From Autoregressive to Non-Autoregressive: Studies on Text Generation in Neural Sequence Models

by

Syed Najam Abbas Zaidi

Supervisors:

Assoc. Prof. Gholamreza Haffari

Prof. Trevor Cohn (University of Melbourne)

Prof. Hans De Sterck (University of Waterloo)

A Thesis Submitted for the Degree of Doctor of Philosophy at

Monash University in 2022

Department of Data Science and Artificial Intelligence

Faculty of Information Technology

Monash University

To my late mother: Ami you gave me wings, Father: Abu you taught me to fly and my beloved wife who retaught me the meaning of life

From Autoregressive to Non-Autoregressive: Studies on Text Generation in Neural Sequence Models

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Syed Najam Abbas Zaidi
April 26, 2022

From Autoregressive to Non-Autoregressive: Studies on Text Generation in Neural Sequence Models

Syed Najam Abbas Zaidi
syed.zaidi1@monash.edu
Monash University, 2022

Supervisors:
Assoc. Prof. Gholamreza Haffari
Prof. Trevor Cohn (University of Melbourne)
Prof. Hans De Sterck (University of Waterloo)

Abstract

Many tasks in natural language processing (NLP) require predicting complex structures such as a sequence, tree or graph as output and therefore can be framed as a structured prediction problem. In this thesis, we narrow our focus on text generation among various other structured prediction problems. Sequence generation in NLP requires predicting text as output and cover application areas such as machine translation, summarization and story generation. The current state of the art methods for text generation relies on deep neural networks to model the relationship between the input and the output structure. Despite all the progress, the current autoregressive text generation methods, where the output is generated monotonically from left to right conditioned on the source and previously generated tokens, ignore the structural nuances of human languages and rely on simplistic assumptions and error prone search procedures. Furthermore, the current approaches suffer from label-bias where previously generated errors are propagated throughout the generation and exposure-bias where there is a discrepancy between training and inference procedures.

Non-autoregressive models have been proposed recently to generate output sequence in a fixed number of time-steps independent of the length of the output sequence. As the whole sequence is generated in parallel, non-autoregressive models have neither label nor exposure bias. But due to complete conditional independence in the output variables, their output quality suffers. In both of these model families, the length of the target sequence is either fixed or increase monotonically as the generation proceeds. This does not allow dynamic length changes and reediting incorrectly generated parts of the sequence. Further, both these model families struggle to keep coherence in the text when generating long sequences.

We begin, in Chapter 3, by proposing an effective method for improving decoding in discrete autoregressive models using dynamic programming, thus reducing the search error. The core idea is to introduce auxiliary variables to decouple the non-Markovian aspects of the model, permitting an exact solution to an approximate model. This solution is then used to create the next model approximation, and the process iterates. Our results show that our decoding framework is effective, leading to substantial improvements over greedy and beam search baselines.

In Chapter 4, we move our attention to long sequence generation. We propose a novel semi-autoregressive document generation model capable of revising and editing the generated text. Semi-autoregressive models provide the best of both worlds by decoding substantially faster than autoregressive but with some performance cost. We formulate document generation as a hierarchical Markov decision process with a two-level hierarchy. The combination of editing actions at high (sentence) and low (word) level allow the model to revisit and edit part of the text. This not only allows dynamic length changes but also allow rectifying the generated mistakes. We apply our semi-autoregressive document generation model to the challenging task of story generation and summarization. Our approach produces promising results but underperforms strong baselines. We suggest various future directions that may improve the results.

The final contribution of this thesis, in Chapter 5, is to propose an innovative architectural design for generating quality outputs for non-autoregressive models based on similar examples from the training set. Non-autoregressive models lag behind autoregressive models due to loss of crucial information about dependencies among the variables. We propose to inject this information by using an informative and effective prior based on similar examples. We build an augmented version of the dataset by fetching k nearest neighbours from the training set. We also propose changes in the attention mechanism, whereby attention heads are distributed between nearest neighbours and the sentence being translated. This allows the model to incorporate similar examples effectively and not get distracted by extra information. Our proposed approach outperforms the vanilla non-autoregressive baseline model but underperforms other similar non-autoregressive baseline approaches. We suggest various ways to improve results as future directions.

Acknowledgments

It was an incredible journey of learning and self discovery. A journey mostly filled with disappointments and frustrations but with little moments of joy in the way. A journey that took me out of my comfort zone into the unknown territories of knowledge. Now that I look back at it, I feel proud to have come out of it successfully. The knowledge I have gained and the skills I have learned will be beneficial for me in a number of ways.

It is said that research is a lonely quest but I was lucky to have friendly and supporting supervisors to guide me throughout my candidature. I cannot thank Associate Professor Gholamreza (Reza) Haffari, Professor Trevor Cohn (University of Melbourne) and Professor Hans De Sterck (University of Waterloo) much for their support and encouragement. They patiently supervised and answered even the most dumbest questions, showed me the end of the tunnel when I was completely lost and provided me with encouraging words to boost my morale throughout the project. I would like to express my sincere gratitude to my supervisors as this thesis would not have been possible without their guidance and support.

I would like to thank Monash University for generously supporting me with Monash Graduate Scholarship. I have also been greatly benefited from Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE) and Monash Advanced Research Computing Hybrid (MonARCH) by providing me with the opportunity to run many resource-intensive experiments.

I would also like to thanks my panel members Prof. Wray Buntine and Dr. Daniel Schmidt for providing me with constructive feedback throughout the candidature. A special thanks goes to the examiners of the thesis Professor Christof Monz (Informatics Institute, University of Amsterdam) and Dr Carolina Scarton (University of Sheffield). Their valuable feedback has further improved the quality of this thesis.

This whole journey would have not been possible without the support of my family. Everything I am and everything I have achieved is because of them. Their countless encouragement and support gave me the reason to keep on making an effort in times of continuous failures. I thank you guys a lot.

Special gratitude goes to my beloved wife Sumbleen for her unconditional love, motivation, encouragement and moral support over many years. She patiently stood with me in times of extreme pressure and frustration and provided me emotional support when I was down.

Last but not the least, I am grateful to my colleagues and officemates in Monash, Fahimeh, Ming, Poorya, Narjes, Philip, Sameen, Snow, Trang, Xuanli for their support, kindness, and the fun we have had over in the lab before COVID hit.

Syed Najam Abbas Zaidi

Monash University
April 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Many tasks in natural language processing require modelling complex inter-dependencies between the output variables. Examples of such tasks include but are not limited to machine translation (Sutskever, Vinyals, & Le, 2014), summarization (Wan, Luo, Sun, Huang, & Yao, 2019), parsing (Socher, Lin, Manning, & Ng, 2011) and semantic role labeling (J. Zhou & Xu, 2015). The dependencies between the output variables lead to complex structures such as a sequence, tree or a graph as output. For example, automatic machine translation involves predicting a sequence of words in a target language for a given source language. The generated output sentence should obey the structural and statistical constraints of the target language so that the resulting sentence is well formed and convey the idea of the source sentence as faithfully as possible. This is only possible if the underlying model is capable of predicting structurally rich and interdependent output variables. In the case of machine translation, the output structure is a sequence of words which needs to be syntactically and semantically correct. In the parlance of machine learning, such problems are categorised as structured prediction problems (Murphy, 2012).

Structured prediction problems differ from other classification problems such as multiclass and multilabel. In multiclass classification, an instance is classified into one of three or more classes. A simpler case of multiclass classification is binary classification where the instance is classified into one of the two classes. In multilabel classification, multiple labels may be assigned to a given instance. Structured prediction problem, on the other hand, requires modelling deeper properties of instances and their relationship to each other. Formally, consider an input vector $\mathbf{x} = \langle x_1, x_2, x_3, ..., x_n \rangle$ such that each $x_i \in \mathcal{X}$ and output vector $\mathbf{y} = \langle y_1, y_2, y_3, ...y_n \rangle$ such that each $y_i \in \mathcal{Y}$. For machine translation $\mathcal{X}$ is the source vocabulary and $\mathcal{Y}$ is a target vocabulary. We are interested in finding a mapping such that $\mathcal{X}^n \longmapsto \mathcal{Y}^m$ where n and m are the length of the source and target sequence respectively. Martins (2012) presented three conditions that must be satisfied by the problem to be categorised as a structured prediction problem. First, in case of fewer classes, we predict $\mathbf{y}$ by searching the whole output space $\mathcal{Y}$ but this is not feasible in case of structured prediction as some structures are not legitimate. For example in machine translation, the output structure should obey the grammar of the target language. Therefore, $\mathcal{Y}(x)$ is defined as the set of admissible outputs. Secondly, the output set $\mathcal{Y}(x)$ is large such that it is impossible to enumerate all the elements in the set. This set grows exponentially with length of the input $\mathbf{x}$. Finally, there are strong dependencies among the atomic components due to structural or statistical constraints and therefore components cannot be predicted individually.

At its core, structured prediction requires solving an optimisation problem at the training and inference time. Given some training dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_1^N$, the optimisation problem at the training time, to learn the model, can be solved by maximum likelihood estimate (R. A. Johnson, Miller, & Freund, 2000) and is given by:

$$\mathbf{w}^* = \arg\max_{\mathbf{w} \in \mathcal{W}} \sum_{i=1}^{N} log\ p(\mathbf{y}_i | \mathbf{x}_i; \mathbf{w}) \tag{1.1}$$

This problem requires searching for a model parameter $\mathbf{w}$ that maximises the likelihood of data under the model. The optimisation problem at inference time, to predict the output of given input, is given by:

$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} p(\mathbf{y} | \mathbf{x}; \mathbf{w}^*) \tag{1.2}$$

This problem requires searching for a high scoring structure under the given model. A Challenge lies in finding $\mathbf{y}^*$ as this problem in general is not tractable. The possible search space $\mathcal{Y}(\mathbf{x})$ is in general exponentially large and grows with the input size. Exact inference is desirable but requires huge computational cost (Kulesza & Pereira, 2008). Therefore, current approaches use simplistic and approximate procedures where a sacrifice is made towards accuracy in favour of time.

In this thesis, we consider sequence generation among other structured prediction problems in NLP. Sequence generation problems require generating text as output and therefore encompass problems such as machine translation (Wu et al., 2016), summarization (El-Kassas, Salama, Rafea, & Mohamed, 2021), story generation (Fang et al., 2021) and various document level tasks such as document level machine translation, discourse analysis and document level post processing (Z. Ma, Edunov, & Auli, 2021). Current state-of-the-art text generation technologies rely on deep neural networks to parametrise the probability distribution of output sequence $\mathbf{y}$ given an input sequence $\mathbf{x}$ (Sutskever et al., 2014; Bahdanau et al., 2014). An encoder builds an internal representation which is used by the decoder to generate the output autoregressively from left to right (Figure 1.1).

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

The autoregressive factorization that is the output token is generated conditioned on all the previously generated tokens as shown by $\mathbf{y}_{<i}$ comes with several advantages. As this word by word left to right, generation corresponds to the order in which humans produce languages therefore the

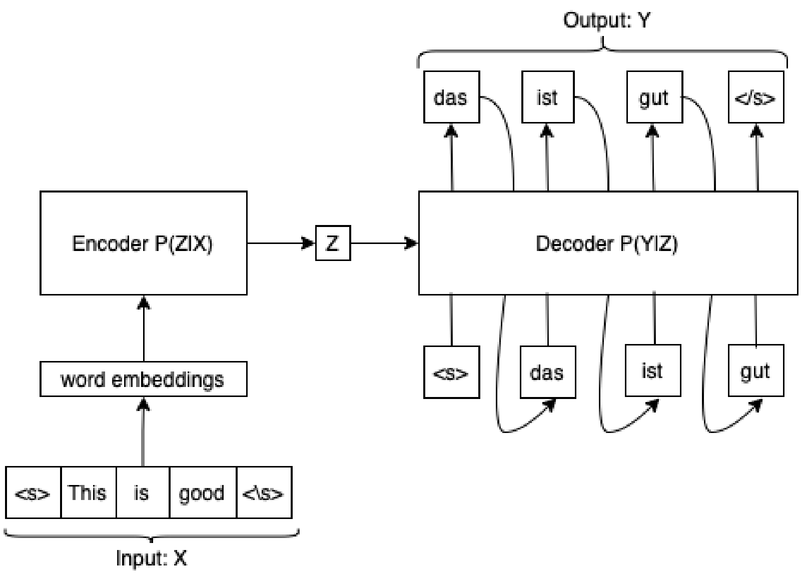


Figure 1.1: The figure shows an encoder-decoder architecture used for machine translation. An encoder convert the input sentence X to an internal representation z. The decoder use this internal vector z to generate the output Y in the target language. Typically, the decoder is autoregressive that is output words are generated one by one from left to right which each word using previously generated words as conditioning context.

underlying model effectively captures the real data distribution. Generation, which is referred to as decoding or inference, is done via greedy or beam search procedures. Autoregressive models have achieved state-of-the-art performance on many tasks with large-scale corpora (Vaswani et al., 2017; G. Bao, Zhang, Teng, Chen, & Luo, 2021). They are easy to train and beam search provides an effective local search method for finding approximately-optimal output translations.

But there are several disadvantages. First, the decoder must output tokens sequentially, rather than in parallel, resulting in a slower generation process (Ramachandran et al., 2017). The ability to do fast generation is useful for production environments with tight latency constraints such as fast off-line speech generation or machine translation systems (Inaguma, Higuchi, Duh, Kawahara, & Watanabe, 2021) . The issue of latency is further exacerbated when autoregressive models are used for generating long sequences such as stories and summaries. Not only does the generation become slow, hindering its use in industrial applications but the models also struggle to maintain a coherent event sequence throughout the generated text. B. Tan, Yang, AI-Shedivat, Xing, and Hu (2020) noted in their work on long text generation that state-of-the-art language models show excessive repetitiveness and incoherence between sentences. The issue of repetitiveness was also reported by Cohen and Beck (2019) and Holtzman, Buys, Du, Forbes, and Choi (2019), citing simple generation procedures as the culprit for this behaviour. Further, Cohen and Beck (2019) showed that increasing the beam size leads to performance degradation.

Current autoregressive models also have exposure bias, meaning that there is a discrepancy between training and inference procedure (Ranzato, Chopra, Auli, & Zaremba, 2015). At training time the ground truth words are used as context whereas at the test time the model relies on its own generations. As a result the predicted word at training and inference time are drawn from different distributions (Zhang, Feng, Meng, You, & Liu, 2019). Further, monotonic generation of words also results in the label-bias problem where the previously generated errors are propagated throughout the generation, resulting in a poor quality solution (Goyal, Dyer, & Berg-Kirkpatrick, 2019).

Recent work by Gu, Bradbury, et al. (2017) introduced non-autoregressive generation models capable of generating the whole output sequence in a fixed number of time-steps independent of the length of the output sequence. This provide significant speedups compared to autoregressive models (Figure 1.2). Further as the whole sequence is generated in parallel, non-autoregressive models neither have exposure nor label bias. But all these advantages comes at a cost. Complete conditional independence makes it difficult for non-autoregressive models to capture the underlying data distribution which exhibit strong correlation across time. Since the generation of target tokens is now independent, non-autoregressive models have to utilize alternative information to compensate for the lack of target side history. This makes training of non-autoregressive models harder, as they have to handle the task conditioned on less and weaker target side information, resulting in inferior quality outputs. Further, non-autoregressive models have been shown to perform poorly when generating long sequences due to lack of target side information (Guo et al., 2019). Non-autoregressive generation models also require predicting target sequence length before generation. The lengths are implicitly model by autoregressive models using a special <*sep*> end of sentence symbol. As the whole sequence is now predicted in parallel, non-autoregressive models need to explicitly model sequence lengths. In both of these model families, the length of generated sequences is either fixed or monotonically increase as the decoding proceeds. This does not allow dynamic length changes that is the capability of the model to revisit and edit any generated text.

Semi-autoregressive models (C. Wang, Zhang, & Chen, 2018; Ghazvininejad, Levy, & Zettlemoyer, 2020) can decode substantially faster but with some cost to performance. These models usually perform an iterative refinement of the sequence or generate chunks of successive words non-autoregressively while generating each chunk autoregressively (See figure 1.2). As the model lose some dependency information, semi-autoregressive model lags behind autoregressive models but perform better than non-autoregressive models in terms of output quality.

Thus, there is a balancing act between quality and time. Both model families come with their advantages and disadvantages. Autoregressive models are slow but their output is better compared to non-autoregressive models which are faster. Semi-autoregressive models provide the best of both worlds by providing a compromise in both time and quality. Therefore all three model families remain prevalent in the generation literature and we address their limitations in this thesis.

Concretely, one of the limitation for autoregressive models, as mentioned above, is that the standard decoding procedures such as greedy and beam search are suboptimal. This is due to error propagation and myopic decisions which do not account for future steps in the generation process. Secondly, autoregressive models are slow when generating long sequences. Non-autoregressive models can speed up generation but perform badly due to the loss of conditioning context. In both of these model families, the sequences are bounded by fixed length whereas humans can edit any part of their generated text resulting in output of different length. Furthermore, the generation

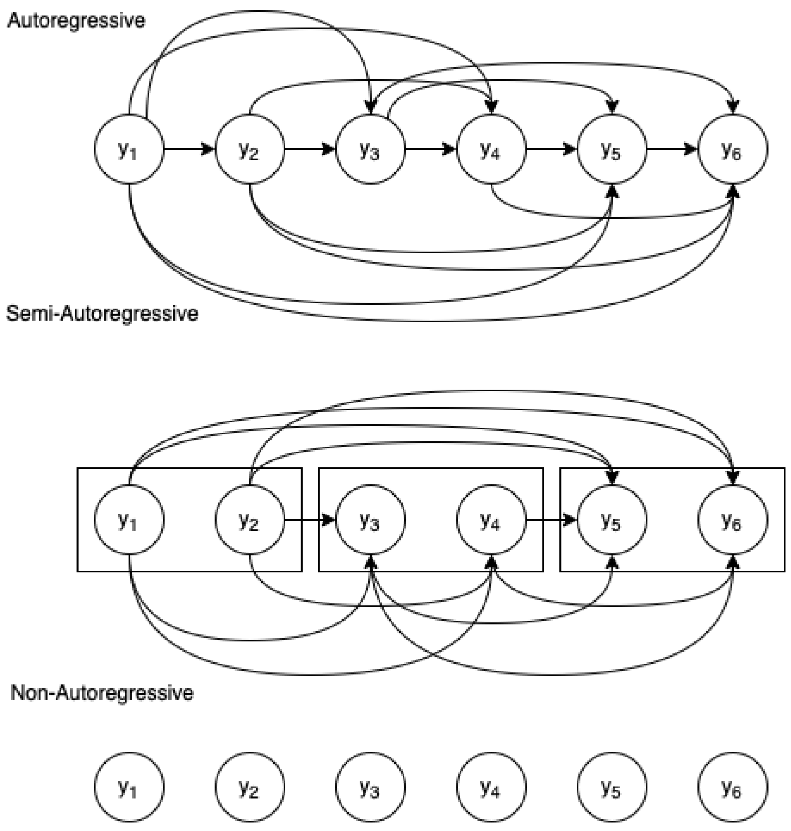


Figure 1.2: Conditional dependencies among model families. Autoregressive models generate token conditioned on all the previously generated tokens. Semi-autoregressive generate blocks autoregressively whereas within each block generation is done in parallel. Tokens in a block are conditioned upon the tokens generated in the previous blocks Non-autoregressive models, on the other hand show no conditional dependence among tokens.

procedure in both model families are simple and ignore the underlying structure in languages. Finally, there is still a large gap in translation quality between autoregressive and non autoregressive models due to lack of target side information.

1.2 Research Objectives

The overarching aim of this research is to improve generation across both autoregressive and non-autoregressive generation models by addressing the shortcomings mentioned above. Therefore motivated by the literature gaps, this thesis will address the following objectives:

- **Effective decoding method for faster and higher quality output in autoregressive models (Part I).** Current prevalent decoding methods for autoregressive models such as greedy and beam search are ineffective in finding a high scoring sequence under a given model. They are too simplistic, error prone and unidirectional, meaning that once a wrong token has been generated, it will be propagated throughout the process in the conditioning context affecting the subsequent tokens (Sutskever et al., 2014; Bahdanau et al., 2014). This results in serious mistakes in the output. In order to overcome this deficiency, this thesis suggests an iterative approach capable of revisiting and revising the generated text. Building on the method of auxiliary coordinates (Carreira-Perpinan & Wang, 2014), the proposed method in Chapter 3 alternate between generating a sequence and updating the state variables allowing the model to revise and fix any mistake in the generated sequence. This is in contrast to greedy and beam search, where once a token has been generated it cannot be updated.

- **Novel model design leveraging document structure for faster and higher quality long text generation with length flexibility (Part II).** Current autoregressive models are slow when generating long sequences such as documents. On the other hand, non-autoregressive models show degraded performance when generating long sequences. Furthermore, in both of these model families, the length of generated sequences is either fixed or monotonically increased as the decoding proceeds. This makes them incompatible with human-level intelligence where humans can revise and edit any part of their generated text. Furthermore, simple generation procedures ignore the structural nuances of human languages leading to mistakes in the output. This thesis suggests a semi-autoregressive model which is capable of revising and editing the generated text. A central premise of natural languages is that words in a sentence are interrelated according to a latent hierarchical structure such as syntactic tree (Lees, 1957). This idea can be extended to documents where sentences in a document are interrelated according to the topic and theme. Therefore the suggested approach in Chapter 4 frames the document generation problem as a hierarchical Markov decision process, where the high level program is responsible for keeping coherency throughout the document and the low level program is responsible for generating individual sentences.

- **Innovative architectural design for generating quality outputs by incorporating informative prior in non-autoregressive models (Part III).** Despite all the progress that has been made in non-autoregressive models, they still lag behind their autoregressive counterparts. Previous work on non-autoregressive models by (Gu, Bradbury, et al., 2017; Guo et al., 2019; Y. Wang et al., 2019) noted that the input to the decoder plays a crucial rule in modeling the target side dependencies among the tokens. The decoder in autoregressive models leverages the previous tokens as input whereas the decoder in non-autoregressive models takes target independent signal such as copy of the source sentence as input. Obviously, they are not rich enough to compensate for the loss of information. This thesis suggests an informative and effective decoder input, that is capable of capturing the complex dependencies in the target side data distribution. Therefore, instead of copying the source sentence which is the prevalent method in non-autoregressive models, Chapter 5 propose using k nearest neighbours to the given sentence as input to the decoder.

1.3 Thesis Outline and Contributions

Chapter 2: Background

In Chapter 2, we review the foundations and related work for the research in this thesis. We start by discussing the foundations of deep learning and its use in natural language processing. Then, we review sequence generation techniques for neural models. We specifically focus on autoregressive, semi-autoregressive and non-autoregressive models and their generation techniques. We also ground the motivations of this thesis by analysing shortcomings of the current approaches in this chapter.

Part I: This part is dedicated to proposing new generation methods for autoregressive models.

Chapter 3: Decoding as Dynamic programming For Recurrent Autoregressive Models

This chapter is based on:
Zaidi, Najam, Trevor Cohn, and Gholamreza Haffari. "Decoding as dynamic programming for recurrent autoregressive models." International Conference on Learning Representations. 2019.

To handle errors in autoregressive generation models, we propose an approach based upon the method of auxiliary coordinates (Carreira-Perpinan & Wang, 2014). Our method introduces discrete variables for output tokens, and auxiliary continuous variables representing the states of the underlying autoregressive model. The auxiliary variables lead to a factor graph approximation of the model, whose maximum a posteriori (MAP) solution is found exactly using dynamic programming. The MAP solution is then used to recreate an improved factor graph approximation of the autoregressive model via updated auxiliary variables. Alternating between generation and updating state variables allows fixing mistakes that could possibly leads to inferior quality outputs. The proposed approach is applied on the challenging text infilling task, which consists of filling missing part of the sentence. The task is challenging as it requires both the left and the right context. Experiments applying the proposed approach shows that our decoding method is superior to strong competing decoding methods. Following previous work on text infilling, we use BLEU score (Papineni, Roukos, Ward, & Zhu, 2002) to evaluate the models.

Part II: This part is dedicated to proposing new model for long sequence generation. Our contribution is a semi-autoregressive model that allows dynamic length changes.

Chapter 4: A Hierarchical Model For Document Generation

This chapter is based on:
Zaidi, Najam, Trevor Cohn, and Gholamreza Haffari. "Document Level Hierarchical Transformer." Workshop of the Australasian Language Technology Association. 2021.

Autoregressive models are slow and struggle in maintaining coherency when generating long sequences. We propose a semi-autoregressive model by framing document generation as a hierarchical Markov decision process with a two level hierarchy, where the high and low level editing programs generate and refine the document. Our proposed approach allows revisiting and updating the generated text with the help of editing operators, allowing words to be deleted or inserted thus allowing dynamic length changes. We applied the proposed approach on synthetic dataset, where input-output pair is an unsorted and sorted sequence of numbers, story generation as well as summarization dataset. We used BLEU and ROGUE score to evaluate the tasks. Our result shows that applying the proposed approach **underperform** state-of-the-art models but shows promise and conveys various insights on the problem of long text generation using our model.

Part III: This part is dedicated to improving quality of output in non-autoregressive models. Our contribution is a data-driven prior as well as architecture design to incorporate it.

Chapter 5: Exemplar Transformer Non-autoregressive models suffers from inferior quality outputs because of the complete conditional independence in the output variables. We propose an informative prior based upon the k-nearest neighbours of given source test sentence to the sentences in the training dataset. We used a similarity measure such as ℓ_2 distance to find the nearest neighbours from the training dataset. We also propose changes in the attention mechanism, whereby attention heads are distributed between nearest neighbours and the sentence being translated. This allows model to incorporate similar examples effectively.We applied the proposed approach on two widely used machine translation datasets and used BLEU score to evaluate the results. Our proposed approach shows comparable performance with vanilla non-autoregressive model and **inferior** performance to other state-of-the-art non-autoregressive models.

Chapter 6: Conclusions and Future Directions

Chapter 6 concludes this thesis by outlining the contributions as well as potential directions for future research.

Chapter 2

Background

The aim of this thesis is to improve sequence generation across autoregressive and non-autoregressive model families by addressing the following shortcomings in the current approaches (Section 1.1):

- Ineffective decoding methods for autoregressive models that allow error to be propagated in the generation process.
- Inability of autoregressive and non-autoregressive models to revisit and reedit the generated text for long text generation.
- Uninformative prior in non-autoregressive models for injecting dependency information.

Therefore, in this chapter, we overview the foundations and prior works related to autoregressive, semi-autoregressive and non-autoregressive models along with their generation techniques. The chapter starts by giving a brief overview of deep learning and NLP in Section 2.1. This section covers perceptrons, recurrent neural network (RNN) and its variants along with word embeddings and language models. The content in this section is helpful for understanding the advance models presented later in the chapter. After establishing the basics, Section 2.2 presents autoregressive sequence modelling techniques. It covers encoder-decoder models and its variants along with the current state of the art Transformer model. Section 2.3 outlines the recently proposed non-autoregressive models. We present extensive overview of non-autoregressive models along with their training and inference schemes. Section 2.4 summarizes literature in current semi-autoregressive models. Section 2.5 presents an alternative view for autoregresive models as locally normalised models. Finally, the chapter ends with a summary in Section 2.6.

Beginning words: Formally, we define an input sequence $\mathbf{x} = \{x_1, ..., x_n\}$ and an output sequence $\mathbf{y} = \{y_1, ..., y_m\}$. The conditional probability distribution $P_\theta(\mathbf{y}|\mathbf{x})$ is parameterized by θ. This complete chapter is concerned with presenting different models for learning θ. Furthermore, as the factorization of this probability distribution give rise to autoregressive, non-autoregressive and semi-autoregressive models, this chapter also presents various generation techniques for these models.

2.1 Deep Learning and NLP

2.1.1 Deep Learning

Deep learning has become the buzz word of modern times yet its history can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks of the human brain (McCulloch & Pitts, 1943). Later work by Rosenblatt (1958) and Minsky and Papert (1969) further established the field. These methods, along with large amount of data, required huge computational resources and were way ahead of their time. Thanks to recent advances in computational power, deep learning methods have revolutionised so many areas such as computer vision, natural language processing (NLP) and speech processing.

Deep learning or Deep neural networks, as a branch of Machine Learning, employs algorithms to process data and develop abstractions. Starting with the raw data, information is passed through each layer, with the output of the previous layer providing input for the next layer. Each layer transforms the input to a slightly higher abstract representation by using simple non-linear modules. Deep neural networks are in fact function approximators. They can learn any complex function if enough layers are stacked (LeCun, Bengio, & Hinton, 2015). For example, in a classification task, higher level layers show representations that reflect important information for discriminating between classes without containing irrelevant information.

In the rest of this section, we will cover neural network methods that will form the building block of more complicated models.

2.1.1.1 Multi-Layer Perceptron (MLP)

Perceptron is the building block of MLP model. It takes weighted sum of its inputs $(\sum_i w_i x_i + b)$ and pass it through a non-linear function $\phi(\sum_i w_i x_i + b)$ called the activation function. These activation functions should always be non-linear, otherwise a multi-layer network reduces to linear single layer network. Popular choices of activation functions includes a `sigmoid` function with range in [0,1]:

$$\phi(v) = \frac{1}{1 + e^{-v}} \tag{2.1}$$

Another popular choice is the `hyperbolic tangent` function with range in [-1, 1]:

$$\tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \tag{2.2}$$

A single layer of perceptron can learn simple mappings but is not powerful enough to learn non-linearly separable data. Rumelhart, Hinton, and Williams (1985) proposed to stack multiple layers of perceptron and called the resulting model Multi-Layer Perceptron (MLP) or a feed-forward neural network. The first layer in MLP is called the input layer, the last layer as the output layer and the middle layers as hidden layers. MLP is a composition of functions and can be mathematically expressed as:

$$\mathbf{y} = \phi_o(\mathbf{U}\phi_n(\mathbf{W}_n\phi_{n-1}(...\phi_1(\mathbf{W}_1\mathbf{x})))) \tag{2.3}$$

where $\mathbf{W}_i$ is the weight matrix of *i*-th hidden layer, **U** is the weight matrix connecting the last hidden layer to the output layer and ϕ_i is the activation function. Generally `sigmoid` and `tanh` activation functions are used as hidden layer activations and normalisation functions such as `softmax` is used for output layer (ϕ_o). Each layer in MLP transforms the input to a slightly more abstract representation that is useful for learning. MLP is simple yet powerful and can be a universal function approximator under certain conditions (Hornik, 1991).

2.1.1.2 Recurrent Neural Network (RNN)

Recurrent neural networks (RNN) are a family of neural networks for processing sequential data. As compared to the fixed-length feed-forward deep neural networks, RNN can readily scale to sequences of variable lengths. This is possible due to parameter sharing across different parts of the model. The same set of parameters is used at each step. This allows the model to generalise sequences of variable lengths. Since RNNs are applied to sequences, there is a concept of time or position attached to them. At each time step, an RNN calculates its output as a function of previous outputs as well its input. Previous outputs are represented by hidden state of an RNN. This hidden state act as a "lossy" summary of the task-relevant aspect of the sequence up to a certain time period.

Elman network is one of the most successful and popular architecture (Elman, 1990a). It consist of a cyclic directed graph with a single layer perception, having a feedback connection to itself as shown in Figure 2.1. Unfolding it through time results in a network similar to a multi-layer perceptron. Mathematically, hidden state is calculated as:

$$\mathbf{h}_i = \phi_h(\mathbf{W}_{hh}\mathbf{h}_{i-1} + \mathbf{W}_{hi}\mathbf{x}_i) \tag{2.4}$$

where $\mathbf{W}_{hh}$ and $\mathbf{W}_{hi}$ are weight matrices corresponding to feedback loop and input to the hidden layer, respectively and ϕ_h is the activation function such as `tanh` . $\mathbf{x}_i$ is the input at position i and $\mathbf{h}_{i-1}$ is the previous hidden state. The output at position i can be calculated as:

$$\mathbf{y}_i \sim \phi_o(\mathbf{W}_{oh}\mathbf{h}_i) \tag{2.5}$$

where $\mathbf{W}_{oh}$ is the output weight matrix between the hidden layer and the output layer and ϕ_o is the normalisation functions such as `softmax` . Usually the output activation function gives a probability distribution over the output variables. The output is then sampled and is used as an input for generating the next hidden state. RNNs have shown remarkable performance on

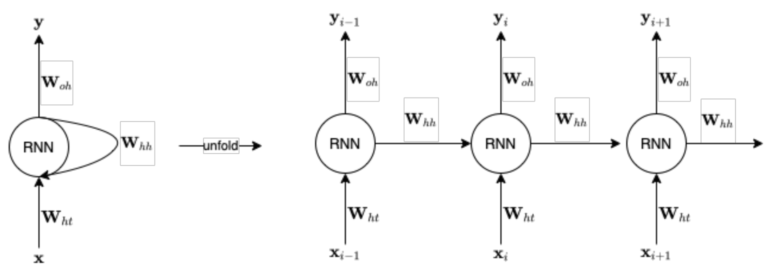


Figure 2.1: Recurrent neural network. Left side shows a folded RNN. The right side shows an unfolded version where each hidden state is shown to be a function of previous hidden state and the input at that time step.

various tasks but they suffer from vanishing and exploding gradient problem when trained using backpropogation (Hochreiter, 1991). Backpropogation requires calculating the derivative of the output with respect to inputs. The derivative is calculated using a chain rule because of the hidden layers. As the same weights are used repeatedly, they can either saturate or decay to zero resulting in a vanishing or exploding of the gradient. RNN cannot learn long-term dependencies efficiently because of this issue. Long Short-Term memory has been proposed to rectify this problem.

2.1.1.3 Long Short-Term Memory (LSTM)

LSTM addresses the vanishing and exploding gradient problems. The problem was analysed in detail by Hochreiter and Schmidhuber (1997). The authors observed that the gradient of the error function is scaled by a certain factor whenever it is propagated back through a layer of neural network. This factor is either greater or smaller than one. As a result, the gradient either grows or decays exponentially over time causing the weights to either saturate to 1 or decay to 0.

