Disentanglement of Syntactic Components for Text Generation

by

Utsav Tushar Das

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

Waterloo, Ontario, Canada, 2022

Author's Declaration

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

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

Abstract

Modelling human generated text, i.e., natural language data, is an important challenge in artificial intelligence. A good AI program should be able to understand and analyze natural language, and generate fluent and accurate responses. This standard is seen in applications of AI for natural language like machine translation, summarization, and dialog generation, all of which require the above ability. This work examines the application of deep neural networks for natural language generation. We explore how graph convolutional networks (GCNs) can be paired with recurrent neural networks (RNNs) for text generation.

GCNs have the advantage of being able to leverage the inherent graphical nature of text. Sentences can be expressed as dependency trees, and GCNs can incorporate this information to generate sentences in a syntax-aware manner. Modelling sentences with both dependency trees and word representations allows us to disentangle the syntactic components of sentences and generate sentences while fusing parts of speech from multiple sentences. Our methodology combines the sentence representations from an RNN with that of a GCN to allow a decoder to gain syntactic information while reconstructing a sentence. We explore different ways of separating the syntax components in a sentence and inspect how the generation operates.

We report BLEU and perplexity scores to evaluate how well the model incorporates the content based on its syntax from multiple sentences. We also observe, qualitatively, how the model generates fluent and coherent sentences while assimilating syntactic components from multiple sentences.

Acknowledgements

I would like to thank Dr. Olga Vechtomova for her constant guidance during my studies. Her insights and feedback were instrumental for the completion of this work. I am grateful to have had all the opportunities to explore new topics, pursue my studies in an interdisciplinary fashion, and work on my own pace to make the most of my time at the University of Waterloo.

I would like to thank Prof. Ming Li and Prof. Jesse Hoey for taking the time to review the thesis.

I have had the good fortune of meeting people from all over the world and forming great friendships during my time here at the University of Waterloo, with people showing up and engaging despite, as well as because of the pandemic, in true Warrior spirit.

In my own research group, I have to thank Egill Gudmundsson, Gaurav Sahu, Vikash Balasubramanian, Amirpasha Ghabussi, Dhruv Kumar, Kashif Khan, Zhidong Zhang, Brian Zimmerman, Olivier Poulin, Ruihan Wei, and Dongshu Luo for our many conversations and who made my studies here lively and fun.

I have to thank my friends and family, who were only a phone call away, for their constant support during my studies.

Finally, I would like to acknowledge the sublime natural beauty of the parks and trails of Waterloo, whose unique forms over the fall, winter, and summer during the pandemic I was grateful to witness, and were the constant elegant backdrop for my studies, this work, and my time here in Waterloo.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Artificial intelligence (AI) has been responsible for some of the most striking breakthroughs in all of science and technology in recent times. A relatively recent form of AI called deep learning has been the driving force behind these breakthroughs. Since 2012, when Alexnet achieved the best scores in the Imagenet Competition ([GE, 2019]), deep learning has been setting new benchmarks and achieving superhuman performance in various tasks in computer vision and natural language processing (NLP), among others. In 2016, deep learning captured the world's attention with Deepmind's remarkable victory at the complex game of Go ([Silver et al., 2016]), and later in December 2020, with AlphaFold ([Evans et al., 2021]), which made improvements to the extremely difficult problem of protein folding ([AlphaFold, 2020]). AlphaGo evolved to become AlphaZero ([Silver et al., 2018]), which became the best player in the world in Shogi, Chess, and Go. AlphaZero, in turn, has been refined into MuZero ([Schrittwieser et al., 2020]) which is also an expert on several visually complex Atari games. We now have self driving cars that can travel fully autonomously on highways ([Zon and Ditta, 2016]), conversational agents that can understand complex spoken information ([Hussain et al., 2019]), and AI-powered medical and legal assistants ([TMF, 2021], [Sobowale, 2016]).

In the domain of natural language, again in 2020, OpenAI released their third version of Generative Pretraining with Transformers, or GPT-3 ([Brown et al., 2020]). This model could write fluent paragraphs on both general and specialized topics and perform astonishingly well on a diverse set of natural language tasks. GPT-3 received plenty of media

attention for how close it seemed to a human as it wrote poems on Elon Musk, breaking news stories, and explained medical symptoms ([Piper, 2020]).

All of these successes are underpinned by a seemingly simple yet powerful mechanism: machine learning (ML), and, more specifically, deep learning. The paradigm of machine learning is based on learning the rules of making a prediction from data, rather than explicitly programming the rules. Machine learning algorithms extract patterns from data belonging to a certain distribution and use those patterns to make predictions on novel data from the same distribution. For example, an ML algorithm can be fed factors relevant to house prices in a city like square feet, location, number of rooms, etc. Using this data and the corresponding prices, the algorithm can predict house prices for a new set of factors. Deep learning improves this ability by scaling to significantly larger amounts of data and boosting its performance as the data increases ([Copeland, 2016]). It uses an architecture called neural networks to approximate a function, or fit a curve, to the data distribution. While this approach was computationally very expensive historically, since Alexnet in 2012, we have been able to leverage improvements in computer hardware, i.e., graphical processing units (GPUs), to make deep learning on large datasets feasible.

This work is in the sphere of applying deep neural networks to natural language data, or human-generated text. Modelling natural language is a unique and important challenge in AI due to the nature of the data. It is unstructured data, in that it does not conform to a pre-defined format and varies in the content and number of words. Natural Language Processing or NLP is the field of using computers to process and analyze text. Deep learning has proven to be an effective means of understanding and generating text data and has been implemented with great success to NLP tasks involving text generation like machine translation, dialogue generation, summarization, and question answering.

We use two architectures that have been used for NLP tasks; sequence models, or recurrent neural networks (RNNs), and graph convolutional networks (GCNs), for the task of sentence generation. We use the encoding and decoding abilities of RNNs with the capacity of GCNs to capture the syntactic information in sentences. We combine these two architectures to generate fluent sentences while incorporating informaton from more than one sentence.

1.2 Motivation and Problem Definition

Graph Convolutional Networks (GCNs) are a recent class of neural networks that have been used in the domain of natural language. First proposed in [Kipf and Welling, 2016],

they have the advantage of being able to leverage the graphical information inherent in natural language. This makes them an architecture worth considering alongside others in vogue during these times like Transformers ([Vaswani et al., 2017]).

Disentanglement is a current research problem in deep learning where the goal is to separate two or more latent spaces in the data so as to be able to use them for downstream tasks. This work explores disentangling the syntactic components of sentences. This lets us generate sentences while controlling its syntactic components and allows us to manipulate text at the level of granularity of syntax. Figure 1.1 shows an example of a sentence from the dataset we used in our experiments. Each word is labelled with its dependency grammar with the arrow going from the head word to the dependent word as well as whether it belongs to Subject-Verb-Object category of words, or SVO, or not, denoted by NonSVO.

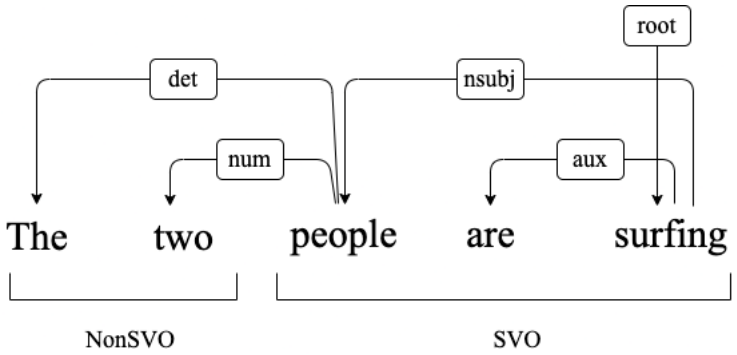

Figure 1.1: An example of splitting a sentence into SVO and NonSVO

1.3 Contributions

The contributions of this work are listed below:

- We explore different ways of separating syntactic components in sentences and how they can be used in natural language generation.
- We investigate how combining graph convolutional networks (GCNs) with RNNs impacts text generation.
- We conduct experiments on the sentence generation task and evaluate the generated sentences both with quantitative metrics and qualitatively.

1.4 Chapter Outline

The remainder of this thesis is organized as follows:

- Chapter 2 provides an overview of the deep learning methods and architectures used in this work
- Chapter 3 describes the proposed model and the training process
- Chapter 4 explains the different experiments performed and the results of the qualitative and quantitative evaluations.
- Chapter 5 gives a summary of the work, the conclusions we reach, and future directions of work.

Chapter 2

Background and Related Work

In this chapter, we first describe machine learning, and two of its widely used versions, supervised and unsupervised learning. We then focus on deep learning, and how it applies to NLP. To this end, we outline, recurrent neural networks (RNNs), and its two versions, Long Short Term Memory (LSTM) networks and Gated Recurrent Units (GRUs). Following this, we go over word embeddings, a foundational component of deep learning in NLP, and then sequence to sequence (Seq2Seq) models and autoencoders, which are used to learn representations and for text generation. We then review dependency trees and graph convolutional networks which form the second component of the proposed methodology for text generation. Finally, we briefly go over the task of sentence generation, with which we evaluate the proposed model.

2.1 Machine Learning

Machine learning (ML) has seen tremendous success by learning from example data. An ML algorithm goes over data points and learns a function to approximate the data. Using backpropagation ([Rumelhart et al., 1986]), and certain optimization techniques and activation functions, the algorithm starts with a random set of parameters for the objective function and eventually converges to a function that best estimates the data. The algorithm can be thought of as a function representing a curve and the data consisting of data points lying in some n-dimensional space. The algorithm wants to converge to a curve that manages to fit the data points as accurately as possible.

This technique now forms a significant part of our everyday lives. From the spam filter

in our email inboxes, to the voice assistants on our phones, and from the recommendations on our online shopping and social media apps, to the fraud prevention alert in our banking apps, ML is versatile and reliable enough to deliver effective predictions in a wide variety of contexts and scenarios. ML has successfully integrated itself into technology by being able to scale to billions and trillions of data points and being faster than humans in processing this volume of data, while being at times more accurate than humans at making complex predictions. We can describe ML in two ways as either using supervised learning or unsupervised learning. ML can also take the form of semi-supervised learning and reinforcement learning, but they will not be covered in this work.

2.1.1 Supervised Learning

In supervised learning, the algorithm, or model, is given the data points with the correct predictions that it is supposed to make from the data. These correct predictions are called labels or ground truth data. The algorithm starts off with a random set of parameters and makes a prediction. It then compares its predictions to the labels and makes adjustments to itself to better predict the label. It keeps repeating this process to improve its estimation of all the labels. Once it has been trained in this way for long enough, it can accept some new data and make a prediction based on the data it has seen so far.

For example, image classifier can be trained to recognize pictures with cats and those without. We would label all images with cats as 1 and those without as 0. The learning algorithm would be fed the pixel information of all the images and would then make a prediction. It would then compare its predictions with the labels and make adjustments to itself to be able to better predict which pictures have cats and which don't. With a good model architecture and the right amount of data, this method can lead to models that can make extremely accurate predictions, especially on highly specialized tasks.

2.1.2 Unsupervised Learning

In unsupervised learning, we give the model data as before but without giving it any explicitly annotated labels. The model learns the patterns in the data without making any prediction about a specific data point for a label. This method can be used for clustering algorithms to separate the data into different groups and assign new data points to those groups. This is useful in fraud and anomaly detection systems to see if incoming data points are outliers.

In some cases, even without labels, we can designate a part of the data as the label the model needs to predict. In NLP, for example, a model can be trained as a language model by teaching it to predict the next word after certain intervals in a dataset of documents. This method and others similar to it were applied with great success to build very powerful language models like the transformer ([Vaswani et al., 2017]), and its variants like BERT([Devlin et al., 2018]) and GPT([Brown et al., 2020]). In other NLP tasks like topic modelling, an unsupervised algorithm can learn which words occur in which documents, and group documents by the words, and therefore, by the topics that occur in those documents. Thus, a collection of documents can be divided into the topics it represents like technology, politics, medical, sports, entertainment, etc.

