SAPIENZA
UNIVERSITÀ DI ROMA

Cycle-Consistent Story Generation

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Corso di Laurea Magistrale in Artificial Intelligence and Robotics

Candidate
Amr Fathy
ID number 1848399

Thesis Advisor
Prof. Roberto Navigli

Co-Advisor
Dr. Rexhina Blloshmi

Academic Year 2020/2021

Cycle-Consistent Story Generation
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 15, 2021

Author's email: amru.a.fathy@gmail.com

Abstract

Even though we are currently experiencing a technological revolution, with virtual assistants becoming a vital part of it, many of these agents are still lacking when it comes to proper communication. Humans are natural storytellers, thus it is fitting that these Artificial Intelligence-powered agents are able to properly understand and tell stories.

Automated story generation is a field of AI where the goal is to create agents that tell *good* stories. A story being good is subjective and hard to define. In this thesis, we will discuss and deal with some well-defined metrics to measure the quality of generated stories.

Previous methods relied on symbolic planning and knowledge databases to generate stories, which made them domain-rigid and limitedly coherent. Recently, neural-based approaches have shown more promise in terms of the quality of generated stories and their ability to reduce the need for knowledge databases. Nevertheless, these systems still lack originality, long-term coherence, and other problems.

In this study, we try to improve story generation by leveraging the knowledge of large pretrained models and utilizing extra training signals through a cycle-consistency framework. More formally, we exploit the dual tasks of text generation and summarization as means to improve the performance of the story generation task. We show that the cycle-consistency framework offers better results in text overlap and text diversity metrics.

Contents

Chapter 1

Introduction

Storytelling is the act of telling or writing stories or narratives. Stories have long been a fundamental medium for information exchange, entertainment, learning, and teaching. Telling stories is a universal human experience since ancient times. Ancient cultures shared folklores, epics, legends, and myths through generations of oral storytelling. The goal of a good narrative is not only to create a mental image of some events but a way of understanding what we see and perceive [Welsch, 1998].

One of the oldest accounts of storytelling is the visual representations found in the Chauvet cave over 30,000 years ago [Clottes, 2003]. Media of storytelling have widely changed over time, be that as it may, humans always have been consumers of stories. Today, with the wide use of computers, stories can be printed on paper, shared on blogs and social media, or even recorded via images, audio clips, and videos.

It is no wonder that Artificial Intelligence (AI) scientists were drawn to study how to make creative programs that could tell stories. The question about computational creativity is not limited to story generation, but any activity that would be deemed creative by humans [Colton et al., 2009].

There is no universal definition for the task of Automated Story Generation. We define it as "using creative AI-based systems to produce coherent and fluent passages of text about a topic" as defined by [Fan et al., 2018]. Concretely, the goal is to generate a correlated – temporally and logically – chain of events and build the in-between connections using a fluent narrative.

Automated Story Generation is considered a problem of Natural Language Generation (NLG). NLG is a software process that consumes inputs – if any – and produces human-readable text. Some NLG tasks like machine translation and text segmentation might be considered to be less difficult than story generation. The reason behind this is that to generate a story, a model is required to keep track of concepts, previous experiences, events, characters with their goals and actions, locations, and even more – all of that while also being able to generate coherent, diverse, and fluent text.

Automatic story generators can prove useful in many applications. They can be used in virtual agents and chatbots to create a sense of rapport with human users through story-like conversations. In education, interactive story generators have been used to help children collaboratively generate a story [Robertson and Good, 2003] or even evaluate the quality of story writing attempts by students [Halpin et al., 2004]. Professionally, interactive story generators were used to elicit new ideas for writers [Clark et al., 2018]. In the field of entertainment, many solutions address the problem of procedural narrative generation for video games [Griffith, 2018]. [Martin, 2021] claims that story generators can be used by criminal investigators to hypothesize about different criminal scenarios.

Early methods for automatic story generation used a combination of hand-crafted grammar rules [Ryan, 2017; Rumelhart, 1975; Thorndyke, 1977; Wilensky, 1983] and classical planning techniques [Meehan, 1977; Dehn, 1981; Lebowitz, 1984, 1985; Cavazza et al., 2002; Porteous and Cavazza, 2009; Riedl and Young, 2010] to automatically generate stories. However, these methods suffer from unoriginality and being constricted to a limited domain, could not model logical relationships between events and entities to a certain extent, need massive amounts of knowledge bases, and could not search a large domain of possible scenarios/cases.

Recently, recurrent neural architectures dominated many NLP tasks including story generation. With the introduction of sequence-to-sequence models (§2.4) and attention mechanisms (§2.5), many tasks witnessed a great deal of improvement with story generation being one of them. Researchers based their approaches on recurrent-based language models (LMs) [Choi et al., 2016; Harrison et al., 2017]. Some extended the idea to use a hierarchical generation approach to better model the semantic dependencies between the story's events [Martin et al., 2018; Xu et al., 2018; Fan et al., 2018]. Others adapt experiences shared by human writers to computational methods [Yao et al., 2019; Yu et al., 2020]. We also see works that focus on visual storytelling [Huang et al., 2016; Wang et al., 2018; Kim et al., 2018], and works that build interactive storytelling systems to improve human engagement and stimulate new ideas for writers [Clark et al., 2018; Goldfarb-Tarrant et al., 2019].

One of the common problems encountered by RNN-based generation models is their inability to maintain long-term dependencies. Although they provide an improvement over planning methods, RNN methods still struggle to be consistent after a small number of sentences. Generating coherent text is also a problem. Some methods still can not fully "understand" and maintain logical relationships between events which leads to an unreasonable story. Additionally, RNN models are known to generate repeated text either copied from input distribution or sampled again and again from the output distribution [Herrera-González et al., 2020].

More recently, with the introduction of Transformer models [Vaswani et al., 2017], many of the problems of RNN networks were addressed. Transformers can process longer sequences with a high ability to manage long-term dependencies thanks to their self-attention mechanism. They can also produce more coherent text with less repetition. This led researchers to augment the story generation task with extra tasks to help Transformers overcome a specific problem when it comes to generation. Some focused on modeling emotions while generating the story [Brahman and Chaturvedi, 2020; Kong et al., 2021], others improved the diversity by ordering permuted sentences [Yu et al., 2021], and an effort was made to model sentence-level and discourse-level coherence [Guan et al., 2021]. Although Transformers improved a lot over RNNs, they still suffer from the same basic problems, like modeling logical relationships, and interconnecting events and characters to produce a coherent story.

In our work, we propose to use a Transformer foundation model [Bommasani et al., 2021] in a cycle-consistency framework. Cycle-consistency loss was popularized as a method to optimize Generative Adversarial Networks (GANs) to perform image-to-image translation in the absence of parallel data [Zhu et al., 2017]. Given a source domain X and a target domain Y, the goal was to learn a mapping $G\colon X \mapsto Y$, or an inverse mapping $F\colon Y \mapsto X$ coupled with a cycle- consistency loss $F(G(X)) \approx X$ (and vice versa). This method was also used in language translation to verify and improve translations by humans [Brislin, 1970] and machines [He et al., 2016] where it is known as "back translation".

We define consistency in our case as the generated story being semantically consistent with the human story. Our two domains are textual summaries and stories, and we use a foundation model (e.g., BART [Lewis et al., 2019] or T5 [Raffel et al., 2019]) to learn the mappings. The main research question of this thesis is:

> *Can the extra training signals obtained from exploiting the dual tasks of text summarization and generation, in a cycle consistency framework, be utilized to improve the performance of said tasks?* [1]

We propose a simple modification – compared to other works – of placing a foundation model in a cycle consistency framework and then study how that affects the process of story generation. We follow [Hou et al., 2019] and analyze the generated stories according to their consistency, coherence, and diversity.

A story is *consistent* if it adheres to the same theme along the generation process. We attempt to measure that by the lexical token overlap between the generated and

[1] In this thesis we focus on the story generation task and leave the summarization task for future work.

reference story using the BLEU score (§5.2.2). *Coherence* of a story relates to the logical relationships between events and sentence order. We study this criterion by measuring the perplexity (§5.2.1) of the generation model to test its ability to model the test data. Finally, *diversity* is the ability of the model to generate text with better and more vivid meaning, rather than just choosing words with high frequency. We use a metric called DISTINCT-n (§5.2.3) to measure the percentage of unique words in the corpus.

In this thesis, we make the following contributions:

1. We cast the story generation task from a seq2seq task, into a new formulation where a model generates an intermediate output $\mathbf{y}$ which is further enhanced by the signals of a second model, which is tasked to reconstruct the input $\mathbf{x}$ starting from the previously generated text $\mathbf{y}$.

2. We propose a cycle-consistency-based model, $\textbf{Cycle}_{\text{GEN}}$, which aims to improve the task of story generation through exploiting extra training signals.

3. We report empirical results on two datasets (ROC Stories and Writing Prompts) that are widely used in the literature. We show that using the cycle model attains higher or comparative results with respect to other models, thus proving the effectiveness of our approach.

The rest of this thesis is organized as follows: In Chapter 2, we discuss the basic terminology, deep learning algorithms, and natural language processing (NLP) techniques. In Chapter 3, we explain, in detail, how previous works approached the story generation problem, where they come short, and how we differ. Chapter 4 is dedicated to explaining how the cycle consistency framework is applied and other details that we propose. Chapter 5 displays different experimental settings, discusses metrics, and shows empirical results followed by an analysis. Finally, we conclude and lay out further extensions to our work.

Chapter 2

Background

In this chapter, we develop the theoretical intuition for the basic approaches used in Deep Learning and Natural Language Processing. The goal is a concrete understanding of the foundation upon which our work is built.

2.1 Artificial Neural Networks

Scientists have long studied the human brain and sustained large interest in its versatility in learning. A substantial amount of research has gone into figuring out how the human brain can achieve such a high level of versatility. This has provided some useful insights into how the human brain works, however, we are still pretty far from fully understanding how a brain functions. In the field of artificial intelligence (AI), researchers have been able to provide some effective solutions to many problems by translating the observations and conclusions of biological research of the brain.

The human brain is part of the nervous system, which consists of nerve cells (or neurons). In fact, the brain is a huge interconnected network of neurons. The robustness of this huge network makes it virtually impossible to model it with the currently available technologies. The aim of computer scientists is, however, different. The goal is to rather simulate this thinking or learning process without the need to fully model or understand it.

An Artificial Neural Network (ANN) [McCulloch and Pitts, 1943] is a computing system designed to mimic the human brain. An ANN is a network of interconnected components or units called perceptrons. The connections between these units are modeled as numerical real numbers – weights. A positive weight indicates an excitatory connection, while a negative one indicates an inhibitory connection. These weights adjust according to the learning procedure and modify the input to produce an output. Finally, an activation function is applied to this output to control its magnitude.

The perceptron [Rosenblatt, 1958] is an algorithm for binary classification, where it performs a weighted sum of the (input) feature vector elements to decide whether or not an input belongs to some class (as shown in Figure 2.1). Concretely, the perceptron takes as input a real-valued vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and computes the following function:

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \sigma(\sum_{i=1}^{n} w_i \cdot x_i + b)$$

where $\mathbf{w}$ is the weight vector, and b is a bias term used to shift the decision boundary and is independent of the inputs. $\sigma = \dfrac{1}{1 + \exp(-x)}$ is the sigmoid activation function that squishes the output between 0 and 1 to create a probability output.

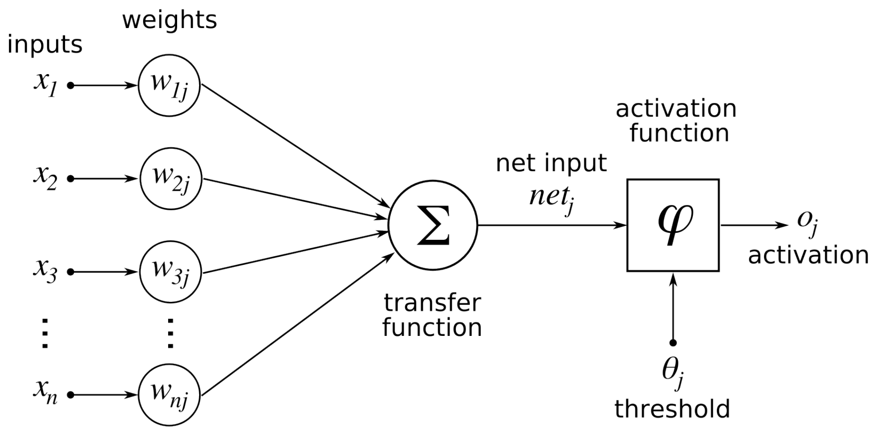


Figure 2.1. The Artificial Neuron (Perceptron) [1]

To learn more powerful models that can perform more complex tasks, the perceptron units are grouped in layers, and these layers are stacked to build deep neural networks (also called multilayer perceptron – MLP). Traditionally, there are three types of layers; input layers which receive the instances of the data, output layers which produce the results of the network's computations, and hidden layers which lie in between input and output layers and are responsible for computing hidden representations of the previous layers' outputs. The deeper (i.e. latter in the stack) the hidden layer, the more complex representations it is supposed to learn. A network is considered deep when it has two or more hidden layers. In this context, we are more interested in dense layers or fully connected layers, that is layers that have connections between all the neurons.