To avoid this scaling effect, the RNN was re-designed in such a way that its corresponding scaling factor was fixed to one. The new unit type called the LSTM was enriched with several gating units to enhance its learning capabilities (See Figure 2.2). Mathematically, an LSTM is defined by the following equations:

$$\mathbf{f}_i = \sigma(\mathbf{W}_f[\mathbf{h}_{i-1}, \mathbf{x}_i] + \mathbf{b}_f) \tag{2.6}$$

where $\mathbf{W}_f$ is the weight matrix, $\mathbf{b}_f$ is a bias vector , $\mathbf{h}_{i-1}$ is the previous hidden state and σ is a `sigmoid` function to convert the values between 0 and 1 . The first equation is known as the forget gate operation, where the current input and learned representation from previous inputs are concatenated and an element-wise non-linearity is applied. This generates numbers in the range [0, 1]. These numbers show, how much the corresponding component can participate in the information flow. In order to update the cell state, we also need to determine which information should be added to it. Therefore, a new candidate vector $\tilde{\mathbf{c}}_i$ is created, that will be added to the cell state:

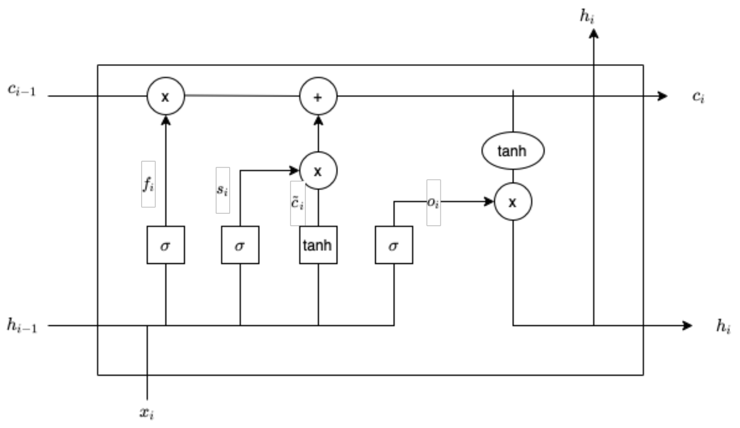


Figure 2.2: Long Short-Term Memory (LSTM). The figure shows the inner working of an LSTM cell.

$$\tilde{\mathbf{c}}_i = tanh(\mathbf{W}_c[\mathbf{h}_{i-1}, \mathbf{x}_i] + \mathbf{b}_c) \tag{2.7}$$

where $\mathbf{W}_c$ is the weight matrix, $\mathbf{b}_c$ is a bias vector, $\mathbf{h}_{i-1}$ is the previous hidden state and `tanh` is a non-linearity function applied to each value in the vector. For regulating the added information, another input similar to the forget gate but with a different set of weights is calculated as:

$$\mathbf{s}_i = \sigma(\mathbf{W}_s[\mathbf{h}_{i-1}, \mathbf{x}_i] + \mathbf{b}_i) \tag{2.8}$$

where $\mathbf{W}_s$ is the weight matrix, $\mathbf{b}_i$ is a bias vector , $\mathbf{h}_{i-1}$ is the previous hidden state and σ is a `sigmoid` function to convert the values between 0 and 1 . The Equations 2.7 and 2.8 are called the update operations. This operation selects required information from the candidates in order to add this information to the cell state. The cell state is updated as:

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{s}_i \odot \tilde{\mathbf{c}}_i \tag{2.9}$$

where $\odot$ is a point-wise multiplication. The output gate operation determines the information to use from forget and update gate. This is needed for calculating the new hidden state. The output therefore is a filtered version of cell state and is calculated as:

$$\mathbf{o}_i = \sigma(\mathbf{W}_o[\mathbf{h}_{i-1}, \mathbf{x}_i] + \mathbf{b}_o) \tag{2.10}$$

where $\mathbf{W}_o$ is the weight matrix, $\mathbf{b}_o$ is a bias vector , $\mathbf{h}_{i-1}$ is the previous hidden state and σ is a `sigmoid` function to convert the values between 0 and 1. A non-linear function `tanh` is applied to push the values between [-1, 1]. Finally the hidden state is calculated as:

$$\mathbf{h}_i = \mathbf{o}_i \odot tanh(\mathbf{c}_i) \tag{2.11}$$

Inspired by LSTM, Cho, Van Merriënboer, Gulcehre, et al. (2014) proposed a simpler version of LSTM called the Gated Recurrent Unit (GRU). GRU is faster than LSTM as it uses two gates to regulate the flow of information without having a separate memory cell.

2.1.2 Deep Learning for NLP

Contemporary approaches to NLP rely heavily on deep learning which makes it possible to build complex computer programs such as machine translation and summarization. In this section, we will review some basic concepts of NLP systems.

2.1.2.1 Word Embedding

A basic unit of language is a word. Therefore, NLP starts with ways of representing words such that they can be employed as an input for the model. These numerical representation of the words are called word embeddings. There are three ways to represent them:

Dictionary Lookup: The simplest way is to put all the words in a data structure similar to a list or array and use the index as its embedding. The issue with this approach is that, it imposes an explicit ordering on the words and may give more importance to words with higher index than the other.

One-Hot Encoding: This approach represent word as a vector of the size of vocabulary where all elements are zero except one corresponding to a particular word. This addresses the ordinal representation of the previous approach but it is computationally and memory intensive. Furthermore, it is not suitable for neural networks because of its sparse representation.

Distributed Representation: In this approach, the words are mapped to a low dimensional continuous space. Each dimension is responsible for a latent feature of the word. These embeddings are the backbone of current neural networks, as the continuous representation of the words allow gradient to pass through them. In the learned mapped space, semantically similar words are located close to each other. There embeddings can be learned as part of the training or can be pre-trained and used for transfer learning. Some of the pre-trained methods include Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013) and GloVe (Pennington, Socher, & Manning, 2014).

Before the neural revolution, term frequency-inverse document frequency (TF-IDF) scores were used for word representations (Zhang, Yoshida, & Tang, 2011). TF-IDF is a statistical measure that evaluates the relevance of a word to a document in a collection of documents. TF-IDF paves a way to associate each word in a document with a number that represents how relevant each word is in that document. Therefore, documents with similar and relevant words will have similar vectors. As these vectors can get huge, Latent Semantic Analysis (LSA) is used for dimensionality-reduction (Dumais, 2004). LSA ultimately reformulates text data in terms of r

latent features, where r is less than m, the number of terms in the data. For a detailed explanation see (Dumais, 2004).

2.1.2.2 Language Models

This section presents language models (LM), which allow modelling probability of a sequence $\mathbf{x}$. They are unconditional models and therefore cannot be used to model $p(\mathbf{y}|\mathbf{x})$. Yet they are important, as we will see in the coming sections that LMs can be modified to model the distribution for conditional sequences.

Early Statistical Language Models: LM forms the basic building block of various natural language processing (NLP) applications such as speech recognition and machine translation. It is defined as the task of predicting linguistic tokens, such as words, given preceding context (Eisenstein, 2019).

Early LMs used statistical techniques to calculate the probability of next word given its context (Charniak, 1996) . The probabilities were based upon the word occurrence or count in the corpus. These early LMs made the Markovian assumption that the probability of calculating the next token is dependent upon previous n tokens and therefore are known as n-gram LM. In n-gram LMs, the probability of sequence $p(x_1, x_2, ...x_l)$ is a product of word probabilities based upon $n - 1$ preceding words. This can be mathematically formulated as:

$$p(x_1, x_2, ...x_l) = \prod_{i=1}^{L} p(x_i|x_{i-n+1}, ..., x_{n-1}) \tag{2.12}$$

where n is the order of n-gram model and determines the number of proceeding words in the context. The estimation of word prediction probabilities is often based on maximum likelihood estimation (MLE) on large corpora. This is mathematically defined as

$$p(x_i|x_{i-n+1}, ..., x_{i-1}) = \frac{\text{count}(x_{i-n+1}, ..., x_{n-1}, x_i)}{\sum_{x'} \text{count}(x_{i-n+1}, ..., x_{i-1}, x')} \tag{2.13}$$

where $\text{count}(.)$ is the count of word sequence occurrence in a large corpus. These models although useful, have various drawbacks. First, some counts can result in a zero when encountering unseen sequences or out-of-vocabulary words. This leads to data sparsity problem. Various solution have been proposed such as smoothing techniques and back off methods but no optimal solutions exist (Kneser & Ney, 1995). Second, due to the Markov assumption, they are unable to deal with long distance dependencies. Finally, languages contain millions of words and even with a small n, the matrix of co-occurrence would be enormously large. This is called the curse of dimensionality problem.

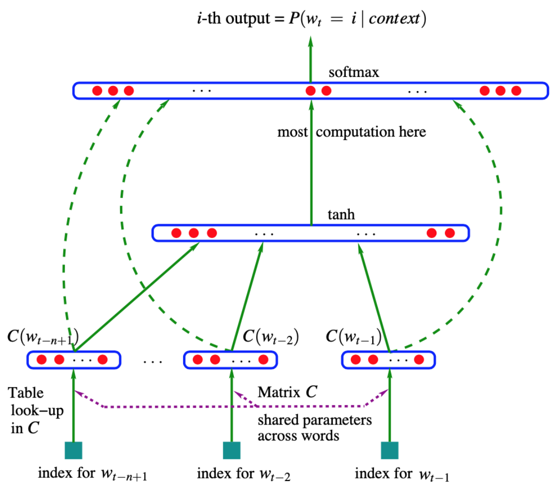


Figure 2.3: Architecture of an n-gram neural probabilistic language model. Picture courtesy: (Bengio et al., 2003)

Neural Probabilistic Language Model: Bengio et al. (2003) proposed the first Neural Probabilistic Language Model (NPLM) to address the problem of sparsity and curse of dimensionality in n-gram LMs. The core idea of NPLM is to learn the conditional probability of generating next word, given a history of $n-1$ previous words, by learning a distributed representation for words or word embeddings. Figure 2.3 shows a feed-forward neural network that is used to learn word embeddings as part of the training.

Word embeddings, as discussed above, allows word to be mapped to a real-valued vector in low-dimensional continuous space, where each dimension is responsible for a latent feature of the word. Ideally, in the mapped space, semantically similar words would be located close to each other. This allows better modeling of the relationships between words as well as the prediction of similar words. But, as the feed-forward neural network requires a fixed-size input, each word can be conditioned on fixed-size window of its context $p(x_i|x_{i-n+1}, ..., x_{i-1})$. Therefore this model cannot deal with long distance dependencies.

Recurrent Neural Network Language Model: Although NPLM addresses the sparsity and curse of dimensionality issues of the n-gram LM, it still suffers from the fixed-length context. Mikolov et al. (2010); Mikolov, Kombrink, Burget, Černockỳ, and Khudanpur (2011) address this weakness by using a recurrent neural network (RNN) instead of a feed-forward neural network (see Figure 2.4). This allows the model to process arbitrary-length context. At each step i, the hidden state of RNN is calculated as follows:

$$\mathbf{h}_i = \mathrm{RNN}(\mathbf{h}_{i-1}, \mathbf{E}[w_i]) \tag{2.14}$$

where E is the embedding table. The hidden state capture the context of previous words and the i-th word is samples as:

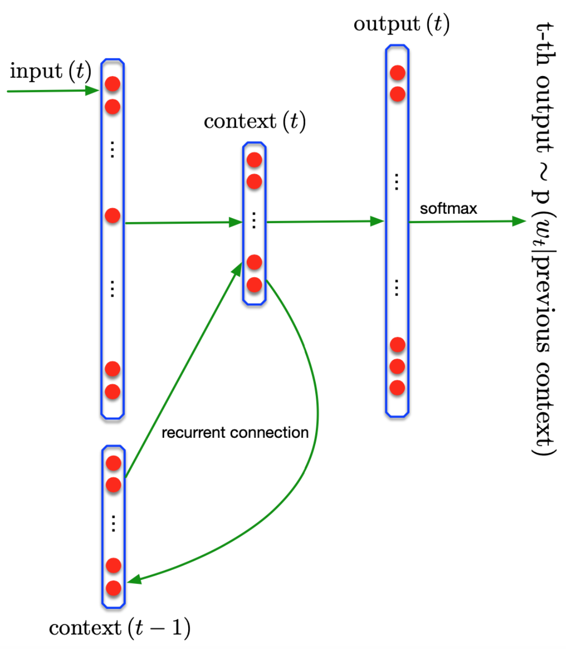


Figure 2.4: Architecture of an recurrent neural network language model. Picture courtesy: (Mikolov et al., 2010)

$$w_i \sim \mathrm{softmax}(\mathbf{W}_{oh}\mathbf{h}_t) \tag{2.15}$$

2.2 Autoregressive Models

In this section we present sequence generation (SEQ2SEQ) models. SEQ2SEQ refers to a family of models that are capable of generating an output sequence given an input sequence. They are conditional language models, that is, they generate output conditioned on some given input. In this section, we will focus on autoregressive models and their generation techniques.

2.2.1 Sequence-to-Sequence Model

Languages consist of sequences of various lengths, for-example, in machine translation the target language sequence may have a different length then the source language sequence. This poses a challenge for RNNLM as they cannot produce an output of different length than the input. Sutskever et al. (2014) proposed SEQ2SEQ models. SEQ2SEQ model consist of an encoder-decoder architecture, where an encoder consumes the entire input sequence into a vector representation of a fixed dimensionality, called the context. This context acts as a summary of the source sequence. A decoder uses this context to produce the output sequence. This process is illustrated in Figure 2.5. Each box represents an LSTM or RNN at different steps of processing. While encoding, the output of RNN is ignored, as the only interesting part is the internal memory of the encoder. Once the whole sequence has been processed by the encoder, the internal memory of the last encoder

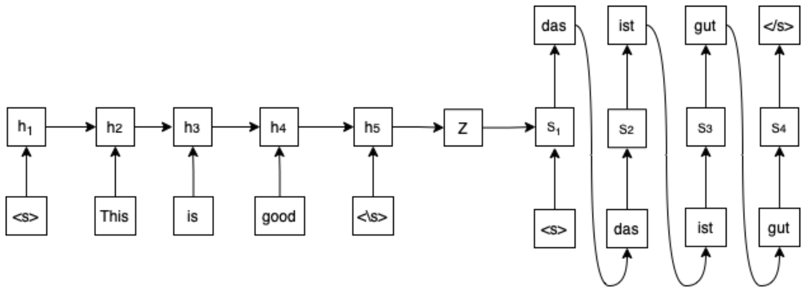


Figure 2.5: Example of Seq2Seq model for English to German translation.

RNN cell is extracted and is used to initialise the internal memory of the decoder. The first input to the decoder is a special symbol indicating the start of a sentence. This is used to generate a probability distribution over all known tokens. From this distribution, a token is sampled and fed back to the decoder to generate the probability distribution for the next token. These sampled tokens together form the translated sentence.

Encoder The encoder is a uni-directional RNN whose hidden states represent tokens of the input sequence. These representations capture information not only of the corresponding token but also other tokens in the sequence to leverage the context. The RNN runs in the left-to-right direction over the input sequence:

$$\mathbf{h}_i = \mathrm{RNN}(\mathbf{h}_{i-1}, \mathbf{E}[x_i]) \tag{2.16}$$

where E is the embedding of token x_i and $\mathbf{h}_i$ is the hidden state of the RNN. Special symbols are used which allow the model to process arbitrary length sequences. The input and output sequences are wrapped in special symbols <*s*> and <*/s*> to indicate the start and end of the sentence. The hidden state, after reading the last symbol is the fixed-length vector representation of the source sentence. Note that the input sequence is best processed in reversed order. This reversal facilitates easier correspondences between the input and output sequences due to the introduction of some short-term dependencies.

Decoder The decoder is again a uni-directional RNN which generates the token of the output sequence one-by-one from left to right, thus making these models autoregressive. In fact, decoder plays an important role in determining whether the model is autoregressive or non-autoregressive. The generation of each token y_i is conditioned on all of the previously generated tokens $\mathbf{y}_{<i}$ as well as the context. Each token y_i is drawn from a probability distribution $p(y_i|\mathbf{y}_{<i}, \mathbf{z})$, where $\mathbf{z}$ is the context vector. Using RNN this can be formulated as:

$$\mathbf{h}_i = \mathrm{RNN}(\mathbf{h}_{i-1}, \mathbf{E}[x_i], \mathbf{z})$$

$$p(y_i|\mathbf{y}_{<i}, \mathbf{z}) \sim \mathrm{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b})$$

where $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is a bias.

Training The SEQ2SEQ model is learned end-to-end based on the loss function computed via individual cross-entropy losses at each step. The typical training objective is to maximize log-likelihood on a set of training examples $D = |\mathbf{x}^k, \mathbf{y}^k|_{k=1}^K$

$$\mathcal{L}(\theta) = \arg\max_{\theta} \sum_{k=1}^{K} \log P(\mathbf{y}^k | \mathbf{x}^k; \theta) \tag{2.17}$$

As the model is autoregressive, the above objective function can be written as:

$$\mathcal{L}(\theta) = \arg\max_{\theta} \sum_{k=1}^{K} \sum_{i=1}^{M} \log P(y_i^k | \mathbf{y}_{<i}^k, \mathbf{x}^k; \theta) \tag{2.18}$$

Minimising this loss $\mathcal{L}$ is equivalent to maximising the probability of predicting a correct symbol in the output sequence at each step. The bottleneck of this model is that, the model should compress all the necessary information of source sentence into a fixed-length vector. Cho, Van Merriënboer, Bahdanau, and Bengio (2014) showed that this method performs relatively well on short sentences, but its performance deteriorates rapidly with the increase in the length of the source sentence.

2.2.2 Attentional Sequence-to-Sequence Model

One of the weaknesses of the encoder-decoder model is the use of a fixed-size context vector representation ($\mathbf{z}$ in Figure 2.5). It is too much for one context vector to carry all important information from the input sequence. Particularly for long sequences, the model is prone to forgetting the information of distant words in the sequence. This is due to the recurrent structure of the encoder, even with the use of LSTM or GRU. To address the bottleneck of fixed-vector representation in the SEQ2SEQ, Bahdanau et al. (2014) proposed an attention mechanism which dynamically attends to relevant parts of the input sequence necessary for generating the next token in the output sequence; see Figure 2.6. Moreover, attentional SEQ2SEQ model is usually equipped with a bi-directional encoder to capture context of both past and future.

Bi-directional encoder: The encoder is a bi-directional RNN, consisting of two RNNs running in the left-to-right and right-to-left directions over the input sequence:

$$\overrightarrow{\mathbf{h}_i} = \text{RNN}(\overrightarrow{\mathbf{h}}_{i-1}, \mathbf{E}[x_i])$$

$$\overleftarrow{\mathbf{h}_i} = \text{RNN}(\overleftarrow{\mathbf{h}}_{i+1}, \mathbf{E}[x_i])$$

where $\overrightarrow{\mathbf{h}_i}$ and $\overleftarrow{\mathbf{h}_i}$ are the hidden states of the forward and backward RNNs which can be based on the LSTM or GRU units. Each source token is then represented by the concatenation of the corresponding bidirectional hidden states, $\mathbf{h}_i = [\overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i}]$. The forward and backward RNNs take their

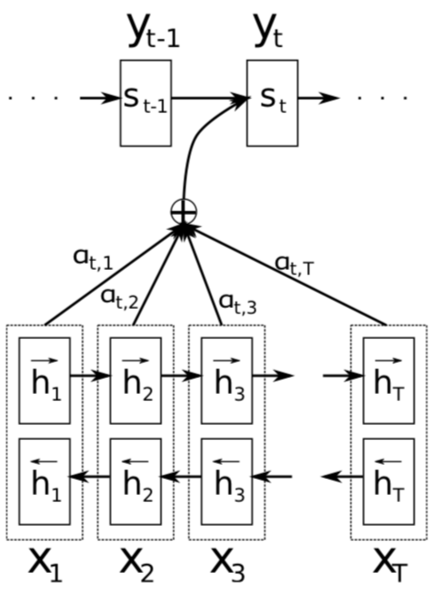

y_{t-1}
y_t
s_{t-1}
s_t
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
X_1 X_2 X_3 X_T

Figure 2.6: Encoder decoder with attention (Source (Bahdanau et al., 2014))

inputs by reading the input sequence in the forward and backward directions, respectively. This results in two hidden states for each input symbol. This bidirectional encoding scheme captures not only the position specific information, but also the richer information coming from both left and right contexts.

Attentional decoder: Bahdanau et al. (2014) also proposed the use of attention over the input sequence. The main idea is to use a dynamic mechanism for the context vector $\mathbf{z}_i$ that enables the decoder to attend to different parts of the input sequence at each step of generating the output sequence. In that sense, the model uses a dynamic context vector denoted as $\mathbf{z}_i$ for each position i in the output sequence $\mathbf{y}$. Formally, the conditional probability of $\mathbf{y}_i$ is given by:

$$p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}) = RNN(y_{i-1}, \mathbf{h}_i, \mathbf{z}_i)$$

where $\mathbf{h}_i$ is the hidden state of RNN at time-step i and is calculated as:

$$\mathbf{h}_i = f(\mathbf{h}_{i-1}, y_{i-1}, \mathbf{z}_i)$$

The context vector $\mathbf{z}_i$ is computed as weighted sum of hidden states of encoder as:

$$\mathbf{e}_{ij} = f(\mathbf{s}_{i-1}, \mathbf{h}_j)$$

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{L} exp(e_{ik})}$$

$$\mathbf{z}_i = \sum_{j=1}^{L} \alpha_{ij} \mathbf{h}_j$$

This is called an alignment model which scores how well the inputs around position j and the output at position i matches. This model is parametrised as a feed-forward neural network which is jointly trained with all the other components of the proposed system. The RNN-based attentional SEQ2SEQ models have two limitations. First, as RNN needs to maintain a hidden state that imposes sequential dependency, therefore it cannot be parallelised. Second, RNNs including LSTM are infamous for capturing long term dependencies.

2.2.3 Convolutional Sequence-to-Sequence Model

Gehring et al. (2017) proposed ConvS2S to address the parallelisation issue in RNNs. ConvS2S uses multi-layer convolution neural networks (CNN) for encoding and decoding as shown in Figure 2.7. Each convolution layer is parameterized as a large matrix $W \in \mathcal{R}^{2d \times kd}$ and $b_w \in \mathcal{R}^{2d}$ where k is the kernal width and d is the embedding size. The input to the convolution layer is the concatenation of either the input to encoder or the output of previous convolution layer. Besides convolutional encoder and decoder, Gehring et al. (2017) also used a special attention mechanism called multi-step attention. It is essentially taking global attention for every decoder layer. To further equip the model with a sense of time, positional embedding are added to the original input word embeddings. In addition, residual connections is added to enable deep networks.

The convolutional layers operate over a fixed-size window of input tokens and therefore can be operated in parallel. A multi-layer convolutional neural network creates a hierarchical representation over the input sentence where the lower layers capture nearby dependencies and higher layers capture long-term dependencies. Assume we are given a sentence with n tokens, a convolutional network with kernal size k can capture dependencies by applying $\mathcal{O}(\frac{n}{k})$ operations. The number of operations for RNN for similar size input is $\mathcal{O}(n)$.

ConvS2S are not widely used for sequence modelling and are presented here for historical context. Therefore, this model will not be presented in the rest of the thesis.

2.2.4 Self-attention Sequence-to-Sequence Model

Although convolutional neural network-based models are able to parallelise the training of SEQ2SEQ model, they still suffer from the inability to capture long-term dependencies. Vaswani et al. (2017) proposed Transformer, an architecture solely based on self-attention. The Transformer can be seen as an adaptive weighted convolutional kernel with the window-length of the sequence size. In the convolution kernel, the weights are dependent on the position and not the content. Therefore, when we slide the window through the sequence, weights are fixed. However, in the Transformer, weights are determined with respect to the representation of the current position and the representation of other positions. In other words, the idea is that the position should not be a limitation, and if two tokens are related, they should have higher weights for each other, even if they are distant.

As depicted in Figure 2.8, each layer consists of a Multi-head attention sublayer followed by a feed-forward neural network. The difference between the encoder and the decoder is two-fold:

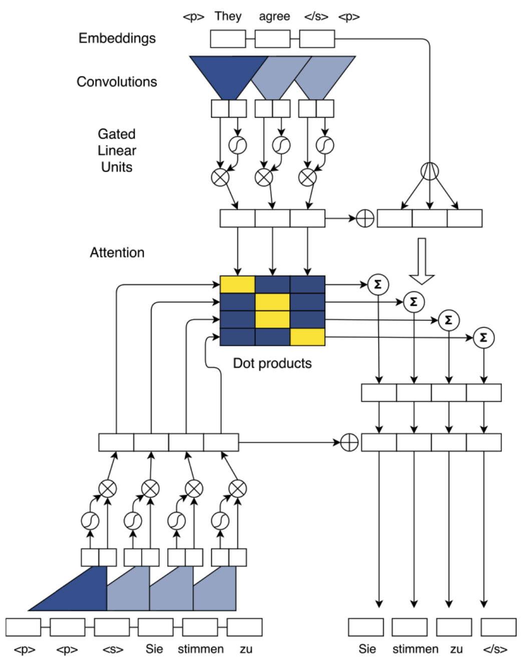


Figure 2.7: The general architecture of convulutional `seq2seq` model (Source (Gehring et al., 2017))

(1) the decoder also attends to the encoder representations; (2) the decoder does not have attention to the future.

Attention Mechanism in Transformer: Transformer is solely based on self- attention, and benefits from two extensions to the current attention mechanism: (1) scaled dot-product attention; (2) multi-head attention.

Vaswani et al. (2017) proposed a scaled version of the dot-product attention where the input to the attention mechanism consist of queries and keys with d_k dimensions and values with d_v dimensions. The scaled dot-product attention is similar to vanilla dot-product attention that is scaled by the factor of $\sqrt{d_k}$:

$$ScaledAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Qk}^T}{\sqrt{d_k}})$$

The scaling extension is proposed to address the poor performance of the vanilla dot-product attention in comparison to additive attention (Bahdanau et al., 2014) for large values of d_k. It is suspected that for large values of d_k, the vanilla dot-product grows large in magnitude and pushes the `softmax` function into regions with extremely small gradients, and scaling can fix this issue.

The Transformer also uses a novel multi-head attention mechanism with the idea of projecting the key, query and value into several representation subspaces, and jointly attend to information in these subspaces. Intuitively, each subspace may capture a different kind of information, and the result could be considered as an ensemble of attentions.

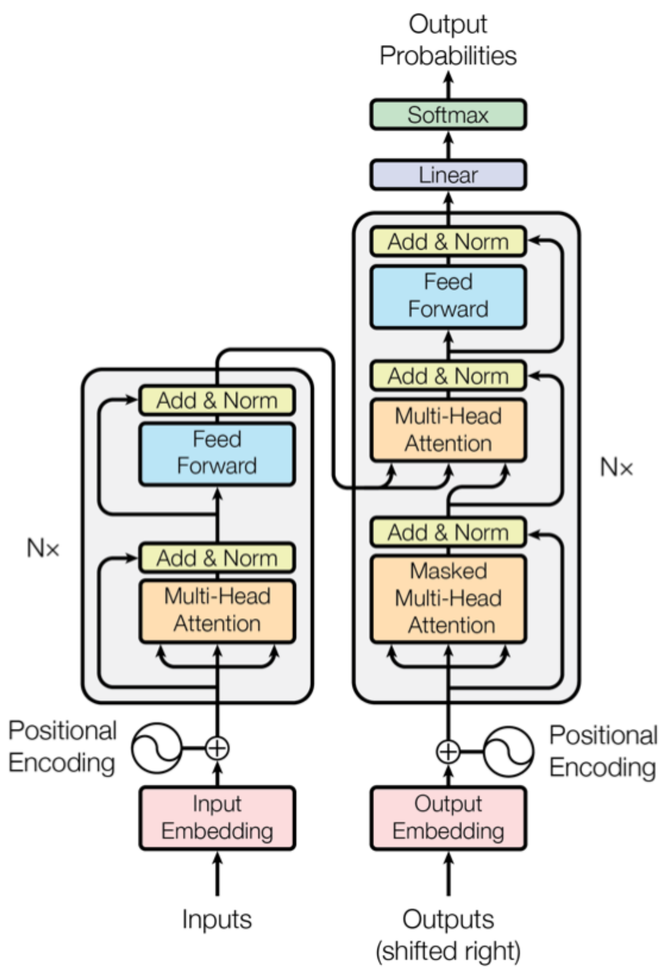


Figure 2.8: The general architecture of trasformer `seq2seq` model (Source (Vaswani et al., 2017))

$$\text{multihead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{concat}(head_1, head_2, ...head_n)\mathbf{W}^o$$

where **Q**, **K**, **V** are query, key and value vectors and $\mathbf{W}^0$ is a learnable matrix for linearly projecting attention heads outputs.

$$\text{head}_i = \text{attention}(\mathbf{Q}\mathbf{W}_i^q, \mathbf{K}\mathbf{W}_i^k, \mathbf{V}\mathbf{W}_i^v)$$

where $\mathbf{W_i}$ are learnable parameters. The Transformer applies the mentioned attention mechanism in three different ways:

1. **Self-attention in Encoder** In order to obtain the representation of the source sentence, the Transformer employs self-attention where the **Q**, **K** and **V** come from the encoder, that is the embedding of words for the first layer or embedding of the previous layer for the rest.

2. **Cross-attention from Decoder to Encoder** As the generation of the target sequence should be conditioned on the input sentence, the decoder needs to attend to the relevant part of the source sentence. Therefore, the attention mechanism is employed with queries **Q** that comes from the previous layer of decoder along with key **K** and values **V** from the encoder output.

3. **Self-attention in Decoder** In order to obtain the representation of the generated sequence so far, self-attention is applied on **Q**, **K** and **V** come from the decoder, that is embedding of generated words for the first layer or embedding of the previous layer for the rest.

Positional embedding: Unlike RNN, the Transformer does not process input tokens with respect to their temporal position. Therefore, it needs to be informed about the relative or absolute position of the tokens in the sequence in order to make use of sequence order. To encode the position, Vaswani et al. (2017) used sinusoidal functions with different frequencies to extrapolate to sequence lengths even larger than those observed during the training. Finally, the position embedding is added to the word embeddings to enrich them with positional information.

2.2.5 Generation

We will now cover autoregressive generation techniques. By adopting different generation methods, we can place restrictions or preferences on the sampling process to alter the generated samples without modifying any model weights. Even though decoding strategies do not change the values of any trainable parameter, it is a quite important component.

This section will start with common decoding methods such as greedy and beam search and afterwards move towards more advance methods. We assume that we have access to a trained model p_θ. The model has learned the distribution over token sequences by optimizing for the next token prediction.

2.2.5.1 Common Decoding Methods

The final layer of the model predicts logits $\mathbf{o}$ over the vocabulary space. The next token can be sampled from a distribution obtained by applying `softmax` function with temperature T on the logits. The probability of sampling a token at the *i*-th position is given by:

$$\mathbf{p}_i[w] \propto \frac{exp(\mathbf{o}_i[w]/T)}{\sum_{w'} exp(\mathbf{o}_i[w']/T)}$$

where $p_i[w]$ is the probability of word w at position i and $\mathbf{o}_i[w]$ is its respective logit value. T is the temperature. A low temperature would make the distribution sharper and a high value makes it softer.

Greedy Search: This is the simplest and most common decoding method where we always pick the next token with the highest probability. This is equivalent to setting temperature $T = 0$. However, it tends to create repetitions of phrases, even for well-trained models.

Beam Search: This is the most widely used method of decoding in autoregressive models. Beam search is essentially restricted breadth-first search. At each level of the search tree, beam search keeps track of k (called "beam width") best candidates and expands all the successors of these candidates in the next level. Beam search could stop expanding a node if it hits a special end-of-sentence token. However, maximization-based decoding does not guarantee high-quality generation.

Top-k sampling: Fan, Lewis, and Dauphin (2018) proposed to use top-k sampling. At each sampling step, only the top k most likely tokens are selected and the probability mass is redistributed among them. The authors also suggested to use top-k random sampling where the next token is randomly selected among the top k most likely candidates. They argued that this approach can generate more novel and less repetitive content than beam search.

Nucleus sampling: Holtzman et al. (2019) proposed to use "Top-p sampling". One drawback of top-k sampling is that the predefined number k does not take into consideration how skewed the probability distribution might be. The nucleus sampling selects the smallest set of top candidates with the cumulative probability exceeding a threshold and then the distribution is rescaled among selected candidates. Both top-k and nucleus sampling have less repetitions with a proper set of hyperparameters.

Penalized sampling: To avoid the common failure case of generating duplicate sub-strings, Keskar, McCann, Varshney, Xiong, and Socher (2019) proposed a new sampling method to penalize repetitions by discounting the scores of previously generated tokens. The probability distribution for the next token with repetition penalty is defined as:

$$\mathbf{p}_i[w] = \frac{exp(\mathbf{o}_i[w]/T(\mathbb{1} \in \mathbf{g}))}{\sum_{w'} exp(\mathbf{o}_i[w']/T(\mathbb{1} \in \mathbf{g}))}$$

where **g** contains a set of previously generated tokens, $\mathbb{1}(.)$ is an identity function which is equal to θ when true otherwise it is 0. The value $\theta = 0.2$ has been found to yield a good balance between less repetition and truthful generation.

2.2.5.2 Guided Decoding

The above methods sample tokens according to the predicted probability. The predicted probability can be modified to inject our preferences. This can be done by guiding the candidate ranking score. A candidate ranking function determines the score of a token at a particular time-step. The simplest function is the output distribution generated by the `softmax` function. To inject our preferences, the ranking score for token selection at each decoding step becomes a combination of log-likelihood and a set of desired features. The features are designed to quantify our preferences using heuristics.

Ghazvininejad, Shi, Priyadarshi, and Knight (2017) built a system called "Hafez" for generating poetry in a desired style. This is done by adjusting sampling weights in beam search at each decoding steps. The likelihood of sampling for the next token y_{i+1} at step i is augmented by a scoring function:

$$\text{score}(y_{i+1}, b_i) = \text{score}(b_i) + \log p(y_{i+1}) + \sum_j a_j f_j(y_{i+1}, \mathbf{y}_{<i})$$

where $\log p(y_{i+1})$ is the log-likelihood predicted by the model, $\text{score}(b_i)$ is the accumulated score of the already-generated words in the current beam state b_i and $\sum_j a_j f_j(y_{i+1}, \mathbf{y}_{<i})$ can incorporate many different features for steering the style of the output. A set of feature functions $f_j(.)$ define the preferences and the associated weights a_j. These features work like "control knobs" that can be easily customized at decoding time. Features can measure a variety of attributes and can be easily combined; for example, features can determine if the output token exists in a bag of desired or banned words. It may also indicate certain sentiments or repeated token (and thus f_j needs to take the history as input too). Features can also include length of a word if longer or shorter words are in particular preferred. Baheti, Ritter, Li, and Dolan (2018) manually designed features for ranking and altered the sampling distribution by appending the similarity scores between topic distribution or embeddings of the context and the completion.