2.2 Deep Learning

One variety of machine learning involves the use of Artificial Neural Networks (ANNs). Inspired by the neurons in our brains, a neuron in an ANN is a function that processes some input passing through it. An ANN consists of multiple such functions or *nodes* connected to each other by *edges*, which form the weights for the operations in the nodes. The process of input data passing through multiple such neurons as part of a *network* is intended to evoke the idea of information signals passing through the neurons in the human brain. With the advent of powerful GPUs and the availability of massive amounts of data as a result of our increasingly digitized lives, architectures based on using neural networks have enjoyed great success and have become the most prominent and effective form of machine learning in the last decade. The paradigm of using neural networks has been termed as *deep learning* and model architectures using dense configurations of these neural networks are called deep neural networks [LeCun et al., 2015].

Multiple forms of neural networks exist and some are more suited to certain use cases and problems. For example, feed-forward neural networks can handle structured data like data from sensors, etc., while image data benefits from being processed with convolutional neural networks (CNNs) [Bengio and Lecun, 1997]. RNNs [Williams and Zipser, 1989], [Elman, 1990] are suited for sequential data like time series (stock ticker data), and text data.

2.2.1 Introduction to Neural Networks

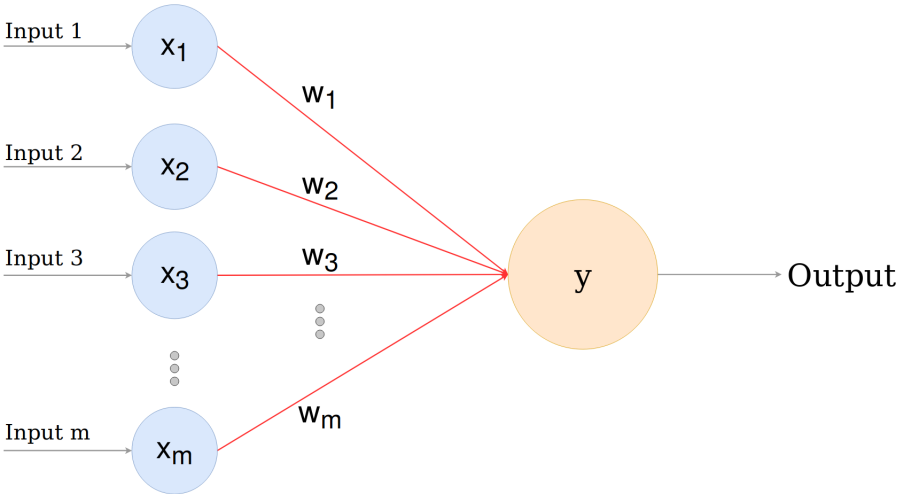

Figure 2.1: Perceptron from [Arc, 2018]

At the heart of all the extraordinary achievements of neural networks lies the perceptron [Rosenblatt, 1958], as shown in figure 2.1. In its simple form, it is a binary classifier that accepts multiple inputs $x_1, x_2, x_3...$ and gives an output y that either belongs to one of two categories that the output can belong to. Each input x_m has a weight w_m assigned to it; they are multiplied and added to a bias term b, in order to better fit the function curve lying in the data space the model is trying to learn. Depending on whether the result of this calculation falls above or below a certain threshold, the output is either class 0 or 1.

In modern neural networks, this binary classifier is replaced with a special type of mathematical function called an *activation function*, and the inputs are said to be fed to *neurons* rather than perceptrons. Activation functions like the Rectified Linear Unit (ReLU), sigmoid and tanh are the most popular functions used in deep learning. These functions are necessary to introduce non-linear transformations in the data, to capture complex functions like which subset of pixel values among all possible pixel permutations indicate cat pictures. The *deep* in deep learning refers to multiple such sets of neurons stacked in literal 'deep' layers, like in figure 2.2, to perform numerous computations on the inputs being fed to them. The multiple layers between the input and output layers are referred to as *hidden* layers.

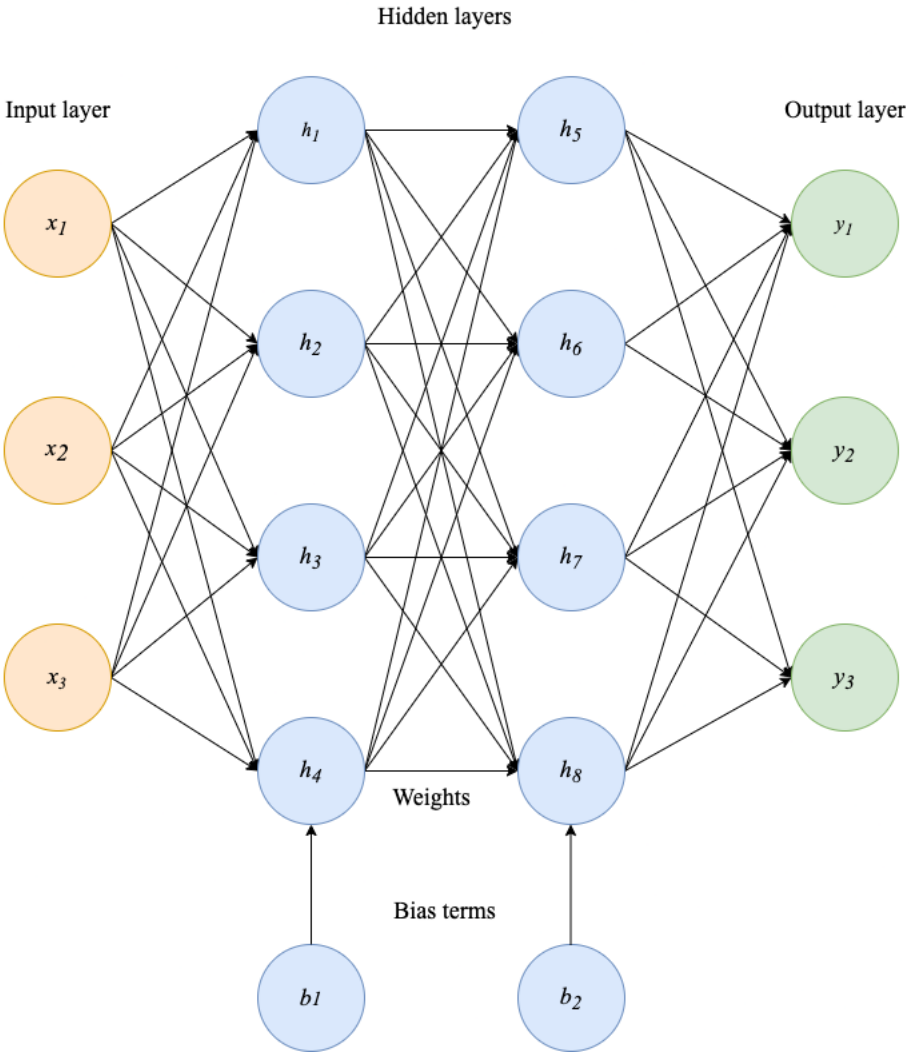

Figure 2.2: Feed-forward neural network

Initially, the weights for the inputs are randomly assigned, and thus the prediction of the network will be distant from the correct prediction, or ground truth. The weights are then adjusted, which forms the *learning* part of the algorithm. Knowing how much to adjust the weights is crucial to the ability of deep neural nets to fit a high-dimensional function to data points in an even higher-dimensional space. This 'distance' between the prediction and the target is determined by something called a *loss function* like mean-squared error or cross-entropy depending on the task. The loss function is also called the

objective function or the cost function. The adjustment to the weights happens through a process called *gradient descent*.

A deep neural net can be thought of as attempting to *model* a *curve*. This curve will lie in the high-dimensional space that encompasses all possible data points. The data in the training set, or the ground truth is lying somewhere in this space. The goal of the neural net is to learn a curve that fits this data. Hence, a deep model is often said to be trying to learn a *latent* representation of the data presented to it. Gradient descent takes the derivative of this function representing the curve to get its slope. This indicates whether the slope is approaching the global optimum, say the global minimum in this case, since we are trying to *reduce* the distance between the ground truth and the prediction. This slope, or *gradient*, then lets us perform backpropagation [Rumelhart et al., 1986], to use the *error* in the prediction to update the weights of the network or model.

More concretely, the weight update can be described as:

$$w_{n+1} = w_n - \alpha \nabla f(w_n) \tag{2.1}$$

where w_{n+1} is the updated weight and w_n is the current weight. α denotes the *learning rate* and indicates how big the adjustments to the weights or *steps* should be along the gradient.

With every data point that the model uses its weights to make a prediction on, gradient descent with backpropagation updates the weights in order to decrease the score from the loss function, i.e., the error. This process repeats for the entirety of the dataset, and multiple passes over the full dataset, called *training epochs*, until it makes as accurate predictions as it can; the loss value cannot be reduced any more. We then say that our network or model is *trained*.

2.2.2 Recurrent Neural Networks

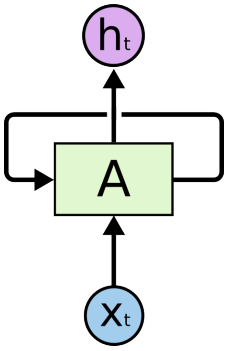

Figure 2.3: Recurrent Neural Network from [Olah, 2015]

When it comes to natural language processing, the data have dependencies with each other. Words are part of a sequence, with the sequence being a sentence or a document. Feed-forward neural networks however, are designed in such a way that all the data are processed independently of each other. In natural language, the occurrence of a word in a sentence is contingent on the word that appeared previous to it. In order to build systems that can process this information, it is important that model architectures are able to preserve information that appeared previously in a sequence to inform the computation and decision making for the current data point. RNNs are suited for this case because they contain loops that carry information from previous tokens in a sequence to the next, as shown in figure 2.3. Each loop deals with a token in the sequence while carrying over information from the previous loops.

Tokens in a sequence, such as words in a sentence, or characters of words in a sentence, are fed in sequentially, one by one, or in each *timestep* to the RNN. As shown in figure 2.4, the input to the network is denoted by x_t at timestep t. The RNN computes a hidden representation for the input h_t, which is sent as input to the step $t+1$. A hidden weight also exists for the previous timestep $t-1$. This h_{t-1} also goes as an input. The RNN uses these inputs to compute the output o_t. y_t is the target at step t, L^t is the loss, while V, W and U are the learned weight matrices for the output, hidden, and input states respectively.

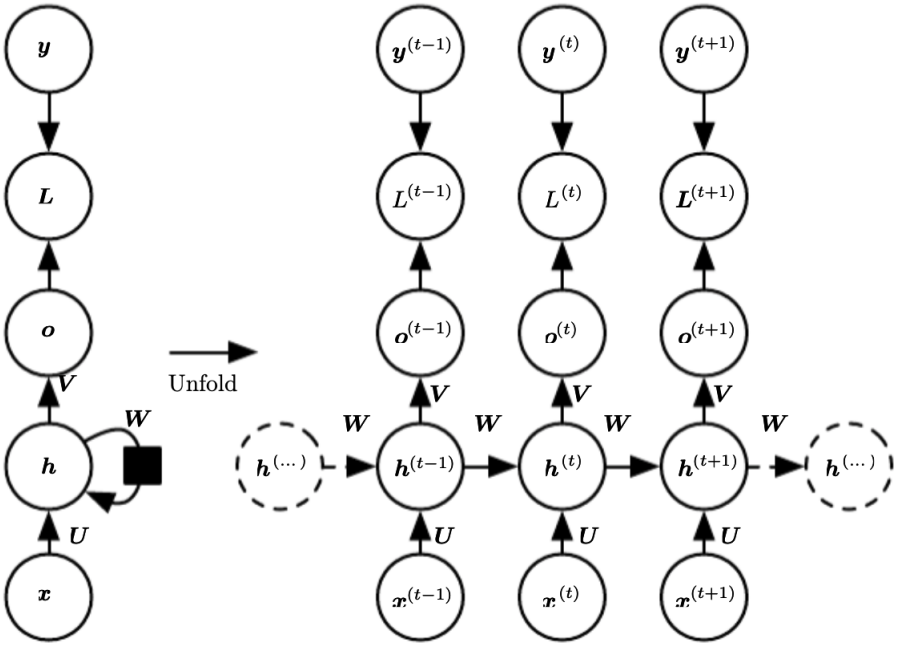

Figure 2.4: Unfolded RNN from [Goodfellow et al., 2016]