A deep neural network – shown in Figure 2.2 – is a machine learning (ML) algorithm that uses a hierarchical sequence of n parametric functions to model the

[1] `https://en.wikipedia.org/wiki/Talk:Artificial_neuron`

input $\mathbf{x}$. Each function f_i ($i \in \{1, 2, \ldots, n\}$) is modeled as a layer of neurons that apply the perceptron algorithm on the previous layer's output. As a generalization of the perceptron, each layer l_i is parameterized by a weight matrix $\mathbf{W}_i$ applied on the immediate input of l_i. Formally, a deep neural network computes the following function:

$$\hat{y} = f_n(\mathbf{W}_n, f_{n-1}(\mathbf{W}_{n-1}, \ldots, f_1(\mathbf{W}_1, \mathbf{x})))$$

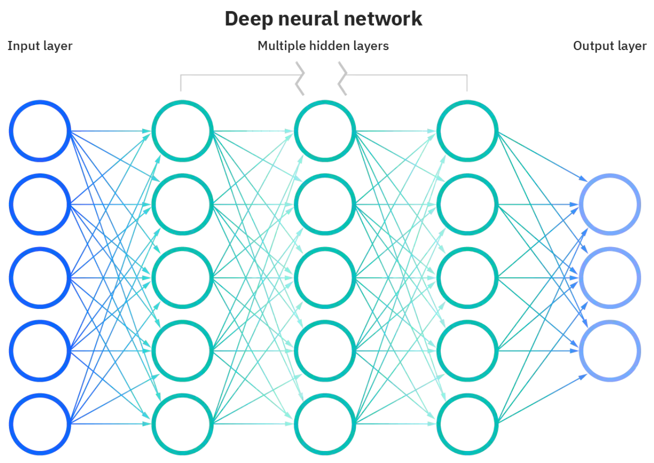


Figure 2.2. Deep Neural Network [2]

The ability to learn is the key concept of neural networks. During the training phase, the network tries to find the optimal parameters $\mathbf{W}_i$, $i \in \{1, 2, \ldots, n\}$ for solving a given task. Before the training process, the parameters are initialized to some random values which are tuned later according to some error measure. Learning is then carried through an iterative process where we feed the data to the network (called forward pass), analyze the outputs, and repeatedly tune the network for better performance. The outputs are analyzed through a cost function $J(\hat{y}, y)$ that measures the error between the prediction $\hat{y}$ and the true label y for the associated input vector $\mathbf{x}$. The weight tuning procedure is, formally, finding the set of optimal parameters that minimize the cost function J. This is done by using the backpropagation algorithm (BP), which computes the gradients of the cost function with respect to the network's parameters ($\nabla_{\mathbf{w}} J$), and then propagates them back from the output layer to the input layer to determine the weight update value according to the formula:

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^{t} - \eta \; \nabla_{\mathbf{w}} J$$

[2] `https://www.ibm.com/cloud/learn/neural-networks`

where η – the learning rate, defines the amount of weight adjustment. In the evaluation phase, the network's parameters are frozen and the predictions are obtained through a forward pass.

Activation functions are functions that map the input space to a different output space, and they control the information flow between layers in an MLP. These functions have to be continuously differentiable to allow gradients to flow in the backward pass of BP.

According to the universal approximation theorem [Csáji et al., 2001], neural networks are universal function approximators. That is, an arbitrary continuous function can be approximated using an MLP with one hidden layer and an arbitrary number of neurons in that layer. For this theorem to hold, the activation functions need also be non-linear, so that the output is non-linearly dependent on the input.

The most commonly used activation functions are `sigmoid`, `softmax`, `tanh` and `relu` among others.

2.2 Word Embeddings

Deep neural networks deal with numbers and not symbols. In general, an embedding is a representation of a symbol (e.g., word, character, or sentence) in a low-dimensional space of continuous-valued vectors. In natural language, words are not discrete isolated symbols, rather they are correlated with each other. Projecting these continuous-valued vectors into euclidean space visualizes the relationships between words. Depending on the task, we can learn the similarity (distance in Euclidean space) between different word representations.

Word2Vec [Mikolov et al., 2013] aims to learn a universal embedding matrix $\mathbf{E} \in \mathbb{R}^{V \times d}$ where V is the vocabulary size and d is the dimension of the vector representation. Each row $\mathbf{e}_i$ of $\mathbf{E}$ corresponds to the unique embedding of token i in the vocabulary V. This can be learned through one of two methods: continuous bag-of-words (CBOW) or skip-gram. In CBOW, the model predicts the current word from a window of adjacent context words, while skip-gram is the reverse.

Traditional word embeddings suffer from their inability to discriminate different meanings of the same words under different contexts [Camacho-Collados and Pilehvar, 2018]. For instance, the word *bank* can refer to two different meanings depending on the context: a financial institution or a sloping land near a body of water. Such words are said to be ambiguous and each meaning is called a word sense.

Sense embeddings attempt to model individual word senses into independent vector representations. This technique mainly relies on fixed sense inventories curated by linguistic experts, such as WordNet [Miller, 1995]. A sense inventory is a lexical resource (e.g., dictionary) that for each word lists all the possible meanings associated with it [Camacho-Collados and Pilehvar, 2018].

Contextualized embeddings techniques also attempt to solve the problem in an unsupervised way. ELMo [Peters et al., 2018] associates each token with a representation that is a function of the input sequence. These vector representations are obtained from a Bi-LSTM (§2.3.1) that is trained with an LM [3] objective. ELMo representations are deep, where they are a function of all internal layers of the Bi-LSTM. The higher-level layers capture context-dependent information, while lower-levels capture syntactical information.

2.3 Recurrent Neural Networks

Many problems attempted in AI are of sequential and/or temporal nature. Language texts, speech signals, stock prices, and successive video frames are just some examples of real-life applications. Humans normally reason about situations with respect to previous experiences. For example, when reading a language text, each word is thought of in the context of previous words (and maybe future ones). An MLP would be able to output a prediction for every single input, but would not be able to capture the temporal correlation between successive inputs.

To this end, recurrent neural networks were introduced [Hochreiter, 1991]. An RNN differs from a feedforward network by a feedback loop that connects the current input with past inputs to the RNN – as shown in Figure 2.3. Consuming its past outputs as current inputs, the RNN is said to have a memory mechanism where it can take into account past inputs into making a decision.

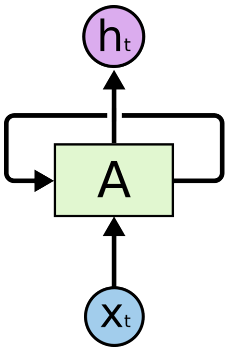


Figure 2.3. Recurrent Neural Network [4]

The sequential information is preserved in the hidden state of the RNN, which spans many time steps as it cascades forward to affect the processing of future

[3] A language model is a probability distribution over sequences of words.
[4] `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

inputs. This builds a correlation between temporal events that was impossible to do in MLPs. Since an event or input may depend on a far (in time) predecessor, these correlations are called "long-term dependencies". Mathematically this information cascade can be modeled as the recursive relation:

$$\mathbf{h}_t = f(\mathbf{W}_x \cdot \mathbf{x}_t + \mathbf{W}_h \cdot \mathbf{h}_{t-1})$$

where $\mathbf{W}_x, \mathbf{W}_h$ are weight matrices and f is an activation function. That is to say; the hidden state at the current time step $\mathbf{h}_t$ is a function of the current input $\mathbf{x}_t$ and the hidden state at the previous time step $\mathbf{h}_{t-1}$.

RNNs are trained through an extension of backpropagation called backpropagation through time (BPTT) [Werbos, 1988]. Time, in this context, is modeled as an ordered series of calculations linking one time step to the next, allowing backpropagation to work.

2.3.1 Bidirectional Recurrent Neural Networks

One of the flaws of vanilla RNNs is that it suffers from the limitation of using input information just up to the present time step. In word classification applications that rely heavily on context (e.g., named entity recognition), vanilla RNNs fall short. The limitation restraints the RNN from learning any information about the future context. In fact, an easy solution was proposed [Schuster and Paliwal, 1997] by stacking two RNNs, one working in the forward direction while the other works in the backward direction.

A Bidirectional RNN (Bi-RNN) is a composition of two RNNs, where the input sequence is fed normally to the first network and fed in reverse to the second one – as shown in Figure 2.4. At any time step t, the Bi-RNN would have learned two representations of the input text; h_{t-1} the hidden representation of the past inputs up to input x_t, and h'_{t-1} the hidden representation of the future inputs after the input x_t.

Mathematically, the information propagation is modeled similarly to the RNN:

$$\mathbf{h}_t = f(\mathbf{W}_x \cdot \mathbf{x}_t + \mathbf{W}_h \cdot \mathbf{h}_{t-1})$$

$$\mathbf{h}'_t = f'(\mathbf{W}'_x \cdot \mathbf{x}_t + \mathbf{W}'_h \cdot \mathbf{h}'_{t-1})$$

Bi-RNNs are trained similarly to RNNs since the two stacked networks are independent of each other. However, applying BP to Bi-RNNs is exceedingly slow due to the very long dependency chain.

[5]`https://colah.github.io/posts/2015-09-NN-Types-FP/`

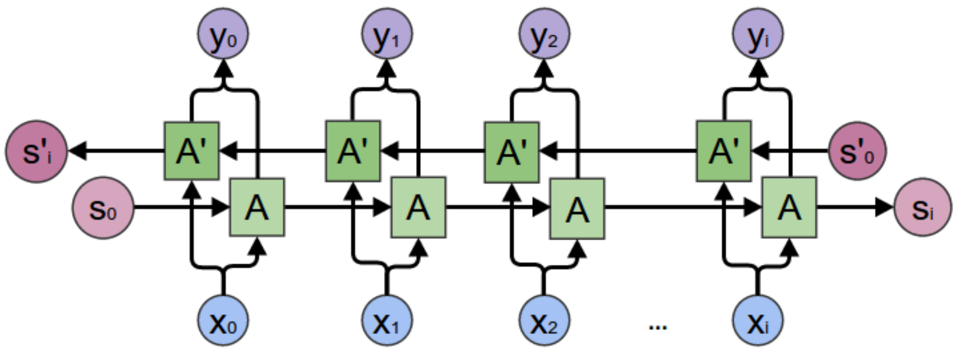


Figure 2.4. Bidirectional Recurrent Neural Network [5]

2.3.2 Long Short-Term Memory Networks

A fundamental problem for vanilla RNNs is vanishing or exploding gradients. As RNNs became deeper to process longer sequences, the successive multiplication of the gradients through the many layers leads to the gradients becoming exceedingly small or large, thus making the learning process unstable. Exploding gradients can be easily handled through clipping or squashing.

Vanishing gradients, however, can cause a network to stop learning by not updating the weights. Long Short-Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997] were designed specifically to solve this problem. The LSTM unit replaces the standard RNN memory cell to preserve and control the gradient flow during backpropagation.

An LSTM unit as shown in Figure 2.5 consists of a cell and a gating system that controls the information flow through the cell. This control is based on the information's magnitude and importance, through a filtering system that is adjusted through the training process. This filtering system is composed of three gates; forget gate, input gate, and output gate.

The core idea behind LSTMs is manipulating the cell state by having the ability to remove, retain or add information to the cell state, regulated by the three gates. Gates learn to optionally let information through. They are composed of a sigmoid layer and a pointwise multiplication operation. The sigmoid output produces a weight [0, 1] of how much information should flow through the gate.

