Neural Structured Prediction using Iterative Refinement with Applications to Text and Molecule Generation

by

Elman Mansimov

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2021

Professor Kyunghyun Cho

Acknowledgements

First and foremost I would like to thank my advisor, Kyunghyun Cho, to whom I am forever indebted. I have benefited enormously from his guidance throughout my Ph.D. His ideas, technical advice, the vision of many different topics, and our discussions helped me grow into the researcher I am today. His enthusiasm and optimism helped me go through many of the challenges during the Ph.D. Our recurring meetings was one of the things I was looking forward to during the week and will miss after graduation. I would like to thank my thesis committee members He He, Rob Fergus, Yaser Al-Onaizan, and Dipanjan Das for providing me valuable feedback for improving my thesis.

I would like to thank Kyunghyun Cho, Sam Bowman, and He He for co-organizing weekly meetings of the Machine Learning and Language (ML^2) group. The lively discussions and great presentations of the current state of NLP and current work in progress in the lab have been a source of many ideas and inspirations.

I am deeply thankful to Shirley Palma and Shenglong Wang for keeping the GPUs running smoothly and helping resolve many of the technical difficulties on the compute cluster. I am thankful to Hong Tam and Santiago Pizzini for administrative help.

I had the pleasure of starting Ph.D. together with Ilya Kostrikov, Roberta Raileanu, Alex Rives, Will Whitney, Isaac Henrion, and Jake Zhao in 2016. Throughout this journey, I had the pleasure of sharing the office space with Ilya Kostrikov, Isaac Henrion, Roberta Raileanu, Yacine Jernite, Jake Zhao, Dieterich Lawson, and Kianté Brantley. I also had a lot of fun living together with Cinjon Resnick, Martin Arjovsky, Will Whitney, David Brandfonbrener, and Evgenii Nikishin at 385 Himrod St.

I am very grateful that I had an opportunity to interact and spend time with many of my brilliant colleagues at NYU: Aahlad Puli, Aaron Zweig, Adina Williams, Aditya

Ramesh, Adrian Lancucki, Aishwarya Kamath, Alex Nowak, Alex Rives, Alex Wang, Alex Warstadt, Alfredo Canziani, Amanpreet Singh, Andrew Drozdov, Behnam Neyshabur, Cinjon Resnick, Clara Vania, Daniele Panozzo, Davi Colli Tozoni, Denis Yarats, Denis Zorin, Emily Denton, Ethan Perez, Francis Williams, Hanxiao Shen, He He, Iacer Calixto, Kyle Cranmer, Ilia Kulikov, Ilya Kostrikov, Isaac Henrion, Jake Zhao, Jason Lee, Jason Phang, Jiakai Zhang, Joan Bruna, Katharina Kann, Katrina Evtimova, Kelly Zhang, Konrad Zolna, Krzysztof Geras, Kyunghyun Cho, Laura Graesser, Luca Venturi, Luiza Sayfullina, Jaan Altosaar, Mark Goldstein, Martin Arjovsky, Matt Morse, Michael Mathieu, Mikael Henaff, Mimee Xu, Min Jae Song, Dieterich Lawson, Mukund Sudarshan, Nan Wu, Nikita Nangia, Nishant Subramani, Omar Mahmood, Pablo Sprechmann, Patrick Cousot, Phu Mon Htut, Raghav Singhal, Rahul Krishnan, Rajesh Ranganath, Raphel Shu, Ren Yi, Kianté Brantley, Richard Pang, Rob Fergus, Roberta Raileanu, Rodrigo Nogueira, Sainbayar Sukhbaatar, Sam Bowman, Sean Welleck, Sebastien Jean, Seokho Kang, Sheng-Fu Wang, Stanislas Lauly, Subhash Khot, Thibault Févry, Tim Vieira, Vlad Sobol, Will Whitney, William Falcon, Xiang Zhang, Xintian Han, Yacine Jernite, Yada Pruksachatkun, Yann LeCun, Yixin Hu, Youngduck Choi, Zhongshi Jiang, and other people that I'm surely forgetting.

During my Ph.D., I had an opportunity to do research internships at Google Research, DeepMind, and Facebook AI Research. I would like to thank my mentors and collaborators at Google in Bay Area and Montreal for such a great learning experience: Mitchell Stern, Mia Chen, Orhan Firat, Jakob Uszkoreit, Puneet Jain, Ruoming Pang, Yonghui Wu, Ross Goroshin, Tyler Zhu, Hugo Larochelle, and many others I interacted with at Brain and Translate teams. I would like to thank Gábor Melis, Lei Yu, and Language Team at DeepMind for a wonderful internship experience in London. I am thankful to Rob Fergus for giving me a chance to intern at Facebook AI Research in New

York during my first year. Finally, I am very grateful to Yaser Al-Onaizan, Yi Zhang, Saab Mansour, and the folks at Amazon for offering me an amazing opportunity to start my first full-time job as an Applied Scientist at Amazon AWS AI in New York.

The times before I started my Ph.D. were as important to this thesis as the Ph.D. journey itself. I would like to give special thanks to Ruslan Salakhutdinov for allowing me to kick-start my career in deep learning research as an undergraduate student. I am indebted to Nitish Srivastava for being patient with me and teaching me the essentials of being a great researcher, Jimmy Ba for many interesting discussions and teaching me how to think big in research, Roger Grosse, Emilio Parisotto, Tony Wu, and Shun Liao for being great collaborators during and after my time at the University of Toronto.

I would like to thank my Toronto pals Fahad, Bola, Shawn, David, and Jeff for being a great company and staying in touch.

Finally, I would like to thank my family and love of my life Derin Cengiz for love and support during this journey.

Abstract

Humans excel at generating structured data in the form of images, text, speech, molecules, computer code, and others. Researchers have spent several decades proposing various solutions for the effective generation of these structured objects in a data-driven way, known as *structured prediction*. With the revival of deep neural networks, autoregressive models that process structured objects in fixed left-to-right monotonic ordering became a de-facto solution for this problem. Notable successes of autoregressive models include neural machine translation [Sutskever et al., 2014, Bahdanau et al., 2014, Vaswani et al., 2017], open-ended text generation [Radford et al., 2019, Brown et al., 2020], text-to-speech synthesis [van den Oord et al., 2016a], among many.

Despite the considerable success of autoregressive models on many applications, a natural question arises whether alternative approaches are possible for structured prediction. This thesis describes a novel method for structured prediction based on the principle of *iterative refinement* with a particular focus on applications to text and molecule generation. We first introduce the iterative refinement framework for text generation. Starting from the blank sentence, the iterative refinement approach gradually refines text over multiple steps. Using this approach, we show that we can flexibly generate the text in various ways, such as generate all or some words in parallel and generate text according to the ordering learned from the data. We show that iterative refinement achieves competitive performance compared to autoregressive models while delivering a speedup in decoding. We conclude this thesis by showing how we can adapt the iterative refinement framework originally introduced for text generation for molecule generation. In particular, we demonstrate two iterative refinement approaches for molecular graph generation and molecular geometry prediction. We anticipate that models based on the iterative refinement will be broadly applicable to other domains of interest.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans process and interact with a vast amount of structured data daily. These objects take a particular structure and are in the form of images, text, audio, computer code, etc. Humans excel at generating such data and communicating the findings to the rest of the world. But how can we create machines that generate such structured data as well or better than humans do?

Structured prediction is an area of machine learning focusing on generating structured objects in a data-driven way. One of the earliest examples of statistical structured prediction systems goes back to IBM's speech transcription system [Jelinek, 1976] developed in the 1970s. Another notable early success of structured prediction systems is the Graph Transformer network [Bottou et al., 1997]. It was the first end-to-end structured prediction system trained with backpropagation [LeCun, 1985, Rumelhart et al., 1986] that read the bank checks. Other conventional applications of structured prediction systems include machine translation [Brown et al., 1993, Och and Ney, 2004, Tillmann and Ney, 2003], sequence labeling and segmentation [Lafferty et al., 2001, Sarawagi and Cohen, 2004], discriminative dependency and constituency parsing [McDonald et al., 2005, Finkel et al., 2008], among others.

With the revival of deep neural networks starting from the 2000s-2010s [Krizhevsky et al., 2012, Ciresan et al., 2012, Hinton et al., 2012], structured prediction models became dominated by neural networks. In particular, neural autoregressive models [Mikolov et al., 2010] that generate structured data in a fixed left-to-right monotonic order became a de facto approach for neural structured prediction. Some major recent successes of neural autoregressive models include neural machine translation [Bahdanau et al., 2014, Sutskever et al., 2014, Vaswani et al., 2017], neural dialogue generation [Vinyals and Le, 2015, Serban et al., 2016, Bordes and Weston, 2017, Adiwardana et al., 2020], neural language modeling [Mikolov et al., 2010, Zaremba et al., 2014, Melis et al., 2018, Dai et al., 2019], open-ended text generation [Radford et al., 2019, Brown et al., 2020], response suggestion for email [Kannan et al., 2016, Chen et al., 2019], text-to-speech synthesis [van den Oord et al., 2016a, Wang et al., 2017b, Shen et al., 2018], in-silico molecule generation [Brown et al., 2018, Gupta et al., 2018], protein property prediction [Rives et al., 2019, Rao et al., 2019, Ingraham et al., 2019a] and etc.

Despite the considerable success of neural autoregressive models, a natural question arises whether alternative approaches are possible for the generation of structured objects. This thesis introduces the novel approach for structured prediction based on the principle of *iterative refinement*. First, we present the iterative refinement framework for parallel decoding of sentences in neural machine translation (chapter 3). Then, we present the generalized framework of sequence generation that unifies decoding from directed and undirected sequence models. Using this framework, we show that we can iteratively decode text using a BERT-like machine translation model [Devlin et al., 2019, Lample and Conneau, 2019] (chapter 4). Finally, we demonstrate the generality of iterative refinement framework beyond text, particularly to molecular graph generation (chapter 5) and molecular geometry prediction (chapter 6).

1.1 List of Contributions

- Jason Lee*, **Elman Mansimov***, Kyunghyun Cho.

  Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. *EMNLP, 2018.*

  Source code, preprocessed datasets and pretrained models can be found at `https://github.com/nyu-dl/dl4mt-nonauto`.

  Together with Jason Lee and Kyunghyun Cho, I started the project and conceived the initial idea. Together with Jason Lee, I implemented the code, ran, and analyzed the experiments. Myself, Jason Lee and Kyunghyun Cho iteratively refined the idea of the project and wrote the paper.

- **Elman Mansimov**, Alex Wang, Sean Welleck, Kyunghyun Cho.

  A Generalized Framework of Sequence Generation with Application to Undirected Sequence Models. *arXiv preprint, 2019.*

  Source code, preprocessed datasets and pretrained models can be found at `https://github.com/nyu-dl/dl4mt-seqgen`.

  Together with Alex Wang and Kyunghyun Cho, I started the project and conceived the initial idea. With the help of Alex Wang, I implemented code, ran, and analyzed experiments. Myself, Alex Wang, and Kyunghyun Cho have written the initial version of the draft. Sean Welleck joined in the later stages of the project. He implemented code and ran experiments with reinforcement learning models in the paper.

- Omar Mahmood, **Elman Mansimov**, Richard Bonneau, Kyunghyun Cho.

Masked graph modeling for molecule generation. *arXiv preprint, 2020.*

Source code, preprocessed datasets and pretrained models can be found at `https://github.com/nyu-dl/dl4chem-mgm`.

Together with Omar Mahmood we conceived the initial idea and started the project. Omar Mahmood ran all of the experiments, except for the baseline models. I participated in the weekly meetings and advised on the next steps for the project. I wrote the paper together with Omar Mahmood and Kyunghyun Cho.

- **Elman Mansimov**, Omar Mahmood, Seokho Kang, Kyunghyun Cho.

  Molecular geometry prediction using a deep generative graph neural network, *Nature Scientific Reports, 2019*

  Source code, preprocessed datasets and pretrained models can be found at `https://github.com/nyu-dl/dl4chem-geometry`.

  Seokho Kang and Kyunghyun Cho conceived the initial idea and started the project. Myself, Omar Mahmood, and Seokho Kang ran the experiments and further refined the project's idea. Myself, Omar Mahmood, Seokho Kang, and Kyunghyun Cho refined the idea and wrote the paper.

Chapter 2

Background

We start this section by defining the sequences and what it means to build generative models of the sequences (chapter 2.1). We then describe denoising autoencoder (chapter 2.2), an approach initially proposed for representation learning, that is behind the iterative refinement framework. We describe two fundamental generative models of sequences, including latent-variable models (chapter 2.3) and autoregressive models (chapter 2.4). Finally, we outline several ways of parameterizing these models using deep neural networks (chapter 2.5) and explain how they can be extended to other structured objects in particular graphs (chapter 2.6).

2.1 Sequences and generative models

Formally, lets define a variable $Y = (y_1, y_2, ..., y_T)$ as a sequence of length T containing symbols y_i that belong to the vocabulary V. An example of the sequence is the sentence ”I was running in the park” consisting of symbols (tokens): ”I”, ”was”, ”running”, ”in”, ”the”, ”park” that are part of the English vocabulary. The goal of the sequence generation is to define a process of generating such sequences Y.

Under the probabilistic formulation, we assume that exists a distribution $p^*(Y)$ of valid sequences Y. The process of generating valid sequences becomes a sampling procedure from the distribution $p^*(Y)$. We can extend this distribution into $p^*(Y|X)$ by conditioning the variable Y on another variable X. When variable X is a non-empty set such as a sequence the generative modeling task becomes conditional. Examples of such conditional tasks include machine translation, text summarization, image captioning. When variable X is an empty set $X = \emptyset$, the generative modeling task becomes unconditional. A representative example of an unconditional sequence generation task is language modeling.

Generative modeling under the probabilistic formulation becomes the task of finding a distribution $p_\theta(Y|X)$ that approximates underlying distribution $p^*(Y|X)$ well. The approximate distribution $p_\theta(Y|X)$ is modeled using a function f_θ containing parameters θ. The parameters θ are learned by minimizing the distance between ground-truth and approximate distributions. Typically, a KL-divergence metric $KL(p^*||p_\theta)$ is used as a distance function.

In practice we don't have direct access to the ground-truth distribution $p^*(Y|X)$. Instead we estimate it using a distribution $p(Y|X)$ over the dataset $D = \big((X_1, Y_1), (X_2, Y_2)..., (X_n, Y_n)\big)$ of valid sequences X and Y. The learning objective becomes $KL(p||p_\theta)$ which is equivalent to maximizing the likelihood $p_\theta(D)$ of the dataset D. Under this formulation, the challenge of building a successful generative model of sequences ends up lying in finding a right parameterization function f_θ and decomposition of probability distribution $p(Y|X)$. In the next sections, we describe several foundational ways of parametrizing such a function and probability distribution to build a generative model of sequences.

2.2 Denoising autoencoder

The idea behind the classical autoencoder [Oja, 1991], such as the one used in [Bengio, 2007], is to represent the input Y using the low-dimensional continuous representation h. To accomplish that, the input Y gets encoded into a low dimensional representation $h = f(Y)$, followed by decoding that hidden representation back into the original space $\hat{Y} = g(h) = g(f(Y))$. The parameters of autoencoder are learned by minimizing the distance between the original input Y and reconstructed input $\hat{Y}$. Typically, the cross-entropy error is used for discrete variables, while the mean-squared error is used for continuous variables. Due to autoencoder's deterministic nature, this model can not be used to generate new objects and can only be used to downsample existing input into low-dimensional representation. The idea of autoencoders for representation learning has been part of the historical landscape of neural networks for decades [LeCun, 1987, Bourlard and Kamp, 1988, Hinton and Zemel, 1993].

While the deep autoencoder outperforms shallow methods such as PCA for learning low-dimensional representation of data [Bengio et al., 2006, Hinton and Salakhutdinov, 2006], it can easily overfit to data and learn an obvious solution by simply copying the input. To avoid this issue, [Vincent et al., 2008] hypothesized that good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input. In particular, [Vincent et al., 2008] proposed the version of an autoencoder that reconstructs a clean input from a corrupted version of it. Specifically, we design a corruption function $C(\tilde{Y}|Y)$ and pass the noisy input sampled using this corruption function $\tilde{Y} \sim C(\tilde{Y}|Y)$ to the autoencoder. The resulting model is called the *denoising autoencoder*. Similar to the autoencoder, parameters of denoising autoencoder are learned by minimizing the reconstruction error. The corruption function is designed so that the corrupted input is not too far from the clean input. Typically for continuous variables, salt-and-pepper noise is used [Vincent et al., 2008]. For discrete variables, random replacement/dropping/swapping of symbols is used [Hill et al., 2016].

While denoising autoencoder was originally proposed as an improved way of learning representations of data, it was shown later that it is possible to sample data from denoising autoencoder. [Alain and Bengio, 2012] showed that the denoising autoencoder trained on continuous data with small Gaussian corruption and mean-squared error loss implicitly learns to find the direction $\nabla_Y \log p(Y)$ in the output space that maximizes the underlying true, data-generating distribution $\log p(Y)$. Equivalently, the denoising autoencoder captures the score (the derivative of log density with respect to input). Additionally, [Alain and Bengio, 2012] showed that novel samples can be generated on artificial datasets by using the denoising autoencoder as a score estimator for Metropolis-Hastings MCMC accept/reject decision. [Bengio et al., 2013] has generalized the observation made by [Alain and Bengio, 2012] for any data, by showing that we can recover a consistent estimator of $p(Y)$ through a pseudo-Gibbs Markov chain that alternates between sampling from $p(Y|\tilde{Y})$ (denoising autoencoder) and sampling from $C(\tilde{Y}|Y)$ (corruption function) for several iterations. Using this pseudo-Gibbs Markov chain we can generate novel sequence samples from a denoising autoencoder. While [Bengio et al., 2013] have presented a general framework for using denoising autoencoders as generative models of data, they empirically validated it on artificial datasets and MNIST digits [LeCun et al., 1998].

2.3 Latent-variable models of sequences

One popular approach for building a generative model of data is to introduce a latent variable Z that aims to capture dependencies among the outputs inside the compact representation. In case of sequences, this latent-variable model can be straightforwardly extended by introducing the set of latent variables $Z = (z_1, z_2, ..., z_T)$ that aim to capture dependencies among the outputs of the sequence Y at each timestep t. The likelihood of the sequence becomes

$$p(Y|X) = \prod_{t=1}^{T} \int_{Z_t} p(Y_t|X_t, Z_t)p(Z_t|Z_{<t})dZ_t \tag{2.1}$$

Under the first-order Markov assumption, where latent variable at each timestep only depends on the latent variable at the previous timestep the likelihood objective in Eq. 2.1 decomposes into

$$p(Y|X) = \prod_{t=1}^{T} \int_{Z_t} p(Y_t|X_t, Z_t)p(Z_t|Z_{t-1})dZ_t \tag{2.2}$$

Hidden Markov Model (HMM) [Rabiner and Juang, 1986] and linear-Gaussian state-space models [Kalman, 1960, Schweppe, 1965] are the well-known latent variable models of sequence that assume the described first-order Markov property. The difference between these two models lies in that the linear-Gaussian state-space model has continuous latent variables, whereas HMMs have discrete latent variables.

Learning the parameters θ of the latent-variable model requires maximizing the likelihood in Eq. 2.2. However, directly maximizing this likelihood is difficult since it requires marginalizing latent variables at each timestep. Instead we simplify this problem by first defining the lower bound $F(Q)$ of the log-likelihood $\log p(Y|X)$ in Eq. 2.2.

$$\log p(Y|X) = \log \sum_{t=1}^{T} \int_{Z_t} p(Y_t|X_t, Z_t) p(Z_t|Z_{t-1}) dZ_t = \log \sum_{t=1}^{T} \int_{Z_t} p(X_t, Y_t, Z_t) dZ_t = \log \sum_{t=1}^{T} \int_{Z_t} Q(Z_t) \frac{p(X_t, Y_t, Z_t)}{Q(Z_t)} dZ_t \geq \sum_{t=1}^{T} \int_{Z_t} Q(Z_t) \log \frac{p(X_t, Y_t, Z_t)}{Q(Z_t)} = \sum_{t=1}^{T} \int_{Z_t} Q(Z_t) \log P(X_t, Y_t, Z_t) dZ_t - \log \sum_{t=1}^{T} \int_{Z_t} Q(Z_t) \log Q(Z_t) dZ_t = F(Q) \quad (2.3)$$

The inequality is known as the Jensen's inequality and follows from the fact that log function is concave. $Q(Z_t)$ is defined as an approximate of the true posterior $P(Z_t|X_t)$. The lower bound $F(Q)$ is maximized by alternating between

- Updating the approximate posterior $Q(Z_t)$ while keeping parameters θ fixed;
- Updating the parameters θ while keeping approximate posterior fixed $Q(Z_t)$;

which is known as the EM (Expectation-Maximization) algorithm [Dempster et al., 1977]. The EM algorithm has been proposed several times before it was given its name by [Dempster et al., 1977]. One particular instance of it is the Baum-Welch algorithm, which was originally proposed to train HMMs. Baum-Welch algorithm recursively computes the probabilities $\alpha_i(t)$ and $\beta_i(t)$ using forward-backward procedure. $\alpha_i(t)$ represents a joint probability $p(X_1, X_2, ...X_t, Y_t = i)$ of input $X_1, X_2, ..., X_t$ and latent $Y_t = i$. $\beta_i(t)$ represents a joint probability $p(X_{t+1}, X_{t+2}, ...X_T, Y_t = i)$ of input $X_{t+1}, X_{t+2}, ..., X_T$ and latent $Y_t = i$. Given those forward and backward probabilities the Baum-Welch algorithm updates the approximate posterior. Generation in HMMs, i.e. finding the most likely sequence $\hat{Y} = \arg\max_Y \log p(Y|X)$, is done approximately using a dynamic programming approach named Viterbi decoding [Viterbi, 1967].

Although latent-variable models of sequences, in particular Kalman filter [Kalman, 1960] and HMMs, have been successfully applied to navigation, control of satellites, numerous problems in NLP [Brown et al., 1993] and speech processing [Jelinek et al., 1975], these models suffer from several limitations. The linear relationship between latent variables and observations in linear-Gaussian state-space models makes it not well suited for many real-world applications where the dynamics are nonlinear. HMMs, on the other hand, require exponential growth in the number of discrete latent variables to model the higher-order dynamics of the sequence. Finally, under the first-order Markov assumption, the future predictions are only dependent on most recent observations that make these models unsuitable to sequences with long-term dependencies.

Recent latent-variable models focus on maximizing the lower bound in Eq. 2.3 in an end-to-end manner. The approximate posterior $Q(Z_t)$ is predicted by an inference network and is shared (i.e., amortized) across the dataset [Kingma and Welling, 2014, Rezende et al., 2014]. We refer the readers to [Kingma, 2017], which covers the recent foundational advances in amortized latent-variable models. [Kim et al., 2018] gives an overview of applications of amortized latent-variable models to natural language processing.

2.4 Autoregressive sequence models

Autoregressive sequence models directly decompose the distribution $p(Y|X)$ as a product of conditionally dependent per step distributions $p(y_t|y_{<t}, x)$. Unlike latent variable models, there are no additional variables introduced into the model. This decomposition results in the likelihood in the form of:

$$p(y|x) = \prod_{t=1}^{T} p(y_t|y_{<t}, x) \tag{2.4}$$

Unlike latent-variable sequence models, the likelihood objective in Eq. 2.4 can be maximized directly without requiring expensive marginalization term. Similar to latent variable models, generation is done via ancestral sampling, i.e. by generating each output variable y_t sequentially at each timestep t.

A classical example of the autoregressive model of sequences is an n-gram model [Shannon, 1948, Brown et al., 1992, Kneser and Ney, 1993, Niesler et al., 1998]. The n-gram model assumes the n-th order Markov property, where each output variable y_t is only conditioned on the previous $n-1$ output variables $(y_{t-n}, y_{t-n+1}, ..., y_{t-1})$. Using the definition of conditional probability, the likelihood in Eq 2.4 becomes

$$p(y|x) = \prod_{t=1}^{T} p(y_t|y_{t-n}, y_{t-n+1}, ..., y_{t-1}, x) = \frac{p(y_{t-n}, y_{t-n+1}, ..., y_{t-1}, y_t, x)}{p(y_{t-n}, y_{t-n+1}, ..., y_{t-1}, x)} \tag{2.5}$$

The numerator in Eq 2.5 is computed by counting the number of occurrences of $(y_{t-n}, y_{t-n+1}, ..., y_{t-1}, y_t, x)$ in the training corpus. The denominator is computed by summing the occurrences of $(y_{t-n}, y_{t-n+1}, ..., y_{t-1}, y_t = v, x)$ in the training corpus for each symbol v in the vocabulary V. Due to the counting procedure, if the n-gram in the numerator is not found in the training set, the conditional probability $p(y|x)$ becomes equal to 0. To avoid assigning zero probability to sequences never observed in the training set, variety of smoothing [Chen and Goodman, 1996] and back-off [Katz, 1987, Kneser and Ney, 1995] techniques are used. When using add-α smoothing in an n-gram model, a small constant α is added to the numerator and constant $\alpha|V|$ is added to the denominator in Eq 2.5. In a back-off n-gram model, if the n-gram is not found in the training set, it is approximated by backing off to the $(n-1)$-gram model.

Although n-gram models have been successful in many applications such as machine translation, the main limitation of the n-gram model lies in its inability to generalize to sentences with unseen n-grams [Bengio et al., 2003]. This limitation motivates neural sequence models that use non-linear function approximators such as a neural network to map the discrete symbols into the continuous space. After learning such a neural network, the similar sequences will have similar representations in the continuous space. The next section describes several ways of building a neural network-based non-linear function approximator for sequence models.

2.5 Parameterization of neural sequence models

The first representative way of parametrizing neural sequence models is the neural probabilistic language model by [Bengio et al., 2003]. The model is a two-layer feed-forward neural network with a tanh nonlinearity. Similar to n-gram models, the neural probabilistic language model assumes n-th order Markov property and takes a fixed number of tokens as an input (number of tokens ≤ 10). The neural network outputs a probability distribution for each word in the vocabulary using a softmax layer. [Bengio et al., 2003] has shown that the neural probabilistic language model achieves lower perplexity (or equivalent higher likelihood) on the test set compared to classical n-gram models. [Schwenk and Gauvain, 2005] has further shown that the neural probabilistic language model outperforms the n-gram language model when reranking the output candidates of the speech transcription system. Although the feedforward neural language model doesn't suffer from the generalization issue present in the classical n-gram model, it is still limited by the n-th order Markov property.

The recurrent neural language model overcomes this limitation by conditioning the variable at timestep t on all preceding variables in the sequence. In particular, the hidden layer h_t of recurrent neural network at timestep t representation captures information from all preceding symbols $(y_1, y_2, ..., y_{t-1})$ and the currently processed symbol y_t in the sequence. Although recurrent neural networks have been proposed in the early 1990s [Elman, 1990], training them remained challenging. [Bengio et al., 1994] has identified vanishing and exploding gradient descent problems as challenges when training recurrent neural networks. [Mikolov et al., 2010] has successfully trained a recurrent neural language model that outperformed the n-gram model and used a gradient clipping technique to deal with the exploding gradient problem. Weighted temporal shortcut connections (skip-connections through time) were shown to be effective in a recurrent neural network to deal with the vanishing gradient problem. Famous examples of such connections include Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997a] with forget gates [Gers et al., 2000] and Gated recurrent unit (GRU) [Cho et al., 2014].

Although LSTMs and GRUs have demonstrated promising results on machine translation [Sutskever et al., 2014] and speech transcription [Graves et al., 2013] tasks in the early 2010s, the performance of these models quickly deteriorated as the target sequence length increased (for example in Figure 2, [Bahdanau et al., 2014] showed that performance of purely RNN based neural machine translation model deteriorates as sentence length becomes ≥ 20). Instead of using a fixed low-dimensional vector h_t that compresses the entire source sequence $(x_1, x_2, ..., x_{T'})$, [Bahdanau et al., 2014] proposed neural attention module that attends to the hidden states of tokens in the source sequence. In particular, given query vector Q and key-value vector pairs (K and V), the neural attention module computes the output vector as a weighted sum of all value vectors. The weights are computed by passing all combinations of (query, key) vector pairs through the multi-layer perceptron and normalizing those outputs with softmax.

LSTM/GRU combination with neural attention achieved state-of-the-art results on variety of conditional sequence generation problems such as machine translation [Bahdanau et al., 2014, Wu et al., 2016], parsing [Vinyals et al., 2015], text summarization [Rush et al., 2015] and others. Despite the enormous success of recurrent neural network models with attention, the sequential nature of recurrence during forward and backward pass prohibits the user from parallelizing computation across time. Specifically, to process the token at timestep t, the recurrent neural network needs first sequentially process tokens at timesteps $1, ..., t-1$. This issue becomes more evident when using recurrent neural networks to generate long sequences such as images pixel-by-pixel.

To be able to parallelize computation through time, several architectures without recurrent connections have been proposed [van den Oord et al., 2016a, van den Oord et al., 2016b, Gehring et al., 2017, Vaswani et al., 2017]. Among these, Transformer architecture [Vaswani et al., 2017] that only consists of feedforward and neural attention layers is the most impactful architecture [Devlin et al., 2019, Liu et al., 2019b, Yang et al., 2019, Radford et al., 2019, Brown et al., 2020]. The Transformer architecture outperforms LSTM with neural attention across several machine translation benchmarks.

Although during training the computation can be parallelized in these architectures through time, the decoding remains sequential. During decoding, tokens in the sequence need to be generated one at a time before passing these generated tokens as input to the model at the next timestep. The decoding procedure is approximate, i.e., there is no known polynomial algorithm for solving $\arg\max_Y \log p(Y|X)$ exactly. Among practical approximate decoding algorithms, greedy [Germann et al., 2001] and beam-search [Tillmann and Ney, 2003] decoding are most widely used. For sampling novel sequences, top-k sampling [Fan et al., 2018], and nucleus sampling [Holtzman et al., 2019] are typically used. Despite these limitations, autoregressive models with Transformer architecture remain to be a first choice for building generative models of sequences. A recent line of work scaling these models to billions of parameters have shown an impressive results on dialogue generation [Adiwardana et al., 2020], natural language understanding tasks [Wang et al., 2018a,Wang et al., 2019a,Raffel et al., 2019], machine translation [Lepikhin et al., 2020], open-ended text generation [Radford et al., 2019, Shoeybi et al., 2019, Brown et al., 2020] and protein property prediction [Rives et al., 2019]. It remains to been seen how far we can go performance-wise by continuing to scale these massive autoregressive models.

2.6 Graphs as a generalized version of sequences

This section shows that the latent-variable models and autoregressive models described in previous chapters can be adapted for the generation of graphs. Before we describe some of the representative generative models of graphs, we will first define the graph. The graph G is a pair of vertices V and edges E consisting of N vertices v_i and up to $\frac{N\times(N-1)}{2}$ edges e_{ij}. Vertex is denoted by $v_i = (i, t_i)$, where i is the unique index assigned to it, and $t_i \in C_v$ is its type. Edge is denoted by $e_{ij} = (v_i, v_j, r_{ij})$, where $i < j$ are the indices to the incidental vertices of this edge and $r_{ij} \in C_e$ is the type of this edge. We note that graphs can be in the form of a single large graph or a collection of small graphs. Examples of large graphs are social networks and community graphs. Examples of small graphs are molecules, computer programs, etc. In this thesis, we focus on the latter case, where given a dataset of small graphs, we build a generative model of graphs.

A simplest way to adapt a generative model of sequences to graphs is to apply invertible transformation of graphs into sequences, a process known as linearization.

Two canonical ways of linearizing the graphs are depth-first search (DFS) traversal and breadth-first search (BFS) traversal. Some of the earlier approaches of adapting generative models of sequences to graphs include: an autoregressive LSTM with attention [Bahdanau et al., 2014] that is successful at generating the linearized version of the parse trees obtained using depth-first search (DFS) traversal [Vinyals et al., 2015] and the variational autoencoder (VAE)-based model [Kingma and Welling, 2014] for generating molecules in their SMILES string representations [Gómez-Bombarelli et al., 2016].

In addition to generating sequences, autoregressive models are a popular choice for generating graphs. [Li et al., 2018a] proposed a deep generative model of graphs that predicts a sequence of transformation operations to generate a graph. [You et al., 2018b] proposed an RNN-based autoregressive generative model that generates components of a graph in breadth-first search (BFS) ordering. Further work on applying autoregressive modeling to graph generation has focused on speeding up autoregressive models during decoding and scaling them to larger graphs. [Liao et al., 2019] extended autoregressive models of graphs by adding blockwise parallel generation to speed up generation and improve scalability. [Dai et al., 2020] proposed an autoregressive generative model of graphs that utilizes sparsity to avoid generating the full adjacency matrix and generates novel graphs in log-linear time complexity.

While graph generation remains an exciting research area, the field's progress has been limited by the available datasets. The datasets used by [You et al., 2018a, Liao et al., 2019, Dai et al., 2019] are tiny compared to the size of the representative datasets such as ImageNet [Deng et al., 2009a], WMT'14 English-German translation [Bojar et al., 2014] used in computer vision and natural language processing. Datasets for benchmarking generative models of graphs [You et al., 2018a, Liao et al., 2019, Dai et al., 2019] include 100 2D grid graphs, 918 protein graphs, 3D point cloud graphs of 41 household objects, and 100 random lobster graphs. That's why in this thesis, we focus on molecular generation, which, unlike other graph datasets, have an order of hundreds of thousands of molecules [Brown et al., 2018].

Chapter 3

Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement

We propose a conditional non-autoregressive neural sequence model based on iterative refinement. The proposed model is designed based on the principles of latent variable models and denoising autoencoders, and is generally applicable to any sequence generation task. We extensively evaluate the proposed model on machine translation (En$\leftrightarrow$De and En$\leftrightarrow$Ro) and image caption generation, and observe that it significantly speeds up decoding while maintaining the generation quality comparable to the autoregressive counterpart.

3.1 Introduction

Conditional neural sequence modeling has become a *de facto* standard in a variety of tasks (see e.g. [Cho et al., 2015] and references therein). Much of this recent success is built on top of autoregressive sequence models in which the probability of a target sequence is factorized as a product of conditional probabilities of next symbols given all the preceding ones. Despite its success, neural autoregressive modeling has its weakness in decoding, i.e., finding the most likely sequence. Because of intractability, we must resort to suboptimal approximate decoding, and due to its sequential nature, decoding cannot be easily parallelized and results in a large latency (see e.g., [Cho, 2016]). This has motivated the recent investigation into non-autoregressive neural sequence modeling by [Gu et al., 2017] in the context of machine translation and [Oord et al., 2017] in the context of speech synthesis.

