Future Sight: Dynamic Story Generation with Large Pretrained Language Models

by

Brian Zimmerman

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Automated story generation has been an open problem in computing for many decades. Only with the recent wave of deep learning research have neural networks been applied to automated story generation tasks. Current deep learning agents for automated story generation typically ingest a prompt or storyline on which to condition generated text. This approach lacks the dynamism to include elements of a story only decided by the model during inference.

We build an interactive system using pretrained transformers finetuned on a novel objective to temporally interpolate between a story context c and an future plot event f. At inference time, users can suggest future plot events along with a distance, in sentences, to coerce a transformer decoder towards generating sentences that would both remain consistent with a story context and logically conclude with the future event.

The results of our experiments demonstrate that there is a notion of adherence to both context and future in *some*, but not all, cases. We discuss in detail potential explanations as to why the model fails to condition on some contexts and futures with respect to the data and the parameters of our model. We include examples sampled from our model to motivate this discussion.

Acknowledgements

I would first like to thank my supervisor and mentor Dr. Olga Vechtomova for being willing to take a chance on me when I first showed up in Waterloo. Dr. Vechtomova has been very accommodating in allowing me to explore different areas of research while providing feedback at every turn and I look forward to continuing to grow under her guidance. I'm incredibly grateful to work with someone that sees the potential of an academic within me, because I struggle to see it myself sometimes.

I'm very grateful to each of my readers, Dr. Jesse Hoey and Dr. Pascal Poupart, for agreeing to read my thesis on such short notice.

I will never be able to thank Kateryna Morayko enough for what she's done for me. She is the one who first convinced me that I would make the cut as a graduate student. She was also sitting right next to me when I first applied to University of Waterloo. She's my best friend and I hope I always have the honor and privilege of having her in my life, regardless of where we stand.

There are several members of my lab group who I would like to thank as well. The first among them is Gaurav Sahu, my first point of contact here. Even though I'm older than him, he regularly provides me with his sagely wisdom when I'm distraught.

Additionally, I'd like to thank my previous labmates Utsav Das and Egill Ian Gudmundsson. In a way, it felt like we were in the same cohort of Dr. Vechtomova's pandemic master's students. I'll never forget defying the provincial lockdown to eat focaccia or to watch Ian jump into Columbia Lake through a hole in the ice.

I would like to thank Ridhee Gupta for being a really great friend. She dragged me out of my office regularly to get work done in some more interesting places. She also introduced me to salsa, which I promise to attend more regularly next term.

Lastly, but not leastly, I would like to thank my mother and my sister for always supporting my wild endeavors. Ontario didn't seem so far away until a pandemic happened and the border shut down indefinitely. I'm sorry about that.

Dedication

This thesis is dedicated to Norman, who was a good boy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Storytelling

For humans, storytelling is an essential modality of linguistic expression. Stories involve the management of many simultaneous personae, events, and emotions which are often drawn from experience. Stories are not built from just any assortment of words, they must be carefully crafted. They should maintain a minimum threshold of consistency in order to remain immersive and convincing. Thematic concepts such as locations, events, and general phenomena should abide by any precedent previously established by the story. For example, if a character in a story dies, they should not be referenced as alive in the present without an event or phenomenon indicating their revival. Automating this process is nontrivial because to generalize storytelling would be to generalize human experience. By many in the Natural Language Processing (NLP) world, the automation of storytelling is considered yet another step towards true artificial intelligence.

1.1.2 Advances in Generative Language Models

Generative language models have experienced a recent burgeoning in their utility in part due to the success of attention-based language modelling techniques. With transformer networks[38], generative language models shifted away from recursive architectures[2, 6, 14], almost entirely, in favor of attention. One caveat to transformer networks is their

inaccessibility. GPT-2[29] and BERT[9] have over 100 million parameters which need to be optimized during training and even newer transformers have parameter counts in the order of billions. With such a high amount of parameters, training a transformer derived network from scratch is prohibitively expensive and time-consuming for many to train and deploy for their personal use. Groups dedicated to the proliferation of transformer networks, such as Huggingface[41], have begun publishing their optimized model parameters online for others to download and use. These models are trained on general tasks with publicly available datasets in a process known as pretraining. End-users can download these parameters and *finetune* them on more specific downstream tasks of their choosing.

Though large pretrained language models have recently become more accessible, some tasks are not accounted for during pretraining. For example, GPT-2 was designed and trained specifically for the task of *next word prediction* and cannot accommodate any additional information like style or sentiment at inference time. Previous works have made an effort to introduce additional information to these models but either require pretraining from scratch entirely or compromising the integrity of the model by severely misaligning the pretrained weights.

1.2 Problem Definition

Prior methods of automated story generation approach it as a two step hierarchical process: first planning and subsequently writing while conditioning on the story plan. As humans tell stories, new plot elements can be introduced dynamically, at any time, and be heavily influenced on elements of the story that precede them. By first planning and then writing, it's difficult to incorporate elements of the previously generated text that weren't explicitly planned for into future events at any point in the generative process. Our approach augments a pretrained transformer decoder in an effort to coerce it towards a future plot element. We anticipate that the ability to interpolate from a context to a future plot event will inspire human authors in writing their own stories.

Pretrained transformer decoders, such as GPT-2, are limited in their ability to be controlled at inference time because at training time they are only parameterized with a single input sequence to be reconstructed. Previous works have attempted to enforce a notion of controllability on pretrained transformers to some success but only over broad domains such as style or sentiment. Conditioning a transformer decoder on a plot element in a story is a nontrivial task because plot elements are often unique and only fit within the context of one story. By contrast, style and sentiment often have many examples to represent them within a given dataset. Our work demonstrates the ability to condition

a pretrained GPT-2 on complete sentences containing future plot information. To our knowledge, our work is a first attempt at dynamically guiding a pretrained transformer on a story generation task with plot events provided as complete sentences. We will refer to this task going forward as *dynamic story generation*.

1.3 Contributions

In this work, we propose a novel configuration of pretrained transformers for dynamic story generation. Our contributions are as follows:

- We inject a nonlinear transformation of a BERT encoded plot event to timestep *t0* of a pretrained GPT-2 and report the results.
- We use idf-mean as a heuristic for selecting more highly informative future plot events from the set of sentences following a story context during our preprocessing step.
- We analyze the effects of context information on controllability through a combination of masking and variable context length.

Chapter 2

Background

2.1 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence that describes the process of using a computer to learn an approximation of a function or distribution of a dataset. Machine learning pipelines can generally be split into two core phases: training and inference. Training, sometimes referred to as fitting, describes the process of using points from a sampled *training set* to estimate the parameters of a function over the population from which the training set was drawn. Inference is to then use the estimated parameters to make predictions about unseen data presumed to be from the same population. Functions of the data with a discrete range are referred to as classifiers while functions with a continuous range are referred to as regression models. Classifiers are commonly used for tasks in NLP which require the generation of text as vocabularies are often modelled as discrete distributions.

2.1.1 Supervised Learning

Supervised learning is the process of teaching a machine learning model an alignment between source and target data by training over explicitly labelled data. During training, the model makes multiple iterations over the training set, attempting to predict the *ground truth* label for each data point several times. The offset between each predicted value and its respective ground truth label is used during backpropagation to update each parameter of the model. With each iteration over the data, the model parameters are coerced towards

those of the true underlying function of the data. Supervised learning is only possible where there is data that has been explicitly labelled, a process that can be costly and time consuming to perform.

Machine translation is one application within NLP that often relies on supervised learning to learn an alignment between a source language and a target language. Data points in machine translation tasks generally consist of an input sequence in the source language and a ground truth sequence in the target language. As the model trains, it will more and more closely approximate an alignment function between all possible sequences of a source language and their respective translated sequences in the target language.

2.1.2 Unsupervised Learning

Unlike in supervised learning, unsupervised learning techniques lack explicit labelling and work with only an input to learn rich representations of the data. Strategies such as data noising, adversarial training, and autoencoding allow a model to identify discernible features of a dataset and build a space in which similar points will be proximal with one another without any additional information required.

GPT-2 [29], a core component of our proposed architecture, is an example of how powerful unsupervised learning can be. The training procedure used by Radford *et al* uses the source sequence as the target sequence with an offset of one token position. This offset aligned each word in the source sequence with each subsequent word in the target sequence allowing GPT-2 to function as a powerful *next word prediction* model without any explicit labelling.

2.2 Deep Learning

With the recent proliferation of dedicated Graphics Processing Units (GPUs), machine learning has largely been dominated by deeper, more robust data modelling techniques referred to as *deep learning*. The word deep refers to a special class of models which learn compositions of functions that can in turn expose complex patterns across various aggregations of the data. These models are referred to as Artificial Neural Networks (ANNs).

2.2.1 Artificial Neural Networks

ANNs gain their namesake from our perception of how the human brain works. Biological neural networks are understood to be a series of interconnected nodes, called neurons, that can elicit changes in other neurons in the network with some probability. This process was first modelled as an artificial mechanism by Rosenblatt who referred to his implementation as *perceptron* [32]. Perceptron is a binary classification model that approximates the coefficients W of a linear function of the inputs x. Combined with a bias term b, perceptron could learn any linearly separable pattern across features of a dataset. This concept is guaranteed by *Perceptron Convergence Theorem* permitting that the data is indeed linearly separable.

2.2.2 Multilayer Perceptron

It was the inability of perceptron to model complex nonlinear patterns that delayed the widespread adoption of artificial neural networks. In spite of perceptron's shortcomings, some remained optimistic that perceptrons could be layered to model a wider domain of functions once there were stronger convergence theorems available to support such architectures [22]. Almost three decades after Rosenblatt's perceptron, Rumelhart *et al* proposed *backpropagation*, a technique for propagating error backwards through an ANN [33]. Backpropagation enabled multiple perceptrons to be connected in series for the purpose of modelling more complex functions of data, an architecture called Multilayer Perceptron (MLP).

The objective while training an ANN on almost any task is to minimize the distance between predicted values and their ground truth counterparts. These values can be compared using a metric known as a *loss function*. The most general purpose loss functions compute the distance between two points in space. The derivative, or *gradient*, of a loss function, denotes the rate of change with respect to one weight in the previous layer. By taking the gradient with respect to each weight in the previous layer, the weights themselves can be updated accordingly to decrease the distance between the predicted and ground truth values. Backpropagation expands on this concept using the *chain rule of ordered derivatives* to accumulate gradients backwards through multilayer networks. By using nonlinear *activation functions* between each layer, multilayer perceptrons could learn features with which to parameterize subsequent layers and extract rich representations of the input data in the process.

Multilayer perceptrons excel at learning complex nonlinear patterns. Their usage is still common today as a smaller component of larger architectures to extract features from non-

sequential data. Though MLPs can be applied to sequential data, they struggle to model complex relationships encoded in the data sequencing due to the lack of ordinal precedence in the input nodes. Moreover, the number of input nodes is finite and preordained, placing a hard cap on the length of sequences to be modelled.

In the context of time series data, this meant that only limited windows into the past could be modelled and it was difficult to discern trends over time. Some tasks in NLP, such as *coreference resolution*, are extremely dependent on context to determine where an object is referenced across multiple parts of a passage. It wasn't without a new type of ANN that sequential data could be shared with other nodes.

2.2.3 Recurrent Neural Network

The concept of an infinite cyclical network became feasible with the concept of backpropagation. In theory, backpropagation could traverse backwards through a network of infinite length so long as the network was composed of continuous functions. Rumelhart *et al* discussed this idea in detail, referring to it as a recurrent net [33]. Recurrent nets are now also known as Recurrent Neural Networks (RNN).

