Université de Montréal

Sequential Decision Modeling In Uncertain Conditions

par Kyle Kastner

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Août, 2023

Université de Montréal
Faculté des arts et des sciences

Cette thèse intitulée:

Sequential Decision Modeling In Uncertain Conditions

présentée par:

Kyle Kastner

a été évaluée par un jury composé des personnes suivantes:

Aishwarya Agrawal ,	président-rapporteur
Aaron Courville ,	directeur de recherche
Cheng-Zhi Anna Huang ,	membre du jury
Gus (Guangyu) Xia ,	examinateur externe
Marlène Frigon ,	représentant du doyen de la FESP

Résumé

Cette thèse consiste en une série d'approches pour la modélisation de décision structurée - c'est-à-dire qu'elle propose des solutions utilisant des modèles génératifs pour des tâches intégrant plusieurs entrées et sorties, ces entrées et sorties étant dictées par des interactions complexes entre leurs éléments. Un aspect crucial de ces problèmes est la présence en plus d'un résultat correct, des résultats structurellement différents mais considérés tout aussi corrects, résultant d'une grande mais nécessaire incertitude sur les sorties du système. Cette thèse présente quatre articles sur ce sujet, se concentrent en particulier sur le domaine de la synthèse vocale à partir de texte, génération symbolique de musique, traitement de texte, reconnaissance automatique de la parole, et apprentissage de représentations pour la parole et le texte. Chaque article présente une approche particulière à un problème dans ces domaines respectifs, en proposant et étudiant des architectures profondes pour ces domaines. Bien que ces techniques d'apprentissage profond utilisées dans ces articles sont suffisamment versatiles et expressives pour être utilisées dans d'autres domaines, nous resterons concentrés sur les applications décrites dans chaque article.

Le premier article présente une approche permettant le contrôle détaillé, au niveau phonétique et symbolique, d'un système de synthèse vocale, en utilisant une méthode d'échange efficace permettant de combiner des représentations à un niveau lexical. Puisque cette combinaison permet un contrôle proportionné sur les conditions d'entrée, et améliore les prononciations faisant uniquement usage de caractères, ce système de combinaison pour la synthèse vocale a été préféré durant des tests A/B par rapport à des modèles de référence équivalents utilisant les mêmes modalités. Le deuxième article se concentre sur un autre système de synthèse vocale, cette fois-ci centré sur la construction d'une représentation multi-échelle de la parole à travers une décomposition structurée des descripteurs audio. En particulier, l'intérêt de ce travail est dans sa méthodologie économe en calcul malgré avoir été bâti à partir de travaux antérieurs beaucoup plus demandant en ressources de calcul. Afin de bien pouvoir faire de la synthèse vocale sous ces contraintes computationelles, plusieurs nouvelles composantes ont été conçues et intégrées à ce qui devient un modèle efficace de synthèse vocale. Le troisième article un nouveau modèle auto-régressif pour modéliser des chaînes de symboles. Ce modèle fait usage de prédictions et d'estimations itérative et répétées afin de construire une sortie structurée respectant plusieurs contraintes correspondant au domaine sous-jacent. Ce modèle est testé dans le cadre de la génération symbolique

de musique et la modélisation de texte, faisant preuve d'excellentes performances en particulier quand la quantité de données s'avère limitée. Le dernier article de la thèse se concentre sur l'étude des représentations pour la parole et le texte apprise à partir d'un système de reconnaissance vocale d'un travail antérieur. À travers une série d'études systématiques utilisant des modèles pré-entraînés de texte et de durée, relations qualitatives entre les données de texte et de parole, et études de performance sur la récupération transmodal "few shot", nous exposons plusieurs propriétés essentielles sous-jacent à la performance du système, ouvrant la voie pour des développements algorithmiques futurs. De plus, les différents modèles résultants de cette étude obtiennent des résultats impressionnants sur un nombre de tâches de référence utilisant des modèles pré-entraîné transféré sans modification.