In this chapter, we propose a non-autoregressive neural sequence model based on iterative refinement, which is generally applicable to any sequence generation task beyond machine translation. The proposed model can be viewed as both a latent variable model and a conditional denoising autoencoder. We thus propose a learning algorithm that is hybrid of lowerbound maximization and reconstruction error minimization. We further design an iterative inference strategy with an adaptive number of steps to minimize the generation latency without sacrificing the generation quality.

We extensively evaluate the proposed conditional non-autoregressive sequence model and compare it against the autoregressive counterpart, using the state-of-the-art Transformer [Vaswani et al., 2017], on machine translation and image caption generation. In the case of machine translation, the proposed deterministic non-autoregressive models are able to decode approximately $2-3\times$ faster than beam search from the autoregressive counterparts on both GPU and CPU, while maintaining 90-95% of translation quality on IWSLT'16 En$\leftrightarrow$De, WMT'16 En$\leftrightarrow$Ro and WMT'14 En$\leftrightarrow$De. On image caption generation, we observe approximately $3\times$ and $5\times$ faster decoding on GPU and CPU, respectively, while maintaining 85% of caption quality.

3.2 Non-Autoregressive Sequence Models

Sequence modeling in deep learning has largely focused on autoregressive modeling. That is, given a sequence $Y = (y_1, \ldots, y_T)$, we use some form of a neural network to parametrize the conditional distribution over each variable y_t given all the preceding variables, i.e.,

$$\log p(y_t | y_{<t}) = f_\theta(y_{<t}),$$

where f_θ is for instance a recurrent neural network. This approach has become a *de facto* standard in language modeling [Mikolov et al., 2010]. When this is conditioned on an extra variable X, it becomes a conditional sequence model $\log p(Y|X)$ which serves as a basis on which many recent advances in, e.g., machine translation [Bahdanau et al., 2014, Sutskever et al., 2014, Kalchbrenner and Blunsom, 2013] and speech recognition [Chorowski et al., 2015, Chiu et al., 2017] have been made.

Despite the recent success, autoregressive sequence modeling has a weakness due to its nature of sequential processing. This weakness shows itself especially when we try to decode the most likely sequence from a trained model, i.e.,

$$\hat{Y} = \arg\max_Y \log p(Y|X).$$

There is no known polynomial algorithm for solving it exactly, and practitioners have relied on approximate decoding algorithms (see e.g. [Cho, 2016, Hoang et al., 2017]). Among these, beam search has become the method of choice, due to its superior performance over greedy decoding, which however comes with a substantial computational overhead [Cho, 2016].

As a solution to this issue of slow decoding, two recent works have attempted non-autoregressive sequence modeling. [Gu et al., 2017] have modified the Transformer [Vaswani et al., 2017] for non-autoregressive machine translation, and [Oord et al., 2017] a convolutional network [van den Oord et al., 2016a] for non-autoregressive modeling of waveform. Non-autoregressive modeling factorizes the distribution over a target sequence given a source into a product of conditionally independent per-step distributions:

$$p(Y|X) = \prod_{t=1}^{T} p(y_t|X),$$

breaking the dependency among the target variables across time. This allows us to trivially find the most likely target sequence by taking $\arg\max_{y_t} p(y_t|X)$ for each t, effectively bypassing the computational overhead and sub-optimality of decoding from an autoregressive sequence model.

This desirable property of exact and parallel decoding however comes at the expense of potential performance degradation [Kaiser and Bengio, 2016]. The potential modeling gap, which is the gap between the underlying, true model and the neural sequence model, could be larger with the non-autogressive model compared to the autoregressive one due to challenge of modeling the factorized conditional distribution above.

3.3 Iterative Refinement for Deterministic Non-Autoregressive Sequence Models

3.3.1 Latent variable model

Similarly to two recent works [Oord et al., 2017,Gu et al., 2017], we introduce latent variables to implicitly capture the dependencies among target variables. We however remove any stochastic behavior by interpreting this latent variable model, introduced immediately below, as a process of iterative refinement.

Our goal is to capture the dependencies among target symbols $y_1, \ldots, y_T$ given a source sentence X without auto-regression by introducing L intermediate random variables $Y^0, \ldots, Y^L$, where $Y^l = (y^l_1, \ldots, y^l_T)$, and marginalizing them out:

$$p(Y|X) = \sum_{Y^0,\ldots,Y^L} \left(\prod_{t=1}^{T} p(y_t|Y^L, X) \right) \left(\prod_{t=1}^{T} p(y_t^L|Y^{L-1}, X) \right) \cdots \left(\prod_{t=1}^{T} p(y_t^0|X) \right). \tag{3.1}$$

Each product term inside the summation is modelled by a deep neural network that takes as input a source sentence and outputs the conditional distribution over the target vocabulary V for each t.

Deterministic Approximation The marginalization in Eq. (3.1) is intractable. In order to avoid this issue, we consider two approximation strategies; deterministic and stochastic approximation. Without loss of generality, let us consider the case of single intermediate latent variable, that is $L = 1$. In the *deterministic* case, we set $\hat{y}^0_t$ to the most likely value according to its distribution $p(y^0_t|X)$, that is $\hat{y}^0_t = \arg\max_{y^0_t} p(y^0_t|X)$.

The entire lower bound can then be written as:

$$\log p(Y|X) \geq \left(\sum_{t=1}^{T} \log p(y_t|\hat{Y}^L, X)\right) + \cdots$$
$$+ \left(\sum_{t=1}^{T} \log p(y_t^1|\hat{Y}^0, X)\right) + \left(\sum_{t=1}^{T} \log p(\hat{y}_t^0|X)\right).$$

Stochastic Approximation In the case of *stochastic* approximation, we instead sample $\hat{y}_t^0$ from the distribution $p(y_t^0|X)$. This results in the unbiased estimate of the marginal log-probability $\log p(Y|X)$. Other than the difference in whether most likely values or samples are used, the remaining steps are identical.

Latent Variables Although the intermediate random variables could be anonymous, we constrain them to be of the same type as the output Y is, in order to share an underlying neural network. This constraint allows us to view each conditional $p(Y^l|\hat{Y}^{l-1}, X)$ as a single-step of refinement of a rough target sequence $\hat{Y}^{l-1}$. The entire chain of L conditionals is then the L-step iterative refinement. Furthermore, sharing the parameters across these refinement steps enables us to dynamically adapt the number of iterations per input X. This is important as it substantially reduces the amount of time required for decoding, as we see later in the experiments.

Training For each training pair (X, Y^*), we first approximate the marginal log-probability. We then minimize

$$J_{\text{LVM}}(\theta) = -\sum_{l=0}^{L+1} \left(\sum_{t=1}^{T} \log p_\theta(y_t^*|\hat{Y}^{l-1}, X)\right), \tag{3.2}$$

where $\hat{Y}^{l-1} = (\hat{y}_1^{l-1}, \ldots, \hat{y}_T^{l-1})$, and θ is a set of parameters. We initialize $\hat{y}_t^0$ (t-th target word in the first iteration) as $x_{t'}$, where $t' = (T'/T) \cdot t$. T' and T are the lengths of the source X and target Y^*, respectively.

3.3.2 Denoising Autoencoder

The proposed approach could instead be viewed as learning a conditional denoising autoencoder which is known to capture the gradient of the log-density. That is, we implicitly learn to find a direction Δ_Y in the output space that maximizes the underlying true, data-generating distribution $\log P(Y|X)$. Because the output space is discrete, much of the theoretical analysis by [Alain and Bengio, 2012] are not strictly applicable. We however find this view attractive as it serves as an alternative foundation for designing a learning algorithm.

Training We start with a corruption process $C(Y|Y^*)$, which introduces noise to the correct output Y^*. Given the reference sequence Y^*, we sample $\tilde{Y} \sim C(Y|Y^*)$ which becomes as an input to each conditional in Eq. (3.1). Then, the goal of learning is to maximize the log-probability of the original reference Y^* given the corrupted version. That is, to minimize

$$J_{\text{DAE}}(\theta) = -\sum_{t=1}^{T} \log p_\theta(y_t^* | \tilde{Y}, X). \tag{3.3}$$

Once this cost J_{DAE} is minimized, we can recursively perform the maximum-a-posterior inference, i.e., $\hat{Y} = \arg\max_Y \log p_\theta(Y|X)$, to find $\hat{Y}$ that (approximately) maximizes $\log p(Y|X)$.

Corruption Process C There is little consensus on the best corruption process for a sequence, especially of discrete tokens. In this work, we use a corruption process proposed by [Hill et al., 2016], which has recently become more widely adopted (see, e.g., [Artetxe et al., 2017, Lample et al., 2017]). Each y_t^* in a reference target $Y^* = (y_1^*, \ldots, y_T^*)$ is corrupted with a probability $\beta \in [0, 1]$. If decided to corrupt, we either (1) replace y_{t+1}^* with this token y_t^*, (2) replace y_t^* with a token uniformly selected from a vocabulary of all unique tokens at random, or (3) swap y_t^* and y_{t+1}^*. This is done sequentially from y_1^* until y_T^*.

3.3.3 Learning

Cost function Although it is possible to train the proposed non-autoregressive sequence model using either of the cost functions above (J_{LVM} or J_{DAE},) we propose to stochastically mix these two cost functions. We do so by randomly replacing each term $\hat{Y}^{l-1}$ in Eq. (6.5) with $\tilde{Y}$ in Eq. (3.3):

$$J(\theta) = -\sum_{l=0}^{L+1} \left(\alpha_l \sum_{t=1}^{T} \log p_\theta(y_t^* | \hat{Y}^{l-1}, X) \right. \\ \left. + (1 - \alpha_l) \sum_{t=1}^{T} \log p_\theta(y_t^* | \tilde{Y}, X) \right), \tag{3.4}$$

where $\tilde{Y} \sim C(Y | Y^*)$, and α_l is a sample from a Bernoulli distribution with the probability p_{DAE}. p_{DAE} is a hyperparameter. As the first conditional $p(Y^0|X)$ in Eq. (3.1) does not take as input any target Y, we set $\alpha_0 = 1$ always.

Distillation [Gu et al., 2017], in the context of machine translation, and [Oord et al., 2017], in the context of speech generation, have recently discovered that it is important to use knowledge distillation [Hinton et al., 2015, Kim and Rush, 2016] to successfully train a non-autoregressive sequence model. Following [Gu et al., 2017], we also use knowledge distillation by replacing the reference target Y^* of each training example (X, Y^*) with a target Y^{AR} generated from a well-trained autoregressive counterpart. Other than this replacement, the cost function in Eq (3.4) and the model architecture remain unchanged.

Target Length Prediction One difference between the autoregressive and non-autoregressive models is that the former naturally models the length of a target sequence without any arbitrary upper-bound, while the latter does not. It is hence necessary to separately model $p(T|X)$, where T is the length of a target sequence, although during training, we simply use the length of each reference target sequence.

3.3.4 Inference: Decoding

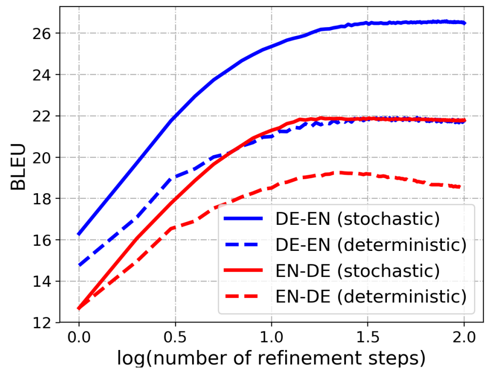
Inference in the proposed approach is entirely deterministic. We start from the input X and first predict the length of the target sequence $\hat{T} = \arg\max_T \log p(T|X)$. Then, given X and $\hat{T}$ we generate the initial target sequence by $\hat{y}_t^0 = \arg\max_{y_t} \log p(y_t^0|X)$, for $t = 1, \ldots, T$ We continue refining the target sequence by $\hat{y}_t^l = \arg\max_{y_t} \log p(y_t^l|\hat{Y}^{l-1}, X)$, for $t = 1, \ldots, T$.

Because these conditionals, except for the initial one, are modeled by a single, shared neural network, this refinement can be performed as many iterations as necessary until a predefined stopping criterion is met. A criterion can be based either on the amount of change in a target sequence after each iteration (i.e., $D(\hat{Y}^{l-1}, \hat{Y}^l) \leq \epsilon$), or on the amount of change in the conditional log-probabilities (i.e., $|\log p(\hat{Y}^{l-1}|X) - \log p(\hat{Y}^{l-1}|X)| \leq \epsilon$) or on the computational budget. In our experiments, we use the first criterion and use Jaccard distance as our distance function D.

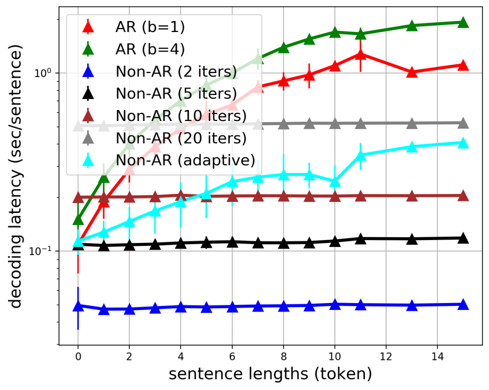
3.4 Related Work

Non-Autoregressive Neural Machine Translation [Schwenk, 2012] proposed a continuous-space translation model to estimate the conditional distribution over a target phrase given a source phrase, while dropping the conditional dependencies among target tokens. The evaluation was however limited to reranking and to short phrase pairs (up to 7 words on each side) only. [Kaiser and Bengio, 2016] investigated neural GPU [Kaiser and Sutskever, 2015], for machine translation. They evaluated both non-autoregressive and autoregressive approaches, and found that the non-autoregressive approach significantly lags behind the autoregressive variants. It however differs from our approach that each iteration does not output a refined version from the previous iteration. The recent paper by [Gu et al., 2017] is most relevant to the proposed work. They similarly introduced a sequence of discrete latent variables. They however use supervised learning for inference, using the word alignment tool [Dyer et al., 2013]. To achieve the best result, [Gu et al., 2017] stochastically sample the latent variables and rerank the corresponding target sequences with an external, autoregressive model. This is in contrast to the proposed approach which is fully deterministic during decoding and does not rely on any extra reranking mechanism.

Parallel WaveNet Simultaneously with [Gu et al., 2017], [Oord et al., 2017] presented a non-autoregressive sequence model for speech generation. They use inverse autoregressive flow IAF [Kingma et al., 2016] to map a sequence of independent random variables to a target sequence. They apply the IAF multiple times, similarly to our iterative refinement strategy. Their approach is however restricted to continuous target variables, while the proposed approach in principle could be applied to both discrete and continuous variables.



(a)



(b)

Figure 3.1: (a) BLEU scores on WMT'14 En-De w.r.t. the number of refinement steps (up to 10^2). The x-axis is in the logarithmic scale. (b) the decoding latencies (sec/sentence) of different approaches on IWSLT'16 En$\rightarrow$De. The y-axis is in the logarithmic scale.

Post-Editing for Machine Translation [Novak et al., 2016] proposed a convolutional neural network that iteratively predicts and applies token substitutions given a translation from a phase-based translation system. Unlike their system, our approach can edit an intermediate translation with a higher degree of freedom. QuickEdit [Grangier and Auli, 2017] and deliberation network [Xia et al., 2017] incorporate the idea of refinement into neural machine translation. Both systems consist of two autoregressive decoders. The second decoder takes into account the translation generated by the first decoder. We extend these earlier efforts by incorporating more than one refinement steps without necessitating extra annotations.

Infusion Training [Bordes et al., 2017] proposed an unconditional generative model for images based on iterative refinement. At each step l of iterative refinement, the model is trained to maximize the log-likelihood of target Y given the weighted mixture of generated samples from the previous iteration $\hat{Y}^{l-1}$ and a corrupted target $\tilde{Y}$. That is, the corrupted version of target is "infused" into generated samples during training.

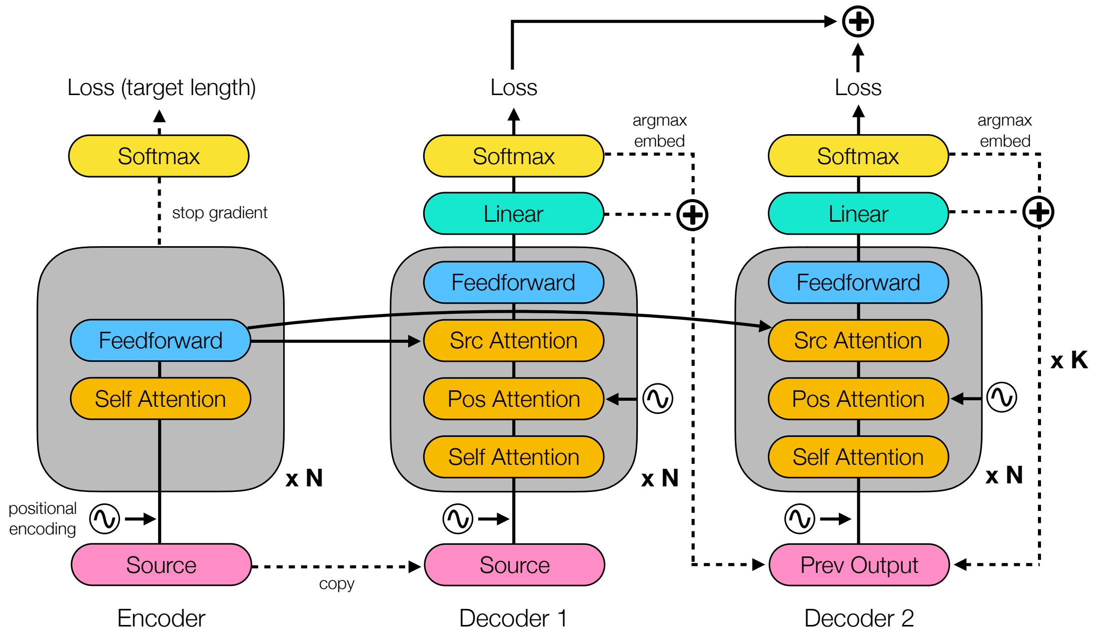


Figure 3.2: We compose three transformer blocks ("Encoder", "Decoder 1" and "Decoder 2") to implement the proposed non-autoregressive sequence model.

In the domain of text, however, computing a weighted mixture of two sequences of discrete tokens is not well defined, and we propose to stochastically mix denoising and lowerbound maximization objectives.

3.5 Network Architecture

We use three transformer-based network blocks to implement our model. The first block ("Encoder") encodes the input X, the second block ("Decoder 1") models the first conditional $\log p(Y^0|X)$, and the final block ("Decoder 2") is shared across iterative refinement steps, modeling $\log p(Y^l|\hat{Y}^{l-1}, X)$. These blocks are depicted side-by-side in Fig. 3.2. The encoder is identical to that from the original Transformer [Vaswani et al., 2017]. We however use the decoders from [Gu et al., 2017] with additional positional attention and use the highway layer [Srivastava et al., 2015] instead of the residual

		IWSLT'16 En-De				WMT'14 En-De			
		En→	De→	GPU	CPU	En→	De→	GPU	CPU
AR	$b = 1$	28.64	34.11	70.3	32.2	23.77	28.15	54.0	15.8
	$b = 4$	28.98	34.81	63.8	14.6	24.57	28.47	44.9	7.0
NAT	FT	26.52	–	–	–	17.69	21.47	–	–
	FT+NPD	28.16	–	–	–	19.17	23.30	–	–
Our Model	$i_{\text{dec}} = 1$	22.20	27.68	573.0	213.2	13.91	16.77	511.4	83.3
	$i_{\text{dec}} = 2$	24.82	30.23	423.8	110.9	16.95	20.39	393.6	49.6
	$i_{\text{dec}} = 5$	26.58	31.85	189.7	52.8	20.26	23.86	139.7	23.1
	$i_{\text{dec}} = 10$	27.11	32.31	98.8	24.1	21.61	25.48	90.4	12.3
	Adaptive	27.01	32.43	125.9	29.3	21.54	25.43	107.2	20.3

Table 3.1: Generation quality (BLEU↑) and decoding efficiency (tokens/sec↑) on IWSLT'16 En-De and WMT'14 En-De. Decoding efficiency is measured sentence-by-sentence. AR: autoregressive models. b: beam width. i_{dec}: the number of refinement steps taken during decoding. Adaptive: the adaptive number of refinement steps. NAT: non-autoregressive transformer models [Gu et al., 2017]. FT: fertility. NPD reranking using 100 samples.

layer [He et al., 2016].

The original input X is padded or shortned to fit the length of the reference target sequence before being fed to Decoder 1. At each refinement step l, Decoder 2 takes as input the predicted target sequence $\hat{Y}^{l-1}$ and the sequence of final activation vectors from the previous step.

3.6 Experimental Setting

We evaluate the proposed approach on two sequence modeling tasks: machine translation and image caption generation. We compare the proposed non-autoregressive model against the autoregressive counterpart both in terms of generation quality, measured in terms of BLEU [Papineni et al., 2002], and generation efficiency, measured in terms of (source) tokens and images per second for translation and image captioning, respectively.

		WMT'16 En-Ro				MS COCO		
		En→	Ro→	GPU	CPU	BLEU	GPU	CPU
AR	$b = 1$	31.93	31.55	55.6	15.7	23.47	4.3	2.1
	$b = 4$	32.40	32.06	43.3	7.3	24.78	3.6	1.0
NAT	FT	27.29	29.06	–	–	–	–	–
	FT+NPD	29.79	31.44	–	–	–	–	–
Our Model	$i_{\text{dec}} = 1$	24.45	25.73	694.2	98.6	20.12	17.1	8.9
	$i_{\text{dec}} = 2$	27.10	28.15	332.7	62.8	20.88	12.0	5.7
	$i_{\text{dec}} = 5$	28.86	29.72	194.4	29.0	21.12	6.2	2.8
	$i_{\text{dec}} = 10$	29.32	30.19	93.1	14.8	21.24	2.0	1.2
	Adaptive	29.66	30.30	118.3	16.5	21.12	10.8	4.8

Table 3.2: Generation quality (BLEU↑) and decoding efficiency (tokens/sec↑) for translation, (images/sec↑) for image captioning on WMT'16 En-Ro and MSCOCO datasets. Decoding efficiency is measured sentence-by-sentence. AR: autoregressive models. b: beam width. i_{dec}: the number of refinement steps taken during decoding. Adaptive: the adaptive number of refinement steps. NAT: non-autoregressive transformer models [Gu et al., 2017]. FT: fertility. NPD reranking using 100 samples.

Machine Translation We choose three tasks of different sizes: IWSLT'16 En↔De (196k pairs), WMT'16 En↔Ro (610k pairs) and WMT'14 En↔De (4.5M pairs). We tokenize each sentence using a script from Moses [Koehn et al., 2007] and segment each word into subword units using BPE [Sennrich et al., 2016]. We use 40k tokens from both source and target for all the tasks. For WMT'14 En-De, we use newstest-2013 and newstest-2014 as development and test sets. For WMT'16 En-Ro, we use newsdev-2016 and newstest-2016 as development and test sets. For IWSLT'16 En-De, we use test2013 for validation.

We closely follow the setting by [Gu et al., 2017]. In the case of IWSLT'16 En-De, we use the small model ($d_{\text{model}} = 278, d_{\text{hidden}} = 507, p_{\text{dropout}} = 0.1, n_{\text{layer}} = 5$ and $n_{\text{head}} = 2$).[1] For WMT'14 En-De and WMT'16 En-Ro, we use the base transformer by [Vaswani et al., 2017] ($d_{\text{model}} = 512, d_{\text{hidden}} = 512, p_{\text{dropout}} = 0.1, n_{\text{layer}} = 6$ and

[1]Due to the space constraint, we refer readers to [Vaswani et al., 2017,Gu et al., 2017] for more details.

$n_{\text{head}} = 8$). We use the warm-up learning rate scheduling [Vaswani et al., 2017] for the WMT tasks, while using linear annealing (from 3×10^{-4} to 10^{-5}) for the IWSLT task. We do not use label smoothing nor average multiple check-pointed models. These decisions were made based on the preliminary experiments. We train each model either on a single P40 (WMT'14 En-De and WMT'16 En-Ro) or on a single P100 (IWSLT'16 En-De) with each minibatch consisting of approximately 2k tokens. We use four P100's to train non-autoregressive models on WMT'14 En-De.

Image Caption Generation: MS COCO We use MS COCO [Lin et al., 2014]. We use the publicly available splits [Karpathy and Li, 2015], consisting of 113,287 training images, 5k validation images and 5k test images. We extract 49 512-dimensional feature vectors for each image, using a ResNet-18 [He et al., 2016] pretrained on ImageNet [Deng et al., 2009b]. The average of these vectors is copied as many times to match the length of the target sentence (reference during training and predicted during evaluation) to form the initial input to Decoder 1. We use the base transformer [Vaswani et al., 2017] except that n_{layer} is set to 4. We train each model on a single 1080ti with each minibatch consisting of approximately 1,024 tokens.

Target Length Prediction We formulate the target length prediction as classification, predicting the difference between the target and source lengths for translation and the target length for image captioning. All the hidden vectors from the n_{layer} layers of the encoder are summed and fed to a softmax classifier after affine transformation. We however do not tune the encoder's parameters for target length prediction. We use this length predictor only during test time. We find it important to accurately predict the target length for good overall performance.

Training and Inference We use Adam [Kingma and Ba, 2014] and use $L = 3$ in Eq. (3.1) during training ($i_{\text{train}} = 4$ from hereon.) We use $p_{\text{DAE}} = 0.5$. We use the deterministic strategy for IWSLT'16 En-De, WMT'16 En-Ro and MS COCO, while the stochastic strategy is used for WMT'14 En-De. These decisions were made based on the validation set performance. After both the non-autogressive sequence model and target length predictor are trained, we decode by first predicting the target length and then running iterative refinement steps until the outputs of consecutive iterations are the same (or Jaccard distance between consecutive decoded sequences is 1). To assess the effectiveness of this adaptive scheme, we also test a fixed number of steps (i_{dec}). In machine translation, we remove any repetition by collapsing multiple consecutive occurrences of a token.

3.7 Results and Analysis

We make some important observations in Table 3.1 and Table 3.2. First, the generation quality improves across all the tasks as we run more refinement steps i_{dec} even beyond that used in training ($i_{\text{train}} = 4$), which supports our interpretation as a conditional denoising autoencoder in Sec. 3.3.2. To further verify this, we run decoding on WMT'14 (both directions) up to 100 iterations. As shown in Fig. 3.1 (a), the quality improves well beyond the number of refinement steps used during training.

Second, the generation efficiency decreases as more refinements are made. We plot the average seconds per sentence in Fig. 3.1 (b), measured on GPU while sequentially decoding one sentence at a time. As expected, decoding from the autoregressive model linearly slows down as the sentence length grows, while decoding from the non-autoregressive model with a fixed number of iterations has the constant complexity.

However, the generation efficiency of non-autoregressive model decreases as more refinements are made. To make a smooth trade-off between the quality and speed, the adaptive decoding scheme allows us to achieve near-best generation quality with a significantly lower computational overhead. Moreover, the adaptive decoding scheme automatically increases the number of refinement steps as the sentence length increases, suggesting that this scheme captures the amount of information in the input well. The increase in latency is however less severe than that of the autoregressive model.

We also observe that the speedup in decoding is much clearer on GPU than on CPU. This is a consequence of highly parallel computation of the proposed non-autoregressive model, which is better suited to GPUs, showcasing the potential of using the non-autoregressive model with a specialized hardware for parallel computation, such as Google's TPUs [Jouppi et al., 2017]. The results of our model decoded with adaptive decoding scheme are comparable to the results from [Gu et al., 2017], without relying on any external tool. On WMT'14 En-De, the proposed model outperforms the best model from [Gu et al., 2017] by two points.

Lastly, it is encouraging to observe that the proposed non-autoregressive model works well on image caption generation. This result confirms the generality of our approach beyond machine translation, unlike that by [Gu et al., 2017] which was for machine translation or by [Oord et al., 2017] which was for speech synthesis.

Ablation Study We use IWSLT'16 En-De to investigate the impact of different number of refinement steps during training (denoted as i_{train}) as well as probability of using denoising autoencoder objective during training (denoted as p_{DAE}). The results are presented in Table 3.3.

First, we observe that it is beneficial to use multiple iterations of refinement dur-

	i_{train}	p_{DAE}	distill	En→De rep	En→De no rep	De→En rep	De→En no rep
AR		$b=1$		28.64		34.11	
		$b=4$		28.98		34.81	
Our Models	1	0		14.62	18.03	16.70	21.18
	2	0		17.42	21.08	19.84	24.25
	4	0		19.22	22.65	22.15	25.24
	4	1		19.83	22.29	24.00	26.57
	4	0.5		20.91	23.65	24.05	28.18
	4	0.5	$\surd$	26.17	27.11	31.92	32.59

Table 3.3: Ablation study on the dev set of IWSLT'16.

ing training. By using four iterations (one step of decoder 1, followed by three steps of decoder 2), the BLEU score improved by approximately 1.5 points in both directions. We also notice that it is necessary to use the proposed hybrid learning strategy to maximize the improvement from more iterations during training ($i_{\text{train}} = 4$ vs. $i_{\text{train}} = 4, p_{\text{DAE}} = 1.0$ vs. $i_{\text{train}} = 4, p_{\text{DAE}} = 0.5$.) Knowledge distillation was crucial to close the gap between the proposed deterministic non-autoregressive sequence model and its autoregressive counterpart, echoing the observations by [Gu et al., 2017] and [Oord et al., 2017]. Finally, we see that removing repeating consecutive symbols improves the quality of the best trained models ($i_{\text{train}} = 4, p_{\text{DAE}} = 0.5$) by approximately +1 BLEU. This suggests that the proposed iterative refinement is not enough to remove repetitions on its own. Further investigation is necessary to properly tackle this issue, which we leave as a future work.

We then compare the deterministic and stochastic approximation strategies on IWSLT'16 En→De and WMT'14 En→De. According to the results in Table 3.4, the stochastic strategy is crucial with a large corpus (WMT'14), while the deterministic strategy works as well or better with a small corpus (IWSLT'16). Both of the strategies

stochastic	distill	IWSLT'16 (En→)	WMT'14 (En→)
		23.65	7.56
√		22.80	16.56
	√	27.11	18.91
√	√	25.39	21.22

Table 3.4: Deterministic and stochastic approximation

benefit from knowledge distillation, but the gap between the two strategies when the dataset is large is much more apparent without knowledge distillation.

Impact of Length Prediction The quality of length prediction has an impact on the overall translation/captioning performance. When using the reference target length (during inference), we consistently observed approximately 1 BLEU score improvement over reported results in the tables and figures across different datasets in the chapter (see Table 3.5 for more detailed comparison).

We additionally compared our length prediction model with a simple baseline that uses length statistics of the corresponding training dataset (a non-parametric approach). To predict the target length for a source sentence with length L_s, we take the average length of all the target sentences coupled with the sources sentences of length L_s in the training set. Compared to this approach, our length prediction model predicts target length correctly twice as often (16% vs. 8%), and gives higher prediction accuracy within five tokens (83% vs. 69%).

3.7.1 Qualitative Analysis

Machine Translation In Table 3.6, we present three sample translations and their iterative refinement steps from the development set of IWSLT'16 (De→En). As expected,

	IWSLT'16		WMT'16		WMT'14	
	En→	→En	En→	→En	En→	→En
pred	27.01	32.43	29.66	30.30	21.54	25.43
ref	28.15	33.11	30.42	31.26	22.10	26.40

Table 3.5: BLEU scores on each dataset when using reference length (ref) and predicted target length (pred).

the sequence generated from the first iteration is a rough version of translation and is iteratively refined over multiple steps. By inspecting the underlined sub-sequences, we see that each iteration does not monotonically improve the translation, but overall modifies the translation towards the reference sentence. Missing words are added, while unnecessary words are dropped. For instance, see the second example. The second iteration removes the unnecessary "were", and the fourth iteration inserts a new word "mostly". The phrase "at the time" is gradually added one word at a time.

Image Caption Generation Table 3.7 and Table 3.8 shows two examples of image caption generation. We observe that each iteration captures more and more details of the input image. In the first example (left), the bus was described only as a "yellow bus" in the first iteration, but the subsequent iterations refine it into "yellow and black bus". Similarly, "road" is refined into "lot". We notice this behavior in the second example (right) as well. The first iteration does not specify the place in which "a woman" is "standing on", which is fixed immediately in the second iteration: "standing on a tennis court". In the final and fourth iteration, the proposed model captures the fact that the "woman" is "holding" a racquet.

3.8 Conclusion

Following on the exciting, recent success of non-autoregressive neural sequence modeling by [Gu et al., 2017] and [Oord et al., 2017], we proposed a deterministic non-autoregressive neural sequence model based on the idea of iterative refinement. We designed a learning algorithm specialized to the proposed approach by interpreting the entire model as a latent variable model and each refinement step as denoising.

We implemented our approach using the Transformer and evaluated it on two tasks: machine translation and image caption generation. On both tasks, we were able to show that the proposed non-autoregressive model performs closely to the autoregressive counterpart with significant speedup in decoding. Qualitative analysis revealed that the iterative refinement indeed refines a target sequence gradually over multiple steps.

3.9 Since the chapter release

Since the release of the chapter in 2018, many different non-autoregressive sequence generation approaches have been released. Among these, the sequence generation by iterative refinement using the conditional masked language model [Devlin et al., 2019] has superseded our work. [Ghazvininejad et al., 2019] proposed conditional masked language models for non-autoregressive machine translation, reaching 95% of autoregressive performance with as low as 10 iterations. The performance gap between autoregressive and non-autoregressive models by iterative refinement was bridged in the follow-up paper [Ghazvininejad et al., 2020]. [Ghazvininejad et al., 2020] passed the intermediate model's predictions as an input to the next refinement step, which was also used in our work. Apart from machine translation, [Wang and Cho, 2019] and [Lawrence et al., 2019] proposed iterative generation using masked language models for open-ended text generation and conversational modeling respectively. We further discuss various ways of generating sequences using masked language models by formulating the generalized sequence generation framework discussed in Chapter 4 of this thesis.