2.2.3 Long Short Term Memory

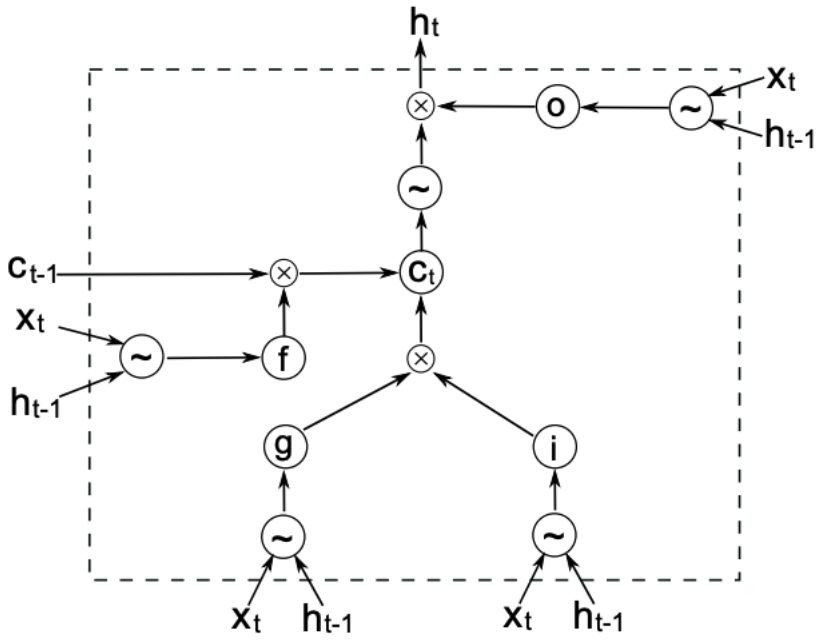

Figure 2.5: LSTM from [Xu et al., 2015]

While RNNs facilitate taking information from the previous token to the next one, the basic version or vanilla forms of the RNN are not suited for capturing very long-term dependencies. For example, in a reading comprehension task, the information in a sentence near the beginning of a paragraph may be needed to infer or understand information a considerable number of words later near the end of the paragraph. This is because RNNs, just like other neural networks, are trained with backpropagation, and suffer from the vanishing and exploding gradient problems [Pascanu et al., 2012]. In the case of RNNs, backpropagating a signal over a long distance in a sequence causes the gradient values to become too small, leading to numerical underflow [Pascanu et al., 2013].

While exploding gradients can be solved by techniques like gradient clipping ([Pascanu et al., 2012]), some modifications have been proposed to vanilla RNNs to improve their ability to retain information over longer distances in a sequence. The two

widely used architectures in this instance are Long-Short Term Memory Networks (LSTMs) [Williams and Zipser, 1989] and Gated Recurrent Units (GRUs) [Cho et al., 2014].

In LSTMs, in addition to a hidden state. there is a cell state at time step t, c_t, to retain information over longer distances. In the same way as an RNN, there is an input x_t and the previous hidden state h_{t-1}. The previous cell state c_{t-1} learns the long-term information needed and sends it to the next step. How much information is to be carried by the cell state is determined by three *gates*, the input gate i_t, the forget gate f_t and the output gate o_t

The equations of the gates are:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \tag{2.2}$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \tag{2.3}$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{2.4}$$

σ is the sigmoid activation function. A candidate cell state $\tilde{C}_t$ is computed to determine whether to replace the previous cell state should be replaced or carried forward to the next time step.

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.5}$$

The cell state for the current time step can then be computed as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.6}$$

The forget gate can thus be seen as learning how much of the information from previous time steps to be carried forward with C_{t-1} and the input gate i_t determines how much of the current word should be incorporated into the cell state.

$$h_t = o_t * tanh(C_t) \tag{2.7}$$

The final latent representation at that time step is then computed with the output gate as above.

Bidirectional LSTMs

In some cases, this architecture can be made more powerful by allowing it to look 'ahead' in time and use information from further ahead in a sequence. In the description above, the LSTM is able to 'see' and retain information from the start of a sequence and then further up at each timestep. In the bidirectional case, another LSTM takes in information from the reverse direction from the end of a sequence. This LSTM is stacked with the first 'unidirectional' one and both outputs are concatenated at the end.

2.2.4 Gated Recurrent Units

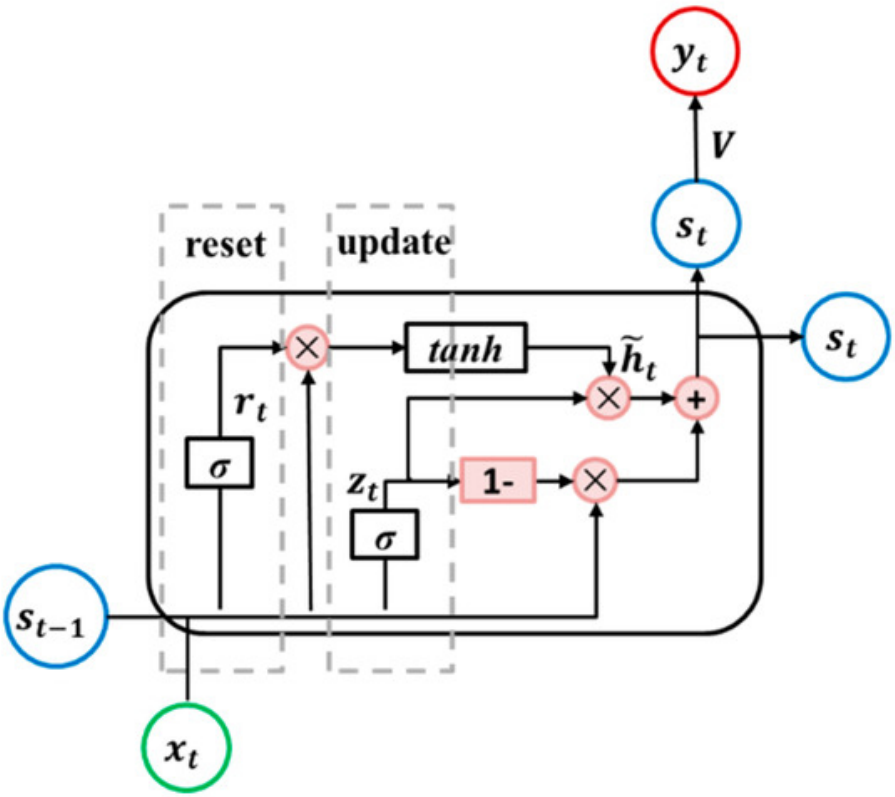

Figure 2.6: GRU from [Zhao et al., 2019]

In GRUs, there are two gates z_t and r_t that take over the function of the gates in an LSTM. This architecture combines the input and forget gates and outputs only a single hidden

state instead of both h_t and the cell state c_t. The r_t gate is used to compute the candidate hidden state $\tilde{h_t}$ at time step t and z_t determines how much of the current hidden state to incorporate in the final output hidden.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{2.8}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2.9}$$

$$\tilde{h_t} = tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{2.10}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t} \tag{2.11}$$

We use GRUs in our experiments for simplicity as there are fewer hidden states to deal with.

2.2.5 Word Embeddings

Machines have no conception of words and language as we know it. We have to represent words as numbers; indices in a dictionary or vocabulary. Each word in the vocabulary, could, for e.g., be represented as a one-hot vector. The size of the vector would be the size of the vocabulary and it would be populated by zeros except at the position where the word occurs in the vocabulary, which would be a 1. This leads to one-hot vectors being very *sparse* representations of words as most of the values are 0. The vectors are also very high dimensional, as they are of the same size as the vocabulary, and this size can be values like 20000, depending on the task and the dataset. This makes one-hot vectors less efficient and thus more compute-intensive representations.

More fundamentally, choosing a certain way to represent words means making an assumption about how the input features to the model are structured in the feature space. With one-hot vectors, the assumption becomes that all the features, i.e., the words, are independent of each other in the space. This is because each one-hot vector of a word is *orthogonal* to all the other ones. However, as we know from our everyday usage of language, that is clearly not the case. Words are very much related to each other, and thus should ideally be represented in a similar way in a feature space. For example, 'house', 'apartment', 'building', 'condo', 'home' are all closely related and can be swapped with

each other in a lot of contexts. Words are semantically related and should be represented as such.

This is where word embeddings come in. They capture the similarities between semantically related words through the distance between them in the vector space, as measured through a metric like cosine similarity. In addition to that, they are also much more dense, lower dimensional vectors than one-hot encodings. In modern neural nets, word embedding sizes commonly used are 256, 300, 768, etc as opposed to in the tens of thousands in the case of one-hot vectors.

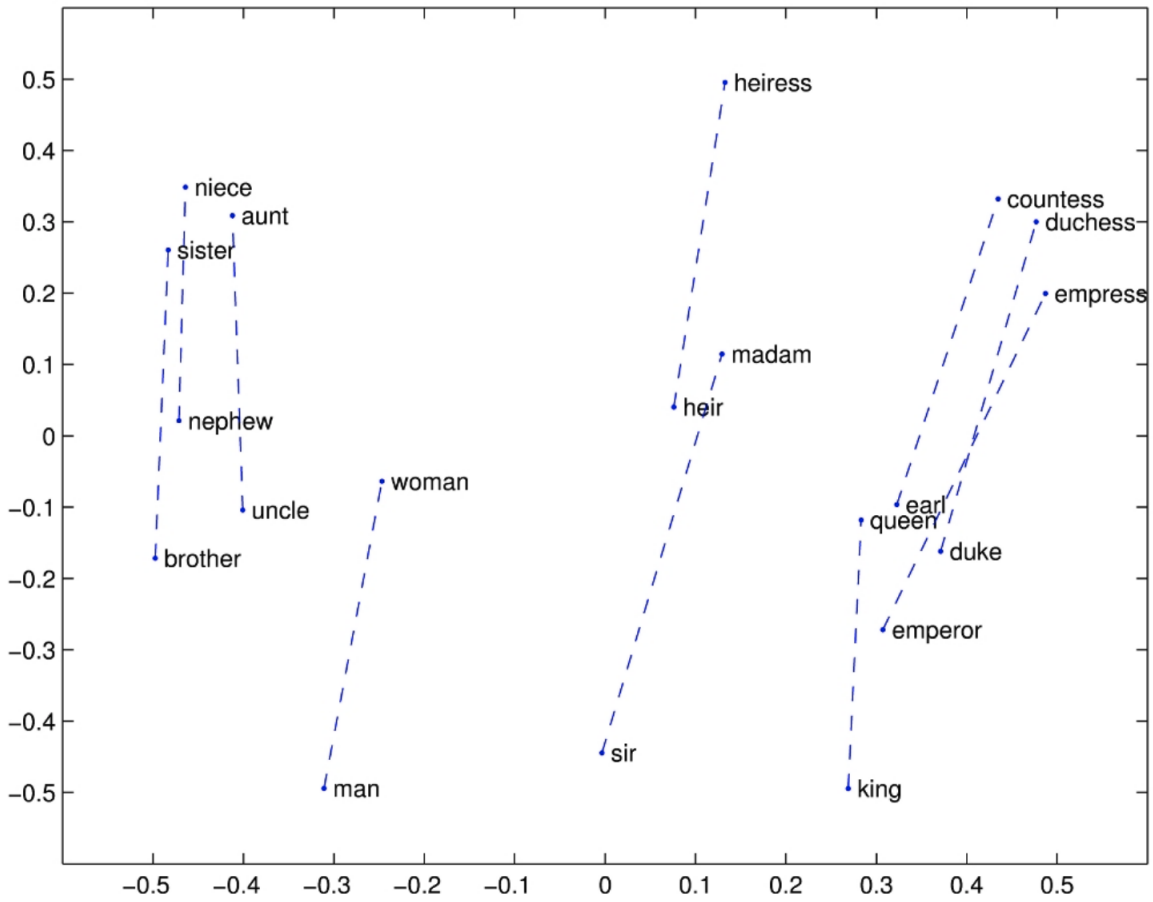

Figure 2.7: 2D projection of 300 dimensional GloVe word embeddings from [Caliskan et al., 2017]

Not only are word embeddings dense, they are also *structured*, which imparts very useful properties to them. The structure enables semantically related word embeddings to be placed close to each other in the embedding space. For example, in figure 2.7, we can see the embedding vectors for several words placed in a 2D space. We can see the words 'niece', 'aunt' and 'sister' are close together whereas 'nephew', 'uncle', and 'brother' are grouped closer to each other. Additionally, the distance between 'sister' and 'brother' is comparable to 'aunt' and 'uncle' and so on. Figure 2.7 shows words used to identify female and male people in different contexts. The distance between each female term and the corresponding male term is similar between different types of words, like 'woman' and 'man' and 'queen' and 'king'. Subtracting the vector from, say, 'queen', to get to 'king' can then be thought of as the *gender* vector, and representing the gender attribute or dimension of words. Correspondingly, other dimensions representing 'color', 'occupation', 'location', etc are captured in the embedding vectors of the words. In practice, vector sizes like 300 in the case of Word2Vec have empirically been seen to capture a good number of useful attributes of words. Thus, 300 numbers for the word 'apple' can be seen as containing the necessary attributes like its color, that it's a food and a fruit and so on.