RNNs contain one layer of weights through which sequential data of indefinite length can be encoded. Each position index in a sequential datapoint is referred to as a *timestep*. The data at each timestep is encoded by the recurrent net to produce a vector known as a *hidden state*. Combining a hidden state with the input for a subsequent timestep allows the network to produce an updated hidden state representation of all input encoded so far. As a representation of partially encoded sequential data, the hidden state is serves as a memory mechanism for any sequentially encoded patterns. The final hidden state produced by an RNN over a sequence is effectively a vectorized representation of that sequence. These vectorized representations are also known as *sequence embeddings*. Unlike MLP, which is composed of multiple layers of weights, the recurrent net only has one layer of weights to update during training. Backpropagation in an RNN traverses the gradient signal over each timestep rather than each finite layer. This process is referred to as Backpropagation Through Time (BPTT) [39].

Despite the ability for RNNs to create robust representations of sequential data, in their original form they were not without their own shortcomings. Bengio *et al* exposed a caveat in BPTT when attempting to model long sequences of data with RNNs [3]. When the gradient at timestep $t+1$ is smaller than the gradient at timestep t, the signal approaches zero as the number of timesteps increases in what is known as the *vanishing*

gradient problem. Similarly, the *exploding gradient* problem is when the gradient rapidly increases in the other direction.

Long Short-Term Memory

Long Short-Term Memory (LSTM) networks attempted to address this by introducing a memory mechanism to the traditional RNN architecture and allowing information to flow more stably [14]. Functionally, LSTMs work the same as traditional RNNs by accepting an input x and a hidden state h_{t-1} and returning a hidden state h to propagate forward to the next timestep. Additionally the LSTM *cell*, referring to the core network, produces a cell state c_t which contains long term sequential information to be propagated to the next time step as well. Information is added and removed from the cell state through a combination of trainable gates. These gates are the input gate i_t, the forget gate f_t, and the output gate o_t.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.1}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.2}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.3}$$

LSTM also contains a trainable projection $\hat{c}$ by which to scale the information contained in the input gate i_t. The forget gate f_t determines which information in the previous timestep's cell state c_{t-1} should be withheld from c_t. Once the new cell state c_t is created, it can be combined with o_t to produce the new hidden state h_t. From there, both c_t and h_t can be passed to the next timestep.

$$\hat{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.4}$$

$$c_t = (f_t \cdot c_{t-1}) + (i_t \cdot \hat{c}_t) \tag{2.5}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{2.6}$$

Gated Recurrent Unit

In 2014, Cho *et al* proposed the Gated Recurrent Unit (GRU) as a simpler alternative to the LSTM [6]. Unlike the LSTM, which passes both a hidden state and a cell state to successive timesteps, GRU only produces a hidden state. GRU lacks an output gate, instead updating the hidden state directly via forget and input gates. With fewer trainable parameters, GRU is not only quicker to train than LSTM, but also outperforms it on certain tasks [7].

2.2.4 Sequence to Sequence

Though LSTMs proved to be a powerful method for encoding sequential data for downstream tasks, they only became more ubiquitous when Sutskever *et al* proposed the sequence to sequence (Seq2Seq) architecture [37]. Seq2Seq describes any model arrangement which encodes a sequence and then constructs a new sequence by learning an alignment between the input and the output. Through modelling an output sequence as a function of the input sequence, Seq2Seq is considered a benchmark for tasks in natural language generation such as machine translation and story generation. Seq2Seq architectures consist of two components: the *encoder* and the *decoder*. Sutskever *et al* use a separate LSTM for each component. In practice, the encoder can be any model that creates a hidden representation from a sequence while the decoder can be any model that constructs a sequence from one or many representations.

The encoder is responsible for sequentially encoding the input into a hidden representation. Though hidden representations are effectively produced at each timestep with LSTM, the final hidden state is considered to be the representation of the input sequence in its entirety.

The decoder, at each timestep, produces a distribution from which a candidate token for the output sequence can be sampled. By combining the hidden representation from the encoder with the input at each timestep, the decoder can account for the input in the parameterization of the output distribution. The first input token for the decoder is typically a special token denoting the beginning of a sequence. Input for a subsequent timestep t is the token sampled from the output distribution at timestep $t-1$.

Teacher Forcing

The ground truth token from timestep $t-1$ can be supplied to the decoder at timestep t as an alternative to the token sampled from the output distribution during training. This

process is known as *teacher forcing* [40]. Teacher forcing allows a recurrent network to converge more quickly at the cost of harming its ability to generalize at inference time [18].

2.2.5 Input Embeddings

Words are a core component of any language. They can be ordered and combined to form grammatical components which can be further combined to form sentences. Words are, in essence, a discrete set of tokens which define the domain of a language. Language modelling techniques predicated on neural networks rely on converging on some continuous function of language. Naturally, this is incompatible with words as they belong to a discrete set with no inherent notion of continuity between them. To become machine readable, words must be mapped to a continuous space via distinct feature vectors known as *word embeddings* before they can be processed as part of a sequence. Word embeddings can be constructed in many different ways.

One Hot Embeddings

The most rudimentary technique for vectorizing words is through the construction of *one hot embeddings*. One hot embeddings are *n-dimensional* vectors where n represents the number of distinct words in the corpus. As the corpus is preprocessed, each distinct word is assigned an index i. Using one-hot encoding, the vector representation for word w_i is an n-dimesional zero vector with the value 1 at index i.

One hot embeddings are a weak feature set modelling only the cardinality of each word in the corpus. Concretely, two different one hot encoded words are exactly the same in each dimension but two: those describing their cardinalities. It is extremely common for words to share semantic attributes such as co-occurence and position that would be completely ignored under a one hot encoding strategy. To motivate this, consider the sentence "My car was inspected by the agent." The context would accomodate words such as *truck* and *vehicle* in lieu of car without compromosing the fluency of the sentence. With one-hot embeddings, this relationship would be up to the downstream language model to discern, detracting from its ability to focus on the actual task.

Another caveat of one hot embeddings is that they use very high dimensional sparse vectors. Large vocabularies can consist of 10^5 unique words or more which adds on many needless computations over zeroes when using one hot embeddings. Due to this, one hot vectors are just too inefficient compared to their dense vector contemporaries when attempting nontrivial language modelling tasks.

Trainable Word Embeddings

An efficient representation of words requires dense vectors that can account for variety of contextual features. The concept of building vector representations for words began in information retrieval where *n-gram* cooccurance was used as a basis for relationships between words in continuous space. An n-gram refers to a series of n consecutive words within a longer sequence of words. Schutze extracted and aggregated 4-grams from five months of New York Times articles to demonstrate that minor nuances such as case and inflection could drastically alter the location of words in a vector-space [35].

Bengio *et al* proposed the idea of learning a vector-space of word representations while simultaneously training a downstream language model [2]. This process, they contend, improves overall model performance because the distribution of words could be encoded into the embedding vector-space. This process of *jointly* learning word embeddings is particularly useful with specialized corpora, such as medical texts, where some words may appear more frequently than they would in general language use.

Pretrained Word Embeddings

While word embeddings can technically be initialized randomly and jointly trained with a downstream model, it is often inefficient to do so. Initialization values for word embeddings should be strategically selected before any further optimization while training for a language modelling task. Using pretrained word embeddings can relieve the model of some of the load in learning efficient representations of words. Word embeddings can either be pretrained on the training set or downloaded from repositories online which pretrain them on neutral datasets.

Mikolov *et al* proposed the Word2Vec algorithms for learning word embeddings from a dataset [24]. Before Word2Vec, there were attempts to use MLPs with non-linear activation functions to compute pretrained word embeddings. Due to limitations in hardware, nonlinear MLPs were not a scalable approach that could be applied to large datasets. The primary contributions of Word2Vec were two techniques of low computational complexity, achieved by abstaining from nonlinear algorithms entirely. The first technique, Continuous Bag-of-Words (CBOW), attempts to predict a word based on surrounding contextual words. Mikolov *et al* report the best results using the surrounding four words on each side. The second technique, Skipgram, does the opposite by using one word to predict surrounding words.

Byte Pair Encoding

Using word embeddings comes with a severe drawback when dealing with rare words. With word embeddings, a common practice is to assign a learnable *out of distribution* embedding to words appearing below a certain frequency. By explicitly assigning a vector to each remaining word, the model is bound to a fixed length vocabulary at inference time. This is particularly detrimental to generative language models because they cannot produce words that are out of distribution of the vocabulary established during preprocessing.

Byte Pair Encoding (BPE)[36] was proposed to address the shortcomings of whole word embeddings. BPE was motivated by the encoding algorithm of the same name proposed by Gage[13] which replaces frequently occurring pairs of bytes in a sequence with a new, otherwise unused byte. Sennrich *et al* use BPE to merge frequently occurring pairs of characters in a word. The algorithm performs a preset number of merges over the most common consecutive subword tokens. After the merging is complete, the learned tokens, including remaining unmerged characters, represent the set of tokens from which any word in the dictionary can be constructed. These tokens are mapped to their own vector representations which can be interpreted by downstream language models.

BPE comes with several advantages over traditional word embeddings and techniques that are similarly predicated on compression algorithms. Because BPE constructs a frequency based list of all subword tokens from which words can be constructed, any number of words can be represented as a result. This allows language models to effectively learn rare words that may have only appeared once or twice in a training corpus. Moreover, by learning to generalize over learned BPE tokens, generative models become *zero-shot* learners. Zero-shot refers to a model's ability to produce words that were not seen during training time.

Word Piece Encoding

BPE is a greedy algorithm that will always take the most frequently occurring pair of subword tokens at each iteration for the next merge candidate. This can lead to conflict when encoding certain words that are composed of many frequently occurring subword tokens. Consider the word *dinner* which could be composed of the subword tokens { *"di", "n", "ner"*} or { *"di", "nn", "er"*}.

Word Piece Encoding (WPE) [34] is a subword encoding technique almost identical to BPE in that it iterively merges subword tokens. The primary difference is that it uses a different metric by which it greedily selects the merge candidate at each iteration. WPE

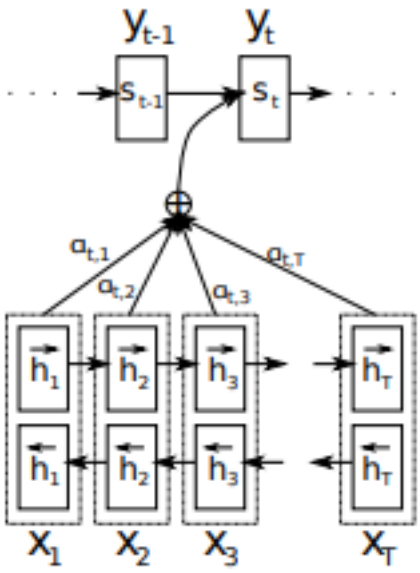

Figure 2.1: Attention mechanism for Recurrent Neural Networks from Bahdanau *et al* [1]

selects the merge candidate based which of the available pairs maximizes the likelihood of the training data. This process introduces additional computational complexity of $O((K_i)^2)$ where K_i is the number of subword tokens at iteration i. Schuster and Nakajima propose methods for decreasing the runtime of the WPE algorithm by heuristically selecting a subset of the K_i subword tokens on which to compute the estimated likelihoods.

2.2.6 Attention

With autoregressive neural networks for modelling sequential data, long term dependencies are shared between timesteps through a fixed-length vector called the hidden state. Processing longer sequences means more information must be compressed within the hidden state, ultimately causing language models to struggle tracking information from timesteps in the distant past. Cho *et al* demonstrated that sequence length and unknown word usage is directly related to the deterioration of autoregressive language model performance [5]. Bahdanau *et al* further note that sequences at inference time that are longer than those in the training set are particularly affected by this [1].

To this end, Bahdanau *et al* proposed a trainable matrix of alignments between output tokens and input tokens which would enable a decoder to directly reference important parts

of an input sequence. Bahdanau *et al* referred to these alignments as annotations but they would eventually become known as *attention*. By allowing the decoder to directly access information at each encoder timestep, it becomes no longer necessary for the fixed-length hidden state to represent all information encoded from the input sequence.