Holtzman et al. (2018) adopted a set of learned discriminators, each specializing in a different principle of communication guided by Griceś maxims: quality, quantity, relation and manner (Frederking, 1996). The discriminators learn to encode these desired principles by measuring repetition, entailment, relevance, and lexical diversity, respectively. Given some ground truth completion, all the discriminator models are trained to minimize the ranking log-likelihood, $\log \sigma(f_i(\mathbf{y}_g) - f_i(\mathbf{y}))$ because the gold continuation $\mathbf{y}_g$ is expected to obtain a higher score than the generated one $\mathbf{y}$. The discriminator predicts a label for each token instead of for the entire sequence. The discriminator log probablity is added to the score to guide sampling towards the human-written style.

Meister, Vieira, and Cotterell (2020) studied beam search in a regularized decoding framework:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\mathcal{Y}}(\log p_\theta(\mathbf{y}|\mathbf{x}) - \lambda\mathcal{R}(\mathbf{y})) \tag{2.19}$$

Since we expect maximum probability to have minimum surprise, the "surprisal" of a model at step i can be defined as follows:

$$\mu_t(y) = \begin{cases} 0 & \text{if i=0; where index 0 is a special BOS token} \\ -\log p_\theta(y|\mathbf{y}_{<i}, \mathbf{x}) & \text{otherwise} \end{cases}$$

The first term in Equation 2.19 is the maximum a posteriori (MAP) part. It demands sequences with maximum probability given the context, while the regularizer (second term) introduces other constraints. It is possible that a global optimal strategy may need to have a high-surprisal step occasionally so that it can shorten the output length or produce more low-surprisal steps afterwards.

Beam search has gone through the test of time in the field of NLP. The question is: If we want to model beam search as exact search in a regularized decoding framework, how should $\mathcal{R}(\mathbf{y})$ be modeled? The authors proposed a connection between beam search and the uniform information density (UID) hypothesis which states that, subject to the constraints of the grammar, humans prefer sentences that distribute information (in the sense of information theory) equally across the linguistic signal, e.g., a sentence (Levy & Jaeger, 2007). In other words, it hypothesizes that humans prefer text with evenly distributed surprisal. Popular decoding methods like top-k sampling or nucleus sampling actually filter out high-surprisal options, thus implicitly encouraging the UID property in output sequences. The paper experimented with several forms of regularizers and found that a regularized beam search greatly helps when beam size increases. Guided decoding essentially runs a more expensive beam search where the sampling probability distribution is altered by side information about human preferences.

2.2.5.3 Trainable Decoding

Given a trained model, Gu, Cho, and Li (2017) proposed a trainable greedy decoding algorithm to maximize an arbitrary objective for sampling sequences. The idea is based on the noisy, parallel approximate decoding (NPAD). NPAD injects unstructured noise into the model hidden states and runs noisy decoding multiple times in parallel to avoid potential degradation. To take a step further, trainable greedy decoding replaces the unstructured noise with a learnable random variable, predicted by a reinforcement learning (RL) agent that takes the previous hidden state, the previous decoded token and the context as input. In other words, the decoding algorithm learns a RL actor to manipulate the model hidden states for better outcomes.

Grover et al. (2019) trained a binary classifier to distinguish samples from data distribution and samples from the generative model. This classifier is used to estimate importance weights for constructing a new un-normalized distribution. The proposed strategy is called likelihood-free importance weighting (LFIW).

Let p be the real data distribution and p_θ be a learned generative model. A classical approach for evaluating the expectation of a given function f under p using samples from p_θ is to use importance sampling as:

$$\mathbb{E}_{\mathbf{y}\sim p}[f(\mathbf{y})] = \mathbb{E}_{\mathbf{y}\sim p_\theta}[\frac{p(\mathbf{y})}{p_\theta(\mathbf{y})}(f(\mathbf{y}))] \approx \frac{1}{N}\sum_{1}^{N} w(\mathbf{y}_i)f(\mathbf{y}_i)$$

However, $p(\mathbf{y})$ can only be estimated via finite datasets. Let $c_\phi : \mathcal{Z} \to [0,1]$ be a probabilistic binary classifier for predicting whether a sample $\mathbf{y}$ is from the true data distribution. The joint distribution over $\mathcal{Y} \times \mathcal{Z}$ is denoted as $q(\mathbf{y}, z)$

$$q(\mathbf{y}|z) = \begin{cases} p_\theta(\mathbf{y}) & \text{if y=0; predicted to be generated data} \\ p(\mathbf{y}) & \text{otherwise; from the true data distribution} \end{cases}$$

if c_ϕ is the bayes optimal classifier (Opper & Haussler, 1991), the importance weight can be estimated by:

$$w_\phi(\mathbf{y}) = \gamma\frac{c_\phi(\mathbf{y})}{1 - c_\phi(\mathbf{y})}$$

where γ is a fixed ratio. Since we cannot learn a perfect optimal classifier, the importance weight would be an estimation. A couple of practical tricks such as self-normalization, flattening and clipping, can be applied to offset cases when the classifier exploits artifacts in the generated samples to make very confident predictions. To sample from an importance re-sampled generative model, the authors suggested to use Sampling-Importance-Resampling (McAllister & Ianelli, 1997).

Deng, Bakhtin, Ott, Szlam, and Ranzato (2020) proposed to use energy based models (EBM) (LeCun & Huang, 2005). EBM steer the model in the residual space, $p_\theta(\mathbf{y}) \propto p_M(\mathbf{y})exp(\mathbb{E}_\theta(x))$ where p_M is the joint model and E_θ is the residual energy function to be learned. If we know the partition function $\mathbf{Z}$, we can model the generative process easily. The goal is to learn the parameters of the energy function E_θ such that the joint model p_θ gets closer to the desired data distribution. The residual energy function is trained by noise contrastive estimation (Gutmann & Hyvärinen, 2010). However, the partition function is intractable in practice. The paper proposed a simple way to first sample from the original model and then to resample from it according to the energy function. This is unfortunately quite expensive.

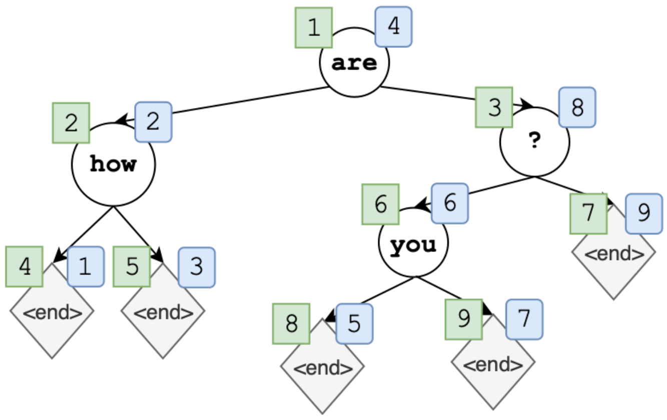


Figure 2.9: The example of a binary tree where the model first generate the word "are" and then recursively generates words left and right. (Source (Welleck et al., 2019))

2.2.5.4 Other Generation Orders

Welleck et al. (2019) proposed a framework for training models of text generation that operates in a non-monotonic orders. Unlike conventional approaches with a fixed generation order, often from left-to-right (or right-to-left), the authors build a sequence generator that generates tokens in an order automatically determined by the sequence generator. The model directly learns good orders, without any extra annotation nor supervision of what might be a good order.

Their framework operates by generating a word at an arbitrary position, then recursively generating words to its left and words to its right, yielding a binary tree (shown in Figure 2.9). The generation process is viewed as deterministically navigating a state space $\mathcal{S} = \tilde{\mathbf{V}}^*$ where a state $\mathbf{s} \in \mathcal{S}$ corresponds to a sequence of tokens from $\tilde{\mathbf{V}}^*$. A top-down traversal of a binary tree produces the sequence $\mathbf{s}$. The generation starts with an empty sentence $\mathbf{s}_0$. An action a is an element of $\tilde{\mathbf{V}}$ which is deterministically appended to the state. Terminal states are those for which all sub-trees have been end.

A policy π is a stochastic mapping from states to actions, and is denoted as $\pi(a|\mathbf{s})$, the probability of action a given the state $\mathbf{s}$. A policy's behavior determines which words should appear before and after the token of the parent node. Typically there are many unique binary trees with an in-order traversal. Each of these trees has a different level-order traversal, thus the policy is capable of choosing from many different generation orders, rather than a single predefined order.

Learning is framed as an imitation learning problem, where the model learns to imitate the actions produced by the oracle policy. The oracle policy is designed such that it randomly picks a word w and then recursively generates words to the left and right resulting in a tree. The model policy starts imitating the oracle and then move towards reinforcing its own preferences.

2.2.6 Final Words on Autoregressive models

This section presented various models and autoregressive generation techniques. Although they have shown significant improvements, other than beam search, none of them have been able to become the standard decoding method. The reason is two fold; first they require complex changes in the generation, secondly the gains are marginal. Yet these methods have improved our understanding of the generation process.

Autoregressive models are the standard models that are used in the sequence generation tasks. They have been applied towards machine translation, summarization, story generation and various other sequence generation tasks. They have shown state-of-the-art results according to the criteria used for evaluating different tasks (for example BLEU for machine translation and ROGUE for summarization). The work presented in this section has been applied towards text generation in general without regards to the applications. The methods presented in this section have resulted in various improvements such as diverse text generation, better model perplexity and better task specific metric.

All the mentioned methods, including greedy and beam search, show degraded performance on long sequences. These methods find it difficult to maintain the coherency when generating long sequences such as stories and summaries. Further, all of them suffer from the label bias as they do not allow modifying an already generated sequence. In Chapter 3, we will present a method to address the second issue. Our method will be able to revisit and revise its generated text. In Chapter 4, we will present a model for long text generation along with its generation techniques.

2.3 Non-autoregressive Models

Autoregressive models factorize the joint probability of the output sequence $\mathbf{y}$ given the input sequence $\mathbf{x}$, as the product of probabilities over the next token in the sequence given the input sequence and previously generated tokens:

$$P(\mathbf{y}|\mathbf{x};\theta) = \prod_{t=1}^{M} P(y_i|\mathbf{y}_{<i},\mathbf{x};\theta) \tag{2.20}$$

where $\mathbf{y}_{<i}$ indicates the partial sequences and θ is a set of trainable parameters. Non-autoregressive models attempt to model the joint distribution $P_\theta(\mathbf{y}|\mathbf{x})$ directly by breaking the probabilistic factorization. Each prediction is modeled as a product of probabilities, independent of the decoding history during generation:

$$P(\mathbf{y}|\mathbf{x};\theta) = P(M|\mathbf{x}) \prod_{i=1}^{M} P(y_i|\mathbf{x}, M;\theta) \tag{2.21}$$

where $P(M|\mathbf{x})$ indicates an auxiliary length predictor, which is used to determine the translation length before translating the sentence. In this section, we will review models for parameterising the probability distribution given in Equation 2.21. The section will start by presenting the basic non-autoregressive model built on a Transformer model. All non-autoregressive models have been built by enhancing some component of the basic non-autoregressive model. We have divided these enhancements as internal, i.e. improving an internal component of the model such as attention mechanism and external, i.e improving inputs of the model. The section ends with a discussion on generation techniques.

2.3.1 Basic Non-autoregressive Model

Gu, Bradbury, et al. (2017) proposed a non-autoregressive model by removing the autoregressive connection in the `seq2seq` Transformer model therefore bringing complete conditional independence in the output variable. This conditional independence results in a huge drop in the output quality. The authors of the paper suggested to use source sentence as the decoder input, with the hope that the source sentence would be able to provide some light-weight dependency information lost due to conditional independence. The architecture of the basic non-autoregressive is shown in Figure 2.10. It shows various modifications:

Encoder: The encoder is similar to the autoregressive Transformer encoder with multi head-self attention and feed forward layers. The encoder reads in an input sequence and then adds position encoding to the word embeddings. The resulted sequence pass through attention layers and feed-forward layers. As the complete input sequence is available, the whole process runs in parallel.

Decoder: Multiple changes are made in the decoder to allow it to generate sequence non-autoregressively.

1. **Decoder Input:** The decoder needs to know the length of the output sequence before starting the decoding. This is in contrast to autoregressive models where the length is modelled implicitly. Autoregressive models predict a special end of sentence symbol to indicate termination of the decoding process. This is not the case with non-autoregressive models as they require to model the length explicitly. There are various options: (1) First another classifier can be added which models the length of the output sequence conditioned on the input sequence. Using the predicted length, tokens from the source sequence are copied such that at each position i in the decoder input, source sequence at index $\text{round}((N*i)/M)$ is copied where N is the source sequence length and M is the target sequence length. The authors of the paper also suggested to use fertility of the word. (2) Instead of the classifier predicting the length, the authors suggested to use the outputs of the encoder to predict fertility. The source word is copied to build the decoder input based upon its fertility. The fertility unit is trained end-to-end with the rest of the model.

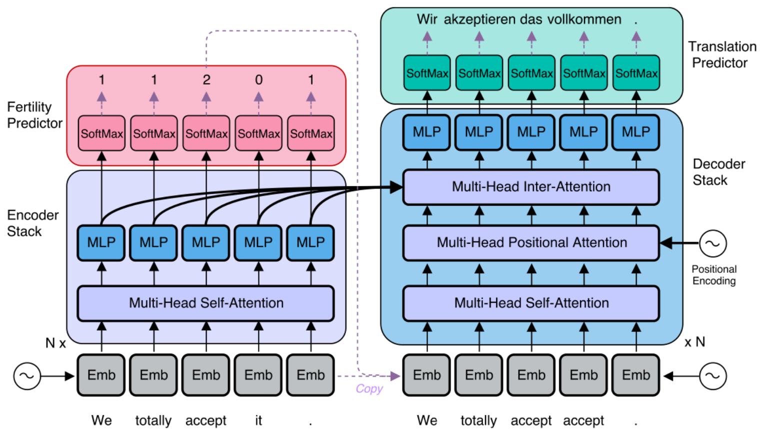

Wir akzeptieren das vollkommen .
Translation Predictor
SoftMax
Fertility Predictor
1 1 2 0 1
Decoder Stack
MLP
Multi-Head Inter-Attention
Encoder Stack
Multi-Head Positional Attention
Positional Encoding
Multi-Head Self-Attention
N x
x N
Emb
Copy
We totally accept it .
We totally accept accept .

Figure 2.10: Basic Non-autoregressive model architecture. Picture courtesy: (Gu, Bradbury, et al., 2017)

2. **Non-causal self-attention:** In an autoregressive decoder, the predictions at position i can depend only on the known outputs before positions i. This is done by introducing a *causal mask*. This mask prevents the model from attending to subsequent positions by setting the mask value to $-\infty$ for the values corresponding to the forbidden states in the `softmax` layer of the dot-product attention modules. But In non-autoregressive decoding, there is no need to prevent the earlier decoding steps from accessing information from later steps in non-autoregressive decoder. This is done by removing the causal mask in the self-attention module.

3. **Positional attention:** In order to incorporate positions, an additional attention module is incorporated in the decoder. This module helps in explicitly modeling dependencies which are lost because of the non-autoregressive nature of the decoder. Formally, we define positional attention as $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Qk}^T}{\sqrt{d_k}})$ with both query and key as position embeddings. The authors hypothesize that this additional information improves the decoder's ability to perform local reordering.

The model is trained end-to-end by maximising the log-likelihood of data under the model. This basic model was the first step towards non-autoregressive models. It showed promise but lower performance than the autoregressive models but was able to decode much faster. Later works built on this model by providing enhancements and bringing it closer to autoregressive models.

2.3.2 External Enhancements

Work on external enhancements consist of improving the decoder input so that it can provide information about the dependencies in the output variable. As the decoder input can inject missing information, it is important and therefore many studies have been done on this topic. (Guo et al., 2019) suggested two methods to improve decoder input. The first one directly leverages a phrase

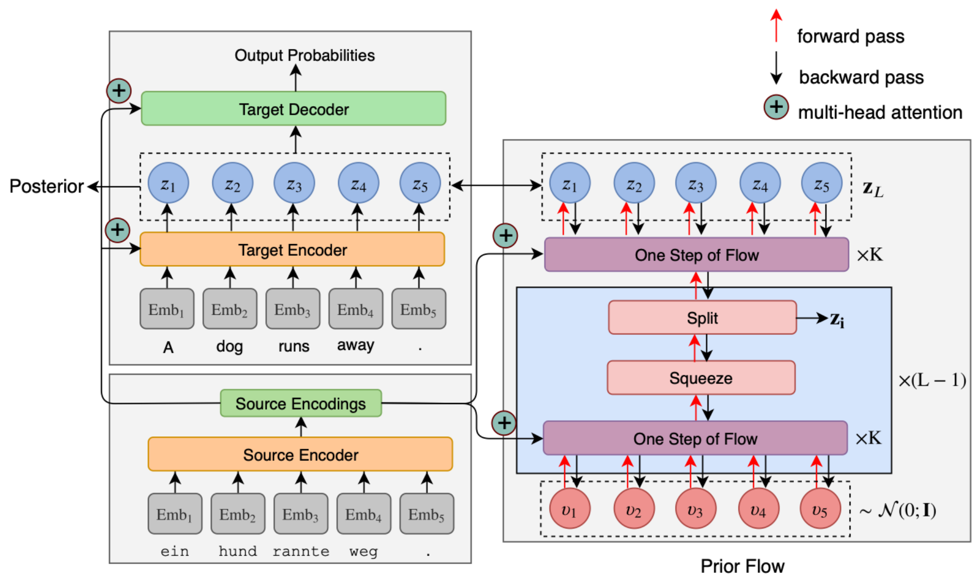


Figure 2.11: Flowseq model for learning prior. Picture courtesy: (X. Ma et al., 2019)

table generated by conventional statistical machine translation approaches to translate input tokens to output tokens. This translation is then fed into the decoder as inputs. The second method transforms input-side word embeddings to output-side word embeddings by learning a projection matrix $\mathbf{W}$. In order to learn projection matrix $\mathbf{W}$ with the rest of the model, the authors regularize the learning with sentence-level alignment and word-level adversary learning.

Ran, Lin, Li, and Zhou (2019) proposed reordering input side information according to the output side. The authors introduced a light-weight autoregressive model that was trained to reorder the input sequence. The authors found that reordering reduces the decoding space by restricting the model to outputs satisfying the constraints of reordered input. It allowed the decoder to capture the dependencies among target words and choose words belonging to the correct translation.

Y. Bao et al. (2019) suggested to model positions explicitly. They argue that modelling positions can inject reordering information back into the model and can lead to better output quality. The authors introduced a position prediction module, a light-weight autoregressive model that predicts permutations for reordering the decoder input. However their approach was not successful and did not provide improvements in the output.

X. Ma et al. (2019) proposed generative flow for learning a distribution of latent variables. The output from this distribution is then used as decoder input. Generative flow is a statistical technique that allows learning a complex distribution from a simple distribution (Rezende & Mohamed, 2015). At training time, the authors used the output sequence to generate latent variable $\mathbf{z}$. This is done by having another output sequence encoder. The output from this encoder are used to train a generative flow layer. A generative flow consist of operations such as split and squeeze (see Figure 2.11) which is applied on simple normal distribution. The model is trained by reducing the evidence lower bound. Among all the work on enhancing inputs to the decoder, flowseq has performed exceptionally well and is considered state of the art.

2.3.3 Internal Enhancements

Now we look at all other enhancements that either improve the training procedures or add new layer inside the transformer. Libovický and Helcl (2018) proposed to formulate the sequence generation as a sequence labeling problem. As the output can be of different length than the input, the authors suggested to project the encoder output into k times longer sequences. These projected vectors pass through the decoder and are labelled with the output or as null. For learning, the authors used CTC loss (Graves, Fernández, Gomez, & Schmidhuber, 2006), which employs dynamic programming to compute the negative log-likelihood of the output sequence, summed over all the combinations. Shao, Zhang, Feng, Meng, and Zhou (2020) proposed to minimise the n-grams difference between the output and the reference. They argued that the word level cross entropy loss cannot model the output side sequential dependency. As the output of non-autoregressive model may not be aligned with the reference, optimising the n-gram is a better method of training. Optimising for n-gram is similar to improving BLEU score therefore their model performed well.

Another line of work used autoregressive model to enhance various components of non-autoregressive model. Guo et al. (2020) used curriculum learning by starting with an autoregressive model and then gradually removing the dependencies in the decoder. They achieved this by introducing an attention mask that allowed the model to progressively look into later outputs. Wei, Wang, Zhou, Lin, and Sun (2019) proposed an imitation learning framework to imitate autoregressive decoder attention. Z. Li et al. (2019) suggested similar idea where they employed an autoregressive teacher and used regularization techniques to reduce the discrepancy in outputs of self-attention and cross-attention modules of autoregressive and non-autoregressive models.

Y. Wang et al. (2019) suggested to use two regularizations. First, to make the hidden states more distinguishable, they regularize the similarity between consecutive hidden states based on the corresponding target tokens. Second, to force the hidden states to contain all the information in the input sentence, they leverage the dual nature of translation tasks (e.g., English to German and German to English) and minimize a backward reconstruction error to ensure that the hidden states of the non-autoregressive decoder are able to recover the input side sequence.

Sun et al. (2019) incorporated a conditional random field (CRF) as an output of the decoder (see Figure 2.12). CRF is a framework for building probabilistic models to segment and label sequence data. The CRF module was trained jointly with the rest of the model. To overcome the challenge of huge transition tables because of the vocabulary, the authors suggested to learn low dimensional matrix as a transition functions. With CRF, exact decoding is possible by using techniques such as dynamic programming.

2.3.4 Generation

Almost all non-autoregressive models use the following generation strategies:

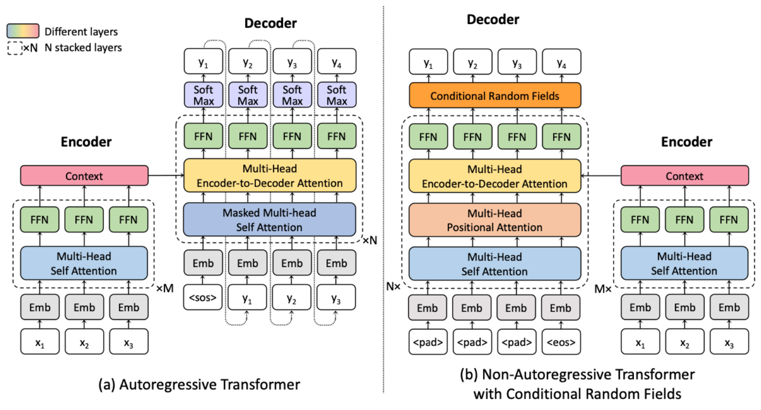

Different layers
×N N stacked layers
Decoder
Encoder
Context
Multi-Head Encoder-to-Decoder Attention
Masked Multi-head Self Attention
Multi-Head Self Attention
Soft Max
FFN
Emb
<sos>
(a) Autoregressive Transformer
Conditional Random Fields
Multi-Head Positional Attention
<pad>
<eos>
(b) Non-Autoregressive Transformer with Conditional Random Fields

Figure 2.12: Illustration of transformer based CRF model. Picture courtesy: (Sun et al., 2019)

Argmax decoding: This is the simplest decoding method where token corresponding to the highest probability is selected at each position. This is similar to greedy decoding in autoregressive models.

Average decoding: Instead of selecting a token with the highest probability, a token corresponding to the average of the probability distribution is selected at a certain position.

Noisy parallel decoding (NPD) In this method, multiple translations are generated corresponding to different lengths. These translations are then ranked by the autoregressive model. The translation with the highest score is selected as the output of the model. NPD is a stochastic search method, and it also increases the computational resources required linearly by the sample size. However, because all the search samples can be computed and scored entirely independently, the process only doubles the latency compared to computing a single translation if sufficient parallelism is available.

Other generation methods have also been proposed. X. Li, Meng, Yuan, Wu, and Li (2020) proposed "look around decoding". In this approach the model predicts the neighbour tokens in order to predict the current token. This approach cannot be generalized to other non-autoregressive models therefore it has not been used in other works.

2.3.5 Final Words on Non-autoregressive Model

We looked in to the literature of non-autoregressive models, starting with the basic model along with its various improvements. Non-autoregressive models have been applied towards machine translation as well as towards semantic parsing. This is because these models still lag behind their autoregressive counterparts when evaluated with BLEU score. Therefore their application towards long text generation is limited in the literature. The majority of work on non-autoregressive models have used benchmark datasets WMT14 English-German (En-De) and WMT16 English-Romanian (En-Ro) from experimentation.

Our work in Chapter 5, can be categorised in external enhancements. We introduce an informative prior to inject information about the dependencies in the out variables. As opposed to other similar works, our method is simple and does not require complex training procedures.

2.4 Semi-autoregressive Models

The complexity of decoding for autoregressive models is $\mathcal{O}(n)$ where n is the length of the output sequence, whereas the complexity of decoding for non-autoregressive models is $\mathcal{O}(1)$. All other approaches whose complexity lies between $\mathcal{O}(n)$ and $\mathcal{O}(1)$ are categorised as semi-autoregressive approaches. Semi-autoregressive models provide the best of both worlds by decoding faster compared to autoregressive models with some cost to the output quality. There are two ways in which an output sequence is generated semi-autoregressively. First is the iterative refinement, where the model produce the whole sequence in parallel and then refine it. Second, the model produces chunks such as phrases autoregressively while the content inside the chunk is produced in parallel. We cover the former approach below as the second approach has not attracted much attention of the research community. There is, however, work by C. Wang et al. (2018) which kept the autoregressive property globally by generating chunk of words autoregressively but locally multiple successive words are produced in parallel.

2.4.1 Iterative refinement

Ghazvininejad et al. (2020) proposed to use mask-predict. Mask-predict repeatedly predicts the entire target sequence in parallel while conditioned on the most confident word predictions from the previous iteration (Ghazvininejad, Levy, Liu, & Zettlemoyer, 2019b). The underlying model, a conditional masked language model, is trained by masking part of the output sequence and predicting the missing tokens. During training, all observed (un-masked) tokens come from the ground truth data. However, at inference time, the observed tokens are high-confidence model predictions, creating a discrepancy between training and inference. To solve this problem, the authors introduced a new training procedure. They build training examples by starting with the gold output sequence and masking a subset of its tokens, just like the original training process. They used the resulting dataset for training. This reduced the discrepency in the training and inference procedure leading to better output sequences.

Kasai, Cross, Ghazvininejad, and Gu (2020b) proposed an attention masking based model. The model learned to predict all outputs in parallel by using a different context for each token. This is done by learning an attention mask. The mask allows to use different target context for different tokens. Each word is predicted by attending to the words that the model is more confident about. Their decoding algorithm allows for predicting all tokens with different contexts in each iteration and terminates when the output prediction converges. Lee, Mansimov, and Cho (2018) proposed a model that can be viewed as both a latent variable model and a conditional denoising autoencoder. The authors suggested a learning algorithm that is a hybrid of lowerbound maximization and reconstruction error minimization. The inference strategy is iterative. It starts with the model

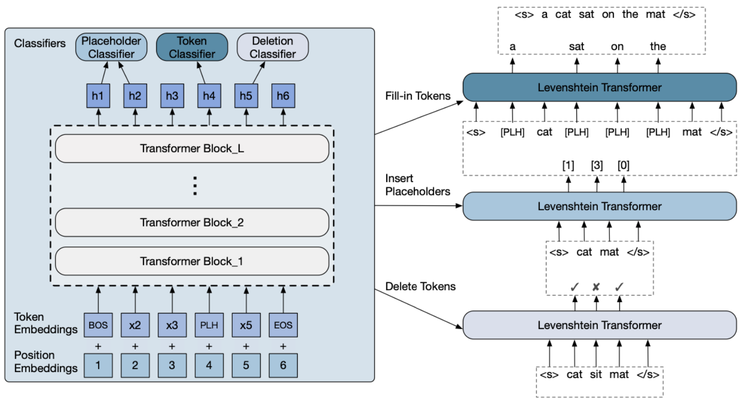


Figure 2.13: Illustration of Levenstein Transformer. Picture courtesy: (Gu et al., 2019)

predicting an output sequence conditioned on its length. The subsequent iterations improve this output.

Gu et al. (2019) proposed Levenstein Transformer for addressing the lack of dynamic length changes in the current models. They framed sequence generation as a Markov decision process. Their model consists of two policies, *insertion* and *deletion*. The policies are implemented as classifiers by using the hidden states of the Transformer model (see Figure 2.13). The model iteratively refines the sequence by applying insertion and deletion operators until convergence. Although, the operators are applied autoregressively, the whole operation is applied in parallel. Once the model has decided which sentences to delete, all the sentences will be deleted in parallel. The insert classifier is made up of placeholder prediction, which predicts the number of tokens to be inserted in two consecutive words, and token prediction, which fills the placeholders. The model is trained using imitation learning by following an expert policy. The authors used Levenstein distance to get the expert actions. Xu and Carpuat (2020) improved Levenstein Transformer by introducing a reposition classifier instead of a deletion classifier. The reposition classifier is able to apply a permutation that is reorder words to different places. This operator can also delete words. The authors modified the training procedure but still used the same expert policy with minor changes.

2.4.2 Final Words on Semi-autoregressive Model

Semi-autoregressive models have not been explored much compared to non-autoregressive models. They are unique in the sense, that they allow user the flexibility to pick either time or quality as a preference. If the application requires quick outputs, the user can reduce the iterations but if quality is of concern the user can increase the number of iterations to the desired level. Semi-autoregressive models have mostly been applied towards machine translation. Their application towards other areas such as text generation has also been explored but is limited to few works. The majority of work on these models have used benchmark datasets WMT14 English-German (En-De) and WMT16 English-Romanian (En-Ro) from experimentation.

We employ semi-autoregressive model based on Levenstein Transformer, in Chapter 4, towards long sequence generation. Insertion and deletion operators allow dynamic length changes.

We extend this concept towards document generation and show the utility of semi-autoregressive models towards long sequences generation.

2.5 Locally vs Globally Normalised Models

We present another view of looking at autoregressive models as locally normalised models. The content in this section has been added for completeness. This view of autoregressive models clearly shows the reason for their major deficiencies, i.e. exposure and label bias. This section can be skipped without loss to understanding of the upcoming chapters.

Under a locally normalised model, the probability of the output sequence $\mathbf{y}$ given the input sequence $\mathbf{x}$ is formulated as:

$$p_{\mathcal{M}_L}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_t|\mathbf{y}_{<i}, \mathbf{x}) = \prod_{i=1}^{N} \frac{\phi_i(y_i, \mathbf{y}_{<i}, \mathbf{x})}{Z_{L,i}(\mathbf{y}_{<i}, \mathbf{x})}$$

where $\mathcal{M}_L$ is a locally normalised model, $\phi(y_i, \mathbf{y}_{<i}, \mathbf{x})$ is a scaler quantity for predicting the token y at index i. The probability is conditioned on the input sentence $\mathbf{x}$ and the history of generated words $\mathbf{y}_{<i}$. The term $Z_{L,i}(\mathbf{y}_{<i}, \mathbf{x})$ is a local normaliser. An autoregressive decoder utilise the input sequence, which is encoded by the encoder and the contextual representation which summarises the predicted output history, represented by the hidden state $\mathbf{h}$ to calculate the logit vectors. The vectors are normalised by the `softmax` function. As the normaliser is easy to compute, likelihood maximisation based training schemes such as *cross-entropy* loss has become the de-facto training standard. At training time teacher forcing is employed where ground-truth sequences are used to train the model but at the decoding time the model has to rely on its own outputs. As the training is not *search-aware*, it suffers from *exposure bias*. Exposure bias can be mitigated by training the model under the same conditions it will encounter at decoding time.

Locally normalised models also have label bias. Incorrectly generated label at position i will be propagated throughout the generation, influencing future tokens with incorrect information. Globally normalised models can help mitigate label bias problem. Formally, under a globally normalised model $\mathcal{M}_G$, the probability of output sequence $\mathbf{y}$ given and input sequence $\mathbf{x}$ is given by:

$$p_{\mathcal{M}_G}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} \frac{\phi_i(y_i, \mathbf{y}_{/i}, \mathbf{x})}{Z_G(\mathbf{x})}$$

where $\phi_i(y_i, \mathbf{y}_{/i}, \mathbf{x})$ is a scalar term and $\mathbf{y}_{/i}$ denotes all tokens of $\mathbf{y}$ except at position i. The term $Z_G(\mathbf{x})$ is formulated as:

$$Z_G(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{Y}} \prod_{i=1}^{N} s(y_i, \mathbf{y}_{/i}, \mathbf{x})$$

As the normaliser requires summation over all input sequences, $Z_G(\mathbf{x})$ is intractable to compute. The search space of most problems of interest is larger therefore exact likelihood maximisation based approaches are not suitable for these methods.

2.6 Summary

In this chapter, we covered the foundations and prior works related to this thesis. We overviewed deep learning fundamentals and its usage in NLP. Then, we reviewed autoregressive, semi-autoregressive and non-autoregressive models along with their generation techniques. This thesis aims to explore further and extend the three model families. In Part I, we propose a decoding method for autoregressive models that is capable of revisiting and revising the generated text thus correcting generation errors. In Part II, we suggest a semi-autoregressive model for long text generation that is capable of modifying its length. The model can also modify the text it has generated. Finally, in Part III, we propose a non-autoregressive model with informative and effective prior. The prior helps to better model the dependencies in the output variables.

Part I

Generation in Autoregressive Models

Chapter 3

Decoding as Dynamic programming For Recurrent Autoregressive Models

Decoding in autoregressive models (ARMs) consists of searching for a high scoring output sequence under the trained model. Standard decoding methods, based on unidirectional greedy algorithm or beam search, are suboptimal due to error propagation and myopic decisions which do not account for future steps in the generation process.