Pre-trained word embeddings

Word embeddings can be either learned jointly with the task that the model is being trained for or using a separate task and then included in the model. In the first case, the embedding vectors are initialized randomly, just like the model weights, and learned along with the model as it trains. In the latter, a model is trained on a large dataset of text, also called a corpus, with a task that enables it to learn good embeddings for the words in the vocabulary for the text. For example, one task that is used is to predict the words within a certain window around the word whose embedding we are trying to learn.

The idea behind pre-training, is that training a big model on a large text dataset helps it learn a lot of generic useful properties of the language. Training the model on a general simple task gives it a lot of useful information that is applicable across a number of other NLP tasks, because the knowledge it has gained is transferable. This idea has been validated through strong empirical results. For example, the idea behind BERT is that we learn a general purpose language model and then fine-tune it on specific NLP tasks like translation, question answering, sentiment prediction etc. Pre-training using the BERT architecture has been shown to be very effective for this.

The idea of learning these word embeddings was first proposed in [Bengio et al., 2000], but just as with the rest of deep learning, the method received widespread attention after improvements in hardware and availability of data. The first widely used success-

ful implementation was in Word2Vec [Mikolov et al., 2013a], [Mikolov et al., 2013b], followed rapidly by GloVe [Pennington et al., 2014], which stands for Global Vectors. In [Mikolov et al., 2013a], they train embeddings in two ways as shown in figure 2.8. Both of these ways are simple, yet highly effective techniques to learn good embeddings. The first is the Continuous Bag of Words or CBOW approach. In CBOW, the word index for the word we are trying to learn the embedding for is hidden from the model, and the model has to predict it using the words around it within a certain range. The surrounding words are also called context words. In the second way, called skip-gram, the opposite strategy is used: given the target word, the model has to predict the context words in a certain window span. For example, in the sentence *'I want a glass of orange juice'*, let's say we are trying to learn the word vector for 'glass'. In CBOW, the model would try to predict the word 'glass', while in skip-gram, it would try to predict the context words like 'orange' and 'juice'. After training for several iterations, the weight matrix that the model would update would form the embedding vectors for the word. For our work, we use pre-trained Word2Vec embeddings trained with the skip-gram method with a window size of 5.

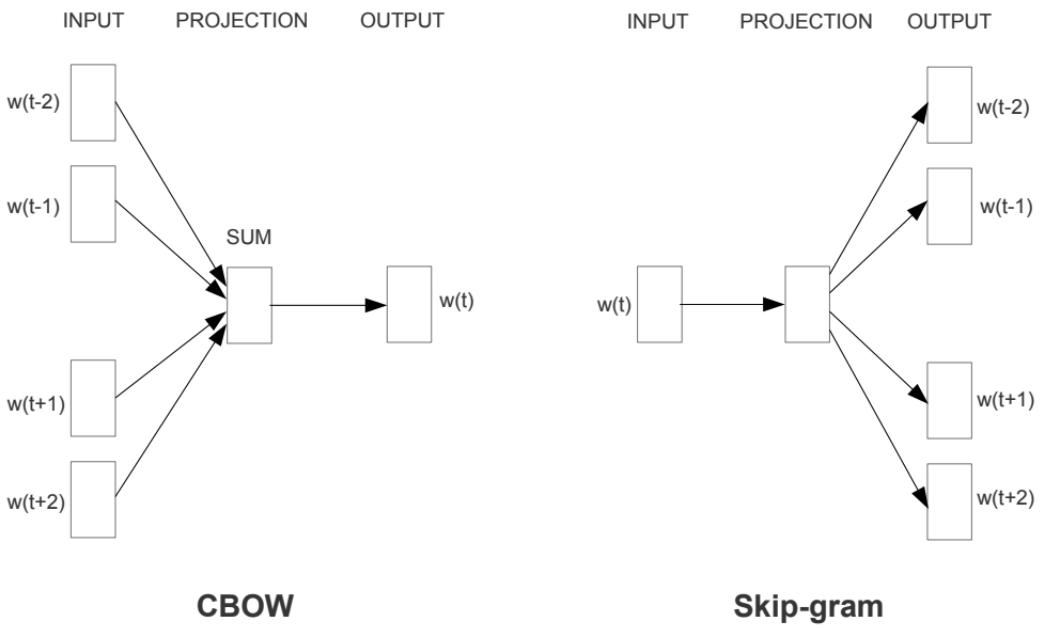

Figure 2.8: The CBOW and skipgram models from [Mikolov et al., 2013a]

2.2.6 Sequence to Sequence Models

In text generation scenarios like dialogue generation in conversational agents, machine comprehension, question answering, and machine translation, it's necessary to *encode* a sequence of words in one RNN and then using a *decoder*, generate the desired output sentence, which will be of a different length than the input sentence.

The encoder processes the input sentence with an architecture like the LSTM or GRU and emits a hidden state vector at the final time step. This vector can be thought of as the representation of the input capturing the most salient information for the entire input sentence. It is often called the context C as shown in the figure below. The context vector is then fed to the decoder RNN which generates the output sentence, token by token, at each time step.

This approach was first featured in [Sutskever et al., 2014], which showed the benefits of this architecture in generating sequences with sequences as input without having to worry about the nature of the sequence. Since then, they have formed the basis of several works instrumental in the development of NLP for a wide variety of generation tasks like in transformers [Vaswani et al., 2017]. Since our work involves the generation of sentences, the architecture of our proposed approach uses this type of *Seq2Seq* model.

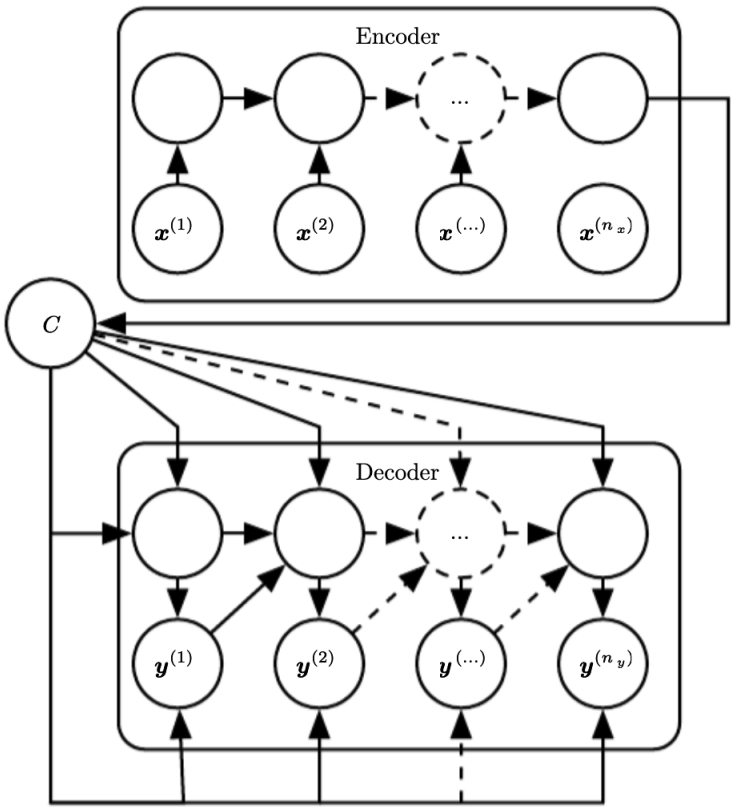

Figure 2.9: Seq2Seq model from [Goodfellow et al., 2016]

2.2.7 Auto-encoders

In a Seq2Seq model, instead of using the decoder to generate a different output sequence like a translated sentence in another language, or a response sentence to a question asked of a chatbot, we can instead use the hidden context vector to reconstruct the provided input, as accurately as possible. The architecture then emphasises learning a good *latent* or *intermediate* representation that captures the most important information to reproduce the input. This hidden vector is of a lower dimensionality than the input representation and is thus a more efficient way of encoding the input and using it for further downstream applications.

For example, as shown in the figure below, let's say that the input is an image of a

handwritten digit, 4. We encode this using a neural network into a latent representation z, and then attempt to reconstruct the input representation of the image using a decoder. After training the model to do this well enough, the learned z can then be used for an application like learning a model to classify handwritten images of digits. This learning method is called auto-encoding

This same approach applies to text data in NLP, and the neural networks used are usually RNNs. Autoencoders can be used to learn how to reconstruct sentences, or paragraphs [Li et al., 2015a] and then these useful representations can be used in tasks like sentiment analysis and recommender systems [Li et al., 2015b], [Hewlett et al., 2017].

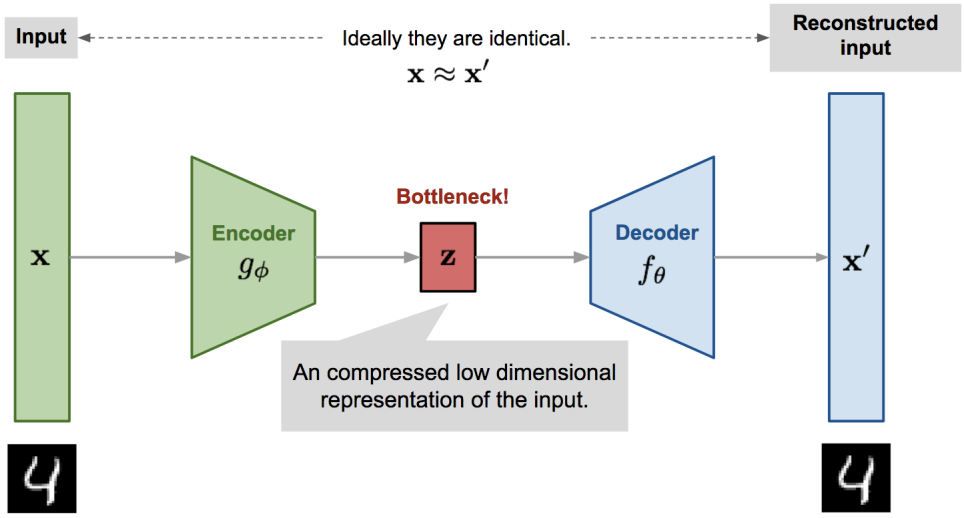

Figure 2.10: Autoencoder from [Weng, 2018]

2.2.8 Dependency Trees

A sentence can be modelled as a set of directed binary grammatical relations between the words in it. In the sentence in the figure 2.11, each word can be thought of as either the *head* or the *dependent* in the relationship. The arrows indicate the dependency grammar of the relationship going from the heads to the dependents. One of the arrows does not connect to any of the words and indicates that the word in the sentence, in this case 'prefer', is the *root* word and forms the root of the dependency tree for this sentence.