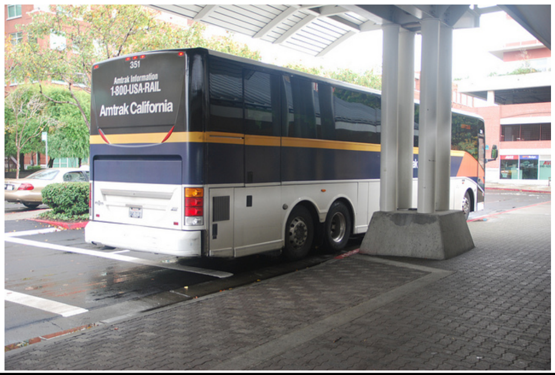
Other notable non-autoregressive sequence generation approaches include continuous latent-variable models [Shu et al., 2019, Ma et al., 2019, Lee et al., 2020b], latent-alignment model [Saharia et al., 2020] based on CTC [Graves et al., 2006] and energy-based models [Tu et al., 2020, Lee et al., 2020a]. We believe that non-autoregressive sequence generation will remain an exciting avenue of research with many more approaches proposed after this thesis.

Src	seitdem habe ich sieben Häuser in der Nachbarschaft mit den Lichtern versorgt und sie funktionierenen wirklich gut .
Iter 1	and I ’ve been seven homes since in neighborhood with the lights and they ’re really functional .
Iter 2	and I ’ve been seven homes in the neighborhood with the lights , and they ’re a really functional .
Iter 4	and I ’ve been seven homes in neighborhood with the lights , and they ’re a really functional .
Iter 8	and I ’ve been providing seven homes in the neighborhood with the lights and they ’re a really functional .
Iter 20	and I ’ve been providing seven homes in the neighborhood with the lights , and they ’re a very good functional .
Ref	since now , I ’ve set up seven homes around my community , and they ’re really working .

Src	er sah sehr glücklich aus , was damals ziemlich ungewöhnlich war , da ihn die Nachrichten meistens deprimierten .
Iter 1	he looked very happy , which was pretty unusual the , because the news was were usually depressing .
Iter 2	he looked very happy , which was pretty unusual at the , because the news was s depressing .
Iter 4	he looked very happy , which was pretty unusual at the , because news was mostly depressing .
Iter 8	he looked very happy , which was pretty unusual at the time because the news was mostly depressing .
Iter 20	he looked very happy , which was pretty unusual at the time , because the news was mostly depressing .
Ref	there was a big smile on his face which was unusual then , because the news mostly depressed him .

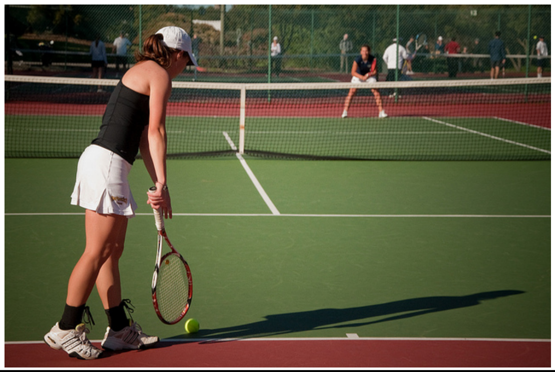
Src	furchtlos zu sein heißt für mich , heute ehrlich zu sein .
Iter 1	to be , for me , to be honest today .
Iter 2	to be fearless , me , is to be honest today .
Iter 4	to be fearless for me , is to be honest today .
Iter 8	to be fearless for me , me to be honest today .
Iter 20	to be fearless for me , is to be honest today .
Ref	so today , for me , being fearless means being honest .

Table 3.6: Three sample De→En translations from the non-autoregressive sequence model. Source sentences are from the dev set of IWSLT’16. The first iteration corresponds to Decoder 1, and from thereon, Decoder 2 is repeatedly applied. Sub-sequences with changes across the refinement steps are underlined.



Generated Caption	
Iter 1	a yellow bus parked on parked in of parking road .
Iter 2	a yellow and black on parked in a parking lot .
Iter 3	a yellow and black bus parked in a parking lot .
Iter 4	a yellow and black bus parked in a parking lot .
Reference Captions	
a tour bus is parked on the curb waiting	
city bus parked on side of hotel in the rain .	
bus parked under an awning next to brick sidewalk	
a bus is parked on the curb in front of a building .	
a double decked bus sits parked under an awning	

Table 3.7: Sample image caption from the proposed non-autoregressive sequence model. The image is from the development set of MS COCO. The first iteration is from decoder 1, while the subsequent ones are from decoder 2. Subsequences with changes across the refinement steps are underlined.



Generated Caption	
Iter 1	a woman standing on playing tennis on a tennis racquet .
Iter 2	a woman standing on a tennis court a tennis racquet .
Iter 3	a woman standing on a tennis court a a racquet .
Iter 4	a woman standing on a tennis court holding a racquet .
Reference Captions	
	a female tennis player in a black top playing tennis
	a woman standing on a tennis court holding a racquet .
	a female tennis player preparing to serve the ball .
	a woman is holding a tennis racket on a court
	a woman getting ready to reach for a tennis ball on the ground

Table 3.8: Sample image caption from the proposed non-autoregressive sequence model. The image is from the development set of MS COCO. The first iteration is from decoder 1, while the subsequent ones are from decoder 2. Subsequences with changes across the refinement steps are underlined.

Chapter 4

A Generalized Framework of Sequence Generation with Application to Undirected Sequence Models

Undirected neural sequence models such as BERT [Devlin et al., 2019] have received renewed interest due to their success on discriminative natural language understanding tasks such as question-answering and natural language inference. The problem of generating sequences directly from these models has received relatively little attention, in part because generating from undirected models departs significantly from conventional monotonic generation in directed sequence models. We investigate this problem by proposing a generalized model of sequence generation that unifies decoding in directed and undirected models. The proposed framework models the process of generation rather than the resulting sequence, and under this framework, we derive various neural sequence models as special cases, such as autoregressive, semi-autoregressive, and refinement-based non-autoregressive models. This unification enables us to adapt decoding algorithms originally developed for directed sequence models to undirected sequence models. We demonstrate this by evaluating various handcrafted and learned decoding strategies on a BERT-like machine translation model [Lample and Conneau, 2019]. The proposed approach achieves constant-time translation results on par with linear-time translation results from the same undirected sequence model, while both are competitive with the state-of-the-art on WMT'14 English-German translation.

4.1 Introduction

Undirected neural sequence models such as BERT [Devlin et al., 2019] have recently brought significant improvements to a variety of discriminative language modeling tasks such as question-answering and natural language inference. Generating sequences from these models has received relatively little attention. Unlike directed sequence models, each word typically depends on the full left and right context around it in undirected sequence models. Thus, a decoding algorithm for an undirected sequence model must specify both how to select positions and what symbols to place in the selected positions. We formalize this process of selecting positions and replacing symbols as a general framework of sequence generation, and unify decoding from both directed and undirected sequence models under this framework. This framing enables us to study generation on its own, independent from the specific parameterization of the sequence models.

Our proposed framework casts sequence generation as a process of determining the length of the sequence, and then repeatedly alternating between selecting sequence positions followed by generation of symbols for those positions. A variety of sequence models can be derived under this framework by appropriately designing the length distribution, position selection distribution, and symbol replacement distribution. Specifically, we derive popular decoding algorithms such as monotonic autoregressive, non-autoregressive by iterative refinement, and monotonic semi-autoregressive decoding as special cases of the proposed model.

This separation of coordinate selection and symbol replacement allows us to build a diverse set of decoding algorithms agnostic to the parameterization or training procedure of the underlying model. We thus fix the symbol replacement distribution as a variant of BERT and focus on deriving novel generation procedures for undirected neural sequence models under the proposed framework. We design a coordinate selection distribution using a log-linear model and a learned model with a reinforcement learning objective to demonstrate that our model generalizes various fixed-order generation strategies, while also being capable of adapting generation order based on the content of intermediate sequences.

We empirically validate our proposal on machine translation using a translation-variant of BERT called a masked translation model [Lample and Conneau, 2019]. We design several generation strategies based on features of intermediate sequence distributions and compare them against the state-of-the-art monotonic autoregressive sequence model [Vaswani et al., 2017] on WMT'14 English-German. Our experiments show that generation from undirected sequence models, under our framework, is competitive with the state of the art, and that adaptive-order generation strategies generate sequences in different ways, including left-to-right, right-to-left and mixtures of these.

Due to the flexibility in specifying a coordinate selection mechanism, we design constant-time variants of the proposed generation strategies, closely following the experimental setup of [Ghazvininejad et al., 2019]. Our experiments reveal that we can do constant-time translation with the budget as low as 20 iterations (equivalently, generating a sentence of length 20 in the conventional approach) while achieving similar performance to the state-of-the-art-monotonic autoregressive sequence model and linear-time translation from the same masked translation model. This again confirms the potential of the proposed framework and generation strategies.

4.2 A Generalized Framework of Sequence Generation

We propose a generalized framework of probabilistic sequence generation to unify generation from directed and undirected neural sequence models. In this generalized framework, we have a *generation sequence* G of pairs of an *intermediate sequence* $Y^t = (y_1^t, \ldots, y_L^t)$ and the corresponding *coordinate sequence* $Z^t = (z_1^t, \ldots, z_L^t)$, where V is a vocabulary, L is a length of a sequence, T is a number of generation steps, $y_i^t \in V$, and $z_i^t \in \{0, 1\}$. The coordinate sequence indicates which of the current intermediate sequence are to be replaced. That is, consecutive pairs are related to each other by $y_i^{t+1} = (1 - z_i^{t+1})y_i^t + z_i^{t+1}\tilde{y}_i^{t+1}$, where $\tilde{y}_i^{t+1} \in V$ is a new symbol for the position i. This sequence of pairs G describes a procedure that starts from an empty sequence $Y^1 = (\langle\text{mask}\rangle, \ldots, \langle\text{mask}\rangle)$ and empty coordinate sequence $Z^1 = (0, ..., 0)$, iteratively fills in tokens, and terminates after T steps with final sequence Y^T. We model this procedure probabilistically as $p(G|X)$:

$$\underbrace{p(L|X)}_{\text{(c) length predict}} \prod_{t=1}^{T} \prod_{i=1}^{L} \underbrace{p(z_i^{t+1}|Y^{\leq t}, Z^t, X)}_{\text{(a) coordinate selection}} \underbrace{p(y_i^{t+1}|Y^{\leq t}, X)}_{\text{(b) symbol replacement}}{}^{z_i^{t+1}} \tag{4.1}$$

We condition the whole process on an input variable X to indicate that the proposed model is applicable to both conditional and unconditional sequence generation. In the latter case, $X = \emptyset$.

We first predict the length L of a target sequence Y according to $p(L|X)$ distribution to which we refer as (c) length prediction. At each generation step t, we first select the next coordinates Z^{t+1} for which the corresponding symbols will be replaced according to $p(z_i^{t+1}|Y^{\leq t}, Z^t, X)$, to which we refer as (a) coordinate selection. Once the coordinate sequence is determined, we replace the corresponding symbols according to the distribution $p(y_i^{t+1}|Y^{\leq t}, Z^{t+1}, X)$, leading to the next intermediate sequence Y^{t+1}. From this sequence generation framework, we recover the sequence distribution $p(Y|X)$ by marginalizing out all the intermediate and coordinate sequences except for the final sequence Y^T. In the remainder of this section, we describe several special cases of the proposed framework, which are monotonic autoregressive, non-autoregressive, semi-autoregressive neural sequence models.

4.2.1 Special Cases

Monotonic autoregressive neural sequence models We first consider one extreme case of the generalized sequence generation model, where we replace one symbol at a time, monotonically moving from the left-most position to the right-most. In this case, we define the coordinate selection distribution of the generalized sequence generation model in Eq. (4.1) (a) as $p(z_{i+1}^{t+1} = 1|Y^{\leq t}, Z^t, X) = \mathbb{1}(z_i^t = 1)$, where $\mathbb{1}(\cdot)$ is an indicator function and $z_1^1 = 1$. This coordinate selection distribution is equivalent to saying that we replace one symbol at a time, shifting from the left-most symbol to the right-most symbol, regardless of the content of intermediate sequences. We then choose the symbol replacement distribution in Eq. (4.1) (b) to be $p(y_{i+1}^{t+1}|Y^{\leq t}, X) = p(y_{i+1}^{t+1}|y_1^t, y_2^t, \ldots, y_i^t, X)$, for $z_{i+1}^{t+1} = 1$. Intuitively, we limit the dependency of y_{i+1}^{t+1} only to the symbols to its left in the previous intermediate sequence

$y^t_{<(i+1)}$ and the input variable X. The length distribution (4.1) (c) is implicitly defined by considering how often the special token $\langle\text{eos}\rangle$, which indicates the end of a sequence, appears after L generation steps: $p(L|X) \propto \sum_{y_{1:L-1}} \prod_{l=1}^{L-1} p(y^{l+1}_{l+1} = \langle\text{eos}\rangle\, | y^{\leq l}_{\leq l}, X)$. With these choices, the proposed generalized model reduces to $p(G|X) = \prod_{i=1}^{L} p(y_i|y_{<i}, X)$ which is a widely-used monotonic autoregressive neural sequence model.

Non-autoregressive neural sequence modeling by iterative refinement We next consider the other extreme in which we replace the symbols in all positions at every single generation step, as shown in Chapter 3. We design the coordinate selection distribution to be implying that we replace the symbols in all the positions. We then choose the symbol replacement distribution to be as it was in Eq. (4.1) (b). That is, the distribution over the symbols in the position i in a new intermediate sequence y_i^{t+1} is conditioned on the entire current sequence Y^t and the input variable X. We do not need to assume any relationship between the number of generation steps T and the length of a sequence L in this case. The length prediction distribution $p(L|X)$ is estimated from training data.

Semi-autoregressive neural sequence models [Wang et al., 2018b] recently proposed a compromise between autoregressive and non-autoregressive sequence models by predicting a chunk of symbols in parallel at a time. This approach can also be put under the proposed generalized model. We first extend the coordinate selection distribution of the autoregressive sequence model into

$$p(z^{t+1}_{k(i+1)+j} = 1 | Y^{\leq t}, Z^t, X) =$$
$$= \begin{cases} 1, & \text{if } z^t_{ki+j} = 1, \forall j \in \{0, 1, \ldots, k\} \\ 0, & \text{otherwise,} \end{cases}$$

where k is a *group size*. Similarly we modify the symbol replacement distribution:

$$\begin{aligned} p(y^{t+1}_{k(i+1)+j}|Y^{\leq t}, X) =& p(y^{t+1}_{k(i+1)+j}|y^{t}_{<k(i+1)}, X), \\ & \forall j \in \{0, 1, \ldots, k\}, \end{aligned}$$

for $z^t_i = 1$. This naturally implies that $T = \lceil L/k \rceil$.

Non-monotonic neural sequence models The proposed generalized framework subsumes recently proposed variants of non-monotonic generation [Welleck et al., 2019, Stern et al., 2019, Gu et al., 2019a]. Unlike the other special cases described above, these non-monotonic generation approaches learn not only the symbol replacement distribution but also the coordinate selection distribution, and implicitly the length distribution, from data. Because the length of a sequence is often not decided in advance, the intermediate coordinate sequence Z^t and the coordinate selection distribution are reparameterized to work with relative coordinates rather than absolute coordinates. We do not go into details of these recent algorithms, but we emphasize that all these approaches are special cases of the proposed framework, which further suggests other variants of non-monotonic generation.

4.3 Decoding from Masked Language Models

In this section, we give an overview of masked language models like BERT, cast Gibbs sampling under the proposed framework, and then use this connection to design a set of approximate, deterministic decoding algorithms for undirected sequence models.

4.3.1 BERT as an undirected sequence model

BERT [Devlin et al., 2019] is a masked language model: It is trained to predict a word given the word's left and right context. Because the model gets the full context, there are no directed dependencies among words, so the model is undirected. The word to be predicted is masked with a special $\langle \text{mask} \rangle$ symbol and the model is trained to predict $p(y_i|y_{<i}, \langle \text{mask} \rangle, y_{>i}, X)$. We refer to this as the *conditional BERT distribution*. This objective was interpreted as a stochastic approximation to the pseudo log-likelihood objective [Besag, 1977] by [Wang and Cho, 2019]. This approach of full-context generation with pseudo log-likelihood maximization for recurrent networks was introduced earlier by [Berglund et al., 2015]. More recently, [Sun et al., 2017] use it for image caption generation.

Recent work [Wang and Cho, 2019, Ghazvininejad et al., 2019] has demonstrated that undirected neural sequence models like BERT can learn complex sequence distributions and generate well-formed sequences. In such models, it is relatively straightforward to collect unbiased samples using, for instance, Gibbs sampling. But due to high variance of Gibbs sampling, the generated sequence is not guaranteed to be high-quality relative to a ground-truth sequence. Finding a good sequence in a deterministic manner is also nontrivial.

A number of papers have explored using pretrained language models like BERT to initialize sequence generation models. [Ramachandran et al., 2017], [Song et al., 2019], and [Lample and Conneau, 2019] use a pretrained undirected language model to initialize a conventional monotonic autoregressive sequence model, while [Edunov et al., 2019] use a BERT-like model to initialize the lower layers of a generator, without finetuning. Our work differs from these in that we attempt to directly generate from the pretrained model, rather than using it as a starting point to learn another model.

4.3.2 Gibbs sampling in the generalized sequence generation model

Gibbs sampling: uniform coordinate selection To cast Gibbs sampling into our framework, we first assume that the length prediction distribution $P(L|X)$ is estimated from training data, as is the case in the non-autoregressive neural sequence model. In Gibbs sampling, we often uniformly select a new coordinate at random, which corresponds to $p(z_i^{t+1} = 1|Y^{\leq t}, Z^t, X) = 1/L$ with the constraint that $\sum_{i=1}^{L} z_i^t = 1$. By using the conditional BERT distribution as a symbol replacement distribution, we end up with Gibbs sampling.

Adaptive Gibbs sampling: non-uniform coordinate selection Instead of selecting coordinates uniformly at random, we can base selections on the intermediate sequences. We propose a log-linear model with features ϕ_i based on the intermediate and coordinate sequences:

$$p(z_i^{t+1} = 1|Y^{\leq t}, Z^t, X) \propto \exp\left\{\frac{1}{\tau}\sum_{i=1}^{L} \alpha_i \phi_i(Y^t, Z^t, X, i)\right\} \tag{4.2}$$

again with the constraint that $\sum_{i=1}^{L} z_i^t = 1$. $\tau > 0$ is a temperature parameter controlling the sharpness of the coordinate selection distribution. A moderately high τ smooths the coordinate selection distribution and ensures that all the coordinates are replaced in the infinite limit of T, making it a valid Gibbs sampler [Levine and Casella, 2006].

We investigate three features ϕ_i: (1) We compute how peaked the conditional distribution of each position is given the symbols in all the other positions by measuring its *negative entropy*: $\phi_{\text{negent}}(Y^t, Z^t, X, i) = -\mathcal{H}(y_i^{t+1}|y_{<i}^t, \langle\text{mask}\rangle, y_{>i}^t, X)$. In other words, we prefer a position i if we know the change in i has a high potential to alter the joint probability $p(Y|X) = p(y_1, y_2, ..., y_L|X)$. (2) For each position i we measure how unlikely the *current* symbol (y_i^t, not y_i^{t+1}) is under the *new* conditional distribution: $\phi_{\text{logp}}(Y^t, Z^t, X, i) = -\log p(y_i = y_i^t | y_{<i}^t, \langle\text{mask}\rangle, y_{>i}^t, X)$. Intuitively, we prefer to replace a symbol if it is highly incompatible with the input variable and all the other symbols in the current sequence. (3) We encode a *positional preference* that does not consider the content of intermediate sequences: $\phi_{\text{pos}}(i) = -\log(|t - i| + \epsilon)$, where $\epsilon > 0$ is a small constant scalar to prevent $\log 0$. This feature encodes our preference to generate from left to right if there is no information about the input variable nor of any intermediate sequences.

Unlike the special cases of the proposed generalized model in §4.2, the coordinate at each generation step is selected based on the intermediate sequences, previous coordinate sequences, and the input variable. We mix the features using scalar coefficients α_{negent}, α_{logp} and α_{pos}, which are selected or estimated to maximize a target quality measure on the validation set.

Adaptive Gibbs sampling: learned coordinate selection We learn a coordinate selection distribution that selects coordinates in order to maximize a reward function that we specify. In this case, we refer to the coordinate selection distribution as a *policy*, $\pi_\theta(a_t|s_t)$, where a state s_t is $(Y^{\leq t}, Z^t, X)$, an action $a_t \in \{1, \ldots, L\}$ is a coordinate, so that Z^{t+1} is 1 at position a_t and 0 elsewhere, and π_θ is parameterized using a neural network. Beginning at a state $s_1 \sim p(s_1)$ corresponding to an input X along with an empty coordinate and output sequence, we obtain a generation by repeatedly sampling a coordinate $a_t \sim \pi_\theta(\cdot|s_t)$ and transitioning to a new state for T steps. Each transition, $s_{t+1} \sim p(\cdot|s_t, a_t)$, consists of generating a symbol at position a_t. Given a scalar reward function $r(s_t, a_t, s_{t+1})$, the objective is to find a policy that maximizes expected reward, with the expectation taken over the distribution of generations obtained using the policy for coordinate selection,

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \right], \tag{4.3}$$

$$\pi_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|a_t, s_t), \tag{4.4}$$

where $\tau = (s_1, a_1, s_2, \ldots, a_T, s_{T+1})$, and $\gamma \in [0, 1]$ is a discount factor (with $0^0 = 1$). We maximize this objective by estimating its gradient using policy gradient methods [Williams, 1992]. We discuss our choice of reward function, policy parameterization, and hyperparameters later in Section 4.4.

4.3.3 Optimistic decoding and beam search from a masked language model

Based on the adaptive Gibbs sampler with the non-uniform and learned coordinate selection distributions we can now design an inference procedure to approximately find the most likely sequence $\arg\max_Y p(Y|X)$ from the sequence distribution by exploiting the corresponding model of sequence generation. In doing so, a naive approach is to marginalize out the generation procedure G using a Monte Carlo method: $\arg\max_{Y^T} \frac{1}{M} \sum_{G^m} p(Y^T|Y^{m,<T}, Z^{m,\leq T}, X)$ where G^m is the m-th sample from the sequence generation model. This approach suffers from a high variance and non-deterministic behavior, and is less appropriate for practical use. We instead propose an optimistic decoding approach following equation (4.1):

$$\underset{\substack{L,Y^1,\ldots,Y^T\\Z^1,\ldots,Z^T}}{\arg\max} \log p(L|X) + \sum_{t=1}^{T}\sum_{i=1}^{L}\Big(\log p(z_i^{t+1}|Y^{\leq t}, Z^t, X) + z_i^{t+1}\log p(y_i^{t+1}|Y^{\leq t}, X)\Big) \tag{4.5}$$

The proposed procedure is *optimistic* in that we consider a sequence generated by following the most likely generation path to be highly likely under the sequence distribution obtained by marginalizing out the generation path. This optimism in the criterion more readily admits a deterministic approximation scheme such as greedy and beam search, although it is as intractable to solve this problem as the original problem which required marginalization of the generation path.

Length-conditioned beam search To solve this intractable optimization problem, we design a heuristic algorithm, called length-conditioned beam search. Intuitively, given a length L, this algorithm performs beam search over the coordinate and intermediate token sequences. At each step t of this iterative algorithm, we start from the hypothesis set $\mathcal{H}^{t-1}$ that contains K generation hypotheses: $\mathcal{H}^{t-1} = \left\{h_k^{t-1} = ((\hat{Y}_k^1, \ldots, \hat{Y}_k^{t-1}), (\hat{Z}_k^1, \ldots, \hat{Z}_k^{t-1}))\right\}_{k=1}^{K}$. Each generation hypothesis has a score:

$$s(h_k^{t-1}) = \log p(L|X) + \sum_{t'=1}^{t-1}\sum_{i=1}^{L}\left(\log p(\hat{z}_i^{t'}|\hat{Y}_k^{<t'}, \hat{Z}^{t'-1}, X) + \hat{z}_i^{t'}\log p(\hat{y}_i^{t'}|\hat{Y}^{\leq t}, X)\right).$$

For notational simplicity, we drop the time superscript t. Each of the K generation hypotheses is first expanded with K' candidate positions according to the coordinate selection distribution:

$$\text{arg top-}K'_{u\in\{1,\dots,L\}} \underbrace{s(h_k) + \log p(z_{k,u} = 1|\hat{Y}^{<t}, \hat{Z}^{t-1}, X)}_{=s(h_k\|\text{one-hot}(u))}$$

so that we have $K \times K'$ candidates $\left\{\hat{h}_{k,k'}\right\}$, where each candidate consists of a hypothesis h_k with the position sequence extended by the selected position $u_{k,k'}$ and has a score $s(h_k\|\text{one-hot}(u_{k,k'}))$.[1] We then expand each candidate with the symbol replacement distribution:

$$\text{arg top-}K''_{v\in V} \underbrace{s(h_k\|\text{one-hot}(u_{k,k'})) + \log p(y_{z_{k,k'}} = v|\hat{Y}^{\leq t}, X)}_{=s(h_{k,k'}\|(\hat{Y}^{t-1}_{<z_{k,k'}}, v, \hat{Y}^{t-1}_{>z_{k,k'}}))}.$$

This results in $K \times K' \times K''$ candidates $\left\{\hat{\hat{h}}_{k,k',k''}\right\}$, each consisting of hypothesis h_k with intermediate and coordinate sequence respectively extended by $v_{k,k',k''}$ and $u_{k,k'}$. Each hypothesis has a score $s(h_{k,k'}\|(\hat{Y}^{t-1}_{<z_{k,k'}}, v_{k,k',k''}, \hat{Y}^{t-1}_{>z_{k,k'}}))$,[2] which we use to select K candidates to form a new hypothesis set $\mathcal{H}^t = \text{arg top-}K_{h\in\left\{\hat{\hat{h}}_{k,k',k''}\right\}_{k,k',k''}} s(h)$.

After iterating for a predefined number T of steps, the algorithm terminates with the final set of K generation hypotheses. We then choose one of them according to a prespecified criterion, such as Eq. (4.5), and return the final symbol sequence $\hat{Y}^T$.

4.4 Experimental Settings

Data and preprocessing We evaluate our framework on WMT'14 English-German translation. The dataset consists of 4.5M parallel sentence pairs. We preprocess this

[1] $h_k\|\text{one-hot}(u_{k,k'})$ appends one-hot$(u_{k,k'})$ at the end of the sequence of the coordinate sequences in h_k

[2] $h_{k,k'}\|(\hat{Y}^{t-1}_{<z_{k,k'}}, v_{k,k',k''}, \hat{Y}^{t-1}_{>z_{k,k'}})$ denotes creating a new sequence from $\hat{Y}^{t-1}$ by replacing the $z_{k,k'}$-th symbol with $v_{k,k',k''}$, and appending this sequence to the intermediate sequences in $h_{k,k'}$.

dataset by tokenizing each sentence using a script from Moses [Koehn et al., 2007] and then segmenting each word into subword units using byte pair encoding [Sennrich et al., 2016] with a joint vocabulary of 60k tokens. We use newstest-2013 and newstest-2014 as validation and test sets respectively.

Sequence models We base our models off those of [Lample and Conneau, 2019]. Specifically, we use a Transformer [Vaswani et al., 2017] with 1024 hidden units, 6 layers, 8 heads, and Gaussian error linear units [Hendrycks and Gimpel, 2016]. We use a pretrained model[3] trained using a masked language modeling objective [Lample and Conneau, 2019] on 5M monolingual sentences from WMT NewsCrawl 2007-2008. To distinguish between English and German sentences, a special language embedding is added as an additional input to the model.

We adapt the pretrained model to translation by finetuning it with a masked translation objective [Lample and Conneau, 2019]. We concatenate parallel English and German sentences, mask out a subset of the tokens in either the English or German sentence, and predict the masked out tokens. We uniformly mask out $0 - 100\%$ tokens as in [Ghazvininejad et al., 2019]. Training this way more closely matches the generation setting, where the model starts with an input sequence of all masks.

Baseline model We compare against a standard Transformer encoder-decoder autoregressive neural sequence model [Vaswani et al., 2017] trained for left-to-right generation and initialized with the same model pretrained using a masked language modeling objective [Lample and Conneau, 2019, Song et al., 2019]. We train a separate autoregressive model to translate an English sentence to a German sentence and vice versa, with the same hyperparameters as our model.

[3]`https://dl.fbaipublicfiles.com/XLM/mlm_ende_1024.pth`

	Gold	# of length candidates			
		1	2	3	4
En→De	22.50	22.22	22.76	23.01	**23.22**
De→En	28.05	26.77	27.32	27.79	**28.15**

Table 4.1: Effect of the number of length candidates considered during decoding on BLEU, measured on the validation set (newstest-2013) using the **easy-first** strategy.

Training details We train the models using Adam [Kingma and Ba, 2014] with an inverse square root learning rate schedule, learning rate of 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.98$, and dropout rate of 0.1 [Srivastava et al., 2014]. Our models are trained on 8 GPUs with a batch size of 256 sentences.

Handcrafted decoding strategies We design four generation strategies for the masked translation model based on the log-linear coordinate selection distribution in §4.2:

1. **Uniform**: $\tau \to \infty$, i.e., sample a position uniformly at random without replacement
2. **Left2Right**: $\alpha_{\text{negent}} = 0$, $\alpha_{\text{logp}} = 0$, $\alpha_{\text{pos}} = 1$
3. **Least2Most** [Ghazvininejad et al., 2019]: $\alpha_{\text{negent}} = 0$, $\alpha_{\text{logp}} = 1$, $\alpha_{\text{pos}} = 0$
4. **Easy-First**: $\alpha_{\text{negent}} = 1$, $\alpha_{\text{logp}} = 1$,[4] $\alpha_{\text{pos}} = 0$

We use beam search described in §4.3.3 with K' fixed to 1, i.e., we consider only one possible position for replacing a symbol per hypothesis each time of generation. We vary $K = K''$ between 1 (greedy) and 4. For each source sentence, we consider four length candidates according to the length distribution estimated from the training pairs, based on early experiments showing that using only four length candidates performs as well as using the ground-truth length (see Table 4.1). Given the four candidate

[4]We set $\alpha_{\text{logp}} = 0.9$ for De→En based on the validation set performance.

translations, we choose the best one according to the pseudo log-probability of the final sequence [Wang and Cho, 2019]. Additionally, we experimented with choosing best translation according to log-probability of the final sequence calculated by an autoregressive neural sequence model.

Learned decoding strategies We train a parameterized coordinate selection policy to maximize expected reward (Eq. 4.3). As the reward function, we use the change in edit distance from the reference,

$$r(s_t, a_t, s_{t+1}) = (d_{\text{edit}}(Y^{\leq t}, Y) - d_{\text{edit}}(Y^{\leq t+1}, Y)),$$

where s_t is $(Y^{\leq t}, Z^t, X)$. The policy is parameterized as,

$$\pi_\theta(a_t|s_t) = \text{softmax}\left(f_\theta(h_1, \bar{h}), \ldots, f_\theta(h_L, \bar{h})\right),$$

where $h_i \in \mathbb{R}^{1024}$ is the masked language model's output vector for position i, and $\bar{h} \in \mathbb{R}^{1024}$ is a history of the previous k selected positions, $\bar{h} = \frac{1}{k}\sum_{j=1}^{k}(\text{emb}_\theta(j) + h^j_{a_j})$. We use a 2-layer MLP for f_θ which concatenates its inputs and has hidden dimension of size 1024.

Policies are trained with linear time decoding ($T = L$), with positions sampled from the current policy, and symbols selected greedily. At each training iteration we sample a batch of generations, add the samples to a FIFO buffer, then perform gradient updates on batches sampled from the buffer. We use proximal policy optimization (PPO), specifically the clipped surrogate objective from [Schulman et al., 2017] with a learned value function $V_\theta(s_t)$ to compute advantages. This objective resulted in stable training compared to initial experiments with REINFORCE [Williams, 1992]. The value function is a 1-layer MLP, $V_\theta(\frac{1}{L}\sum_{i=1}^{L}(h_i, \bar{h}))$.

Training hyperparameters were selected based on validation BLEU in an initial grid search of generation batch size $\in \{4, 16\}$ (sequences), FIFO buffer size $\in \{1k, 10k\}$ (timesteps), and update batch size $\in \{32, 128\}$ (timesteps). Our final model was then selected based on validation BLEU with a grid search on discount $\gamma \in \{0.1, 0.9, 0.99\}$ and history $k \in \{0, 20, 50\}$ for each language pair, resulting in a discount γ of 0.9 for both pairs, and history sizes of 0 for De$\rightarrow$En and 50 for En$\rightarrow$De.

Decoding scenarios We consider two decoding scenarios: linear-time and constant-time decoding. In the linear-time scenario, the number of decoding iterations T grows linearly w.r.t. the length of a target sequence L. We test setting T to L and $2L$. In the constant-time scenario, the number of iterations is constant w.r.t. the length of a translation, i.e., $T = O(1)$. At the t-th iteration of generation, we replace o_t-many symbols, where o_t is either a constant $\lceil L/T \rceil$ or linearly anneals from L to 1 ($L \rightarrow 1$) as done by [Ghazvininejad et al., 2019].