In this chapter we present a novel decoding approach based on the method of auxiliary coordinates (Carreira-Perpinan & Wang, 2014) which has previously been proposed for gradient-free training of deep neural networks. Our method introduces discrete variables for output tokens, and auxiliary continuous variables representing the states of the underlying ARM. The auxiliary variables lead to a factor graph approximation of the ARM, whose maximum a posteriori (MAP) solution is found exactly using dynamic programming. The MAP solution is then used to recreate an improved factor graph approximation of the ARM via updated auxiliary variables. We then extend our approach to decode in an ensemble of ARMs, possibly with different generation orders, which is out of reach for the standard unidirectional decoding algorithms. Experiments on the text infilling task over SWAG and Daily Dialogue datasets show that our decoding method is superior to strong competing decoding methods.

3.1 Introduction

Neural autoregressive models (ARMs) have shown remarkable performance on various natural language processing tasks such as question answering, machine translation, summarization and reading comprehension (Anderson et al., 2018; Bahdanau et al., 2014; Wan et al., 2019). These models are usually trained in an end-to-end manner by optimizing the training objective to learn model parameters.Once the model has been trained, the output is generated by searching for a high scoring sequence given the context and the trained model. This is referred to as the decoding problem.

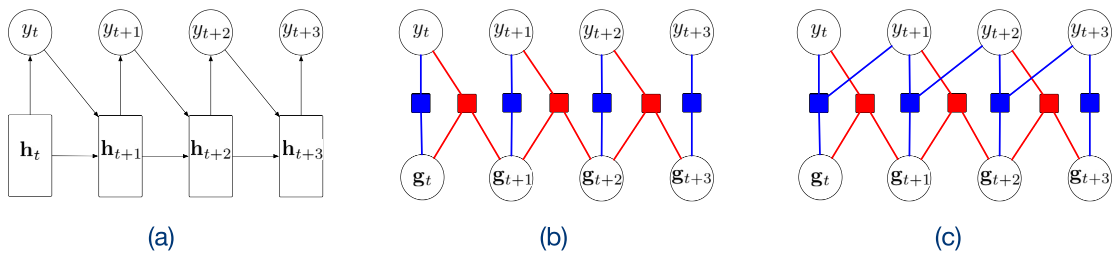


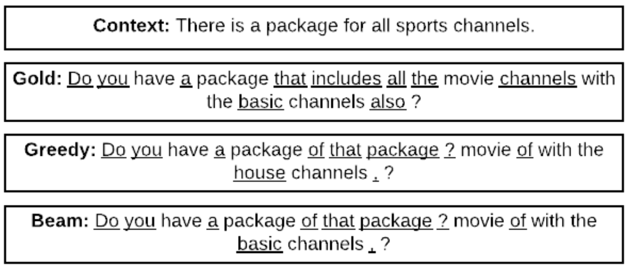
Figure 3.1: A typical RNN with unbounded Markov order is shown in (a). The factor graphs of our zero-order and first-order Markov approximations are illustrated in (b) and (c), respectively. The blue and red factors correspond to likelihood terms and the constraint violations, respectively, from equations (3.3), for order $k = 0$, and (3.4), for $k = 1$.

ARMs create a sequence by repeatedly generating the next symbol conditioned on *all* previous symbols generated. Symbols play dual duty: first they are generated, and next they are incorporated into the conditioning of subsequent decisions. As subsequent decisions can be arbitrarily distant in the sequence, these generation models are non-Markovian, i.e. they do not have a *bounded* Markov order. As prominent examples, ARMs include Elman's recurrent neural networks (RNNs) (Elman, 1990b), conditional text generation models with RNN-based decoder (Sutskever et al., 2014), and transformers (Vaswani et al., 2017).

Exact decoding in ARMs is computationally hard, as the output search space is exponentially large and does not lend itself to efficient algorithms. This is due to *non-decomposable* long-range inter-dependencies among the output variables, i.e. an output token *directly* depends on all of the previously generated tokens. Standard uni-directional decoding algorithms, e.g. greedy and beam search, are ineffective in producing high-scoring output sequences, as errors in the decoding history can adversely affect the future.These algorithms make local decisions to extend an incomplete sequence (hypothesis) by selecting the token with the maximum likelihood at each time step, hoping to get a globally optimal complete sequence (Bahdanau et al., 2014; Sutskever et al., 2014; Mikolov et al., 2010).

In this chapter, we present a novel decoding method, based on the method of auxiliary coordinate (MAC), which has been mainly investigated for training deep neural networks (Carreira-Perpinan & Wang, 2014).Our approach introduces discrete variables for output tokens, and *auxiliary* continuous variables representing the states of the underlying ARM. The auxiliary variables lead to a factor graph approximation of the ARM with a bounded Markov order (see Figure 3.1). We then alternate between optimizing over the output variables and the state variables. The state variables are updated to respect the state dynamics of the underlying ARM and the currently fixed output tokens. The output variables are updated by dynamic programming to exactly optimise a global scoring function, decomposed over local factors determined by the currently fixed state variables. We then extend our MAC-based decoding approach to decode under product of ARM experts (Hinton, 1999), i.e. an *ensemble* of ARMs combined additively in log-space, with each using a different generation order.

To validate our approach, we evaluate on the text infilling task, which consists of filling missing parts of a sentence or a paragraph (Horvat & Byrne, 2014; Tromble & Eisner, 2009; Schmaltz,



Context: There is a package for all sports channels.

Gold: Do you have a package that includes all the movie channels with the basic channels also ?

Greedy: Do you have a package of that package ? movie of with the house channels . ?

Beam: Do you have a package of that package ? movie of with the basic channels . ?

Figure 3.2: Output of greedy and beam search

Rush, & Shieber, 2016). Text infilling is challenging as it requires the global structure of the sentence to fill a blank properly. As shown in the Figure 3.2, both greedy and beam search fail to fill the blanks correctly. We conduct experiments on two datasets: SWAG (Zellers, Bisk, Schwartz, & Choi, 2018) and Daily dialogue (Y. Li et al., 2017) with various mask rates. These datasets are chosen for two reasons. First, they help us to compare our approach with the previous approaches which have employed these two datasets. Secondly, they help us to properly demonstrate the usefulness of our approach. Using open text generation problems such as machine translation would increase the computational complexity thus requiring huge computational and memory resources. Using these datasets, we show that our decoding approach achieve remarkable improvements against the greedy and beam search algorithms as well as TIGS (D. Liu, Fu, Liu, & Lv, 2019), a recently introduced strong inference method for this task. TIGS decodes by iteratively optimizing the output variables in a continuous relaxation of the discrete output space, and then projecting back the fractional solution to the discrete space. In contrast, the outputs in our approach stay in the discrete space, and we iteratively optimize over both the outputs and the continuous state variables. Our approach does have limitations, most notably the computational complexity which is polynomial in the vocabulary size, thus limiting its application to open text generation problems such as machine translation, summarization and story generation.

3.2 Decoding Framework

Notations. We denote scalars, vectors and matrices using lower-case, bold lower-case and bold upper-case letters, e.g. y, $\mathbf{y}$ and $\mathbf{Y}$. Individual elements of $\mathbf{y}$ are denoted as y_i.

3.2.1 Problem Formulation

Consider a recurrent neural network model for text generation. The probability of generating a sequence $y_1, .., y_n$ under the RNN is decomposed as

$$P(y_1, .., y_n) = \prod_{i=1}^{n} P(y_i | \mathbf{y}_{<i}) , \tag{3.1}$$

where $P(.|\mathbf{y}_{<i}) := \mathrm{softmax}(\mathbf{W}\mathbf{h}_i + \boldsymbol{b})$, and the state dynamics is $\mathbf{h}_i = \boldsymbol{f}(\mathbf{h}_{i-1}, y_{i-1})$. Decoding then refers to the following optimization problem,

$$\arg\max_{y_1,..,y_n} \log P(y_1, .., y_n) = \sum_{i=1}^{n} \log P(y_i|\mathbf{y}_{<i}) \tag{3.2}$$

The above optimization problem is computationally hard as it decomposes to conditional probabilities with unbounded length for the conditioning contexts, i.e. a non-Markovian model.

3.2.2 Method of Auxiliary Coordinate (MAC)

Our goal is to decompose the optimisation problem in eqn (3.2) into smaller optimisation problems, which can be solved jointly and coupled via the state variables. This would make decoding more resilient compared to uni-directional decoding, where early errors can adversely affect the future.

Let us start by considering the following decomposition of the decoding problem,

$$\boxed{\begin{aligned} \arg\min_{\mathbf{y}_1^n, \mathbf{g}_1^n} \quad & -\sum_i \log P(y_i|\mathbf{g}_i) \\ & \text{s.t.} \\ & \mathbf{g}_i = \boldsymbol{f}(\mathbf{g}_{i-1}, y_{i-1}) \quad \forall i \in [2, .., n]\,. \end{aligned}} \tag{3.3}$$

Notice the appearance of the new explicit variables $\mathbf{g}$ in this optimisation formulation of the decoding problem, which mirror the role of $\mathbf{h}$ in the underlying RNN. Eliminating the $\mathbf{g}$ variables would make this optimisation problem exactly equivalent to the original decoding problem in eqn (3.2). The $\mathbf{g}$ serve as continuous auxiliary coordinates, following the MAC technique (Carreira-Perpinan & Wang, 2014).

The objective function in eqn (3.3) links the state variable $\mathbf{g}_i$ to token variable y_i generated at the time step i. This is a *zero-order* decomposition of the log-likelihood function in eqn (3.2), as the terms in the objective do not condition on the previous tokens. Interestingly, we can generalise to a k^{th} *order* decomposition by linking the state variable $\mathbf{g}_{i-k}$ from k steps in the past to the generation of the current token y_i. This is due to the fact that the current hidden state, responsible for generating the current token in RNNs, is a *deterministic* function of the past hidden state $\mathbf{g}_{i-k}$ and all of the k generated tokens between that time step and the current time step $\mathbf{y}_{i-k}^{i-1}$. Hence, our k^{th} order decoding optimisation problem is written as follows:

$$\boxed{\begin{aligned} \arg\min_{\mathbf{y}_1^n, \mathbf{g}_1^n} \quad & -\sum_i \log P(y_i|\boldsymbol{f}^{(k)}(\mathbf{g}_{i-k}, \mathbf{y}_{i-k}^{i-1})) \\ & \text{s.t.} \\ & \mathbf{g}_i = \boldsymbol{f}(\mathbf{g}_{i-1}, y_{i-1}) \quad \forall i \in [2, .., n]\,, \end{aligned}} \tag{3.4}$$

where $\boldsymbol{f}^{(k)}(\mathbf{g}_{i-k}, \mathbf{y}_{i-k}^{i-1})$ denotes k repeated applications of the RNN's state transition function $\boldsymbol{f}(.)$ to compute the current hidden state from the past state $\mathbf{g}_{i-k}$ and the tokens observed since then $\mathbf{y}_{i-k}^{i-1}$. For example with $k = 2$, $\boldsymbol{f}^{(2)}(\mathbf{g}_{i-2}, \mathbf{y}_{i-2}^{i-1}) = \boldsymbol{f}(\mathbf{W}\boldsymbol{f}(\mathbf{W}\mathbf{g}_{i-2}, \mathbf{y}_{i-2}), \mathbf{y}_{i-1})$. The variables g_i and y_i for indices $i \leq 0$ are assigned null values, i.e., 0, or a sentence start sentinel. Assuming the constraints are satisfied, this constrained optimisation problem is equivalent to the decoding problem in eqn (3.2). Otherwise, when the constraints are not met, the k^{th} order objective results in a more accurate approximation to the original decoding problem compared to the first-order formulation in eqn (3.3).

As k is made larger, the reliance on $\mathbf{g}$ is reduced, as $f^{(k)}$ uses the RNN's state transition function directly. This will be important in early stages of optimisation, where $\mathbf{g}$ does not accurately reflect the RNN state dynamic. The cost of using a higher order, is a higher complexity of inference, which we discuss in Section 4.

In summary, our decoding approach results in a constrained optimisation problem, incorporating two types of *factors* corresponding to the likelihood terms in the objective function and the constraint violations, respectively, which are denoted by blue and red, in the factor graph of Figure 3.1.

3.3 Optimisation Algorithm

We now turn to solving the constrained optimisation problem in eqn (3.4). Using the quadratic-penalty (QP) method (Nocedal & Wright, 2006), we turn it to an unconstrained optimisation problem,

$$\min_{\mathbf{g}_1^n, \mathbf{y}_1^n} \mathcal{L}(\mathbf{g}_1^n, \mathbf{y}_1^n, \mu) := \sum_i -\log P(y_i | \boldsymbol{f}^{(k)}(\mathbf{g}_{i-k}, \mathbf{y}_{i-k}^{i-1})) + \mu ||\mathbf{g}_i - \boldsymbol{f}(\mathbf{g}_{i-1}, y_{i-1})||_2^2. \quad (3.5)$$

This suggests a two-step block coordinate descent algorithm to alternate between (i) optimizing $\mathbf{y}$'s while $\mathbf{g}$'s are fixed, and (ii) optimizing $\mathbf{g}$'s while $\mathbf{y}$'s are fixed. Other methods for constrained optimization can be used, e.g. the augmented Lagrangian, rather than the quadratic-penalty method (Nocedal & Wright, 2006; Taylor et al., 2016; J. Wang, Yu, Chen, & Zhao, 2019). However, the focus of our work is to investigate the effectiveness of decomposing the non-Markovian decoding objective to the bounded-Markov constrained optimisation problem in eqn (3.4), so we leave it to the future work to investigate the effect of different optimisation strategies. It is worth noting that the above optimisation problem can be also interpreted as inference objective in an *stochastic* RNN, where the next hidden state $\mathbf{g}_i$ is conditionally generated based on the previous hidden state according to a multivariate Gaussian distribution with the mean vector $\mathbf{g}_{i-1}$ and the diagonal covariance matrix whose diagonal elements are μ^{-1}.

3.3.1 Updating the Output Variables

Assuming $\mathbf{g}$'s are fixed, the discrete output variables $\mathbf{y}$'s can be updated to *exactly* solve eqn (3.5) using a variant of the Viterbi algorithm (Viterbi, 1967). Let $P_{\mathbf{g}_{i-k}}(.|\mathbf{y}_{i-k}^{i-1}) := \mathrm{softmax}(\mathbf{W} \cdot \boldsymbol{f}^{(k)}(\mathbf{g}_{i-k}, \mathbf{y}_{i-k}^{i-1}) + \boldsymbol{b})$ be the conditional probability of the next token given the previous k tokens and $\mathbf{g}_{i-k}$. Consider a dynamic programming table $T \in R^{|\mathcal{Y}^k| \times n}$. The table is filled from left to right, and at each time i, the element corresponding to the DP state $\mathbf{y}_{i-k+1}^{i} \in \mathcal{Y}^k$ is computed as,

$$T[\mathbf{y}_{i-k+1}^{i}, i] \leftarrow \min_{y' \in \mathcal{Y}} T[y' \circ \mathbf{y}_{i-k+1}^{i-1}, i-1] - \log P_{\mathbf{g}_{i-k}}(y_i | y' \circ \mathbf{y}_{i-k+1}^{i-1}) + \mu ||\mathbf{g}_i - \boldsymbol{f}(\mathbf{g}_{i-1}, y_{i-1})||_2^2$$

where $y' \circ \mathbf{y}_{i-k+1}^{i-1}$ denotes the concatenation of the token y' to the beginning of the sequence of tokens $\mathbf{y}_{i-k+1}^{i-1}$, which results in a sequence of tokens of length k. Once the DP table is built, the optimal sequence can be read off by traversing the table from the end towards the beginning. The time complexity of the DP is $O(n|\mathcal{Y}|^{k+1})$ where n is the length of the sequence, k is the Markov order, and $|\mathcal{Y}|$ is the size of the vocabulary.

3.3.2 Updating the State Variables

Next, we turn to optimizing the penalized decoding objective (eqn 3.5) with respect to the continuous auxiliary variables, $\mathbf{g}$, assuming fixed outputs, $\mathbf{y}$. For particular combinations of state transition and the likelihood functions, the optimal state values may have gradient-free closed form solution. However, we assume the general case, where the state transition and likelihood functions are given typical nonlinear functions. In the absence of closed-form solution, the simplest method is to use gradient-based optimization algorithms. Interestingly, the computational graph corresponding to the penalized decoding objective considers the state variables of *all* positions as the input when computing the output $\mathcal{L}(\mathbf{g}_1^n, \mathbf{y}_1^n, \mu)$. This means, there is no need for backpropagation through the time (BPTT), as opposed to the underlying RNN, to compute the gradient as the contribution of all positions is taken into account in parallel.

We also propose an approximation method to simplify the $\mathbf{g}$ update, by ignoring the log-likelihood term in the objective, which we denote by *forced decoding*. In this case, the penalty term can be optimized exactly, and reduced to zero by computing $\mathbf{g}$'s according to the RNN state transition function $\boldsymbol{f}(.)$ from-left-to-right while the output variables are fixed. This update does not involve computing the gradients, and only involves a forward pass through the RNN.

3.4 Decoding In An Ensemble

In this section, we extend our MAC-based decoding approach to decode under an ensemble of ARMs, each using a different generation order. For example, consider two RNN-based language models, left-to-right (L2R) and right-to-left (R2L), where each of which gives a score to an assignment of words to the blank positions in the text infilling task. We are then interested to find an assignment of words which has the maximum sum of the scores under these two models. As both models are in the exponential family, this corresponds to a product of experts (Hinton, 1999). Unidirectional decoding algorithms cannot decode under such ensembles.

Let us consider the decoding problem in an ensemble of left-to-right and right-to-left RNNs,

$$\arg\max_{y_1,..,y_n} \sum_i \log \overrightarrow{P}(y_i|\mathbf{y}_{<i}) + \log \overleftarrow{P}(y_i|\mathbf{y}_{>i})$$

where each RNN has its own parameters. We then reformulate this optimisation problem, using auxiliary variables $\mathbf{g}_i$ and $\mathbf{g}'_i$ for the L2R and R2L RNNs, as follows,

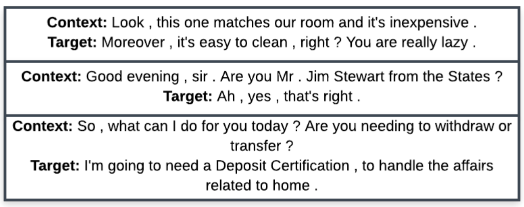
$$\boxed{\begin{aligned} &\arg\min_{\mathbf{y}_1^n,\mathbf{g}_1^n,\mathbf{y}'^n_1,\mathbf{g}'^n_1} \; -\sum_i \log \overrightarrow{P}(y_i|\overrightarrow{\boldsymbol{f}^{(k)}}(\mathbf{g}_{i-k},\mathbf{y}_{i-k}^{i-1})) + \log \overleftarrow{P}(y'_i|\overleftarrow{\boldsymbol{f}^{(k)}}(\mathbf{g}'_{i+k},\mathbf{y}'^{i+1}_{i+k})) \\ &\qquad\qquad \text{s.t.} \\ &\qquad\qquad \mathbf{g}_i = \overrightarrow{\boldsymbol{f}}(\mathbf{g}_{i-1},y_{i-1}) \quad \forall i \in [2,..,n] \\ &\qquad\qquad \mathbf{g}'_i = \overleftarrow{\boldsymbol{f}}(\mathbf{g}'_{i+1},y_{i+1}) \quad \forall i \in [1,..,n-1] \\ &\qquad\qquad \boldsymbol{e}[y_i] = \boldsymbol{e}[y'_i] \qquad\quad \forall i \in [1,..,n] \end{aligned}} \tag{3.6}$$

where the constraints in eqn (3.6) couple the two optimisation problems corresponding to the L2R and R2L RNNs. Note that we have enforced the equality between the *embeddings* of the words y_i and y'_i produced by the L2R and R2L models, respectively (denoted $\boldsymbol{e}[y]$). This provides a denser signal, e.g. from synonyms or words with related syntactic categories, in order to couple the two optimisation problems, compared to a sparse signal from constraints using *identity* of the tokens.

To solve the above optimisation problem, we use of the quadratic penalty method, similar to the previous section, i.e.,

$$\begin{aligned} \min_{\mathbf{g}_1^n,\mathbf{y}_1^n,\mathbf{g}'^n_1,\mathbf{y}'^n_1} \sum_i & -\log \overrightarrow{P}(y_i|\overrightarrow{\boldsymbol{f}^{(k)}}(\mathbf{g}_{i-k},\mathbf{y}_{i-k}^{i-1})) - \log \overleftarrow{P}(y'_i|\overleftarrow{\boldsymbol{f}^{(k)}}(\mathbf{g}'_{i+k},\mathbf{y}'^{i+1}_{i+k})) \\ +\mu||\mathbf{g}_i - \overrightarrow{\boldsymbol{f}}(\mathbf{g}_{i-1},y_{i-1})||_2^2 &+ \mu'||\mathbf{g}'_i - \overleftarrow{\boldsymbol{f}}(\mathbf{g}'_{i+1},y'_{i+1})||_2^2 + \mu''||\boldsymbol{e}[y_i] - \boldsymbol{e}[y'_i]||^2 . \end{aligned}$$

This suggests an optimisation algorithm which alternates between (i) updating $\{\mathbf{y}_1^n,\mathbf{g}_1^n\}$ in the first phase while the other variables are fixed, and (ii) updating $\{\mathbf{y}'^n_1,\mathbf{g}'^n_1\}$ in the second phase while the other variables are fixed. The updates for each of these phases is done using an iterative algorithm, similar to those presented in section 3.3, for updating the output and state variables. The only modification in the objective function of each phase (compared to the previous section) is the inclusion of the token embedding constraints $||\boldsymbol{e}[y_i] - \boldsymbol{e}[y'_i]||^2$, which can be easily accounted for in the dynamic programming algorithm when updating the output variables.

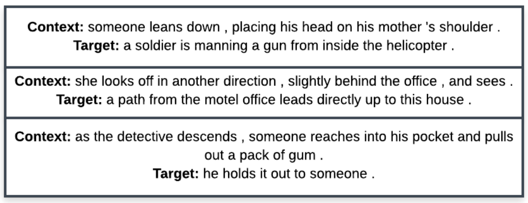


Context: Look , this one matches our room and it's inexpensive .
Target: Moreover , it's easy to clean , right ? You are really lazy .

Context: Good evening , sir . Are you Mr . Jim Stewart from the States ?
Target: Ah , yes , that's right .

Context: So , what can I do for you today ? Are you needing to withdraw or transfer ?
Target: I'm going to need a Deposit Certification , to handle the affairs related to home .

(a)



Context: someone leans down , placing his head on his mother 's shoulder .
Target: a soldier is manning a gun from inside the helicopter .

Context: she looks off in another direction , slightly behind the office , and sees .
Target: a path from the motel office leads directly up to this house .

Context: as the detective descends , someone reaches into his pocket and pulls out a pack of gum .
Target: he holds it out to someone .

(b)

Figure 3.3: Some test examples from (a) Daily Dialogue and (b) SWAG datasets.

3.5 Experiments

3.5.1 Experimental Setup

The Text Infilling Task Text infilling consists of predicting missing parts of a sentence or a paragraph. The task is encountered in everyday applications of restoring historical or damaged documents (Zhu, Hu, & Xing, 2019), writing articles or contracts with templates (Ippolito, Grangier, Callison-Burch, & Eck, 2019) and text editing (Feng, Li, & Hoey, 2019). Although important, text infilling is less explored and has been studied under simplified and restricted settings (Fedus, Goodfellow, & Dai, 2018; Zweig & Burges, 2011; Holtzman et al., 2018; Fan et al., 2018). For example, Horvat and Byrne (2014) restricted the vocabulary to the set of gold standard words blanked in the sentence. Here we follow a setup similar to (D. Liu et al., 2019), which also limits the vocabulary, but places no restriction on the number and position of the blanks, and thus is more similar to situations encountered in real life applications.

More formally let B be a *mask*, comprising set of indices where blanks appear in the sentence. Let $\mathbf{y}^B$ be a target sentence where tokens at the positions of the sentence in B have been masked. For example if $B = \{i, i+1\}$ then $\mathbf{y}^B$ is $\{y_1, ...y_{i-1}, \_, \_, y_{i+2}, ..y_n\}$. Given the context $\mathbf{x}$ and masked sentence $\mathbf{y}^B$, the aim is to fill in the blanks as they appear in sentence. This requires considering global structure of the sentence along with the conditioning context.

Datasets. We evaluate our proposed approach on two text infilling tasks over two widely used publicly available corpora. The first task is conversation reply with a template (denoted as Daily) which is conducted on the DailyDialog dataset (Y. Li et al., 2017). Similar to D. Liu et al. (2019) we convert the multi-turn dialogues into single-turn dialogues, resulting in 82,372 conversation pairs. The query sentence is used as input to the encoder $\mathbf{x}$ where as the reply is the output $\mathbf{y}$ from the decoder.

The second task is captions from movies along with an ending (denoted as SWAG) which is conducted on the SWAG dataset (Zellers et al., 2018). We only consider the correct endings to build the target side. This gives us 73,000 pairs of sentences. The input to the encoder is the caption $\mathbf{x}$ whereas the decoder has to produce the ending $\mathbf{y}$ conditioned on $\mathbf{x}$. Some examples of these datasets are shown in Figure 3.3.

Training. We trained both left-to-right (L2R) and right-to-left (R2L) models, where R2L models are trained by reversing the target side sentence. All models are trained with a word embedding size and hidden dimension size of 512. We use ADAM optimiser to train the models with an initial learning rate of 0.001. Since the source and target sides are in the same language, we shared the word embeddings between the encoder and decoder. The models were trained for 10 epochs.

Baselines. Our baselines include: greedy decoding, beam search, and a strong recently proposed inference algorithm for the text infilling task TIGS (D. Liu et al., 2019). TIGS decodes by iterating through the following steps: (i) relaxing the space of output variables from discrete to continuous and optimise over the continuous output variables using gradient based optimization, and (ii) projecting back the solution from the continuous space to discrete. In contrast, the outputs in our approach stay in the discrete space, and we optimize over both discrete and the continuous state variables, as part of an iterative coordinate descent procedure. Our methods and the baselines are implemented on top of OpenNMT (Klein, Kim, Deng, Senellart, & Rush, 2017).

Evaluation Metrics. For a fair comparison with previous works, we have used BLEU score (Papineni et al., 2002) to evaluate the models. As all models are evaluated under the same conditions, unmasked tokens will not affect the conclusions drawn from the results. Along with BLEU score, we have also reported the perplexity of the model. For BLEU score, higher values are better whereas for perplexity lower values are better.

Decoding Parameters We use the Nesterov optimiser with a learning rate of 0.1. We experiment with Adam and simple SGD and find empirically that Nesterov works better than the other optimisers. Nesterov is a momentum based optimiser that stabilize the update directions and seems to better escape from poor local optima during the decoding iterations. Rather than taking a step in the direction of updated accumulated gradient, Nesterov optimiser first moves in the direction of previously accumulated gradient. It calculates the new gradient and then makes a correction. This prevents the optimiser from going too fast and results in increased responsiveness, which significantly improves the performance.

All μ's for the penalty terms corresponding to different constraints, are initialised with 0.5 and are multiplied by 1.2 after 5 iterations, and decoding was run for 10 iterations, chosen as the first order method had reliably converged in terms of objective value and the output string.

We use the RNN states $\mathbf{h}$ corresponding to the beam search solution to initialize the $\mathbf{g}$ variables in our decoding method. We experiment with initialising the hidden states with random values, or using the values corresponding to the greedy solution. We find that using beam search for initialisation gives better results. Compared to beam search, random initialisation requires an average of 12x more iterations, and would occasionally suffer from non-convergence, where the solution oscillates. We find negligible difference between the number of iterations needed for convergence between the beam and greedy search initialised hidden states.

		Dialogue		SWAG	
	Decoding Method	BLEU	PPLX	BLEU	PPLX
L2R	Greedy	71.2	5.88	71.6	5.64
	Beam	71.3	5.84	71.8	5.58
	TIGS	73.0	4.31	74.0	3.49
	Ours	**79.3**	**3.49**	**83.9**	**2.31**
R2L	Greedy	70.2	6.96	63.9	7.07
	Beam	70.4	6.90	64.1	6.97
	TIGS	71.8	4.87	66.5	4.68
	Ours	**77.9**	**3.80**	**78.7**	**3.04**
Both	Ours	80.2	-	79.3	-
	L2R component	-	3.30	-	3.39
	R2L component	-	3.73	-	4.15

Table 3.1: BLEU score and perplexity of various models on the two datasets with 50% masking rate. The results of our decoding approach is based on the 1st order approximation.

3.5.2 Results

The results are reported in Table 3.1. Following D. Liu et al. (2019), we build a test set of 5000 sentences for each dataset. We use a 50% masking rate and randomly place the blanks for each sentence. We perform experiments on left-to-right (L2R), right-to-left (R2L), and an ensemble of the two.

Generally, L2R models outperform R2L models. This may be due to sentences being generated inherently in a left to right manner. Hence modelling the writing process with a right to left model may make it difficult to learn useful patterns.The trend across all the models and both datasets is that a decrease in perplexity leads to a better BLEU score. The benchmark TIGS method (D. Liu et al., 2019) outperforms both the greedy algorithm and beam search. However, our decoding method outperforms TIGS and other baselines significantly. Compared to the greedy algorithm and beam search, our method and TIGS leverage information from both future and the past. Compared to TIGS, our method operates by keeping the output variables in the discrete space, whereas in TIGS the output variables are relaxed to the continuous space and projected back to the discrete space.

Given that context from both directions helps in better decoding when working with unidirectional models, we perform experiments on an ensemble of L2R and R2L models. Notably, the unidirectional greedy algorithm and beam search cannot operate on the ensemble. Our method, instead, can decode with the ensemble, which further improves the BLEU score for the Dialogue task.

3.5.3 Analysis

Varying the Masking Rate. Increasing the masking rate makes it difficult for all the models to correctly fill in the blanks. We experiment on the dialogue dataset by randomly masking the test set with the rates 25%, 50%, and 75%. Results are reported in Table 3.2. As the masking rate increases, BLEU score decreases, whereas perplexity increases. Compared to the other techniques, our method is able to achieve better results even with high masking rates.

Mask Rate	25%		50%		75%	
	BLEU	PPLX	BLEU	PPLX	BLEU	PPLX
Greedy	85.4	4.43	71.2	5.88	60.4	4.25
Beam	85.4	4.43	71.3	5.84	62.0	4.03
TIGS	88.0	3.47	73.0	4.31	62.5	3.53
Ours (1st order)	**90.9**	**2.80**	**79.3**	**3.49**	**64.3**	**2.47**

Table 3.2: Performance with varying masking rates for the different decoding methods.

Varying the Markov order and State Variables Update. Changing the state variable update method from forced decoding to gradient based results in a lower perplexity. This is because the state variables are updated based upon the gradient of the objective function. Table 3.3 shows the results on Daily dataset with random 50% masking rate of varying the state variable update method for different orders of Markov model. The perplexity goes down as the k increases whereas the perplexity of gradient based approach is lower than the forced decoding approach for state variable update.

Varying the Markov Order. Our *penalized* decoding objective is composed of the negative log-likelihood term and the penalty (see eqn 3.5). As the algorithm proceeds, both of these terms decrease, showing consistency in the auxiliary variables. We hypothesise that, although higher order Markov models produce more accurate approximations to the original decoding problem, they result in *harder* optimization problems. The following figure (right) plots the penalized decoding objective in k-th order Markov approximations for $k \in \{0, 1, 2\}$. The results are based on 100 examples in the test set for the Daily Dialogue dataset, with random masking using 50% masking rate. Observe that the penalized decoding objective tends to decrease less for the 2nd order method compared to the others. Table 3.3 reports the perplexity, BLEU score, and decoding time for these different Markov approximations. The results confirm the increase in the BLEU score and decrease in the perplexity, as the Markov order increases. Furthermore, it shows the trade-off in the solution quality vs. decoding time for these different approximations. In Table 3.3, we report results based on two different methods for updating the state variables in our method, i.e. gradient-based and forced decoding. As expected, gradient-based updates produced more accurate results compared to forced decoding, but at the cost of a longer run-time.

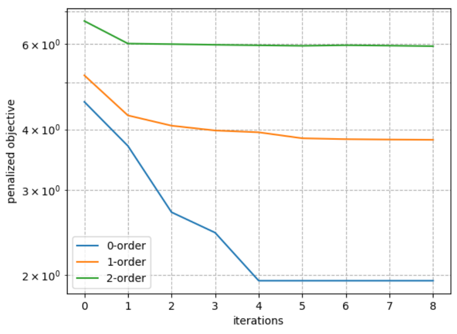
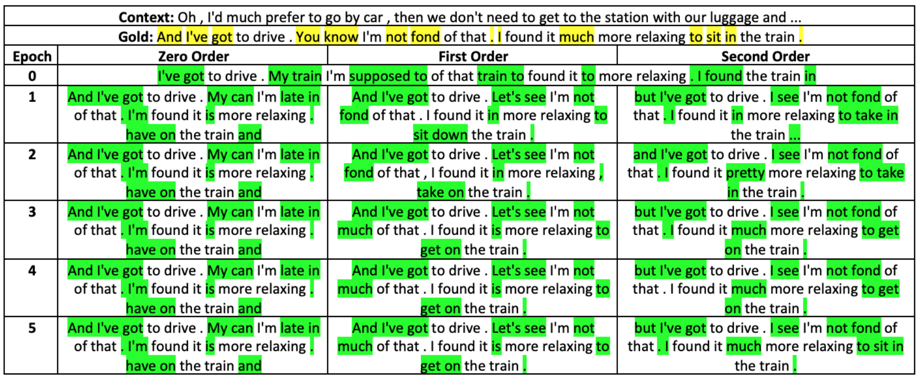


Figure 3.4: The figure shows the penalised decoding objective for zero, first and second order Markov models

Figure 3.5 shows the iterative improvement of an example test sentence in each iteration of our decoding methods with varying Markov order. Each method is given the same initial solution

	Gradient Based			Forced Decoding		
	BLEU	PPLX	Time	BLEU	PPLX	Time
0-order	77.4	6.55	297s	75.4	6.86	65s
1-order	84.2	3.42	348s	82.6	3.75	93s
2-order	85.5	2.33	893s	83.4	3.34	711s

Table 3.3: The results of varying Markov order and state variable update method. Time is reported for processing 100 sentences.