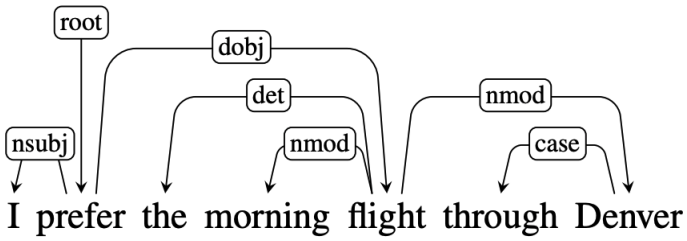

Figure 2.11: A dependency tree from [Jurafsky and Martin, 2009]

These head-dependent relations also have a useful characteristic of approximating the semantic information of the words and thus can be used in downstream NLP tasks [Jurafsky and Martin, 2009]. As shown in the figure, the relations between the words are captured using Universal Dependency (UD) relations [Jurafsky and Martin, 2009]. For example, the subject and direct object of the VERB word *prefer* are *I* and *flight* respectively. Dependency trees, expressed through adjacency matrices, are necessary to feed as the input to Graph Convolutional Networks (GCNs), as we will see below. This helps us assimilate the syntactic knowledge of the dependency trees into the representation learned from the encoder, and provide this combined information to the decoder to generate a sentence.

2.2.9 Graph Convolutional Networks

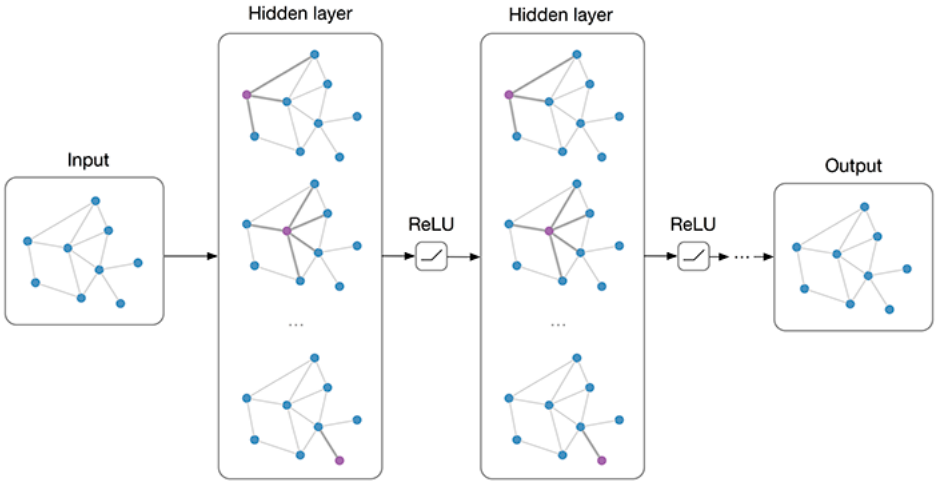

Figure 2.12: A multilayer GCN from [Kipf, 2016]

GCNs are a variant of convolutional neural networks (CNNs) [Lecun et al., 1998] first proposed in [Kipf and Welling, 2016]. They operate on graph data by using the features in nodes of some data represented as a graph. As shown in 2.12, the nodes in the graph would be represented as an adjacency matrix, taking the place of a feature matrix for a feed-forward net.

Let's say we have a graph $G = (V, E)$, where $\mathcal{V}(|V| = n)$ is the set of nodes and $\mathcal{E}$ denotes the set of edges. We can then define a matrix $X \epsilon \mathbb{R}^{m \times n}$ containing the features of the nodes, with each column $x_v \epsilon \mathbb{R}^m \, (v \epsilon \mathcal{V})$ encoding the features of the node. The representation of a node is then computed as:

$$h_v = ReLU\left(\sum_{u \epsilon \mathcal{N}(v)} (W x_u + b)\right) \tag{2.12}$$

where $W \epsilon \mathbb{R}^{m \times m}$ and $b \epsilon \mathbb{R}^m$ are the weight matrix and bias. $\mathcal{N}(\nu)$ are neighbors of ν and ReLU is the Rectified Linear Unit. By stacking more GCNs like in RNNs, it's possible to gather information from further nodes in the graph.

$$h_v^{(k+1)} = ReLU\left(\sum_{u \epsilon \mathcal{N}(v)} W^{(k)} h_u^{(k)} + b^{(k)}\right) \tag{2.13}$$

k is the layer number and $h_v^{(1)} = x_v$

Complementarity of RNNs and GCNs

As explained in [Marcheggiani and Titov, 2017], RNNs like GRUs and LSTMs are very good at capturing long-term dependencies for different NLP tasks. Instead of feeding a GCN word embeddings as we would in a neural network, we feed them to an RNN and then give the RNN embeddings, such as the final state of an LSTM, as input. While LSTMs capture syntactic information too without explicitly modelling it, GCNs explicitly enable them to integrate this information and reduce the number of steps over which an LSTM has to retain information, by taking in the syntactic information of the neighbouring nodes/words. This makes LSTMs and GCNs complementary to each other and improves the ability of a model on tasks like text generation, which we will look at in more detail in the next section.

2.2.10 Natural Language Generation

Natural Language Generation or NLG is a branch of NLP that deals with the generation of text that attempts to emulate the way humans write and speak. This is an important research area within NLP because in addition to understanding and processing text, a powerful and ideal natural language system should be able to generate text that is fluent, coherent, diverse, and which humans can engage with in a meaningful and complete way. Before the era of deep generative learning, NLG attempts were based on pre-defined grammar rules [Reiter and Dale, 2000] or statistical approaches [Cambria and White, 2014].

In the current stage of NLG, deep learning has led to meaningful progress in several tasks:

Neural Machine Translation

Since notable deep learning attempts on machine translation using different architectures [Sutskever et al., 2014], [Cho et al., 2014], [Bahdanau et al., 2016], Neural Machine Translation or NMT has become the most effective method of automated translation and powers

widely used tools like Google Translate. NMT is defined where a target text has to be generated with the same content, i.e., with the same semantic composition as the source sentence, and the target and source are different languages, such as Hindi and English. Transformer-based architectures are currently the best NMT methods for the English to German and English to French benchmarks, which are two standard evaluation datasets [Ruder, 2022].

Stylized Text Generation

A sentence or text can be thought as possessing some *content* information and *style* information, i.e., some content written in a particular style. Given a source text and target text with differing content and styles, it is then possible to generate a text with the content of the source and style of the target. This has applications in generating a document in a certain author's style or a style of writing adhering to that of a scientific or news publication. Recent progress in this field has leveraged Variational Autoencoders (VAEs) [Kingma and Welling, 2014], [Rezende et al., 2014], adversarial learning, RL, edit-based techniques and a host of different architectures specific to the task [Mou and Vechtomova, 2020].

Dialogue Response Generation

This is an important research problem to develop automated systems that can hold conversations and respond to queries from humans on a wide range of topics. Applications of this field include travel assistants, general purpose voice assistants like Siri and Alexa as well as specialized medical and legal assistants. In this task, the text generated needs to be conditioned on the previous text or 'utterance' in the conversation. Thus, the target is conditioned on the source and this continues throughout the entire conversation history. Using a mix of approaches containing attention [Bahdanau et al., 2016], RNNs, and other task specific modifications have set the current state-of-the-art in dialogue generation [Ruder, 2022].

Text Summarization

This task involves producing a fluent, coherent, relevant, and accurate summary of an original longer document. The subset of the task that uses a generative model to truly generate the summary is called abstractive summarization. Reinforcement learning techniques, graph-based approaches, and architectures using Generative Adversarial Networks

(GANs) [Goodfellow et al., 2014] have been successful for popular benchmarks for this task, like the CNN / Daily Mail dataset [Ruder, 2022].

This work focuses more on the importance of syntax in NLG, specifically for sentence generation, such as that seen in dialogue response generation and style transfer. Along the lines of this work, efforts have been made to explicitly incorporate syntax information in deep learning models for different purposes. [He et al., 2020] have explicitly leveraged syntax to enhance the generalization of deep models like BERT. [Bai et al., 2021] incorporate syntax trees into checkpoints in models based on the transformer architecture and demonstrate improvement on multiple such models. [Kuncoro et al., 2019] have used knowledge distillation to transfer knowledge from a syntactic language model to an LSTM to enable it to generate more structurally coherent sentences. In [Hu et al., 2021], syntax is used to improve performance on a downstream application task of style transfer. For further information, [Zhao et al., 2021] have surveyed the progress in both modelling syntax as an end goal in itself and using it in different tasks.

Chapter 3

Approach

3.1 Model architectures

3.1.1 Deterministic Autoencoder (DAE)

For our approach, we start things off by implementing a baseline we can use to compare our later models with. In this case, we have a deterministic autoencoder (DAE) to reconstruct the input sentences. The function of the DAE is exactly like that as described in section 2.2.7 and its architecture is as described in section 2.2.6. The model consists of an encoder GRU to form a representation of the sentences and then a decoder GRU to reconstruct the sentences.

The input are the word embeddings of the sentences acquired from Word2Vec. The embedding size is 300. A batch size of 250 is used to process 250 sentences at a time. The input is thus a *(batch-size, sequence-length, embedding-size)* tensor. Tensors are essentially a generalization of matrices to any number of dimensions and are the data types used by deep learning frameworks like PyTorch [Paszke et al., 2019] to compute data. The reader is referred to [Goodfellow et al., 2016] or [Chollet, 2017] for further reference on data representations for deep learning. The encoder is a bidirectional 2-layer GRU. Bidirectionality helps the encoder learn as good representations as it can for the decoder. We found two layers gave the model enough capacity to encode the input well. The hidden or latent size is set to 200, so we get a *(batch-size, sequence-length, 2 * hidden size)* hidden state. This is then compressed to a size of *(batch-size, 1, embedding-size)* with a fully-connected dense network. The idea is to have a single vector that represents the entire sentence and that would form the latent information that would be carried forward to the decoder.

This hidden vector for each sentence for each batch is appended to the original embedding of all the words in the sentence. This concatenated representation of the first word in the sequence, which is a 'start of sentence' or '$< SOS >$' token, is fed as the first input to the decoder. The decoder is a single layer unidirectional GRU. We want the GRU to learn how to reconstruct the input without making it too easy for it. The GRU decoder makes a prediction from this input picking the highest probability word from a softmax layer. Once the loss is calculated, we replace the prediction with the ground truth target to train the model with *teacher forcing*. The embedding vector for this token undergoes the same appending operation as above and is fed to the decoder to generate a token at each time step.

3.1.2 DAE with GCN

In this model, we sandwich a GCN between the encoder and decoder to incorporate syntactic information in the learning pipeline. The goal of the model is reconstruction as before, but the hidden representation contains information from the GCN. The encoder output is fed to the GCN with the adjacency matrix of the sentence. The output of the GCN is of the same dimensions as the encoder output, but having undergone the operations described in section 2.2.9. As with the DAE, this output is then fed to the decoder at the first time step to start the reconstruction. Figure 3.1 shows how the input is processed in the model.