The forget gate decides what information is going to be removed from the cell state. It observes h_{t-1} and x_t, and outputs a weight for every value in C_{t-1}. This mechanism is useful when the context changes for example. If our subject changes from one sentence to another, we want the LSTM to forget about the old subject

[6]`https://www.mdpi.com/2079-9292/10/9/1026`

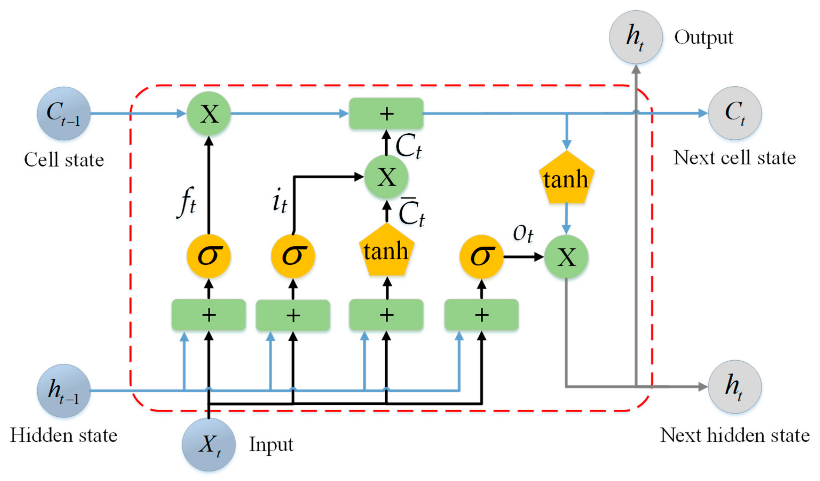


Figure 2.5. LSTM Unit [6]

and pay attention to the new one. The forget gate learns the following function:

$$f_t = \sigma(\mathbf{W}_f \cdot [h_{t-1}, x_t] + b_f)$$

In order to register new information, the input gate is used to update the cell state. This is a two step process, first, we decide which values to update. Then, we create a candidate vector $\tilde{C}_t$ that would replace some elements in the cell state.

$$i_t = \sigma(\mathbf{W}_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(\mathbf{W}_C \cdot [h_{t-1}, x_t] + b_C)$$

Next, we perform the cell update. Intuitively, the equation to update the cell state is:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, the output gate decides which pieces of the cell state to filter out and which to pass through as the output of the LSTM unit. We squish the cells state with a `tanh` activation and multiply it by the learned output gate to obtain the new hidden state h_t.

$$o_t = \sigma(\mathbf{W}_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Training an LSTM network means repeatedly adjusting the weights for each gate to learn the optimal behavior to solve a specific task. A popular variant of the LSTM is the Gated Recurrent Unit (GRU). The GRU combines the forget and input gates into a single update gate, and it merges the cell state and the hidden state.

The resulting model is much simpler and faster to train without much compromise on performance.

It is worth noting that while feedforward networks map one input to one output, recurrent structures support a variety of mappings; one-to-many, many-to-one, and many-to-many.

2.4 Sequence-to-Sequence Architectures

Many applications in NLP are in the form of sequence transformations, that is transforming the input sequence to an output sequence (e.g. machine translation, text summarization, conversational agents), where both sequences can be of arbitrary length. A Sequence-to-Sequence architecture (Seq2Seq) as shown in Figure 2.6 consists, mainly, of two networks; an encoder network and a decoder network. In the context of NLP, these networks are typically RNNs or a variant (§2.3).

The seq2seq model works, first, by scanning the input sequence, token-by-token, with the encoder network. The encoder network is used to "understand" the input sequence and learns an associated fixed-size vector representation (context vector) of the whole sequence. The decoder network is initialized with this context vector and proceeds to generate an output sequence, in a token-by-token manner. This decoupling of reading and generation works well in practice on a variety of problems.

Seq2seq models were first introduced [Sutskever et al., 2014] in the context of machine translation. This new LSTM based model was able to achieve state-of-the-art (SOTA) performance, without the need for any manual feature engineering. The advantage of seq2seq models is the ability to train them end-to-end, assuming the availability of parallel corpora.

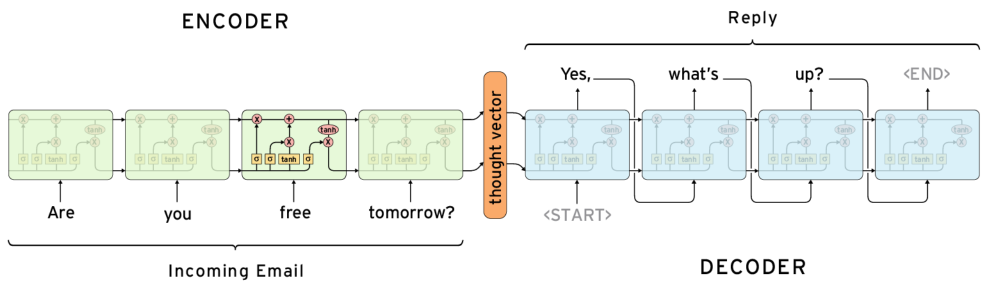


Figure 2.6. Encoder-Decoder Architecture [7]

The encoder network of the model is of autoencoding property, where it is trained to build a vector representation, of the whole input sequence, that should enable the network to reconstruct the input. In other words, the goal is to learn an efficient

[7] `https://suriyadeepan.github.io/2016-12-31-practical-seq2seq/`

encoding that stores the right pieces of information required for reconstruction. Recently, models have been proposed that use the encoder part only of the architecture. Encoders are usually used for language understanding tasks, such as sentence classification or token classification.

The decoder network of the model is of autoregressive property, where it is trained to generate outputs depending on the previous inputs and the hidden state. The decoder defines a probability over the output **y** by decomposing the joint probability into conditional probabilities:

$$p(\mathbf{y}) = \prod_{t=1}^{T_y} p(y_t \mid \{y_1, \ldots, y_{t-1}\}, \mathbf{c})$$

with **c** being the context vector learned by the encoder.

Similarly, networks with decoder-only components have been in use. Decoders are usually used for language generation tasks, such as text summarization or dialog systems (chatbots).

2.5 Attention Mechanism

Although seq2seq models work well in practice, the attempt to compress the whole input sequence in a fixed-size context vector is very limiting and sometimes a difficult task. The encoder is very likely to forget some relevant information in longer input sequences. Also, the decoder may need to "focus" more on different parts of the input while generating the output at different time steps.

The attention mechanism was first proposed [Bahdanau et al., 2014] to help memorize longer source sentences in neural machine translation. The main concept behind attention is to enable the decoder to have a direct view of the source sequence via the entire sequence of encoder hidden states. These shortcut connections (between the encoder and the decoder) are learnable for each output time step.

Even though the context vector has access to the entire input sequence, its function is now different. Instead of trying to compress the input, it now controls the alignment between the source and target. This alignment is conditioned on: (1) encoder hidden states, (2) decoder hidden states, and (3) alignment weights which represent how well input and output tokens match.

The context vector is computed as a weighted sum of the encoder hidden states:

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

The alignment weights α_{ij} are computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \quad e_{ij} = a(s_{i-1}, h_j)$$

where e_{ij} is parameterized as a feedforward network conditioned on s_{i-1} – the decoder hidden state and h_j – the encoder hidden state. This network is trained jointly with other seq2seq components to learn the optimal alignment between target words and source words.

A variety of attention mechanisms have since been studied. We focus on a family of attention mechanisms called self-attention.

2.5.1 Self-Attention

Self-attention [Vaswani et al., 2017] is an attention mechanism that relates different positions of the same sequence to compute a contextual representation of that sequence. The scores of self-attention are computed in parallel which results in an order of magnitude improvement over RNNs, which need to process data sequentially.

To calculate the self-attention model, we create three distinct representations of the sequence $\mathbf{X}$; $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$ – query, $\mathbf{K} = \mathbf{X}\mathbf{W}_K$ – key, and $\mathbf{V} = \mathbf{X}\mathbf{W}_V$ – value, with $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ learnable weight matrices. This query-key-value concept is borrowed from information retrieval systems. When you search for a *query*, the search engine will map the query against a set of *keys* associated with possible results. Then an algorithm will return the best-matching results (*values*).

This process in information retrieval literature is called content-based lookup [Mitra and Chaudhuri, 2000], hence the name *content-based* self-attention. In the self-attention mechanism, this idea of retrieving an object is relaxed. We define a degree of similarity between our representations to be able to weigh our query. Instead of choosing where to look depending on the position in the sequence, we can attend to the content we need itself. That is why we need to separate keys from values; the keys are used to calculate the attention weights while the values define the information we *retrieve.*

We can now define the self-attention mechanism mathematically:

$$\text{Self-Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\Big)\mathbf{V}$$

where we notice the similarity measure using a scaled dot-product. The scaling is to prevent the problem of exploding gradients. The `softmax` activation is used to get the final attention weights as a probability distribution.

2.5.2 Multi-Head Self-Attention

Multi-Head Self-Attention [Vaswani et al., 2017] can be thought of, simply, as an ensemble of independent self-attention mechanisms running also in parallel. In this context, the output of a single self-attention model is called a "head", the outputs of different heads are concatenated and linearly transformed into the expected dimension (see Figure 2.7). This composition of heads allows the model to jointly learn different representations of the input which improves the final attention-based representation.

$$\text{Multi-Head}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \ldots; \text{head}_n]\mathbf{W}_O$$

where each head is the output of the scaled dot-product attention (§2.5.1).

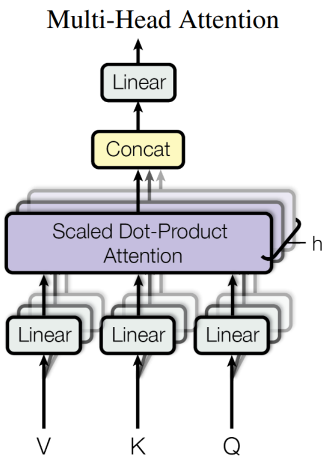


Figure 2.7. Multi-Head Self-Attention [8]

2.6 Transformers

Recurrent architectures process input data sequentially to build a vector representation that is used in downstream tasks. Several models have been proposed to improve this limitation and have sequential data processed in parallel. ByteNet [Kalchbrenner et al., 2016] and ConvS2S [Gehring et al., 2017] use convolutional neural networks (CNNs) as basic building block to compute hidden representations in parallel for input and output positions. In these models, the number of calculations required to relate two arbitrary input and output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more expensive to learn representations for longer sequences.

[8]Attention is all you need. [Vaswani et al., 2017]

The Transformer architecture was introduced [Vaswani et al., 2017] to solve this problem. Transformers are entirely built on the multi-head self-attention mechanism, without any recurrent units. The number of required calculations required to relate two arbitrary positions is now constant.

The Transformer is an encoder-decoder architecture – shown in Figure 2.8 – based on multi-head self-attention modules and normalization layers with residual skip connections.

An encoder block consists of: (1) multi-head self-attention layers (discussed in §2.5.2), (2) feedforward layers applied on the attention layer output to project it in a higher-dimensional space, (3) residual skip connections which are shortcut connections to add the input of the sub-layer to its output; this was found to improve gradient flow while adding more layers, and finally, (4) layer normalization [Ba et al., 2016] which independently normalizes the vector representation for each token – this improves convergence stability.

A decoder block is similar to an encoder block with exceptions. The first multi-head attention is modified to mask the latter positions to prevent the decoder from attending to subsequent positions during predicting the current token. The second multi-head attention takes input from both the previous sub-layer and the encoder.

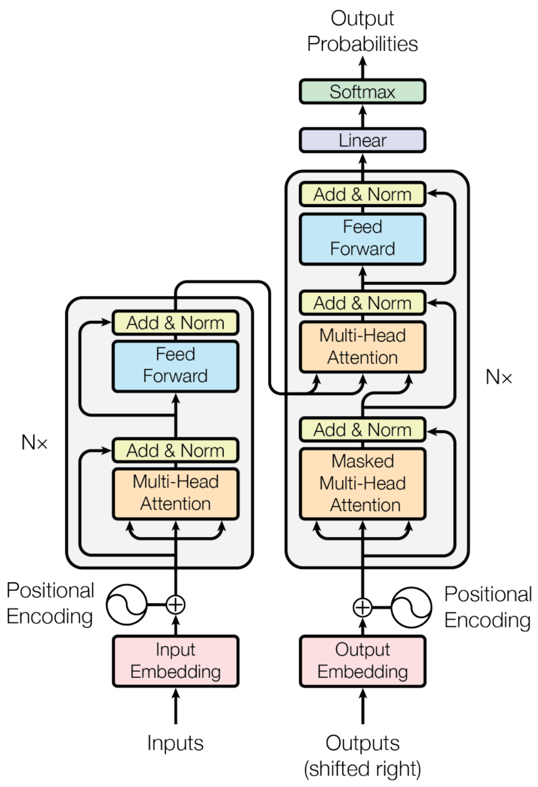


Figure 2.8. Full Architecture of the Transformer. Encoder (left) and Decoder (right). [9]

2.6.1 Positional Embeddings

All word embedding techniques (§2.2) incorporate information about the token's position in some way – a context window, a recurrence, or a convolution. However, in Transformers we completely remove the sequential processing mechanism of the RNN in favor of the parallelism of self-attention. In turn, this led to the loss of positional information for input tokens.