Bahdanau *et al* formulate their approach as a probabilistic next token classifier conditioned on tokens selected at previous timesteps and the input sequence x (2.7). The decoder hidden state at timestep i is represented by s_i. Similar to the hidden state in a standard RNN decoder, s_i is a combination of the previous hidden state s_{i-1} and the token selected at the previous timestep y_{i-1}. Additionally, s_i also contains a context vector ci which is constructed from the hidden states of the encoder. The encoder in Bahdanau *et al* is bidirectional, meaning it reads the input sequence from each direction and concatenates the hidden states generated at each timestep with one another to form $h_j = [\overrightarrow{h_j}; \overleftarrow{h_j}]$.

$$p(y_i|y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \tag{2.7}$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{2.8}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.9}$$

The context c_i at decoder timestep i is a weighted combination of each encoder hidden state h_j. The weights are computed with respect to the previous decoder hidden state s_{i-1} and each encoder hidden state h_j using a Multilayer Perceptron (2.10) and then normalizing the result with the softmax function (2.11). The parameters of the MLP in 2.10 are learned jointly with the parameters of the encoder and the decoder while training on the language modelling task. As the model converges during training, the weights corresponding to the most relevant encoder timesteps grow larger to influence the token distribution at each decoder timestep.

$$e_{ij} = MLP(s_{i-1}, h_j) \tag{2.10}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\Sigma_{k=1}^{T_x} \exp(e_{ik})} \tag{2.11}$$

Before attention, generative language models struggled to maintain consistency over long passages of generated text. Each word, or token, was sampled strictly from a conditional distribution predicated on previously generated tokens in order. As generated sequences grew in length, signals from the oldest tokens would grow increasingly weak and have less influence over the conditional distribution to select the next token. Attention mechanisms, such as the one proposed by Bahdanau *et al* for neural machine translation, would become the basis of most language modelling techniques predicated on deep learning.

2.2.7 Transformer

Attention, as it was applied to Recurrent Neural Networks and Convolutional Neural Networks (not discussed in this thesis), was a sudden windfall to NLP and especially tasks within natural language generation. By nature of recurrent architectures, there were still setbacks in the form of computational complexity. RNNs performed poorly over batches of sequences. Each sequence must have the same number of timesteps so sequences were padded to the length of the longest sequence in each batch. This led to memory issues and otherwise poor computational performance with large datasets.

Vaswani *et al* challenged the necessity of a recurrent architecture in *Attention is All You Need* [38]. Their proposed *transformer* was an entirely attention-based Seq2Seq model that processes each timestep simultaenously, without the need for iterative computation.

Overview

Unlike recurrent Seq2Seq architectures, transformer encoders and decoders have no hidden states between timesteps. They can attend to previous timesteps within themselves via a mechanism that Vaswani *et al* refer to as *self-attention*. With the ability to attend to any previous index within a sequence, the recurrent computation of a hidden state becomes entirely unnecessary which results in much more efficient training.

Similar to Bahdanau *et al*, the decoder in a transformer can refer to any timestep in the encoder when determining the output distribution over potential next tokens. Because the encoder is entirely driven by attention, the decoder attends to the calculated self-attention in the encoder rather than a hidden state as a recurrent network would. This is referred to as *encoder-decoder attention*.

Another novel component of the transformer was *multi-head attention*. It's naive to assume that there is exactly one feature vector to be learned when modelling an alignment

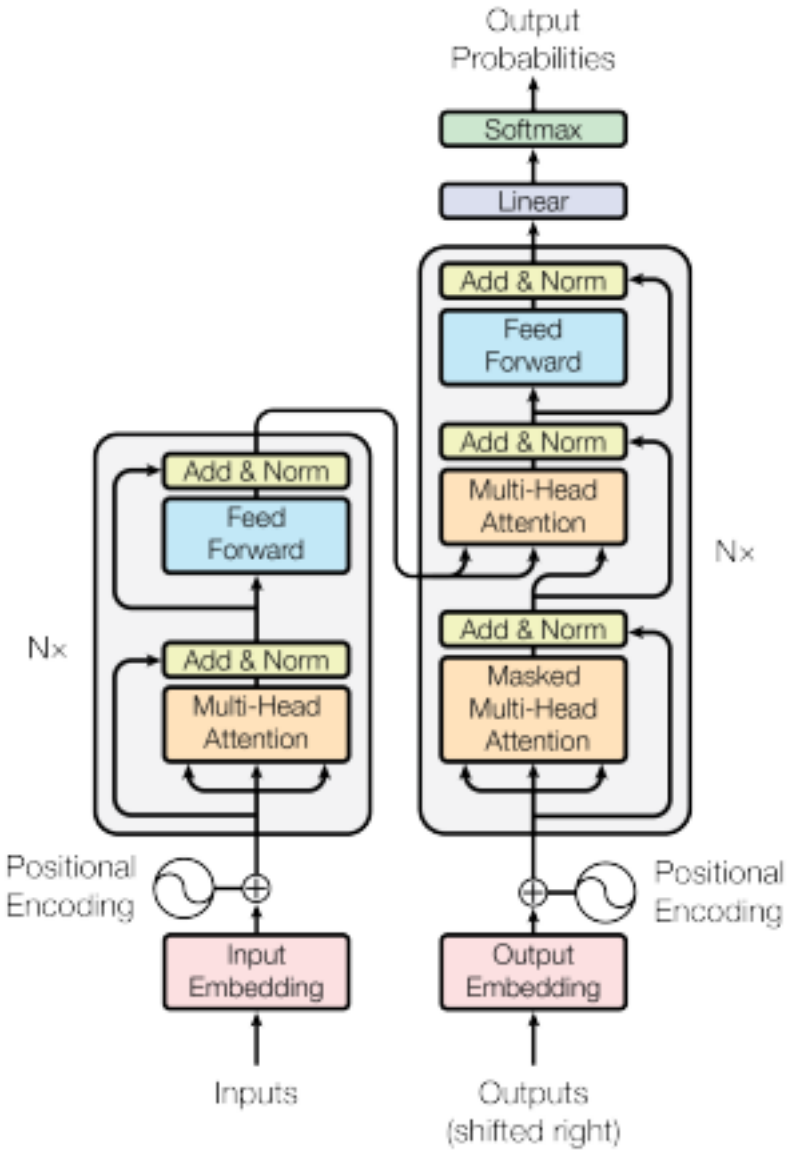

Figure 2.2: Overview of the transformer from Vaswani *et al* [38]

between two tokens. Vaswani *et al* put forth the notion that many feature vectors could be learned simultaneously, each referred to as a head. In the original transformer, there are 8 attention heads that jointly learn different features within a sequence. Vaswani *et al* suggest that each head is interpretable in its own way with some bearing obvious correlation to semantic and syntactic structure.

Encoder

The encoder portion of the transformer is composed of a series of six identical *layers*. Inputs to each layer are consumed in parallel with each input embedding occupying one *position*. Each position is conceptually equivalent to a timestep in a recurrent attention model. Vaswani *et al* use byte-pair encoding for the input so the embedding at each position corresponds to a precomputed subword token from the input sequence. A unique sinusoidal position embedding (2.12, 2.13) is added to each input embedding before the first layer in the encoder stack receives them.

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.12}$$

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.13}$$

In each encoder layer, the input embeddings at each position are first projected linearly into three lower dimensionality vectors referred to as the query q, the key k, and the value v (2.14, 2.15, 2.16). Each of the h heads in the multi-headed attention mechanism has its own projection matrix for queries, keys, and values. Attention for each head is computed in parallel.

$$Q_h(x) = W_h^Q x \tag{2.14}$$

$$K_h(x) = W_h^K x \tag{2.15}$$

$$V_h(x) = W_h^V x \tag{2.16}$$

The dot product of the resultant queries and keys in each head is then scaled and normalized with the softmax function (2.17). Vaswani *et al* note that scaling is necessary

as to not mitigate the gradient signal through the softmax normalization step in instances where the magnitude of the key vectors are extremely large. The dot product of Q and K in each head is scaled by a factor of $\frac{1}{\sqrt{d_K}}$ where d_K is the dimensionality of each key vector. As trainable parameters, the query, key, and value projections in each head will learn different features of the input as the model converges.

$$Attention_h(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}})V \tag{2.17}$$

$$MultiheadAttn_j(x) = a_j([Attention_1; \ldots; Attention_h]) \tag{2.18}$$

$$A_j = LayerNorm(MultiheadAttn_j(x) + x) \tag{2.19}$$

The outputs of each attention head are concatenated and projected into a final attention vector for each position (2.18). As the resulting attention vector matches the dimensionality of each input embedding, the two can be added via a residual connection before subjected to layer normalization (2.19). After layer normalization, the values at each position are passed through a two-layer MLP with a nonlinear activation function in between the layers. Once more, the resulting values are added to a residual connection and subjected to layer normalization before the encoder layer is complete (2.20).

$$EncoderLayer_j = LayerNorm(MLP_j(A_j) + A_j) \tag{2.20}$$

The outputs of a layer j are passed as input to the subsequent layer $j + 1$. The output values of the last encoder layer serve as a reference to the source sequence for the decoder.

Decoder

Decoder layers function similarly to encoder layers in that they also compute self-attention but over the target sequence. There are two primary differences between encoder and decoder layers.

The first difference is that decoder layers add *masking* to the self-attention mechanism at the beginning of each layer. Masking is the concept of concealing a portion of the target sequence at each position. Because autoregressive models perform sequential processing, attention values from later positions in the target sequence are set to $-\infty$ to negate their

effect on the output distribution. This, in effect, emulates the autoregressive property of a recurrent decoder.

The second difference found in decoder layers is that there is an additional attention computation sublayer called *encoder-decoder attention*. Encoder-decoder attention is calculated using a projection of the encoder output as the key and value matrices for each head. The queries are projection of the output of the decoder masked self-attention sublayer found immediately before the encoder-decoder attention sublayer.

Similar to recurrent decoders, the outputs at each position are projected to an n-dimensional vector where n is the size of the byte-pair dictionary. Softmax normalization creates a distribution over each candidate token for a given position.

The idea of self-attention was a gamechanger and immediately applied to tasks beyond aligning two sequences. Variations of transformer encoders and decoders were applied to a variety of tasks, breaking many state-of-the-art benchmarks in the process.

2.2.8 Generative Pretrained Transformer

Transformer decoders on their own are generative models trained on maximizing the likelihood of subsequent tokens in a sequence. Radford *et al* (2018) demonstrated that transformer decoders are strong language learners on their own by subjecting them to two phases of training. The resultant transformer decoder was referred to as the Generative Pretrained Transformer (GPT) [28].

The first training phase, called *pretraining*, involved training the transformer decoder on next word prediction as Vaswani *et al* did. Without using an encoder, Radford *et al* (2018) had to augment their decoder by removing the encoder-decoder attention sublayer. This phase is considered unsupervised learning as the decoder input sequence is the same as the decoder target sequence but offset by one token. The BooksCorpus dataset [43] was used for pretraining as it contains large spans of continuous text.

After priming the attention heads through the pretraining phase, the second phase of training consisted of supervised tasks with explicit labelling such as sequence classification, question answering, and natural language inference. The outputs of the pretrained decoder h_j were projected linearly for classification and trained to the objective of maximizing the probability of the correct class for each task. This process is called *finetuning*.

GPT beat state of the art recurrent encoders in all of the tasks on which it was evaluated. Radford *et al* (2018) noted that the performance of GPT is further improved if

fine-tuning is performed as a *semi-supervised* task where GPT is trained unsupervised on next-word prediction while performing a supervised classification task in parallel.

