel Krahmer. Neural data-to-text generation based on small datasets: Comparing the added value of two semi-supervised learning approaches on top of a large language model. *arXiv preprint arXiv:2207.06839*, 2022.

[97] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[98] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[99] Xinyi Wang, Hieu Pham, Zihang Dai, and Graham Neubig. Switchout: an efficient data augmentation algorithm for neural machine translation. *arXiv preprint arXiv:1808.07512*, 2018.

[100] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[101] Sam Wiseman, Stuart M Shieber, and Alexander M Rush. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*, 2017.

[102] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[103] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 7(8):4, 2015.

[104] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[105] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

[106] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International journal of machine learning and cybernetics*, 1:43–52, 2010.

[107] Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. Bridging the structural gap between encoding and decoding for data-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2481–2491, 2020.

[108] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[109] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.

[110] Wangchunshu Zhou, Tao Ge, Ke Xu, Furu Wei, and Ming Zhou. Bert-based lexical substitution. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3368–3373, 2019.

[111] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*, 2015.