The authors of the original Transformer paper [Vaswani et al., 2017] introduced a function $f\colon \mathbb{N} \mapsto \mathbb{R}^d$ to compute the positional encoding p_t for a token at position t as follows:

$$p_t^{(i)} = f(t)^{(i)} = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

where $\omega_k = \dfrac{1}{10000^{2k/d}}$ and i is the dimension. This output vector p_t is then added to the input embeddings – where they have the same size.

This type of embeddings was chosen since: (1) it is unique and deterministic for each token, (2) allows the model to generalize to longer sequences than those encountered in training, and (3) allows the model to attend by relative positions efficiently.

Finally, thanks to the use of residual skip connections through the Transformer blocks, the positional information is efficiently propagated to higher-level layers where more complex interactions are handled.

2.6.2 Pretrained Transformer Models

The ability to train models quickly through enhanced hardware capabilities combined with improved learning algorithms has led to a paradigm shift in deep learning. In NLP, this shift came with the rise of transformers which could be trained faster, by order of magnitudes, than recurrent architectures and on longer sequences with better results.

Transfer Learning is a method of reusing the information (weights) learned by one trained model, on some task, in another model that is required to perform a different but related, task [Goodfellow et al., 2016]. This is done by using *pretrained* models in some way. One can either continue training those learned weights on the required task (fine-tuning) or simply freeze the weights and train only the output layer.

Transfer learning opened the window for accessible state-of-the-art models to be used in all scenarios without the difficult requirements of obtaining such models.

[9] Attention is all you need. [Vaswani et al., 2017]

BERT

One of the most famous pretrained transformer models is BERT which was developed by Google. BERT [Devlin et al., 2018] – or Bidirectional Encoder Representations from Transformers – outperformed other NLP models when it was released in many language-related tasks. It builds on top many clever ideas that were proposed by several researchers, like semi-supervised sequence learning [Dai and Le, 2015], ELMo [Peters et al., 2018], ULMFiT [Howard and Ruder, 2018], and the Transformer [Vaswani et al., 2017] among others.

BERT uses encoder blocks from the vanilla transformer and is most suitable for language understanding tasks (e.g., sequence classification, next token prediction). The first innovative addition in BERT is that it uses a bidirectional self-attention mechanism in contrast to left-to-right self-attention used in the vanilla transformer. This idea leads to an improved contextual representation for tokens. However, it also creates a problem where each word would be able to attend to itself in a multi-layered context, thus leading to corrupted training.

To solve this problem, the authors propose to randomly mask some of the input tokens and have the model learn how to reconstruct these tokens based on the context. This task is called Masked Language Modeling (MLM). The mask's value can be a generic value or sometimes a word is replaced with another to improve fine-tuning on downstream tasks.

Another pretraining task used is Next Sentence Prediction (NSP). Many downstream tasks require the understanding of the relationship between two sentences (e.g., Question Answering, Natural Language Inference). This task is cast to a binary classification task where BERT simply decides whether the latter sentence follows the first.

After pretraining BERT on the two previously mentioned tasks, it can be fine-tuned to solve a variety of language understanding problems. One other way to use BERT is as a feature extractor, where BERT is used to produce contextual word embeddings which are then fed to any model of choice on some related task.

BART

BERT-like models perform well where the prediction at position i is allowed to use information from later positions. This is, however, not practical in text generation tasks, where the prediction for position i can depend only on previous positions. On the other hand, autoregressive models, such as GPT [Radford et al., 2018], are pretrained to predict the next word while masking future positions making it more suitable for generation tasks. It follows that GPT-like models are unable to generalize well on downstream tasks where the whole input contributes information for the output.

BART [Lewis et al., 2019] combines the best of both approaches. It utilizes the full transformer architecture, with a BERT bidirectional encoder and a GPT autoregressive decoder, along with a variety of pretraining tasks. BART is pretrained on (1) token masking similar to BERT, (2) token deletion, where random tokens are deleted and the model must predict their correct positions, (3) text infilling, where a span of text (sequence of tokens) is masked by a single mask and the model must predict how many tokens were masked, (4) sentence permutation, where sentences of a document are shuffled in random order, and finally, (5) document rotation, where a document is rotated to start from a given token and the model is asked to predict the correct starting token.

BART performs best when fine-tuned on text generation tasks, but also works well for language comprehension tasks.

Chapter 3

Related Work

In cognitive science, story grammars view stories as scripts – structures that describe a sequence of events in a particular context [Schank and Abelson, 2013]. This comes from the fact that in most situations, events usually occur in stereotypical patterns. These patterns are used as schemas or grammars to guide story generation. Structured story generation is to automatically generate stories by dividing the stories into slots following a given schema. Slots are then filled by selecting similar plots from previously annotated stories [Alhussain and Azmi, 2021].

Propp's model was one of the earliest to formalize stories into structural models [Propp, 1968]. Propp studied hundreds of Russian folktales and concluded that they follow a similar global structure. He extracted 31 functions (character actions) of which a subset is sure to drive any given story.

Inspired by Propp's analysis, JE Grimes pioneered the field of automated story generation in the early 1960s. Grimes used the Monte Carlo approach to randomly sample a subset of Propp's functions and order them based on a set of intelligent grammar rules to build a story as seen in Table 3.1 [Ryan, 2017]. Back then story generators were non-learning rule-based systems that focused on the quality of a certain aspect (e.g., syntax or plot...). In the 1970s and 1980s, several researchers published story grammars which are necessary for story understanding and generation [Rumelhart, 1975; Thorndyke, 1977; Wilensky, 1983].

A LION HAS BEEN IN TROUBLE FOR A LONG TIME. A DOG STEALS SOMETHING THAT BELONGS TO THE LION. THE HERO, LION, KILLS THE VILLAIN, DOG, WITHOUT A FIGHT. THE HERO, LION, THUS IS ABLE TO GET HIS POSSESSION BACK.

Table 3.1. One of the first computer-generated stories by JE Grimes

Nevertheless, grammar-based approaches focus only on the syntax of the story rather than the semantics. Logical relationships between story events and character goals are ignored which affects the story coherence and believability [Alhussain and Azmi, 2021]. Also, structural models are restricted to a certain domain and cannot modify their knowledge rules to generate different stories.

The realization that story generation is a *creativity* problem begged the usage of more intelligent techniques. Story planners assume that narrative generation is a goal-driven process and apply a symbolic planner to generate a story. In this context, a story is viewed as a chain of causally related events (story points) that satisfy an end goal. Such systems are restricted to solve a theme and can easily delimit the path followed to generate a story [Herrera-González et al., 2020].

The first widely-known system to use a planner is TALE-SPIN [Meehan, 1977]. This system generates different characters with their respective goals and proceeds to find a resolution of these goals using an inference engine based on common sense reasoning theories. TALE-SPIN showed a substantial improvement over Grimes' fairy tales system, thanks to the "reasoning" capability provided by the inference engine.

Thanks to Meehan's work in TALE-SPIN and the modern formalization of symbolic planning in predicate logic, many works that use planners for story generation in different approaches came to be [Dehn, 1981; Lebowitz, 1984, 1985; Cavazza et al., 2002; Porteous and Cavazza, 2009; Riedl and Young, 2010]. However, many stories generated using planners had an inherent problem; each step depends only on the current state of the story and not past states.

Drawing from cognitive theories, computational analogy identifies similarities and transfers knowledge from a source domain to a target domain [Zhu and Ontañón, 2013]. This approach is applied in story generation systems by searching the knowledge bases for a story world state similar to the current story world, then the next state in the knowledge base would be the new state for the generated story. The similarity measure would depend, in this context, on the generation system.

MINSTREL [Turner, 1993] is one the earliest analogy-based generators. It is driven by character and author goals where case-based reasoning is used to resolve the goals. The system stores the cases in its memory and indexes them by important features such as locations and actions to group related cases. The system then adapts these cases/scenes through a model of computational creativity to ensure novelty. Similarly, MEXICA [Pérez and Sharples, 2001] uses a method of engagement and reflection, to use already known narratives and generate a new story that maintains coherence, novelty, and an increasing tension throughout the story.

To increase the variety of generated stories, the stories search domain needs to be expanded. However, as the search space increases, it becomes more difficult for planners to find a solution. Heuristic search techniques were used to find a story in a story search space. HEFTI [Ong and Leggett, 2004] and PlotGA [McIntyre and Lapata, 2010] use Genetic Algorithms to search for plot points and then generate a narrative. [Cavazza et al., 2009] use real-time A* for story generation in an interactive setting.

With the recent boom in learning algorithms and hardware capabilities, the focus started to shift from symbolic rule-based systems to learnable generation techniques. Machine Learning (ML) methods can be used for knowledge acquisition, in contrast to non-learning systems which need prior encoding of schemas and scenarios. Most ML systems, in the context of story generation, attempt to model, probabilistically, the process of writing a story in some way. After a probability distribution is learned, the model is then able to sample from this learned distribution to generate a new story.

Different works classify recent story generation systems according to different criteria. [Hou et al., 2019] use the constraints applied to the generation process to separate systems. [Herrera-González et al., 2020] survey different systems according to the type of guideline used to generate the story. [Alhussain and Azmi, 2021] differentiate between required tasks in the domain of story generation; filling the gaps, ending completion, and from-scratch generation.

We use Hou's system of classification which organizes story generators into theme-oriented models, storyline-oriented models, and human-machine interaction-oriented models [Hou et al., 2019]. Theme-Oriented models are characterized by a consistent theme throughout the whole generation process. Storyline-Oriented models are constrained by complete story plots to guide the generation. Story plots are forms of abstract descriptions of the development of the storyline, and can also be in the form of a given story that needs a suitable ending. Finally, Human-Machine Interaction-Oriented models are dynamic with user input guiding the generation process.

3.1 Theme-Oriented Models

Theme-oriented models refer to models in which the user constraint is static and gives a high-level description of the story's topic/theme. This type of models has the highest degree of flexibility and weakest constraints on the generation process which results in a difficulty of generating consistent, coherent, and vivid stories [Hou et al., 2019]. In this section, we review some of the methods that follow this paradigm.

[Choi et al., 2016] use two RNN encoder-decoder LMs to generate a story. The first network maps an input sentence to a vector representation, while the second network decodes this representation to generate the next sentence in the story.

[Harrison et al., 2017] modeled the chain of story events as a Markov Chain and used an RNN to guide a Markov Chain Monte Carlo (MCMC) sampler to generate new stories. RNN-based LMs proved their ability to generate grammatically correct sentences, however, generated stories lacked overall coherence.

[Martin et al., 2018] adopted the idea of hierarchically generating the story to improve long-term dependency and global coherence. Martin separated the task into two phases; first generate a sequence of events that outlines the progression of the story, then generate a narrative describing the relationships between the events. An event is a 4-tuple $\langle s, v, o, m \rangle$, where v is a verb, with s, o its subject and object respectively, and m is a wildcard which represents a dependency of some kind. To generate the sequence of events, an encoder-decoder network is used with a greedy decoding mechanism. To reconstruct human-readable sentences from the intermediate sequence of events, another encoder-decoder network is used with a beam search decoder to find optimal sentences.

[Xu et al., 2018] draw inspiration from classical schema-based methods and generate a skeleton – the most critical phrases that reflect the semantic connections between sentences. The skeleton is then further expanded into complete sentences. LSTM-based encoder-decoder networks (with attention decoders) are used for both phases; generating the skeleton from the input, and expanding the skeleton into a fluent story. However, unlike classical methods, skeleton extraction is trained using a reinforcement learning (RL) method to preserve only key information necessary to create a story. The skeleton model show improved coherence and story fluency.

Similarly, [Fan et al., 2018] use a two-step process to generate stories. First, they generate a premise describing the topic of the story then they condition the story generation on this premise. To generate the conditioning prompt, they build on top of the ConvS2S [Gehring et al., 2017] LM with a novel gated multi-scale self-attention mechanism to attend to different positions and be able to make better fine-grained selections. Next, to improve the dependency on the conditioning prompt, they use a fusion model – where a seq2seq network has access to the weights of a pretrained seq2seq network. The goal is for the second network to improve where the first network failed to learn.

[Yao et al., 2019] propose to dynamically generate the story plan adopting experiences from many human writers. Instead of generating the whole plan first then moving on to the story, they generate the plot point and then the corresponding sentence in the story – thus interleaving the plan and story generation one step at a time.

[Yu et al., 2020] use a modified conditional variational autoencoder (CVAE) in a multi-pass editing scheme to generate a story. The architecture uses a GRU-based encoder with a hierarchical decoder consisting of a global-level decoder responsible for guiding local-level decoders into generating story sentences. They use two CVAEs; the first generates a draft of the story based on the title, then the second consumes both the title and the generated draft to produce the final story. This scheme allows the second CVAE to focus on overall semantic consistency and coherence.