Context: Oh , I'd much prefer to go by car , then we don't need to get to the station with our luggage and ...			
Gold: And I've got to drive . You know I'm not fond of that . I found it much more relaxing to sit in the train .			
Epoch	**Zero Order**	**First Order**	**Second Order**
0	I've got to drive . My train I'm supposed to of that train to found it to more relaxing . I found the train in		
1	And I've got to drive . My can I'm late in of that . I'm found it is more relaxing . have on the train and	And I've got to drive . Let's see I'm not fond of that . I found it in more relaxing to sit down the train .	but I've got to drive . I see I'm not fond of that . I found it in more relaxing to take in the train ...
2	And I've got to drive . My can I'm late in of that . I'm found it is more relaxing . have on the train and	And I've got to drive . Let's see I'm not fond of that , I found it in more relaxing , take on the train .	and I've got to drive . I see I'm not fond of that . I found it pretty more relaxing to take in the train .
3	And I've got to drive . My can I'm late in of that . I'm found it is more relaxing . have on the train and	And I've got to drive . Let's see I'm not much of that . I found it is more relaxing to get on the train .	but I've got to drive . I see I'm not fond of that . I found it much more relaxing to get on the train .
4	And I've got to drive . My can I'm late in of that . I'm found it is more relaxing . have on the train and	And I've got to drive . Let's see I'm not much of that . I found it is more relaxing to get on the train .	but I've got to drive . I see I'm not fond of that . I found it much more relaxing to get on the train .
5	And I've got to drive . My can I'm late in of that . I'm found it is more relaxing . have on the train and	And I've got to drive . Let's see I'm not much of that . I found it is more relaxing to get on the train .	but I've got to drive . I see I'm not fond of that . I found it much more relaxing to sit in the train .

Figure 3.5: Improvement of the sentence in different iterations

produced by the beam search. The zero-order method converges to a bad solution in the first iteration and gets stuck there in future iterations. The first-order method arrives to a further improved solution while the second order method can find the best solution among all.

Discussion on masked language models. Recent development in transformer model (Vaswani et al., 2017) has paved the way for pre-trained language models such as BERT and GPT2 (Devlin, Chang, Lee, & Toutanova, 2018; Brown et al., 2020). These massive pre-trained models are trained using a large corpora of generic text, and then fine-tuned with small domain-specific data. The models are trained using a masked language modelling objective where similar to text infilling part of the sentence is masked. The model predicts all masked tokens in parallel by utilising the attention mechanism in the transformer model. Unlike BERT our approach does not require huge amounts of data for training and can be used on top of any trained language model. We have left this as a future investigation to quantify the gains of using our approach on top of self-attention models such as transformers. Further, our method is for better decoding/inference on autoregressive models, compared to the typical decoding algorithms such as greedy and beam search algorithms. Therefore, the comparison with non-autoregressive models, such as BERT, may not be that helpful and are out of the scope.

3.6 Summary

This work presented a method for improving decoding in discrete autoregressive models using dynamic programming. The core idea is to introduce auxiliary variables to decouple the non-Markovian aspects of the model, permitting an approximate solution. This solution is used to create the next model approximation, and the process iterates. The problem is hard as it requires implementing the constraints in the generated solution. Thanks to our framework we have been able to satisfy most of the constraints. Our results show that our decoding framework is effective, leading to substantial improvements over greedy and beam search baselines.

Part II

Generation in Semi-Autoregressive Models

Chapter 4

A Hierarchical Model For Document Generation

Generating long and coherent text is an important and challenging task encompassing many application areas such as summarization, document level machine translation and story generation. Despite the success in modeling intra-sentence coherence, existing long text generation models (e.g., BART and GPT-3) still struggle to maintain a coherent narrative throughout the generated text. We conjecture that this is because of the difficulty for the model to revise, replace or revoke any part that has been generated by the model. Revisiting and editing part of the text helps model in rectifying the generated mistakes and results in a better output sequences.

In this chapter, we present a novel semi-autoregressive document generation model capable of revising and editing the generated text. Building on recent models by (Gu et al., 2019; Xu & Carpuat, 2020), we formulate document generation as a hierarchical Markov decision process with a two level hierarchy, where the high and low level editing programs generate and refine the document. The top level editing actions edit and improve intra-sentence coherence whereas the low level editing actions are responsible for inter-sentence coherence. We train our model using imitation learning (Hussein, Gaber, Elyan, & Jayne, 2017) and introduce roll-in policy such that each policy learns on the output of applying the previous action. Experiments applying the proposed approach underperform state-of-the-art but show promise and convey various insights on the problem of long text generation using our model. We suggest various remedies such as using distilled dataset, designing better attention mechanisms and using autoregressive models as a low level program as future directions.

4.1 Introduction

Generating long and coherent text encompass various tasks such as summarization, story generation, document level machine translation and document level post editing. Each task is characterised by modelling long range dependencies to make the document coherent as well as modelling a high level plot to make the document thematically consistent (Fan et al., 2018). This is challenging as the models need to plan content, while producing local words consistent with the global context in a timely manner.

Recent work on autoregressive generation models, such as GPT-3 and BART (Lewis et al., 2019; Brown et al., 2020), have shown impressive performance in generating short fluent text with a maximum length ranging from 150 to 350 tokens (Bosselut et al., 2018; Shen et al., 2019; L. Zhao et al., 2020). But applying the same model to generate longer passages of text (e.g., 1000 tokens) has resulted in syntactic and semantic errors throughout the document requiring extensive human curations (B. Tan et al., 2020). These massive language models are usually pre-trained using large corpora of generic text, and then fine-tuned with small domain-specific data. Most of the time, the models are not publicly available to adapt to arbitrary desired domains.

On the other hand, recent non-autoregressive approaches allow generation to be done within a much smaller number of decoding iterations (Gu, Bradbury, et al., 2017; Y. Wang et al., 2019; Kasai, Cross, Ghazvininejad, & Gu, 2020a). But due to its problems with modelling dependencies among the tokens, the approach still lags behind its autoregressive counterparts and has not yet been applied to long text generation (C. Zhou, Neubig, & Gu, 2019; Gu & Kong, 2020). In both of these model families, the length of generated sequences is either fixed or monotonically increased as the decoding proceeds. This makes them incompatible with human-level intelligence where humans can revise and edit any part of their generated text.

In this chapter, we present a novel semi-autoregressive document generation model capable of revising and editing the generated text. Revisiting and editing generated text allows model to rectify mistakes and improve coherence which cannot be done by autoregressive and non-autoregressive models. We build on recent models by (Gu et al., 2019; Xu & Carpuat, 2020), who framed generation as a Markov decision process (Garcia & Rachelson, 2013) and showed that iteratively refining output sequences via insertions and repositions yields a fast and flexible generation process for machine translation and automatic post editing task. We extend their model by proposing document generation as a hierarchical Markov decision process (M. Liu, Buntine, & Haffari, 2018) with a two level hierarchy. The high level program produces actions $a_H \in \{\textit{reposition, insert, update}\}$ which capture global context and plan content while the low level program produces actions $a_L \in \{\textit{reposition, insert}\}$ to generate local words in a consistent and timely manner. Due to unavailability of large-scale data to train our model, we propose a noising process to simulate the error patterns observed in document level tasks such as redundancy of words, key information omission and disordered sentences. The noising process can be reversed by applying a set of high and low level actions to get back the original document. This serves as an efficient oracle to train our model using imitation learning (Hussein et al., 2017). The roll-in policy is defined such that each policy learns on the output of applying the previous action.

To validate our model in section 4.4.1, we conduct experiments on synthetic dataset, where input-output pair is an unsorted and sorted sequence of numbers and on ROC stories dataset (Mostafazadeh et al., 2016). We also experiment with summarization datasets such as multinews (Fabbri, Li, She, Li, & Radev, 2019) and DUC2004 (Over & Yen, 2004) and compare the results with recent summarization model by (J. Zhao et al., 2020). Although our model lags behind the baseline model, it does shed light on the problems of long sequence generation. We present various approaches to mitigate these problems by using distilled dataset which has been found useful in non-autoregressive generation, designing better attention mechanism and using an autoregressive model as low level program to benefit from unbounded conditioning context.

4.2 Problem Formulation

4.2.1 Hierarchical Markov decision process

We cast document generation and refinement as a hierarchical Markov decision process (HMDP) with a two level hierarchy. The high level program is defined by the tuple $(\mathcal{D}, \mathcal{A_H}, \mathcal{E}, \mathcal{R}, \mathbf{d_0})$ where a state $\mathbf{d} \in \mathcal{D}$ corresponds to a set of sequences $\mathbf{d} = (\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_L})$ up to length L, and $\mathbf{d_0} \in \mathcal{D}$ is the initial document. The low level program corresponds to the tuple $(\mathcal{S}, \mathcal{A_L}, \mathcal{E}, \mathcal{R}, \mathbf{s_0})$ where a state $\mathbf{s} \in \mathcal{S}$ corresponds to a sequence of tokens $\mathbf{s} = (w_1, w_2, ..., w_n)$ from the vocabulary V up to length n, and $\mathbf{s_0} \in \mathcal{S}$ is the initial sequence.

At any time step t, the model takes as input $\mathbf{d_{t-1}}$, the output from the previous iteration, chooses an action $a_H \in \mathcal{A_H}$ to refine the sequence into $\mathbf{d_t} = \mathcal{E}(\mathbf{d_{t-1}}, a_H)$, and receives a reward $r_t = \mathcal{R}(\mathbf{d_t})$. The policy π_H maps the input sequence $\mathbf{d_{t-1}}$ to a probability distribution $P(A_H)$ over the action space $\mathcal{A_H}$. A high level program may call a low level program with the initial input $\mathbf{s_0}$. The low level program is similar to high level program with its set of actions $a_L \in \mathcal{A_L}$, reward function $r_t = \mathcal{R}(\mathbf{s_t})$ and the policy π_L. Instead of sequences, the low level actions are applied to individual tokens. This results in a trajectory $\sigma := \{\mathbf{d_1}, a_H^1, \tau_1, r_1, \mathbf{d_2}, ..., \mathbf{d_N}, a_H^N, \tau_N, r_N, \mathbf{d_{N+1}}\}$ which is the concatenation of high-level trajectory $\tau_H := (\mathbf{d}_1, a_H^1, r_1, \mathbf{d}_2, a_H^2, r_2, ..., \mathbf{d}_{H+1})$ and the low level trajectory $\tau_L := (\mathbf{s}_1, a_L^1, \mathbf{s}_2, a_L^2, ..., \mathbf{s}_{T+1})$. We define a reward function $R = dist(\mathbf{d}, \mathbf{d}^*)$ which measures the distance between the generation and the ground-truth sequence. We use Levenstein distance (Levenshtein et al., 1966) as our distance metric.

4.2.2 HMDP policies

Following the formulation of HMDP, we define a high level policy $\pi_H : \mathbf{d} \rightarrow A_H$, as well as the low level policy $\pi_L : \mathbf{s} \rightarrow A_L$ as a mapping from states to actions. The high level actions consist of $a_H \in \{reposition, insert, update\}$ and the low level actions consist of $a_L \in \{reposition, insert\}$.

INSERT$_\mathbf{H}$: The insertion policy reads the input document $\mathbf{d}$ consisting of a set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$, and for every possible slot $i, i+1$, the insertion policy $\pi_H^{ins}(x|i, \mathbf{d})$ makes a binary decision which is 1 (insert here) or 0 (do not insert). For each insertion position, the low level MDP is called to generate the new sequence from scratch. This allows the model to generate a sentence conditioned on the surrounding context resulting in outputs that are consistent with the theme and plot of the document.

$\mathbf{UPDATE_H}$**:** The update policy reads the input document $\mathbf{d}$, consisting of a set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$, and for every sequence position i, the update policy $\pi_H^{upd}(x|i, \mathbf{d})$ makes a binary decision which is 1 (update this sentence) or 0 (do not update). In order to make the update, the low level MDP is called to refine the given sequence. This allows the model to correct mistakes and improve the sentences generated by the insert policy.

$\mathbf{REPOSITION_H}$**:** The reposition policy reads in the document $\mathbf{d}$ consisting of a set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$. For every sentence position i, the reposition policy $\pi_H^{rep}(x|i, \mathbf{d})$ makes a categorical decision between 0 and $L + 1$ where L is the number of sequences in the document. The given sequence is repositioned to the output value. If x is 0 then the sequence is deleted. This policy allows the model to observe the complete document and make it more coherent by repositioning and deleteing sentences.

$\mathbf{INSERT_L}, \mathbf{REPOSITION_L}$**:** The Low level MDP is made up of actions reposition and insert. They work in a similar manner as defined in (Gu et al., 2019; Xu & Carpuat, 2020) with the difference that the conditioning context contains document d along with the sentence s. Therefore the reposition policy at the word level is defined by $\pi_L^{rep}(x|i, \mathbf{y}, \mathbf{d})$. The insertion policy is made up of a placeholder and token prediction policy as defined by $\pi_L^{plh}(x|i, \mathbf{y}, \mathbf{d})$ and $\pi_L^{tok}(x|i, \mathbf{y}, \mathbf{d})$ respectively. The placeholder policy first determines the number of words that need to be inserted at a given position. Special <*mask*> tokens are then inserted. These <*mask*> tokens are filled by the token prediction policy.

4.2.3 Generative process

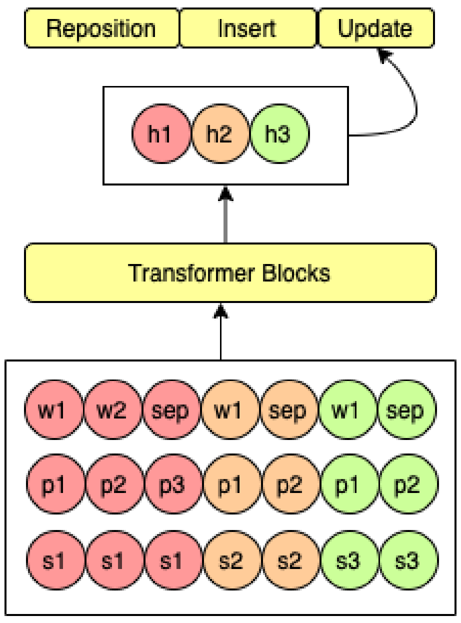
The generative process is outlined in Algorithm 1. The combination of high and low level policies can either generate a document from scratch or edit a given initial document. The insertion and update policy call the low level program in Lines 6 and 11. Line 2 in Algorithm 2 builds the initial scaffolding which is later used by the algorithm for its set of actions. If the low level program is called by the high level update action, the initial scaffolding is created by concatenating the sentences identified by the high level update policy. Otherwise in case of high level insert action, it is the concatenation of empty sentences. Although one iteration is made up of multiple stages, within each stage an action is performed in parallel. The program can be terminated if there is no improvement in the sentence for certain number of iterations or if the maximum number of iterations has been done. The documntUpdate function in line 14 in algorithm 2 insert the sentence back into the document.

Algorithm 1 Generation in HMDP

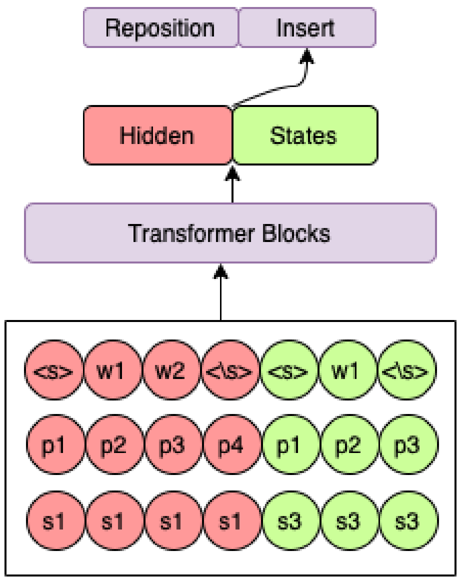
```
Require: Initial document $\mathbf{d}_0$, policy: $\pi_{\theta_H}$
1: $\mathbf{d} \leftarrow \mathbf{d}_0$
2: while Maximum iteration are not done do
3:     $\mathbf{rep\_index} \leftarrow \arg\max_{\mathbf{r}} \sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{rep}_{\theta_H}(r_i|\mathbf{s}_i, \mathbf{d})$        ▷ Do reposition
4:     $\mathbf{d} \leftarrow \mathcal{E}(\mathbf{d}, \mathbf{rep\_index})$
5:     $\mathbf{ins\_index} \leftarrow \arg\max_{\mathbf{p}} \sum_{\mathbf{s}_i, \mathbf{s}_{i+1} \in \mathbf{d}} \log \pi^{ins}_{\theta_H}(p_i|\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{d})$        ▷ Do insertion
6:     $\mathbf{d} \leftarrow \mathcal{E}(\mathbf{d}, \mathbf{ins\_index})$        ▷ Call to Low level MDP
7:     $\mathbf{upd\_index} \leftarrow \arg\max_{\mathbf{u}} \sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{upd}_{\theta_H}(u_i|\mathbf{s}_i, \mathbf{d})$        ▷ Do update
8:     $\mathbf{d} \leftarrow \mathcal{E}(\mathbf{d}, \mathbf{upd\_index})$        ▷ Call to Low level MDP
9: end while
```

Algorithm 2 Low Level MDP

```
Require: Document $\mathbf{d}$, policy: $\pi_{\theta_L}$, Hi Level MDP action: $\mathbf{H}$
1: while Maximum iteration are not done do
2:     $\mathbf{s_0} \leftarrow \mathrm{buildFrame}(\mathbf{d}, \mathbf{H})$
3:     if $\mathbf{s_0}$ is empty then
4:         $\mathbf{s} \leftarrow \mathbf{s_0}$        ▷ Skip reposition
5:     else
6:         $\mathbf{rep\_index} \leftarrow \arg\max_{\mathbf{r}} \sum_{w_i \in \mathbf{s}} \log \pi^{rep}_{\theta_L}(r_i|w_i, \mathbf{s}, \mathbf{d})$        ▷ Do reposition
7:         $\mathbf{d} \leftarrow \mathcal{E}(\mathbf{s}, \mathbf{rep\_index})$
8:     end if
9:     $\mathbf{plh\_index} \leftarrow \arg\max_{\mathbf{p}} \sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi^{ins}_{\theta_L}(p_i|w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$        ▷ Insert placeholders
10:    $\mathbf{s} \leftarrow \mathcal{E}(\mathbf{s}, \mathbf{plh\_index})$
11:    $\mathbf{tok\_index} \leftarrow \arg\max_{\mathbf{t}} \sum_{w_i \in \mathbf{s}, w_i == <mask>} \log \pi^{tok}_{\theta_L}(t_i|w_i, \mathbf{s}, \mathbf{d})$        ▷ Fill placeholders
12:    $\mathbf{s} \leftarrow \mathcal{E}(\mathbf{s}, \mathbf{tok\_index})$
13: end while
14: $\mathbf{d} \leftarrow \mathrm{documentUpdate}(\mathbf{d}, \mathbf{s})$
```



(a) Transformer blocks extract the sentence representations which are used by high level policy classifiers. Suppose that the update policy predicts to refine sentences 1 and 3.



(b) The input to the low level transformer is the concatenated sentences identified by the high level update policy.

Figure 4.1: The illustration of the proposed model for the update iteration. The same architecture can be applied for different tasks with specific classifiers. We have omitted attention from transformer blocks for simplicity. p stands for position embedding wheras s is for segment embedding.

4.3 Hierarchical Transformer

4.3.1 Architectures

Our model is based on the Transformer encoder-decoder architecture (Vaswani et al., 2017). We extract the hidden representations $(\mathbf{h_1}, ..., \mathbf{h_n})$ to make the policy predictions. We extract sentence representations by concatenating all sentences with a special <*sep*> token. The hidden states corresponding to these special tokens are then used as sentence representation by the policies. Along with position embeddings for individual tokens, we also introduce segment embeddings for sentences, which identify the position of a sentence in a document. We show the illustration of the proposed model in Figure 4.1.

4.3.2 Policy classifiers

We implement policies as classifiers whose prediction depend upon the hidden state representations generated by the transformer layers.

Reposition classifier: The reposition classifier gives a categorical distribution over the index of the input, where the input can be the representation of a sentence or a word. The input sequence is then repositioned accordingly. Along with reordering, this classifier can also perform deletion by predicting special delete token. This classifier is implemented as:

$$\pi_\theta^{rep}(r|\mathbf{s}_i, \mathbf{d}) = \mathrm{softmax}(\mathbf{h}_i \cdot [\mathbf{b}, \mathbf{e_1}, ..., \mathbf{e_n}])$$

for $i \in \{1..n\}$ where $\mathbf{e}$ can be the embedding of a sentence or token and $\mathbf{b} \in \mathbb{R}^{\mathbf{d_{model}}}$ is a special token to predict deletion. Note that in case of low level program, we also condition on the complete document. This is done by having cross-attention on the hidden representation of the sentences.

Insertion classifier: The high level insert classifier scans over the consecutive sentences and make a binary decision to insert or not.

$$\pi_\theta^{ins}(p|\mathbf{s}_i, \mathbf{d}) = \mathrm{softmax}([\mathbf{h}_i; \mathbf{h}_{i+1}] \cdot \mathbf{A})$$

for $i \in \{1..n\}$ and $\mathbf{A} \in \mathbb{R}^{2 \times \mathbf{d_{model}}}$ is a parameter. The low level insert classifier is made up of placeholder insertion followed by token insertion. The placeholder classifier predicts the number of tokens to be inserted at every consecutive position pairs, by casting the representation to a categorical distribution

$$\pi_\theta^{ins}(p|w_i, \mathbf{s}, \mathbf{d}) = \mathrm{softmax}([\mathbf{h}_i, \mathbf{h}_{i+1}] \cdot \mathbf{B})$$

for $i \in \{1..n\}$ and $\mathbf{B} \in \mathbb{R}^{(k_{max}+1) \times (2\mathbf{d_{model}})}$ is a parameter. Following (Gu et al., 2019), k_{max} is 255. The token classifier then fills the placeholders

$$\pi_\theta^{tok}(t|w_i, \mathbf{s}, \mathbf{d}) = \mathrm{softmax}(\mathbf{h_i} \cdot \mathbf{C})$$

for $i \in \{1..n\}$ where w_i is a placeholder and $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}| \times \mathbf{d_{model}}}$ is a parameter.

Update classifier: The update classifier is only present in the high level program. It scans over the sentences and make a binary decision to update a given sentence

$$\pi_\theta^{upd}(u|\mathbf{s}_i, \mathbf{d}) = \mathrm{softmax}(\mathbf{h}_i \cdot \mathbf{D})$$

for $i \in \{1..n\}$ and $\mathbf{D} \in \mathbb{R}^{2 \times \mathbf{d_{model}}}$ is a parameter.

4.3.3 Noise

There are no large-scale labeled training datasets for document-level rewriting. Accordingly we train on a synthetic dataset involving corrupting text documents and learning to restore them to their original state. To generate artificial broken text, we apply transformation techniques both at the sentence and word level and then learn to reverse the transformation to recover the original document. The techniques we use at the sentence level include: i) *sentences reordering* where

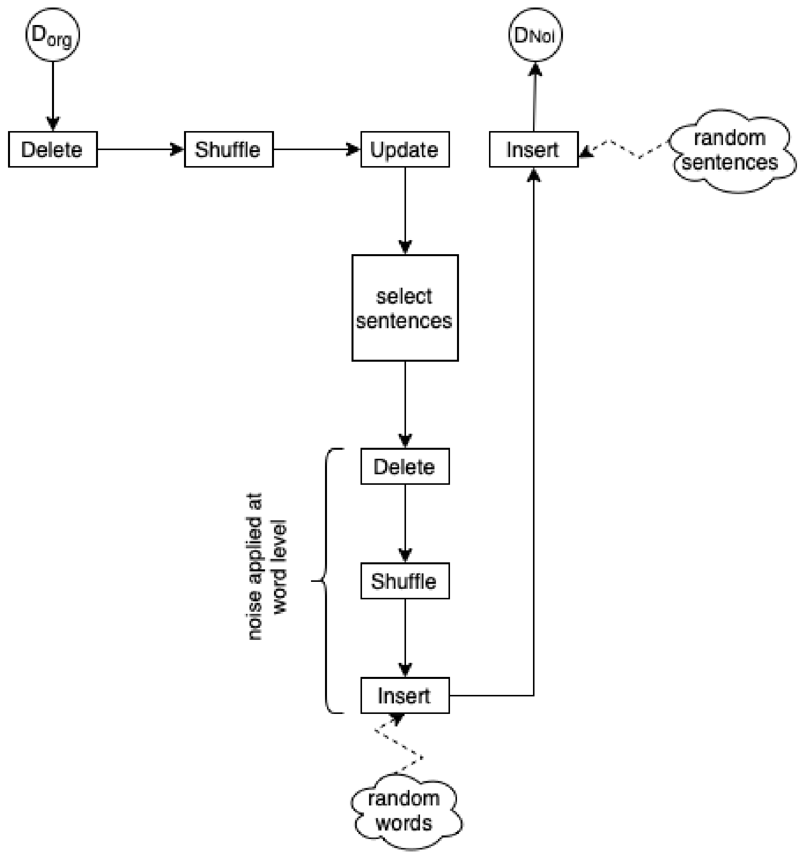


Figure 4.2: The figure shows the noising process to corrupt the original document. Noise is applied to the original document D_{org} with delete, shuffle, update and insert. The noise update applies noise on the selected sentences. We have a pre-built resource of random words and sentences built from the training set that is used in word and sentence level insertion. Each noise action is applied if the probability of selection is greater than 0.5. As some actions are skipped, the noising process generates different trajectories of noise.

sentences are randomly shuffled and/or deleted; ii) *sentence insertion* where a totally independent sentence is inserted into the source. iii) *sentence update* where a sentence is slightly modified. For the lower-level transformation, we apply: i) *word insertion* that we insert a random word from another pre-defined vocabulary into the source. ii) *shuffle and delete* where we shuffle and delete some words. Each transformation is applied if the probability of selection is greater than 0.5. We use uniform probability distribution and applied noise in the order *Delete, Shuffle, Update, Insert*.

The noising process is shown in Figure 4.2. The corruption process start by deleting random sentences from the original document. The resulting document is shuffled and then sentences are randomly selected to be updated. For each sentence to be updated, the words are deleted, shuffled and randomly inserted from a pre-defined vocabulary. After the update, locations are randomly selected in the document for sentence insertion. For each randomly selected position in the document, another sentence from a pre-defined sentence bank is inserted. The final corrupted document is then used with the original document as a source and target pair where source is the corrupted document and target is the original document.

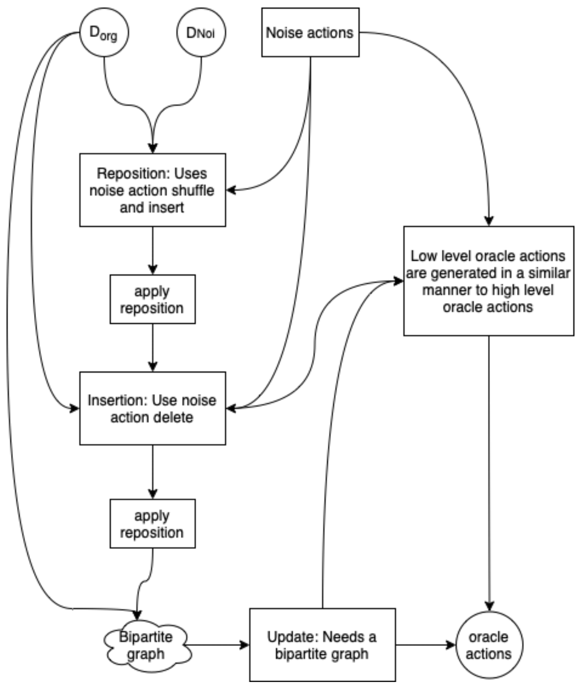
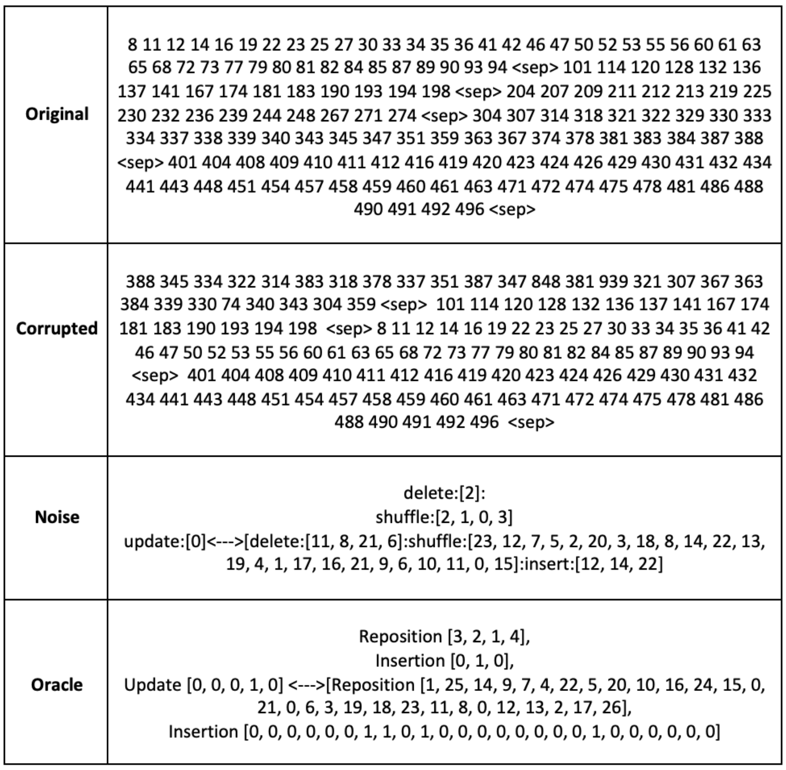

D_{org}
D_{Noi}
Noise actions
Reposition: Uses noise action shuffle and insert
apply reposition
Insertion: Use noise action delete
apply reposition
Low level oracle actions are generated in a similar manner to high level oracle actions
Bipartite graph
Update: Needs a bipartite graph
oracle actions

Figure 4.3: The figure shows the process by which the oracle actions are generated. Reposition uses shuffle and insert noise action to build its output. Sentences deleted by the noise are inserted back and are use for building the output for insert classifier. Finally a bipartite graph is build and the alignments are generated to get the output for update classifier. The same process is applied at the low level to get the oracle output at word level.

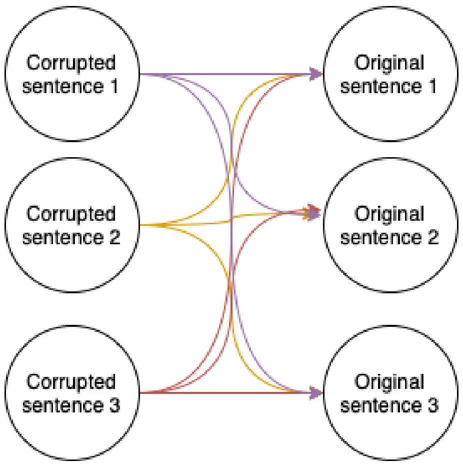
4.3.4 Oracle

Expert policy actions $\mathbf{a}^*$ are created by reversing the noise in the data. This is done by keeping track of the noise actions that have been used to create a corrupted output. In order to get alignment among sentences, we create a bipartite graph where the nodes are the sentences and the edge weight is the Levenstein distance between those sentences. We use max-flow min-cut algorithm to get the alignment (Dantzig & Fulkerson, 2003). The oracle action generation process along with sample output are shown in Figures 4.3 and 4.4 respectively. The oracle process starts using the outputs of noise actions shuffle and insert. These two actions inform the oracle of what needs to be deleted and repositioned. The reposition action is applied on the corrupted document. The resulting document is then used to get the oracle outputs for insertion. The noise action delete inform the oracle of the deletions performed by the corruption process. The oracle action is applied and the resulting document is used for creating update oracle action. The update oracle action creates a bipartite graph between the corrupted document and the original document as shown in Figure 4.5. It then runs a max-flow min-cut algorithm to get the alignments. Alignment score which are not zero indicate a sentence that need to be update. The low level oracle actions are generated similarly to the high level actions. The noise sequence used for corrupting the sentence is used to get the oracle low level actions. Figure 4.4 shows original and corrupted document along with the corruptions sequence that resulted in the given output sequence. The corresponding oracle actions are also mentioned. Applying the oracle actions on the corrupted document will revert it back to the original document.

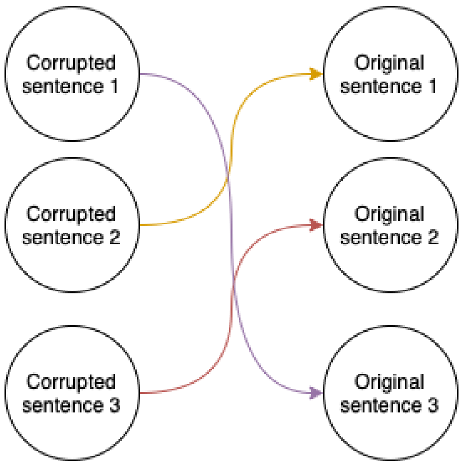


Original	8 11 12 14 16 19 22 23 25 27 30 33 34 35 36 41 42 46 47 50 52 53 55 56 60 61 63 65 68 72 73 77 79 80 81 82 84 85 87 89 90 93 94 <sep> 101 114 120 128 132 136 137 141 167 174 181 183 190 193 194 198 <sep> 204 207 209 211 212 213 219 225 230 232 236 239 244 248 267 271 274 <sep> 304 307 314 318 321 322 329 330 333 334 337 338 339 340 343 345 347 351 359 363 367 374 378 381 383 384 387 388 <sep> 401 404 408 409 410 411 412 416 419 420 423 424 426 429 430 431 432 434 441 443 448 451 454 457 458 459 460 461 463 471 472 474 475 478 481 486 488 490 491 492 496 <sep>
Corrupted	388 345 334 322 314 383 318 378 337 351 387 347 848 381 939 321 307 367 363 384 339 330 74 340 343 304 359 <sep> 101 114 120 128 132 136 137 141 167 174 181 183 190 193 194 198 <sep> 8 11 12 14 16 19 22 23 25 27 30 33 34 35 36 41 42 46 47 50 52 53 55 56 60 61 63 65 68 72 73 77 79 80 81 82 84 85 87 89 90 93 94 <sep> 401 404 408 409 410 411 412 416 419 420 423 424 426 429 430 431 432 434 441 443 448 451 454 457 458 459 460 461 463 471 472 474 475 478 481 486 488 490 491 492 496 <sep>
Noise	delete:[2]: shuffle:[2, 1, 0, 3] update:[0]<--->[delete:[11, 8, 21, 6]:shuffle:[23, 12, 7, 5, 2, 20, 3, 18, 8, 14, 22, 13, 19, 4, 1, 17, 16, 21, 9, 6, 10, 11, 0, 15]:insert:[12, 14, 22]
Oracle	Reposition [3, 2, 1, 4], Insertion [0, 1, 0], Update [0, 0, 0, 1, 0] <--->[Reposition [1, 25, 14, 9, 7, 4, 22, 5, 20, 10, 16, 24, 15, 0, 21, 0, 6, 3, 19, 18, 23, 11, 8, 0, 12, 13, 2, 17, 26], Insertion [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

Figure 4.4: The figure shows an example output of oracle generated actions. The noise is applied to the original document to get the corrupted document. Noise is applied at the sentence level as well as at the word level. Oracle actions are then generated to reverse the inserted noise.