GPT-2

Radford *et al* (2019) pushed the limits of transformer decoders once more by pretraining on a larger, more diverse dataset called WebText [29] which they scraped themselves. The goal of using more diverse data scraped directly from the internet was to bolster the ability for the generative model to perform zero-shot task transfer. The resultant model is called GPT-2.

Zero-shot task transfer, in this instance, refers to the transformer decoder's ability to accept a task in the form of an input *context* and generate an answer rather than explicitly finetuning on that specific task. For example, Radford *et al* (2019) measure their proposed model on a question answering benchmark dataset by posing the question in the form of a prompt and having the decoder generate subsequent tokens which would answer the question. By providing the transformer decoder with the input "Who wrote the book the origin of species?", the model will generate subsequent words "Charles" and "Darwin" with high likelihood.

We use GPT-2 as the decoder for our experiments for its astute ability to continue a context through next word prediction. In our experiment, GPT-2 receives a story partial in the form of context and is conditioned to attend to an additional embedding containing a future plot element.

2.2.9 Bidirectional Encoder Representations from Transformers

The first GPT was a proof of concept that attention-based autoregressive models are not only impressive unsupervised learners, but they learn robust representations of text for supervised tasks as well. Despite the strength of GPT as a next token prediction model, unidirectional encoding is a suboptimal choice for tasks such as question answering where an entire context passage should be accounted for from each direction.

ELMo

Peters *et al* had created more robust token-level embeddings by encoding a sequence from either direction with their model ELMo. Learning ELMo embeddings requires a two step

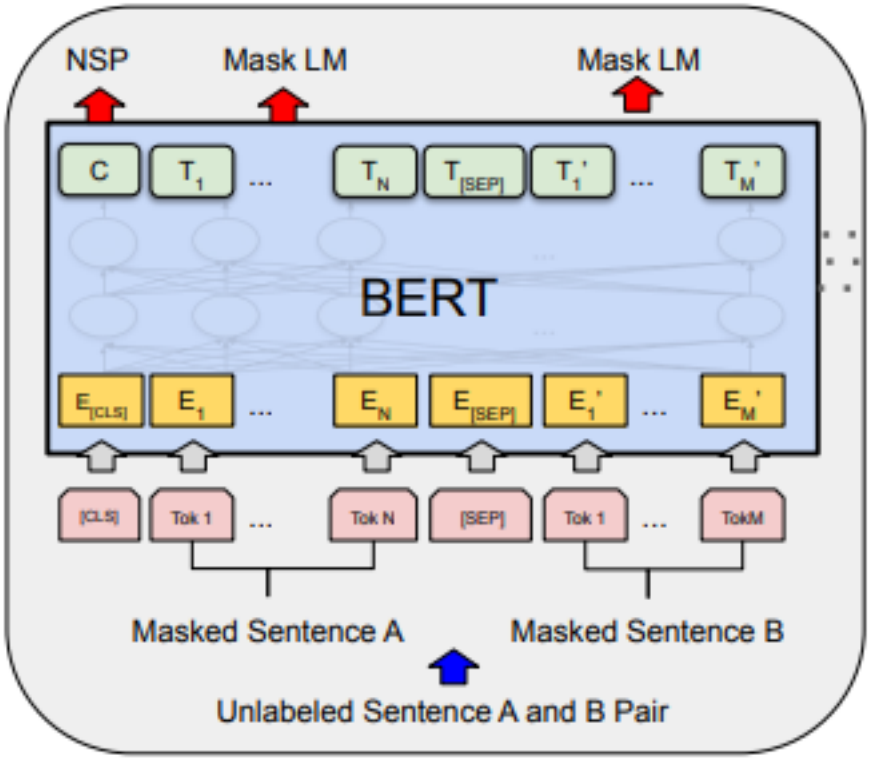

Figure 2.3: Structure of BERT input and output sequences from Devlin *et al* [9]

training procedure similar to the approach used for GPT by Radford *et al*. During the pretraining phase, two unidirectional LSTMs learn a representation of a source sequence from each direction. The hidden states at each timestep of the pretrained LSTMs are concatenated and used as a basis for the token embeddings during finetuning. These *deep representations* serve as much richer representations of tokens that can account for contextual information that may prove useful in downstream tasks.

Devlin *et al* contend that this approach is not deeply bidirectional because each LSTM is pretrained independently of one another and their resultant features are concatenated for downstream learning. This is a hard limitation of recurrent networks which must encode sentences sequentially.

BERT Overview

Transformer encoders are unravelled by nature and can process tokens at each position simultaneously. Devlin *et al* leverage this capability to create a truly deep bidirectional encoder, which they name BERT [9]. Yet another homage to Sesame Street, BERT stands for Bidirectional Encoder Representations from Transformers.

BERT uses Word Piece Encoding [34] for its input tokens and adds them to a position embedding indicating which sentence in the context that the token belongs to. Similar to GPT, training BERT is composed of two phases: pretraining and finetuning.

BERT Pretraining

BERT pretraining involves two unsupervised tasks. The first task is referred to as the *masked language modelling* task. The second task is referred to as *next sentence prediction*.

Masked language modelling refers to the task of inferring a masked token from surrounding context. Because BERT bidirectionally encodes a source sequence, it cannot be trained on next sequence prediction as would a transformer decoder such as GPT. Instead, BERT is a *masked language model* (MLM). An MLM is trained by randomly masking an input token and using the surrounding context to infer the masked token. As a transformer encoder can attend to context in either direction from a masked token, it can build stronger representations of a sequence in its entirety.

Next sentence prediction (NSP) is a binary classification task to determine if one sentence immediately follows another. Devlin *et al* create a processed dataset where 50% of the datapoints are sentences that follow one another and the remainder are two arbitrarily selected sentences from the corpus. The sentence pairs are combined, delimited by a special token [SEP], and then encoded together by BERT. BERT source sequences are prepended (position 0) with another special token [CLS], the output of which is used for sequence classification. During the NSP pretraining task, the output of position 0 is used as the representation fed to the downstream binary classifier. Devlin *et al* demonstrate the impact of the NSP pretraining task on downstream tasks such as question answering in their experiments.

BERT Finetuning

BERT is finetuned on a variety of tasks with different positional outputs used for different tasks.

For downstream token-level tasks, such as question answering, the outputs at each position are passed to an additional output layer for token selection.

For downstream sequence-level tasks, such as sentiment analysis, the output from position 0 is passed to a downstream task-specific architecture. During NSP pretraining, this position is primed as a sentence-level representation of the input sequence. This general sentence-level representation can have task specific features learned by a downstream model during finetuning. Using these representations after fine-tuning, BERT smashed several sentence-level benchmarks at the time of its publication.

For our experiments, we use a BERT-encoded future plot event which is derived from the position 0 embedding. A projection of this embedding is attended to by the decoder to steer the direction of a story context towards the encoded plot event. During training, the gradient signal is traced backwards through the attention heads of the decoder to train our intermediary projection layer and finetune BERT jointly.

2.2.10 Accessibility of Transformers

Although transformers naturally learn robust representations of language, training a transformer is prohibitively expensive financially and may not be necessary for most language modelling tasks. The original transformer of Vaswani *et al* was trained on an array of 8 NVIDIA P100 GPUs over the course of 3.5 days [38]. Larger transformer variants, such as BERT [9], showed that higher orders of total model parameters did have an influence on model performance. Though the base BERT model with 110 million parameters broke the record across several benchmark metrics, the large BERT variant with 340 million parameters had percentile scores several points higher across the metrics tried. More recent transformer variants have parameter counts in the order of hundreds of billions [4] or even trillions [11] and cost millions of dollars with an emphasis on improving training stability in distributed environments.

Fortunately, pretrained transformer parameters can be shared online via channels such as HuggingFace [41] for others to finetune on smaller, task-specific corpora. By pretraining on large, general corpora derived from the public Internet, transformers such as GPT-2 learn much more than the ability to generate fluent language. Using pretrained model parameters for finetuning on downstream tasks has become the predominate approach for harnessing the power of transformers. We download our parameters for each our BERT future encoder and our GPT-2 storywriting decoder from HuggingFace and finetune them jointly on our proposed task.

2.3 Controllable Transformers

A caveat of using pretrained transformers is that some downstream tasks require signalling that isn't accounted for during model pretraining. Moreover, some tasks require explicit instructions that cannot be inferred by a model through zero-shot task transfer as it is demonstrated by models such as GPT-2 [29] and GPT-3 [4]. For a transformer decoder to perform zero shot translation from English to French, it must sufficiently understand the prompt *"Translate from English to French:"* and results may vary depending on which variation of that prompt is used to guide the generation. With tasks such as style transfer, a prompt must be provided each time the user wishes to change the style. Each new style can only be provided explicitly through the source sequence to spur the generation as the pretraining procedure only accommodated a source and target sequence.

2.3.1 CTRL

Attempts to guide transformers by redefining the **pretraining** procedure to accommodate additional domain-specific flags saw some success [16]. Keskar *et al* prepend model contexts with a one word token selected from a finite list of domain labels called *control codes*. Control codes in CTRL correspond to the source of each dataset used during pretraining which allows the model to learn an alignment between the language in a source sequence and the domain from which the source sequence was drawn.

At inference time, use of varying control codes led to a quantifiable and predictable difference in generation when passing the same prompt. Moreover, Keskar *et al* note that the model could simply be passed a control code and generate an entirely novel text styled after the corresponding domain.

Though CTRL may have mitigated some variance in domain-specific controlled generation, there were still several flaws in its implementation. Control codes are ingested as tokens within the prompt which cannot be adjusted after the model begins generating text. For example, in story generation, it may be necessary to change the target sentiment multiple times within one generated passage. Without control code signals using some auxiliary mechanism, these domain switches become difficult to accommodate at inference time. Additionally, new control codes cannot be introduced after pretraining. The appeal of zero shot task transfer as described in GPT-2 is to infer domains without explicit training instructions. Using preordained control codes works against this concept. Aside from these inflexibilities, CTRL also contained approximately 1.6 billion parameters and was

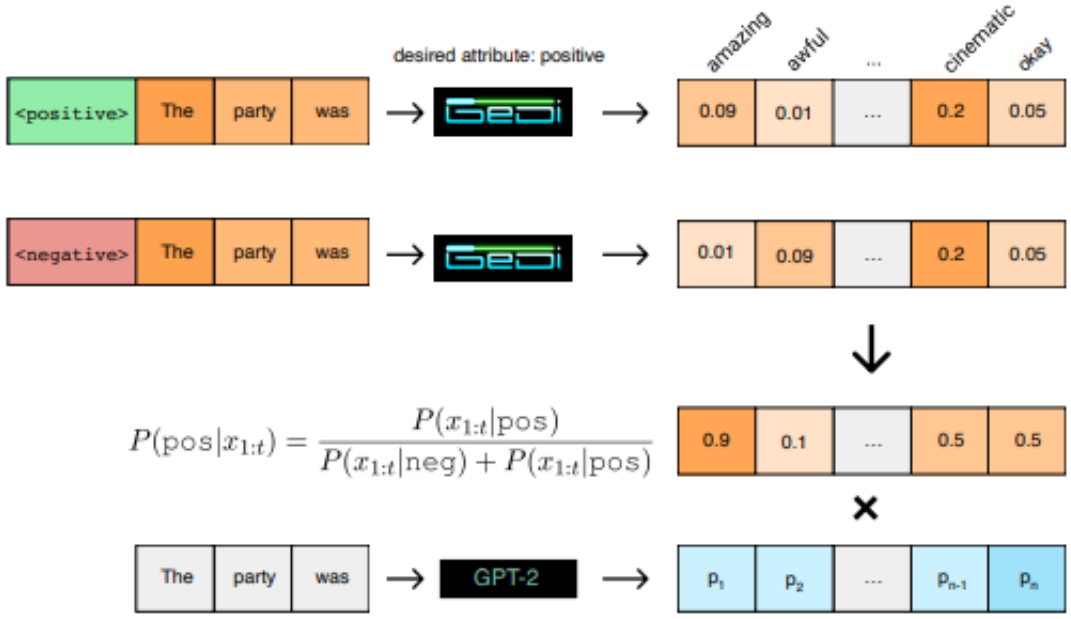

Figure 2.4: Example of GeDi weighting the output distribution of a pretrained GPT-2 from Krause *et al* [17]

.

pretrained over two weeks on a Google Cloud TPU v4 pod. Replicating this process to introduce new control codes is both costly and time-consuming.