Many other works adopt this hierarchical generation framework [Ammanabrolu et al., 2019; Fan et al., 2019; Zhai et al., 2019; Ammanabrolu et al., 2020]. The reason for the wide adoption of this framework is the ability to model the relationship between plot points at a higher level than words and maintain long-term dependencies yielding improved results over standard seq2seq models in terms of global coherence and semantic consistency.

The recent success of Transformer models motivated researchers to pursue more complex enhancements to improve the generation process.

[Brahman and Chaturvedi, 2020] use COMET – a knowledge base – to identify the emotional reaction of the main character in each sentence, and then use this "emotional arc" to guide the generation. To correctly classify an emotion, they use a pretrained BERT model then finetune it on an emotion identification dataset then further finetune it on an emotion annotated (using COMET) story dataset. Next, they use this classifier loss to optimize the model (GPT-2) to continually align to the emotional arc using a reinforcement learning policy.

[Kong et al., 2021] use a similar approach where they use stylistic keywords as a higher-level representation for the story. They first learn the probability distribution for the stylistic keywords through an encoder and then use this distribution directly to condition the decoder while generation. The types of styles they consider are emotion-driven and event-driven styles. To extract emotion labels they make use of a lexical tool (NRCLex) and for the event labels, they use NLTK to extract non-common verbs. Finally, to guide the model during generation, they explicitly add a token that signifies the type of style the story should follow.

[Yu et al., 2021] aim to improve the diversity of output text by permuting the sentence order. Their method, PermGen, maximizes the expected log-likelihood of output paragraph distributions with respect to all possible sentence orders. To do so, they use three modules; (1) hierarchical positional embedding, which keeps track of the token's position in the sentence and in the paragraph, (2) sentence-permuted learning, which maximizes the log-likelihood of generating every possible paragraph permutation, and (3) sentence based decoding, which generates the sentences out of order then proceeds to order them according to their rank with log-probability.

[Guan et al., 2021] propose two additional pretraining tasks in addition to the language modeling task. The first task is a sentence similarity task where the model learns sentence-level representations to minimize a similarity score using SentenceBERT as a reference. The second task is a sentence ordering task where the model learns discourse-level representation to correctly determine whether two sentences are in order. Coupled with negative sampling, the goal for these extra tasks is to allow the model to effectively capture the high-level semantics and discourse structures in the context.

3.2 Storyline-Oriented Models

Storyline-oriented models are still characterized by a static user constraint, but in this case, the constraint directly describes plot points which are then used to generate the story. Conceptually, this type of models resembles the second part/phase of theme-oriented models (§3.1) where the input is a high-level representation of the plot points and the semantic connections between them.

[Huang et al., 2016] introduced the first dataset for visual storytelling (VIST). They present sets of pictures in sequences coupled with text captions that describe the pictures.

[Wang et al., 2018] use an adversarial reward learning framework on VIST to learn an implicit reward function from human experience, and then use this learned function to optimize the generation policy. In their architecture they make use of two models; a policy model that takes an image sequence and takes a sequence of actions (choosing words for the story), and a reward model that computes partial fine-grained rewards for each sentence representation along with the visual features learned from the input image.

[Kim et al., 2018] propose an attention-based approach to encode information between adjacent pictures in VIST. They use a global-local attention mechanism where the global attention focuses on overall semantic correlations while local attentions focus on extracting information from images to generate the story. To output sampled sentences, they use a custom decoding scheme where the probability of sampling a new word is inversely related to its frequency. This prevents repetitive text and insures diversity of generated stories.

Another method to encode storylines is in the form of some intermediate semantic representation. This representation can be a set of abstract sentences or event representations. The need to encode in some abstract representation arises from the fact that stories in their textual form contain many insignificant details and add extra overhead for the learning process.

[Martin et al., 2018] use a 4-tuple to represent an event (discussed in §3.1). [Mostafazadeh et al., 2016] make use of a novel annotation framework to simultaneously capture temporal and causal relations between events. [Jain et al., 2017] use a set of independent event descriptions to generate coherent stories. [Yao et al., 2019] create a story abstraction by extracting each sentence's most important word.

We add the task of predicting/generating an ending for a story in this section since it requires an understanding of the storyline up to that point. [Zhao et al., 2018] use RL to generate story endings that have high semantic relevance with story plots by applying a copy and coverage mechanism. [Hu et al., 2017] uses a context-aware LSTM to capture the temporal order of past sub-events and generate a prediction of a future sub-event. [Guan et al., 2019] argue that generating a rational ending for a story is an indicator of story comprehension since it requires understanding key storyline points and inherent commonsense knowledge.

3.3 Human-Machine Interaction-Oriented Models

This class of models deals with a dynamic user constraint. The input constraints vary with human interaction and the problem is cast more as an interaction problem rather than an algorithmic one.

[Clark et al., 2018] studied a machine-in-the-loop story generation framework where an LM suggests new sentences to stimulate ideas for the writer. They conclude that participants appreciated creative and diverse suggestions more than random less meaningful ones. [Goldfarb-Tarrant et al., 2019] use different varieties of interaction to improve human engagement. In their system, writers can outline the plot of the story, control the diversity of generated plots and story sentences, edit and modify previously generated content. They show improved user satisfaction with creative and relevant generated content.

3.4 Other Models

This section is dedicated to works that are not directly related to automated story generation but contributed to the development of this work.

As discussed previously, the idea of cycle framework was used before in the context of machine-generated language translation by [He et al., 2016]. They cast the problem as a two-agents game. The first agent (model) translates the message from language A to B and sends the translated message to the second agent. The second agent evaluates the readability of the received message (*not* translation correctness) with respect to language B, translates it back to language A and retransmits it back to the first agent. The first agent then evaluates the reconstruction quality and signals the second agent for feedback. The game is then started the other way around until both models converge. They use RL policy gradient methods to optimize models on the readability of the intermediate sentence and the quality of the reconstructed sentence. Since they use RL for optimization, the training procedure is unstable, the models need pretraining to be able to converge, and the training process needs a big amount of time.

[Baziotis et al., 2019] propose a text compression mechanism using an autoencoder architecture that uses differentiable approximations for text sampling which can use gradient-based optimization leading to better performance over seq2seq models. The compressor is a seq2seq with attention that adaptively determines the length of the compressed sequence. Then they use the Gumbel-softmax reparameterization trick with a straight-through estimator to sample words from the compressor's output distribution. In contrast to argmax sampling, using this sampling method allows for the backpropagation of gradients allowing a gradient-based optimization. The reconstructor is similarly a seq2seq that reconstructs the input again. However, to make the models converge they make use of multiple losses: (1) reconstruction loss, (2) prior loss, which is an LM loss that measures the readability of the compressed text, and (3) a topic loss, which is a TF-IDF based cosine similarity loss that forces the compressed text to draw words from the same distribution as the original text.

3.5 Position Of Our Framework

The majority of the hierarchical generation methods presented in (§3.1) are either recurrent or convolutional-based models which are generally outperformed by the recently introduced Transformers. Moreover, authors of these works still report tendencies to repeated text generation, inconsistencies, incoherency, inability to adhere to the theme, produce shorter stories, copy from input prompt, and focus on and generate more frequent words.

Storyline models (§3.2) adhere too much to the inputs resulting in less generalized and original stories. They are also highly dependent on the input when it comes to generating a coherent and logically consistent story. Additionally, they exhibit some of the problems of theme-oriented models.

Observing the Transformer-based approaches reviewed previously, we see that these methods use complex feature engineering that requires a considerable amount of extra computing power and/or training time. Instead, we propose a much simpler method that provides better results and is easily extendable.

The dual learning algorithm [He et al., 2016] uses an RL optimization procedure which is inefficient – since it required a long time to train the models – and unstable – where pretraining was required to initialize the models. In contrast, we use a gradient-based optimization algorithm, that proves to be more efficient when training bigger models.

The text compression mechanism [Baziotis et al., 2019] although efficient but still report instability – where the topic loss is a key factor for convergence. The model also tends to copy words from the original text to the compressed text. Additionally, they report that the model weakly recovers from early errors that have cascading effects.

Chapter 4

Cycle-consistency Framework

In this Chapter, we detail our approach to automated storytelling. First, we provide a formalization of the problem (§4.1). Then, we proceed to explain our baseline architecture (§4.2). Finally, we explain how we build on top of the baseline to get the cycle architecture and some possible extensions (§4.3).

4.1 Problem Formulation

Previously the task of story generation has been mostly viewed as a translation task, where a model is given as input a summary/topic and is required to produce a story. A simple schema is shown in Figure 4.1. More formally, given a conditional sequence $\mathbf{x} = (x_1, x_2, \ldots, x_{|\mathbf{x}|})$, that indicates a title, a topic or a short summary, the model is required to translate it into a sequence $\mathbf{y} = (y_1, y_2, \ldots, y_{|\mathbf{y}|})$ indicating a story that follows some constraints.

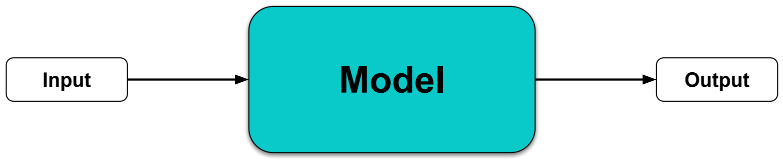


Figure 4.1. Traditional formulation for the story generation task

In our work instead, we re-formulate the problem in the light of the cycle-consistency framework as shown in Figure 4.2. To this end, given a conditional sequence $\mathbf{x} = (x_1, x_2, \ldots, x_{|\mathbf{x}|})$, a model is first required to translate it into an intermediate sequence $\mathbf{y} = (y_1, y_2, \ldots, y_{|\mathbf{y}|})$ and then another model translates $\mathbf{y}$ into an output (reconstructed) sequence $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{|\hat{\mathbf{x}}|})$. The goal for $\mathbf{x}$ and $\hat{\mathbf{x}}$ is to be as close as possible.

Forcing the original input and its associated reconstruction to be as close as possible leads the models to work together for achieving optimal performance. Indeed,

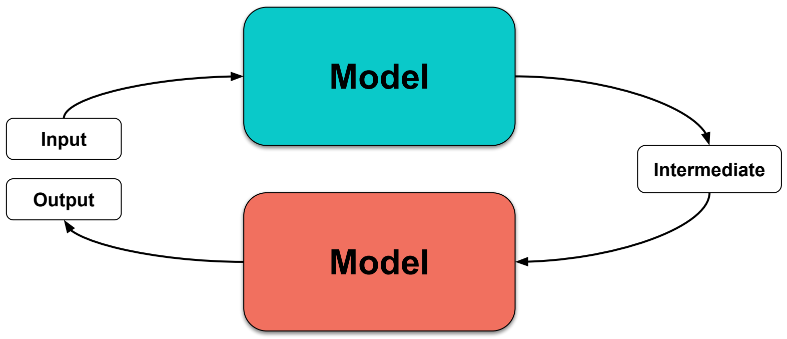


Figure 4.2. New formulation for the story generation task

the first model produces text that satisfies a certain task and also conveys the critical information for the other model to operate. The second model instead uses the critical signals in the intermediate text to accurately reconstruct the input.

Our proposed framework is general enough to consider any two tasks that are considered duals (e.g., translation and back-translation). In this thesis, we specifically study text generation (expansion) and summarization (compression) applied to stories. The expanded text represents a story built from a number of sentences, while the compressed text can be a summary of the story or a title highlighting the theme.

4.2 Baseline Architecture

Since our main task of consideration is story generation, the baseline can be any seq2seq architecture like most of previous work (Chapter 3). The task is then formalized as a conditional generation task (see Figure 4.3) where we want to maximize the output probability distribution

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{|\mathbf{y}|} p(y_i \mid \mathbf{y}_{<i}, \mathbf{x})$$

where $\mathbf{x}$ is the conditional input sequence (summary) and $\mathbf{y}$ is the output sequence (story).

We use BART [Lewis et al., 2019] as our baseline architecture (discussed in §2.6.2). Using BART gives us the advantage of using the massive knowledge gained from the corpora used to pretrain the model.