(a) Bipartite graph is built between the sentences in corrupted document and the original document. The weights are Levenstein distance between the sentences.



(b) The alignments after running the max-flow min cut algorithm. If the distance between sentences is not zero. That particular sentence need to be updated.

Figure 4.5: The figure shows a bipartite graph created in order to generate the oracle update action. The graph is created between the sentences in corrupted and original document.

4.3.5 Training

Training involves learning a model to imitate the expert policy. We design roll-in policy such that each classifier is trained on the output of the other classifier. This reduces exposure bias, as the model is trained on conditions it will encounter at decoding. The algorithm for training is shown in Algorithm 3. It start by sampling a training pair consisting of a corrupted sentence along with the actual target without the noise. Low and high level oracle actions are generated for the given training example. These actions correspond to the true output of the low and high level classifiers. The corrupted document goes through the transformer layers. The hidden states corresponding to the sentence representations are then used by the reposition classifier to predict its output. The output of the classifier along with the actual output from the oracle are used for log-loss calculations. The true reposition output is applied to the corrupted sentence and the resulting document is used by the high level insertion classifier. The insertion classifier calls the low level program which learns to generate the sentence conditioned on the whole document. The true insertion classifier output is applied to the document and the resulting document is used for training the update classifier. The update classifier trains a low level program. Instead of starting with a blank sentence as in the insertion case, the low level program starts with the initial corrupted sentences. The function buildFrame is responsible for passing the correct initial input to the low level program. The objective function is the product of decisions made during the generation process. It is the loses incurred by both the high level and low level program and is shown on line 14.

4.4 Experiments

4.4.1 Experimental Setup

Data sets. We conduct experiments on synthetically generated dataset consisting of sorted sequences of numbers. These are shuffled and the challenge is to restore them into numerical sorted order. This task mimic language problems in coherence where sentences are shuffled and the task is to put the sentence in the right order such that coherence of the document is maximised. Each document contains 5 - 10 sentences and each sentence has between 20 to 100 tokens. The document is sorted in numerical order with tens coming before hundreds (see Figure 4.4). The numbers lie between 1 and 1000. We generated 300K such pairs for training consisting of unsorted sequence as input and sorted sequence as output.

We further use real world datasets including ROC stories (Mostafazadeh et al., 2016), consisting of multiple 5 lines stories to check the capabilities of our model. We also conducted experiments on Multi-news and DUC-2004 datasets. Multi-news (Lebanoff, Song, & Liu, 2018) is a large-scale dataset for summarization whereas DUC-2004 (Over & Yen, 2004) is a benchmark dataset in multi-document summarization. The documents are truncated to 1500 tokens. We merge the two datasets to create a training corpus consisting of approximately 1.8M pairs and exclude 15K as test and validation sets each. To generate our input and output pairs, we inserted

Algorithm 3 Training for Hierarchical Levenshtein Transformer

```
Require: Training data $\mathcal{T}$, Model policy: $\pi_\theta$, Expert policy: $\pi_*$
 1: while Maximum training steps reached do
 2:     $(\mathbf{d}, \mathbf{d}_*) \sim \mathcal{T}$                                  ▷ Sample a training pair

 3:     $\mathbf{repH}^*, \mathbf{insH}^*, \mathbf{updH}^* \leftarrow \pi^H_*(\mathbf{d}, \mathbf{d}_*)$          ▷ Get oracle actions
 4:     $\mathbf{repL1}^*, \mathbf{insL1}^*, \mathbf{tokL1}^*, \mathbf{repL2}^*, \mathbf{insL2}^*, \mathbf{tokL2}^* \leftarrow \pi^L_*(\mathbf{d}, \mathbf{d}_*)$

 5:     $\mathcal{L}^{rep}_{\theta_H} \leftarrow -\sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{rep}_{\theta_H}(repH^*_i | \mathbf{s}_i, \mathbf{d})$
 6:     $\mathbf{d} \leftarrow \mathrm{applyAction}(\mathbf{d}, \mathbf{repH}^*)$

 7:     $\mathcal{L}^{ins}_{\theta_H} \leftarrow -\sum_{\mathbf{s}_i, \mathbf{s}_{i+1} \in \mathbf{d}} \log \pi^{ins}_{\theta_H}(insH^*_i | \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{d})$
 8:     $\mathbf{s} \leftarrow \mathrm{buildFrame}(\mathbf{insH}^*, \mathbf{d})$

 9:     $\mathcal{L}^{rep1}_{\theta_L} \leftarrow -\sum_{w_i \in \mathbf{s}} \log \pi^{rep}_{\theta_L}(repL1^*_i | w_i, \mathbf{s}, \mathbf{d})$          ▷ Low Level
10:     $\mathbf{s} \leftarrow \mathrm{applyAction}(\mathbf{s}, \mathbf{repL1}^*)$
11:     $\mathcal{L}^{ins1}_{\theta_L} \leftarrow -\sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi^{ins}_{\theta_L}(insL1^*_i | w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$
12:     $\mathbf{s} \leftarrow \mathrm{applyAction}(\mathbf{s}, \mathbf{insL1}^*)$
13:     $\mathcal{L}^{tok1}_{\theta_L} \leftarrow -\sum_{w_i \in \mathbf{s}, w_i = <mask>} \log \pi^{tok}_{\theta_L}(tokL1^*_i | w_i, \mathbf{s}, \mathbf{d})$
14:     $\mathbf{d} \leftarrow \mathrm{applyAction}(\mathbf{d}, \mathbf{insH}^*)$

15:     $\mathcal{L}^{upd}_{\theta_H} \leftarrow -\sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{upd}_{\theta_H}(updH^*_i | \mathbf{s}_i, \mathbf{d})$
16:     $\mathbf{s} \leftarrow \mathrm{buildFrame}(\mathbf{updH}^*, \mathbf{d})$

17:     $\mathcal{L}^{rep2}_{\theta_L} \leftarrow -\sum_{w_i \in \mathbf{s}} \log \pi^{rep}_{\theta_L}(repL2^*_i | w_i, \mathbf{s}, \mathbf{d})$          ▷ Low Level
18:     $\mathbf{s} \leftarrow \mathrm{applyAction}(\mathbf{s}, \mathbf{repL2}^*)$
19:     $\mathcal{L}^{ins2}_{\theta_L} \leftarrow -\sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi^{ins}_{\theta_L}(insL2^*_i | w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$
20:     $\mathbf{s} \leftarrow \mathrm{applyAction}(\mathbf{s}, \mathbf{insL2}^*)$
21:     $\mathcal{L}^{tok2}_{\theta_L} \leftarrow -\sum_{w_i \in \mathbf{s}, w_i = <mask>} \log \pi^{tok}_{\theta_L}(tokL2^*_i | w_i, \mathbf{s}, \mathbf{d})$

22:     $\theta \leftarrow \theta - \lambda \nabla [\mathcal{L}^{rep}_{\theta_H} + \mathcal{L}^{ins}_{\theta_H} + \mathcal{L}^{upd}_{\theta_H} + \mathcal{L}^{rep1}_{\theta_L} + \mathcal{L}^{ins1}_{\theta_L} + \mathcal{L}^{tok1}_{\theta_L} + \mathcal{L}^{rep2}_{\theta_L} + \mathcal{L}^{ins2}_{\theta_L} + \mathcal{L}^{tok2}_{\theta_L}]$
23: end while
```

noise in the output sequences as outlined in Section 4.3.3. The output consists of the gold summary with the inserted noise. The gold summaries are not generated via abstractive or extractive summarization techniques.

Previous work by citeAgu2019levenshtein, xu2020editor tested their approaches on machine translation and automatic post-editing. This work is different as we are interested in long text generation. Therefore,, we cannot apply our model on these single sentence tasks. The work done by citeAgu2019levenshtein, xu2020editor is used as a component in our approach.

Evaluation Metrics. Rouge (Hovy, Lin, Zhou, & Fukumoto, 2006), an automatic evaluation metric, is commonly used in Summarization to evaluate the quality of summaries. We use Rouge-l, Rouge-2 and Rouge-L which measure unigram-overlap, bigram-overlap, and the longtest common sequence respectively between system and actual summaries. Synthetic and ROC stories are evaluated with BLEU score (Papineni et al., 2002). For both Rouge and BLEU score the higher the value the better is the result.

Baselines. We compare three models: i) *Copy*: the original text is copied without any change. This establishes the lower bound for the task. ii) *Transformer*: a vanilla Transformer (Vaswani et al., 2017) is used to generate a sequence of text by reconstructing the source text. Without explicit editing guidance, we have little control over its generation process. iii) *Levenshtein Transformer (LevT)*: LevT is a semi-autoregressive model for parallel sentence-level sequence generation (Gu et al., 2019). It refines a given sequence in an iterative manner with three operations, including *deletion*, *placeholder prediction* and *token prediction*. The iteration terminates when a certain stopping criterion is met. iv) *Editor transformer*: It is similar to the LevT, with the exception that it introduce a reposition operator instead of the deletion operator (Xu & Carpuat, 2020). Along with deletion, a reposition operator can also rearrange words in a sentence.

Implementation Details. To train the our models, we follow most of the hyper-parameter settings in (Gu et al., 2019). Specifically for all models we use dropout of 0.3, num_heads of 8, layers as 6 model_dimension as 512 and hidden_dimension as 2048. We use weight decay of 0.01 as well as label smoothing of 0.1. Our models are implemented in open-source toolkit fairseq (Ott et al., 2019). The models are trained on 3 Nvidia V100 GPU for 200K updates with a batch size of 64K tokens. We used adam optimiser in our training and adopt fastbpe (Sennrich, Haddow, & Birch, 2015). We reproduce results from baseline models by training our own models on their publicly available code.

4.4.2 Results

We present results from above mentioned datasets to show different capabilities of our model. The experiments on synthetic dataset are designed to test whether different components of the model improve different document level phenomena such as whether the high level reposition module

	Synthetic
Transformer	40.76
LevT	43.11
Editor	43.65
Ours	73.86

Table 4.1: Results of running experiments on Synthetic with only shuffling noise. We report accuracy of the model in repositioning the sentence correctly.

improves coherence [1]. The experiments on ROC stories tests length flexibility and the behaviour of our model on a real world dataset. Finally, experiments on summarization dataset are designed to show one potential application area for our model as a document level post editing system.

4.4.2.1 Synthetic dataset

Noise: A document can be viewed as a sequence of segments, each of which is cohesive in its content and functions. A document, for example, related to biographies should contain segments arranged such that the whole document is coherent and cohesive. Cohesion is a property shared by well structured discourse and can be used to evaluate document level tasks. One difficulty with evaluating metrics of discourse coherence is that human generated text usually meet some minimal threshold of coherence. For this reason much of the research in measuring coherence has focused on synthetic data. A typical setting is to permute the sentences of a text and then determine whether the original sentence ordering scores higher according to the proposed coherence measure. Therefore we use our synthetic data related to sorting to measure the coherence capabilities of our proposed model. We inject only shuffling noise in the training data, that is the sentences in the document are only shuffled. No other noise operation is performed. We measure coherence as the accuracy of the model to reposition the sentence correctly.

Results The main results of conducting experiments on synthetic dataset are shown in Table 4.1. We report the accuracy of the model in repositioning the sentences correctly in the table. The accuracy of our model is higher and better than the baselines. The high accuracy is due to introducing a high level program which is better able to capture discourse level phenomena by using sentence representations. The baseline models use individual tokens/words and find it difficult to capture long range document-level phenomena correctly. The results also show that there is no need to apply editing operations at the word level if they can be rectified by the high level program.

Discussion: We train the model on a dataset generated by inserting noise as shown in Section 4.3.3. We compare our model with other baselines and report the BLEU score in Table 4.2. Transformer performed well across all the models, indicated by the highest BLEU score in the table. We

[1]Coherence has been mentioned for motivation of the task which is long text generation. Instead of using some metric for coherency, we can indirectly measure coherence by analysing the BLEU score of those shuffled sentences. If the reposition classifier can correctly reposition the sentences then we can conclude that the coherence of the document has increased. This is further mentioned in Section 4.4.2.1Synthetic dataset/noise.

	Synthetic	ROC-Stories
Copy	23.59	28.82
Transformer	30.17	35.72
LevT	22.42	25.29
Editor	22.78	25.89
Ours	20.63	23.10

Table 4.2: Results of running experiments on Synthetic and ROC-stories test dataset. We report the BLEU score in the table which is higher the better.

	Noise		
Classifier	20	50	80
Reposition	46.3	41.1	37.9
Delete	45.6	41.7	39.6
Insert	30.3	27.8	21.6
Update	43.6	40.2	36.8

Table 4.3: The table shows the accuracy of each high level classifier on synthetic dataset. Reposition and Delete accuracy are from the same reposition classifier with different type of noise.

conjecture that the transformer model was better able to capture the underlying distribution due to its non-Markovian nature. LevT and Editor performed comparably but not as good as the transformer which is expected due to its semi-autoregressive nature. Semi-autoregressive model lags behind autoregressive models due to loss of conditional context. Finally our model lags behind all the mentioned models.

As our model is made up of various components consisting of four high level classifiers and three low level classifiers along with various transformer layers, we individually look at the performance of each classifier by measuring its accuracy. We use the trained model on complete noise. Then we build test sets by inserting one type of noise targeted at a particular classifier. For example the document is shuffled to measure the performance of reposition classifier. Sentences are inserted to measure the performance of deletion classifier. The Deletion classifier is part of reposition classifier therefore in the results, reposition accuracy shows the performance on a shuffled document whereas deletion accuracy shows the performance of reposition classifier on a document with random sentences inserted. We measure the accuracy of classifier as we increase the noise ratio, which determine how many sentences are either shuffled, deleted, inserted or updated. The results are shown in Table 4.3. As the noise increases, the accuracy across all the classifiers goes down indicating further training is required to make the classifiers more robust to the noise. Although, we see a decrease in accuracy, the high level classifiers were able to improve the coherence in most of the documents with reposition, deletion and update accuracy remaining high even as the noise ratio increases. The insert classifier accuracy remain low. The classifier makes a decision by considering the hidden state of two adjacent sentences. We conjecture that this may confuse the classifier as there is insufficient context and adding more context may improve improve its performance.

Next we test, the performance of our model as we increase the document length. We compare the performance with other baselines by dividing the dataset into buckets of different lengths. Table 4.4 shows the BLEU score of models across different length buckets. We observe transformer model performs well across all the buckets, although its performance does drop on longer

	Bucket				
	5-100	101-200	201-300	301-400	401-500
Copy	34.6	32.3	28.6	25.4	23.5
Transformer	39.3	35.5	31.6	27.5	25.3
LevT	35.6	32.5	28.2	24.6	21.3
Editor	35.1	32.7	27.6	24.1	21.0
Ours	36.3	32.9	26.3	22.6	19.2

Table 4.4: The table shows the BLEU score of models with different input length sizes. Each bucket size refers to the number of characters.

	Time	Standard Deviation
Transformer	245s	$\pm$0.12
LevT	157s	$\pm$0.06
Editor	149s	$\pm$0.05
Ours	187s	$\pm$0.05

Table 4.5: Table shows the time for running 100 test examples. No batching is used to mimic the real world scenario.

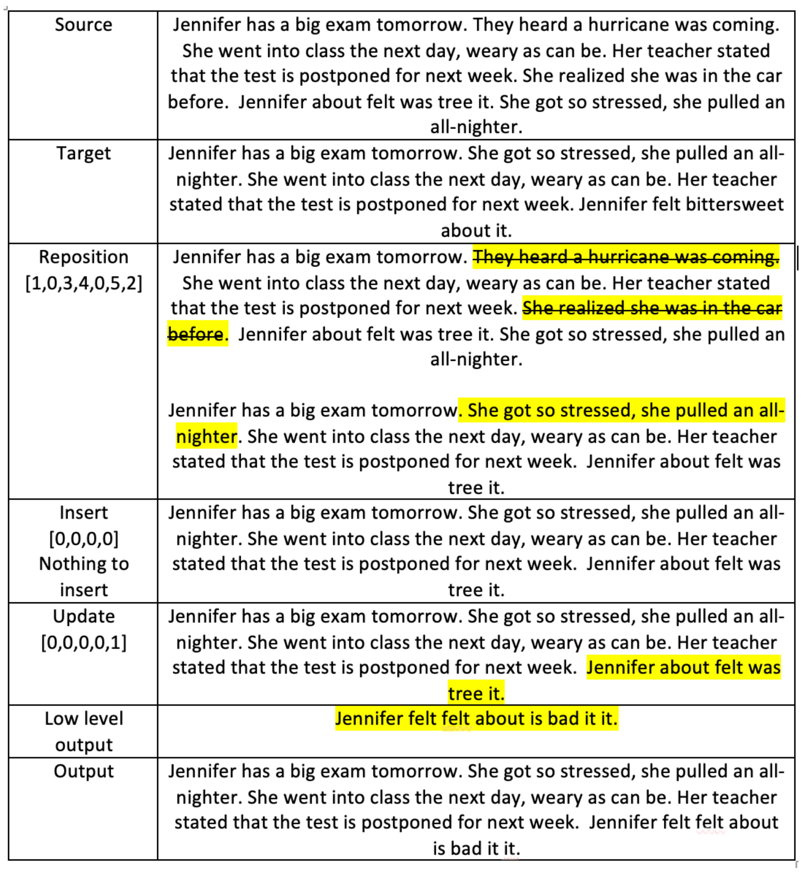
sequences. Our model improves BLEU score on shorter sequences as shorter sequences compared to other semi-autoregressive models. We conjecture that as the shorter sequences do not have long range dependencies therefore iterative improvement by our model was better able to improve the sentence. As the length of sequence increase our model find it difficult to refine and improve the noisey input. This points to improving the model by specifically targeting long sequences in training. Techniques such as curriculum learning where the model is progressively trained on harder examples may improve model performance.

We also measure the decoding time for different models. All model are evaluated on the same harware architecture. Table 4.5 shows that the decoding time for transformer is higher compared to other semi-autoregressive models. The decoding time of our model is higher than the other semi-autoregressive models due to the complexity introduced by the high level program.

4.4.2.2 ROC stories

The main results of experiments on ROC stories are shown in Table 4.2. We see a similar pattern as compared to synthetic dataset where transformer model outperformed the other models.

Discussion: Our model has the capability to change the sequence length. We show this by showing an example output from ROC stories dataset in Figure 4.6. The deletion and update operator modified the length of the sequence dynamically. The insert operator also has the capability to make changes to the document length. This is in contrast to the transformer model where the length of the sequence increases monotonically and cannot be modified once the output has been generated.



Source	Jennifer has a big exam tomorrow. They heard a hurricane was coming. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. She realized she was in the car before. Jennifer about felt was tree it. She got so stressed, she pulled an all-nighter.
Target	Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. Jennifer felt bittersweet about it.
Reposition [1,0,3,4,0,5,2]	Jennifer has a big exam tomorrow. ~~They heard a hurricane was coming.~~ She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. ~~She realized she was in the car before~~. Jennifer about felt was tree it. She got so stressed, she pulled an all-nighter. Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. Jennifer about felt was tree it.
Insert [0,0,0,0] Nothing to insert	Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. Jennifer about felt was tree it.
Update [0,0,0,0,1]	Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. Jennifer about felt was tree it.
Low level output	Jennifer felt felt about is bad it it.
Output	Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week. Jennifer felt felt about is bad it it.

Figure 4.6: The figure shows sample output from ROC stories dataset. Observe that the editing operators can modify the length of the document. The yellow highlights shows the part being edited by the model.

	Multi-News			DUC-2004		
	R-1	R-2	R-L	R-1	R-2	R-L
Copy	42.32	13.28	37.86	36.30	8.47	32.52
Transformer	40.62	12.42	36.37	35.4	7.78	31.71
LevT	25.93	8.59	28.95	23.45	4.89	25.12
Editor	25.56	8.13	28.33	23.17	4.21	25.01
Ours	21.67	5.89	24.03	18.22	2.17	20.87

Table 4.6: Experiment Results on Multi-News and DUC2004 dataset

4.4.2.3 Summarization

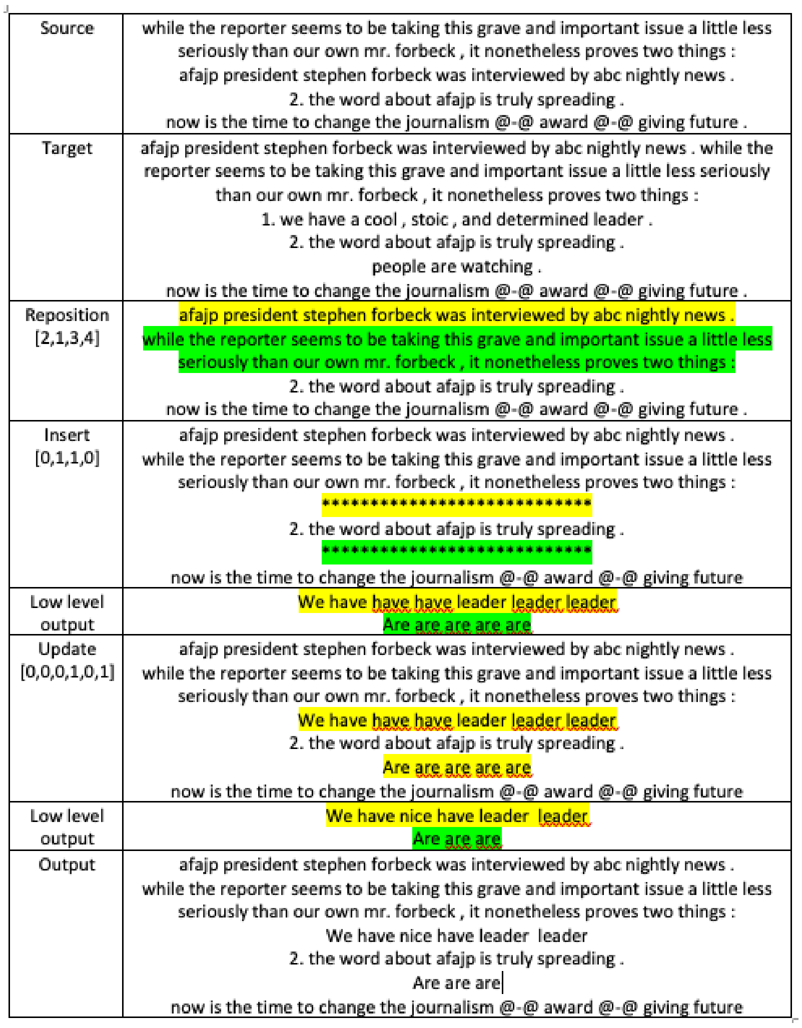
The main results for summarization are shown in Table 4.6. The best result is obtained by copy across both dataset indicating that post editing of long sequences may hurt its quality. Copy consist of output from SummPip system (J. Zhao et al., 2020). SummPip uses graph clustering to find relevant sentences which are then used to generate the summary. Among other models, the Vanilla transformer performed better showing a strong bias present in the languages for autoregressive monotone generation. Levenshtein and the Editor transformer performed comparably, whereas as our model showed no improvement over the baselines.

Discussion: Our model can be employed as a post-editing document level system. The high level program has the capability to capture the underlying document structure and can provide context and structure aware information which can then improve the coherence of the document. Figure 4.7 shows that most of the mistakes are made by the low level program. This is acceptable as the low level program has to generate sequence semi-autoregressively that means less conditional dependencies in the output variables. The low level program also has to take a larger document level context into account. This further confuses the model as it becomes harder for the model to determine which part it should put its attention. We conjecture that a carefully designed attention module may mitigate this problem and improve results.

4.4.3 Discussion on negative results

Our model lags behind the baselines but has shown promise and can be improved further. In this section, we outline various ways to improve the results of our model:

Evaluation metrics sensitivity towards document level ordering: We measure the sensitivity of our evaluation metrics towards capturing sentence reordering. We permuted sentences in a document and measure the metric's mean and standard deviation. The results in Table 4.7 shows the inadequacy of using these metrics (BLEU, ROUGE) towards document level phenomenons. The almost perfect BLEU and ROGUE score shows that the metrics were unable to capture sentence permutations and therefore document level phenomenons. It further suggests that the two levels should be trained separately. A low level program should be initially trained and tested on BLEU scores since it is responsible for word generation. The high level program is then trained while keeping the low level program frozen.



Source	while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : afajp president stephen forbeck was interviewed by abc nightly news . 2. the word about afajp is truly spreading . now is the time to change the journalism @-@ award @-@ giving future .
Target	afajp president stephen forbeck was interviewed by abc nightly news . while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : 1. we have a cool , stoic , and determined leader . 2. the word about afajp is truly spreading . people are watching . now is the time to change the journalism @-@ award @-@ giving future .
Reposition [2,1,3,4]	afajp president stephen forbeck was interviewed by abc nightly news . while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : 2. the word about afajp is truly spreading . now is the time to change the journalism @-@ award @-@ giving future .
Insert [0,1,1,0]	afajp president stephen forbeck was interviewed by abc nightly news . while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : ***************************** 2. the word about afajp is truly spreading . ***************************** now is the time to change the journalism @-@ award @-@ giving future
Low level output	We have have have leader leader leader Are are are are are
Update [0,0,0,1,0,1]	afajp president stephen forbeck was interviewed by abc nightly news . while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : We have have have leader leader leader 2. the word about afajp is truly spreading . Are are are are are now is the time to change the journalism @-@ award @-@ giving future
Low level output	We have nice have leader leader Are are are
Output	afajp president stephen forbeck was interviewed by abc nightly news . while the reporter seems to be taking this grave and important issue a little less seriously than our own mr. forbeck , it nonetheless proves two things : We have nice have leader leader 2. the word about afajp is truly spreading . Are are are now is the time to change the journalism @-@ award @-@ giving future

Figure 4.7: The figure shows sample output from summarization dataset.

	Mean	Standard Deviation
Synthetic	97.84	$\pm$0.05
ROC stories	98.94	$\pm$0.03
Multi-News	97.95	$\pm$0.05
DUC-2004	97.73	$\pm$0.05

Table 4.7: Sensitivity of metrics towards capturing sentence reordering. For synthetic and ROC stories, we report the BLEU score. For Multi-news and DOC-2004, we report the R1 score. Mean and standard deviation is measured over 10 runs.

Distilled Dataset: Semi/non-autoregressive models struggle to achieve quality similar to autoregressive models. As the dependencies are broken, it become difficult for the model to generalise across multimodal dataset. Multimodal dataset arises because of competing outputs. There can be multiple outputs for a given input. Each output corresponds to one of the modes of conditional probability distribution of the target given the source; hence, this conditional distribution is multi-modal, i.e. it has multiple modes. For example, consider an English source sentence like "Thank you". This can be accurately translated into German as any one of "Danke", "Danke schon", or "Vielen Dank". A conditionally independent distribution such as the one arising in non-autoregressive models will allow "Danke Dank" and "Danke Vielen" as possible translations. Therefore output from semi/non-autoregressive models contain syntatic and semantic errors (Gu, Bradbury, et al., 2017). The situation is further aggravated when the sequences are long. Distilled dataset has been found useful in dealing with multomodality problem in non-autoregressive modals (C. Zhou et al., 2019). Instead of using the actual output, the outputs generated from an autoregressive teacher model are used with the input sequence. It is not obvious as to how we can use distilled data in our model. One way is to insert the noise in distilled dataset to get input sequences. Another way is to use curriculum learning (Bengio, Louradour, Collobert, & Weston, 2009), starting with distilled dataset and then moving to harder actual examples.

Better Training: Pre-training and fine-tuning approach has been found useful in various tasks. Our model consist of various components including classifiers at two levels. These classifiers can be individually pre-trained. Once the pre-training step is done, the whole model can be fine tuned for better model generalisation.

Use of Autoregressive model: The low level program is responsible for word generation. Due to the inherent left to right generation bias, autoregressive models have shown better results in our experiments. We can take advantage of this bias by using autoregressive model as a low level program but this can lead to longer decoding times.

Attention Mechanism: Wider context have been shown to improve results for document level tasks (Kim, Tran, & Ney, 2019). Designing an attention mechanism such that more attention is given to the sentences around the given sentence than those far away in the document can improve results. This can be done by having more attention heads for the near context then the far away context.

4.5 Summary

We present a hierarchical document generation model, that is capable of revising and editing its generated text thus bringing it closer to human-level intelligence. Although results showed that our approach lags behind the baselines, it did shed light into various problems present in semi-autoregressive models and long document generation. In the future, we will be incorporating these insights into our model to make it more robust. Along with incorporating the insights, we will also be conducting further experiments on other simple tasks such as text simplification.

Part III

Generation in Non-Autoregressive Models

Chapter 5

Exemplar Transformer

Current state-of-the-art translation models are autoregressive that is the generation is done word by word from left to right (Cho, Van Merriënboer, Gulcehre, et al., 2014). This autoregressive property becomes a bottleneck in taking full advantage of the the underlying parallel architecture therefore recently non-autoregressive translation (NAT) models have been proposed to make serial computation parallel (Gu, Bradbury, et al., 2017). NAT models generate all tokens in parallel as opposed to generating tokens autoregressively from left to right. This speeds up decoding but comes at a cost of inferior output quality. The complete conditional independence makes it harder for the model to approximate the true multimodal target distribution which exhibits strong correlation across time. Multimodal target distribution arises because of competing translations. As the tokens are generated in parallel, each word is conditioned only on the source sentence and is oblivious of what has been generated. This results in repeating or semantically incorrect translations. To overcome the loss of conditional information, non-autoregressive models are usually conditioned on latent variables along with the source sentence to inject prior knowledge about the underlying distribution. This prior knowledge takes the form of input sentence fertilities (Gu, Bradbury, et al., 2017), generative flows (X. Ma et al., 2019) or blank sentences (Ghazvininejad, Levy, Liu, & Zettlemoyer, 2019a).

We postulate that current priors based on fertilities and flow are simplistic and are not rich enough to capture complex interactions in the multimodal distribution of the target translation. Therefore the goal of this chapter is to show that an informative and effective prior based on data can improve non-autoregressive translation quality. We propose a data-based prior based on the most similar examples. More concretely, we build the prior based on the translations of the most similar source sentences to the test sentence. To effectively use the examples, we modify the attention mechanism such that attention heads are distributed between the similar examples and the sentence being translated. We conduct experiments on two benchmark datasets to show the effectiveness of our approach. Our results show comparable performance with state-of-the-art NAT models and considerable speed ups compared to autoregressive model.

5.1 Introduction

Neural sequence to sequence models have achieved state-of-the-art results on various tasks such as machine translation, summarization and automatic speech synthesis (Fan et al., 2021; El-Kassas et al., 2021; X. Tan, Qin, Soong, & Liu, 2021). These models are characterised by an encoder-decoder architecture where the input sequence is first transformed by the encoder to a set of hidden states. These hidden states are subsequently used by the decoder to generate the output sequence autoregressively from left to right (Sutskever et al., 2014). Previously, the autoregressive property allowed the use of recurrent neural networks (RNN) to parametrise these neural networks. RNNs are inherently good at processing sequential data but they are hard to parallelize and therefore slow to execute on modern hardware, which is optimised for parallel execution (Graves, 2013). Recently Transformers have been proposed as an alternative to RNNs (Vaswani et al., 2017). Transformers have not only shown better results on various sequence generation tasks, but with the use of self-attention, they allow training to be done in parallel thus taking advantage of the underlying hardware. Unfortunately, the decoding in Transformer is still sequential as the probability of emitting a token is conditioned on the previously generated token.

In order to accelerate decoding, non-autoregressive translation models (NAT) have been introduced. Instead of sequential generation in autoregressive translation, NAT models output the entire target sentence at once. Although this greatly improves the decoding latency, it comes with a huge cost in terms of translation quality (Du, Tu, & Jiang, 2021; Tokarchuk et al., 2021). Removing the sequential dependencies in the target sequence makes it challenging for the NAT model to capture a highly multimodal target distribution (C. Zhou et al., 2019). Multimodal target distribution arises because of alternative translations for a given sentence. As each token is generated conditioned on the source sentence, the generated token is unaware of what has been generated thus may contain tokens from competing translation. In particular, consider an English source sentence like “Thank you”. This can be accurately translated into German as any one of “Danke”, “Danke schon”, or “Vielen Dank”. A conditionally independent distribution will allow “Danke Dank” and “Danke Vielen” as possible translations. Therefore output from NAT models contain errors such as repetitive words and incomplete translation where the semantics of several tokens are not properly translated (Gu, Bradbury, et al., 2017).

To mitigate such performance degradation, NAT models usually employ a latent variable. These latent variables help to incorporate some light-weight sequential information into the non-autoregressive decoder and take the form of source sentence fertility (Gu, Bradbury, et al., 2017). That is the source sentence tokens are repeated multiple times based on their fertility, reordered source sentence based on target language (Ran et al., 2019), discrete variables (Shu, Lee, Nakayama, & Cho, 2020), generative flows (X. Ma et al., 2019) or blank sentences where a sequence made up of special blank tokens is iteratively refined (Ghazvininejad et al., 2019a). However using such latent variables not only complicates the training resulting in a badly trained model, they are also not rich and expressive enough to capture the complex interactions of tokens in the highly multimodal target distribution.