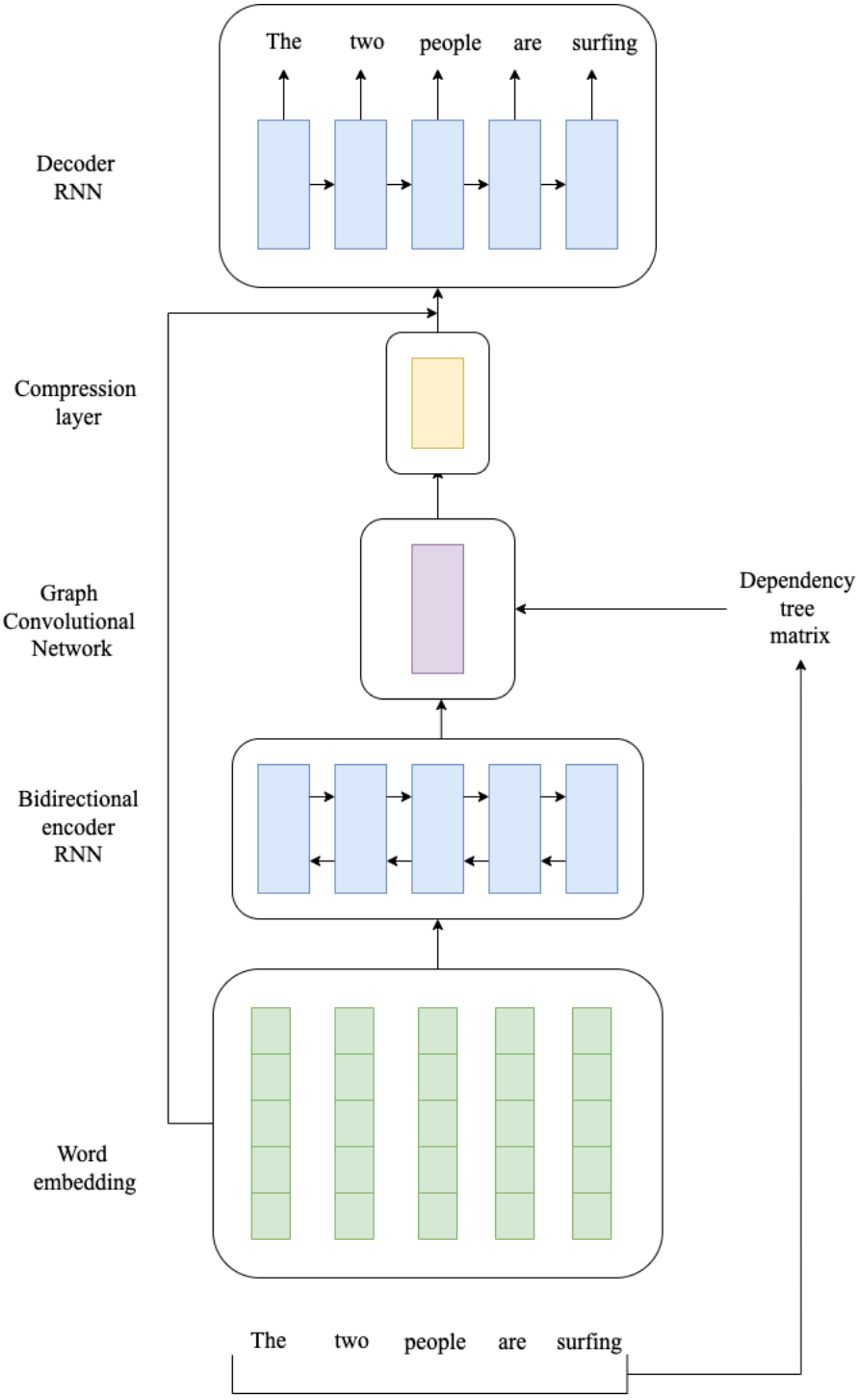

Figure 3.1: Reconstructing a sentence with GCN

3.1.3 Dependency graph construction

In the subsequent models, we split the syntactic components of a sentence and feed them into separate GCNs to capture the information from the separated sets independently. We try two different sets of parts of speech to perform the splitting. In each way, or configuration, we split the words in the sentence into two sets. The first set contains words belonging to either the *subject*, *object*, or *verb* parts of speech, and the second set contains words with all other parts of speech and dependency relations like conjunctions, auxiliaries, adpositions, etc.

Configuration 1

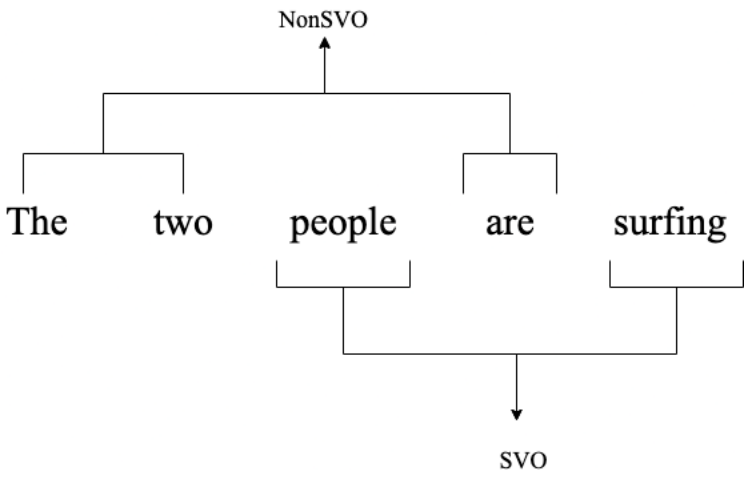

Figure 3.2: An example of SVO and NonSVO split using configuration 2

In the first way, words that have *nsubj*, *dobj*, and VERB or ROOT are included as SVO with the remaining tokens in the sentence as NonSVO. Figure 3.2 shows an example of producing the splits in this way with the dependency relations of the words.

Configuration 2

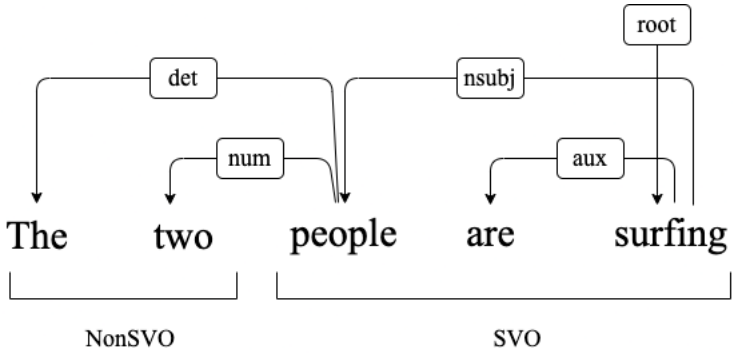

Figure 3.3: An example of SVO and NonSVO split using configuration 1

In the second way, we split the sentence with the following as SVO:

1. Dependents of NOUN that has *nsubj* relation with VERB.
2. AUX verb that has VERB as head.
3. Dependents of NOUN that has *dobj* relation with VERB.

The remaining tokens are considered part of NonSVO. Figure 3.3 shows an example of producing the splits from a sentence using the above rules.

In the experiments, we will refer to these two ways of splitting the components, as described above, as *configuration 1* and *configuration 2*. The words belonging to the SVO set are termed *split 1* and the ones in the NonSVO set are termed *split 2*.

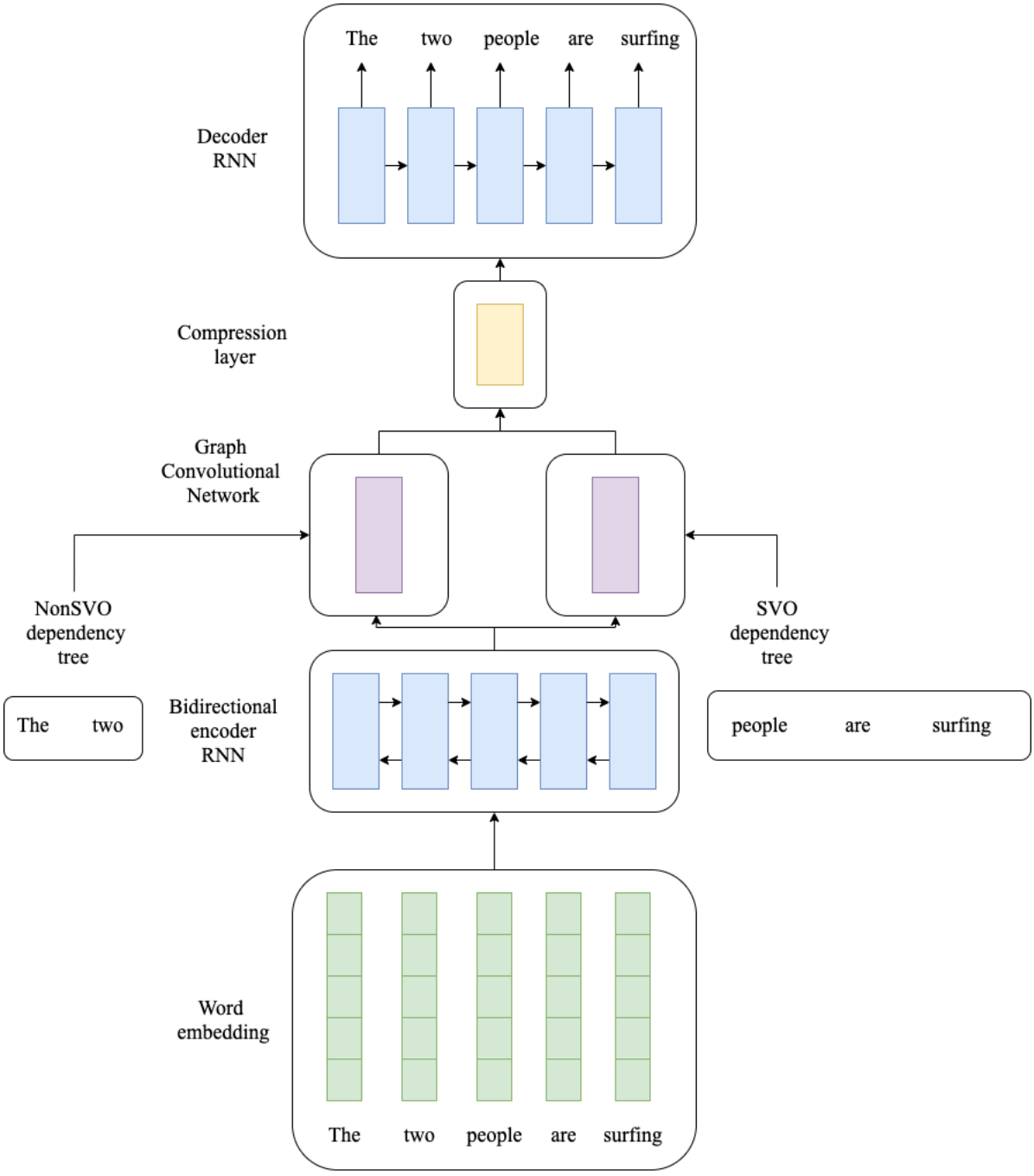

Figure 3.4: Splitting and reconstructing a sentence

3.1.4 Splitting a sentence and reconstructing it with a GCN

Next, to teach the model to construct sentences with different syntactic components, a sentence is split into its SVO and NonSVO components using either configuration 1 or 2 before feeding it to the decoder. The word embeddings of the sentence are encoded as before. However, there are two different GCNs to accept the dependency trees corresponding the SVO and NonSVO words. The representations from the two GCNs are concatenated and then compressed as before. Figure 3.4 shows this model. We will call this model *GCN-split* in the experiments.

3.1.5 Splitting and combining syntactic information from multiple sentences

Finally, we split two sentences into their SVO and NonSVO components and examine the generation when we mix components from different sentences. Specifically, we take the SVO parts from the first sentence and the NonSVO parts from the second sentence and feed them to two different GCNs as in the model before. Their output is concatenated and the generation thus merges syntax information from both sentences. The process is repeated for the SVO parts from the second sentence and the NonSVO parts from the first sentence. Figure 3.5 shows this model. We call this model *GCN-multi-split* in the experiments.