4.5 Linear-Time Decoding: Result and Analysis

Main findings We present translation quality measured by BLEU [Papineni et al., 2002] in Table 4.2. We identify a number of important trends. (1) The deterministic coordinate selection strategies (**left2right**, **least2most**, **easy-first** and **learned**) significantly outperform selecting coordinates uniformly at random, by up to 3 BLEU in both directions. Deterministic coordinate selection strategies produce generations that not only have higher BLEU compared to uniform coordinate selection, but are also more likely according to the model. We do so by computing the energy (negative logit) of the sequence of intermediate sentences generated while using an algorithm, and com-

	b	T	Baseline Autoregressive	Decoding from an undirected sequence model Uniform	Left2Right	Least2Most	Easy-First	Learned
En→De	1	L	25.33	21.01	24.27	23.08	23.73	24.10
	4	L	26.84	22.16	25.15	23.81	24.13	24.87
	4	L*	–	22.74	**25.66**	24.42	24.69	**25.28**
	1	$2L$	–	21.16	24.45	23.32	23.87	24.15
	4	$2L$	–	21.99	25.14	23.81	24.14	24.86
De→En	1	L	29.83	26.01	28.34	28.85	29.00	28.47
	4	L	30.92	27.07	29.52	29.03	29.41	29.73
	4	L*	–	28.07	**30.46**	29.84	30.32	**30.58**
	1	$2L$	–	26.24	28.64	28.60	29.12	28.45
	4	$2L$	–	26.98	29.50	29.02	29.41	29.71

Table 4.2: Results (BLEU↑) on WMT'14 En↔De translation using various decoding algorithms and different settings of beam search width (b) and number of iterations (T) as a function of sentence length (L). For each sentence we use 4 most likely sentence lengths. * denotes rescoring generated hypotheses using autoregressive model instead of proposed model.

paring to the average energy of intermediate sentences generated by picking positions uniformly at random. We plot this energy difference over decoding in Figure 4.2. We additionally plot the evolution of energy of the sequence by different position selection algorithms throughout generation process in Figure 4.3. Overall, we find that left-to-right, least-to-most, and easy-first do find sentences that are lower energy than the uniform baseline over the entire decoding process. Easy-first produces sentences with the lowest energy, followed by least-to-most, and then left-to-right. The success of these relatively simple handcrafted and learned coordinate selection strategies suggest avenues for further improvement for generation from undirected sequence models. (2) The proposed beam search algorithm for undirected sequence models provides an improvement of about 1 BLEU over greedy search, confirming the utility of the proposed framework as a way to move decoding techniques across different paradigms of sequence modeling. (3) Rescoring generated translations with an autoregressive model adds about 1 BLEU across all coordinate selection strategies. Rescoring adds minimal

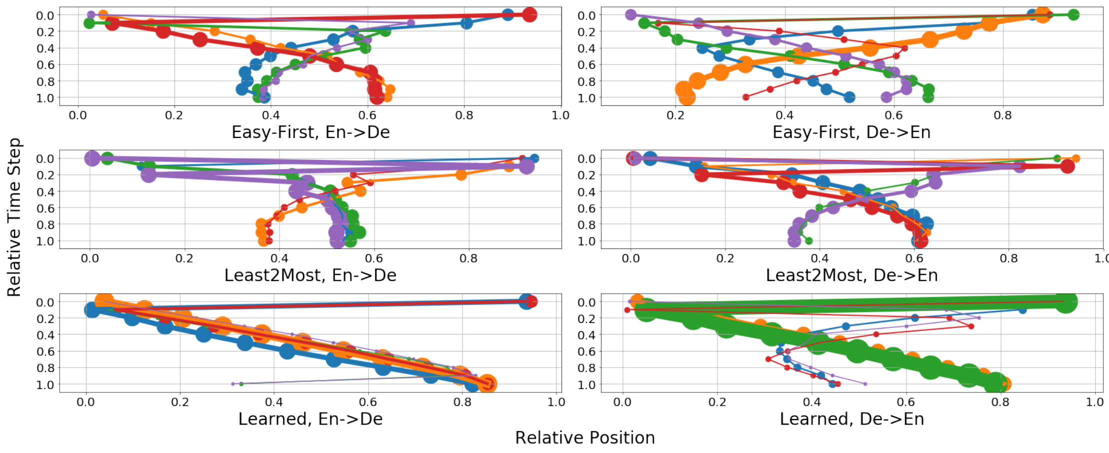


Figure 4.1: Generation orders given by **easy-first**, **least2most**, and **learned** coordinate selection. We use greedy search with L iterations on the development set. We group the orders into five clusters using and visualize cluster centers with normalized positions (x-axis) over normalized generation steps (y-axis). The thickness of a line is proportional to the number of examples in the corresponding cluster.

overhead as it is run in parallel since the left-to-right constraint is enforced by masking out future tokens. (4) Different generation strategies result in translations of varying qualities depending on the setting. **Learned** and **left2right** were consistently the best performing among all generation strategies. On English-German translation, **left2right** is the best performing strategy slightly outperforming the **learned** strategy, achieving 25.66 BLEU. On German-English translation, **learned** is the best performing strategy, slightly outperforming the **left2right** strategy while achieving 30.58 BLEU. (5) We see little improvement in refining a sequence beyond the first pass. (6) Lastly, the masked translation model is competitive with the state of the art neural autoregressive model, with a difference of less than 1 BLEU score in performance. We hypothesize that a difference between train and test settings causes a slight performance difference of the masked translation model compared to the conventional autoregressive model. In the standard autoregressive case, the model is explicitly trained to generate in left-to-right order, which matches the test time usage. By randomly selecting tokens to mask during training, our undirected sequence model is trained to follow all possible generation orders and to use context from both directions, which is not available when generating left-to-right at test time.

Adaptive generation order The **least2most**, **easy-first**, and **learned** generation strategies automatically adapt the generation order based on the intermediate sequences generated. We investigate the resulting generation orders on the development set by presenting each as a 10-dim vector (downsampling as necessary), where each element corresponds to the selected position in the target sequence normalized by sequence length. We cluster these sequences with k-means clustering and visualize the clusters centers as curves with thickness proportional to the number of sequences in the cluster in Fig. 4.1.

The visualization reveals that many sequences are generated monotonically, either left-to-right or right-to-left (see, e.g., green, purple and orange clusters in **easy-first**, De→En, and orange, blue, and red clusters in **learned**, En→De). For the **easy-first** and **least2most** strategies, we additionally identify clusters of sequences that are generated from outside in (e.g., blue and red clusters in **easy-first**, De→En, and red and purple clusters in **least2most**, En→De).

On De→En, in roughly 75% of the generations, the **learned** policy either generated from left-to-right (orange) or generated the final token, typically punctuation, followed by left-to-right generation (green). In the remaining 25% of generations, the **learned** policy generates with variations of an outside-in strategy (red, blue, purple). On En→De, the **learned** policy has a higher rate of left-to-right generation, with roughly 85% of generations using a left-to-right variation (blue, orange). These variations are however typically not strictly monotonic; the learned policy usually starts with the final token, and often skips tokens in the left-to-right order before generating them at a later time. We hypothesize that the learned policy tends towards variations of left-to-right since (a) left-to-right may be an easy strategy to learn, yet (b) left-to-right achieves reasonable performance.

We present sample decoding processes on De→En with $b = 1, T = L$ using the **easy-first** decoding algorithm in Figures 4.4, 4.5, 4.6, and 4.7, and the **learned** decoding algorithm in Figures 4.8, 4.9, and 4.10.

For easy-first decoding, we highlight examples decoding in right-to-left-to-right-to-left order, outside-in, left-to-right, and right-to-left orders, which respectively correspond to the orange, purple, red, and blue clusters from Figure 4.1.

For learned decoding, we highlight examples with right-to-left-to-right, outside-in, and left-to-right orders, corresponding to the blue, red, and green clusters. The examples demonstrate the ability of the coordinate selection strategies to adapt the generation order based on the intermediate sequences generated. Even in the cases of largely monotonic generation order (left-to-right and right-to-left), each algorithm has the capacity to make small changes to the generation order as needed.

In general, we explain the tendency towards either monotonic or outside-in generation by the availability of contextual evidence, or lack thereof. At the beginning of generation, the only two non-mask symbols are the beginning and end of sentence symbols, making it easier to predict a symbol at the beginning or end of the sentence. As more symbols are filled near the boundaries, more evidence is accumulated for the decoding strategy to accurately predict symbols near the center. This process manifests either as monotonic or outside-in generation.

T	o_t	Uniform	Left2Right	Least2Most	Easy-First	Hard-First	Learned
10	$L \to 1$	22.38	22.38	27.14	22.21	26.66	12.70
10	$L \to 1$*	23.64	23.64	**28.63**	23.79	**28.46**	13.18
10	$\lceil L/T \rceil$	22.43	21.92	24.69	25.16	23.46	26.47
20	$L \to 1$	26.01	26.01	28.54	22.24	28.32	12.85
20	$L \to 1$*	27.28	27.28	**30.13**	24.55	**29.82**	13.19
20	$\lceil L/T \rceil$	24.69	25.94	27.01	27.49	25.56	27.82

Table 4.3: Constant-time machine translation on WMT'14 De→En with different settings of the budget (T) and number of tokens predicted each iteration (o_t). * denotes rescoring generated hypotheses using autoregressive model instead of proposed model.

4.6 Constant-Time Decoding: Result and Analysis

The trends in constant-time decoding noticeably differ from those in linear-time decoding. First, the **left2right** strategy performs comparably worse compared to the best performing strategies in constant-time decoding. The performance gap is wider (up to 4.8 BLEU) with a tighter budget ($T = 10$). Second, the **learned** coordinate selection strategy performs best when generating $\lceil L/T \rceil$ symbols every iteration, despite only being trained with linear-time decoding, but performs significantly worse when annealing the number of generated symbols from L to 1. This could be explained by the fact that the learned policy was never trained to refine predicted symbols, which is the case in $L \to 1$ constant-time decoding. Third, **easy-first** is the second-best performing strategy in the $\lceil L/T \rceil$ setting, but similarly to the **learned** strategy it performs worse in the $L \to 1$ setting. This may be because in the $L \to 1$ setting it is preferable to first generate hard-to-predict symbols and have multiple attempts at refining them, rather than predicting hard tokens at the end of generation process and not getting an opportunity to refine them, as is done in **easy-first** scenario. To verify this hypothesis, we test a **hard-first** strategy where we flip the signs of the coefficients of **easy-first** in the log-linear model. This new **hard-first** strategy works on par with **least2most**, again confirming that decoding strategies must be selected based on the target tasks and decoding setting.

With a fixed budget of $T = 20$, linearly annealing o_t from L to 1, and **least2most** decoding, constant-time translation can achieve translation quality comparable to linear-time translation with the same model (30.13 vs. 30.58), and to beam-search translations using the strong neural autoregressive model (30.13 vs 30.92). Even with a tighter budget of 10 iterations (less than half the average sentence length), constant-time translation loses only 1.8 BLEU points (28.63 vs. 30.58). The average length of the target sentence in the test set is 25.

We present the comparison of the results of our approach with other constant-time machine translation approaches in Table 4.4. Our model is the most similar to the conditional model by [Ghazvininejad et al., 2019]. However, there are differences in both model and training hyperparameters between our work and work by [Ghazvininejad et al., 2019]. We use a smaller Transformer model with 1024 hidden units vs 2048 units in [Ghazvininejad et al., 2019]. We also train the model with more than twice smaller batch size since we use 8 GPUs on DGX-1 machine and [Ghazvininejad et al., 2019] use 16 GPUs on two DGX-1 machine with float16 precision. Finally, we don't average best 5 checkpoints and don't use label smoothing for our model. Compared to other constant-time machine translation approaches, our model outperforms approaches by [Gu et al., 2017, Lee et al., 2018, Wang et al., 2019b, Ma et al., 2019], while being comparable to [Ghazvininejad et al., 2019, Chan et al., 2019, Shu et al., 2019] and slightly worse than [Saharia et al., 2020, Ghazvininejad et al., 2020].

4.7 Conclusion

We present a generalized framework of neural sequence generation that unifies decoding in directed and undirected neural sequence models. Under this framework, we separate position selection and symbol replacement, allowing us to apply a diverse set of generation algorithms, inspired by those for directed neural sequence models, to undirected models such as BERT and its translation variant.

We evaluate these generation strategies on WMT'14 En-De machine translation using a recently proposed masked translation model. Our experiments reveal that undirected neural sequence models achieve performance comparable to conventional, state-of-the-art autoregressive models, given an appropriate choice of decoding strategy. We further show that constant-time translation in these models performs similar to linear-time translation by using one of the proposed generation strategies. Analysis of the generation order automatically determined by these adaptive decoding strategies reveals that most sequences are generated either monotonically or outside-in.

4.8 Since the chapter release

In addition to non-autoregressive machine translation papers mentioned at the end of chapter 3, several papers investigating generation of text using the masked language models (MLMs) have been released after our chapter. Some of these papers investigating MLMs for text generation include [Kasai et al., 2020, Liao et al., 2020, Kreutzer et al., 2020, Shen et al., 2020]. Overall, we hope that our generalized framework opens new avenues in developing and understanding generation algorithms beyond masked language models for a variety of settings.

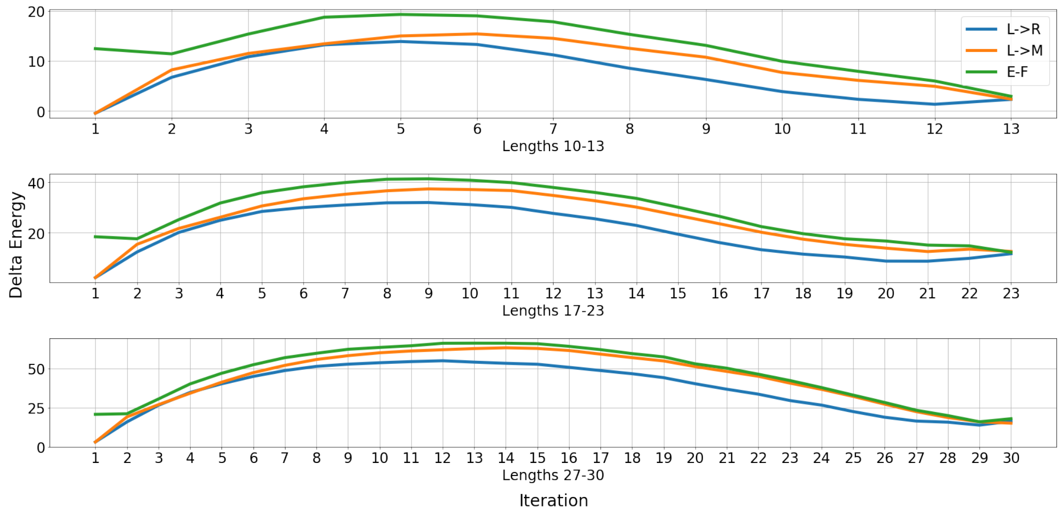


Figure 4.2: Average difference in energy ↑ between sequences generated by selecting positions uniformly at random versus by different algorithms, over the course of decoding.

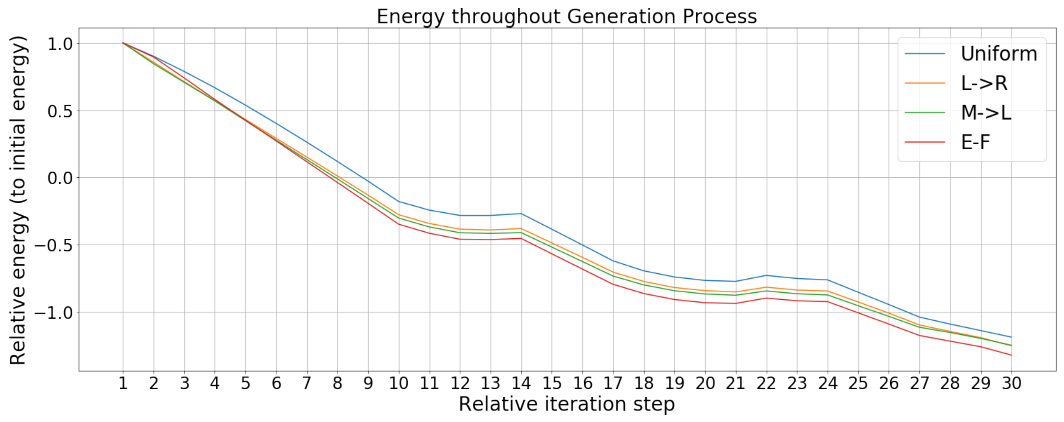


Figure 4.3: Evolution of the energy of the sequence ↓ over the course of decoding by different position selection algorithms.

Models	**WMT2014** **EN-DE**	**DE-EN**
AR Transformer-base [Vaswani et al., 2017]	27.30	–
AR [Gu et al., 2017]	23.4	–
NAR (+Distill +FT +NPD S=100)	21.61	–
AR [Lee et al., 2018]	24.57	28.47
Adaptive NAR Model	16.56	–
Adaptive NAR Model (+Distill)	21.54	25.43
AR [Wang et al., 2019b]	27.3	31.29
NAT-REG (+Distill)	20.65	24.77
NAT-REG (+Distill +AR rescoring)	24.61	28.90
AR [Ghazvininejad et al., 2019]	27.74	31.09
CMLM with 4 iterations	22.25	–
CMLM with 4 iterations (+Distill)	25.94	29.90
CMLM with 10 iterations	24.61	–
CMLM with 10 iterations (+Distill)	27.03	30.53
AR [Chan et al., 2019]	27.80	31.20
KERMIT (+Distill)	27.80	30.70
AR [Shu et al., 2019]	26.1	–
Latent-Variable NAR	11.8	–
Latent-Variable NAR (+Distill)	22.2	–
Latent-Variable NAR (+Distill +AR Rescoring)	25.1	–
AR [Ma et al., 2019]	27.16	31.44
FlowSeq-base (+NPD n = 30)	21.15	26.04
FlowSeq-base (+Distill +NPD n = 30)	23.48	28.40
AR [Ghazvininejad et al., 2020]	27.61	31.38
SMART with 4 iterations (+Distill)	27.03	30.87
SMART with 8 iterations (+Distill)	27.65	31.27
AR [Saharia et al., 2020]	27.80	31.20
Imputer with 8 iterations (+Distill)	28.00	31.00
Imputer with 8 iterations (+Distill)	28.20	31.30
AR (ours)	26.84	30.92
Contant-time 10 iterations	21.98	27.14
Contant-time 10 iterations (+AR Rescoring)	24.53	28.63
Contant-time 20 iterations	23.92	28.54
Contant-time 20 iterations (+AR Rescoring)	25.69	30.13

Table 4.4: BLEU scores on WMT'14 En→De and De→En datasets showing performance of various constant-time machine translation approaches. Each block shows the performance of autoregressive model baseline with their proposed approach. AR denotes autoregressive model. Distill denotes distillation. AR rescoring denotes rescoring of samples with autoregressive model. FT denotes fertility. NPD denotes noisy parallel decoding followed by rescoring with autoregressive model.

Iteration	Right-to-Left-to-Right-to-Left
(source)	Würde es mir je gelingen , an der Universität Oxford ein normales Leben zu führen ?
1	_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ **?**
2	_ _ _ _ _ _ _ _ _ _ _ _ _ _ **Oxford** ?
3	_ _ **ever** _ _ _ _ _ _ _ _ _ _ _ Oxford ?
4	_ **I** ever _ _ _ _ _ _ _ _ _ _ _ Oxford ?
5	_ I ever _ _ _ _ _ _ _ _ _ _ **of** Oxford ?
6	**Would** I ever _ _ _ _ _ _ _ _ _ _ of Oxford ?
7	Would I ever _ _ _ _ _ **normal** _ _ _ _ of Oxford ?
8	Would I ever _ _ _ _ _ normal _ **at** _ _ of Oxford ?
9	Would I ever _ _ _ _ _ normal _ at **the** _ of Oxford ?
10	Would I ever _ _ _ _ _ normal _ at the **University** of Oxford ?
11	Would I ever _ _ _ _ _ normal **life** at the University of Oxford ?
12	Would I ever _ _ _ **live** _ normal life at the University of Oxford ?
13	Would I ever _ _ _ live **a** normal life at the University of Oxford ?
14	Would I ever _ **able** _ live a normal life at the University of Oxford ?
15	Would I ever **be** able _ live a normal life at the University of Oxford ?
16	Would I ever be able **to** live a normal life at the University of Oxford ?
(target)	Would I ever be able to lead a normal life at Oxford ?

Figure 4.4: Example sentences generated following an right-to-left-to-right-to-left generation order, using the **easy-first** decoding algorithm on De→En.

Iteration	Outside-In
(source)	Doch ohne zivilgesellschaftliche Organisationen könne eine Demokratie nicht funktionieren .
1	_ _ _ _ _ _ _ _ _ _ **.**
2	_ _ _ _ _ _ _ _ **cannot** _ .
3	_ _ _ _ _ _ _ **democracy** cannot _ .
4	_ **without** _ _ _ _ _ democracy cannot _ .
5	_ without _ _ _ _ _ democracy cannot **work** .
6	**But** without _ _ _ _ _ democracy cannot work .
7	But without _ _ _ _ **a** democracy cannot work .
8	But without _ **society** _ _ a democracy cannot work .
9	But without _ society _ **,** a democracy cannot work .
10	But without **civil** society _ , a democracy cannot work .
11	But without civil society **organisations** , a democracy cannot work .
(target)	Yet without civil society organisations , a democracy cannot function .

Figure 4.5: Example sentences generated following an outside-in generation order, using the **easy-first** decoding algorithm on De→En.

Iteration	Left-to-Right
(source)	Denken Sie , dass die Medien zu viel vom PSG erwarten ?
1	_ _ _ _ _ _ _ _ _ _ _ _ **?**
2	**Do** _ _ _ _ _ _ _ _ _ _ _ ?
3	Do **you** _ _ _ _ _ _ _ _ _ _ ?
4	Do you **think** _ _ _ _ _ _ _ _ _ ?
5	Do you think _ _ _ _ _ _ _ **PS** _ ?
6	Do you think _ _ _ _ _ _ _ PS @**G** ?
7	Do you think _ **media** _ _ _ _ PS @G ?
8	Do you think **the** media _ _ _ _ PS @G ?
9	Do you think the media **expect** _ _ _ PS @G ?
10	Do you think the media expect _ **much** _ PS @G ?
11	Do you think the media expect **too** much _ PS @G ?
12	Do you think the media expect too much **of** PS @G ?
(target)	Do you think the media expect too much of PS @G ?

Figure 4.6: Example sentences generated following an left-to-right generation order, using the **easy-first** decoding algorithm on De→En.

Iteration	Right-to-Left
(source)	Ein weiterer Streitpunkt : die Befugnisse der Armee .
1	_ _ _ _ _ _ _ _ _ _ _ _ **.**
2	_ _ _ _ _ _ _ _ _ _ _ **army** .
3	_ _ _ _ _ _ _ _ _ **of** _ army .
4	_ _ _ _ _ _ _ _ _ of **the** army .
5	_ _ _ _ _ _ _ **powers** of the army .
6	_ _ _ _ _ _ **the** powers of the army .
7	_ _ _ _ _ **:** the powers of the army .
8	_ _ **point** : the powers of the army .
9	_ **contentious** point : the powers of the army .
10	**Another** contentious point : the powers of the army .
(target)	Another issue : the powers conferred on the army .

Figure 4.7: Example sentences generated following an right-to-left generation order, using the **easy-first** decoding algorithm on De→En.

Iteration	Right-to-Left-to-Right
(source)	Die Aktien von Flight Centre stiegen gestern um 3 Cent auf 38,20 Dollar .
1	_ _ _ _ _ _ _ _ _ _ _ _ _ **.**
2	_ _ _ _ _ _ _ _ _ _ _ _ **20** .
3	_ _ _ _ _ _ _ _ _ _ _ **38.** 20 .
4	_ _ _ _ _ _ _ _ _ _ **$** 38. 20 .
5	_ _ _ _ _ _ _ _ _ **to** $ 38. 20 .
6	**Flight** _ _ _ _ _ _ _ _ to $ 38. 20 .
7	Flight **Centre** _ _ _ _ _ _ _ to $ 38. 20 .
8	Flight Centre **'s** _ _ _ _ _ _ to $ 38. 20 .
9	Flight Centre 's **shares** _ _ _ _ _ to $ 38. 20 .
10	Flight Centre 's shares **rose** _ _ _ _ to $ 38. 20 .
11	Flight Centre 's shares rose **by** _ _ _ to $ 38. 20 .
12	Flight Centre 's shares rose by _ _ **yesterday** to $ 38. 20 .
13	Flight Centre 's shares rose by **3** _ yesterday to $ 38. 20 .
14	Flight Centre 's shares rose by 3 **cents** yesterday to $ 38. 20 .
(target)	Flight Centre shares were up 3c at $ 38.20 yesterday .

Figure 4.8: Example sentences generated following an Right-to-Left-to-Right generation order, using the **learned** decoding algorithm on De→En.

Iteration	Outside-In
(source)	Terminal 3 wird vor allem von kleineren US-Fluggesellschaften bedient .
1	_ _ _ _ _ _ _ _ _ **.**
2	**Terminal** _ _ _ _ _ _ _ _ .
3	Terminal **3** _ _ _ _ _ _ _ .
4	Terminal 3 _ _ _ _ _ _ **airlines** .
5	Terminal 3 _ _ _ _ _ **US** airlines .
6	Terminal 3 _ _ _ _ **smaller** US airlines .
7	Terminal 3 _ _ _ **by** smaller US airlines .
8	Terminal 3 **is** _ _ by smaller US airlines .
9	Terminal 3 is **mainly** _ by smaller US airlines .
10	Terminal 3 is mainly **served** by smaller US airlines .
(target)	Terminal 3 serves mainly small US airlines .

Figure 4.9: Example sentences generated following an Outside-In generation order, using the **learned** decoding algorithm on De→En.

Iteration	Left-to-Right
(source)	Die Gewinner des Team- und Einzelwettkampfs erhalten Preise .
1	_ _ _ _ _ _ _ _ _ _ _ **.**
2	**The** _ _ _ _ _ _ _ _ _ _ .
3	The **winners** _ _ _ _ _ _ _ _ _ .
4	The winners **of** _ _ _ _ _ _ _ _ .
5	The winners of **the** _ _ _ _ _ _ _ .
6	The winners of the **team** _ _ _ _ _ _ .
7	The winners of the team **and** _ _ _ _ _ .
8	The winners of the team and **individual** _ _ _ _ .
9	The winners of the team and individual **competitions** _ _ _ .
10	The winners of the team and individual competitions **will** _ _ .
11	The winners of the team and individual competitions will _ **prizes** .
12	The winners of the team and individual competitions will **receive** prizes .
(target)	The winners of the team and individual contests receive prizes .

Figure 4.10: Example sentences generated following an left-to-right generation order, using the **learned** decoding algorithm on De→En.

Chapter 5

Masked Graph Modeling for Molecule Generation

De novo, in-silico design of molecules is a challenging problem with applications in drug discovery and material design. Here, we introduce a masked graph model which learns a distribution over graphs by capturing all possible conditional distributions over unobserved nodes and edges given observed ones. We train our masked graph model on existing molecular graphs and then sample novel molecular graphs from it by iteratively masking and replacing different parts of initialized graphs. We evaluate our approach on the QM9 and ChEMBL datasets using the distribution-learning benchmark from the GuacaMol framework. The benchmark contains five metrics: the validity, uniqueness, novelty, KL-divergence and Fréchet ChemNet Distance scores, the last two of which are measures of the similarity of the generated samples to the training, validation and test distributions. We find that KL-divergence and Fréchet ChemNet Distance scores are anti-correlated with novelty scores. By varying generation initialization and the fraction of the graph masked and replaced at each generation step, we can increase the Fréchet score at the cost of novelty. In this way, we show that our model offers transparent and tunable control of the trade-off between these metrics, a key point of control in design applications currently lacking in other approaches to molecular graph generation. Our model outperforms previously proposed graph-based approaches and is competitive with SMILES-based approaches. Finally, we observe that minimizing validation loss on the training task is a suitable proxy for improving generation quality, which shows the suitability of optimizing the training objective for improving generation.

5.1 Introduction

The design of de novo molecules in-silico with desired properties is an essential part of drug discovery and material design but remains a challenging problem due to the very large combinatorial space of all possible synthesizable molecules [Bohacek et al., 1996]. Recently, various deep generative models for the task of molecular graph generation have been proposed, including: neural autoregressive models [Hochreiter and Schmidhuber, 1997b, Vaswani et al., 2017], variational autoencoders [Kingma and Welling, 2014, Rezende et al., 2014], adversarial autoencoders [Makhzani et al., 2015], and generative adversarial networks [Goodfellow et al., 2014, Elton et al., 2019]. A unifying theme behind these approaches is that they model the underlying distribution of molecular graphs. Once the underlying distribution is captured, new molecular graphs are sampled accordingly.

Each of these approaches makes unique assumptions about the underlying probabilistic structure of a molecular graph. Autoregressive models specify an ordering of atoms and bonds in advance to model the graph. Latent variable models such as variational autoencoders and adversarial autoencoders assume the existence of unobserved (latent) variables that capture complicated dependencies among the atoms and bonds. Unlike variational autoencoders, generative adversarial networks (GAN) do not use KL-divergence to measure the discrepancy between the model distribution and data distribution and instead estimate the divergence as a part of learning.

In this chapter, we propose a *masked graph model*, a generative model of graphs that learns the conditional distribution of masked graph components given the rest of the graph, induced by the underlying joint distribution. This allows us to use a procedure similar to Gibbs sampling to generate new molecular graphs, as Gibbs sampling requires access only to conditional distributions. By using conditional distributions, we circumvent the assumptions made by previous approaches to model the unconditional distribution. Our approach is inspired by masked language models [Devlin et al., 2019] that model the conditional distribution of masked words given the rest of a sentence, which have shown to be successful in natural language understanding tasks [Wang et al., 2018a, Wang et al., 2019a, Nogueira and Cho, 2019, Liu et al., 2019b, Lan et al., 2020, Lample and Conneau, 2019] and text generation [Mansimov et al., 2019]. We build a model for graphs rather than use a language model because the ability of a language model to model molecules is limited by the string representation used [Krenn et al., 2019]. By directly modeling molecular graphs, we bypass the need to find better ways of serializing molecules as strings.

We evaluate our approach on two popular molecular graph datasets, QM9 [Ruddigkeit et al., 2012, Ramakrishnan et al., 2014] and ChEMBL [Mendez et al., 2018], using a set of five distribution-learning metrics introduced in the GuacaMol benchmark [Brown et al., 2018]: the validity, uniqueness, novelty, KL-divergence [Kullback and Leibler, 1951] and Fréchet ChemNet Distance [Preuer et al., 2018] scores. After careful analysis, we find that the validity, Fréchet ChemNet Distance and KL-divergence scores are highly correlated with each other and inversely correlated with the novelty score. We show that our masked graph model offers higher flexibility than other models by more effectively trading off the novelty for the validity, Fréchet ChemNet Distance, and KL-divergence scores. Overall, the proposed masked graph model, trained on the graph representations of molecules, outperforms previously proposed graph-based generative models of molecules and performs comparably to several SMILES-based models. Additionally, our model achieves comparable performance on validity, uniqueness, and KL-divergence scores compared to state-of-the-art autoregressive SMILES-based models, but with lower Fréchet ChemNet Distance scores.

In order to verify the effectiveness of our training strategy for generation, we calculate the evaluation metrics for molecules generated from different training checkpoints, which correspond to different validation losses. We find that in general the values of the metrics increase as the validation loss decreases, demonstrating the suitability of the proposed training task for generation.

5.2 Background

We frame the problem of graph generation as sampling a graph G from a distribution $p^\star(G)$ defined over all possible graphs. As we do not have access to this underlying distribution, it is typical to explicitly model $p^\star(G)$ by a distribution $p_\theta(G)$. This is done using a function f_θ so that $p_\theta(G) = f_\theta(G)$. The parameters θ are learned by minimizing the KL-divergence $KL(p^\star \| p_\theta)$ between the true distribution and the parameterized distribution. Since we do not have access to $p^\star(G)$, we approximate $KL(p^\star \| p_\theta)$ by using a training set $D = (G_1, G_2, ..., G_M)$ which consists of samples from $p^\star$. Once we have trained our model on this distribution, we carry out generation by sampling from the trained model.

One powerful approach for parameterizing and sampling from such an unconditional distribution is autoregressive modeling [Hochreiter and Schmidhuber, 1997b, Mikolov et al., 2011, Vaswani et al., 2017, Bengio and Bengio, 1999, Larochelle and Murray, 2011]. An autoregressive model decomposes the distribution $p(G)$ as a product of temporal conditional distributions $p(g_t|G_{<t})$, where g_t is the vertex or edge to be added to G at time t and $G_{<t}$ are the vertices and edges that have been added in previous steps. Generation from an autoregressive model is often done sequentially by ancestral sampling. Defining such a distribution requires fixing an ordering of the nodes and vertices of a graph in advance. Although directed acyclic graphs have canonical orderings based on breadth-first search (BFS) and depth-first search (DFS), graphs can take a variety of valid orderings. The choice of ordering is largely arbitrary, and it is hard to predict how a particular choice of ordering will impact the learning process [Vinyals et al., 2016].

Another approach for building a generative model of graphs is to introduce a set of latent variables $Z = \{z_1, z_2, ..., z_k\}$ that aim to capture dependencies among the vertices V and edges E of a graph G. Unlike an autoregressive model, a latent variable model does not necessarily require a predefined ordering of the graph [Shu et al., 2019]. The generation process consists of first sampling latent variables according to their prior distributions, followed by sampling vertices and edges conditioned on these latent variable samples. However, learning the parameters θ of a latent variable model is more challenging than learning the parameters of an autoregressive model. It requires marginalizing latent variables to compute the marginal probability of a graph, i.e., $p(G) = \int_Z p(G|Z)p(Z)dZ$, which is often intractable. Recent approaches have focused on deriving a tractable lower-bound to the marginal probability by introducing an approximate posterior distribution $q(Z)$ and maximizing this lowerbound instead [Kingma and Welling, 2014, Rezende et al., 2014, Makhzani et al., 2015].

5.3 Model

In this chapter, we explore another approach to probabilistic graph generation based on the insight that we do not need to model the joint distribution $p(G)$ directly to be able to sample from it. Our approach, to which we refer as *masked graph modeling*, instead parameterizes and learns conditional distributions $p(\eta|G_{\setminus\eta})$ where η is a subset of the components (nodes and edges) of G and $G_{\setminus\eta}$ is a graph without those components (or equivalently with those components masked out). With these conditional distributions estimated from data, we sample a graph by iteratively updating its components. At each generation iteration, this involves choosing a subset of components, masking them, and sampling new values for them according to the corresponding conditional distribution.

There are two advantages to the proposed approach. First, we do not need to specify an arbitrary order of graph components, unlike in autoregressive models. Second, learning is exact, unlike in latent variable models where it is often necessary to maximize a tractable lowerbound instead of the exact likelihood. In the remainder of this section, we describe in detail parameterization, learning and generation.

5.3.1 Parameterization

A masked graph model (MGM) operates on a graph G, which consists of a set of N vertices $\mathcal{V} = \{v_i\}_{i=1}^{N}$ and a set of edges $\mathcal{E} = \{e_{i,j}\}_{i,j=1}^{N}$. A vertex is denoted by $v_i = (i, t_i)$, where i is the unique index assigned to it, and $t_i \in C_v = \{1, ..., T\}$ is its type, with T the number of node types. An edge is denoted by $e_{i,j} = (i, j, r_{i,j})$, where i, j are the indices to the incidental vertices of this edge and $r_{i,j} \in C_e = \{1, ..., R\}$ is the type of this edge, with R the number of edge types.