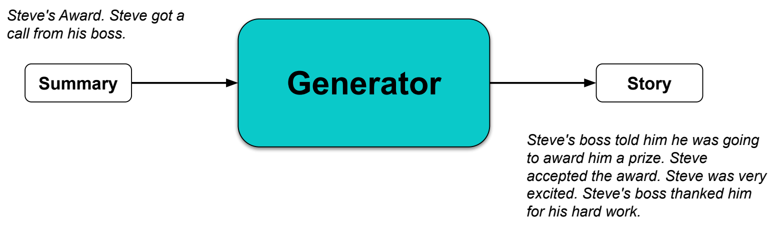

Steve's Award. Steve got a call from his boss.
Summary
Generator
Story
Steve's boss told him he was going to award him a prize. Steve accepted the award. Steve was very excited. Steve's boss thanked him for his hard work.

Figure 4.3. Baseline seq2seq model

4.3 Cycle Architecture

The key concept in this architecture is simplicity. We simply augment the baseline architecture (§4.2) with another seq2seq model, connect them and let them interact.

The first model performs a conditional generation task where it takes an input sequence $\mathbf{x}$ and maximizes a probability distribution $p(\mathbf{y} \mid \mathbf{x})$ for an intermediate sequence $\mathbf{y}$

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{|\mathbf{y}|} p(y_i \mid \mathbf{y}_{<i}, \mathbf{x})$$

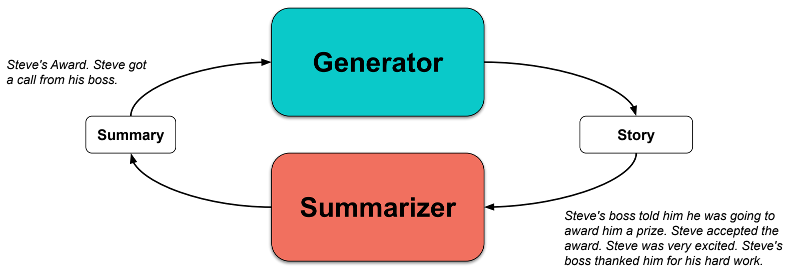
Then the second model samples its input $\mathbf{y}$ from the intermediate probability distribution $\mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$ and in turn maximizes a probability distribution $p(\hat{\mathbf{x}} \mid \mathbf{y})$ for an output sequence $\hat{\mathbf{x}}$

$$p(\hat{\mathbf{x}} \mid \mathbf{y}) = \prod_{i=1}^{|\hat{\mathbf{x}}|} p(\hat{x}_i \mid \hat{\mathbf{x}}_{<i}, \mathbf{y})$$

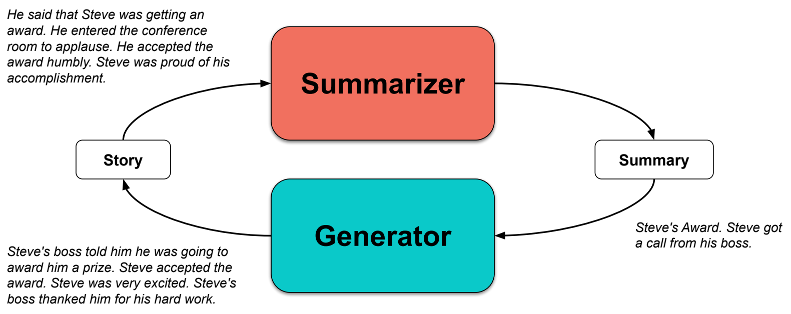
Similar to the baseline, we use BART as our architecture for both models. How to sample the intermediate sequence $\mathbf{y}$ from the probability distribution is discussed in the next section (§4.4).

Due to the cyclic nature of the framework, we can set $\mathbf{x}$ and $\hat{\mathbf{x}}$ to be the original and reconstructed summaries respectively, and set $\mathbf{y}$ to the generated story (see Figure 4.4a); and also vice versa where $\mathbf{x}$ and $\hat{\mathbf{x}}$ are the original and reconstructed stories respectively, and $\mathbf{y}$ to the generated summary (see Figure 4.4b).

An immediate observation from our proposal is the possibility to obtain two good-performing models instead of one; one performs well in expansion (generation), while the other performs well in compression (summarization). However, we only analyze the effect on the expansion process and leave the compression for another study.


Generator
Steve's Award. Steve got a call from his boss.
Summary
Story
Summarizer
Steve's boss told him he was going to award him a prize. Steve accepted the award. Steve was very excited. Steve's boss thanked him for his hard work.

(a) First direction: Summary → Story → Summary


He said that Steve was getting an award. He entered the conference room to applause. He accepted the award humbly. Steve was proud of his accomplishment.
Summarizer
Story
Summary
Generator
Steve's boss told him he was going to award him a prize. Steve accepted the award. Steve was very excited. Steve's boss thanked him for his hard work.
Steve's Award. Steve got a call from his boss.

(b) Second direction: Story → Summary → Story

Figure 4.4. Cyclic nature of the framework

4.4 Differentiable Sampling

To generate the intermediate sequence $\mathbf{y}$, we need to sample the words $(y_1, y_2, \ldots, y_{|\mathbf{y}|})$ from the first model's output categorical distribution:

$$p(y_t \mid \mathbf{y}_{<t}, \mathbf{x}) = \mathrm{softmax}(v_t)$$

where v is the unnormalized output (logits) of the model. The simplest method to do so is to use greedy decoding where at each time step t we look for the token with the highest probability:

$$y_t = \mathrm{argmax}\; p(y \mid \mathbf{y}_{<t}, \mathbf{x})$$

Since this process uses the argmax operation it is non-differentiable. This results in a training mode similar to [He et al., 2016] where each model trains separately and they are related only by any information passed through the intermediate text.

To retain the ability to train the whole model using gradient-based optimization where the gradients can backpropagate through both models, we use the Gumbel-softmax (GS) reparameterization trick as an approximation of sampling from categorical distributions [Jang et al., 2016]. Sampling from the softmax distribution is equivalent to adding to each logit an independent noise factor ζ from the

Gumbel distribution G(0, 1) ($\zeta = -\log(-\log(x_i)),\ x_i \sim Uniform(0,1)$), and then applying the argmax operation:

$$y_t \sim \text{softmax}(v_t) \quad \equiv \quad y_t = \text{argmax}(v_t + \zeta)$$

Next, we follow [Goyal et al., 2017] in using a soft approximation of the argmax using a peaked softmax function:

$$e_t = \sum_{i}^{|V|} e(w_i)\ \text{softmax}((v_t + \zeta)/\tau)$$

where the input for each time step is a weighted sum of all the vocabulary's embeddings and $\tau \in [0, \infty[$ is the temperature used to control the softmax. As $\tau \to \infty$, the equation approaches a true argmax embedding. Therefore, for a finitely large τ, the obtained embedding is a continuous function of a linear combination of all the embeddings dominated by the most probable word.

By combining the Gumbel-softmax reparameterization trick with the soft argmax, we obtain embeddings that are continuous (thus, differentiable), and maintain the ability to sample from a distribution that adds randomness to make the model robust.

Straight-Through Estimator

In our application, we are constrained to sample discrete values (words) from the distribution. As a result, we can not use the continuous approximation shown previously. To this end, we make use of the biased Straight-Through Estimator [Bengio et al., 2013]. We use the argmax to discretize $\mathbf{y}$ in the forward pass, but in the backward pass, we compute the gradients using the GS soft approximation.

Chapter 5

Evaluation

In this section, we overview the datasets we use and what settings/formats they are used in (§5.1). Then, we discuss the metrics used to evaluate the models (§5.2). Finally, we report empirical results and comment on them (§5.3).

5.1 Datasets

We use two datasets that are popular and widely used in story generation related works.

5.1.1 ROCStories

We use ROCStories [Mostafazadeh et al., 2016, ROC] to train and evaluate our model. ROC is a dataset of short commonsense stories that focuses on: (1) building causal and temporal relations between commonly encountered day-to-day events that span multiple domains, and (2) high-quality non-fictional events that one could easily relate to, making storytelling models trained on it more believable.

ROC contains 98,161 five-sentence short stories paired with their titles. We use the dataset in two settings shown in Table 5.1. In the first setting, we use only the title as the input (summary) and the model is required to generate the five sentences. While the second is where we also concatenate the first sentence of the story to the title and the model is asked to predict a story of four sentences. Statistics for the two settings are shown in Table 5.2.

5.1.2 Writing Prompts

Writing Prompts [Fan et al., 2018, WP] was collected from an online community where users post story prompts and other users are allowed to respond freely with a story that follows the prompt. The prompts vary in terms of length, topic, and level

Setting	Summary	Story
# 1	Steve's Award.	Steve got a call from his boss. He said that Steve was getting an award. He entered the conference room to applause. He accepted the award humbly. Steve was proud of his accomplishment.
# 2	Steve's Award. Steve got a call from his boss.	He said that Steve was getting an award. He entered the conference room to applause. He accepted the award humbly. Steve was proud of his accomplishment.

Table 5.1. A sample story from ROC stories

	Input	Output
Setting #1	5	53.35
Setting #2	14.39	43.78

Table 5.2. Average number of tokens in the input and output for both ROC settings

of detail. The stories collected have to be more than 30 words, generally follow the prompt, and avoid profanity.

WP contains 303,358 prompt-story pairs and a prompt can have more than one story. An example is presented in Table 5.3. Similar to ROC, we explore two settings in WP. The first uses the dataset as-is during training with a maximum story length of 1024 and limits the story to a maximum of 150 tokens at evaluation time. The second limits the story to the first ten sentences (split using NLTK) and uses only a subset of the dataset which is discussed later.

Summary	The witch stole his heart, and replaced it with ice. Now he can not love, lest it melts.
Story	I stare at the retreating figure of the witch who had just cursed me. I feel the cold spread from the lump of ice now in place of my heart. Did she seriously just do this so I can't love again? Love doesn't create actual heat. Don't you understand the laws of thermodynamics? I shout after her slowly fading form. I see a minute shrug as she disappears from my eyeline and give out a weak gurgle as I fall to my knees. As the floor rises to greet my face, it occurs to me that she doesn't much understand the laws of biology either.

Table 5.3. A sample story from WP

	Input	Output
Setting #1	28.22	659.14
Setting #2	28.22	164.85

Table 5.4. Average number of tokens in the input and output for both WP settings

5.2 Evaluation Metrics

We use a combination of automatic evaluation metrics to assess the quality of the generated stories from multiple aspects.

5.2.1 Perplexity

We use perplexity as a method of intrinsic evaluation for the models. For autoregressive language models, perplexity is the inverse of the probability of the dataset normalized by the number of word tokens. Maximizing such probability is equivalent to minimizing the perplexity of the model. Intuitively, the perplexity of an LM is the level of uncertainty when predicting the next token [Huyen, 2019]. For models that use the negative log-likelihood loss, the perplexity can be computed as:

$$\mathrm{PPL}(X) = \exp\left\{ -\frac{1}{t} \sum_{i}^{t} \log p_\theta(x_i \mid \mathbf{x}_{<i}) \right\}$$

Generally, smaller perplexity scores indicate better text fluency.

5.2.2 BLEU

Bilingual Evaluation Understudy (BLEU for short) [Papineni et al., 2002] is a string-matching algorithm that is used for evaluating the quality of translations tasks. BLEU measures the correspondence between machine-generated translations and human-written translations.The driving idea is that the closer (more overlapping tokens) the machine translation is to the human's, the better it is.

The BLEU n-gram precision p_n is defined over a corpus $\mathcal{C}$ as follows:

$$p_n = \frac{\sum_{S \in \mathcal{C}} \quad \sum_{n\text{-gram} \in S} \mathrm{Count}_{clip}(n\text{-gram})}{\sum_{S' \in \mathcal{C}} \quad \sum_{n\text{-gram} \in S'} \mathrm{Count}(n\text{-gram})}$$

where S is a sentence or a segment of the corpus, and Count_{clip} bounds the count of an n-gram to the maximum count found in any single reference for that n-gram.

The overall BLEU score is calculated as a geometric mean of the precisions penalized by some brevity penalty to account for short sentences:

$$\text{BLEU} = \text{BP} \cdot \exp\left\{\sum_{i=1}^{N} w_n \log p_n\right\}, \quad \text{BP} = \begin{cases} \exp\left\{1 - \frac{|\text{ref}|}{|\text{hyp}|}\right\}, & \text{if } |\text{ref}| > |\text{hyp}| \\ 1, & \text{otherwise} \end{cases}$$

However, recently BLEU has been criticized for its word-to-word similarity matching making paraphrases or synonyms receive poor scores [Callison-Burch et al., 2006; Cífka and Bojar, 2018; Mathur et al., 2020]. Moreover, texts that contain the right phrases with a wrong/random order can still be considered good. This misleads the overall evaluation of the system.