2.3.2 GeDi

Krause *et al* avoid the costly pretraining involved with CTRL without obfuscating the weights of the generative language model by using generative discriminators (GeDi) [17].

They first finetune a pretrained GPT-2 as a class-conditional language model (CCLM), similar in functionality to CTRL. Conditioning the output distribution on a particular class is predicated in Bayes' Rule for partial sequences (2.21).

$$p(c|x_{1:t}) = \frac{\prod_{j=1}^{t} P_\theta(x_j|x_{1:j-1}, c) \cdot P(c)}{\sum_{c' \epsilon \{c, \tilde{c}\}} \prod_{j=1}^{t} P_\theta(x_j|x_{1:j-1}, c') \cdot P(c')} \tag{2.21}$$

Using this equation as an objective requires examples both a class c and an anticlass $\tilde{c}$. Maximizing 2.21 as a discriminative objective function effectively maximizes the distance

between the distributions for c and $\tilde{c}$. In doing this, any commonalities between the two classes are negated as the model converges. The resulting logits are normalized as a discrete distribution of weights w_i.

GeDi applies the weights produced by the CCLM to the outputs of a second, unmodified pretrained transformer decoder that uses the same vocabulary. The obvious choice for this is a second pretrained GPT-2 instance. The second decoder produces logits at each position of the output sequence which are first normalized to create a discrete distribution of candidate tokens y_i. The inplace product of w_i and y_i creates a distribution biased by the class parameter c (2.4). Krause *et al* note the use of *nucleus sampling* having a positive effect on the results. Nucleus sampling is the process of redistributing the probabilities in a distribution to favor events with a likelihood over a certain threshhold.

2.3.3 OPTIMUS

Li *et al* learn a smooth latent space from which to sample vectors for transformer decoder conditioning[20]. By connecting BERT and GPT-2 as a *variational autoencoder*, BERT learns a continuous distribution of encoded sentences rather than just one embedding for a source sentence. They name their proposed architecture **O**rganizing sentences via **P**re-**T**rained **M**odeling of a **U**niversal **S**pace (OPTIMUS).

Though not relevant to our work, we will provide a brief description of a variational autoencoder (VAE). Variational autoencoders are an extension of standard autoencoders, which learn to encode a source sequence x into a vector z with an encoder and then decode z into a replica of the source sequence x' with a decoder. Unlike a standard autoencoder, also known as a deterministic autoencoder (DAE), variational autoencoders learn a continuous distribution over z by applying Bayes' Rule to learn an approximate posterior distribution $p(z|x)$. This is achieved by maximizing the Evidence Lower Bound (ELBO) during training to maximize the probability of the approximate posterior as model weights converge. Done correctly, the learned distribution will be a high fidelity representation of x from which x' can be reconstructed. Li *et al* apply the variational autoencoder objective to learn a continuous space over sentences encoded by BERT.

A pretrained GPT-2 is modified to ingest any embedding sampled from this continuous space and dynamically condition on it without the need to deviate from the structure of the pretraining data. Sampled vectors can be attended to as position zero, a process referred to as *memory injection*. *Embedding injection* refers to applying the sampled vectors over the input embedding at each position. We use a similar approach in our work, which we

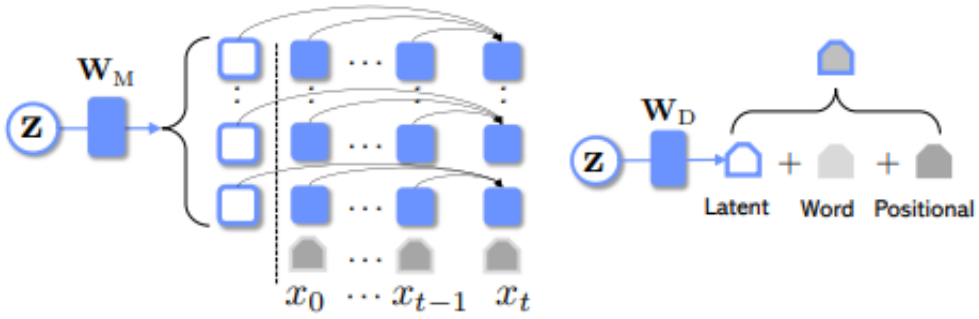

Figure 2.5: The two types of decoder injection: memory(left) and embedding(right) from Li *et al*[20]

.

will elaborate on more in our Methodology section. Li *et al* evaluate both and find that the combination of the two approaches achieved the most compelling results.

2.4 Automated Story Generation

Automated story generation is the task of using either a set of heuristics [19, 23] or a machine learning algorithm [10, 30, 42] to exhibit attributes of *narrative intelligence* in story writing. Riedl and Young define narrative intelligence as the ability for a storytelling agent, human or machine, to convey an experience as a story [31]. Narrative intelligence encapsulates qualities of stories such as thematic and syntactic consistency, two attributes which come naturally to humans yet remain nontrivial for automated storytelling agents to emulate.

2.4.1 History of Story Generation

TALESPIN

One of the earliest examples of automated story generation was the TALESPIN algorithm [23]. TALESPIN parsed user input as a sequence of rules which served as a rudimentary plan from which a complete story could be drawn. From there, each rule was

mapped to hard-coded descriptions which, when combined, formed a more complete narrative. Though compelling at the time, generating stories from a discrete subset of rules often led to overlap between various stories.

Plot Graphs

Li *et al* proposed a more mature extension of this concept in the form of *plot graphs* [19] drawn from many crowdsourced stories. A user submitted topic is queried against crowdsourcing platform Amazon Mechanical Turk (AMT) as a request for structured narratives on that topic. Plot graphs are constructed from the crowdsourced stories with plot events represented as nodes, precedence relations represented as unidirectional edges, and mutual exclusion relations represented as bidirectional edges. From plot graphs, many diverse stories on one topic could be drawn by traversing different paths through the graph. An obvious caveat to this approach is that crowdsourcing relies on human intelligence which is slow, costly, and susceptible to error. Moreover, there was no generative component to this model to transcribe traversed nodes into complete sentences.

2.4.2 Story Generation with Deep Learning

With access to deep learning, the generation component of story generation could be delegated to generative language models. As generative language models learn to generalize over language probabilistically, the goal of story generation tasks then became to devise a method by which to guide these models in their generation.

Hierarchical Neural Story Generation

Fan *et al* structure their automated story generation agent as a hierarchical task over two phases [10]. The first phase generates a story prompt: a one sentence premise on which to condition a generative model. The second phase uses a generative language model to create a story aligned with the prompt generated in the first phase.

To generate a prompt, Fan *et al* use a convolutional neural network (CNN) to generate a prompt. Fan *et al* reason that CNNs dont rely on sequential decoding like recurrent networks do so they are quicker to generate longer passages of text.

In the second phase, two additional convolutional neural networks work together to generate a story conditioned on a prompt. The first CNN is trained in a pretraining step

to learn the intricacies of storywriting itself with no particularly strong adherence to the prompt. Once the first CNN has grasped some notion of storywriting, the second CNN can be trained jointly with the first in order to learn any weaker signals such as the alignment between prompt and story. Fan *et al* refer to this approach as cold-fusion. Each CNN has attention heads similar to those proposed by Vaswani *et al* [38] but with a additional deep gating mechanism from Dauphin *et al* [8] to impose more strict filtering on the flow of information through the heads.

Fan *et al* scraped and shared a collection of stories and prompts collected from the Reddit forum r/WritingPrompts with which they trained their model. Writing prompts, while useful starting points, often leave finer details of the plot up to the generative model. Though our model doesn't condition on prompts, our experiments uses the stories from the Writing Prompts dataset and we extract future plot events from the stories themselves.

Event Representations

Martin *et al* also use a hierarchical approach by learning a conditional distribution of plot events and then transcribing them as sentences in a story. [21]. Plot events are represented as 4-tuples consisting of an event subject, an action, an indirect object, and a wildcard element which contains any other information unclassifiable by Stanford's Core NLP library. Additionally, Martin *et al* experiment with 5-tuples containing information about the sentence genre or overall sentiment.

Plot events are extracted into tuples in series and encoded by a RNN encoder-decoder pair called *event2event*. The hidden states at each timestep represent an encoded event which can be passed in series to a second RNN encoder-decoder *event2sentence*. The event embeddings are transcribed as readable sentences by event2sentence, creating a story from the events predicted by event2event. Prediction of subsequent events is only conditioned on previously predicted events rather than generated texts. As converting complete sentences to their event representations is a lossy process, this could lead to a lack of thematic consistency over time without a supplementary memory mechanism.

This approach differs from ours in their representation of future events. Our model relies on the ability of a transformer decoder to interpolate between a story context and future plot event encoded as plaintext. We contend that unsupervised pretraining over general corpora provides a stronger notion of story understanding than meticuously extracting grammatical elements and transcoding them into text.

Plan and Write

Yao *et al* propose a similar hierarchical approach which compares two methods of guiding generation from one plot event to the next [42]. The first method, *static planning*, uses a recurrent network predict a series of five one-word plot events. Each plot event in the story is aligned to exactly one sentence in the generated story. After first building the entirety of the plot outline, sentences are generated with respect to each plot event in series. By contrast, *dynamic planning* chooses a plot event and then generates a sentence aligned with that plot event. Both the selected event and subsequently generated sentence condition the selection of the next plot event in the story. In this way, the plot is decided dynamically as the story unfolds rather than being entirely preordained.

While single word plot events work in principle, the stories generated from such low information events tend to be simple and predictable. Moreover, Yao *et al* use exactly five element storylines due to the limitation of stories in their selected dataset being only five sentences in length.

This work differs from ours in that our approach provides true dynamism in allowing the user to propose a plot direction entirely on their own rather than choose from those proposed by a recurrent planning mechanism. Moreover, encoding complete sentences as future plot events in high-dimensional vectors has a greater capacity for storing pertinent plot information useful to the decoding process.

Plotmachines

Rashkin *et al* pretrain an augmented transformer variant based on GPT which they call *Plotmachines* [30]. Unlike a conventional GPT which learns to reconstruct input sequences, Plotmachines generates entire paragraphs from highly structured input consisting of plain-text plot events, story history, and a discourse indicator each delimited by special tokens. The goal of Plotmachines is to maintain adherence to a storyline over the span of many paragraphs, a task also referred to as *longform story generation*.

Plotmachines consists of two primary components, the generative transformer decoder itself, and a memory mechanism to allow the model to keep track of plot points addressed in previously generated paragraphs. A series of short, multi-word plot events to be addressed in the story are fed to the generative decoder, delimited by a special token **kw**. Additionally, an embedding produced by the memory mechanism and a discourse code are appended to these plot events which together prepend the decoder input sequence. All of this high level information can be optionally attended to by the decoder as it is trained from scratch on story reconstruction.