We use a single graph neural network to parameterize any conditional distribution induced by a given graph. We assume that the missing components η of the conditional distribution $p(\eta|G_{\backslash\eta})$ are conditionally independent of each other given $G_{\backslash\eta}$:

$$p(\eta|G_{\backslash\eta}) = \prod_{v\in\mathcal{V}} p(v|G_{\backslash\eta}) \prod_{e\in\mathcal{E}} p(e|G_{\backslash\eta}), \tag{5.1}$$

where $\mathcal{V}$ and $\mathcal{E}$ are the sets of all vertices and all edges in η respectively.

We start by embedding the vertices and edges in the graph $G_{\backslash\eta}$ to get continuous representations $h_{v_i} \in \mathbb{R}^{d_0}$ and $h_{e_{i,j}} \in \mathbb{R}^{d_0}$ respectively, where d_0 is the dimensionality of the continuous representation space [Bengio et al., 2003]. We then pass these representations to a message passing neural network (MPNN) [Gilmer et al., 2017]. We use an MPNN as the fundamental component of our model because of its invariance to graph isomorphism. An MPNN layer consists of an aggregation step that aggregates messages from each node's neighboring nodes, followed by an update step that uses the aggregated messages to update each node's representation. We stack L layers on top of each other to build an MPNN; parameters are tied across all L layers. For all except the last layer, the updated node and edge representations output from layer l are fed into layer $l+1$. Unlike the original version of the MPNN, we also maintain and update each edge's representation at each layer.

At each layer l of the MPNN, we first update the hidden state of each node v_i by computing its accumulated message $u_{v_i}^{(l)}$ using an aggregation function J_v and a spatial residual connection R between neighboring nodes:

$$\begin{aligned} u_{v_i}^{(l)} &= J_v(h_{v_i}^{(l-1)}, \{h_{v_j}^{(l-1)}\}_{j\in N(i)}, \{h_{e_{i,j}}^{(l-1)}\}_{j\in N(i)}) + R(\{h_{v_j}^{(l-1)}\}_{j\in N(i)}), \\ J_v(h_{v_i}^{(l-1)}, \{h_{v_j}^{(l-1)}\}_{j\in N(i)}, \{h_{e_{i,j}}^{(l-1)}\}_{j\in N(i)}) &= \sum_{j\in N(i)} h_{e_{i,j}}^{(l-1)} \cdot h_{v_j}^{(l-1)}, \\ R(\{h_{v_j}^{(l-1)}\}_{j\in N(i)}) &= \sum_{j\in N(i)} h_{v_j}^{(l-1)}, \\ h_{v_i}^{(l)} &= \text{LayerNorm}(\text{GRU}(h_{v_i}^{(l-1)}, u_{v_i}^{(l)})), \end{aligned}$$

where $N(i)$ is the set of indices corresponding to nodes that are in the one-hop neighbourhood of node v_i. GRU [Cho et al., 2014] refers to a gated recurrent unit which updates the representation of each node using its previous representation and accumulated message. LayerNorm [Ba et al., 2016] refers to layer normalization.

Similarly, the hidden states of each edge $h_{e_{i,j}}$ are updated using the following rule for all $j \in N(i)$:

$$h_{e_{i,j}}^{(l)} = J_e(h_{v_i}^{(l-1)} + h_{v_j}^{(l-1)}).$$

The sum of the two hidden representations of the nodes incidental to the edge is passed through J_e, a two-layer fully connected network with ReLU activation between the two layers [Nair and Hinton, 2010, Glorot et al., 2011], to yield a new hidden edge representation. The node and edge representations from the final layer are then processed by a node projection layer $A_v : \mathbb{R}^{d_0} \rightarrow \Lambda^T$ and an edge projection layer $A_e : \mathbb{R}^{d_0} \rightarrow \Lambda^R$, where Λ^T and Λ^R are probability simplices over node and edge types respectively. The result are the distributions $p(v|G_{\setminus\eta})$ and $p(e|G_{\setminus\eta})$ for all $v \in \mathcal{V}$ and all $e \in \mathcal{E}$.

5.3.2 Learning

We use fully observed graphs from a training dataset D. We corrupt each graph G with a corruption process $C(G_{\setminus\eta}|G)$, i.e. $G_{\setminus\eta} \sim C(G_{\setminus\eta}|G)$. In this work, following the work of [Devlin et al., 2019] for language models, we randomly replace some of the node and edge features with the special symbol MASK. After passing $G_{\setminus\eta}$ through our model we obtain the conditional distribution $p(\eta|G_{\setminus\eta})$. We then maximize the log probability $\log p(\eta|G_{\setminus\eta})$ of the masked components η given the rest of the graph $G_{\setminus\eta}$. This is analogous to a masked language model [Devlin et al., 2019], which predicts the masked words given the corrupted version of a sentence. This results in the following optimization problem:

$$\arg\max_{\theta} \mathbb{E}_{G\sim D}\mathbb{E}_{G_{\setminus\eta}\sim C(G_{\setminus\eta}|G)} \log p_{\theta}(\eta|G_{\setminus\eta}).$$

5.3.3 Generation

To begin generation, we initialize a molecule in one of two ways, corresponding to different levels of entropy. The first way, which we call training initialization, uses a random graph from the training data as an initial graph. The second way, which we call marginal initialization, initializes each graph component according to a categorical distribution over the values that component takes in our training set. For example, the probability of an edge having type $r \in C_e$ is equal to the fraction of edges in the training set of type r.

We then use an approach motivated by Gibbs sampling to update graph components iteratively from the learned conditional distributions. At each generation step, we sample uniformly at random a fraction α of components η in the graph and replace the values of these components with the MASK symbol. We compute the conditional distribution $p(\eta|G_{\setminus\eta})$ by passing the partially masked graph through the model, sampling new values of the masked components according to the predicted distribution, and placing these values in the graph. We repeat this procedure for a total of T steps, where T is a hyperparameter.

5.4 Methods

We evaluate the proposed masked graph modeling approach for molecular graph generation. Atoms and bonds in a molecule correspond to nodes and edges in a graph, respectively. In this section, we outline the experimental setup used to carry out this evaluation, including datasets, evaluation framework, model details and training and generation procedures.

5.4.1 Datasets and Evaluation

We evaluate our approach using two widely used [Gómez-Bombarelli et al., 2016,Simonovsky and Komodakis, 2018, Li et al., 2018b] datasets of small molecules: QM9 [Ruddigkeit et al., 2012,Ramakrishnan et al., 2014] and ChEMBL [Mendez et al., 2018]. The QM9 dataset consists of approximately 132,000 molecules with a median and maximum of 9 heavy atoms each. Each atom is of one of the following $T = 5$ types: B, C, N, O, and F. Each bond is either a no-bond, single, double, triple or aromatic bond ($R = 5$). The ChEMBL dataset contains approximately 1,591,000 molecules with a median of 27 and a maximum of 88 heavy atoms each. It contains 12 types of atoms ($T = 12$): B, C, N, O, F, Si, P, S, Cl, Se, Br, and I. Each bond is either a no-bond, single, double, triple or aromatic bond ($R = 5$).

The QM9 dataset is split into training and validation sets, while the ChEMBL dataset is split into training, validation and test sets. In the remainder of this chapter, we use the term dataset distribution to refer to the distribution of the combined training and validation sets for QM9, and the combined training, validation and test sets for ChEMBL. Similarly, we use the term dataset molecule to refer to a molecule from the combined QM9 or ChEMBL dataset.

To numerically evaluate our approach, we use the GuacaMol benchmark [Brown et al., 2018], a suite of benchmarks for evaluating molecular graph generation approaches. Specifically, we evaluate our model using distribution-learning metrics from GuacaMol: the validity, uniqueness, novelty, KL-divergence [Kullback and Leibler, 1951] and Fréchet ChemNet Distance [Preuer et al., 2018] scores. GuacaMol uses 10,000 randomly sampled molecules to calculate each of these scores. Validity measures the ratio of valid molecules, uniqueness estimates the proportion of generated molecules that remain after removing duplicates and novelty measures the proportion of generated molecules that are not dataset molecules. The KL-divergence score compares the distributions of a variety of physiochemical descriptors estimated from the dataset and a set of generated molecules. The Fréchet ChemNet Distance score [Preuer et al., 2018] measures the proximity of the distribution of generated molecules to the distribution of the dataset molecules. This proximity is measured according to the Fréchet Distance in the hidden representation space of ChemNet, which is trained to predict the chemical properties of small molecules [Goh et al., 2017].

5.4.2 Property Embeddings

5.4.2.1 Node Property Embeddings

We represent each node using six node properties indexed as $\{\kappa \in \mathbb{Z} : 1 \leq \kappa \leq 6\}$, each with its own one-hot embedding. During the forward pass, each of these embeddings is multiplied by a separate weight matrix $W_\kappa \in \mathbb{R}^{T_\kappa \times d_0}$, where T_κ is the number of categories for property κ. The resulting continuous embeddings are summed together to form an overall embedding of the node. The entries of the one-hot embeddings for each of the properties are:

- **Atom type:** chemical symbol (e.g. C, N, O) of the atom;
- **Number of hydrogens:** number of hydrogen atoms bonded to the atom;
- **Charge:** net charge on the atom;
- **Chirality type:** unspecified, tetrahedral clockwise, tetrahedral counter-clockwise, other;
- **Is-in-ring:** atom is or is not part of a ring structure;
- **Is-aromatic:** atom is or is not part of an aromatic ring.

Each one-hot embedding also has an additional entry corresponding to the MASK symbol.

After processing the graph with the MPNN, we pass the representation of each node through six separate fully-connected two-layer networks with ReLU activation between the layers. For each node, the output of each network is a distribution over the categories of the initial one-hot vector for one of the properties. During training, we calculate the cross-entropy loss between the predicted distribution and the ground-truth for all properties that were masked out by the corruption process.

The choice of nodes for which a particular property is masked out is independent of the choice made for all other properties. The motivation for this is to allow the model to more easily learn relationships between different property types. The atom-level property information that we use in our model is the same as that provided in the SMILES string representation of a molecule.

Since the ChEMBL dataset does not contain chirality information, the chirality type embedding is superfluous for ChEMBL.

5.4.2.2 Edge Property Embeddings

We use the same framework as described for node property embeddings. We only use one edge property with the weight matrix $\mathcal{W} \in \mathbb{R}^{R \times d_0}$, whose one-hot embedding is defined as follows:

- **Bond type:** no, single, double, triple or aromatic bond.

5.4.3 Model Architecture, Training and Generation

For the QM9 dataset, we use one 4-layer MPNN, with parameter sharing between layers. For the ChEMBL dataset, we use one 6-layer MPNN with parameter sharing. We use more layers for ChEMBL because more message passing iterations are needed to cover a larger graph. For both datasets, we use an embedding dimensionality $d_0 = 2048$. We use the Adam optimizer [Kingma and Ba, 2014] with learning rate set to 0.0001, $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We use a batch size of 800 molecules for QM9 and 512 molecules for ChEMBL.[2] We clip the gradient for its norm to be at most 10.

During training, we uniformly at random mask each node feature (including atom type) and edge feature (including bond type) with probability α, while randomly varying α uniformly between 0 and 0.2. Nodes are considered as neighbors in the MPNN if they are connected by an edge that is either masked out, or does not have bond type no-bond. During validation, we follow the same procedure but with α fixed at 0.1, so that we can clearly compare model checkpoints and choose the checkpoint with the lowest validation loss for generation.

For QM9, we carry out generation experiments while using a masking rate of either 10% or 20%, corresponding to the mean and maximum masking rates during training respectively. For ChEMBL, we use a masking rate of either 1% or 5%, as we found that the higher masking rates led to low validity scores in our preliminary experiments. The number of edges masked and replaced for a median ChEMBL molecule with a 1% masking rate and for a median QM9 molecule with a 10% masking rate are both approximately 4. This indicates that the absolute number rather than portion of components masked out directly impacts generation quality. We use the same independence constraint during generation as we use during training when choosing which properties to mask out for each node or edge. We vary the initialization strategy between training and marginal initialization.

For QM9, we run 400 sampling iterations sequentially to generate a sequence of sampled graphs. For ChEMBL, we run 300 iterations. We calculate the GuacaMol evaluation metrics for our samples after every generation step for the first 10 steps, and then every 10-20 steps, in order to observe how generation quality changes with the number of generation steps.

[2]We perform 16 forward-backward steps with minibatches of 32 each to compute the gradient of the minibatch of 512 molecules, in order to cope with the limited memory size on a GPU.

5.4.4 Details of Baseline Models

We train two variants of the Transformer [Vaswani et al., 2017] architecture: Small and Regular. Both variants of Transformer architecture are trained on the SMILES string representation of the molecular graphs. The Transformer Regular architecture consists of 6 layers, 8 attention heads, embedding size of 1024, hidden dimension of 1024, and dropout of 0.1. The Transformer Small architecture consists of 4 layers, 8 attention heads, embedding size of 512, hidden dimension of 512, and dropout of 0.1. Both Transformer Small and Regular are trained with a batch size of 128 until the validation cross-entropy loss stops improving. We set the learning rate of the Adam optimizer to 0.0001, $\beta_1 = 0.9$ and $\beta_2 = 0.98$. The learning rate is decayed based on the inverse square root of the number of updates. We use the same hyperparameters for the Transformer Small and Regular models on both QM9 and ChEMBL.

We follow the open-source implementation of the GuacaMol benchmark baselines[3] for training an LSTM model on QM9. We use SMILES string representation of molecules to train LSTM model. Specifically, we train the LSTM with 3 layers of hidden size 1024, dropout of 0.2 and batch size of 64, using the Adam optimizer with learning rate 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We do not train the rest of the baseline models ourselves. For QM9: CharacterVAE [Gómez-Bombarelli et al., 2016], GrammarVAE [Kusner et al., 2017], GraphVAE [Simonovsky and Komodakis, 2018], and MolGAN [Cao and Kipf, 2018] results are taken from [Cao and Kipf, 2018]. For ChEMBL: AAE [Makhzani et al., 2015], ORGAN [Guimaraes et al., 2017], Graph MCTS [Jensen, 2018], VAE, and LSTM results are taken from [Brown et al., 2018]. NAT GraphVAE results are taken from [Kwon et al., 2019] for both QM9 and ChEMBL.

[3]`https://github.com/BenevolentAI/guacamol_baselines`

	Validity	Uniqueness	Novelty	KL Div	Fréchet Dist
Validity	1.00	-0.56	-0.83	0.73	0.75
Uniqueness	-0.56	1.00	0.50	-0.32	-0.37
Novelty	-0.83	0.50	1.00	-0.94	-0.95
KL Div	0.73	-0.32	-0.94	1.00	0.99
Fréchet Dist	0.75	-0.37	-0.95	0.99	1.00

Table 5.1: Spearman's correlation coefficient between benchmark metrics for results using the masked graph model on the QM9 dataset.

	Validity	Uniqueness	Novelty	KL Div	Fréchet Dist
Validity	1.00	0.03	-0.99	0.98	0.98
Uniqueness	0.03	1.00	0.00	0.03	0.03
Novelty	-0.99	0.00	1.00	-0.99	-0.99
KL Div	0.98	0.03	-0.99	1.00	1.00
Fréchet Dist	0.98	0.03	-0.99	1.00	1.00

Table 5.2: Spearman's correlation coefficient between benchmark metrics for results using LSTM, Transformer Small and Transformer Regular on the QM9 dataset.

5.5 Results and Discussion

5.5.1 Mutual Dependence of Metrics from GuacaMol

We first attempt to determine whether dependence exists between metrics from the Guacamol framework. We do this because we notice that some of these metrics may measure similar properties. For example, the Fréchet and KL scores are both measures of similarity between generated samples and a dataset distribution. If the metrics are not mutually independent, comparing models using a straightforward measure such as the sum of the metrics may not be a reasonable strategy.

To determine how the five metrics are related to each other, we calculate pairwise the Spearman (rank) correlation between all metrics on QM9, presented in Table 5.1, while varying the masking rate, initialization strategy and number of sampling iterations. We carry out a similar run for the Transformer Small, Transformer Regular, and LSTM baselines as follows. Each of these autoregressive models has a distribution output by a softmax layer over the SMILES vocabulary at each time step. We implement a sampling temperature parameter in this distribution to control its sharpness. By increasing the temperature, we decrease the sharpness, which increases the novelty. The Spearman correlation results for these baselines are shown in Table 5.2.

From Tables 5.1–5.2, we make three observations. First, the validity, KL-divergence and Fréchet Distance scores correlate highly with each other. Second, these three metrics correlate negatively with the novelty score. Finally, uniqueness does not correlate strongly with any other metric.

These observations suggest that we can look at a subset of the metrics, namely the uniqueness, Fréchet and novelty scores, to gauge generation quality. In the next section, we carry out experiments to determine how well MGM and baseline models perform on the anti-correlated Fréchet and novelty scores, which are representative of four of the five evaluation metrics. We observe how effectively each model trades these metrics off against each other.

5.5.2 Analysis of Representative Metrics

To examine how the masked graph model and baseline autoregressive models trade off the Fréchet ChemNet Distance and novelty scores, we plot these two metrics against each other in Figure 5.3. To obtain the points for the masked graph models, we evaluate the scores after various numbers of generation steps. For the QM9 MGM points, we use both training and marginal initializations, which start from the top left and bottom right of the graph respectively, and converge in between. For the ChEMBL MGM points, we use only training initialization.

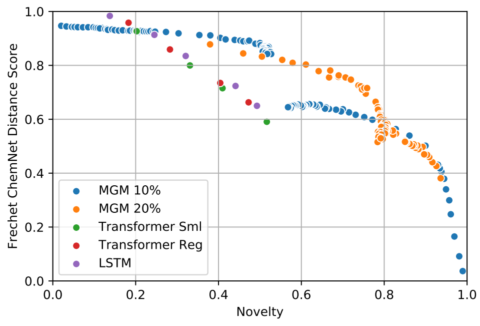


Figure 5.1: QM9

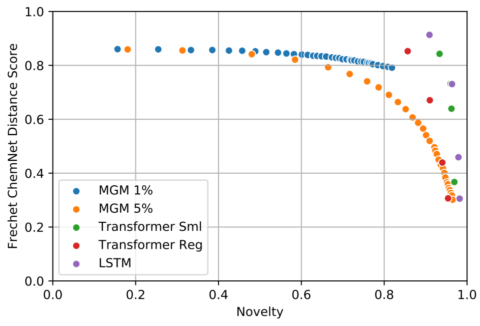


Figure 5.2: ChEMBL

Figure 5.3: Plots of the Fréchet ChemNet Distance score against novelty. The plots are generated by varying generation hyperparameters (number of generation iterations for the masked graph models and sampling temperature for autoregressive models).

On both QM9 and ChEMBL, we see that as novelty increases, the Fréchet ChemNet Distance score decreases for the masked graph models as well as for the LSTM and Transformer models. We also see that the line's slope, which represents the marginal change in Fréchet ChemNet Distance score per unit change in novelty score, has a lower magnitude for the masked graph model than for the autoregressive models. This shows that our model trades off novelty for similarity to the dataset distributions (as measured by the Fréchet score) more effectively relative to the baseline models. This gives us a higher degree of controllability in generating samples that are optimized towards either metric to the extent desired.

On QM9, we see that our masked graph models with a 10% or 20% masking rate maintain a larger Fréchet ChemNet Distance score as the novelty increases, compared to the LSTM and Transformer models. Several of the MGM points on the plot are beyond the Pareto frontier formed by each baseline model. On ChEMBL, the LSTM and Transformer models generally achieve a higher combination of novelty and Fréchet ChemNet Distance score than does the masked graph model with either masking rate.

However, to the bottom right of Figure 5.2, we can see a few points corresponding to the 5% masking rate that are beyond the Pareto frontier of the points formed by the Transformer Regular model.

We also observe that for ChEMBL, which contains larger molecules, using a 1% masking rate yields points that are beyond the Pareto frontier of those obtained using a 5% masking rate. This further indicates that masking a large number of components hurts generation quality, even if this number represents a small percentage of the graph. In the next section, we further explore the relationship between masking rate, initialization strategy and generation quality.

5.5.3 Effect of Generation Hyperparameters on Generation Quality

We analyze the effect of changing the masking rate and graph initialization on generation quality. In order to do so, we must choose results corresponding to a certain number of generation steps for each combination of masking rate and initialization. We therefore evaluate samples at intermediate steps of the generation process, as shown in Figure 5.8, to determine how the values of the evaluation metrics change as the number of generation steps increases.

For training initialization (Figures 5.4 and 5.6), the initialized molecules have perfect validity, uniqueness, KL and Fréchet scores, and zero novelty score. As generation proceeds, changes are made to the training molecules, yielding some invalid molecules, so the validity decreases. Some of the changes yield new, valid molecules, so the novelty increases. These molecules are less similar to the dataset distributions than the training molecules are themselves, so the KL and Fréchet scores decrease. On the other hand, for marginal initializations (Figures 5.5 and 5.7), the initialized molecules are less likely to be valid or similar to the dataset molecules. The probability of obtaining duplicate

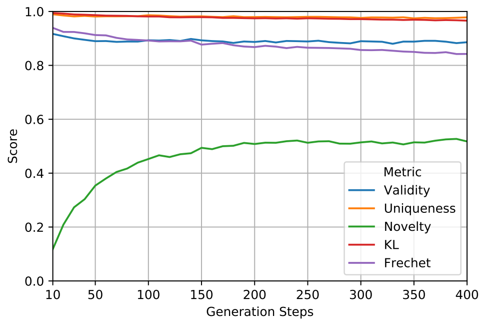


Figure 5.4: Training initialization, 10% masking rate

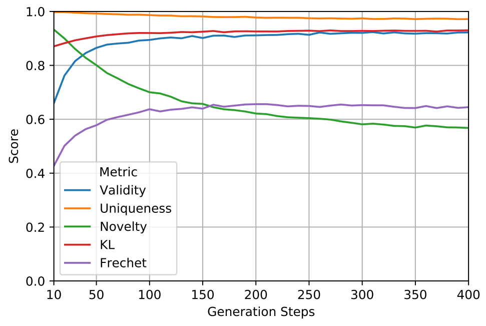


Figure 5.5: Marginal initialization, 10% masking rate

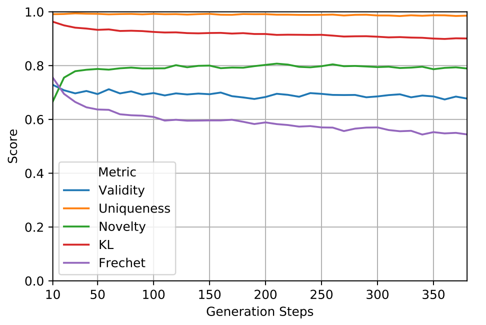


Figure 5.6: Training initialization, 20% masking rate

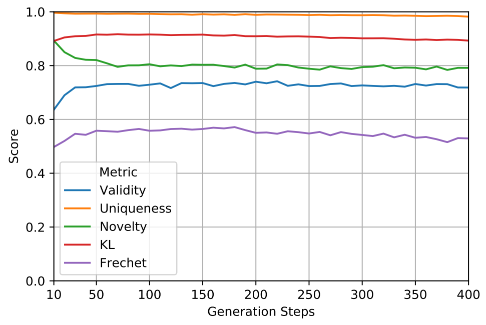


Figure 5.7: Marginal initialization, 20% masking rate

Figure 5.8: Plots of generation scores as a function of number of generation steps for each initialization and masking rate on QM9.

molecules is low as well. Over time, the molecules converge to valid structures similar to the dataset molecules, so the validity, KL and Fréchet scores increase. For both training and marginal initializations, different initialized molecules may converge to the same molecule over time, lowering uniqueness.

For all configurations and all metrics, the slope of the score with respect to the number of generation steps tends to flatten over time. When presenting the results of our model for different masking rates and initializations, we use the benchmark scores at the final generation step.

Dataset	Mask Rate	Graph Init	Valid	Uniq	Novel	KL Div	Fréchet Dist
QM9	10%	train	0.886	0.978	0.518	0.966	0.842
	10%	marginal	0.922	0.972	0.568	0.930	0.645
	20%	train	0.678	0.988	0.789	0.901	0.544
	20%	marginal	0.719	0.982	0.792	0.893	0.529
ChEMBL	1%	train	0.849	1.000	0.722	0.987	0.845
	5%	train	0.558	1.000	0.952	0.869	0.396

Table 5.3: Effect of varying masking rate and graph initialization on the benchmark results for our masked graph model on QM9 and ChEMBL.

We now use these results to analyze the effect of changing the masking rate and graph initialization for generation in Table 5.3. On QM9, we find that using marginal initialization leads to slightly higher validity and novelty scores however with lower KL-divergence and Fréchet ChemNet Distance scores compared with using training initialization. When using marginal initialization, the masked graph model generates marginally more novel molecules at the expense of not capturing the properties of dataset molecules as well. On ChEMBL, the marginal initialization strategy results in validity scores close to 0, which is why we only consider the training initialization strategy in Table 5.3. On both QM9 and ChEMBL, novelty increases significantly when increasing the masking rate while the validity, KL-divergence and Fréchet Distance scores drop.

Close observation of the results in Table 5.3 suggests that the choice of masking rate and initialization strategy impacts the balance among the five metrics. Most significantly, increasing the masking rate results in a higher novelty score, and lower KL-divergence and Fréchet Distance scores. We can trade off between different metrics as desired by adjusting the initialization and masking rate.

	Model	Valid	Uniq	Novel	KL Div	Fréchet Dist
SMILES	CharacterVAE	0.103	0.675	0.900	N/A	N/A
	GrammarVAE	0.602	0.093	0.809	N/A	N/A
	LSTM (ours)	0.980	0.962	0.138	0.998	0.984
	Transformer Sml (ours)	0.947	0.963	0.203	0.987	0.927
	Transformer Reg (ours)	0.965	0.957	0.183	0.994	0.958
Graph	GraphVAE	0.557	0.760	0.616	N/A	N/A
	MolGAN	0.981	0.104	0.942	N/A	N/A
	NAT GraphVAE	0.945	0.343	0.806	N/A	N/A
	MGM (ours proposed)	0.886	0.978	0.518	0.966	0.842

Table 5.4: Distributional Results on QM9. CharacterVAE [Gómez-Bombarelli et al., 2016], GrammarVAE [Kusner et al., 2017], GraphVAE [Simonovsky and Komodakis, 2018] and MolGAN [Cao and Kipf, 2018] results are taken from [Cao and Kipf, 2018]. NAT GraphVAE [Kwon et al., 2019] stands for non-autoregressive graph VAE. Our masked graph model results correspond to a 10% masking rate and training graph initialization, which has the highest geometric mean for all five benchmarks. Values of validity(↑), uniqueness(↑), novelty(↑), KL Div(↑) and Fréchet Dist(↑) metrics are between 0 and 1.

	Model	Valid	Uniq	Novel	KL Div	Fréchet Dist
SMILES	AAE	0.822	1.000	0.998	0.886	0.529
	ORGAN	0.379	0.841	0.687	0.267	0.000
	VAE	0.870	0.999	0.974	0.982	0.863
	LSTM	0.959	1.000	0.912	0.991	0.913
	Transformer Sml (ours)	0.920	0.999	0.939	0.968	0.859
	Transformer Reg (ours)	0.961	1.000	0.846	0.977	0.883
Graph	Graph MCTS	1.000	1.000	0.994	0.522	0.015
	NAT GraphVAE	0.830	0.944	1.000	0.554	0.016
	MGM (ours proposed)	0.849	1.000	0.722	0.987	0.845

Table 5.5: Distributional Results on ChEMBL. LSTM, Graph MCTS [Jensen, 2018], AAE [Polykovskiy et al., 2018], ORGAN [Guimaraes et al., 2017] and VAE [Simonovsky and Komodakis, 2018] (with a bidirectional GRU [Cho et al., 2014] as encoder and autoregressive GRU [Cho et al., 2014] as decoder) results are taken from [Brown et al., 2018]. NAT GraphVAE [Kwon et al., 2019] stands for non-autoregressive graph VAE. Our masked graph model results correspond to a 1% masking rate and training graph initialization, which has the highest geometric mean for all five benchmarks. Values of validity(↑), uniqueness(↑), novelty(↑), KL Div(↑) and Fréchet Dist(↑) metrics are between 0 and 1.

5.5.4 Comparison with Baseline Models

We now compare our results on each dataset using our 'best' initialization strategy to baseline models. In previous sections, we have shown that the GuacaMol benchmark metrics are correlated and that our model can efficiently trade these metrics off against each other. Thus we cannot say that one generation strategy definitively outperforms another unless it achieves a higher score on each of the five metrics. However, for the sake of comparison with baseline models, we pick one generation strategy as follows: we select results from Table 5.3 for each dataset corresponding to the highest geometric mean among all five metrics. The distributional benchmark results on QM9 and ChEMBL are shown in Table 5.4 and Table 5.5 respectively.

On QM9, our model performs comparably to existing methods. Our approach shows higher validity and uniqueness scores compared to CharacterVAE [Gómez-Bombarelli et al., 2016], GrammarVAE [Kusner et al., 2017], GraphVAE [Simonovsky and Komodakis, 2018] and MolGAN [Cao and Kipf, 2018], while having a lower novelty score. Our model has a lower validity and novelty score than non-autoregressive graph VAE [Kwon et al., 2019] while having a significantly higher uniqueness score. Compared to the autoregressive LSTM and Transformer models, our model has lower validity, KL-divergence and Fréchet Distance scores; however it exhibits slightly higher uniqueness and significantly higher novelty scores. Since KL-divergence and Fréchet scores are not available for the graph-based baselines as well as for CharacterVAE and GrammarVAE, we compare graph-based baselines to our model using these metrics on ChEMBL.

On ChEMBL, our approach outperforms existing graph-based methods. Compared to graph MCTS [Jensen, 2018] and non-autoregressive graph VAE [Kwon et al., 2019], our approach shows lower novelty scores while having significantly higher KL-divergence and Fréchet Distance scores. The baseline graph-based models do not capture the properties of the dataset distributions, as shown by their low KL-divergence scores and almost-zero Fréchet scores. This demonstrates that our proposed approach outperforms graph-based methods in generating novel molecules that are similar to the dataset distributions.

The proposed masked graph model is competitive with models that rely on the SMILES representations of molecules. It outperforms the GAN-based model (ORGAN) across all five metrics and outperforms the adversarial autoencoder model (AAE) on all but the uniqueness score (both have the maximum possible score) and the novelty score. It performs comparably to the VAE model with an autoregressive GRU [Cho et al., 2014] decoder on all metrics except novelty.

Our approach lags behind the LSTM, Transformer Small and Transformer Regular SMILES-based models on the ChEMBL dataset. It outperforms both Transformer models on KL-divergence score but underperforms them on validity, novelty and Fréchet score. Our approach also results in lower scores across most of the metrics when compared to the LSTM model.

There are several differences between the QM9 and ChEMBL datasets that could account for this, including number of molecules, median molecule size and presence of chirality information. There has also been extensive work in developing language models compared to graph neural networks, which may account for the greater success of the LSTM and Transformers. We leave further investigation into the reasons behind the difference in performance to future work.

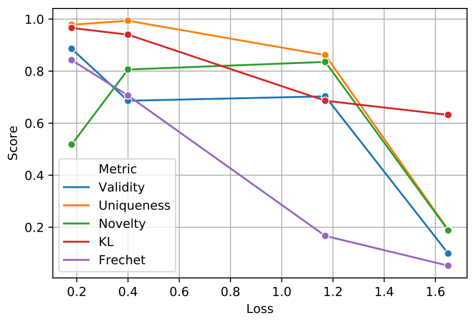


Figure 5.9: Training Initialization

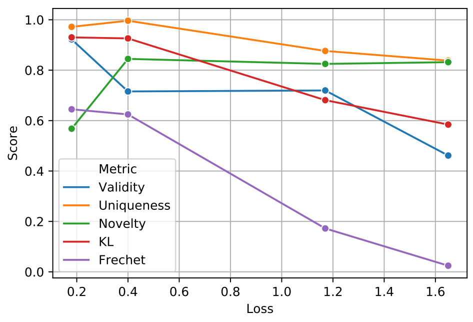


Figure 5.10: Marginal Initialization

Figure 5.11: Benchmark metric results on QM9 corresponding to our model's checkpoints corresponding to different validation loss values. A masking rate of 10% was used.

5.5.5 Effect of Validation Loss on Generation Quality

To determine whether validation loss is a suitable proxy for generation quality, we carry out generation from different training checkpoints of our 'best' QM9 model. During training, we carried out a hyperparameter search to find the configurations with the lowest validation loss, which we used as the criterion to select the best model for generation. The experiments in this subsection explore whether this choice is justified.

Figure 5.11 shows the values of all five benchmark metrics corresponding to different loss values (i.e., different checkpoints) of our model. In general, as the validation loss increases, the metrics' values decrease. We attribute the decrease in validity to the fact that a less well-trained model is less likely to have learned enough about the relationship between different parts of a graph to predict masked components that respect the chemical constraints inherent in this type of data. The increase in novelty and decrease in KL and Fréchet scores are explained by better-trained models being more likely to predict masked components from the most similar context in the training/validation data. Occasionally this causes our model to generate an exact copy of a molecule from the training dataset, lowering the novelty; in general, it produces molecules whose local neighborhoods are similar to those of molecules in the training/validation data, thereby increasing the KL and Fréchet scores. The sharp decrease in novelty and uniqueness as the loss increases from 1.17 to 1.65 can be attributed to the low validity, as GuacaMol implicitly penalizes all metrics when the validity drops below 0.5.

We conclude that selecting the model with the lowest validation loss for generation is a reasonable strategy. This implies that using more powerful graph neural networks within our *masked graph modeling* framework could improve generation quality. Finding model architectures that lower the validation loss is a good direction for future work.

5.5.6 Generation Trajectories

We present a few sampling trajectories of molecules from the proposed masked graph model in Figures 5.14–5.17. Each image represents the molecule after a certain number of sampling iterations; the first image in a figure is the molecular graph initialization before any sampling steps are taken. Figure 5.14 shows a trajectory each for training and marginal initializations with a 10% masking rate. Figure 5.17 shows a trajectory each for 1% and 5% masking rates with training initialization. All molecules displayed in the figures are valid, but molecules corresponding to some of the intermediate steps not shown may not be.

Figure 5.12 shows the trajectory of a molecule initialized as a molecule from the QM9 training set. As generation progresses, minor changes are made to the molecule, yielding novel molecules. After 100 generation steps, the molecule has converged to another non-novel molecule. Further generation steps yield novel molecules once again, with the molecule's structure gradually moving further away from the initialized molecule.