5.2.3 Distinctness

Language models tend to generate generic texts that are frequently repeated in text corpora. This is attributed to the objective function used to optimize most LMs – which is optimizing for the likelihood of outputs given the inputs. Naturally, this leads to neural models picking outputs that maximize the conditional probability which can be a limited set of tokens [Li et al., 2015]. We follow the work of [Li et al., 2015] and compute the DISTINCT-n metric to evaluate the diversity of the generated text.

This metric captures the percentage of unique n-grams as follows:

$$\text{DISTINCT-}n = \frac{\text{number of unique } n\text{-grams}}{\text{total number of } n\text{-grams}}$$

Having a higher distinctness/diversity score would indicate that the model is able to generate novel and interesting text. This is an essential property in storytelling to catch the attention of the reader.

5.3 Experiments

5.3.1 Implementation

Our framework can work with any seq2seq model in theory. For our experiments, we use BART as the base block in all variants of the model. We use the Transformers library by Hugging Face [Wolf et al., 2020] to implement our models. Due to limited resources, we use publicly available pretrained weights of $\text{BART}_{\text{BASE}}$ [1] to initialize our models. Our models are trained on Tesla P100 and V100 GPUs.

For ROC, the batch size for the baseline model is 64 and for the cycle model 32 with gradient accumulation set to 2 steps. In the first setting, the maximum story

[1]`https://huggingface.co/facebook/bart-base`

length is 110 tokens while for the second it is 90 tokens.

Similarly for WP, the batch size for the baseline is 16 and for the cycle model 8 with gradient accumulation set to 2 steps. In the first setting, the maximum story length is 1024 tokens during training and 150 tokens during evaluation while for the second setting it is 350 tokens.

All models are trained for 20 epochs using the early stopping mechanism that monitors the BLEU score on the validation set with a patience of 5 epochs. At evaluation time, we generate texts using beam search ($n = 5$) on top of nucleus sampling [Holtzman et al., 2019] with $p = 0.9$ and default temperature 1.0. A summary of the hyperparameters can be found in Table 5.5.

Parameter	Value
ROC	
Baseline batch size	64
Cycle batch size	32
Cycle gradient accumulation	2 steps
Max story length (setting # 1)	110 tokens
Max story length (setting # 2)	90 tokens
WP	
Baseline batch size	16
Cycle batch size	8
Cycle gradient accumulation	2 steps
Max story length (setting # 1)	1024 tokens
Max story length (setting # 2)	350 tokens
Generation	
Beam size	5
p	0.9
Temperature	1.0

Table 5.5. Hyperparameters used in different models

We test the cycle starting from the two possible endpoints Cycle$_{\text{EXP}}$ (Story $\rightarrow$ Summary $\rightarrow$ Story) and vice versa Cycle$_{\text{COMP}}$ (Summary $\rightarrow$ Story $\rightarrow$ Summary) – recall Figure 4.4. For ROC we report the best out of the two directions (which is Cycle$_{\text{COMP}}$) and for WP we report both for reasons outlined in the results section (§5.3.4).

5.3.2 Comparison Systems

In our experiments, we report results for some variants of our models:

- **Baseline** – the baseline model discussed in §4.2. A BART model that performs a seq2seq task with the summary as input and the story as output.

- **Cycle**$_{\text{COMP}}$ – the cycle architecture explained in §4.3 where we start with a summary, translate it to a story, and then back into a summary – recall Figure 4.4a.
- **Cycle**$_{\text{EXP}}$ – the same as **Cycle**$_{\text{COMP}}$, but in the other direction where we start with a story, translate it to a summary, and then back into a story – recall Figure 4.4b.
- **Cycle**$_{\text{COMP}}$**-GS** – the same as **Cycle**$_{\text{COMP}}$, but the intermediate text is sampled using the Gumbel-softmax reparameterization trick (§4.4) instead of greedy sampling.
- **Cycle**$_{\text{EXP}}$**-GS** – the same as **Cycle**$_{\text{EXP}}$, but the intermediate text is sampled using the Gumbel-softmax reparameterization trick (§4.4) instead of greedy sampling.

We further compare our system with other models from the literature:

- **Hierarchical** – the hierarchical story generation model by [Fan et al., 2018]. They use a ConvS2S LM to generate a premise to guide the story, then they use a seq2seq fusion model to generate the story from the premise.
- **H-CVAE** – the multi-pass CVAE by [Yu et al., 2020]. Their CVAE is built on the seq2seq mechanism with a GRU encoder and a hierarchical decoder that considers global and local semantics. The first CVAE generates a draft of the story from the title, and the second consumes the draft and the title to generate the final story.
- **HINT** – the multi-task model by [Guan et al., 2021]. In addition to the LM task, they add a sentence similarity task to learn sentence-level representations and a sentence ordering task to learn discourse-level representations.

5.3.3 ROC Stories Results

In Table 5.6, we show the results for the first setting of ROC stories, which we recall from Table 5.1, takes as input only the title of the story. B-1 and B-2 stand for BLEU-1 and BLEU-2 scores, and similarly for D-1 and D-2 with the DISTINCT-n score. PPL stands for perplexity.

Inspecting the results in Table 5.6, we see the clear advantage of Transformer-based methods over RNN/CNN methods. The baseline model easily achieves comparable performance to the best model (H-CVAE). The cycle framework improves over the baseline in terms of generating more token overlap with reference stories

Model	B-1 ↑	B-2 ↑	D-1 ↑	D-2 ↑	PPL ↓
Hierarchical	15.01	6.21	1.57	13.36	-
H-CVAE	29.39	11.02	1.99	14.82	-
Baseline	28.97	13.59	3.4	19.9	12.33
Cycle	**30.19**	**14.28**	**3.5**	**21.2**	13.919
Cycle-GS	29.6	13.97	2.9	17.2	**11.88**

Table 5.6. ROC results for setting #1

(shown by BLEU score), and more diverse and vivid texts (shown by DISTINCT-n score). We also see the cycle with differentiable sampling improving over the baseline perplexity-wise indicating the ease of modeling test data. We note that we report here the best cycle direction ($\text{Cycle}_{\text{COMP}}$) since it outperformed the other in most – if not all – the metrics.

In Table 5.7, we show the results for the second setting (see Table 5.1) of ROC stories, which we recall from Table 5.1, considers title+first sentence as the input to the model.

Model	B-1 ↑	B-2 ↑	D-4 ↑	PPL ↓
HINT	33.4	15.4	**69.3**	**9.2**
Baseline	35.26	18.26	58.1	11.107
Cycle	**37.3**	**20.48**	67.9	11.075
Cycle-GS	33.79	17.1	47.6	11.743

Table 5.7. ROC results for setting #2

As for Table 5.7, the task in this setting is easier, provided that the model is fed a richer input. Indeed, we see more improvement gaps when it comes to the BLEU score over other SOTA models. Although the cycle improves text diversity over the baseline considerably, there is still room for improvement. As explained earlier this can be attributed to the objective function of optimizing the likelihood of the outputs which generally benefits metrics like BLEU but can hurt text diversity metrics. This can be approached by using the Maximum Mutual Information as suggested by [Li et al., 2015] to bias to outputs that are specific to the inputs.

For the perplexity, we now see that the vanilla cycle offers a marginal improvement while the differentiable sampling extension is actually not better – suggesting that the stochasticity introduced by the Gumbel-softmax sampling is unstable and does not always offer better performance. Nevertheless, we expect HINT to have better perplexity since it was pretrained on BookCorpus [Zhu et al., 2015] before being finetuned on ROC.

5.3.4 Writing Prompts Results

We report the results for the first setting of WP, which we recall from Table 5.4 trains on stories of 1024 tokens in length, to analyze the effect of long training examples. In this experiment (Table 5.8), [Fan et al., 2018] report only the perplexity.

Model	B-1 ↑	B-2 ↑	D-1 ↑	D-2 ↑	PPL ↓
Hierarchical	-	-	-	-	36.56
Baseline	28.15	12.83	1.1	6.5	**21.24**
Cycle$_{\text{EXP}}$	28.61	13.61	1.0	6.9	21.65
Cycle$_{\text{EXP}}$-GS	28.48	12.92	**1.6**	**10.3**	21.97
Cycle$_{\text{COMP}}$	**31.19**	**13.8**	1.0	6.0	22.243
Cycle$_{\text{COMP}}$-GS	28.37	12.92	1.4	8.9	22.04

Table 5.8. WP results for setting #1

In Table 5.8, we discuss the results of training the cycle on longer training examples – but still testing them on short stories. We can observe how the cycle improved both text overlap and text diversity metrics over the baseline. However, in this case we notice a different behavior where (Cycle$_{\text{COMP}}$) has better BLEU scores and (Cycle$_{\text{EXP}}$) has better diversity scores. This would indicate that different directions extract different training signals, which can have varying effects on the training procedure. We also notice that unlike for ROC, the Gumbel-softmax extension here has a positive effect on diversity. This might be because that the differentiable sampling procedure works better for longer sequences. The perplexity results are, however, intriguing. It was expected that since the cycle helped improve both overlap and diversity metrics that the perplexity would follow – which is not the case. The reason behind this is not clear and needs more investigation.

In Table 5.9, we show the results for the second setting for WP, which we recall from Table 5.4, limits the story to the first ten sentences.

In the second setup (Table 5.9) we follow the parameters set by [Guan et al., 2021] to obtain comparable results. We use ∼30K stories to finetune our models, however, we directly notice that the models are overfitting the training data. We hypothesize that it is difficult for the model to train on longer sentences with fewer instances (compared to ROC). Additionally, [Guan et al., 2021] pretrained their model on BookCorpus before finetuning it on the ∼30K (10%) stories from WP. Due to these reasons, we increase our dataset to use ∼100K (33%) random stories from WP and still follow the rest of the parameters.

We obtain similar observations where (Cycle$_{\text{COMP}}$) has better BLEU scores and (Cycle$_{\text{EXP}}$) has better diversity scores. And with similar perplexity behavior, we attribute these findings to the bigger size of the dataset in terms of both summary and story lengths (compared to ROC). It would appear that in Cycle$_{\text{COMP}}$ the story

Model	B-1 ↑	B-2 ↑	D-4 ↑	PPL ↓
HINT	22.4	8.4	31.3	32.73
	30K stories			
Baseline	23.92	9.68	48.0	43.02
Cycle$_{\text{EXP}}$	22.41	8.9	61.4	43.25
Cycle$_{\text{COMP}}$	23.0	9.33	47.4	43.16
	100K stories			
Baseline	26.22	10.52	32.7	**40.05**
Cycle$_{\text{EXP}}$	27.3	10.84	**47.0**	42.51
Cycle$_{\text{EXP}}$-GS	28.39	11.3	45.4	42.57
Cycle$_{\text{COMP}}$	**29.63**	**11.79**	40.8	42.22
Cycle$_{\text{COMP}}$-GS	26.89	10.69	45.4	44.06

Table 5.9. WP results for setting #2

generators tries to produce as much token overlap as possible so that the summarizer can successfully reconstruct the summary again. However, when it is used in the other direction, the story generator has more freedom over the story and generates stories with more vivid text.

5.4 Qualitative Analysis

In this section, we study some cases of generated stories by different models. In the first example (Table 5.10), we see a sample from ROC – with the gold summary (input), gold story (output), and generated stories by the baseline and cycle models.

Gold Summary (input)	Steve's Award. Steve got a call from his boss.
Gold Story	He said that Steve was getting an award. He entered the conference room to applause. He accepted the award humbly. Steve was proud of his accomplishment.
Generated Story (Baseline)	He was given a big award for his hard work. Steve was so proud of himself. *He took the award home and presented it to his boss.* Steve's boss was very proud of him.
Generated Story (Cycle)	Steve's boss told him he was going to award him a prize. Steve accepted the award. Steve was very excited. Steve's boss thanked him for his hard work.

Table 5.10. Generated stories by different models for the same input (1)

For the story generated by the baseline, we see a case of reverse semantics where the employee gives an award to his boss. This is not the intended meaning (seen in the gold story). We see that the cycle could correctly capture the semantics in this example and produce a coherent story.

In the second example (Table 5.11), we see that the two cycle directions produce semantically different results. The Cycle$_{\text{EXP}}$ model failed to understand the global semantics of the story, thus producing an irrelevant story. This problem is not propagated by the other cycle model, confirming that the signals extracted by the two directions are different.

Gold Summary (input)	Nicole's Boyfriend Nicole did not have a serious boyfriend.
Gold Story	She went on many dates but was not committed. She was at a bar one night and met a man she was interested in. They began dating. Nicole just got engaged to be married.
Generated Story (Cycle$_{\text{EXP}}$)	Nicole did not have a boyfriend. Nicole decided to get a job. Nicole applied for a job at a restaurant. Nicole got a job as a waitress.
Generated Story (Cycle$_{\text{COMP}}$)	Nicole decided to go out with her friends. Nicole met a guy at a bar. Nicole fell in love with the guy. Nicole and her boyfriend are now married.