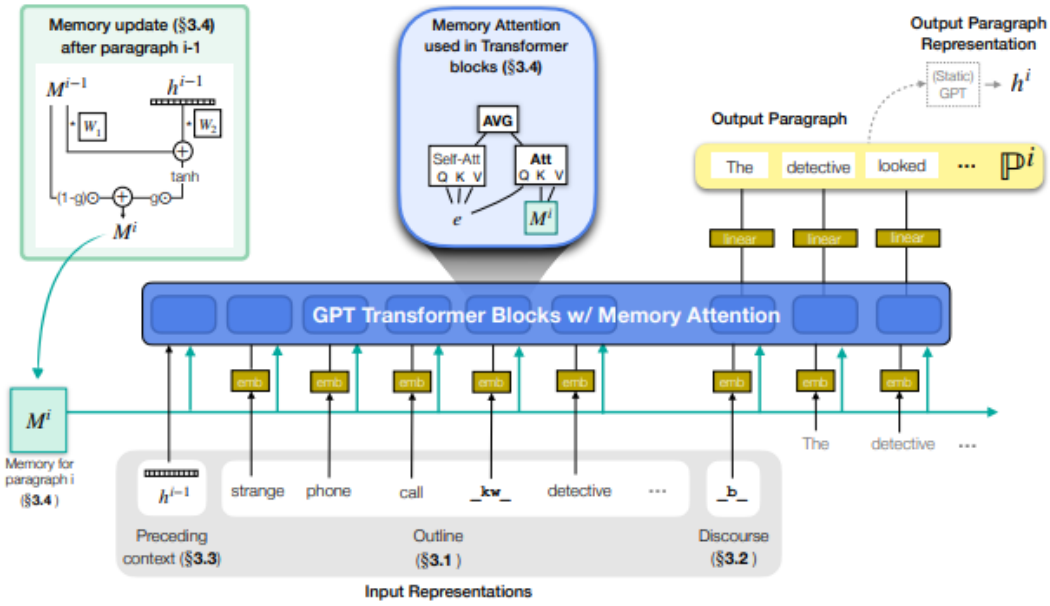

Figure 2.6: Plotmachines model architecture from Rashkin *et al* [30]

Each generated paragraph is passed as input to a pretrained GPT-2 with weights that remain frozen during training. The output embeddings are averaged and passed to a trainable memory network with multiple gates to control the acceptance and rejection of new information into memory. The produced memory embedding is optionally attended to by special attention heads at each position of the generative decoder that are dedicated specifically to the retainment of information between paragraphs.

Unlike our model, Plotmachines leaves it to the model's discretion when to address future plot events. Our model allows an interactive approach where a user can choose a plot event to guide generation towards in the immediate future. Moreover, Plotmachines has an expensive training procedure and many data annotations of the data which aren't easily replicatable. Our semi-supervised training procedure is simple, requiring only a sentence tokenizer and pretrained weights for GPT-2 and BERT.

Chapter 3

Approach

3.1 Task Overview

Our approach deviates from other techniques for automated story generation in that it facilitates a user to interact directly with the generative process. Previous approaches to automated story generation rely on rigid and preordained structure which makes it impossible to incorporate plot events inspired by text generated by the model. We propose the novel task of *future conditioning*, the process of coalescing a story context and a future plot event, each encoded as complete sentences. In pursuit of this task, automated story generation agents can learn to accommodate user input in real time in order to incorporate interesting, context-sensitive events into any story.

To motivate this, consider cases where an automated story agent introduces a character to a story that was previously unmentioned. A human agent working cooperatively with the model may wish to explicitly include the new character as the subject of a future plot event. It would be impossible to incorporate this character into a series of preordained plot events because the inclusion of the character was decided by the generative model at inference time.

3.1.1 Future Conditioning

Future conditioning is the process of guiding a generative language model to interpolate between a context c and a future event f. Interpolation, in this sense, refers to moving

between two events temporally rather than interpolating between two points in a continuous space. Future conditioning can be thought of as smoothly blending together a context paragraph and future plot event. From the end of a context paragraph c, the model should priortize selecting tokens that both expand on c and logically conclude with a sentence f. During inference, the user can specify plot events to continue the story, with or without regard to the text that the model has previously generated.

3.2 Model Architecture

Though our model requires each an encoder and a decoder, it's not a encoder-decoder model in the traditional sense as transformer decoders are capable of generating on their own due to their self-attention mechanism. The BERT encoder is only used to create a high level representation of the future event which is encoded as a complete sentence. Our decoder is an extension of GPT-2 that is augmented to be able to attend to the encoded future plot event from any position in the context. In this way, the context does not need to be prepended with any special tokens such as control codes or a series of plot events. Additionally, during inference, our decoder is functionally similar to a vanilla GPT-2, ingesting a story context and sequentially predicting subsequent tokens. We refer to our complete model architecture as *Future Sight*.

3.2.1 Encoder

As our intention is to incorporate future plot events in the form of complete sentences, we need a strong representation of a sentence to encode as much high level event information as possible. We chose a pretrained BERT base model with 110 million parameters, which we downloaded from the HuggingFace pretrained model repository. The HuggingFace implementation of BERT encodes input sequences and returns an object *BaseModelOutputWithPooling*. From this object, we use the member variable *pooler_output* as our sequence embedding. This *pooler_output* is the position zero **CLS** output that is further processed by a classification head consisting of a linear layer and a *tanh* activation function (3.1).

$$F^E = \tanh(Linear(\mathbf{BERT}(f))) \tag{3.1}$$

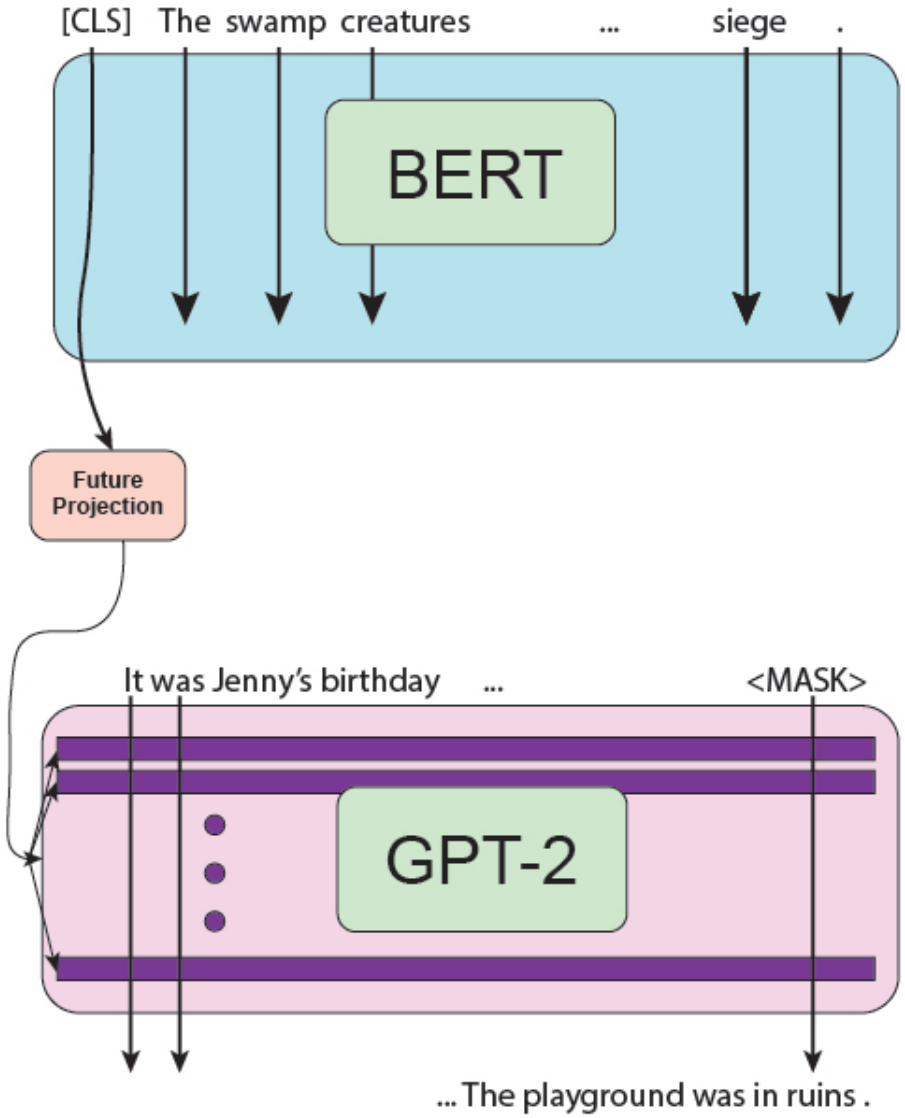

Figure 3.1: Future Sight complete model diagram.

3.2.2 Future Injection

To integrate the encoded future plot event into the decoder, we referred to the embedding injection techniques from Li *et al*'s **OPTIMUS**. Li *et al* conditioned a GPT-2 instance on embeddings sampled from a continuous latent space as part of a variational autoencoder. These embeddings were injected into GPT-2 in two ways: memory injection and embedding injection.

Embedding Injection

Embedding injection refers to concatenating conditionining information directly to the embedding of each input token in the decoder. Though we added an implementation of embedding injection to *Future Sight*, we didn't observe a conditioning effect while using it. Due to this, we focus on memory injection for the duration of this work.

Memory Injection

Memory injection is used in our approach to provide each self-attention head in the decoder with the opportunity to attend to information from an encoded future event. As each decoder layer has its own self-attention weights focusing on different abstractions of the context features, we provide each layer with a different projection of the encoded future.

3.2.3 Layered Memory Injection

In order to create a future representation for each layer, we use an trainable intermediate network. The embedding from the BERT pooler output is projected with a linear layer and ReLU activation function (3.2). This projection results in an $(n \times d)$ dimensional vector. The variable n is derived from the number of blocks or *layers* in the transformer decoder. The variable d refers to the hidden dimensionality of the decoder. We refer to this projection as the *future projection* F^P.

$$F^P = ReLU(Linear(F^E)) \tag{3.2}$$

3.2.4 Decoder

Pretrained transformer decoders, such as GPT-2, possess a high capacity for language knowledge due to their extensive pretraining regimen across a plethora of varying domains. During finetuning, this allows the decoder to focus on learning high level, task-specific information, such as past and future plot elements, while roughly retaining its ability to generate text that is syntactically correct and fluent-sounding. For this reason, we chose to use the pretrained GPT-2 instance from HuggingFace with 117 million parameters for our experiments. We augment the HuggingFace GPT-2 implementation in two ways.

The first modification we make is to the GPT2 module, which contains both the transformer submodule and the language modelling heads. Here we allow the model to take an additional parameter: the $(n \times d)$ dimensional vector output by our future projection. This vector is split into n smaller vectors with each being passed to a different decoder block (3.3). The pretrained GPT-2 that we used for our implementation had an n of 12. Larger implementations contain more decoder blocks so the vector from the future projection would be larger with higher parameter versions of GPT-2. Similarly, 768 is the dimensionality of the hidden states of each decoder block in the smallest pretrained implementation of GPT-2. In larger variations, the hidden states are higher dimesional and this would impact the future projection accordingly.

The second modification we make is to the attention heads in each block. The future injection effectively acts as the keys and values for an artificial position zero, prepending the computed attentions at the position index of each token in the context inputs (3.4, 3.5). The future injection for each transformer block is duplicated as both key and value vectors for each head in the multi-headed attention mechanism. At each position, any head can optionally attend to information from the future plot event. During training, the gradient signal is traced back through the attention heads to the future projection network.

$$F_l^P = \begin{pmatrix} F^P_{l \times d} & \cdots & F^P_{(l+1) \times d} \end{pmatrix} \tag{3.3}$$

$$K_l = K_l^{Inj}(x) = [F_l^P; W_h^K x] \tag{3.4}$$

$$V_l = V_l^{Inj}(x) = [F_l^P; W_h^V x] \tag{3.5}$$

Chapter 4

Experiments

4.1 Datasets

Early prototypes of our model were trained using the ROCStories dataset [25], a collection of short five sentence stories designed for story cloze tasks. We chose to deviate from this dataset for our experiments because training on five sentence stories with low language diversity tends to produce similarly trivial stories at inference time.