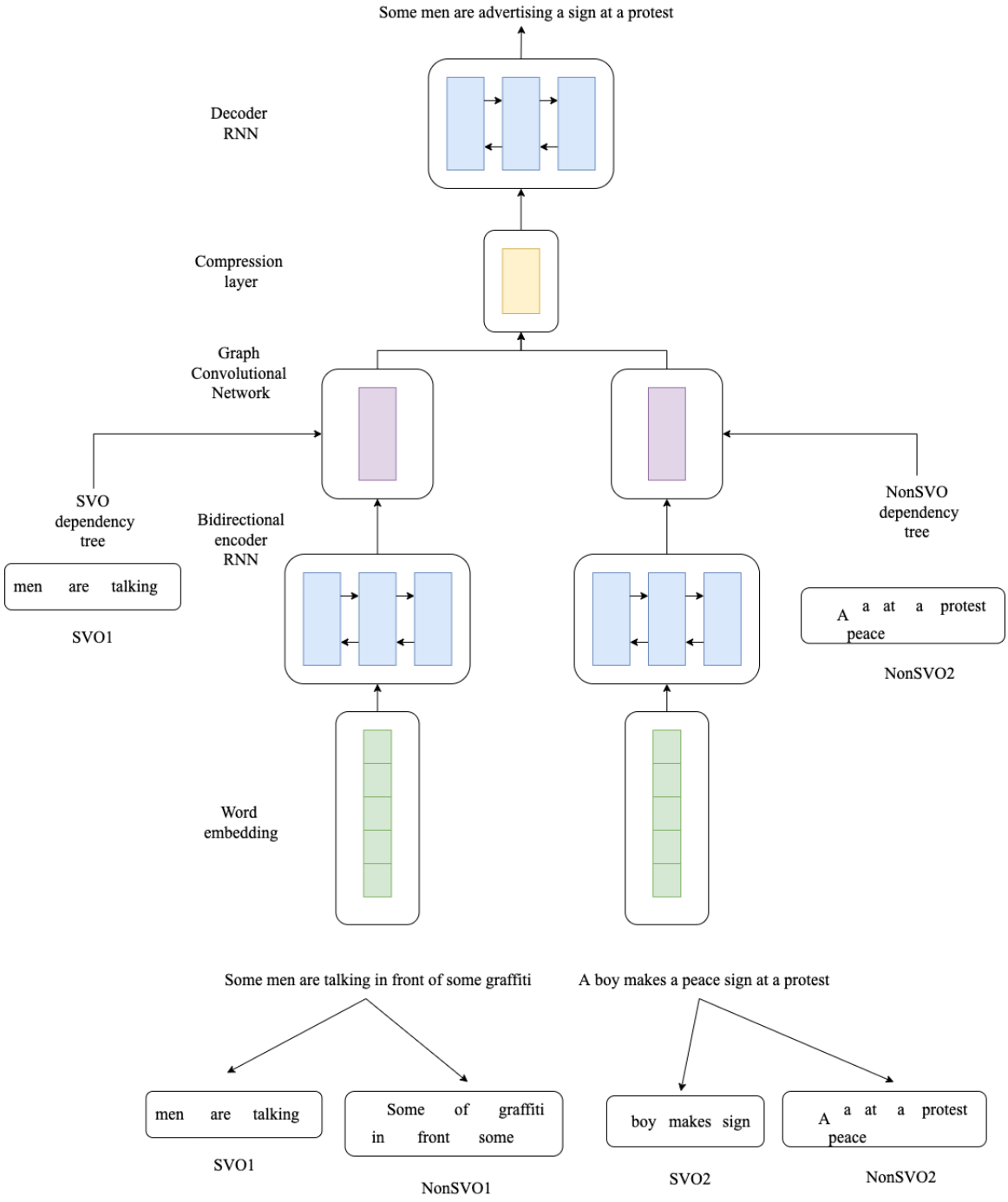

Figure 3.5: Combining SVO and NonSVO from two different sentences

Chapter 4

Experiments

4.1 Dataset

We perform our experiments on the Stanford Natural Language Inference (SNLI) dataset [Bowman et al., 2015]. SNLI [1] is a popular and widely used dataset and can be used for testing NLP models in different contexts. It serves as a benchmark dataset to evaluate representation learning systems. It was introduced in 2015 to fill the gap between increasingly powerful and larger models that would benefit from large datasets and the lack of general-purpose large datasets for evaluating NLP representation learning systems. The sentences are written by humans describing image captions, and are generally short and declarative in nature. They serve as a good testbed to validate generating sentences while separating their syntax components.

4.2 Data Preprocessing

A standard preprocessing pipeline is used.

1. All sentences are converted to lowercase and all trailing and leading whitespaces are removed.
2. All symbols that are not part of the alphabet are removed.

[1] https://nlp.stanford.edu/projects/snli/

Sentences
a dog is with a stick.
a child is in the cold.
a camel is being used to carry things.
a woman is near some water.
a man boxing with an opponent.
a kid shovels his driveway.
there are children leaning against the wall.
dogs gathering outside.
construction workers have started to do work somewhere.
a person in a red helmet riding a bike.
women are walking to their etiquette class.
a man mows a small lawn with a push mower.
two friends are at a diner eating lunch.

Table 4.1: Sentences from SNLI dataset

3. A word2vec model is generated for every word in the dataset with a CBOW model (see section 2.2.5). The embedding dimension is set to 300 with a window size of 5.

4. 90% of the dataset is used for training with 5% for validation and testing each. This yields 566408/31467/31468 sentences for the train/val/test split.

5. Each sentence is appended with a start of sentence or SOS token and end of sentence or EOS token.

6. Each sentence is also appended with a list of padding tokens based on the longest sentence in the dataset.

7. The words in each sentence are tokenized with the spaCy library ([Honnibal and Montani, 2017]).

4.3 Training details

For all the different types of models, we use the following same set of parameters and hyper-parameters. Although some of them have been mentioned before, we place them together here:

We used a batch size of 250. The encoder had 2 layers and was bi-directional. The decoder was single layer and unidirectional. We used the Adam optimizer [Kingma and Ba, 2017], with a learning rate of 0.0001 given the large size of the dataset. The hidden size for the GRUs and GCNs are 200 and the word embedding dimension is 300. The gradients are also clipped at a threshold of 50 to prevent them from exploding [2].

4.4 Results

For quantitative evaluation, we use Bilingual Evaluation Understudy or BLEU scores ([Papineni et al., 2002]) and perplexity. Table 4.2 shows the reconstruction performance for the first three models described in section 3.1. The last two rows are for the model in section 3.1.4 with the two different configurations in section 3.1.3. Table 4.3 shows the perplexity for the GCN-split model.

4.4.1 Sentence Reconstruction

Model	BLEU_1	BLEU_2	BLEU_3	BLEU_4
DAE	93.8	91.2	89.3	87.6
GCN	79.9	70.2	62.8	56.2
GCN config 1	87.8	81.1	75.7	70.8
GCN config 2	81.8	72.7	65.6	59.2

Table 4.2: Sentence reconstruction performance of the various models

Model	Perplexity
GCN config 1	75.447
GCN config 2	87.245

Table 4.3: Perplexity for reconstruction

[2]The implementation is available at: https://github.com/dasUtsav/graph-gen

4.4.2 Sentence construction with syntactic components from multiple sentences

The tables below show the BLEU_1 and perplexity scores for the multi-split models from section 3.1.5 for both dependency configurations.

Model	BLEU_1
SVO 1	30.8
SVO 2	30.3
NonSVO 1	52.7
NonSVO 2	52.6

Table 4.4: BLEU 1 scores for dependency configuration 1

Model	BLEU_1
SVO 1	21.6
SVO 2	21.7
NonSVO 1	75.3
NonSVO 2	75.2

Table 4.5: BLEU 1 scores for dependency configuration 2

Model	Perplexity
Split 1	87.632
Split 2	88.978

Table 4.6: Perplexity scores for dependency configuration 1

Model	Perplexity
Split 1	88.342
Split 2	88.396

Table 4.7: Perplexity scores for dependency configuration 2

4.4.3 Qualitative Evaluation

With a 5% validation and test split on the dataset, there are 31467 sentences in the test set. We divide this collection in two and drop the last batch of sentences to make sure the two sets are aligned. This yields 15500 sentences in each split. to take one sentence from each set and split each one into two with both of the configurations described in the previous section. With these four components of words in total, we cross-match two from each set. More concretely, we take the SVO part from the first group and the NonSVO part from the second, and feed them as input to the multi-split model described in 3.1.5. We randomly take 50 sentences in each split from the test set, and report some of them from each set here.

In each of the tables below, the first sentence is the text created by the model. The second and third sentence are the ground truth sentences from the first and second splits respectively.

Table 4.8 shows the sentences produced from the first split using configuration 1 described in section 3.1.3. The model takes the SVO parts from the first split and NonSVO parts from the second split.

Source and Generated Sentences
Generated sentence: *the young girl is at a summer area.* Source of SVO: *the small girl is at the playground* Source of NonSVO: *the couple is in very private area.*
Generated sentence: *two men standing are standing near rocks holding a flag.* Source of SVO: *two men wearing hats and holding pipes are standing against a tree to the side.* Source of NonSVO: *three children are playing with balls near a flag.*
Generated sentence: *a man is sitting on a ramp with a large shot.* Source of SVO: *a man is sitting on the floor with a burger in front of him.* Source of NonSVO: *a competition snowmobiler is on a large jump.*
Generated sentence: *the woman in snow.* Source of SVO: *woman moving in.* Source of NonSVO: *the snow is packed down in the driveway.*
Generated sentence: *a crowd of people is going at a river.* Source of SVO: *a group of diverse people around a memorial.* Source of NonSVO: *the street performer is going home.*
Generated sentence: *the snowboarder has has gone in his kitchen.* Source of SVO: *the snowboarder has gotten great air.* Source of NonSVO: *an adult is making hot chocolate in his kitchen.*
Generated sentence: *a girl is trying to two kids from a green surface.* Source of SVO: *a girl is trying to lift and put her luggage on the weigh scale.* Source of NonSVO: *two girls swing over a red patterned surface.*
Generated sentence: *a child in blue pants runs away from a woman at a camera.* Source of SVO: *a child running away from the ocean.* Source of NonSVO: *a man in blue and red face paint poses for a picture with a woman.*

Table 4.8: Sentences for split 1 from configuration 1. In each example, the first sentence is the model prediction. The model takes SVO components from the second sentence and NonSVO components from the third sentence. The SVO words in the SVO source sentence are underlined.

Table 4.9 takes the SVO parts from the second split and the NonSVO parts from the first split.

Source and Generated Sentences
Generated sentence: *people in a school bikes and is carrying a variety of plants.* Source of NonSVO: *a child with a collection of roses.* Source of SVO: *people on atvs and dirt bikes are going through an ice plateau.*
Generated sentence: *the man is trying to get a dog.* Source of NonSVO: *a dog sitting.* Source of SVO: *the man is trying to get the toy away.*
Generated sentence: *a child is holding a hand and holding a pink umbrella.* Source of NonSVO: *the guy is wearing red and black.* Source of SVO: *a child is holding their hand over their mouth wearing a birthday hat*
Generated sentence: *some asian girls are looking at a woman in water.* Source of NonSVO: *a group of men standing around with a woman in japan.* Source of SVO: *some girls are looking at fish in the water.*
Generated sentence: *a person is creating a display of paper in a blue window.* Source of NonSVO: *a hunt for the home plate by the blue catcher.* Source of SVO: *a person is viewing a piece of jewelry in a display case.*
Generated sentence: *a man is competing in a bicycle behind a sign.* Source of NonSVO: *a man wearing a cowboy hat holding a protest sign.* Source of SVO: *a man is competing against others on a bike course.*
Generated sentence: *a boy in a sweater sleeps on her head.* Source of NonSVO: *the girl with dark hair holding her head.* Source of SVO: *a dog in a superman shirt sleeps on a blanket.*
Generated sentence: *people look away from a picture.* Source of NonSVO: *a girl in a blue shirt is painting a picture.* Source of SVO: *people face away from a camera.*

Table 4.9: Sentences for split 2 from configuration 1. In each example, the first sentence is the model prediction. The model takes NonSVO components from the second sentence and SVO components from the third sentence. The SVO words in the SVO source sentence are underlined.

Table 4.10 takes the SVO parts from the first split and the NonSVO parts from the second split using configuration 2 described in section 3.1.3.

Source and Generated Sentences
Generated sentence: *a man is putting all colored food.* Source of SVO: *a man is putting cats onto a cart.* Source of NonSVO: *the man is wearing all white clothing.*
Generated sentence: *a person is wearing a suit and tie.* Source of SVO: *a person is riding on some dirt.* Source of NonSVO: *a man is wearing a suit and tie.*
Generated sentence: *the animals are in the desert in the countryside.* Source of SVO: *the animal is holding something in its mouth.* Source of NonSVO: *the runners are in the countryside.*
Generated sentence: *an old man is sleeping in a tent.* Source of SVO: *an old man in sleeping in bed.* Source of NonSVO: *football players are sleeping in a tent.*
Generated sentence: *A group of girls are on the playground.* Source of SVO: *A group of people are close together.* Source of NonSVO: *The girls are playing on the playground.*
Generated sentence: *three people in the background doing a dangerous trick at a show.* Source of SVO: *two women on the patio having drinks* Source of NonSVO: *the stunt riders are performing a dangerous trick at a show.*
Generated sentence: *an elderly girl is holding a flag.* Source of SVO: *an older woman is eating at a gathering.* Source of NonSVO: *the girl is holding a flag.*
Generated sentence: *two musicians on stage playing by the microphone.* Source of SVO: *two ladies are running away from a dog.* Source of NonSVO: *a female singer on stage playing the guitar and singing.*

Table 4.10: Sentences for split 1 from configuration 2. In each example, the first sentence is the model prediction. The model takes SVO components from the second sentence and NonSVO components from the third sentence. The SVO words in the SVO source sentence are underlined.