Figure 5.13 shows the trajectory of a molecule initialized from the marginal dis-

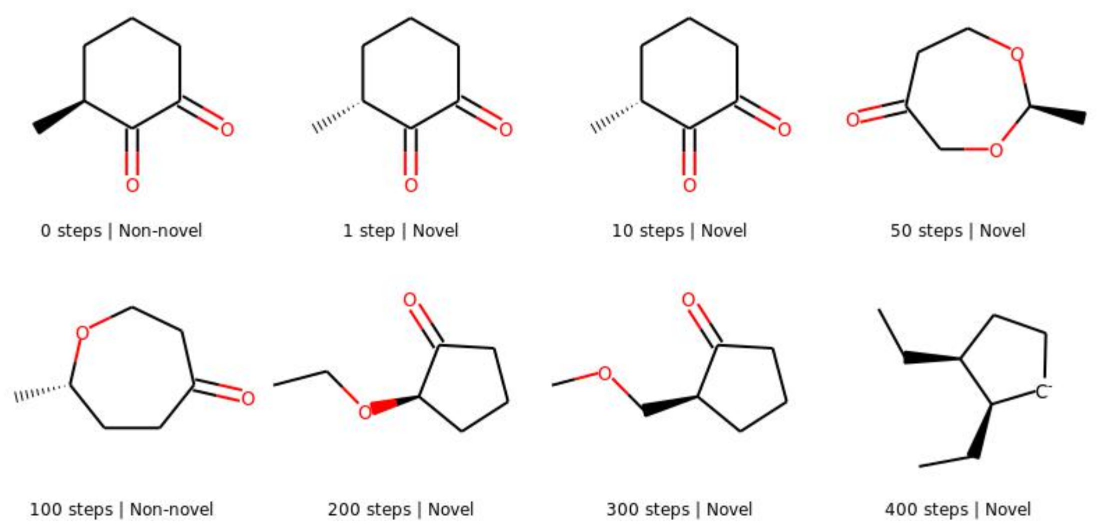


Figure 5.12: Training initialization

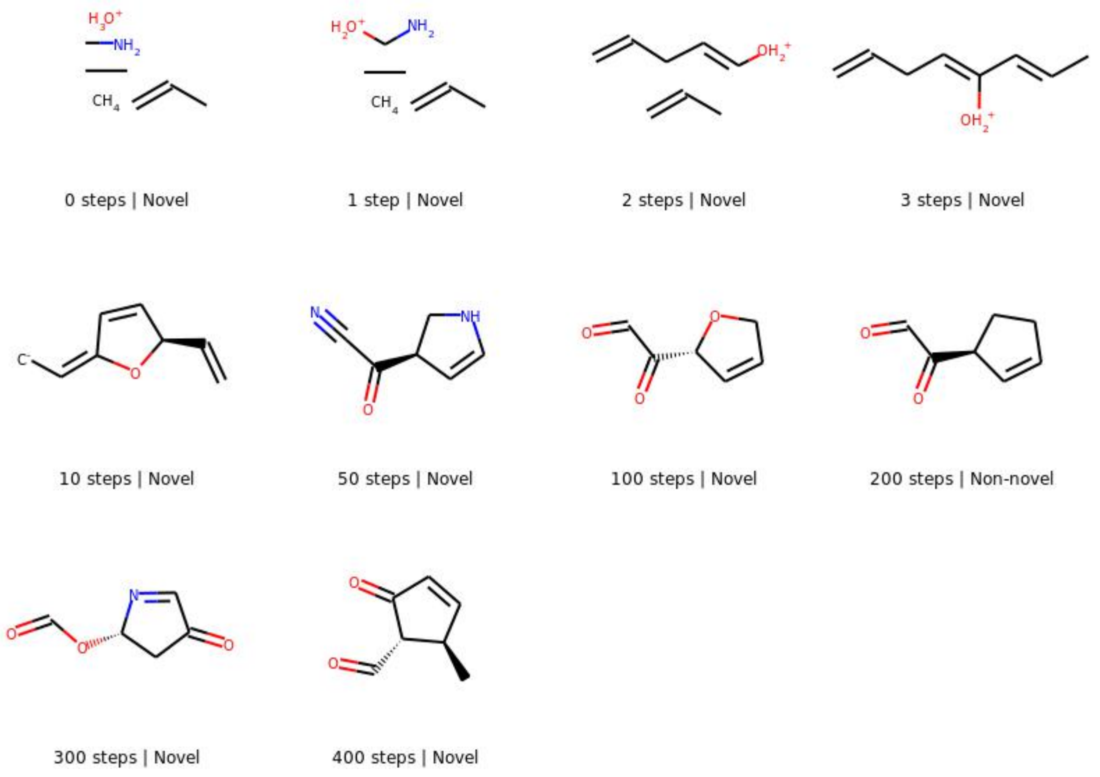


Figure 5.13: Marginal initialization

Figure 5.14: Generation trajectory of a molecule each for training initialization and marginal initialization, for QM9 with a 10 % masking rate.

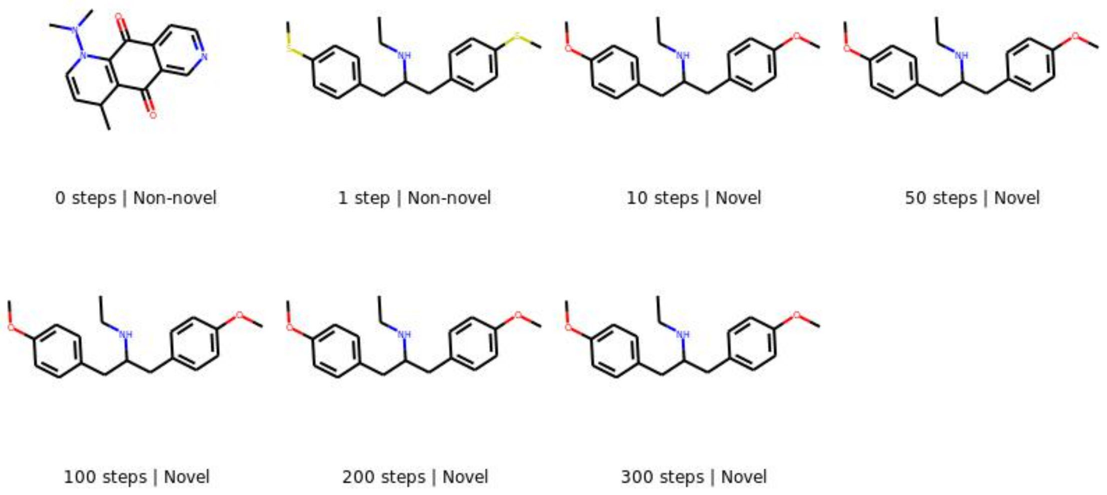


Figure 5.15: 1% masking rate

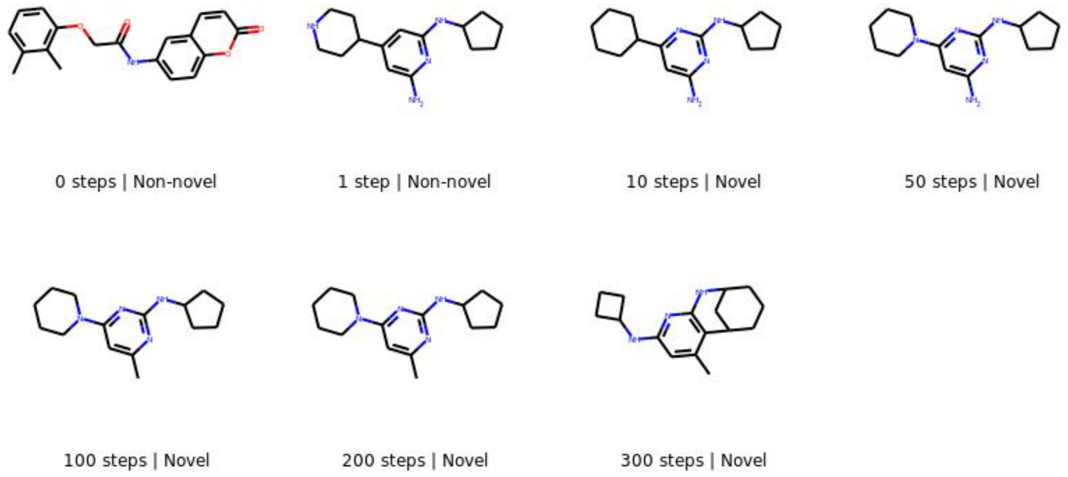


Figure 5.16: 5% masking rate

Figure 5.17: Generation trajectory of a molecule each for a 1% and 5% masking rate, for ChEMBL with training initialization.

tribution of the QM9 training set. The initialized graph consists of multiple disjoint molecular fragments. Over the first three generation steps, the various nodes are connected to form a connected graph. These changes are more drastic than those in the first few steps of generation with training initialization. The molecule undergoes significant changes over the next few steps until it forms a ring and a chiral center by the 10-th step. The molecule then evolves slowly until it converges to a non-novel molecule by 200 steps. Further generation steps yield a series of novel molecules once again.

Figure 5.15 shows the trajectory of a ChEMBL molecule with a 1% masking rate. In the first step, the molecule changes from one training molecule to another non-novel molecule, following which it undergoes minor changes over the next few steps to yield a novel molecule. Figure 5.16 shows the trajectory of a ChEMBL molecule with a 5% masking rate. In the first step, this molecule also changes from one training molecule to another non-novel molecule. Following this, further changes yield a novel molecule. The molecule evolves again in further iterations, albeit forming unexpected ring structures after 300 steps.

From these observations, we see that molecules converge towards the space of dataset molecules regardless of whether training or marginal initialization is used. This implies that the sampler produces molecules from the distribution that it was trained on. We also see that using a higher masking rate results in greater changes between sampling iterations and molecules that are less similar to the dataset used. We hypothesize that this is the case for two reasons. First, a greater proportion of the graph is updated at each step. Second, the predictive distributions are formed from a graph with a greater proportion of masked components, resulting in higher entropy.

5.6 Related Work

In-Silico Molecular Generation Many of the previously proposed generative models of molecules focused on extending the variational autoencoder (VAE) for molecular generation. [Gómez-Bombarelli et al., 2016] proposed the first variational autoencoder (VAE; [Kingma and Welling, 2014]) based model for generating molecules in their SMILES representations. To address the issue of VAEs generating syntactically invalid SMILES strings, [Kusner et al., 2017] explicitly added the grammar of SMILES strings to VAEs for molecule generation. [Simonovsky and Komodakis, 2018] proposed a graph VAE to generate graph representations of molecules. [Jin et al., 2018] proposed using a VAE to generate a junction tree followed by the generation of the molecule itself. [Kang and Cho, 2019] proposed a semi-supervised VAE trained on SMILES strings that performs joint molecular property prediction and molecule generation. [Mahmood and Hernández-Lobato, 2019] proposed a constrained optimization method in the latent space of a VAE for goal-directed generation. [Kwon et al., 2019] proposed a non-autoregressive graph variational autoencoder trained with additional learning objectives for molecular graph generation. In addition to the previous work on extending VAEs for molecule generation, [Wang et al., 2017a], [Guimaraes et al., 2017] and [Cao and Kipf, 2018] used a generative adversarial network (GAN; [Goodfellow et al., 2014]) to build a generative model of small molecular graphs. Unlike most recent work that has focused on neural network-based approaches, [Jensen, 2018] showed that genetic algorithms based on Monte Carlo Tree Search (MCTS) could be competitive on the task of molecular generation. There has been some work applying reinforcement learning objectives to the task of molecular graph generation [You et al., 2018a, Zhou et al., 2019, Simm et al., 2020], which is orthogonal to our model.

Generative Models of Graphs [Li et al., 2018a] proposed a deep generative model of graphs that predicts a sequence of transformation operations to generate a graph. [You et al., 2018b] proposed an RNN-based autoregressive generative model that generates components of a graph in breadth-first search (BFS) ordering. To speed up the autoregressive graph generation and improve scalability, [Liao et al., 2019] extended autoregressive models of graphs by adding blockwise parallel generation. [Dai et al., 2020] proposed an autoregressive generative model of graphs that utilizes sparsity to avoid generating the full adjacency matrix and generates novel graphs in log-linear time complexity. [Grover et al., 2019] proposed a VAE-based iterative generative model for small graphs. They restrict themselves to modeling only the graph structure, whereas we consider generating a full graph including node and edge features for molecule generation. [Liu et al., 2019a] proposed a graph neural network model based on normalizing flows for memory-efficient prediction and generation.

Masked Language Models Masked language models, such as BERT [Devlin et al., 2019], have been shown to bring significant improvements to a variety of discriminative language understanding tasks such as question answering [Rajpurkar et al., 2016, Rajpurkar et al., 2018] and natural language inference [Bowman et al., 2015, Williams et al., 2018]. [Wang and Cho, 2019], [Ghazvininejad et al., 2019] and [Mansimov et al., 2019] proposed ways to generate text directly from trained masked language models. [Wang and Cho, 2019] proposed the use of Gibbs sampling, and [Mansimov et al., 2019] proposed the use of adaptive Gibbs sampling approaches for effective text generation using masked language models. [Ghazvininejad et al., 2019] used conditional masked language models for parallel decoding in machine translation. They first predict all target words in parallel, and then repeatedly mask out and regenerate the subset of words that the model is least confident about for a fixed number of iterations. In parallel to the work investigating masked language models for text generation, [Welleck et al., 2019], [Stern et al., 2019] and [Gu et al., 2019a] proposed methods for non-monotonic sequential text generation. Although these methods could be applied for generating molecular graphs in flexible ordering, there has not been work empirically validating this. Due to the popularity of masked language models in natural language processing tasks, there has been recent work investigating a similar approach for learning graph representations. [Hu et al., 2019] investigated the transfer to downstream tasks of graph neural networks that were trained to predict the masked node and edge attributes of graphs. [Maziarka et al., 2020] proposed the molecule attention transformer architecture that was pretrained to predict masked input nodes and investigated its transfer to downstream property prediction tasks. Unlike our work, neither [Hu et al., 2019] nor [Maziarka et al., 2020] investigated ways of generating novel molecular graphs with their trained models.

5.7 Conclusion

In this work, we propose a masked graph model for molecular graphs. We show that we can sample novel molecular graphs from this model by iteratively sampling subsets of graph components. Our proposed approach models the conditional distribution of subsets of graph components given the rest of the graph, avoiding many of the previously proposed models' drawbacks such as expensive marginalization and fixing an ordering of variables.

We evaluate our approach on the GuacaMol distribution-learning benchmark on the QM9 and ChEMBL datasets. We find that the benchmark metrics are correlated with each other, so models and generation configurations with higher validity, KL-divergence and Fréchet ChemNet Distance scores usually have lower novelty scores. We observe that by varying generation hyperparameters, we can trade off these metrics more efficiently than with state-of-the-art baseline models. We show that overall our model outperforms baseline graph-based methods. We also observe that our model is comparable to SMILES-based approaches on both datasets, but underperforms the LSTM, Transformer Small and Transformer Regular SMILES-based autoregressive models on ChEMBL. We also establish the minimization of validation loss as a reasonable objective for improving generation quality. Finally, we examine molecule trajectories and observe convergence to molecules that are similar to those in the original datasets, indicating that our sampler converges to its target distribution.

Future avenues of work include adapting our model for goal-directed molecular generation and investigating the usefulness of representations learned by our model for downstream molecular property prediction tasks. As our approach is broadly applicable to generic graph structures, we also leave its application to non-molecular datasets to future work.

5.8 Since the chapter release

The chapter was released very close to the time this thesis was written. We hope that our analysis of evaluation metrics for molecular generation and our proposed masked graph model will help the practitioners in the future.

Chapter 6

Molecular Geometry Prediction using a Deep Generative Graph Neural Network

A molecule's geometry, also known as conformation, is one of a molecule's most important properties, determining the reactions it participates in, the bonds it forms, and the interactions it has with other molecules. Conventional conformation generation methods minimize hand-designed molecular force field energy functions that are often not well correlated with the true energy function of a molecule observed in nature. They generate geometrically diverse sets of conformations, some of which are very similar to the lowest-energy conformations and others of which are very different. In this chapter, we propose a conditional deep generative graph neural network that learns an energy function by directly learning to generate molecular conformations that are energetically favorable and more likely to be observed experimentally in data-driven manner. On three large-scale datasets containing small molecules, we show that our method generates a set of conformations that on average is far more likely to be close to the corresponding reference conformations than are those obtained from conventional force field methods. Our method maintains geometrical diversity by generating conformations that are not too similar to each other, and is also computationally faster. We also show that our method can be used to provide initial coordinates for conventional force field methods. On one of the evaluated datasets we show that this combination allows us to combine the best of both methods, yielding generated conformations that are on average close to reference conformations with some very similar to reference conformations.

6.1 Introduction

The three-dimensional (3-D) coordinates of atoms in a molecule are commonly referred to as the molecule's geometry or **conformation**. The task, known as conformation generation, of predicting possible valid coordinates of a molecule, is important for determining a molecule's chemical and physical properties [Hawkins, 2017]. Conformation generation is also a vital part of applications such as generating 3-D quantitative structure-activity relationships (QSAR), structure-based virtual screening and pharmacophore modeling [Schwab, 2010]. Conformations can be determined in a physical setting using instrumental techniques such as X-ray crystallography as well as using experimental techniques. However, these methods are typically time-consuming and costly.

A number of computational methods have been developed for conformation generation over the past few decades [Schwab, 2010]. Typically this problem is approached by using a force field energy function to calculate a molecule's energy, and then minimizing this energy with respect to the molecule's coordinates. This hand-designed energy function yields an approximation of the molecule's true potential energy observed in nature based on the molecule's atoms, bonds and coordinates. The minimum of this energy function corresponds to the molecule's most stable configuration. Although this approach has been commonly used to generate a geometrically diverse set of conformations with certain conformations being similar to the lowest-energy conformations, it has been shown that molecule force field energy functions are often a crude approximation of actual molecular energy [Kanal et al., 2017].

In this chapter, we propose a deep generative graph neural network that learns the energy function from data in an end-to-end fashion by generating molecular conformations that are energetically favorable and more likely to be observed experimentally. This is done by maximizing the likelihood of the reference conformations of the molecules in the dataset. We evaluate and compare our method with conventional molecular force field methods on three databases of small molecules by calculating the root-mean-square deviation (RMSD) between generated and reference conformations. We show that conformations generated by our model are on average far more likely to be close to the reference conformation compared to those generated by conventional force field methods *i.e.* the variance of the RMSD between generated and reference conformations is lower for our method. Despite having lower variance, we show that our method does not generate geometrically similar conformations. We also show that our approach is computationally faster than force field methods.

A disadvantage of our model is that in general for a given molecule, the best conformation generated by our model lies further away from the reference conformation compared to the best conformation generated by force field methods. We show that for the QM9 small molecule dataset, the best of both methods can be combined by using the conformations generated by the deep generative graph neural network as an initialization to the force field method.

6.2 Conformation Generation

We consider a molecule as an undirected, complete graph $G = (V, E)$, where V is a set of vertices corresponding to atoms, and E is a set of edges representing the interactions between pairs of atoms from V. Each atom is represented as a vector $v_i \in \mathbb{R}^{d_v}$ of node features, and the edge between the i-th and j-th atoms is represented as a vector $e_{ij} \in \mathbb{R}^{d_e}$ of edge features. There are M vertices and $M(M-1)/2$ edges. We define a plausible conformation as one that may correspond to a stable configuration of a molecule. Given the graph of a molecule, the task of molecular geometry prediction is the generation of a set of plausible conformations $X_a = (x_1^a, \dots, x_M^a)$, where $x_i^a \in \mathbb{R}^3$ is a vector of the 3-D coordinates of the i-th atom in the a-th conformation.

Molecules can transition between conformations and end up in different local minima based on the stability of the respective conformations and environmental conditions. As a result, there is more than one plausible conformation associated with each molecule; it is hence natural to formulate conformation generation as finding (local) minima of an energy function $\mathcal{F}(X, G)$ defined on a pair of molecule graph and conformation:

$$\{X_1, \dots, X_S\} = \arg\min_X \mathcal{F}(X, G). \tag{6.1}$$

Alternatively, we could sample from a Gibbs distribution:

$$\{X_1, \dots, X_S\} \sim p_{\mathcal{F}}(X|G), \tag{6.2}$$

where

$$p_{\mathcal{F}}(X|G) = \frac{1}{\zeta(G)} \exp\left\{-\mathcal{F}(X, G)\right\}, \tag{6.3}$$

where ζ is a normalizing constant. We use S to indicate the number of conformations we generate for each molecule.

Under this view, the problem of conformation generation is decomposed into two stages. In the first stage, a computationally-efficient energy function $\mathcal{F}(X, G)$ is constructed. The second stage involves either performing optimization as in Eq. (6.1) or sampling as in Eq. (6.2) to generate a set of conformations from this energy function.

6.2.1 Energy Function Construction

A conventional approach is to define an energy function semi-automatically. The functional form of an energy function is designed carefully to incorporate various chemical properties, whereas detailed parameters of the energy function are either computationally or experimentally estimated. Two examples of widely used energy functions are the Universal Force Field (UFF) [Rappé et al., 1992] and the Merck Molecular Force Field (MMFF) [Halgren, 1996]. In contrast to these methods, here we will describe how to estimate the energy function or probability distribution directly from data using the latest techniques from deep learning.

6.2.2 Energy Minimization/Sampling

Once the energy function is defined, a conventional approach is to run the minimization many times starting from different initial conformations. Due to the non-convexity of the energy function, each run is likely to end up in a unique local minimum, allowing us to collect a set of many conformations.

A typical approach is to use distance geometry (DG) [Blaney and Dixon, 1994] or its variants, such as experimental-torsion basic knowledge distance geometry (ETKDG), [Riniker and Landrum, 2015] to randomly generate an initial conformation that satisfies various geometric constraints such as lower and upper bounds on the distances between atoms. Starting from the initial conformation, an iterative optimization algorithm, such as L-BFGS, [Liu and Nocedal, 1989] gradually updates the conformation until it finds a minimum of the energy function. In this chapter, we instead propose an approach based on deep generative models that allow us to sample directly from a distribution over all possible conformations given a molecule graph.

6.3 Deep Generative Model for Molecular Geometry

We propose to "learn" an energy function $\mathcal{F}(G, X)$ from a database containing many pairs of a molecule and its experimentally obtained conformation. Let $\mathcal{D} = \{(G_1, X_1^*), \ldots, (G_N, X_N^*)\}$ be a set of examples from such a database, where X_n^* is "a" reference conformation, often obtained and verified empirically in a certain environment. These reference conformations may not necessarily correspond to the lowest energy configurations of the molecules, but are energetically favorable and more likely to be observed experimentally. Learning an energy function can then be expressed as the following optimization problem:

$$\hat{\mathcal{F}}(G, X) = \arg\max_{\mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \underbrace{\log p_{\mathcal{F}}(X_n^* | G_n)}_{\text{(a)}}, \tag{6.4}$$

where $p_{\mathcal{F}}$ is a Gibbs distribution defined using $\mathcal{F}$ as in Eq. (6.3). In other words, we can learn the energy function $\mathcal{F}$ by maximizing the log-likelihood of the data D. In principle, the term "energy" has a very specific meaning in each context (e.g., potential energy, statistical free energy and etc). In our case, "energy" refers to an objective function that reflects the likelihood of a conformation given a molecular graph.

6.3.1 Conditional Variational Graph Autoencoders

We use a conditional version of a variational autoencoder [Kingma and Welling, 2014] to model the distribution $p_{\mathcal{F}}$ in Eq. (6.4) (a). This choice enables an underlying model to capture the complicated, multi-modal nature of this distribution, while allowing us to efficiently sample from this distribution. This is done by introducing a set of latent variables $Z = \{z_1, \ldots, z_M\}$, where $z_m \in \mathbb{R}^{d_z}$ and rewriting the conditional log-probability $\log p_{\mathcal{F}}(X|G)$ as

$$\log p(X|G) = \log \int p(X|Z, G)p(Z|G)\mathrm{d}Z, \tag{6.5}$$

where we omit the subscript $\mathcal{F}$ for brevity.

The marginal log-probability in Eq. (6.5) is generally intractable to compute, and we instead maximize the stochastic approximation to its lower bound, as is standard practice in problems involving variational inference:

$$\begin{aligned}\log p(X|G) \geq & \mathbb{E}_{Z\sim Q(Z|G,X)}[\log \underbrace{p(X|Z, G)}_{\text{(b) likelihood}}] - \mathrm{KL}(\underbrace{Q(Z|G, X)}_{\text{(c) posterior}} \parallel \underbrace{P(Z|G)}_{\text{(a) prior}}) & (6.6)\\ \approx & \frac{1}{K}\sum_{k=1}^{K} \log p(X|Z^k, G) - \mathrm{KL}(Q(Z|G, X)\|P(Z|G)), & (6.7)\end{aligned}$$

where Z^k is the k-th sample from the (approximate) posterior distribution Q above. We assume that we can compute the KL divergence analytically, for instance by constructing Q and P to be normal distributions.

6.3.1.1 Modeling the Graph using a Message Passing Neural Network

We use a message passing neural network (MPNN) [Gilmer et al., 2017], a variant of a graph neural network, [Scarselli et al., 2009, Bruna et al., 2014] which operates on a graph G directly and is invariant to graph isomorphism. The MPNN consists of L layers. At each layer l, we update the hidden vector $h(v_i) \in \mathbb{R}^{d_h}$ of each node and hidden matrix $h(e_{ij}) \in \mathbb{R}^{d_h \times d_h}$ of each edge using the equation

$$h^l(v_i) = \text{GRU}(h^{l-1}(v_i), J(h^{l-1}(v_i), h^{l-1}(v_{j\neq i}), h(e_{i,j\neq i})), \tag{6.8}$$

where J is a linear one layer neural network that aggregates the information from neighboring nodes according to its hidden vectors of respective nodes and edges. GRU is a gated recurrent network that combines the new aggregate information and its corresponding hidden vector from previous layer [Cho et al., 2014]. The weights of the message passing function J and GRU are shared across the L layers of the MPNN.

6.3.1.2 Prior Parameterization

We use the MPNN described above to model the prior distribution $P(Z|G)$ in Eq. (6.6) (a). We initialize $h^0(v_i)$ and $h(e_{ij})$ in Eq. (6.8) as linear transformations of the feature vectors v_i and e_{ij} of the nodes and edges respectively:

$$h^0(v_i) = U^{\text{prior}}_{\text{node}} v_i; \quad h(e_{ij}) = U^{\text{prior}}_{\text{edge}} e_{ij}, \tag{6.9}$$

where $U^{\text{prior}}_{\text{node}}$ and $U^{\text{prior}}_{\text{edge}}$ are matrices representing the linear transformations for the nodes and edges respectively. The final hidden vector $h^L(v_i)$ of each node is passed through a two layer neural network with hidden size d_f, whose output $\tilde{h}^L(v_i)$ is transformed into the mean and variance vectors of a Normal distribution with a diagonal covariance matrix:

$$\mu_i = W^{\text{prior}}_{\mu}\tilde{h}^L(v_i) + b^{\text{prior}}_{\mu}; \tag{6.10}$$

$$\sigma_i^2 = \exp\left\{W^{\text{prior}}_{\sigma}\tilde{h}^L(v_i) + b^{\text{prior}}_{\sigma}\right\}, \tag{6.11}$$

where W^{prior}_{μ} and $W^{\text{prior}}_{\sigma}$ are the weight matrices and b^{prior}_{μ} and $b^{\text{prior}}_{\sigma}$ are the bias terms of the transformations. These are used to form the prior distribution:

$$\log P(Z|G) = \sum_{i=1}^{N}\sum_{j=1}^{3} -\frac{(\mu_{i,j} - z_{i,j})^2}{2\sigma^2_{i,j}} - \log\sqrt{2\pi\sigma^2_{i,j}}, \tag{6.12}$$

where $\mu_{i,j}$ and $\sigma^2_{i,j}$ are the j-th components of the mean and variance vectors respectively. In other words, we parameterize the prior distribution as a factorized Normal distribution factored over the vertices and the dimensions in the 3-D coordinate.

6.3.1.3 Likelihood Parameterization

We use a similar MPNN to model the likelihood distribution, $P(X|Z, G)$ in Eq. (6.6) (b). The only difference is that this distribution is conditioned not only on the molecular graph $G = (V, E)$ but also on the latent set $Z = \{z_1, \ldots, z_M\}$. We incorporate the latent set Z by adding the linear transformation of the node feature vector v_i

to its corresponding latent variable z_i. This result is used to initialize the hidden vector:

$$h^0(v_i) = U_{\text{node}}^{\text{likelihood}} v_i + z_i; \quad h(e_{ij}) = U_{\text{edge}}^{\text{likelihood}} e_{ij}, \tag{6.13}$$

where $U_{\text{node}}^{\text{likelihood}}$ and $U_{\text{edge}}^{\text{likelihood}}$ are matrices representing the linear transformations for the nodes and edges respectively. From there on, we run neural message passing as in Eqs. (6.8–6.11), with a new set of parameters, $\theta_{\text{likelihood}}$, $W_{\mu}^{\text{likelihood}}$, $b_{\mu}^{\text{likelihood}}$, $W_{\sigma}^{\text{likelihood}}$ and $b_{\sigma}^{\text{likelihood}}$. The final mean and variance vectors are now three dimensional, representing the 3-D coordinates of each atom, and we can compute the log-probability of the coordinates using Eq. (6.12).

6.3.1.4 Posterior Parameterization

As computing the exact posterior $P(Z|G, X)$ is intractable, we resort to amortized inference using a parameterized, approximate posterior $Q(Z|G, X)$ in Eq. (6.6) (c). We use a similar approach to our parameterization of the prior distribution above. However, we replace the input to the MPNN with the concatenation of an edge feature vector e_{ij} and the corresponding distance (proximity) matrix $D(X^*)$ of the reference 3-D conformation X^*:

$$h(e_{ij}) = U_{\text{edge}}^{\text{posterior}} \begin{bmatrix} e_{ij} \\ D(x_i^*) \end{bmatrix}. \tag{6.14}$$

With a new set of parameters, $\theta_{\text{posterior}}$, $W_{\mu}^{\text{posterior}}$, $b_{\mu}^{\text{posterior}}$, $W_{\sigma}^{\text{posterior}}$ and $b_{\sigma}^{\text{posterior}}$, the MPNN outputs a Normal distribution for each latent variable z_i. Linear weight embeddings of nodes U_{node} are shared between prior, likelihood and posterior.

6.3.2 Training the Conditional Variational Graph Autoencoder

With the choice of the Gaussian latent variables z_i, we can use the reparameterization trick [Kingma and Welling, 2014] to compute the gradient of the stochastic approximation to the lower bound in Eq. (6.7) with respect to all the parameters of the three distributions. [Kingma and Welling, 2014] This property allows us to train this model on a large dataset using stochastic gradient descent (SGD). However, there are two major considerations that must be made before training this model on a large molecule database.

6.3.2.1 Post-Alignment Likelihood

An important property of conformation generation over a usual problem of regression is that we must take into account rotation and translation. Let R be an alignment function that takes as input a target conformation and a predicted conformation. The function aligns the reference conformation to the predicted conformation and returns the aligned reference conformation. $\hat{X} = R(X, X^*)$ is the conformation obtained by rotating and translating the reference conformation X^* to have the smallest distance to the predicted conformation X according to a predefined metric such as RMSD:

$$\mathrm{RMSD}(\hat{X}, X^*) = \sqrt{\frac{1}{M}\sum_{i=1}^{M} \|\hat{x}_i - x_i^*\|^2}. \tag{6.15}$$

This alignment function R is selected according to the problem at hand, and we present below its use in a general form without exact specification.

We implement this invariance to rotation and translation by parameterizing the output of the likelihood distribution above to be aligned to the target molecule. That is,

$$\log p(X|G,Z) = \sum_{i=1}^{M} \sum_{j=1}^{3} -\frac{(\mu_{i,j} - \hat{x}^*_{i,j})^2}{2\sigma^2_{i,j}} - \log \sqrt{2\pi\sigma^2_{i,j}}, \tag{6.16}$$

where $\hat{x}^*_i$ is the coordinate of the i-th atom aligned to the mean conformation $\{\mu_1, \ldots, \mu_N\}$. That is,

$$\{\hat{x}^*_1, \ldots, \hat{x}^*_M\} = R(\{\mu_1, \ldots, \mu_M\}, X^*). \tag{6.17}$$

In other words, we rotate and translate the reference conformation X^* to be best aligned to the predicted conformation (or its mean) before computing the log-probability. This encourages the model to assign high probability to a conformation that is easily aligned to the reference conformation X^*, which is precisely the goal of maximum log-likelihood.

6.3.2.2 Unconditional Prior Regularization

The second term in the lower bound in Eq. (6.6), which is the KL divergence between the approximate posterior and prior, does not have a point minimum but an infinitely long valley consisting of minimum values. Consider the KL divergence between two univariate Normal distributions:

$$\mathrm{KL}(\mathcal{N}(\mu_1, \sigma_1^2) \| \mathcal{N}(\mu_2, \sigma_2^2)) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \tag{6.18}$$

When both distributions are shifted by the same amount, the KL divergence remains unchanged. This could lead to a difficulty in optimization, as the means of the posterior and prior distributions could both diverge.

In order to prevent this pathological behavior, we introduce an unconditional prior distribution $P(Z)$ which is a factorized Normal distribution:

$$P(Z) = \prod_{i=1}^{M} \mathcal{N}(z_i|0, I), \tag{6.19}$$

where $\mathcal{N}$ computes a Normal probability density, and I is a $d_z \times d_z$ identity matrix. We minimize the KL divergence between the original prior distribution $P(Z|G)$ and this unconditional prior distribution $P(Z)$ in addition to maximizing the lowerbound, leading to the following final objective function for each molecule:

$$\mathcal{L} = \log p(X|Z^1, G) - \mathrm{KL}(Q(Z|G, X)\|P(Z|G)) - \alpha \cdot \mathrm{KL}(P(Z|G)\|P(Z)), \tag{6.20}$$

where we assume $K = 1$ and introduce a coefficient $\alpha \geq 0$.