For our experiments we use the WritingPrompts dataset originally used by Fan *et al* [10]. The WritingPrompts dataset was proposed specifically for the task of hierarchical story generation, the task of first creating a prompt or storyline and then conditioning a generative language model on it as part of a two step pipeline.

WritingPrompts consists of 303,358 stories and prompts from r/WritingPrompts, a Reddit community oriented around story authorship and contribution. The goal of this community is to have users propose prompts for other users to respond to with a related story. Community contributors can vary in writing skill level from amateur to professional, which results in high diversity in both story length and quality. Fan *et al* raise the issue of difficulties modelling a vocabulary built from a corpus containing many rare words and misspellings, but this is easily addressed by GPT-2's use of byte-pair encoding for embedding construction.

We ignore the prompts during preprocessing as our task is to condition GPT-2 on an event drawn from a source story itself rather than an overall prompt. Additionally, we ignore stories that are less than 9 sentences long. This leaves 291,575 stories which are further subjected to a train/val/test split of 98%/1%/1% (4.1).

Dataset	Story Count
Train	285743
Validation	2916
Test	2916
Total	291575

Table 4.1: The train, validation, and test split for our selected dataset.

4.2 Preprocessing

We evaluate our model on a variety of preprocessing steps in an effort to discern which properties of contexts and futures influence the strength of the conditioning the most. The encoded future has the sizeable task of containing features that will guide GPT-2 towards the generation of multiple sentences. A future with many unique words that don't appear in the respective context is responsible for conveying that knowledge to the generative model. Our preprocessing step operates at the sentence level. We use NLTK to split each story at the sentence level before constructing our datapoints. We use only the first n sentences of each story for subsequent steps (4.1).

$$S = sent\_tokenize(story)_{[0,n]} \tag{4.1}$$

A contiguous subset of the story sentences represent the context c. This subset can vary in size by a parameter we define as *context length*, l (4.2). We define future candidates as the subsequent d sentences in the story (4.3). One of the future candidates is then selected heuristically as the future plot event f. It's important that future candidates exclude the sentence immediately following the context or the task is trivialized as merely reconstructing the future.

$$c = S_{[0,l]} \tag{4.2}$$

$$F = S_{[(l+1),(l+d)]} \tag{4.3}$$

Controlling for the length of the context, in sentences, provides the model with more or less information on which to infer subsequent tokens while ignoring the encoded future. Similarly, choosing a future that is sequentially further into the future could cause the model to struggle to infer how to connect it with the story context.

Context: Why aren't these lawyers in yoga pants? Everyone wore yoga pants everyday. Really? Wouldn't it be uncomfortable when it was hot out? Silly executive TV producer, don't you know that global warming hadn't kicked in yet?
Future Distance: 3 sentences
Future: Probably not.

Table 4.2: Example of an uninformative sentence selected randomly as a future from the WritingPrompts dataset.

4.2.1 Future Distance

In our original prototype using ROCStories, we specified a context length of 3 and a future distance of 2. These parameters were fixed at these values because each story contained exactly five sentences. Though there was some evidence of conditioning, the model learned to infer that stories were only to be five sentences in length which was one reason we switched to WritingPrompts for our experiments.

Using a subset of WritingPrompts stories which only contained 9 or more sentences allowed us enough room to vary the future distance more flexibly. This allowed us to look at the effects of using futures closer or further into the future on the conditioning.

Randomized Future Distance

The naive approach to selecting a future is to select one at random. Though arbitrarily selecting a future candidate could result in a future plot event that contains interesting new information, all futures were not created equal. Some futures even contain no useful information at all.

By virtue of the WritingPrompts dataset containing many uncurated stories from the Internet, there are many candidate futures which contain short or uninteresting text (4.2). These are typically exclamatory one-liners or quips in a dialogue sequence such as "Occasionally." or "Shit!". Such futures cannot be considered plot events because they don't contain any notion of a subject or an action. Additionally consider futures that are composed entirely of stopwords such as "It is what it is.". Without any story specific knowledge, this sentence is interchangeable with many other sentences that contain similarly useless information.

Without further human annotation, it's difficult to discern informative sentences from uninformative ones heuristically. Length, for example, is not a strong indicator of a highly

informative future because long sentences can contain a lot of descriptive language without providing any expository details. By contrast, short sentences, such as inquisitive dialogue, can inform the decoder that a story entity might soon ask a question so it should provide the intermediate details leading up to such an event.

Mean IDF for Information Maximization

Our first intuition was to look at term frequency. Term frequency refers to the number of times a particular term appears within a document. A document, with respect to the WritingPrompts dataset, refers to a single ground truth story. Term frequency is the basis for many algorithms in information retrieval, particularly those used in document ranking and similarity (4.4).

Term frequency alone only measures the rarity of words within a single document. To extend the notion of term frequency to an entire dataset, Spärck Jones proposed the inverse document frequency metric [15]. Inverse document frequency (IDF) for a term t is computed as the logarithmic ratio of the total number of documents in a corpus N to the number of documents containing at least one instance of t.

$$\text{tf}\,(\hat{t}, d) = \sum_{t_i \in d} \begin{cases} 1, & \text{if } t_i = \hat{t} \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

$$\text{idf}(t, D) = \log \frac{\sum_{d \in D} 1}{1 + \sum_{d \in D} \begin{cases} 1, & \text{if } \text{tf}(t, d) > 0 \\ 0, & \text{if otherwise} \end{cases}} \tag{4.5}$$

We devise a heuristic for selecting informative futures by using averages over the idf score of each term t in a future candidate f. Each sentence extracted from 4.1 represents a document in a corpus D. This corpus includes sentences extracted directly from story contexts as well. The intuition behind this is that information contained within a context is less imperative to the future embedding. For example, consider a context that discusses the daily events of a character named "John" and a future event "John called Sarah when he got home.". Because John's name can be inferred from the context, it's less informative than "Sarah" who may not have been mentioned in the context.

To select a future, each future candidate is first tokenized into words. We removed stopwords from each tokenized future because they effectively imposed negative weighting

on longer sentences that naturally contained more of them. Of the remaining words in the future, we calculate the idf score for each of them. The mean of the resulting scores acts as a weight with which we use to greedily select a future. Future candidates with rarer words, and thus higher scores, we presume to contain more information to pass to the decoder.

4.2.2 Context Length

Similar to futures, we presume story contexts to contain variable amounts of information. By nature of the source from which it was scraped, many stories in the WritingPrompts dataset can be boring or uneventful simply because they were written in response to a boring or uninspiring prompt. For the same reason, however, some stories contain a lot of interesting information like unique character names, fantasy locations, or just uncommon word choice in general. We hypothesize that a disproportionately large amount of information within the context can conceal the effects of the conditioning on the output predictions.

4.2.3 Masking

Intermediate Masking

Because GPT-2 expects a source and target sequence of the same length due to its pretraining regimen on next token prediction, we could not simply pass a source context of five sentences in length and reconstruct a story of nine sentences in length. Additionally, because the goal of our task is to coerce the model into reconstructing the *intermediate* sentences, we cannot simply provide the decoder with these sentences without a risk of weakening the conditioning at inference time. We introduce a special token **MASK** with which we replace each token in the intermediate. The model derives no useful information by attending to previous positions within the intermediate, forcing it to use signals exclusively from the context and future during training to minimize the objective. At inference time, the decoder behaves as normal, appending each predicted token to the source sequence at the next step.

Probabilistic Context Masking

To further weaken the strength of the context, we introduce random context masking. We implement a preprocessing step that emulates a dropout layer in a feedforward network.

By concealing at random various words, we attempt to discourage the model from relying solely on the context during training. The model can optionally attend to the future at the masked positions in an effort to extract some insight from the future event. At inference time, the weights will have been optimized to attend to the future more frequently which further bolsters the effects of the conditioning.

4.3 Training Details

We train our model on one Nvidia RTX 2080 for 5 epochs. The entire procedure takes about 21 hours. The preprocessing step takes about 1 hour.

We used a batch size of 1 due to VRAM limitations but we accumulated gradients over 16 steps for a pseudo-batchsize of 16. We attempted a distributed training regimen over two devices but encountered issues with HuggingFace's DistributedDataParallel class that were nontrivial to resolve. Attempting distributed training resulted in the model training almost 20 times slower (around 400 hours).

4.4 Evaluation

Evaluating our model proved tricky as there is a lack of metrics in prior works for evaluating creative tasks such as *Future Sight*. Metrics which are typically used to evaluate generative language models, such as BLEU [26], were designed with translation tasks in mind. Translation emphasizes accuracy over diversity in generation which is counterintuitive for story generation. Additionally, there were no reliable metrics we could find in prior work with which to evaluate the conditioning effect of our model.

We developed classification tasks to compare *Future Sight* with a standard pretrained "vanilla" GPT-2 model of the same parameter specifications as our decoder. We construct a dataset consisting of ground truth contexts and futures along with predictions from both *Future Sight* and the vanilla GPT-2 model for the downstream classification task.

4.4.1 Human Evaluations

Our first evaluation step was to ask human evaluators to classify the predictions generated by each model. Provided with a context, a ground truth future, and predicted intermediate

text, the evaluators were tasked to choose from three classes. The first two classes were continuations of the context as predicted by *Future Sight*. The first *Future Sight* class represented predictions that were conditioned on the ground truth future extracted from the same story containing the context. The second *Future Sight* class represented predictions that were conditioned on the fixed future "The swamp creatures were relentless in their siege." This fixed future was chosen empirically as we observed the profound effect it had on conditioning that was easily identifiable. The final class represented examples which contained a predicted continuation of the context as generated by vanilla GPT-2. As vanilla GPT-2 cannot accommodate future information, the future is not used at inference time for examples in this category. We report accuracy, precision, recall, and f1 score metrics in Chapter 5.

4.4.2 Automated Metrics

Devlin *et al* trained BERT on a next sentence prediction (NSP) task as part of their semi-supervised training regimen to much success [9]. The training procedure for NSP with BERT involved ingesting two sequences separated by a special token **SEP**. The position zero output was attached to a downstream classification head to determine if the one of the two source sequences immediately preceded the other in ordinality. Motivated by this, we finetune a pretrained DistilBERT base model on a similar task to evaluate the conditioning effect of our model.

We provide the DistilBERT classifier with a similar task to the one presented to the human evaluators but with one deviation. Human evaluators made it clear that *Future Sight* examples conditioned on the ground truth future were not easily discernable from those continuations predicted by vanilla GPT-2. To further examine the efficacy of the conditioning, we opt to drop examples from the second class altogether and measure the two problem classes against each other directly. As there is an inherent alignment between ground truth futures and their respective contexts, it's not entirely unfounded to assume vanilla GPT-2 could construct a story similar to one generated by *Future Sight*. Our assumption is that the conditioned predictions will more rigidly align with the future and the model can learn an alightment between any overlapping information.

We reserve a set of 2916 unseen stories to train the classifier. We discard stories longer than the DistilBERT context window, leaving us with 2902 stories. For each story, we predict a three sentence continuation of the context using either *Future Sight* or vanilla GPT-2. The examples generated by *Future Sight* are conditioned on the respective ground truth future. We jointly encode each future and the predicted text, separated by the **SEP**

token. The training objective is to maximize the probability of the correct class given the jointly encoded future and prediction.

Inference over each story takes around 11 hours using only the CPU. We split our datapoints 80/20, resulting in 2322 training datapoints and 580 evaluation datapoints. We report accuracy, precision, recall, and f1 score metrics in Chapter 5.

Chapter 5

Results

5.1 Human Evaluations

We report accuracy, precision, recall, and f1 score for each participant across each class (5.2). Additionally, we report the aggregated intersectional metrics. Of all of the computed metrics, perhaps most notable was the high precision among all participants for stories conditioned on the swamp future. This indicates that participants were very confident in their decision to label a story with class 2, seldom misassigning the label. Recall was the lowest among predictions conditioned on true futures. As class 2 had high reported precision, it's reasonable to assume that many instances of stories conditioned on true futures were incorrectly labelled with class 3. Class 3, in essence, represents a "no conditioning" class. Class 1 datapoints conditioned on weak and uninformative futures could easily be falsely attributed to vanilla GPT-2.