Table 4.11 takes the SVO parts from the second split and the NonSVO parts from the first split.

Source and Generated Sentences
Generated sentence: *the people are buying from the market.* Source of NonSVO: *men buying things from the market.* Source of SVO: *the people are in the basement drinking.*
Generated sentence: *some men are advertising a sign a protest.* Source of NonSVO: *a boy makes a peace sign at a protest.* Source of SVO: *some men are talking in front of some graffiti.*
Generated sentence: *four people are walking on on a sidewalk on a sunny day.* Source of NonSVO: *there are people walking on the sidewalk on a beautiful day.* Source of SVO: *six young women are playing string instruments such as cellos and violins in a room with wood panelling.*
Generated sentence: *a couple are walking down the street in a city.* Source of NonSVO: *some people are walking down the sidewalk in a city.* Source of SVO: *a couple are in a house.*
Generated sentence: *the two girls are playing with a baby.* Source of NonSVO: *a smiling man plays with a baby.* Source of SVO: *the ducks are following the girl to the pond.*
Generated sentence: *the person is laying around grass sitting on a hill.* Source of NonSVO: *a brown dog is sniffing around green grass on a hill.* Source of SVO: *the person is sitting indoors*
Generated sentence: *a girl was watching her favourite sandwich.* Source of NonSVO: *a woman enjoys her favourite novel.* Source of SVO: *a girl was watching television on the swing.*
Generated sentence: *man in a baseball cap while playing an instrument.* Source of NonSVO: *someone giving a concert while playing an instrument.* Source of SVO: *the man in the brown t shirt is sleeping*

Table 4.11: Sentences for split 2 from configuration 2. In each example, the first sentence is the model prediction. The model takes NonSVO components from the second sentence and SVO components from the third sentence. The SVO words in the SVO source sentence are underlined.

Table 4.12 shows some examples of imperfect generation. In example 1, the model adds some words of its own and mainly takes information from the second sentence. This is also seen in example 3. In example 2, the model replaces 'race car' with the semantically related 'motorcycle' but the sentence is absurd and doesn't have any sensible meaning. In example 4 too, the model replaces some words while retaining the semantic context such as 'sword' with 'gun' and 'camouflage color outfit' with 'blue shirt'. This shows that the

model sometimes isolates the syntax components well enough with the relevant semantic context but the generated sentence doesn't do an exact replacement from the ground truth.

Source and Generated Sentences
Generated sentence: *a young girl runs in a creek is filled with snow.* *a man presenting a plate of spaghetti.* *a blond dog walks in a creek with banks filled with snow.*
Generated sentence: *a motorcycle in a hat smiles.* *a race car sparks.* *a man in a hat stands and smiles.*
Generated sentence: *a group of people standing on a large sign near a lot.* *a woman on vacation photographs a fancy red car.* *a large group of people some holding red signs near a bunch of buildings.*
Generated sentence: *a woman in a blue shirt and glasses is holding a sword and swinging at his target.* *a man dressed in a camouflage color outfit is holding a gun and aiming it at his target.* *an asian woman and her children sit on stools at a table doing arts and crafts.*
Generated sentence: *two women are out down a street walk.* *the woman is out of a walk.* *two dogs are going down a trail outside.*

Table 4.12: Examples of generation with errors

Chapter 5

Summary and Conclusions

5.1 Summary of Research Work

In this work we see how we can split text into its syntactic components and generate sentences from them. While RNNs have been demonstrated to be effective for generating text, integrating them with GCNs lets the combined model mix representations of the sentence from the encoder and the syntax components that we choose from the GCN. This lets us influence the model to generate words corresponding to specific syntax elements.

We show with a GCN and RNN model that text reconstruction quality is maintained as opposed to vanilla RNNs. The GCN-RNN model is capable of reconstituting a sentence from its syntax and encoder representations once it has been split into two groups based on its syntax. Finally, the model is able to mix the syntax representations from multiple sentences and create sentences blending the components from them.

5.2 Conclusions

We see how RNNs and GCNs complement each other to generate sentences and control the generation at a syntax level. GCNs are able to capture the relationships between the different heads and dependents and provide them to the RNN to use while producing the sentences. Using GCNs, we can control which syntax components the RNN will be fed while the RNN decoder itself does the work of generating a coherent, fluent sentence with the syntax components provided to it.

5.3 Future Work

The SNLI dataset provides a very specific type of text in the form of simple declarative sentences that act as captions for images. Future work can investigate how to separate syntax components with longer, more complex sentences or even entire paragraphs of text. Additionally, this approach combines the two architectures of RNNs and CNNs, for text generation. We can see how combining GCNs with another architecture, like the transformer, can impact the model's performance across several metrics. Furthermore, we can investigate across what different parts-of-speech or dependency relations can we split a sentence using datasets in different genres.

References

[AlphaFold, 2020] AlphaFold (2020). AlphaFold: A solution to a 50-year-old grand challenge in biology.

[Arc, 2018] Arc (2018).

[Bahdanau et al., 2016] Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.

[Bai et al., 2021] Bai, J., Wang, Y., Chen, Y., Yang, Y., Bai, J., Yu, J., and Tong, Y. (2021). Syntax-BERT: Improving pre-trained transformers with syntax trees. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3011–3020, Online. Association for Computational Linguistics.

[Bengio et al., 2000] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2000). A neural probabilistic language model. In *J. Mach. Learn. Res.*

[Bengio and Lecun, 1997] Bengio, Y. and Lecun, Y. (1997). Convolutional networks for images, speech, and time-series.

[Bowman et al., 2015] Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

[Bowman et al., 2016] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss,

A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *CoRR*, abs/2005.14165.

[Caliskan et al., 2017] Caliskan, A., Bryson, J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science*, 356:183–186.

[Cambria and White, 2014] Cambria, E. and White, B. (2014). Jumping nlp curves: A review of natural language processing research [review article]. *IEEE Computational Intelligence Magazine*, 9(2):48–57.

[Celikyilmaz et al., 2018] Celikyilmaz, A., Bosselut, A., He, X., and Choi, Y. (2018). Deep communicating agents for abstractive summarization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1662–1675, New Orleans, Louisiana. Association for Computational Linguistics.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Chollet, 2017] Chollet, F. (2017). *Deep Learning with Python*. Manning.

[Copeland, 2016] Copeland, M. (2016). What's the difference between artificial intelligence, machine learning and deep learning?

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cogn. Sci.*, 14:179–211.

[Evans et al., 2021] Evans, R., O'Neill, M., Pritzel, A., Antropova, N., Senior, A., Green, T., Žídek, A., Bates, R., Blackwell, S., Yim, J., Ronneberger, O., Bodenstein, S., Zielinski, M., Bridgland, A., Potapenko, A., Cowie, A., Tunyasuvunakool, K., Jain, R., Clancy, E., Kohli, P., Jumper, J., and Hassabis, D. (2021). Protein complex prediction with alphafold-multimer. *bioRxiv*.

[GE, 2019] GE (2019). What is ImageNet and Why 2012 Was So Important. Retrieved on 12 November, 2021.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

[He et al., 2020] He, Q., Wang, H., and Zhang, Y. (2020). Enhancing generalization in natural language inference by syntax. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4973–4978, Online. Association for Computational Linguistics.

[Hewlett et al., 2017] Hewlett, D., Jones, L., Lacoste, A., and Gur, I. (2017). Accurate supervised and semi-supervised machine reading for long documents. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2011–2020, Copenhagen, Denmark. Association for Computational Linguistics.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Honnibal and Montani, 2017] Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

[Hu et al., 2021] Hu, Z., Lee, R. K.-W., and Aggarwal, C. C. (2021). Syntax matters! syntax-controlled in text style transfer. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 566–575, Held Online. INCOMA Ltd.

[Hussain et al., 2019] Hussain, S., Sianaki, O. A., and Ababneh, N. (2019). A survey on conversational agents/chatbots classification and design techniques. In *AINA Workshops*.

[Jurafsky and Martin, 2009] Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes.

[Kipf, 2016] Kipf, T. (2016). Graph convolutional networks.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[Kuncoro et al., 2019] Kuncoro, A., Dyer, C., Rimell, L., Clark, S., and Blunsom, P. (2019). Scalable syntax-aware language models using knowledge distillation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3472–3484, Florence, Italy. Association for Computational Linguistics.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Li et al., 2015a] Li, J., Luong, M.-T., and Jurafsky, D. (2015a). A hierarchical neural autoencoder for paragraphs and documents.

[Li et al., 2016] Li, J., Monroe, W., Ritter, A., Jurafsky, D., Galley, M., and Gao, J. (2016). Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas. Association for Computational Linguistics.

[Li et al., 2015b] Li, S., Kawale, J., and Fu, Y. (2015b). Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 811–820, New York, NY, USA. Association for Computing Machinery.

[Marcheggiani and Titov, 2017] Marcheggiani, D. and Titov, I. (2017). Encoding sentences with graph convolutional networks for semantic role labeling.

[Marin et al., 2019] Marin, J., Biswas, A., Ofli, F., Hynes, N., Salvador, A., Aytar, Y., Weber, I., and Torralba, A. (2019). Recipe1m+: A dataset for learning cross-modal

embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

[Mou and Vechtomova, 2020] Mou, L. and Vechtomova, O. (2020). Stylized text generation: Approaches and applications. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 19–22, Online. Association for Computational Linguistics.

[Narayan et al., 2018] Narayan, S., Cohen, S. B., and Lapata, M. (2018). Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759, New Orleans, Louisiana. Association for Computational Linguistics.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

[Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *ArXiv*, abs/1211.5063.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., and

Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations.

[Piper, 2020] Piper, K. (2020). GPT-3, explained: This new language AI is uncanny, funny — and a big deal. Retrieved on 12 November 2021.

[Reiter and Dale, 2000] Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

[Ruder, 2018] Ruder, S. (2018). Nlp's imagenet moment has arrived. *The Gradient*.

[Ruder, 2022] Ruder, S. (2022). Nlp progress.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[Schrittwieser et al., 2020] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

[Shu et al., 2021] Shu, X., Bao, T., Li, Y., Gong, J., and Zhang, K. (2021). Vae-talstm: a temporal attention and variational autoencoder-based long short-term memory framework for dam displacement prediction. *Engineering with Computers*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Sobowale, 2016] Sobowale, J. (2016). How artificial intelligence is transforming the legal profession.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.

[TMF, 2021] TMF (2021). Artificial intelligence in healthcare: 10 medical fields A.I. will change completely.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[Weng, 2018] Weng, L. (2018). From autoencoder to beta-vae.

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280.

[Wu et al., 2018] Wu, L., Tian, F., Qin, T., Lai, J., and Liu, T.-Y. (2018). A study of reinforcement learning for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3612–3621, Brussels, Belgium. Association for Computational Linguistics.

[Xu et al., 2015] Xu, Y., Mou, L., Li, G., Chen, Y., Peng, H., and Jin, Z. (2015). Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1785–1794, Lisbon, Portugal. Association for Computational Linguistics.

[Zhao et al., 2019] Zhao, H., Chen, Z., Jiang, H., Jing, W., Sun, L., and Feng, M. (2019). Evaluation of three deep learning models for early crop classification using sentinel-1a imagery time series-a case study in zhanjiang, china. *Remote Sensing*, 11:2673.

[Zhao et al., 2021] Zhao, H., Wang, R., and Chen, K. (2021). Syntax in end-to-end natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pages 27–33, Punta Cana, Dominican Republic & Online. Association for Computational Linguistics.

[Zon and Ditta, 2016] Zon, N. and Ditta, S. (2016). Robot, take the wheel: Public policy for automated vehicles.