6.3.3 Inference: Predicting Molecular Geometry

Learning a conditional variational autoencoder above corresponds to the first stage of conformation generation, that is, the stage of energy function construction. Once the energy function is constructed, we need to sample multiple conformations from the Gibbs distribution defined using the energy function, which is $\log P(X|G)$ in Eq. (6.5). Our parameterization of the Gibbs distribution using a directed graphical model [Pearl, 1986] allows us to efficiently sample from this distribution. We first sample from the prior distribution, $\tilde{Z} \sim P(Z|G)$, and then sample from the likelihood distribution, $\tilde{X} \sim P(X|\tilde{Z}, G)$. In practice, we fix the output variance $\sigma_{i,j}$ of the likelihood distribution to be 1 and take the mean set $\{\mu_1, \ldots, \mu_M\}$ as a sample from the model.

6.4 Experimental Setup

6.4.1 Data

We experimentally verify the effectiveness of the proposed approach using three databases of molecules: QM9 [Ruddigkeit et al., 2012, Ramakrishnan et al., 2014], COD [Gražulis et al., 2012] and CSD [Groom et al., 2016]. These datasets are selected as they possess distinct properties from each other, which allows us to carefully study various aspects of the proposed approach. There is an overlap between COD and CSD databases, since both of these databases were based on published crystallography data. We only keep molecules from each database that can be processed by RDKit[1]. We further remove disconnected compounds *i.e.* those whose Simplified Molecular-Input Line-Entry System [Weininger, 1988] (SMILES) representation contains '.'. See Fig. 6.1 for some other properties of these three datasets.

6.4.1.1 QM9

The filtered QM9 dataset contains 133,015 molecules, each of which contains up to 9 heavy atoms of types `C`, `N`, `O` and `F`. Each molecule is paired with a reference conformation obtained by optimizing the molecular geometry with density functional theory (DFT) at the B3LYP/6-31G(2df,p) level of theory, which implies that the reference conformations are obtained from the same environment. We hold out separate 5,000 and 5,000 randomly selected molecules as validation and test sets, respectively.

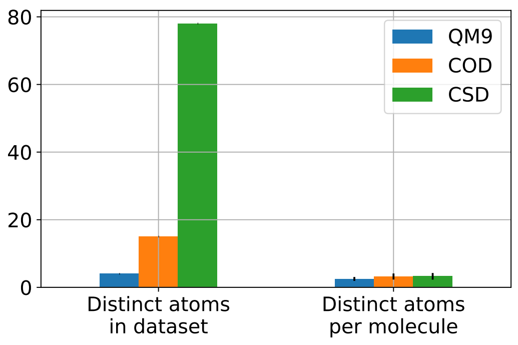
[1]Version 2018.09.1

6.4.1.2 COD

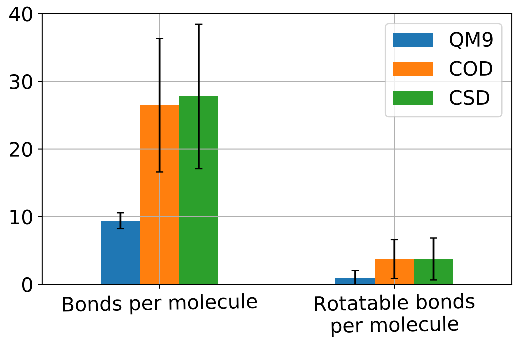
We use the organic part of the COD dataset. We further filter out any molecule that contains more than 50 heavy atoms of types B, C, N, O, F, Si, P, S, Cl, Ge, As, Se, Br, Te and I. This results in 66,663 molecules, out of which we hold out separate 3,000 and 3,000 randomly selected ones respectively for validation and test purposes. Reference conformations are voluntarily contributed to the dataset and are often determined either experimentally or by DFT calculations. [Hautier et al., 2012] Thus, the reference conformations are obtained from different environments.

6.4.1.3 CSD

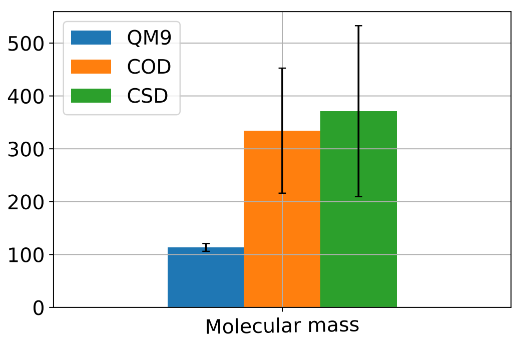
Similarly to COD, we remove any molecule that contains more than 50 heavy atoms, resulting in a total of 236,985 molecules. We hold out separate 3,000 and 3,000 randomly selected molecules for validation and test purposes respectively. This dataset contains organic and metal-organic crystallographic structures which have been observed experimentally [Groom et al., 2016]. The atom types in this dataset are S, N, P, Be, Tc, Xe, Br, Rh, Os, Zr, In, As, Mo, Dy, Nb, La, Te, Th, Ga, Tl, Y, Cr, F, Fe, Sb, Yb, Tb, Pu, Am, Re, Eu, Hg, Mn, Lu, Nd, Ce, Ge, Sc, Gd, Ca, Ti, Sn, Ir, Al, K, Tm, Ni, Er, Co, Bi, Pr, Rb, Sm, O, Pt, Hf, Se, Np, Cd, Pd, Pb, Ho, Ag, Mg, Zn, Ta, V, B, Ru, W, Cl, Au, U, Si, Li, C and I. The reference conformations are obtained from crystal structures.



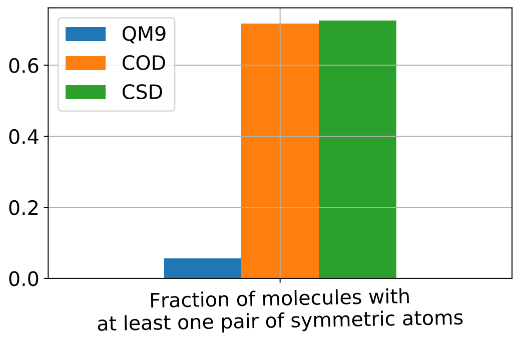
(a) Diversity of atoms overall and per molecule



(b) Number and rotatability of bonds per molecule



(c) Molecular mass per molecule



(d) Proportion of molecules with symmetry

Figure 6.1: Dataset Characteristics: information regarding the atoms, bonds, molecular mass and symmetry of molecules in each dataset.

6.4.2 Models

6.4.2.1 Baselines

As a point of reference, we minimize a force field starting from a conformation created using ETKDG. [Riniker and Landrum, 2015] We test both UFF and MMFF, and respectively call the resulting approaches **ETKDG+UFF** and **ETKDG+MMFF**. The environment from which each conformation is obtained affects the force field calculations. To keep comparisons fair and to abstract away the effects of the environment, we use the implementations in RDKit [Landrum,] with the default hyperparameters. The default implementations have often been used in literature when comparing different conformation generation methods. [Sadowski and Baldi, 2013, Ebejer et al., 2012, Friedrich et al., 2017]

6.4.2.2 Conditional Variational Graph Autoencoder

We build one conditional variational graph autoencoder for each dataset. We use $d_h = 50$ hidden units at each layer of neural message passing (Eq. 6.8) in each of the three MPNNs corresponding to the prior, likelihood and posterior distributions. We use $d_f = 100$ in the two layer neural network that comes after the MPNN. As described earlier, we fix the variance of the output in the likelihood distribution to 1. We use $L = 3$ layers per network for QM9 and $L = 5$ layers per network for COD and CSD. We chose these hyperparameter values by carrying out a grid-search and choosing the values that had the best performance on the validation set. The grid-search procedure and the performance of models with different hyperparameters are shown in the supplementary information.

6.4.2.3 Learning

For all models, we use dropout [Srivastava et al., 2014] at each layer of the neural network that comes after the MPNN with a dropout rate of 0.2 to regularize learning. We set the coefficient α in Eq. (6.20) to 10^{-5}. We train each model using Adam [Kingma and Ba, 2014] with a fixed learning rate of 3×10^{-4}. All models were trained with a batch size of 20 molecules on 1 Nvidia GPU with 12 GB of RAM.

6.4.2.4 Inference

There are two modes of inference with the proposed approach. The first approach is to sample from a trained conditional variational graph autoencoder by first sampling from the prior distribution and taking the mean vectors from the likelihood distribution; we refer to this as **CVGAE**. We can then use these samples further as initializations of MMFF minimization; we refer to this as **CVGAE+MMFF**. The latter approach can be thought of as a trainable approach to initializing a conformation in place of DG or ETKDG.

6.4.3 Evaluation

In principle, the quality of the sampled conformations should be evaluated based on their molecular energies, for instance by DFT, which is often more accurate than force field methods [Kanal et al., 2017]. However, the computational complexity of the DFT calculation is superlinear with respect to the number of electrons in a molecule, and so is often impractical [Ratcliff et al., 2017]. Instead, we follow prior work on conformation generation [Hawkins, 2017] and evaluate the baselines and proposed method using the RMSD (Eq. 6.15) of the heavy atoms between a reference conformation and a predicted conformation which is fast and simple to calculate.

6.5 Results

When evaluating each method, we first sample 100 conformations per molecule for each method in the test set. We can make several observations from Table 6.1. First, compared to other methods, our proposed CVGAE always succeeds at generating the specified number of conformations for any of the molecules in the test set. UFF and

Table 6.1: Number of successfully processed molecules in the test set (success per test set $\uparrow$), number of successfully generated conformations out of 100 (success per molecule $\uparrow$), median of mean RMSD (mean $\downarrow$), median of standard deviation of RMSD (std. dev. $\downarrow$) and median of best RMSD (best $\downarrow$) per molecule on QM9, COD and CSD datasets. ETKDG stands for Distance Geometry with experimental torsion-angle preferences. UFF and MMFF are force field methods and stand for Universal Force Field and Molecular Mechanics Force Field respectively. CVGAE stands for Conditional Variational Graph Autoencoder. CVGAE + Force Field represents running the MMFF force field optimization initialized by CVGAE predictions.

		ETKDG + Force Field		CVGAE	CVGAE + Force Field
Dataset		UFF	MMFF		MMFF
QM9	success per test set	96.440%	96.440%	100%	99.760%
	success per molecule	98.725%	98.725%	100%	98.684%
	mean	0.425	0.415	0.390	0.367
	std. dev.	0.176	0.189	0.017	0.074
	best	0.126	0.092	0.325	0.115
COD	success per test set	99.133%	99.133%	100%	95.367%
	success per molecule	99.627%	99.627%	100%	99.071%
	mean	1.389	1.358	1.331	1.656
	std. dev.	0.407	0.415	0.099	0.425
	best	0.429	0.393	1.206	0.635
CSD	success per test set	97.400%	97.400%	100%	99.467%
	success per molecule	99.130%	99.130%	100%	97.967%
	mean	1.537	1.488	1.506	1.833
	std. dev.	0.421	0.418	0.115	0.434
	best	0.508	0.478	1.343	0.784

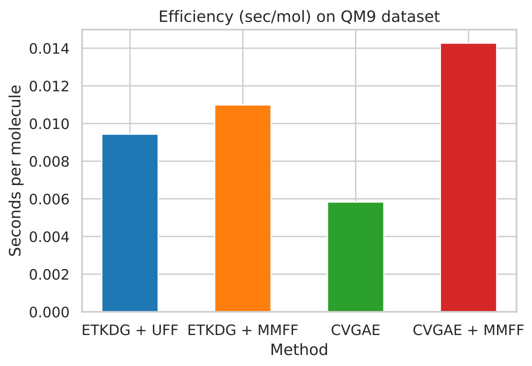
MMFF fail to generate conformations for some molecules, as they do not support handling every element but the pre-defined sets of elements. Since all other evaluated approaches were unsuccessful at generating at least one conformation for a very small number of test molecules, we report results for the molecules for which all evaluated methods generated at least one conformation. We report the *median* of the mean of the RMSD, the *median* of the standard deviation of the RMSD and the *median* of the best (lowest) RMSD among all generated conformations for each test molecule. Across all three datasets, every evaluated method achieves roughly the same median of the mean

Table 6.2: Conformation Diversity. Mean and std. dev. represents the corresponding mean and standard deviation of pairwise RMSD between at most 100 generated conformations per molecule.

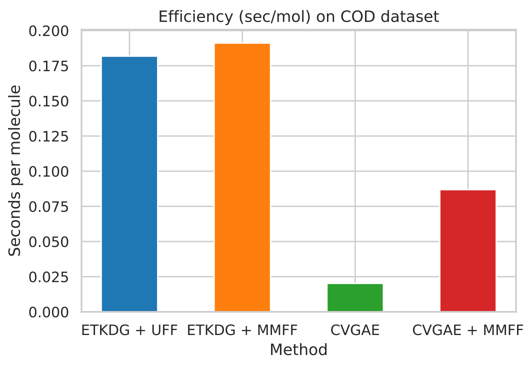
Dataset		ETKDG + MMFF	CVGAE	CVGAE + MMFF
QM9	mean	0.400	0.106	0.238
	std. dev.	0.254	0.061	0.209
COD	mean	1.148	0.239	1.619
	std. dev.	0.699	0.181	0.537
CSD	mean	1.244	0.567	1.665
	std. dev.	0.733	0.339	0.177

RMSD. More importantly, the standard deviation of the RMSD achieved by CVGAE is *significantly* lower than that achieved by ETKDG + Force Field. After the initial generation stage, conformations are usually further evaluated and optimized by running the computationally expensive DFT optimization. Reducing the standard deviation can lower the number of conformations on which DFT optimization has to be run in order to achieve a valid conformation. On the other hand, the best RMSD achieved by ETKDG + UFF/MMFF methods is lower than that achieved by CVGAE. Using MMFF initialized by CVGAE (CVGAE + MMFF) instead of ETKDG (ETKDG + MMFF) improves the mean results on the QM9 dataset for CVGAE, and yields a lower standard deviation and similar best RMSD compared to ETKDG + MMFF. Unfortunately, CVGAE + MMFF worsens the results achieved by CVGAE alone on the COD and CSD datasets. We additionally evaluate single point DFT energy for the subset of 1000 molecules in the QM9 test set for all 100 generated conformations. We find that all three methods ETKDG + MMFF, CVGAE and CVGAE + MMFF achieve similar median energy values of -411.52, -410.87 and -411.50 respectively. The energy was calculated using GAMESS software [Schmidt et al., 1993] with default parameters.

We also report the diversity of conformations generated by all evaluated methods



(a) QM9 dataset



(b) COD dataset

Figure 6.2: Computational efficiency of various approaches on QM9 and COD datasets

in Table 6.2. Diversity is measured by calculating the mean and standard deviation of the pairwise RMSD between each pair of generated conformations per molecule. Overall, we can see that despite having a smaller median of standard deviation of RMSD between generated conformations and reference conformations, CVGAE does not collapse to generating extremely similar conformations. Although, CVGAE generates relatively less diverse samples compared to ETKDG + MMFF baseline on all datasets. The conformations of molecules generated by CVGAE + MMFF are less diverse on the QM9 dataset and more diverse on COD/CSD datasets compared to ETKDG + MMFF baseline.

The computational efficiency of each of the evaluated approaches on the QM9 and COD datasets is shown in Figure 6.2. For consistency, we generated one conformation for one molecule at a time using each of the evaluated methods on an Intel(R) Xeon(R) E5-2650 v4 CPU. On the QM9 dataset, CVGAE is $2\times$ more efficient than ETKDG + UFF/MMFF, while CVGAE + MMFF is slightly slower than ETKDG + UFF/MMFF. On the COD dataset, which contains a larger number of atoms per molecule, CVGAE is almost $10\times$ as fast as ETKDG + UFF/MMFF, while CVGAE + MMFF is about $2\times$ as fast as ETKDG + UFF/MMFF. This shows that CVGAE scales much better than the baseline ETKDG + UFF/MMFF methods as the size of the molecule grows.

Figures 6.3 and 6.4 visualize the median, standard deviation and best RMSD results as a function of the number of heavy atoms in a molecule on the QM9 and COD/CSD datasets respectively. For all approaches, we can see that the best and median RMSD both increase with the number of heavy atoms. The standard deviation of the median RMSD for CVGAE and CVGAE + MMFF is lower than that for ETKDG + MMFF across molecules of almost all sizes. The standard deviation of the best RMSD is slightly higher for CVGAE and CVGAE + MMFF than for ETKDG + MMFF on molecules with at most 12 atoms, but is lower for larger molecules, particularly for CVGAE. Overall, CVGAE yields a lower or similar median RMSD compared to ETKDG + MMFF across molecules of all sizes and a lower standard deviation, whereas ETKDG + MMFF provides a lower best RMSD particularly for larger molecules observed in the COD/CSD datasets.

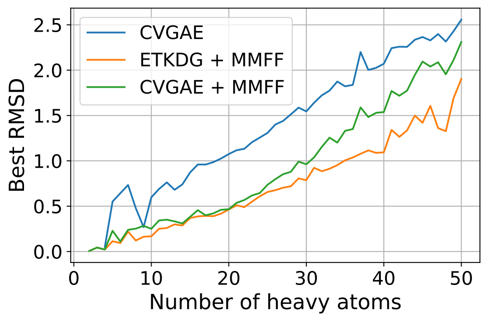
Figures 6.5 and 6.6 qualitatively compare the results of CVGAE against MMFF and CVGAE + MMFF against CVGAE respectively. For each dataset, each figure shows the three molecules for which the first method in each figure outperforms the second method by the greatest amount, and the three molecules for which the second method outperforms the first by the greatest amount. The reference molecules are shown alongside the conformations resulting from each of the methods for comparison.

We can see some general trends from both these figures. The conformations produced by the neural network are qualitatively much more similar to the reference in the case of the QM9 dataset than in the cases of the COD and CSD datasets. In the case of the COD and CSD datasets, the CVGAE predictions appear to be squashed or compressed in comparison to the reference molecules. For example, in almost every case we can see the absence of visible rings and the absence of bonds protruding from the lengthwise dimension of the molecule. At the same time we can see that on COD and CSD, CVGAE does better than ETKDG + MMFF in cases where ETKDG + MMFF creates loops and protrusions in the wrong places.

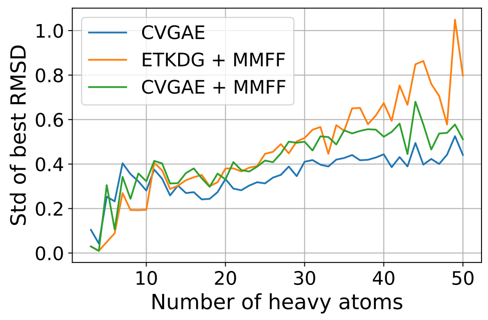
6.6 Analysis and Future Work

Overall we observe that CVGAE performs better than ETKDG + MMFF on QM9 than on COD and CSD. One possible reason that could explain this phenomenon is that COD and CSD contain much larger number of heavy atoms per molecule than QM9. In the absence of adequate number of neural message passing steps and adequate number of hidden units, the network may converge to outputting a conformation that contains atoms largely along a single non-linear dimension in order to minimize outliers, which would be heavily penalized by the sum of squared distances term in the loss function. A neural network architecture with a larger number of neural message passing steps and larger number of hidden units may be needed to generate less conservative conformations and achieve comparable results to those for QM9. This is a recommended direction of future work that will require more computational resources, including distributed training on multiple GPUs with sufficient memory.

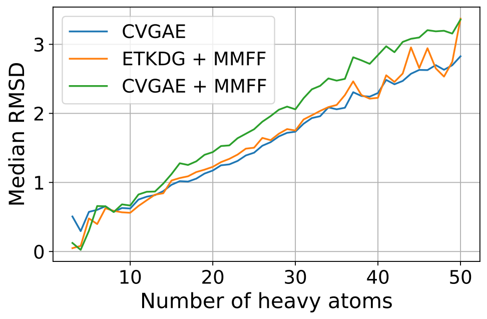
Another concern for COD and CSD is the inconsistency in the environments from which the reference conformations are obtained. The inconsistency would not be a serious concern for small molecules, but it can result in performance degradation with larger molecules. Further investigation should be performed with the dataset of larger molecules and their reference conformations whose corresponding environments are identical. Additionally, conditioning deep generative graph neural networks on the environment could be explored in the future.



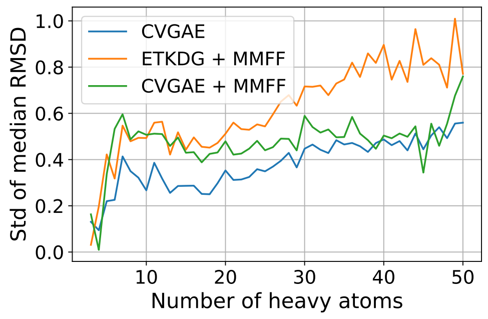
Mean of best RMSD



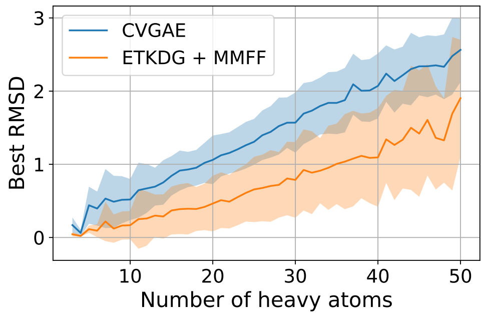
St. dev. of best RMSD



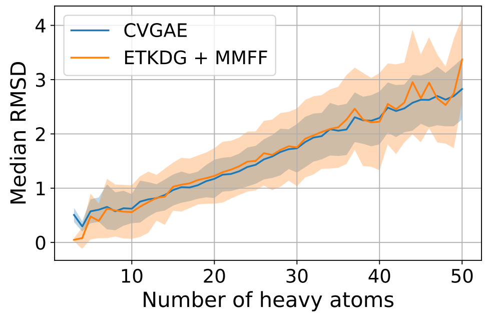
Mean of median RMSD



St. dev. of median RMSD

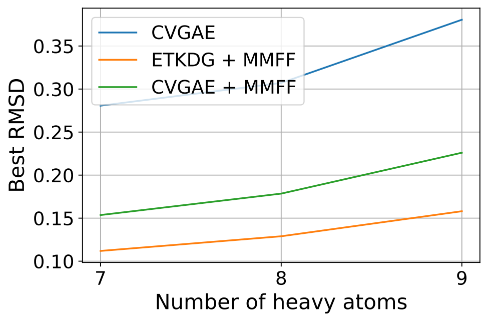


Best RMSD with uncertainty bounds

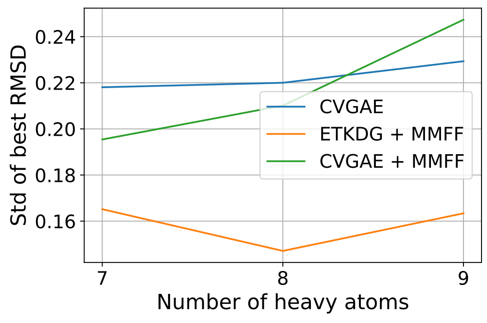


Median RMSD with uncertainty bounds

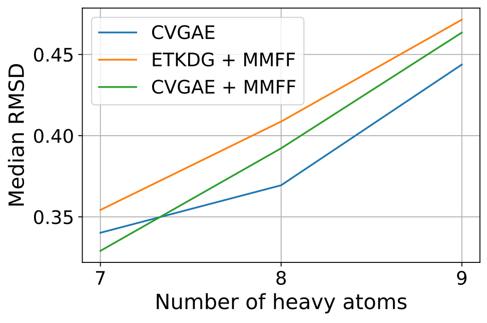
Figure 6.3: This figure shows the means and standard deviations of the best and median RMSDs on the union of COD and CSD datasets as a function of number of heavy atoms. The molecules were grouped by number of heavy atoms, and the mean and standard deviation of the median and best RMSDs were calculated for each group to obtain these plots. Groups at the left hand side of the graph with less than 1% of the mean number of molecules per group were omitted.



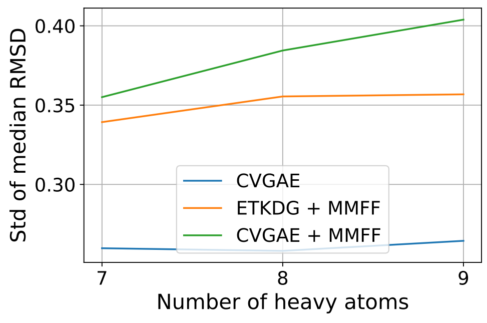
Mean of best RMSD



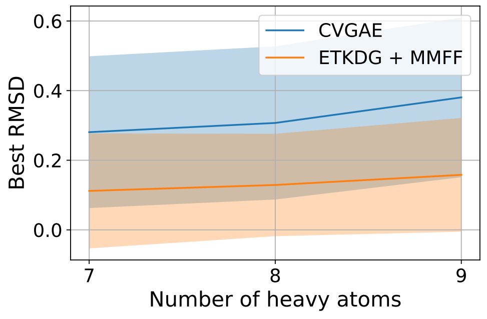
St. dev. of best RMSD



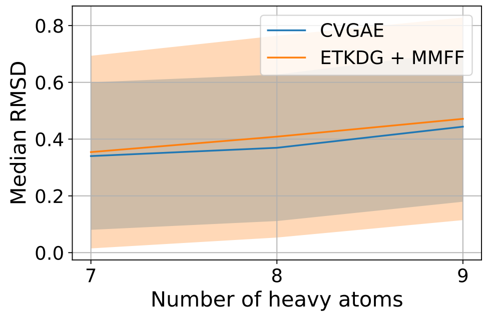
Mean of median RMSD



St. dev. of median RMSD

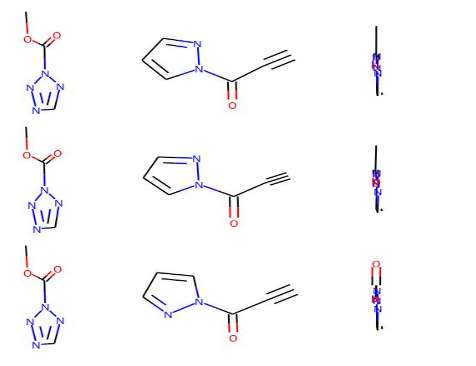


Best RMSD with uncertainty bounds

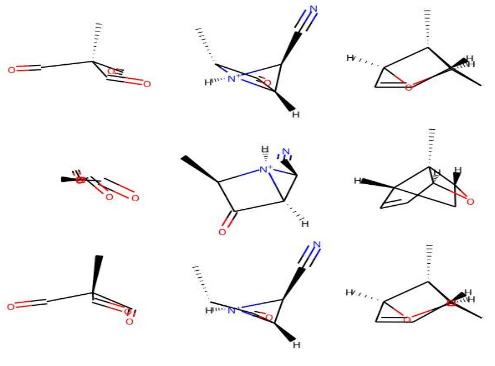


Median RMSD with uncertainty bounds

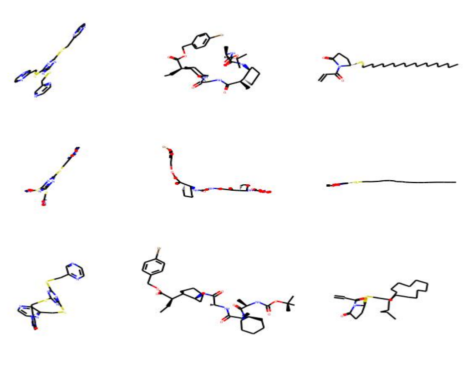
Figure 6.4: This figure shows the means and standard deviations of the best and median RMSDs on the QM9 dataset as a function of number of heavy atoms. The molecules were grouped by number of heavy atoms, and the mean and standard deviation of the median and best RMSDs were calculated for each group to obtain these plots. Groups at the left hand side of the graph with less than 1% of the mean number of molecules per group were omitted.



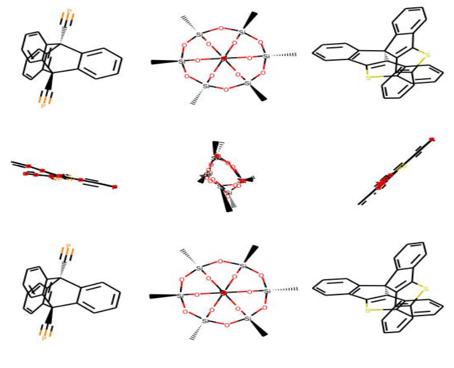
QM9 greatest difference in favour of neural network predictions



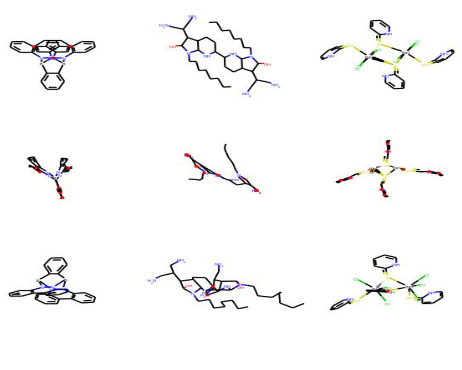
QM9 greatest difference in favour of ETKDG + MMFF predictions



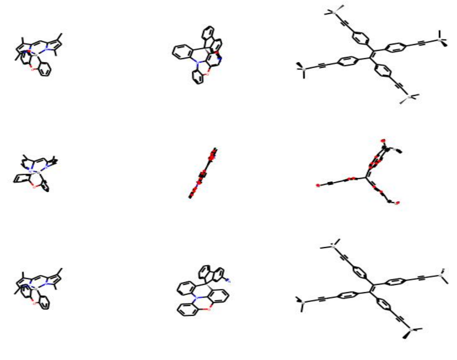
COD greatest difference in favour of neural network predictions



COD greatest difference in favour of ETKDG + MMFF predictions



CSD greatest difference in favour of neural network predictions



CSD greatest difference in favour of ETKDG + MMFF predictions

Figure 6.5: This figure shows the three molecules in each dataset for which the differences between the RMSDs of the neural network predictions and the baseline ETKDG + MMFF predictions were greatest in favour of the neural network predictions ($max\ (RMSD_{CVGAE} - RMSD_{ETKDG+MMFF})$), and the three for which this difference was greatest in favour of the ETKDG + MMFF predictions ($max\ (RMSD_{ETKDG+MMFF} - RMSD_{CVGAE})$). The top row of each subfigure contains the reference molecules, the middle row contains the neural network predictions and the bottom row contains the conformations generated by applying MMFF to the reference conformations.

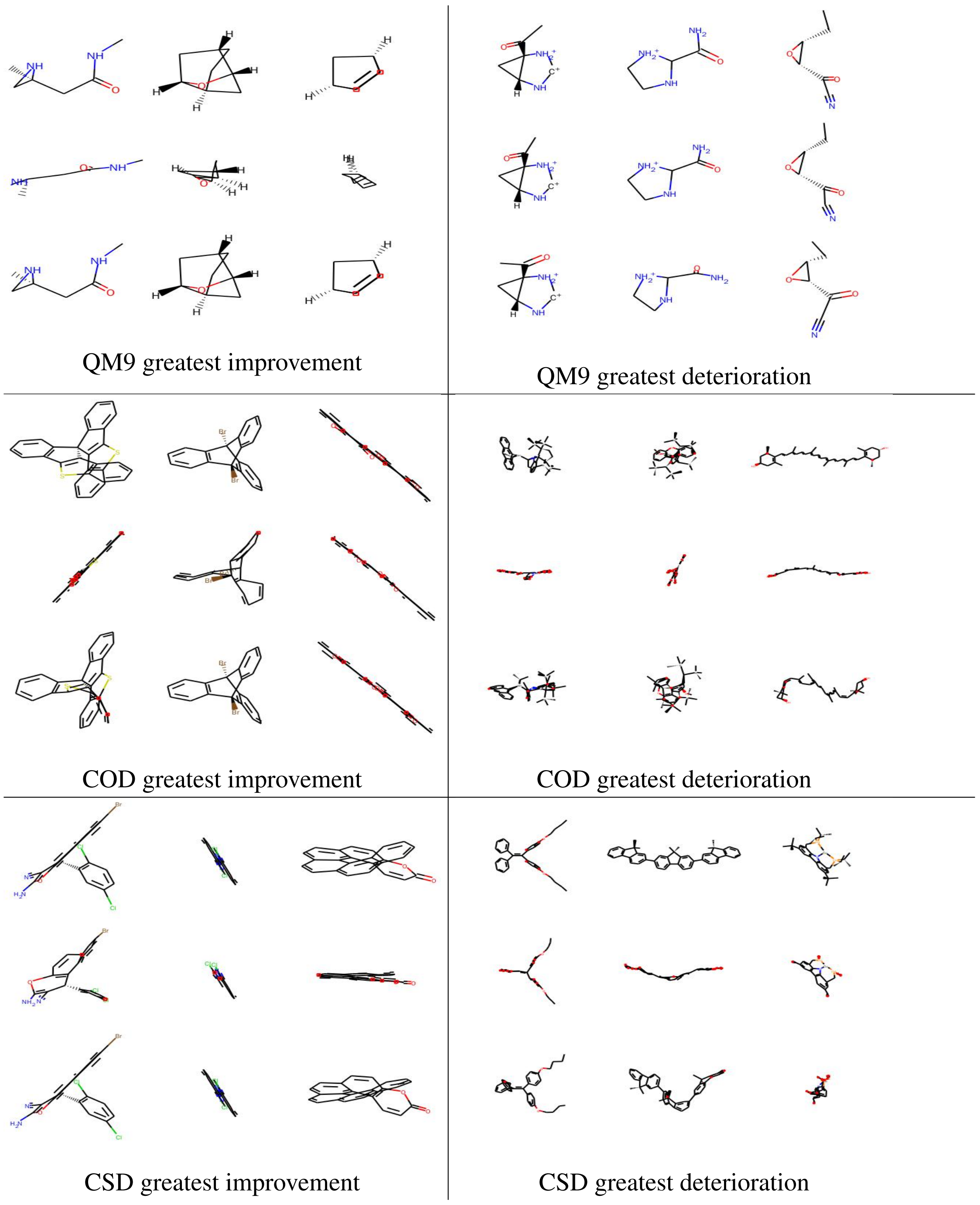

QM9 greatest improvement
QM9 greatest deterioration
COD greatest improvement
COD greatest deterioration
CSD greatest improvement
CSD greatest deterioration

Figure 6.6: This figure shows the three molecules in each dataset whose RMSD decreased the most and the three whose RMSD increased the most on applying MMFF to the conformations predicted by the neural network. The top row of each subfigure contains the reference molecules, the middle row contains the neural network predictions and the bottom row contains the conformations generated by applying MMFF to the neural network predictions.