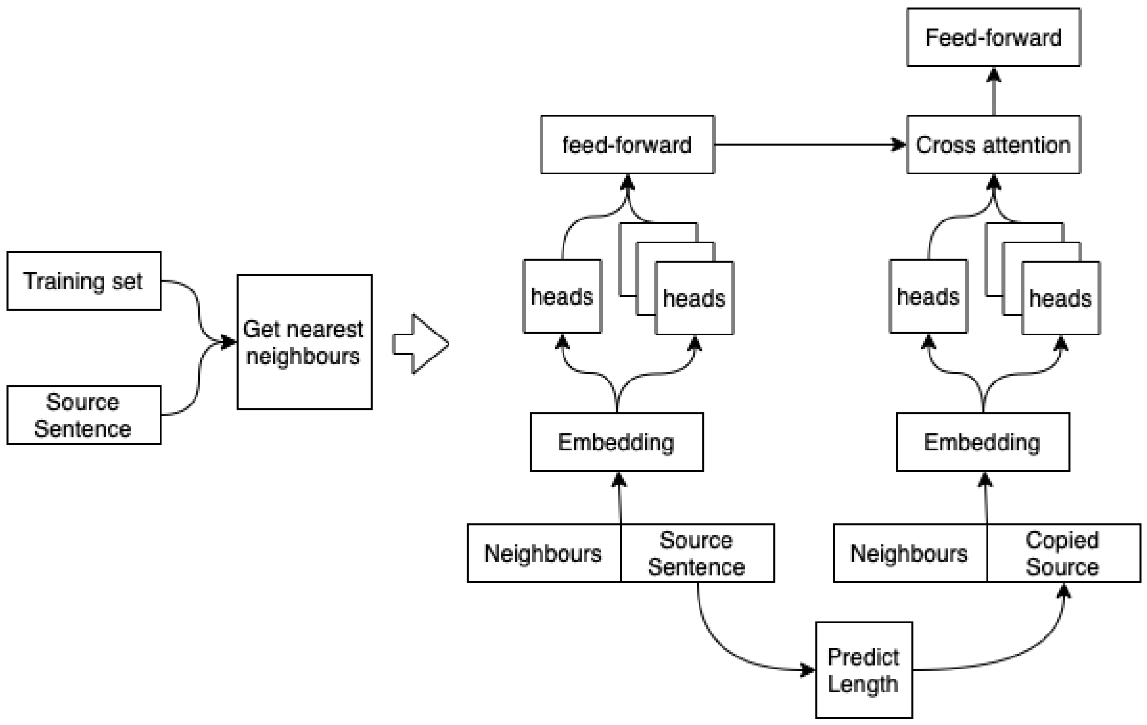


Figure 5.1: The figure shows the structure of the Exemplar Transformer. It consist of a pre-processing step where the nearest neighbours for the dataset are determined. These nearest neighbours are then used by the model to perform translation. The model divides the attention heads among the neighbours and the sentence being translated. This can be efficiently done by using a mask in the attention mechanism.

In this chapter, we therefore introduce an informative and effective prior based on data. We use a similarity metric to find similar examples from the training dataset to the instance being translated. The instances are first transformed into a n-dimensional vector representation. For each instance we determine its k-nearest neighbours. These nearest neighbours are then used as priors in our model as they can provide information such as translation of a phrase or semantically correct arrangement of words. In order to effectively capture the required information from the prior, we split the attention heads among the sentence being translated and the nearest neighbours. Heads are distributed such as more attention heads are given to the sentence being translated than the neighbours (Figure 5.1). This makes the model more robust as it allows to concentrate on the given sentence and not get confused by the neighbouring sentences.

We conduct experiments on two benchmark dataset WMT-14 en-de and WMT-16 en-ro. Results show that our approach is effective and can improve the translation quality against the NAT model by Gu, Bradbury, et al. (2017). Our approach shows lower but comparable performance against other NAT baselines. Further analysis of the model shows the effectiveness of using different similarity measures as well as the effectiveness of using different sentence representations. We also present a qualitative analysis of the model output.

5.2 Background

Neural sequence-to-sequence models generate an output sequence $\mathbf{y} = \{y_1, ..., y_m\}$ given an input sequence $\mathbf{x} = \{x_1, ..., x_n\}$. The conditional probability distribution $P_\theta(\mathbf{y}|\mathbf{x})$ is parameterized by θ which represents a Transformer network in this chapter. Factorization of this probability distribution give rise to autoregressive and non-autoregressive models. This section provides an introduction to these model families.

5.2.1 Autoregressive Neural Machine Translation

Autoregressive models factorize the joint probability of the output sequence $\mathbf{y}$ given the input sequence $\mathbf{x}$ into the product of probabilities over the next token in the sequence given the input sequence and previously generated tokens:

$$P(\mathbf{y}|\mathbf{x};\theta) = \prod_{t=1}^{M} P(y_i|\mathbf{y}_{<i}, \mathbf{x};\theta)$$

where $\mathbf{y}_{<i}$ indicates the partial translation and θ is a set of trainable parameters. The factor, $P_\theta(y_i|\mathbf{y}_{<i}, \mathbf{x})$, is implemented by function approximators such as RNNs and Transformers. This factorization takes the complicated problem of joint estimation over an exponentially large output space of outputs $\mathbf{y}$, and turns it into a sequence of tractable multi-class classification problems predicting y_i given the translation history. This allows the use of simple maximum log-likelihood training. The typical training objective is to maximize log-likelihood on a set of training examples $D = |\mathbf{x}^k, \mathbf{y}^k|_{k=1}^{K}$

$$\mathcal{L}(\theta) = \arg\max_{\theta} \sum_{k=1}^{K} \log P(\mathbf{y}^k|\mathbf{x}^k;\theta)$$

Inference in autoregressive models is usually performed with greedy or beam search (Freitag & Al-Onaizan, 2017). These models produce one token at a time and terminate when a special end of sentence token <*sep*> is encountered. The generated tokens are used as a conditioning context for the next token. This sequential generation from left to right make these models autoregressive therefore resulting in a higher latency and inadequate use of underlying parallel computational architecture.

5.2.2 Non-Autoregressive Neural Machine Translation

Non-autoregressive models attempt to model the joint distribution $P_\theta(\mathbf{y}|\mathbf{x})$ directly by breaking the probabilistic factorization. Each prediction is modeled as a product of the probability, which is independent of the decoding history during generation:

$$P(\mathbf{y}|\mathbf{x};\theta) = P(M|\mathbf{x}) \prod_{i=1}^{M} P(y_i|\mathbf{x}, M;\theta)$$

where $P(M|\mathbf{x})$ indicates an auxiliary length predictor, which is used to determine the translation length before translating the sentence. As opposed to NAT, AT models translation length implicitly and is determined by the special end of sentence token. Unfortunately, the performance of this simple model falls far behind autoregressive models, as sequences usually do have strong conditional dependencies between output variables. This problem can be mitigated by introducing a latent variable $\mathbf{z}$ to model these conditional dependencies. For the sake of simplicity we will be removing the length classifier:

$$P(\mathbf{y}|\mathbf{x};\theta) = \int_z P(\mathbf{y}|\mathbf{z},\mathbf{x};\theta)P(\mathbf{z}|\mathbf{x};\theta)\mathbf{dz}$$

where $P(\mathbf{z}|\mathbf{x};\theta)$ is the prior distribution over the latent variable $\mathbf{z}$ and $P(\mathbf{y}|\mathbf{z},\mathbf{x};\theta)$ is the decoder. Generation from this model is then done as:

$$P(\mathbf{y}|\mathbf{z},\mathbf{x};\theta) = \prod_{i=1}^{M} P(y_i|\mathbf{z},\mathbf{x};\theta)$$

The purpose of the latent variable $\mathbf{z}$ is to inject missing information about the dependencies in the output variable. A naive approach is to use the predicted sentence length as the latent variable as this can reduce the search space for model to sentences only with that particular length. But this results in a highly degraded output quality. Gu, Bradbury, et al. (2017) proposed to use word fertilities, where the input source word is copied multiple times based upon its fertility, to be used as the latent variable. This improved the translation quality. Later works introduced the use of generative flows which models complex distribution from simple distribution (X. Ma et al., 2019), reordered source sentence (Ran et al., 2019) or partially translated source sentence based upon phrase tables (Guo et al., 2019). Other approaches iteratively refine the sentence by starting with a blank sentence (Ghazvininejad et al., 2019a) and thus fall under the umbrella of semi-autoregressive models.

NAT models usually employ an identical encoder as the conventional Transformer architecture, but the decoder is different from the original one as it avoids the utilization of causal masks in the self-attention mechanism and the whole sequence is generated in parallel. The loss of dependency in the target sequence causes multimodality problem (competing translation) which severely degrades the performance resulting in repeated and incomplete translations.

5.3 Approach

As we move from autoregressive to non-autoregressive models, vital information about the dependencies between the output variables is lost. This information can be injected back in the model by using a latent variable $\mathbf{z}$. The latent variable $\mathbf{z}$ should account for as much as possible for the correlations between different outputs across time and it should be simple to infer its value for a given source sentence. The distribution of $\mathbf{z}$ given an input $\mathbf{x}$ forms a prior distribution $P(\mathbf{z}|\mathbf{x})$. Our model uses similar examples, to the test sentence, to form the prior. It is informative, effective and easier to infer. This section presents the prior that we have build using the data and how it has been incorporated in the Transformer model. We call the resulting Transformer as Exemplar Transformer as it utilise similar examples to build the prior. This section address the following questions:

1. How to define the similarity measure?
2. How to efficiently fetch the most similar examples from a large corpus?
3. How to condition the output on the fetched K-nearest neighbours?

5.3.1 K-Nearest Neighbours as Prior

We consider a pre-processing phase where addressing point (1) and (2) can provide an augmented version of the dataset. To build the augmented dataset, each sentence is first encoded into a vector. To get the sentence embedding, we apply a pre-trained BERT on the sentence and then build the sentence embeddings. We consider multiple ways to get the embeddings such as (i) getting the output embedding corresponding to the <*CLS*> token, (ii) summing up the output vectors of the words in the sentence , (iii) max-pooling the output vectors of the sentence words. We present our analysis about the effect of sentence representation in section 5.4.3. As the similarity measure, we experiment with $\ell 1$, $\ell 2$, and cosine distance between vectors. Its analysis is also mentioned in section 5.4.3.

For efficient retrieval of k-nearest neighbours, we use FAISS (J. Johnson, Douze, & Jégou, 2017). It is a fast retrieval library which uses Locality Sensitive Hashing (LSH) and can scale nicely to big corpora. LSH is an algorithmic technique that hashes similar input items into the same “buckets” with high probability (Slaney & Casey, 2008). Since similar items end up in the same buckets, this technique can be used for data clustering and nearest neighbor search. FAISS contains several methods for similarity search. It assumes that the instances are represented as vectors and are identified by an integer, and that the vectors can be compared with $\ell 2$ (Euclidean) distances or dot-products. Vectors that are similar to a query vector are those that have the lowest $\ell 2$ distance or the highest dot product with the query vector. It also supports cosine similarity, since this is a dot-product on normalized vectors.

5.3.2 Exemplar Transformer

We incorporate the prior in the NAT model introduced by Gu, Bradbury, et al. (2017). The model is made up of four components: an encoder stack, a decoder stack, length predictor and translation predictor (Figure 5.2). We provide detail about each component below:

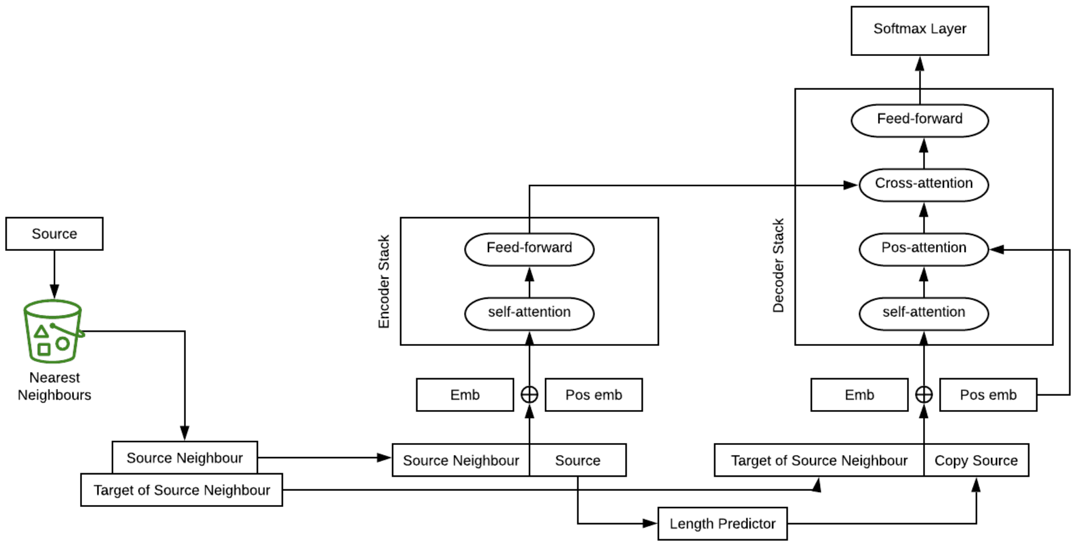


Figure 5.2: The four components of non-autoregressive machine translation model is shown. The encoder is similar to the autoregressive Transformer. The difference lies in the decoder which consist of an additional positional attention module. The nearest neighbours are concatenated withe the source and copied source and passed as input to the encoder and decoder. The attention modules are also modified by using a mask to split the attention heads.

Encoder Stack: A NAT encoder is similar to the autoregressive encoder with feed-forward networks and multihead attention modules. The input to the Transformer is the source sentence. The sentence is first converted into its corresponding word embeddings. As there is no recurrence in Transformers, we must add some information about the positions into the input embeddings. This is done using positional encoding.

Decoder Stack: A NAT decoder consists of various modifications to translate the sentence non-autoregreesively. First, as we cannot use time-shifted target outputs (during training) or previously predicted outputs (during inference) as the inputs to the first decoder layer, we initialize the decoding process using copied source inputs from the encoder side. As the source and target sentence are of different lengths, each encoder input is copied at position i. In order to determine which input token to copy at position i, the index of the word is determined as $\mathrm{round}((N * i)/M)$ times where N is the source length and M is the target sentence length.

As the model is now non-autoregressive there is no need to prevent the model from accessing information from later decoding steps. This is done by removing the causal mask used in the self-attention module of the conventional Transformer's decoder. Gu, Bradbury, et al. (2017) found that introducing a positional attention module with position embeddings as key and query can incorporate positional information directly into the attention process and provides a stronger positional signal than the embedding layer alone. This attention mechanism is incorporated in each decoder layer along with cross and self attention modules.

Length predictor: NAT models need to predict the target length before generating the target sentence. This is done by training a classifier to predict target length given the source sentence. Some studies have experimented with predicting the source and target length difference and have found to be equally effective (Shu et al., 2020). We will use length prediction rather than length difference classifier in our model.

Translation predictor: Translation predictor consist of softmax layer over the target vocabulary. It takes input from the decoder layer and output the whole sentence in parallel.

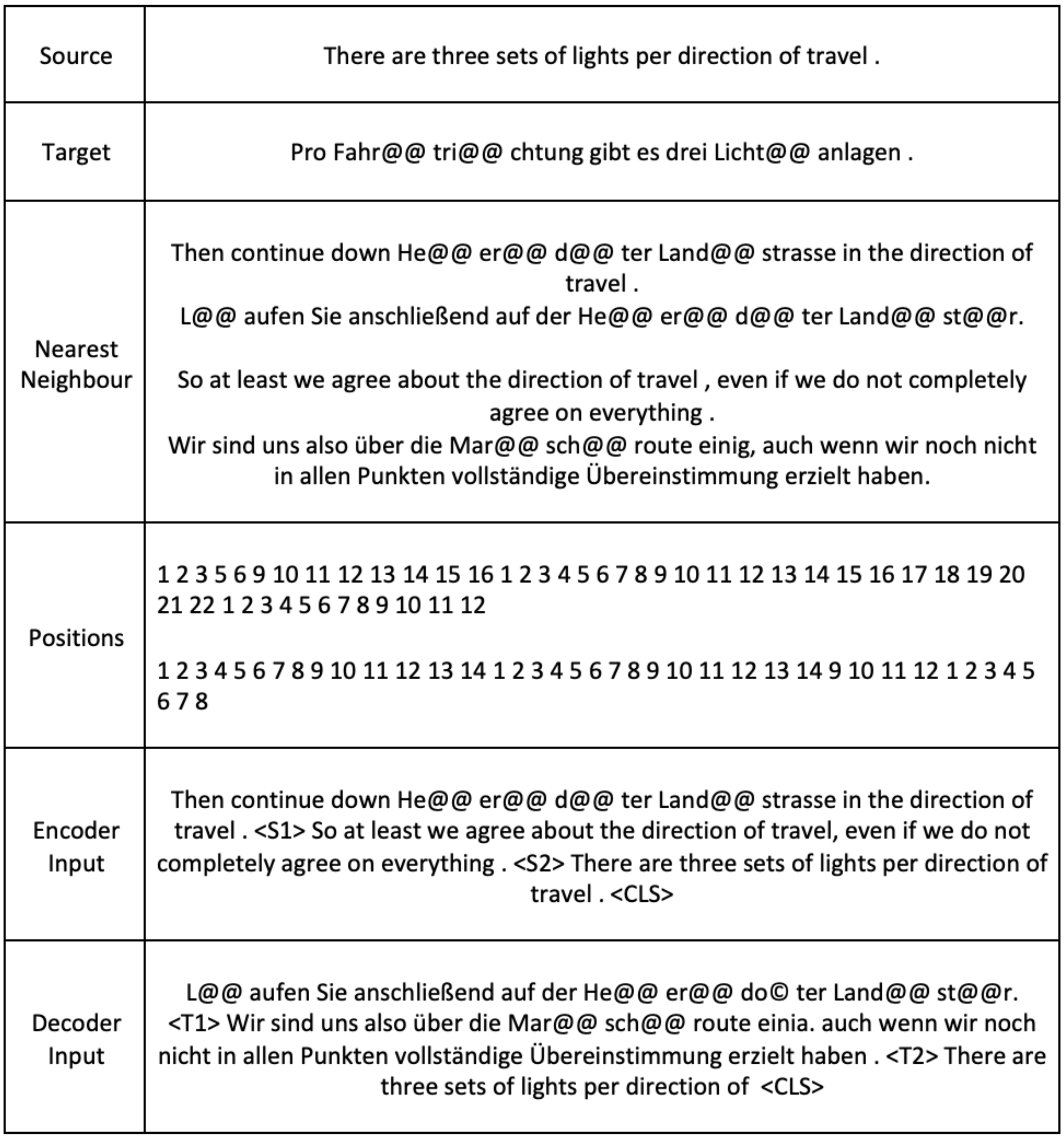
Incorporating Prior: We incorporate the prior by concatenating the K-nearest neighbour with the source and target sentence. Multiple nearest neighbours can be attached to the given sentence. In order to differentiate between the neighbours, a special symbol *<Sl>* and *<Tl>* where l is the number of neighbour and S and T are source and target neighbour respectively is added at the end of each sentence. Position embeddings are added relative to each sentence that is position embeddings resets to 1 when a special symbol is encountered. *<CLS>* token is added at the end of each source sentence Figure 5.3. Each nearest neighbour helps in reducing the search space by providing translation of similar words or phrases. It also provide information about semantically correct arrangement of words. In Figure 5.3, the model exploits information regarding "direction of travel" to properly translate it.

Let $\mathbf{x}$ be an input source sentence, $\mathbf{y}$ as its target and $\mathbf{x}^{'}$ as copied source. As the source $\mathbf{x}$ and target $\mathbf{y}$ can be of different length, the initial copied source $\mathbf{x}^{'}$ is built by selecting tokens from the source side such that at each position i in copied source sentence:

$$\mathbf{x}^{'}[i] = \mathbf{x}[\mathrm{round}(\frac{N * i}{M})]$$

where N is the source length and M is the target sentence length. We then fetch the nearest neighbours of the input pair $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$. The input to the decoder is the concatenation of input and source neighbour $(\tilde{\mathbf{x}} + \mathbf{x})$ where as the input for decoder is the concatenation of copied source and the target sequence of source neighbour $(\tilde{\mathbf{y}} + \mathbf{x}^{'})$. The given sequence pass through the decoder layers and the $\mathrm{softmax}$ output layer is used to predict the output. While training, we ignore the $\mathrm{softmax}$ outputs for token corresponding to the neighbours. We only use the loss from tokens corresponding to the target sentence and the length classifier for training. At decoding time, the source sentence is copied according to the output of the length classifier. It is then concatenated with the nearest neighbour. Outputs from the nearest neighbours are ignored. Following previous work on NAT models, we predict multiple sentence lengths and multiple target sentence. We pick the one best ranked by the autoregressive model.

Attention Heads: Incorporating the nearest neighbours may confuse the model as to which part of the sentence the model should focus. We modify the attention mechanism such that the heads are distributed between neighbours and the sentence being translated. We dedicate more heads to



Source	There are three sets of lights per direction of travel .
Target	Pro Fahr@@ tri@@ chtung gibt es drei Licht@@ anlagen .
Nearest Neighbour	Then continue down He@@ er@@ d@@ ter Land@@ strasse in the direction of travel . L@@ aufen Sie anschließend auf der He@@ er@@ d@@ ter Land@@ st@@r. So at least we agree about the direction of travel , even if we do not completely agree on everything . Wir sind uns also über die Mar@@ sch@@ route einig, auch wenn wir noch nicht in allen Punkten vollständige Übereinstimmung erzielt haben.
Positions	1 2 3 5 6 9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12 13 14 1 2 3 4 5 6 7 8 9 10 11 12 13 14 9 10 11 12 1 2 3 4 5 6 7 8
Encoder Input	Then continue down He@@ er@@ d@@ ter Land@@ strasse in the direction of travel . <S1> So at least we agree about the direction of travel, even if we do not completely agree on everything . <S2> There are three sets of lights per direction of travel . <CLS>
Decoder Input	L@@ aufen Sie anschließend auf der He@@ er@@ do© ter Land@@ st@@r. <T1> Wir sind uns also über die Mar@@ sch@@ route einia. auch wenn wir noch nicht in allen Punkten vollständige Übereinstimmung erzielt haben . <T2> There are three sets of lights per direction of <CLS>

Figure 5.3: The figure shows the the two nearest neighbours concatenated with the source and target sentence. Note that the positions are relative to the sentence. The positions reset when a new sentence starts. The nearest neighbours can provide syntactic and semantic information to translate the sentence properly.

the sentence being translated than to the neighbours. In our experiments, we dedicate quarter of the heads to the neighbours. This allows certain heads to get specialised in getting the required information about syntax and semantics. It also allows to focus the model on the sentence being translated and not get distracted by the concatenated neighbours.

The multi-head attention function can be represented as follows:

$$\mathrm{multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{concat}(head_1, head_2, ...head_n)\mathbf{W}^o$$

where **Q**, **K**, **V** are query, key and value vectors and $\mathbf{W}^0$ is a learnable matrix for linearly projecting attention heads outputs. Each attention head is multiplied with a masking matrix **R**, that allows it to only focus on a certain part of the sentence.

$$\mathrm{head}_i = \mathrm{attention}(\mathbf{QW}_i^q, \mathbf{KW}_i^k, \mathbf{VW}_i^v)\mathbf{R_i}$$

where $\mathbf{W_i}$ are learnable parameters and mask **R** is created such that if the head is concentrating on neighbours than only the neighbours output in allowed to move forward all else is discarded by multiplying by zero.

5.4 Experiments

5.4.1 Setup

Datasets We conduct experiments on two widely used machine translation datasets in non-autoregressive models: WMT14 English-German (En-De) and WMT16 English-Romanian (En-Ro). These dataset is made up of 4.0M and 610K sentence pairs respectively. We preprocessed the data following Gu, Bradbury, et al. (2017). The sentences are tokenised using fairseq library (Ott et al., 2019). The data is preprocessed using byte-pair encoding with 32K merge operations for both language pairs. We learn the shared vocabulary with the joint training corpus in both source and target sides. We use newstest2013 and newstest2014 as the validation and test set respectively for the WMT14 En-De and newsdev2016 and newstest2016 are used as the validation and test set for WMT16 En-Ro task.

Knowledge Distillation Following previous works on NAT models, we used sequence level knowledge distillation (KD). KD is applied during NAT model training by replacing the target side of training samples with the outputs from a pre-trained autoregressive model trained on the same corpus with a roughly equal number of parameters. The resulting dataset is used for training the NAT model. KD benefits the NAT models by reducing the complexity of data, thereby overcoming the multimodality problem (alternative translations for an input) (C. Zhou et al., 2019).

Evaluation Metrics We used BLEU scores (Papineni et al., 2002) to measure the translation quality. Inference latency was measured using a single Nvidia V100 GPU. We measured the time of decoding one sentence at a time to mimic the real world industry scenario.

Model Configuration We follow the configurations from previous works. Specifically for both models we use dropout of 0.3, num_heads of 8, layers as 6, model_dimension as 512 and hidden_dimension as 2048. We use weight decay of 0.01 as well as label smoothing of 0.1. For attention heads division, we dedicated 2 heads to the similar examples and rest of the heads to sentence being translated. We use $\ell 2$ distance as the similarity matrix and use the output embedding corresponding to the $<$*CLS*$>$ token as the sentence representation. Our models are implemented in open-source toolkit fairseq (Ott et al., 2019). The models are trained on Nvidia V100 GPU for 200K updates with a batch size of 64K tokens. We used ADAM optimiser in our training.

Baselines As our approach is generic and can be added to any NAT model built on top of model introduced by Gu, Bradbury, et al. (2017), we only compare with seminal NAT model in the main results. We also compare our approach with NAT models that specifically modify the prior. This includes the flowseq model by X. Ma et al. (2019), which built prior using normalization flows, Ran et al. (2019) introduced reorderNAT which reorder the input before feeding it into the decoder, Guo et al. (2019) introduced encNAT which convert the input sentence in the target sentence using phrase tables and use it as the initial input for the decoder.

5.4.2 Results

We consider three variants of our model. In the first variant we concatenated nearest neighbour only at the source (NAT + source neighbour). The second variant is created by concatenating the target of nearest neighbour only at the target (NAT + target neighbour). The third variant has both the source neighbour and its target (source and target neighbour). We also consider results from dividing the attention heads as compared to attention over the complete input.

Distilled Data We conduct experiments on the distilled data that is the data generated using sequence level distillation. The results are shown in Table 5.1. DH denotes head division in the table. The results without the division of attention heads show low but comparable performance with NAT model. We see a huge drop in BLEU score by using just the source neighbour. This occurs probably because the model was not able to draw associations between the neighbouring source sentence with the target. Furthermore, the long source sequence may confuse the model. Using just the target neighbour improves upon the previous variant. We believe that the target side attention mechanism may have helped in translating the sentence. Target side neighbours may provide an initial scaffold for building the target sequence by providing initial keywords. Our approach works well when both source and target neighbours are used as indicated by higher BLEU score. Together with source and target, the model may learn associations such as translation

	WMT-14		**WMT-16**		
	en-de	**de-en**	**en-ro**	**ro-en**	**speedups**
Transformer	27.48	31.21	33.70	34.05	1.0x
NAT	17.69	21.47	**27.29**	29.06	15.6x
NAT + source neighbour	14.05	15.31	22.55	25.67	13.6x
NAT + target neighbour	15.57	17.23	24.59	26.01	12.7x
NAT + source and target neighbour	17.24	20.42	25.10	28.46	12.1x
NAT + source neighbour + DH	14.15	15.65	23.01	25.95	13.6x
NAT + target neighbour + DH	15.97	17.76	24.84	26.31	12.7x
Van NAT + source and target neighbour + DH	**17.75**	**21.53**	26.30	**29.15**	12.1x

Table 5.1: The table shows the result on wmt-14 and wmt-16 dataset. Only one neighbour is concatenated with the source and target sentence.

	WMT-14		WMT-16		
	en-de	de-en	en-ro	ro-en	speedups
NAT + source neighbour	7.79	7.81	8.35	8.11	13.6x
NAT + target neighbour	7.91	8.15	8.89	9.01	12.7x
NAT + source and target neighbour	9.45	8.21	10.01	9.33	12.1x
NAT + source and target neighbour + DH	9.68	8.73	10.34	9.55	12.1x

Table 5.2: The table shows the result on running experiments on original dataset. The actual outputs are used to train the model instead of using outputs from autoregressive teacher as targets.

of phrases or words that may appear in the nearest neighbours. We observe a drop in speed up as the model has to process sequences of longer length.

We see an increase in BLEU score across the three variants after introducing head division. The specialised heads were able to convey better information regarding regarding sentence structure and meaning. Our approach performed better than the baseline across all datasets except WMT-16 ro-en where it performed comparably by coming close in terms of BLEU value.

Original Data The results of conducting experiments on original dataset are shown in Table 5.2. We use the actual targets instead of the outputs from an autoregressive teacher model. The approach completely fail on the original dataset, showing the difficulty NAT models have on modeling multimodal distribution. Even with original dataset we see better results by using source and target neighbours indicating that model is learning a better distribution by leveraging the neighbours. The best results among our model variants are obtained by dividing the heads. This indicate that specialised heads help by reducing model search space and make it more robust.

Comparison with other models The results are shown in Table 5.3. Comparing our approach with others shows that our model lags behind similar approaches. The best results are either produces by flowseq or encNAT. Unlike our approach both of these works involve difficult training procedures or extra modules. Our approach although lagging, is simple and can be easily incorporated in other models without changing any training or decoding procedure. The speedups

	WMT-14		WMT-16		
	en-de	de-en	en-ro	ro-en	speedups
Transformer	27.48	31.21	33.70	34.05	1x
NAT	17.69	21.47	27.29	29.06	15.6x
Flowseq	23.72	28.39	29.73	30.72	10.0x
Enc NAT	24.28	26.10	29.85	27.30	12.4x
Reorder NAT	22.79	27.28	29.30	29.50	16.1x
NAT + source and target neighbour + DH	17.75	21.53	26.30	29.15	12.1x

Table 5.3: The table shows the result of our approach compared to other similar models.

mentioned in the table should be taken with a grain of salt. The numbers have been taken from the corresponding papers. Each paper implements model using different libraries and framework. Nonetheless, following the approach by other papers where the comparison is done by using these numbers, we present our speedup accordingly.

5.4.3 Analysis

We provide analysis of the model by increasing the number of neighbours and changing the similarity matrix. We also look into ways of better representing the sentences. Finally, we provide some examples outputs from our model.

Effects of increasing the number of examples We analyse the effect on output quality by increasing the number of neighbours. We use WMT-14 de-en dataset and trained model for 2 to 4 neighbours. The results are shown in Table 5.4 for both divided attention and original attention mechanism. We see drop in quality and speedup as we increase the number of similar examples. We hypothesize that the decrease in quality occurs because it becomes difficult for the model to concentrate on the right information. The drop in speedup is because of processing a long sequence. As we divide the attention heads, we see improvement in quality as indicated by higher BLEU scores. Increasing the neighbours again showed degraded quality. More examples tend to distract the model as they may not be as similar to the sentence being translated. This is evidence by lower BLEU scores.

Effects of changing the similarity measure We changed the similarity measure to build the augmented data with nearest neighbours. Along with ℓ_2 distance we also used ℓ_1 and cosine similarity. We run experiments on WMT-14 de-en dataset. The results are shown in Table 5.5. We used a model with attention head division. We observe that ℓ_2 and cosine similarity gives almost similar results while the ℓ_1 distance showed lower BLEU score compared to other metrics. We hypothesise that ℓ_2 can better differentiate between instances as it allows small changes to be amplified with the squared term present in their formulae. The cosine similarity can also take advantage of the squared term present in the denominator to amplify the small changes which helps in capturing better neighbours.

	BLEU	**speedup**
NAT + 1 neighbour	20.42	12.1x
NAT + 2 neighbour	19.33	11.6x
NAT + 3 neighbour	17.76	11.1x
NAT + 4 neighbour	15.49	10.4x
NAT + 1 neighbour + DH	21.53	12.1x
NAT + 2 neighbour + DH	19.67	11.6x
NAT + 3 neighbour + DH	18.12	11.1x
NAT + 4 neighbour + DH	15.89	10.4x

Table 5.4: The table shows the BLEU score and speedup achieved by increasing the number of neighbours. The neighbours are added at both the source and target. DH denote the attention mechanism whereby heads are distributed between neighbours and sentence being translated.

	CLS	**SUM**	**MAX-POOL**
ℓ_1	17.55	17.23	16.67
ℓ_2	21.53	21.12	20.87
Cosine	21.57	21.17	20.95

Table 5.5: The table shows the BLEU score of models trained on data built using different similarity measures. It also shows results of using different sentence representations. CLS refers to using cls token as sentence representations. SUM refers to summing up the output vectors of the words in the sentence and MAX-POOL refers to max-pooling the output vectors of the sentence words.

Effects of changing the sentence representation We changed the sentence representation to build the augmented data with nearest neighbours. We consider using <*CLS*> token which is concatenated to the end of the sentence, summing up the output vectors of the words in the sentence and max-pooling the output vectors of the sentence words. We used outputs from pre-trained BERT to get build the sentence embeddings. We run experiments on WMT-14 de-en dataset and trained a model with attention head division. The results are shown in Table 5.5. We observe dataset built using CLS token to perform better than others. We hypothesise that summing and max-pooling may remove some essential information from the representations thus leading to similar examples which are not helpful in translation.

Effects of changing the number of heads We increase the number of attention heads for similar examples. We train model using WMT-14 de-en dataset. The results are shown in Figure 5.4. We see a decrease in BLEU score as more heads are dedicated to the neighbours. We hypothesis that increasing the number of heads over similar examples distracts the model and make it concentrate on part of the sentence which are not useful for translation. This is evidenced by lower BLEU score as the number of heads for similar examples are increased. The BLEU score dropped from 21 points to less then 11 points as the number of heads are increased from 2 to 7.