Table 5.11. Generated stories by different models for the same input (2)

In the third example (Table 5.12), we notice that none of the models correctly understand the meaning of the word "wake". The intended meaning is "a vigil held over a corpse the night before burial" [2] which means the death of someone and the feeling of sadness for the relatives and close people. Instead, we see two of the models understood the verbal meaning of "to stop sleeping", and the other could not convey the sentiment correctly. We argue that this is a hard example and more relevant information needed to be passed to the models.

[2]All definitions in this paragraph are according to WordNet.

Gold Summary (input)	Wake. Anette put on a wake for her grandmother.
Gold Story	Her grandmother had passed away the previous week. Anette put together a playlist of her grandma's favorite songs. They were all upbeat rock & roll tunes. Anette cried as she danced.
Generated Story (Baseline)	*She was very nervous to wake her grandmother.* But she was determined to make her grandmother feel better. She put on her grandmother's favorite song. Her grandmother was so happy to see Anette again!
Generated Story (Cycle$_{\text{EXP}}$)	She put the wake in her grandmother's bedroom. Anette went to the kitchen to make breakfast. She made pancakes for her grandmother. *Her grandmother thanked Anette for the wake.*
Generated Story (Cycle$_{\text{COMP}}$)	She was very nervous about it. *But when she woke up, she was relieved.* Her grandmother had passed away. Anette was glad she had put on a wake for her.

Table 5.12. Generated stories by different models for the same input (3)

Chapter 6

Conclusion

In this thesis, we present the cycle-consistency framework to the story generation task. To the best of our knowledge, this was the first attempt to do so. We provide a clear and intuitive new formulation for the story generation problem, that is easily understood and extendable.

Previously, the problem was perceived as a conditional generation problem where given an input sequence (summary) the model was required to output a story. Instead, we propose to generate first an intermediate text that is further improved by signals obtained from an input-reconstruction task, through the use of two models.

In addition, we experiment with two sampling methods; greedy sampling and Gumbel-softmax sampling. With greedy sampling, the two models are trained separately and combined only by the intermediate text, while using Gumbel-softmax sampling, makes the whole cycle fully differentiable which allows for better gradient-based optimization.

We prove that this formalization improves over others in several metrics. We show that the cycle model can extract meaningful signals that help with generating text with higher diversity and overlap with reference texts.

As future work, we intend to experiment with new external signals to guide the cycle model. We plan on introducing a semantic similarity signal which serves the purpose of making the generated story more semantically related to the reference. We also plan on using commonsense knowledge bases to improve the logical coherency of the generated text.

Bibliography

Arwa I Alhussain and Aqil M Azmi. 2021. Automatic story generation: A survey of approaches. *ACM Computing Surveys (CSUR)*, 54(5):1–38.

Prithviraj Ammanabrolu, Ethan Tien, Wesley Cheung, Zhaochen Luo, William Ma, Lara Martin, and Mark Riedl. 2019. Guided neural language generation for automated storytelling. In *Proceedings of the Second Workshop on Storytelling*, pages 46–55.

Prithviraj Ammanabrolu, Ethan Tien, Wesley Cheung, Zhaochen Luo, William Ma, Lara J Martin, and Mark O Riedl. 2020. Story realization: Expanding plot events into sentences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7375–7382.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

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

Christos Baziotis, Ion Androutsopoulos, Ioannis Konstas, and Alexandros Potamianos. 2019. SEQ3: Differentiable sequence-to-sequence-to-sequence autoencoder for unsupervised abstractive sentence compression. *arXiv preprint arXiv:1904.03651*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Faeze Brahman and Snigdha Chaturvedi. 2020. Modeling protagonist emotions for emotion-aware storytelling. *arXiv preprint arXiv:2010.06822*.

Richard W Brislin. 1970. Back-translation for cross-cultural research. *Journal of cross-cultural psychology*, 1(3):185–216.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. 2006. Re-evaluating the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.

Jose Camacho-Collados and Mohammad Taher Pilehvar. 2018. From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788.

Marc Cavazza, Fred Charles, and Steven J Mead. 2002. Character-based interactive storytelling. *IEEE Intelligent systems*, 17(4):17–24.

Marc Cavazza, David Pizzi, Fred Charles, Thurid Vogt, and Elisabeth André. 2009. Emotional input for character-based interactive storytelling. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, page 313–320, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

YunSeok Choi, SuAh Kim, and Jee-Hyong Lee. 2016. Recurrent neural network for storytelling. In *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 841–845. IEEE.

Ondřej Cífka and Ondřej Bojar. 2018. Are bleu and meaning representation in opposition? *arXiv preprint arXiv:1805.06536*.

Elizabeth Clark, Anne Spencer Ross, Chenhao Tan, Yangfeng Ji, and Noah A Smith. 2018. Creative writing with a machine in the loop: Case studies on slogans and stories. In *23rd International Conference on Intelligent User Interfaces*, pages 329–340.

Jean Clottes. 2003. *Chauvet cave: The art of earliest times*. University of Utah Press.

Simon Colton, Ramon López De Mántaras, and Oliviero Stock. 2009. Computational creativity: Coming of age. *AI Magazine*, 30(3):11–11.

Balázs Csanád Csáji et al. 2001. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28:3079–3087.

Natlie Dehn. 1981. Story generation after tale-spin. In *IJCAI*, volume 81, pages 16–18. Citeseer.

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

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

Angela Fan, Mike Lewis, and Yann Dauphin. 2019. Strategies for structuring story generation. *arXiv preprint arXiv:1902.01109*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

Seraphina Goldfarb-Tarrant, Haining Feng, and Nanyun Peng. 2019. Plan, write, and revise: an interactive system for open-domain story generation. *arXiv preprint arXiv:1904.02357*.

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

Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. 2017. Differentiable scheduled sampling for credit assignment. *arXiv preprint arXiv:1704.06970*.

Ioseff Griffith. 2018. Procedural narrative generation through emotionally interesting non-player characters. Master's thesis, Linnaeus University.

Jian Guan, Xiaoxi Mao, Changjie Fan, Zitao Liu, Wenbiao Ding, and Minlie Huang. 2021. Long text generation by modeling sentence-level and discourse-level coherence. *arXiv preprint arXiv:2105.08963*.

Jian Guan, Yansen Wang, and Minlie Huang. 2019. Story ending generation with incremental encoding and commonsense knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6473–6480.

Harry Halpin, Johanna D Moore, and Judy Robertson. 2004. Towards automated story analysis using participatory design. In *Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, pages 75–83.

Brent Harrison, Christopher Purdy, and Mark O Riedl. 2017. Toward automated story generation with markov chain monte carlo methods and deep neural networks. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. *Advances in neural information processing systems*, 29:820–828.

Brian Daniel Herrera-González, Alexander Gelbukh, and Hiram Calvo. 2020. Automatic story generation: State of the art and recent trends. In *Mexican International Conference on Artificial Intelligence*, pages 81–91. Springer.

Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Chenglong Hou, Chensong Zhou, Kun Zhou, Jinan Sun, and Sisi Xuanyuan. 2019. A survey of deep learning applied to story generation. In *International Conference on Smart Computing and Communication*, pages 1–10. Springer.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Linmei Hu, Juanzi Li, Liqiang Nie, Xiao-Li Li, and Chao Shao. 2017. What happens next? future subevent prediction using contextual hierarchical lstm. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Ting-Hao Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, et al. 2016. Visual storytelling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1233–1239.

Chip Huyen. 2019. Evaluation metrics for language modeling. *The Gradient*.

Parag Jain, Priyanka Agrawal, Abhijit Mishra, Mohak Sukhwani, Anirban Laha, and Karthik Sankaranarayanan. 2017. Story generation from sequence of independent short descriptions. *arXiv preprint arXiv:1707.05501*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Taehyeong Kim, Min-Oh Heo, Seonil Son, Kyoung-Wha Park, and Byoung-Tak Zhang. 2018. Glac net: Glocal attention cascading networks for multi-image cued story generation. *arXiv preprint arXiv:1805.10973*.

Xiangzhe Kong, Jialiang Huang, Ziquan Tung, Jian Guan, and Minlie Huang. 2021. Stylized story generation with style-guided planning. *arXiv preprint arXiv:2105.08625*.

Michael Lebowitz. 1984. Creating characters in a story-telling universe. *Poetics*, 13(3):171–194.

Michael Lebowitz. 1985. Story-telling as planning and learning. *Poetics*, 14(6):483–502.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*.

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

Lara Jean Martin. 2021. *Neurosymbolic Automated Story Generation*. Ph.D. thesis, Georgia Institute of Technology.

Nitika Mathur, Timothy Baldwin, and Trevor Cohn. 2020. Tangled up in bleu: Reevaluating the evaluation of automatic machine translation evaluation metrics. *arXiv preprint arXiv:2006.06264*.

Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Neil McIntyre and Mirella Lapata. 2010. Plot induction and evolutionary search for story generation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1562–1572.

James R Meehan. 1977. TALE-SPIN, an interactive program that writes stories. In *Ijcai*, volume 77, page 9198.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Mandar Mitra and BB Chaudhuri. 2000. Information retrieval from documents: A survey. *Information retrieval*, 2(2):141–163.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849.

Nasrin Mostafazadeh, Alyson Grealish, Nathanael Chambers, James Allen, and Lucy Vanderwende. 2016. Caters: Causal and temporal relation scheme for semantic annotation of event structures. In *Proceedings of the Fourth Workshop on Events*, pages 51–61.

TeongJoo Ong and John J Leggett. 2004. A genetic algorithm approach to interactive narrative generation. In *Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, pages 181–182.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. 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.

Rafael Pérez Ý Pérez and Mike Sharples. 2001. Mexica: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(2):119–139.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Julie Porteous and Marc Cavazza. 2009. Controlling narrative generation with planning trajectories: the role of constraints. In *Joint International Conference on Interactive Digital Storytelling*, pages 234–245. Springer.

Vladimir Iakovlevich Propp. 1968. *Morphology of the Folktale*, volume 9. University of Texas Press.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Mark O Riedl and Robert Michael Young. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39:217–268.

Judy Robertson and Judith Good. 2003. Using a collaborative virtual role-play environment to foster characterisation in stories. *Journal of Interactive Learning Research*, 14(1):5–29.

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

David E Rumelhart. 1975. Notes on a schema for stories. In *Representation and understanding*, pages 211–236. Elsevier.

James Ryan. 2017. Grimes' fairy tales: a 1960s story generator. In *International Conference on Interactive Digital Storytelling*, pages 89–103. Springer.

Roger C Schank and Robert P Abelson. 2013. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Perry W Thorndyke. 1977. Cognitive structures in comprehension and memory of narrative discourse. *Cognitive psychology*, 9(1):77–110.

Scott R Turner. 1993. *MINSTREL: a computer model of creativity and storytelling*. Ph.D. thesis, University of California, Los Angeles.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Xin Wang, Wenhu Chen, Yuan-Fang Wang, and William Yang Wang. 2018. No metrics are perfect: Adversarial reward learning for visual storytelling. *arXiv preprint arXiv:1804.09160*.

Kathleen A Welsch. 1998. History as complex storytelling.

Paul J Werbos. 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356.

Robert Wilensky. 1983. Story grammars versus story points. *Behavioral and Brain Sciences*, 6(4):579–591.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Jingjing Xu, Xuancheng Ren, Yi Zhang, Qi Zeng, Xiaoyan Cai, and Xu Sun. 2018. A skeleton-based model for promoting coherence among sentences in narrative story generation. *arXiv preprint arXiv:1808.06945*.

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

Meng-Hsuan Yu, Juntao Li, Danyang Liu, Dongyan Zhao, Rui Yan, Bo Tang, and Haisong Zhang. 2020. Draft and edit: Automatic storytelling through multi-pass hierarchical conditional variational autoencoder. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1741–1748.

Wenhao Yu, Chenguang Zhu, Tong Zhao, Zhichun Guo, and Meng Jiang. 2021. Sentence-permuted paragraph generation. *arXiv preprint arXiv:2104.07228*.

Fangzhou Zhai, Vera Demberg, Pavel Shkadzko, Wei Shi, and Asad Sayeed. 2019. A hybrid model for globally coherent story generation. In *Proceedings of the Second Workshop on Storytelling*, pages 34–45.

Yan Zhao, Lu Liu, Chunhua Liu, Ruoyao Yang, and Dong Yu. 2018. From plots to endings: A reinforced pointer generator for story ending generation. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 51–63. Springer.

Jichen Zhu and Santiago Ontañón. 2013. Shall i compare thee to another story?—an empirical study of analogy-based story generation. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2):216–227.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. 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.