We also observe that our CVGAE method has a lower variance than the baseline methods, so a relatively small number of samples needs to be taken before getting a conformation with a good RMSD. In addition, CVGAE is faster than force field methods and uses less computational resources once trained. Using conformations generated by CVGAE as an initialization to force field method showed promising results on the QM9 dataset that allowed to combine the best of two distinct methods. However, applying a force field method on the conformations generated by CVGAE leads to an increase in RMSD on the COD and CSD datasets - future work could explore why this is the case. Another avenue of future inquiry could be the joint training of CVGAE and a force field method, which would involve implementing force field minimization using a deep learning framework, connecting this to CVGAE and backpropagating through this aggregate model. This joint training could further yield better results than either method alone.

6.7 Since the chapter release

Since the release of our chapter, [Simm and Hernández-Lobato, 2019] proposed improvements to our model. They used the conditional variational graph autoencoder proposed in our chapter, with the main difference in predicting the distances between atoms instead of 3D coordinates. By predicting distances, aligning the ground-truth conformations to the predicted conformations becomes unnecessary.

Overall, the generation of 3D structures of small molecules using deep generative models is becoming more popular. Some of the papers on this topic released since our work include [Hoffmann et al., 2019, Gebauer et al., 2019, Axelrod and Gómez-Bombarelli, 2020]. Beyond small molecules investigated in our work, there has been a growing body of work examining the use of deep generative models to generate 3D structures of proteins [AlQuraishi, 2018, Ingraham et al., 2019b, Xu, 2019, Senior et al., 2020, Du et al., 2020], a process known as protein folding.

Chapter 7

Conclusion

This thesis has explored the structured prediction based on the principle of iterative refinement. In chapter 3, we introduced the concept of iterative refinement based on the principle denoising autoencoder and latent variable modeling. In chapter 4, we proposed the generalized framework of sequence generation. Using the proposed framework, we derived novel generation procedures for undirected neural sequence model BERT [Devlin et al., 2019] and showed how we can iteratively generate novel translations using this model. In chapter 5 we introduced the masked graph modeling approach for iterative generation of molecular graph based on the framework introduced in chapter 4. Finally, in chapter 6 we proposed the conditional variational graph autoencoder for molecule conformation generation that is further refined by classical force-field methods. Overall, we showed that iterative refinement is a successful approach for applications such as machine translation, image captioning, molecular graph generation and molecular conformation generation.

While the iterative refinement approach is successful at structured prediction, neural autoregressive left-to-right monotonic sequence modeling remains a state-of-the-art approach for sequence generation. It is important to discuss why we would want to use the iterative refinement for structured prediction in the first place, especially given that from a pure performance standpoint, autoregressive left-to-right monotonic models are incredibly effective.

From the application standpoint, non-autoregressive models based on iterative refinement can provide a speed-up over autoregressive models at decoding time. The recent models based on the iterative refinement [Ghazvininejad et al., 2020, Saharia et al., 2020] match the performance of autoregressive models on machine translation using as little as 4 generation steps. Specifically, the conditional mask-predict non-autoregressive model using iterative refinement [Ghazvininejad et al., 2019] achieves 4x wall-clock time speed up over state-of-the-art autoregressive model when translating one sentence at a time [Kasai et al., 2020]. The decoding speed of both autoregressive and non-autoregressive models depends on several factors, such as software implementation, hardware, and details of neural network architecture. Upon closer inspection, [Kasai et al., 2020] found that the non-autoregressive model's speed-up over the autoregressive model only holds when translating one sentence at a time. In particular, [Kasai et al., 2020] found that non-autoregressive models translate slower than autoregressive models when using mini-batches as large as GPU hardware allows. Using large minibatch, autoregressive models better utilize GPU than non-autoregressive models, which offset non-autoregressive models' benefit. Furthermore, increasing the encoder's depth and decreasing the decoder's depth in an autoregressive model improves the decoding speed while maintaining the vanilla autoregressive model's performance. The paper by [Kasai et al., 2020] highlights that despite the speed-up of non-autoregressive machine translation models using iterative refinement over autoregressive models in a single-sentence setting, there is still room to strengthen the argument that non-autoregressive models are faster at decoding. In particular, further work on improving software implementation and architectural details of non-autoregressive models using iterative refinement can help achieve substantial speed-up over autoregressive models across various applications and settings.

From the logical standpoint, the iterative refinement approach searches the solution space differently compared to beam-search in autoregressive models. Indeed, beam-search's solution space gets more restricted at every timestep because it is impossible to change the choices made in the earlier timesteps. Iterative refinement, in principle, allows us to modify the entire solution and correct for the previous mistakes. This could be especially effective for generating structured objects with a longer length and non-trivial dependencies among the symbols in the output. Future work should explore applications of iterative refinement for such structured objects.

Alternative approaches for structured object generation is a very young and exciting area of research. Aside from the iterative refinement approach discussed in this thesis, there have been several other approaches introduced concurrently to this thesis. Notable approaches include non-monotonic insertion-based sequence generation [Stern et al., 2019, Gu et al., 2019a, Chan et al., 2019, Gu et al., 2019b], non-monotonic sequence generation based on imitation learning framework [Welleck et al., 2019], semi-autoregressive sequence generation [Stern et al., 2018, Wang et al., 2018b], non-autoregressive machine translation approaches based on discrete [Kaiser et al., 2018, Roy et al., 2018] and continuous [Ma et al., 2019, Shu et al., 2019, Lee et al., 2020a] latent-variable models, image and speech generation approaches based on the iterative denoising diffusion principle [Sohl-Dickstein et al., 2015, Ho et al., 2020, Chen et al., 2020]. We believe that iterative refinement and other alternative approaches for structured prediction are an exciting avenue for future work.

Bibliography

[Adiwardana et al., 2020] Adiwardana, D., Luong, M.-T., So, D., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., and Le, Q. V. (2020). Towards a human-like open-domain chatbot. *ArXiv*, abs/2001.09977.

[Alain and Bengio, 2012] Alain, G. and Bengio, Y. (2012). What regularized auto-encoders learn from the data-generating distribution. *J. Mach. Learn. Res.*, 15:3563–3593.

[AlQuraishi, 2018] AlQuraishi, M. (2018). End-to-end differentiable learning of protein structure. *bioRxiv*.

[Artetxe et al., 2017] Artetxe, M., Labaka, G., Agirre, E., and Cho, K. (2017). Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*.

[Axelrod and Gómez-Bombarelli, 2020] Axelrod, S. and Gómez-Bombarelli, R. (2020). Geom: Energy-annotated molecular conformations for property prediction and molecular generation. *ArXiv*, abs/2006.05531.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

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

[Bengio, 2007] Bengio, Y. (2007). Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2:1–127.

[Bengio and Bengio, 1999] Bengio, Y. and Bengio, S. (1999). Modeling high-dimensional discrete data with multi-layer neural networks. In *NIPS*, pages 400–406.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

[Bengio et al., 2006] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006). Greedy layer-wise training of deep networks. In *NIPS*.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[Bengio et al., 2013] Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In *NIPS*.

[Berglund et al., 2015] Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., and Karhunen, J. T. (2015). Bidirectional recurrent neural networks as generative models. In *Advances in Neural Information Processing Systems*, pages 856–864.

[Besag, 1977] Besag, J. (1977). Efficiency of pseudolikelihood estimation for simple gaussian fields.

[Blaney and Dixon, 1994] Blaney, J. M. and Dixon, J. S. (1994). Distance geometry in molecular modeling. *Rev. Comput. Chem.*, pages 299–335.

[Bohacek et al., 1996] Bohacek, R. S., Mcmartin, C., and Guida, W. C. (1996). The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3–50.

[Bojar et al., 2014] Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., and Tamchyna, A. s. (2014). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

[Bordes and Weston, 2017] Bordes, A. and Weston, J. (2017). Learning end-to-end goal-oriented dialog. *ArXiv*, abs/1605.07683.

[Bordes et al., 2017] Bordes, F., Honari, S., and Vincent, P. (2017). Learning to generate samples from noise through infusion training. In *ICLR*.

[Bottou et al., 1997] Bottou, L., Bengio, Y., and LeCun, Y. (1997). Global training of document processing systems using graph transformer networks. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 489–494.

[Bourlard and Kamp, 1988] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*.

[Bowman et al., 2015] Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *ArXiv*, abs/1508.05326.

[Brown et al., 2018] Brown, N., Fiscato, M., Segler, M. H., and Vaucher, A. C. (2018). Guacamol: Benchmarking models for de novo molecular design. *arXiv preprint 1811.09621*.

[Brown et al., 1993] Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

[Brown et al., 1992] Brown, P. F., Pietra, V. D., Souza, P. D., Lai, J., and Mercer, R. (1992). Class-based n-gram models of natural language. *Comput. Linguistics*, 18:467–479.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krüger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *ArXiv*, abs/2005.14165.

[Bruna et al., 2014] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and locally connected networks on graphs. In *Proceedings of the 2nd International Conference on Learning Representations*.

[Cao and Kipf, 2018] Cao, N. D. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs. *arXiv preprint 1805.11973*.

[Chan et al., 2019] Chan, W., Kitaev, N., Guu, K., Stern, M., and Uszkoreit, J. (2019). Kermit: Generative insertion-based modeling for sequences. *arXiv preprint arXiv:1906.01604*.

[Chen et al., 2019] Chen, M., Lee, B., Bansal, G., Cao, Y., Zhang, S., Lu, J. Y., Tsay, J., Wang, Y., Dai, A. M., Chen, Z., Sohn, T., and Wu, Y. (2019). Gmail smart compose: Real-time assisted writing. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*.

[Chen et al., 2020] Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. (2020). Wavegrad: Estimating gradients for waveform generation.

[Chen and Goodman, 1996] Chen, S. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *ACL*.

[Chiu et al., 2017] Chiu, C.-C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, K., et al. (2017). State-of-the-art speech recognition with sequence-to-sequence models. *arXiv preprint arXiv:1712.01769*.

[Cho, 2016] Cho, K. (2016). Noisy parallel approximate decoding for conditional recurrent language model. *arXiv preprint arXiv:1605.03835*.

[Cho et al., 2015] Cho, K., Courville, A., and Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11).

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

[Chorowski et al., 2015] Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *NIPS*.

[Ciresan et al., 2012] Ciresan, D. C., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

[Dai et al., 2020] Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. (2020). Scalable deep generative modeling for sparse graphs. *ArXiv*, abs/2006.15502.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *ArXiv*, abs/1901.02860.

[Dempster et al., 1977] Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the em - algorithm plus discussions on the paper.

[Deng et al., 2009a] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009a). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[Deng et al., 2009b] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009b). Imagenet: A large-scale hierarchical image database. In *CVPR*.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., and Kenton Lee, K. T. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

[Du et al., 2020] Du, Y., Meier, J., Ma, J., Fergus, R., and Rives, A. (2020). Energy-based models for atomic-resolution protein conformations. *ArXiv*, abs/2004.13167.

[Dyer et al., 2013] Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *ACL*.

[Ebejer et al., 2012] Ebejer, J.-P., Morris, G. M., and Deane, C. M. (2012). Freely available conformer generation methods: How good are they? *J. Chem. Inf. Model.*, 52(5):1146–1158.

[Edunov et al., 2019] Edunov, S., Baevski, A., and Auli, M. (2019). Pre-trained language model representations for language generation.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[Elton et al., 2019] Elton, D., Boukouvalas, Z., Fuge, M. D., and Chung, P. W. (2019). Deep learning for molecular design—a review of the state of the art.

[Fan et al., 2018] Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation. In *ACL*.

[Finkel et al., 2008] Finkel, J. R., Kleeman, A., and Manning, C. D. (2008). Efficient, feature-based, conditional random field parsing. In *ACL*.

[Friedrich et al., 2017] Friedrich, N.-O., de Bruyn Kops, C., Flachsenberg, F., Sommer, K., Rarey, M., and Kirchmair, J. (2017). Benchmarking commercial conformer ensemble generators. *J. Chem. Inf. Model.*, 57(11):2719–2728.

[Gebauer et al., 2019] Gebauer, N., Gastegger, M., and Schütt, K. T. (2019). Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. In *NeurIPS*.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. (2017). Convolutional sequence to sequence learning. In *ICML*.

[Germann et al., 2001] Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 228–235, Toulouse, France. Association for Computational Linguistics.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. A. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471.

[Ghazvininejad et al., 2019] Ghazvininejad, M., Levy, O., Liu, Y., and Zettlemoyer, L. (2019). Mask-predict: Parallel decoding of conditional masked language models. In *EMNLP*.

[Ghazvininejad et al., 2020] Ghazvininejad, M., Levy, O., and Zettlemoyer, L. (2020). Semi-autoregressive training improves mask-predict decoding. *ArXiv*, abs/2001.08785.

[Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. JMLR Workshop and Conference Proceedings.

[Goh et al., 2017] Goh, G. B., Siegel, C., Vishnu, A., and Hodas, N. O. (2017). Chemnet: A transferable and generalizable deep neural network for small-molecule property prediction. *ArXiv*, abs/1712.02734.

[Gómez-Bombarelli et al., 2016] Gómez-Bombarelli, R., Duvenaud, D., Hernández-Lobato, J. M., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-

Guzik, A. (2016). Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint 1610.02415*.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS*.

[Grangier and Auli, 2017] Grangier, D. and Auli, M. (2017). Quickedit: Editing text & translations via simple delete actions. *arXiv preprint arXiv:1711.04805*.

[Graves et al., 2006] Graves, A., Fernández, S., Gomez, F. J., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML '06*.

[Graves et al., 2013] Graves, A., Jaitly, N., and rahman Mohamed, A. (2013). Hybrid speech recognition with deep bidirectional lstm. In *ASRU*.

[Gražulis et al., 2012] Gražulis, S., Daškevič, A., Merkys, A., Chateigner, D., Lutterotti, L., Quiros, M., Serebryanaya, N. R., Moeck, P., Downs, R. T., and Le Bail, A. (2012). Crystallography open database (COD): An open-access collection of crystal structures and platform for world-wide collaboration. *Nucleic Acids Res.*, 40(D1):D420–D427.

[Groom et al., 2016] Groom, C. R., Bruno, I. J., Lightfoot, M. P., and Ward, S. C. (2016). The cambridge structural database. *Acta Crystallogr. Sect. B-Struct. Sci.Cryst. Eng. Mat.*, 72(2):171–179.

[Grover et al., 2019] Grover, A., Zweig, A., and Ermon, S. (2019). Graphite: Iterative generative modeling of graphs. In *ICML*.

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

[Gu et al., 2019a] Gu, J., Liu, Q., and Cho, K. (2019a). Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*.

[Gu et al., 2019b] Gu, J., Wang, C., and Zhao, J. (2019b). Levenshtein transformer. In *NeurIPS*.

[Guimaraes et al., 2017] Guimaraes, G. L., Sanchez-Lengeling, B., Farias, P. L. C., and Aspuru-Guzik, A. (2017). Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *CoRR*, abs/1705.10843.

[Gupta et al., 2018] Gupta, A., Müller, A., Huisman, B. J. H., Fuchs, J. A., Schneider, P., and Schneider, G. (2018). Generative recurrent networks for de novo drug design. *Molecular Informatics*, 37.

[Halgren, 1996] Halgren, T. A. (1996). Merck molecular force field. I. basis, form, scope, parameterization, and performance of MMFF94. *J. Comput. Chem.*, 17(5-6):490–519.

[Hautier et al., 2012] Hautier, G., Jain, A., and Ong, S. P. (2012). From the computer to the laboratory: Materials discovery and design using first-principles calculations. *J. Mater. Sci.*, 47(21):7317–7340.

[Hawkins, 2017] Hawkins, P. C. D. (2017). Conformation generation: The state of the art. *J. Chem. Inf. Model.*, 57(8):1747–1756.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv:1606.08415,*.

[Hill et al., 2016] Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *NAACL*.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[Hinton et al., 2012] Hinton, G. E., Deng, L., Yu, D., Dahl, G., rahman Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313:504 – 507.

[Hinton and Zemel, 1993] Hinton, G. E. and Zemel, R. (1993). Autoencoders, minimum description length and helmholtz free energy. In *NIPS*.

[Ho et al., 2020] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *ArXiv*, abs/2006.11239.

[Hoang et al., 2017] Hoang, C. D. V., Haffari, G., and Cohn, T. (2017). Decoding as continuous optimization in neural machine translation. *arXiv preprint arXiv:1701.02854*.

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

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

[Hoffmann et al., 2019] Hoffmann, J., Maestrati, L., Sawada, Y., Tang, J., Sellier, J., and Bengio, Y. (2019). Data-driven approach to encoding and decoding 3-d crystal structures. *ArXiv*, abs/1909.00949.

[Holtzman et al., 2019] Holtzman, A., Buys, J., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. *ArXiv*, abs/1904.09751.

[Hu et al., 2019] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V. S., and Leskovec, J. (2019). Pre-training graph neural networks. *arXiv preprint arXiv:v*.

[Ingraham et al., 2019a] Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019a). Generative models for graph-based protein design. In *DGS@ICLR*.

[Ingraham et al., 2019b] Ingraham, J. B., Riesselman, A. J., Sander, C., and Marks, D. (2019b). Learning protein structure with a differentiable simulator. In *ICLR*.

[Jelinek, 1976] Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64:532–556.

[Jelinek et al., 1975] Jelinek, F., Bahl, L. R., and Mercer, R. L. (1975). Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Trans. Inf. Theory*, 21:250–256.

[Jensen, 2018] Jensen, J. H. (2018). Graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space.

[Jin et al., 2018] Jin, W., Barzilay, R., and Jaakkola, T. S. (2018). Junction tree variational autoencoder for molecular graph generation. In *ICML*.

[Jouppi et al., 2017] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*.

[Kaiser and Bengio, 2016] Kaiser, Ł. and Bengio, S. (2016). Can active memory replace attention? In *NIPS*.

[Kaiser et al., 2018] Kaiser, L., Roy, A., Vaswani, A., Parmar, N., Bengio, S., Uszkoreit, J., and Shazeer, N. (2018). Fast decoding in sequence models using discrete latent variables. In *ICML*.

[Kaiser and Sutskever, 2015] Kaiser, Ł. and Sutskever, I. (2015). Neural GPUs learn algorithms. *arXiv preprint arXiv:1511.08228*.

[Kalchbrenner and Blunsom, 2013] Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *EMNLP*.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

[Kanal et al., 2017] Kanal, I. Y., Keith, J. A., and Hutchison, G. R. (2017). A sobering assessment of small-molecule force field methods for low energy conformer predictions. *Int. J. Quantum Chem.*, 118(5):e25512.

[Kang and Cho, 2019] Kang, S. and Cho, K. (2019). Conditional molecular design with deep generative models. *Journal of chemical information and modeling*, 59 1:43–52.

[Kannan et al., 2016] Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G. S., Lukács, L., Ganea, M., Young, P., and Ramavajjala, V. (2016). Smart reply: Automated response suggestion for email. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[Karpathy and Li, 2015] Karpathy, A. and Li, F. (2015). Deep visual-semantic alignments for generating image descriptions. In *CVPR*.

[Kasai et al., 2020] Kasai, J., Pappas, N., Peng, H., Cross, J., and Smith, N. A. (2020). Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *ArXiv*, abs/2006.10369.

[Katz, 1987] Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoust. Speech Signal Process.*, 35:400–401.

[Kim and Rush, 2016] Kim, Y. and Rush, A. M. (2016). Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

[Kim et al., 2018] Kim, Y., Wiseman, S., and Rush, A. M. (2018). A tutorial on deep latent variable models of natural language. *ArXiv*, abs/1812.06834.

[Kingma, 2017] Kingma, D. P. (2017). Variational inference deep learning: A new synthesis.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma et al., 2016] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *NIPS*.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations*.

[Kneser and Ney, 1993] Kneser, R. and Ney, H. (1993). Improved clustering techniques for class-based statistical language modelling. In *EUROSPEECH*.

[Kneser and Ney, 1995] Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1:181–184 vol.1.

[Koehn et al., 2007] Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *ACL*.

[Krenn et al., 2019] Krenn, M., Häse, F., Nigam, A., Friederich, P., and Aspuru-Guzik, A. (2019). Self-referencing embedded strings (selfies): A 100% robust molecular string representation.

[Kreutzer et al., 2020] Kreutzer, J., Foster, G., and Cherry, C. (2020). Inference strategies for machine translation with conditional masking. *arXiv*, abs/2010.02352.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86.

[Kusner et al., 2017] Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *ICML*.

[Kwon et al., 2019] Kwon, Y., Yoo, J., Choi, Y., joon Son, W., Lee, D., and Kang, S. (2019). Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation. *Journal of Cheminformatics*, 11.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*.

[Lample and Conneau, 2019] Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

[Lample et al., 2017] Lample, G., Denoyer, L., and Ranzato, M. (2017). Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*.

[Lan et al., 2020] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942.

[Landrum,] Landrum, G. Rdkit: Open-source cheminformatics. (accessed December 18, 2018).

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *The Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *JMLR: W&CP*, pages 29–37.

[Lawrence et al., 2019] Lawrence, C., Kotnis, B., and Niepert, M. (2019). Attending to future tokens for bidirectional sequence generation. *ArXiv*, abs/1908.05915.

[LeCun, 1985] LeCun, Y. (1985). A learning scheme for asymmetric threshold networks.

[LeCun, 1987] LeCun, Y. (1987). Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models).

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.

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

[Lee et al., 2020a] Lee, J., Shu, R., and Cho, K. (2020a). Iterative refinement in the continuous space for non-autoregressive neural machine translation.

[Lee et al., 2020b] Lee, J., Tran, D., Firat, O., and Cho, K. (2020b). On the discrepancy between density estimation and sequence generation. *ArXiv*, abs/2002.07233.

[Lepikhin et al., 2020] Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. (2020). Gshard: Scaling giant models with conditional computation and automatic sharding. *ArXiv*, abs/2006.16668.

[Levine and Casella, 2006] Levine, R. A. and Casella, G. (2006). Optimizing random scan gibbs samplers. *Journal of Multivariate Analysis*, 97(10):2071–2100.

[Li et al., 2018a] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. W. (2018a). Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

[Li et al., 2018b] Li, Y., Zhang, L., and Liu, Z. (2018b). Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10(1).

[Liao et al., 2019] Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W. L., Duvenaud, D., Urtasun, R., and Zemel, R. S. (2019). Efficient graph generation with graph recurrent attention networks. In *NeurIPS*.

[Liao et al., 2020] Liao, Y., Jiang, X., and Liu, Q. (2020). Probabilistically masked language model capable of autoregressive generation in arbitrary word order. In *ACL*.

[Lin et al., 2014] Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *ECCV*.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1-3):503–528.

[Liu et al., 2019a] Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019a). Graph normalizing flows. In *NeurIPS*.

[Liu et al., 2019b] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

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

[Mahmood and Hernández-Lobato, 2019] Mahmood, O. and Hernández-Lobato, J. M. (2019). A COLD approach to generating optimal samples. *CoRR*, abs/1905.09885.

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. J. (2015). Adversarial autoencoders. *ArXiv*, abs/1511.05644.

[Mansimov et al., 2019] Mansimov, E., Wang, A., and Cho, K. (2019). A generalized framework of sequence generation with application to undirected sequence models. *arXiv preprint arXiv:1905.12790*.

[Maziarka et al., 2020] Maziarka, L., Danel, T., Mucha, S., Rataj, K., Tabor, J., and Jastrzebski, S. (2020). Molecule attention transformer. *ArXiv*, abs/2002.08264.

[McDonald et al., 2005] McDonald, R., Pereira, F. C., Ribarov, K., and Hajic, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP*.

[Melis et al., 2018] Melis, G., Dyer, C., and Blunsom, P. (2018). On the state of the art of evaluation in neural language models. *ArXiv*, abs/1707.05589.

[Mendez et al., 2018] Mendez, D., Gaulton, A., Bento, A. P., Chambers, J., De Veij, M., Félix, E., Magariños, M., Mosquera, J., Mutowo, P., Nowotka, M., Gordillo-Marañón, M., Hunter, F., Junco, L., Mugumbate, G., Rodriguez-Lopez, M., Atkinson, F., Bosc, N., Radoux, C., Segura-Cabrera, A., Hersey, A., and Leach, A. (2018). ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Research*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

[Mikolov et al., 2011] Mikolov, T., Kombrink, S., Deoras, A., Burget, L., and ernocký, J. (2011). Rnnlm - recurrent neural network language modeling toolkit.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814.

[Niesler et al., 1998] Niesler, T., Whittaker, E. W. D., and Woodland, P. (1998). Comparison of part-of-speech and automatically derived category-based language models for speech recognition. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, 1:177–180 vol.1.

[Nogueira and Cho, 2019] Nogueira, R. and Cho, K. (2019). Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

[Novak et al., 2016] Novak, R., Auli, M., and Grangier, D. (2016). Iterative refinement for machine translation. *arXiv preprint arXiv:1610.06602*.

[Och and Ney, 2004] Och, F. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, 30:417–449.

[Oja, 1991] Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. *Artificial Neural Networks*.

[Oord et al., 2017] Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L. C., Stimberg, F., et al. (2017). Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *ACL*.

[Pearl, 1986] Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288.

[Polykovskiy et al., 2018] Polykovskiy, D., Zhebrak, A., Vetrov, D., Ivanenkov, Y., Aladinskiy, V., Mamoshina, P., Bozdaganyan, M., Aliper, A., Zhavoronkov, A., and Kadurin, A. (2018). Entangled conditional adversarial autoencoder for de novo drug discovery. *Molecular Pharmaceutics*.

[Preuer et al., 2018] Preuer, K., Renz, P., Unterthiner, T., Hochreiter, S., and Klambauer, G. (2018). Fréchet chemnet distance: A metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, 3:4–16.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

[Rajpurkar et al., 2018] Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. *ArXiv*, abs/1806.03822.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100, 000+ questions for machine comprehension of text. *ArXiv*, abs/1606.05250.

[Ramachandran et al., 2017] Ramachandran, P., Liu, P., and Le, Q. (2017). Unsupervised pretraining for sequence to sequence learning. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

[Ramakrishnan et al., 2014] Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Sci. Data*, 1:140022:1–7.

[Rao et al., 2019] Rao, R., Bhattacharya, N., Thomas, N., Duan, Y., Chen, X., Canny, J., Abbeel, P., and Song, Y. S. (2019). Evaluating protein transfer learning with tape. *bioRxiv*.

[Rappé et al., 1992] Rappé, A. K., Casewit, C. J., Colwell, K. S., Goddard III, W. A., and Skiff, W. M. (1992). UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *J. Am. Chem. Soc.*, 114(25):10024–10035.

[Ratcliff et al., 2017] Ratcliff, L. E., Mohr, S., Huhs, G., Deutsch, T., Masella, M., and Genovese, L. (2017). Challenges in large scale quantum mechanical calculations. *Wiley Interdiscip. Rev.-Comput. Mol. Sci.*, 7(1):e1290.

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

[Riniker and Landrum, 2015] Riniker, S. and Landrum, G. A. (2015). Better informed distance geometry: Using what we know to improve conformation generation. *J. Chem. Inf. Model.*, 55(12):2562–2574.

[Rives et al., 2019] Rives, A., Goyal, S., Meier, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., and Fergus, R. (2019). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*.

[Roy et al., 2018] Roy, A., Vaswani, A., Parmar, N., and Neelakantan, A. (2018). Towards a better understanding of vector quantized autoencoders. *arXiv preprint 1805.11063*.

[Ruddigkeit et al., 2012] Ruddigkeit, L., Van Deursen, R., Blum, L. C., and Reymond, J.-L. (2012). Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *J. Chem. Inf. Model.*, 52(11):2864–2875.

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

[Rush et al., 2015] Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. *ArXiv*, abs/1509.00685.

[Sadowski and Baldi, 2013] Sadowski, P. and Baldi, P. (2013). Small-molecule 3d structure prediction using open crystallography data. *J. Chem. Inf. Model.*, 53(12):3127–3130.

[Saharia et al., 2020] Saharia, C., Chan, W., Saxena, S., and Norouzi, M. (2020). Non-autoregressive machine translation with latent alignments. *ArXiv*, abs/2004.07437.

[Sarawagi and Cohen, 2004] Sarawagi, S. and Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. In *NIPS*.

[Scarselli et al., 2009] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Trans. Neural Netw.*, 20(1):61–80.

[Schmidt et al., 1993] Schmidt, M. W., Baldridge, K. K., Boatz, J. A., Elbert, S. T., Gordon, M. S., Jensen, J. H., Koseki, S., Matsunaga, N., Nguyen, K. A., Su, S., Windus, T. L., Dupuis, M., and Montgomery, J. A. (1993). General atomic and molecular electronic structure system. *Journal of Computational Chemistry*, 14:1347–1363.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[Schwab, 2010] Schwab, C. H. (2010). Conformations and 3d pharmacophore searching. *Drug Discov. Today*, 7(4):e245–e253.

[Schwenk, 2012] Schwenk, H. (2012). Continuous space translation models for phrase-based statistical machine translation. *Proceedings of COLING 2012: Posters*.

[Schwenk and Gauvain, 2005] Schwenk, H. and Gauvain, J.-L. (2005). Training neural network language models on very large corpora. In *HLT/EMNLP*.

[Schweppe, 1965] Schweppe, F. (1965). Evaluation of likelihood functions for gaussian signals. *IEEE Trans. Inf. Theory*, 11:61–70.

[Senior et al., 2020] Senior, A., Evans, R., Jumper, J., Kirkpatrick, J. R. M., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D., Silver, D., Kavukcuoglu, K., and Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577:706–710.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *ACL*.

[Serban et al., 2016] Serban, I., Sordoni, A., Bengio, Y., Courville, A. C., and Pineau, J. (2016). Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*.

[Shannon, 1948] Shannon, C. (1948). The mathematical theory of communication.

[Shen et al., 2018] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R., Saurous, R. A., Agiomyrgiannakis, Y., and Wu, Y. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783.

[Shen et al., 2020] Shen, T., Quach, V., Barzilay, R., and Jaakkola, T. (2020). Blank language models. *ArXiv*, abs/2002.03079.

[Shoeybi et al., 2019] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053.

[Shu et al., 2019] Shu, R., Lee, J., Nakayama, H., and Cho, K. (2019). Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. *arXiv preprint 1908.07181*.

[Simm and Hernández-Lobato, 2019] Simm, G. N. C. and Hernández-Lobato, J. M. (2019). A generative model for molecular distance geometry. *ArXiv*, abs/1909.11459.

[Simm et al., 2020] Simm, G. N. C., Pinsler, R., and Hernández-Lobato, J. M. (2020). Reinforcement learning for molecular design guided by quantum mechanics.

[Simonovsky and Komodakis, 2018] Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint 1802.03480*.

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *ArXiv*, abs/1503.03585.

[Song et al., 2019] Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). Mass: Masked sequence to sequence pre-training for language generation.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting.

[Srivastava et al., 2015] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

[Stern et al., 2019] Stern, M., Chan, W., Kiros, J., and Uszkoreit, J. (2019). Insertion transformer: Flexible sequence generation via insertion operations. In *ICML*.

[Stern et al., 2018] Stern, M., Shazeer, N., and Uszkoreit, J. (2018). Blockwise parallel decoding for deep autoregressive models. In *NeurIPS*.

[Sun et al., 2017] Sun, Q., Lee, S., and Batra, D. (2017). Bidirectional beam search: Forward-backward inference in neural sequence models for fill-in-the-blank image captioning.

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

[Tillmann and Ney, 2003] Tillmann, C. and Ney, H. (2003). Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29:97–133.

[Tu et al., 2020] Tu, L., Pang, R. Y., Wiseman, S., and Gimpel, K. (2020). Engine: Energy-based inference networks for non-autoregressive machine translation. *ArXiv*, abs/2005.00850.

[van den Oord et al., 2016a] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499.

[van den Oord et al., 2016b] van den Oord, A., Kalchbrenner, N., Espeholt, L., Kavukcuoglu, K., Vinyals, O., and Graves, A. (2016b). Conditional image generation with pixelcnn decoders. In *NIPS*.

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

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML '08*.

[Vinyals et al., 2016] Vinyals, O., Bengio, S., and Kudlur, M. (2016). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

[Vinyals et al., 2015] Vinyals, O., Kaiser, L., Koo, T. K., Petrov, S., Sutskever, I., and Hinton, G. E. (2015). Grammar as a foreign language. In *NIPS*.

[Vinyals and Le, 2015] Vinyals, O. and Le, Q. V. (2015). A neural conversational model. *ArXiv*, abs/1506.05869.

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 13:260–269.

[Wang and Cho, 2019] Wang, A. and Cho, K. (2019). Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:1902.04094*.

[Wang et al., 2019a] Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019a). Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537.

[Wang et al., 2018a] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018a). Glue: A multi-task benchmark and analysis platform for natural language understanding. *ArXiv*, abs/1804.07461.

[Wang et al., 2018b] Wang, C., Zhang, J., and Chen, H. (2018b). Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*.

[Wang et al., 2017a] Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. (2017a). Graphgan: Graph representation learning with generative adversarial nets. *CoRR*, abs/1711.08267.

[Wang et al., 2017b] Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q. V., Agiomyrgiannakis, Y., Clark, R., and Saurous, R. A. (2017b). Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*.

[Wang et al., 2019b] Wang, Y., Tian, F., He, D., Qin, T., Zhai, C., and Liu, T.-Y. (2019b). Non-autoregressive machine translation with auxiliary regularization. *arXiv preprint 1902.10245*.

[Weininger, 1988] Weininger, D. (1988). SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28(1):31–36.

[Welleck et al., 2019] Welleck, S., Brantley, K., Daumé, H., and Cho, K. (2019). Non-monotonic sequential text generation. In *ICML*.

[Williams et al., 2018] Williams, A., Nangia, N., and Bowman, S. R. (2018). A broad-coverage challenge corpus for sentence understanding through inference. *ArXiv*, abs/1704.05426.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G. S., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

[Xia et al., 2017] Xia, Y., Tian, F., Wu, L., Lin, J., Qin, T., Yu, N., and Liu, T.-Y. (2017). Deliberation networks: Sequence generation beyond one-pass decoding. In *NIPS*.

[Xu, 2019] Xu, J. (2019). Distance-based protein folding powered by deep learning. *bioRxiv*.

[Yang et al., 2019] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

[You et al., 2018a] You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J. (2018a). Graph convolutional policy network for goal-directed molecular graph generation. *ArXiv*, abs/1806.02473.

[You et al., 2018b] You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018b). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*.

[Zaremba et al., 2014] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *ArXiv*, abs/1409.2329.

[Zhou et al., 2019] Zhou, Z., Kearnes, S. M., Li, L., Zare, R. N., and Riley, P. (2019). Optimization of molecules via deep reinforcement learning. *Scientific Reports*, 9.