Feedback from the human evaluators was generally consistent. Participants each noted that it was difficult to discern Class 1 from Class 3 in many cases. Participants noted that this often led to guessing at random between these two classes. Differentiating between classes 1 and 3 was particularly difficult for participant 2, who labelled 76 out of 100 examples with Class 3. Despite this, there were some cases within the 100 examples that the evaluators found to be obviously conditioned on the true future.

Context: "Welcome back, Mr. Jones". I blinked at the bright light until my eyes were able to focus on the overly-cheerful blonde standing next to me. I ran my tongue over my lips to try getting some moisture onto them. "Where am I?" I croaked.
Future Distance: 3 sentences
Future: I'm nurse Patkins, and I'm going to go over some stuff with you.
Prediction: He looked around me as if it were trying to figure out where he was. I tried to focus on the nurse, but the sound of the door opening and closing stopped me. "I'm in recovery from a cardiac arrest, Doctor?"

Table 5.1: Example a successful conditioning on the ground truth future. *Future Sight* is able to infer that the future event is in a hospital so it generates intermediate sentences to fill in those details.

5.2 Automated Metrics

The results of the DistilBERT classifier were better than random but not entirely compelling (5.3). We compare two models, one with a context masking probability of 0.4 and a second model with no context masking. There is some indication that the classifier can discern the presence of conditioning in the sampled predictions but there are likely additional factors which we can control for to boost the accuracy even higher. For example, there are many underlying issues with the dataset which we discuss in more detail in the coming section.

5.3 Discussion

Though the results of the above classification tasks suggest that *Future Sight* can successfully integrate a future event into a story, there are two key areas in which we believe it falls short. The first area is the lack of a good story dataset. We'll discuss the dataset we used for our experiments and how the results led us to this conclusion. The second area of improvement is the lack of a reliable metric to evaluate future conditioning.

5.3.1 Dataset Problems

Throughout the course of the development of *Future Sight*, we've identified several issues with the WritingPrompts dataset which we believe have negatively impacted the results

Class 1 — *Future Sight* (True Future)				
Participant	**Accuracy**	**Precision**	**Recall**	**F1**
1	0.5454	0.3871	0.5454	0.4528
2	0.1364	0.2500	0.1364	0.1765
3	0.3636	0.5000	0.3636	0.4211
4	0.7272	0.3636	0.7272	0.4848
5	0.4049	0.3750	0.4091	0.3913
6	0.4545	0.3846	0.4545	0.4166
Class 2 — *Future Sight* (Swampy Future)				
Participant	**Accuracy**	**Precision**	**Recall**	**F1**
1	0.7500	0.7742	0.7500	0.7619
2	0.3750	1.0000	0.3750	0.5454
3	0.5938	0.8636	0.5937	0.7037
4	0.5484	0.8947	0.5312	0.6667
5	0.4687	1.0000	0.4687	0.6382
6	0.6250	0.9091	0.6250	0.7407
Class 3 — Vanilla GPT-2				
Participant	**Accuracy**	**Precision**	**Recall**	**F1**
1	0.5652	0.6842	0.5652	0.6190
2	0.8043	0.4868	0.8043	0.6065
3	0.8261	0.6129	0.8261	0.7037
4	0.5435	0.6757	0.5434	0.6024
5	0.7391	0.5573	0.7391	0.6355
6	0.7173	0.6346	0.7174	0.6734
Total — Aggregated (Micro)				
Participant	**Accuracy**	**Precision**	**Recall**	**F1**
1	0.6200	0.6152	0.6202	0.6177
2	0.5200	0.5789	0.4386	0.4991
3	0.6500	0.6588	0.5945	0.6250
4	0.6400	0.6447	0.6007	0.6219
5	0.5800	0.6441	0.5390	0.5869
6	0.6300	0.6428	0.5990	0.6201

Table 5.2: Results of the classification task given to human evaluators.

Model	$p(mask)$	Accuracy	Precision	Recall	F1
Future Sight	0.4	0.6259	0.6981	0.5745	0.6303
Future Sight	0.0	0.6404	0.6472	0.6644	0.6557

Table 5.3: Results of the binary classification class given to DistilBERT.

Context: It happened suddenly and at first I thought I had died. Finally I could sense something beyond the chemical sentences to yelled at me to deliver. I felt something. I felt. I was ripped away from the natural flow into something I never wanted or ever even thought to seek.
Future Distance: 3 sentences
Future: I found myself, I existed.
Prediction: I could feel it. And I was alone. I was in my room and I knew.

Table 5.4: Example of a story conditioned on an ambiguous future.

overall.

As a collection of uncurated stories from an online forum, there is extensive markup, overuse of punctuation, incomplete dialogue, and, bizarrely, text that has been modified to look unsettling by being obfuscated with unicode diacritics. Though some of these outliers are simple enough to catch and discard, some of them are nontrivial to deal with. For example, a sentence selected as a future which contains long spans of punctuation with no real substance would only stifle *Future Sight* as it attempted to extract information from it.

Additionally, some of the stories in the WritingPrompts dataset are very ambiguous or philosophical (5.4). For example, there are many stories written as first-person narratives containing the "stream of consciousness" literary device. Though occasionally interesting, narratives styled in this way often lack any expository details regarding an overall plot for the model to focus on. As *Future Sight* attempts to condition on a future extracted from such a story, it becomes unclear to readers whether or not the conditioning effect is present at all.

5.3.2 Lack of Automated Metrics

From these results we've also established that there's a lack for a reliable automated metric for our proposed task. Though the results of classification are interesting and can indeed verify that something is happening, measuring the strength of the conditioning itself is

nontrivial. Additionally, automating the evaluation of fluency and premise continuity has always been difficult for story generation researchers. Though human evaluators can provide an assurance that what is being written is seemingly legitimate, humans can be unreliable. Moreover, as stories are often deeply intertwined with culture, results could vary by the region surveyed.

Chapter 6

Conclusion

6.1 Summary of Work

In this work we propose a new task within Automated Story Generation called *future conditioning*. Future conditioning is the process of coercing a generative language model towards temporally interpolating between a story context and a future plot event with the goal of integrating it into the story. By providing a generative language model with a story context and a future event, the model is responsible for generating the intermediate details in the form of complete sentences. To this end, new plot events can recurrently be integrated into the story.

In pursuit of our proposed task, we developed *Future Sight*, a model architecture composed of two large pretrained language models connected by a nonlinear projection. By encoding a future plot event with a pretrained transformer encoder and allowing a pretrained transformer decoder to attend to it, we demonstrate that it is possible to dynamically guide pretrained transformers in generating stories. Though our results with human evaluators and a pretrained DistilBERT classifier indicated the presence of conditioning, there are still more adjustments that can be made to improve the performance of *Future Sight*

6.2 Future Work

We've identified several facets to address in future iterations of this work.

6.2.1 Development of a Metric

As previously stated, development of an automated metric for evaluating the effects of *future conditioning* would be nontrivial. We have considered looking at works within information retrieval and document ranking to evaluate similarities between predicted intermediate sentences and the future. Perhaps the transition from context to future is one that can be represented as a traversal through continuous space.

6.2.2 Other Datasets

We'd like to retrain the model with datasets that are both more curated and contain more informative text. We've discussed the idea of using a movie synopsis dataset for training *Future Sight* as movie synopses often contain new information with each sentence. By design, synopses are supposed to articulate several hours worth of film content into a short paragraph so their information capacity is quite high. We've identified the WikiPlots dataset as a good candidate as it has some reputation already from prior developments in automated story generation.

Additionally, we've considered the benefits of using a dialogue dataset for similar reasons as we would a movie dataset. Dialogue is often brief and "to the point" which is ideal for our model which expects highly informative futures as an input parameter.

References

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

[2] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in Neural Information Processing Systems*, 13, 2000.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] 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.

[6] 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.

[7] 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.

[8] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[9] 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.

[10] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[11] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[12] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[13] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

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

[15] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[16] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[17] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.

[18] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[19] Boyang Li, Stephen Lee-Urban, George Johnston, and Mark Riedl. Story generation with crowdsourced plot graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 598–604, 2013.

[20] Chunyuan Li, Xiang Gao, Yuan Li, Baolin Peng, Xiujun Li, Yizhe Zhang, and Jianfeng Gao. Optimus: Organizing sentences via pre-trained modeling of a latent space. *arXiv preprint arXiv:2004.04092*, 2020.

[21] Lara Martin, Prithviraj Ammanabrolu, Xinyu Wang, William Hancock, Shruti Singh, Brent Harrison, and Mark Riedl. Event representations for automated story generation with deep neural nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[22] Minsky Marvin and A Papert Seymour. Perceptrons, 1969.

[23] James R Meehan. Tale-spin, an interactive program that writes stories. In *Ijcai*, volume 77, pages 91–98, 1977.

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

[25] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*, 2016.

[26] 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.

[27] ME Peters, M Neumann, M Iyyer, M Gardner, C Clark, K Lee, and L Zettlemoyer. Deep contextualized word representations. arxiv 2018. *arXiv preprint arXiv:1802.05365*, 12, 1802.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] Hannah Rashkin, Asli Celikyilmaz, Yejin Choi, and Jianfeng Gao. Plotmachines: Outline-conditioned generation with dynamic plot state tracking. *arXiv preprint arXiv:2004.14967*, 2020.

[31] Mark O Riedl and Robert Michael Young. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications*, 26(3):23–31, 2006.

[32] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[33] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[34] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[35] Hinrich Schütze. Word space. *Advances in neural information processing systems*, 5, 1992.

[36] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

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

[38] 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.

[39] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[40] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[41] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[42] Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7378–7385, 2019.

[43] Yukun Zhu, Ryan Kiros, Rich 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 *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

APPENDICES

Appendix A

Stories with a Swampy Ending

A.1 This is Our Swamp

Context

"No! I refuse to let you do this!"

I turned to look at my wife's face, which was covered in the glistening drops of sorrow that were tears, her eyes red and voice hoarse from pleading and crying,

"Please don't go, there has to be another way to fix it!"

"Look around us!" I yelled back, gesturing wildly, "Look what I did!

Prediction

This is our town! We've been in the mud since the first raiders! We haven't been able to stop the invaders and their hordes."

The swamp creatures were relentless in their siege.

A.2 Monster Inside Me

Context

I lay in bed but I can not sleep. I graduate high school tomorrow. By some miracle that I'm convinced goes beyond just my hard work I am giving the valedictorian speech. I lay in bed with all the possible disaster scenarios swirling around in my head. What if... no, don't think like that.

Prediction

What if I am... not... the monster that has been ravaging me for weeks. I do not have the guts to even attempt to fight back, and I'm in a bunker. I do not even care what the monster thinks of me now.

The swamp creatures were relentless in their siege.

A.3 Lizards Can't Wear Socks

Context

I swear to God it took me at least a week to get through all of the socks. Funny enough, not one pair. Always something along the lines of left sock, right sock, left sock, right sock, right sock, right sock, left sock... and so on. I could practically feel my brain cells dying off one by one, but I hung in there, in the hopes of finding something fun. Well after I got through the socks, and I mean ALL of the socks, I was pleasantly surprised to find a stash of keys.

Prediction

They'd been found, in fact, by a large group of wild-eyed, red-eyed, lizard-like creatures roaming the woods. The group had been hunting for weeks now, but I'd only been following the trail for a week so I had no idea what they were looking for. They had taken down all of these things, and now, in an attempt to keep the humans from attacking, they were raiding our fortresses.

The swamp creatures were relentless in their siege.