Case Study We present several translation examples sampled from the WMT14 De-En dataset in Figures 5.5, 5.6 and 5.7, including the source sentence, the target reference (i.e., the ground-truth translation), the translation given by the autoregressive teacher model, by the NAT model, the nearest neighbours and output of the Exemplar Transformer. As can be seen, NAT suffers severely

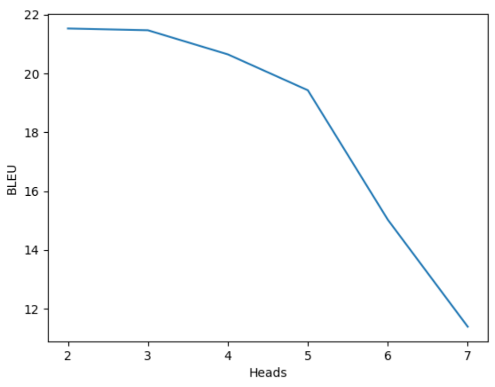


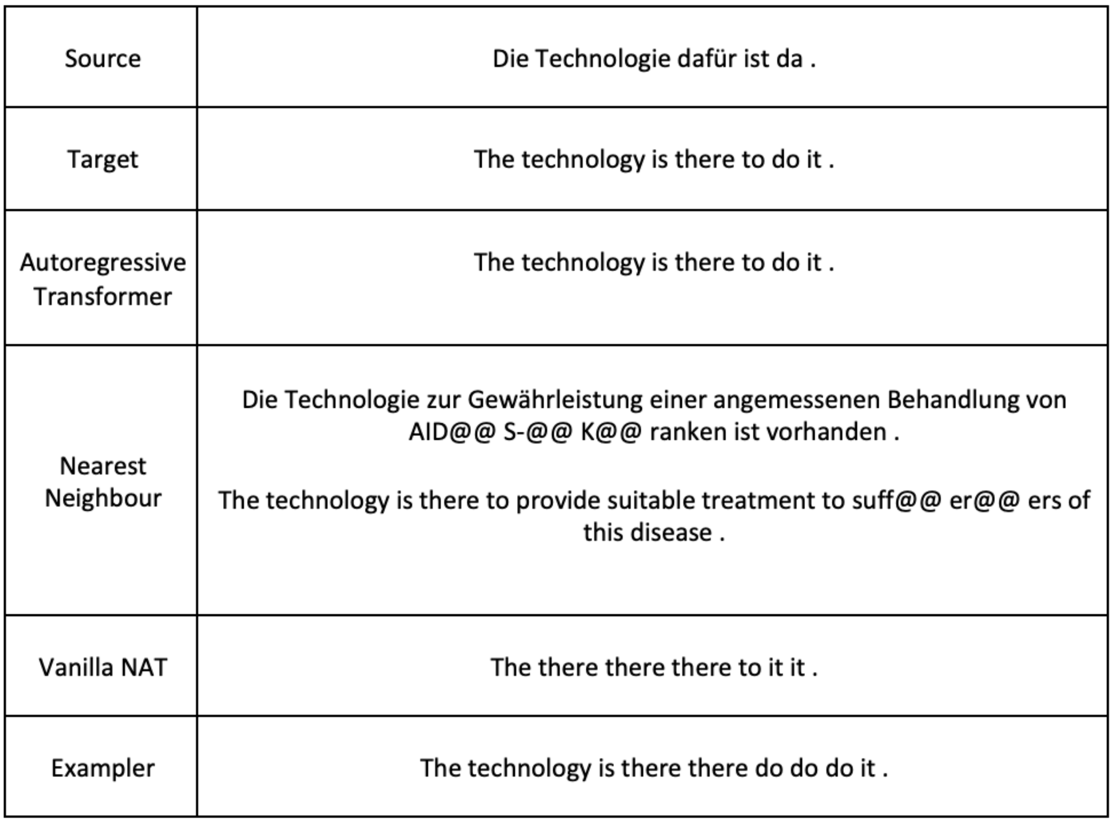
Figure 5.4: The figure shows the effect of increasing the number of heads for similar examples. We observe a decrease in BLEU score as more heads are dedicated to the neighbours.

from the issue of repeated translation (e.g., the 'there' and 'it' in the first example). The similar example helps in translating the source sentence by providing information such as the translation of the phrase. For example 'The technology is there' in Figure 5.5, 'the firm was' in Figure 5.6 and 'A decision is expected' appeared in the nearest neighbour and was translated correctly in the output sequence.

5.5 Summary

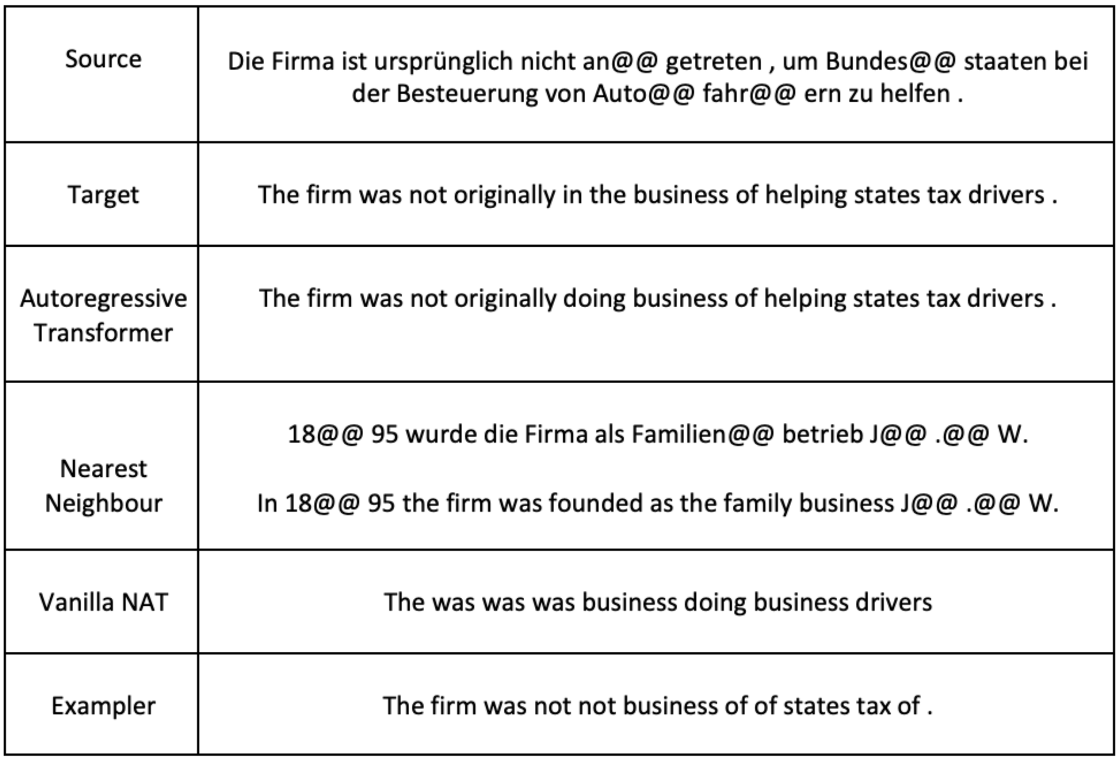
NAT models struggle to capture the underlying highly multimodal target distribution due to the loss of autoregressive connections in decoder. As the sentence is generated in parallel, the decoding time is improved but the output quality degrades. In order to overcome the loss of information, we suggest an informative and effective prior based upon the nearest neighbours. We also present ways of incorporating the prior by dividing the attention heads. Our results suggest that the approach is promising as utilising the nearest source and target sentence provide improvements over the vanilla NAT model. The improvements are minor but indicate that pursuing this approach further may provide some benefits. Nearest source and target sentence may provide associations that may be helpful in translating the given source sentence. Utilising just the source does not improve the results. We conjecture that using just the nearest source sentence confuse the model. Increasing the number of neighbours decreases the BLEU score as it becomes difficult for the model to concentrate on the right information. Dividing the attention heads to focus on a specific neighbour is helpful in this scenario. The dedicated attention heads help in extracting the required information while not distracting the model.

Future work can look into finding novel ways for incorporating the prior such as allowing the model to determine how many heads should be dedicated for a given example. Some empirical studies are also required for studying the effect of dividing the heads among similar examples such that more heads are given to examples nearby than the one which are far in the search space. Another future direction is to learn the similarity metric such that instead of fixing the metric, the model should change the metric to retrieve the neighbours according to the situation.



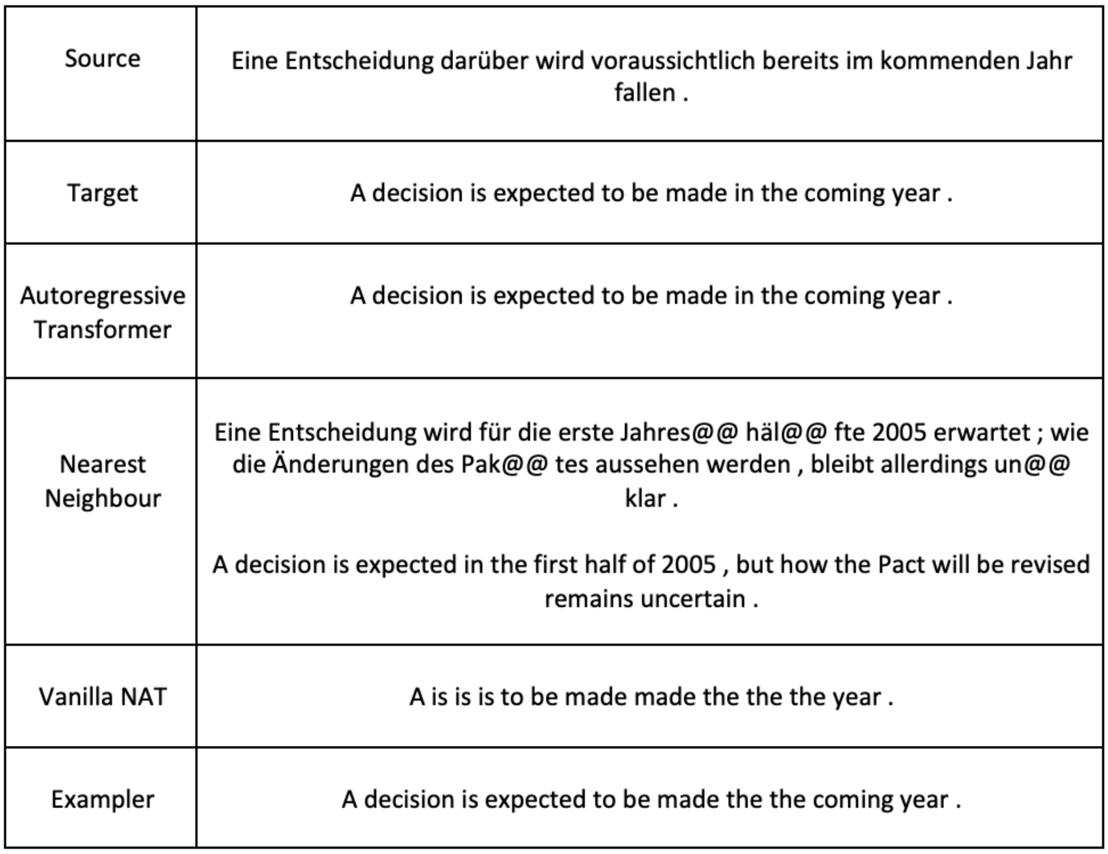
Source	Die Technologie dafür ist da .
Target	The technology is there to do it .
Autoregressive Transformer	The technology is there to do it .
Nearest Neighbour	Die Technologie zur Gewährleistung einer angemessenen Behandlung von AID@@ S-@@ K@@ ranken ist vorhanden . The technology is there to provide suitable treatment to suff@@ er@@ ers of this disease .
Vanilla NAT	The there there there to it it .
Exampler	The technology is there there do do do it .

Figure 5.5: Example output 1: Figure shows the output of Exemplar Transformer compared to NAT model



Source	Die Firma ist ursprünglich nicht an@@ getreten , um Bundes@@ staaten bei der Besteuerung von Auto@@ fahr@@ ern zu helfen .
Target	The firm was not originally in the business of helping states tax drivers .
Autoregressive Transformer	The firm was not originally doing business of helping states tax drivers .
Nearest Neighbour	18@@ 95 wurde die Firma als Familien@@ betrieb J@@ .@@ W. In 18@@ 95 the firm was founded as the family business J@@ .@@ W.
Vanilla NAT	The was was was business doing business drivers
Exampler	The firm was not not business of of states tax of .

Figure 5.6: Example output 2: Figure shows the output of Exemplar Transformer compared to NAT model



Source	Eine Entscheidung darüber wird voraussichtlich bereits im kommenden Jahr fallen .
Target	A decision is expected to be made in the coming year .
Autoregressive Transformer	A decision is expected to be made in the coming year .
Nearest Neighbour	Eine Entscheidung wird für die erste Jahres@@ häl@@ fte 2005 erwartet ; wie die Änderungen des Pak@@ tes aussehen werden , bleibt allerdings un@@ klar . A decision is expected in the first half of 2005 , but how the Pact will be revised remains uncertain .
Vanilla NAT	A is is is to be made made the the the year .
Exampler	A decision is expected to be made the the coming year .

Figure 5.7: Example output 3: Figure shows the output of Exemplar Transformer compared to NAT model

Chapter 6

Conclusion and Future Directions

6.1 Summary

There is a balancing act between quality and time as we move from autoregressive to non autoregressive models. Autoregressive models are slower but produce better quality output compared to non-autoregressive models which are faster. Semi-autoregressive models provide the best of both worlds by decoding faster but at the expense of output quality. This thesis therefore, explored the problem of text generation with the aim of improving generation across the three model families. It identifies and addresses various literature gaps such as ineffective decoding methods for autoregressive models, lack of length flexibility in autoregressive and non-autoregressive models, and inferior output quality by non-autoregressive models compared to autoregressive models.

In Chapter 3, we address the first limitation that is ineffective decoding methods for autoregressive models by proposing an iterative approach capable of revisiting and revising the generated text. This is in contrast to the current prevalent decoding methods such as greedy and beam search. These methods are simple, error prone and unidirectional resulting in error being propagated throughout the decoding process. We propose to use the method of auxiliary coordinate, which introduce discrete variables for output tokens, and auxiliary continuous variables representing the states of the underlying autoregressive model. Our approach alternate between creating a factor graph approximation, whose maximum a posteriori (MAP) solution is found using dynamic programming and updating the auxiliary variable based upon the new factor graph. The iterative approach allows revisiting the sentence and rectifying mistakes. We applied our approach on text infilling task and showed superior output compared to competing decoding methods. We conduct various ablation studies such as varying the masking rate and Markov order. Increasing the masking rate results in a drop in BLEU score. Higher masking rates are challenging for our approach due to reduced context for generating the correct tokens. Increasing the Markov order produces more accurate approximations to the original decoding problem, but they result in harder optimization problems. Finally, we also presented two methods for updating the state variables. Gradient based method results in lower perplexity compared to forced decoding. This is because the state variables are updated based upon the gradient of the objective function.

In Chapter 4, we address the limitation of lack of length flexibility and the difficulty of maintaining coherence when generating long sequences by autoregressive and non-autoregressive models. By framing document generation as a hierarchical Markov decision process, we propose a semi-autoregressive model that is capable of revising and updating the generated text with the help of editing operators. The high level program is responsible for intra-sentence phenomena such as maintaining coherency among sentences. The low level program is responsible for inter-sentence phenomena such as producing syntactically and semantically correct sentences. With the help of editing operation such as insert , delete, update and reposition, our model allowed length flexibility. We applied the proposed approach on document generation tasks such as story generation and summarization. Our models underperform state-of-the-art but show promise. The results on synthetic dataset shows model ability to capture discourse level phenomena by using sentence representations. The baseline models use individual tokens/words and find it difficult to capture long range document level phenomena correctly. The experiments on ROC stories show the model's ability to dynamically change the length using the insert and update operator. We also conduct experiments on real world summarization dataset. Using our model as a document level post-editing system, we show that the high level program has the capability to capture the underlying document structure and can provide context and structure aware information which can then improve the coherence of the document. We identified various areas for improving the model such as using distilled dataset to reduce the modality of data (alternative correct outputs), better training approach including pre-training and fine-tuning, using autoregressive model as a low level program and modifying attention such that more attention is given to the sentences around the given sentence than those far away in the document.

In Chapter 5, we address the problem of inferior output quality by non-autoregressive models compared to autoregressive models. We propose an informative prior based upon the nearest neighbours to inject information about dependencies in the output variables. We also propose changes to the attention mechanism to incorporate the prior effectively. In particular we propose attention heads division where lesser heads are dedicated to the similar examples than to the sentence being translated. This allows the model to be not distracted by extra information. We applied our proposed approach on benchmark datasets. Our approach was effective compared to the vanilla NAT model but underperfom other comparable state-of-the-art models. To further study the behaviour of the model, we conduct various ablation studies. Increasing the number of nearest neighbours results in a drop in quality and speedup. We hypothesise that the decrease in quality occurs because it becomes difficult for the model to concentrate on the right information. The drop in speedup is because of processing a longer sequence. By changing the similarity measure, we observe that ℓ_2 and cosine similarity gives almost similar results while the ℓ_1 distance shows lower BLEU score compared to other metrics. We hypothesise that ℓ_2 and cosine similarity can better differentiate between instances as it allows small changes to be amplified with the squared term present in their formulae. Finally, we increase the number of attention heads for similar examples. We see a decrease in BLEU score as more heads are dedicated to the neighbours. We hypothesise that increasing the number of heads over similar examples distracts the model and makes it concentrate on parts of the sentence which are not useful for translation.

6.2 Future Directions

There are various potential extensions to the findings of this thesis. We briefly discuss some of them as follows:

1. Chapter 3 presented a method for improving decoding in discrete autoregressive models using dynamic programming. The core idea is to introduce auxiliary variables to decouple the non-Markovian aspects of the model, permitting an approximate solution. Our approach does have limitations, most notably the computational complexity which is polynomial in the vocabulary size, thus limiting its application to open text generation problems. Improving the complexity of decoding is an important direction for future research, as is applying the method to other autoregressive models, such as the Transformer, which includes self attention, as well as other structured prediction problems.

2. Chapter 4 presented a semi-autoregressive model for long text generation with dynamic length flexibility. Our approach underperformed the baselines. There are many ways we can improve our model. One way is to train the two levels separately. A low level program should be initially trained and tested separately. The high level program is then trained while keeping the low level program frozen. Distilled dataset has been shown to improve NAT models by dealing with multimodality problem. We can also explore leveraging distilled data using curriculum learning that is starting with distilled dataset and then moving to harder actual examples. Pre-training and fine-tuning approaches have been found useful in various tasks. The classifiers in our model can be individually pre-trained and then fine-tuned for better model generalisation. We can also take advantage of inherent left to right generation bias by using an autoregressive model as a low-level program but this can lead to longer decoding times. Finally, wider context has been shown to improve results for this document-level task therefore exploring and designing better attention mechanism can be a potential extension of this work.

3. Chapter 5 presented an informative prior to inject information about dependencies in NAT models. This work can be extended by finding better ways for incorporating the prior. One way is to allow the model to determine the optimal division of heads for neighbours and actual sentence. Another extension is to divide the heads such that more heads are given to examples nearby than the one which are far in the search space. Future work can also look into learning the similarity metric such that instead of fixing the metric, the model should dynamically determine the metric to retrieve the neighbours.

References

Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., & Zhang, L. (2018). Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 6077–6086).

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Baheti, A., Ritter, A., Li, J., & Dolan, B. (2018). Generating more interesting responses in neural conversation models with distributional constraints. *arXiv preprint arXiv:1809.01215*.

Bao, G., Zhang, Y., Teng, Z., Chen, B., & Luo, W. (2021). G-transformer for document-level machine translation. *arXiv preprint arXiv:2105.14761*.

Bao, Y., Zhou, H., Feng, J., Wang, M., Huang, S., Chen, J., & Li, L. (2019). Non-autoregressive transformer by position learning. *arXiv preprint arXiv:1911.10677*.

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

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning* (pp. 41–48).

Bosselut, A., Celikyilmaz, A., He, X., Gao, J., Huang, P.-S., & Choi, Y. (2018). Discourse-aware neural rewards for coherent text generation. *arXiv preprint arXiv:1805.03766*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... others (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Carreira-Perpinan, M., & Wang, W. (2014). Distributed optimization of deeply nested systems. In *Artificial intelligence and statistics (aistats)* (pp. 10–19).

Charniak, E. (1996). *Statistical language learning*. MIT press.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cohen, E., & Beck, C. (2019). Empirical analysis of beam search performance degradation in neural sequence models. In *International conference on machine learning* (pp. 1290–1299).

Dantzig, G., & Fulkerson, D. R. (2003). On the max flow min cut theorem of networks. *Linear inequalities and related systems*, *38*, 225–231.

Deng, Y., Bakhtin, A., Ott, M., Szlam, A., & Ranzato, M. (2020). Residual energy-based models for text generation. *arXiv preprint arXiv:2004.11714*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Du, C., Tu, Z., & Jiang, J. (2021). Order-agnostic cross entropy for non-autoregressive machine translation. *arXiv preprint arXiv:2106.05093*.

Dumais, S. T. (2004). Latent semantic analysis. *Annual Review of Information Science and Technology (ARIST)*, *38*, 189–230.

Eisenstein, J. (2019). *Introduction to natural language processing*. MIT press.

El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2021). Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, *165*, 113679.

Elman, J. L. (1990a). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Elman, J. L. (1990b). Finding structure in time. *COGNITIVE SCIENCE*, *14*(2), 179–211.

Fabbri, A. R., Li, I., She, T., Li, S., & Radev, D. R. (2019). Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

Fan, A., Bhosale, S., Schwenk, H., Ma, Z., El-Kishky, A., Goyal, S., ... others (2021). Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, *22*(107), 1–48.

Fan, A., Lewis, M., & Dauphin, Y. (2018). Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Fang, L., Zeng, T., Liu, C., Bo, L., Dong, W., & Chen, C. (2021). Transformer-based conditional variational autoencoder for controllable story generation. *arXiv preprint arXiv:2101.00828*.

Fedus, W., Goodfellow, I., & Dai, A. M. (2018). Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*.

Feng, S. Y., Li, A. W., & Hoey, J. (2019). Keep calm and switch on! preserving sentiment and fluency in semantic text exchange. *arXiv preprint arXiv:1909.00088*.

Frederking, R. (1996). Grice's maxims: do the right thing. *Frederking, RE*.

Freitag, M., & Al-Onaizan, Y. (2017). Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*.

Garcia, F., & Rachelson, E. (2013). Markov decision processes. *Markov Decision Processes in Artificial Intelligence*, 1–38.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 1243–1252).

Ghazvininejad, M., Levy, O., Liu, Y., & Zettlemoyer, L. (2019a). Constant-time machine translation with conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Ghazvininejad, M., Levy, O., Liu, Y., & Zettlemoyer, L. (2019b). Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Ghazvininejad, M., Levy, O., & Zettlemoyer, L. (2020). Semi-autoregressive training improves mask-predict decoding. *arXiv preprint arXiv:2001.08785*.

Ghazvininejad, M., Shi, X., Priyadarshi, J., & Knight, K. (2017, July). Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, system demonstrations* (pp.

43–48). Vancouver, Canada: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/P17-4008`

Goyal, K., Dyer, C., & Berg-Kirkpatrick, T. (2019). An empirical investigation of global and local normalization for recurrent neural sequence models using a continuous relaxation to beam search. *arXiv preprint arXiv:1904.06834*.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on machine learning* (pp. 369–376).

Grover, A., Song, J., Agarwal, A., Tran, K., Kapoor, A., Horvitz, E., & Ermon, S. (2019). Bias correction of learned generative models using likelihood-free importance weighting. *arXiv preprint arXiv:1906.09531*.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., & Socher, R. (2017). Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Gu, J., Cho, K., & Li, V. O. (2017). Trainable greedy decoding for neural machine translation. *arXiv preprint arXiv:1702.02429*.

Gu, J., & Kong, X. (2020). Fully non-autoregressive neural machine translation: Tricks of the trade. *arXiv preprint arXiv:2012.15833*.

Gu, J., Wang, C., & Zhao, J. (2019). Levenshtein transformer. *arXiv preprint arXiv:1905.11006*.

Guo, J., Tan, X., He, D., Qin, T., Xu, L., & Liu, T.-Y. (2019). Non-autoregressive neural machine translation with enhanced decoder input. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 3723–3730).

Guo, J., Tan, X., Xu, L., Qin, T., Chen, E., & Liu, T.-Y. (2020). Fine-tuning by curriculum learning for non-autoregressive neural machine translation. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 7839–7846).

Gutmann, M., & Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 297–304).

Hinton, G. E. (1999). Products of experts. In *Ninth international conference on artificial neural networks* (Vol. 1, pp. 1–6).

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

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

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2019). The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Holtzman, A., Buys, J., Forbes, M., Bosselut, A., Golub, D., & Choi, Y. (2018). Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Horvat, M., & Byrne, W. (2014). A graph-based approach to string regeneration. In *Proceedings* *of the student research workshop at the 14th conference of the european chapter of the association for computational linguistics* (pp. 85–95).

Hovy, E. H., Lin, C.-Y., Zhou, L., & Fukumoto, J. (2006). Automated summarization evaluation with basic elements. In *Lrec* (Vol. 6, pp. 604–611).

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, *50*(2), 1–35.

Inaguma, H., Higuchi, Y., Duh, K., Kawahara, T., & Watanabe, S. (2021). Orthros: Non-autoregressive end-to-end speech translation with dual-decoder. In *Icassp 2021-2021 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 7503–7507).

Ippolito, D., Grangier, D., Callison-Burch, C., & Eck, D. (2019). Unsupervised hierarchical story infilling. In *Proceedings of the first workshop on narrative understanding* (pp. 37–43).

Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Johnson, R. A., Miller, I., & Freund, J. E. (2000). *Probability and statistics for engineers* (Vol. 2000). Pearson Education London.

Kasai, J., Cross, J., Ghazvininejad, M., & Gu, J. (2020a). Non-autoregressive machine translation with disentangled context transformer. In *International conference on machine learning* (pp. 5144–5155).

Kasai, J., Cross, J., Ghazvininejad, M., & Gu, J. (2020b). Parallel machine translation with disentangled context transformer. *arXiv preprint arXiv:2001.05136*.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., & Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Kim, Y., Tran, D. T., & Ney, H. (2019). When and why is document-level context useful in neural machine translation? *arXiv preprint arXiv:1910.00294*.

Klein, G., Kim, Y., Deng, Y., Senellart, J., & Rush, A. (2017, July). OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, system demonstrations* (pp. 67–72). Vancouver, Canada: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/P17-4012`

Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram language modeling. In *1995 international conference on acoustics, speech, and signal processing* (Vol. 1, pp. 181–184).

Kulesza, A., & Pereira, F. (2008). Structured learning with approximate inference. In *Advances in neural information processing systems* (pp. 785–792).

Lebanoff, L., Song, K., & Liu, F. (2018). Adapting the neural encoder-decoder framework from single to multi-document summarization. *arXiv preprint arXiv:1808.06218*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., & Huang, F. J. (2005). Loss functions for discriminative training of energy-based models. In *International workshop on artificial intelligence and statistics* (pp. 206–213).

Lee, J., Mansimov, E., & Cho, K. (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Lees, R. B. (1957). *Syntactic structures.* JSTOR.

Levenshtein, V. I., et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, pp. 707–710).

Levy, R., & Jaeger, T. F. (2007). Speakers optimize information density through syntactic reduction. *Advances in neural information processing systems*, *19*, 849.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Li, X., Meng, Y., Yuan, A., Wu, F., & Li, J. (2020). Lava nat: A non-autoregressive translation model with look-around decoding and vocabulary attention. *arXiv preprint arXiv:2002.03084*.

Li, Y., Su, H., Shen, X., Li, W., Cao, Z., & Niu, S. (2017). Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957*.

Li, Z., Lin, Z., He, D., Tian, F., Qin, T., Wang, L., & Liu, T.-Y. (2019). Hint-based training for non-autoregressive machine translation. *arXiv preprint arXiv:1909.06708*.

Libovickỳ, J., & Helcl, J. (2018). End-to-end non-autoregressive neural machine translation with connectionist temporal classification. *arXiv preprint arXiv:1811.04719*.

Liu, D., Fu, J., Liu, P., & Lv, J. (2019). Tigs: An inference algorithm for text infilling with gradient search. *arXiv preprint arXiv:1905.10752*.

Liu, M., Buntine, W., & Haffari, G. (2018). Learning to actively learn neural machine translation. In *Proceedings of the 22nd conference on computational natural language learning* (pp. 334–344).

Ma, X., Zhou, C., Li, X., Neubig, G., & Hovy, E. (2019). Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*.

Ma, Z., Edunov, S., & Auli, M. (2021). A comparison of approaches to document-level machine translation. *arXiv preprint arXiv:2101.11040*.

Martins, A. F. T. (2012). *The geometry of constrained structured prediction: applications to inference and learning of natural language syntax* (Unpublished doctoral dissertation). Carnegie Mellon University.

McAllister, M. K., & Ianelli, J. N. (1997). Bayesian stock assessment using catch-age data and the sampling-importance resampling algorithm. *Canadian Journal of Fisheries and Aquatic Sciences*, *54*(2), 284–300.

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

Meister, C., Vieira, T., & Cotterell, R. (2020). If beam search is the answer, what was the question? *arXiv preprint arXiv:2010.02650*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association.*

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *2011 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5528–5531).

Minsky, M., & Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*.

Mostafazadeh, N., Chambers, N., He, X., Parikh, D., Batra, D., Vanderwende, L., ... Allen, J. (2016). A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (second ed.). New York, NY, USA: Springer.

Opper, M., & Haussler, D. (1991). Generalization performance of bayes optimal classification algorithm for learning a perceptron. *Physical Review Letters*, *66*(20), 2677.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., ... Auli, M. (2019). fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of naacl-hlt 2019: Demonstrations.*

Over, P., & Yen, J. (2004). An introduction to duc-2004. *National Institute of Standards and Technology*.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318).

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

Ramachandran, P., Paine, T. L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., ... Huang, T. S. (2017). Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*.

Ran, Q., Lin, Y., Li, P., & Zhou, J. (2019). Guiding non-autoregressive neural machine translation decoding with reordering information. *arXiv preprint arXiv:1911.02215*.

Ranzato, M., Chopra, S., Auli, M., & Zaremba, W. (2015). Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Rezende, D., & Mohamed, S. (2015). Variational inference with normalizing flows. In *International conference on machine learning* (pp. 1530–1538).

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

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Schmaltz, A., Rush, A. M., & Shieber, S. M. (2016). Word ordering without syntax. *arXiv preprint arXiv:1604.08633*.

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Shao, C., Zhang, J., Feng, Y., Meng, F., & Zhou, J. (2020). Minimizing the bag-of-ngrams difference for non-autoregressive neural machine translation. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 198–205).

Shen, D., Celikyilmaz, A., Zhang, Y., Chen, L., Wang, X., Gao, J., & Carin, L. (2019). Towards generating long and coherent text with multi-level latent variable models. *arXiv preprint arXiv:1902.00154*.

Shu, R., Lee, J., Nakayama, H., & Cho, K. (2020). Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 8846–8853).

Slaney, M., & Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine*, *25*(2), 128–131.

Socher, R., Lin, C. C., Manning, C., & Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 129–136).

Sun, Z., Li, Z., Wang, H., Lin, Z., He, D., & Deng, Z.-H. (2019). Fast structured decoding for sequence models. *arXiv preprint arXiv:1910.11555*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Tan, B., Yang, Z., AI-Shedivat, M., Xing, E. P., & Hu, Z. (2020). Progressive generation of long text with pretrained language models. *arXiv preprint arXiv:2006.15720*.

Tan, X., Qin, T., Soong, F., & Liu, T.-Y. (2021). A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561*.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., & Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning* (pp. 2722–2731).

Tokarchuk, E., Rosendahl, J., Wang, W., Petrushkov, P., Lancewicki, T., Khadivi, S., & Ney, H. (2021). Towards reinforcement learning for pivot-based neural machine translation with non-autoregressive transformer. *arXiv preprint arXiv:2109.13097*.

Tromble, R., & Eisner, J. (2009). Learning linear ordering problems for better translation. In *Proceedings of the 2009 conference on empirical methods in natural language processing: Volume 2-volume 2* (pp. 1007–1016).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Viterbi, A. (1967, April). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260-269. doi: 10.1109/TIT.1967.1054010

Wan, X., Luo, F., Sun, X., Huang, S., & Yao, J.-g. (2019). Cross-language document summarization via extraction and ranking of multiple summaries. *Knowledge and Information Systems*, *58*(2), 481–499.

Wang, C., Zhang, J., & Chen, H. (2018). Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*.

Wang, J., Yu, F., Chen, X., & Zhao, L. (2019). Admm for efficient deep learning with global convergence. *arXiv preprint arXiv:1905.13611*.

Wang, Y., Tian, F., He, D., Qin, T., Zhai, C., & Liu, T.-Y. (2019). Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 5377–5384).

Wei, B., Wang, M., Zhou, H., Lin, J., & Sun, X. (2019). Imitation learning for non-autoregressive neural machine translation. *arXiv preprint arXiv:1906.02041*.

Welleck, S., Brantley, K., Daumé III, H., & Cho, K. (2019). Non-monotonic sequential text generation. *arXiv preprint arXiv:1902.02192*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... others (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, W., & Carpuat, M. (2020). Editor: an edit-based transformer with repositioning for neural machine translation with soft lexical constraints. *arXiv preprint arXiv:2011.06868*.

Zellers, R., Bisk, Y., Schwartz, R., & Choi, Y. (2018). Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*.

Zhang, W., Feng, Y., Meng, F., You, D., & Liu, Q. (2019). Bridging the gap between training and inference for neural machine translation. *arXiv preprint arXiv:1906.02448*.

Zhang, W., Yoshida, T., & Tang, X. (2011). A comparative study of tf* idf, lsi and multi-words for text classification. *Expert Systems with Applications*, *38*(3), 2758–2765.

Zhao, J., Liu, M., Gao, L., Jin, Y., Du, L., Zhao, H., ... Haffari, G. (2020). Summpip: Unsupervised multi-document summarization with sentence graph compression. In *Proceedings of the 43rd international acm sigir conference on research and development in information retrieval* (pp. 1949–1952).

Zhao, L., Xu, J., Lin, J., Zhang, Y., Yang, H., & Sun, X. (2020). Graph-based multi-hop reasoning for long text generation. *arXiv preprint arXiv:2009.13282*.

Zhou, C., Neubig, G., & Gu, J. (2019). Understanding knowledge distillation in non-autoregressive machine translation. *arXiv preprint arXiv:1911.02727*.

Zhou, J., & Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1127–1137).

Zhu, W., Hu, Z., & Xing, E. (2019). Text infilling. *arXiv preprint arXiv:1901.00158*.

Zweig, G., & Burges, C. J. (2011). The microsoft research sentence completion challenge. *Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-2011-129*.