Mots-clés: réseaux de neurones, apprentissage automatique, apprentissage de représentations profondes, apprentissage supervisé, modèles génératifs, prédiction structurée

Summary

This thesis presents a sequence of approaches to structured decision modeling - that is, proposing generative solutions to tasks with multiple inputs and outputs, featuring complicated interactions between input elements and output elements. Crucially, these problems also include a high amount of uncertainty about the correct outcome and many largely equivalent but structurally different outcomes can be considered equally correct. This thesis presents four articles about these topics, particularly focusing on the domains of text-to-speech synthesis, symbolic music generation, text processing, automatic speech recognition, and speech-text representation learning. Each article presents a particular approach to solving problems in these respective domains, focused on proposing and understanding deep learning architectures for these domains. The deep learning techniques used in these articles are broadly applicable, flexible, and powerful enough that these general approaches may find application to other areas however we remain focused on the domains discussed in each respective article.

The first article presents an approach allowing for flexible phonetic and character control of a text-to-speech system, utilizing an efficient "swap-out" method for blending representations at the word level. This blending allows for smooth control over input conditions, and also strengthens character only pronunciations, resulting in a preference for a blended text-to-speech system in A/B testing, compared to an equivalent baselines even when using the same input information modalities. The second article focuses on another text-to-speech system, this time centered on building multi-scale representations of speech audio using a structured decomposition of audio features. Particularly this work focuses on a compute efficient methodology, while building on prior work which requires a much greater computational budget than the proposed system. In order to effectively perform text-to-speech synthesis under these computational constraints, a number of new components are constructed and integrated, resulting in an efficient model for text-to-speech synthesis. The third article presents a new non-autoregressive model for modeling symbolic sequences. This model uses iterative prediction and re-estimation in order to build structured outputs, which respect numerous constraints in the underlying sequence domain. This model is applied to symbolic music modeling and text modeling, showing excellent performance particularly in limited data generative settings. The final article in this thesis focuses on understanding the speech-text representations learned by a text-injected speech recognition system from prior literature. Through a systematic series of studies utilizing pre-trained text and

duration models, qualitative relations between text and speech sequences, and performance studies in few-shot cross-modal retrieval, we reveal a number of crucial properties underlying the performance of this system, paving the way for future algorithmic development. In addition, model variants built during this study achieve impressive performance results on a number of benchmark tasks using partially frozen and transferred parameters.

Keywords: neural networks, machine learning, deep learning, supervised learning, generative modeling, structured prediction

Contents

Résumé . . . . . **iii**

Summary . . . . . **v**

Contents . . . . . **vii**

List of Figures . . . . . **xi**

List of Tables . . . . . **xiii**

List of Abbreviations . . . . . **xiv**

Acknowledgments . . . . . **xv**

1 Introduction **1**
1.1 Creative Decisions 1
1.1.1 Formulation as a Sequence Learning Problem 2
1.1.2 Motivation 3
1.1.3 Document Organization 4

2 Background **5**
2.1 Background Setup and Notation 5
2.2 Autoencoders and Latent Variables 5
2.2.1 VAE 5
2.2.2 VQ-VAE 6
2.3 Sequence Abstractions 9
2.3.1 RNN Families 10
2.3.2 Conditional RNN 13
2.3.3 Attention 14
2.3.4 Attention-centric Models 15
2.3.5 Beam Search 16
2.4 Masked Conditional Modeling 17
2.4.1 PixelCNN 19
2.4.2 Mask Structure 20
2.4.3 Teacher Forcing 21

2.4.4 Order . . . 22
2.5 Augmenting Decision Making in Probabilistic Models . . . 23
2.5.1 Learning and Assignment . . . 24
2.5.2 Markov Masks . . . 24
2.5.3 Limitations of Constraints . . . 26
2.6 Speech Synthesis . . . 26
2.6.1 TTS Components . . . 27
2.6.2 Related TTS Work . . . 29
2.7 Symbolic Music Modeling . . . 30
2.7.1 On the Relationship between Speech Synthesis and Music Modeling . . . 30
2.7.2 Time-frequency Relations . . . 31

3 Prologue to the Article . . . 33

4 Representation Mixing for TTS Synthesis . . . 35
4.1 Introduction . . . 35
4.1.1 Data Representation . . . 35
4.1.2 Motivating Representation Mixing . . . 35
4.2 Representation Mixing Description . . . 36
4.2.1 Combining Embeddings . . . 37
4.2.2 Stacked Multi-scale Residual Convolution . . . 37
4.2.3 The Importance of Noisy Teacher Forcing . . . 38
4.2.4 Attention-based RNN Decoder . . . 38
4.2.5 Truncated Backpropagation Through Time (TBPTT) . . . 39
4.2.6 Converting Features Into Waveforms . . . 39
4.2.7 Inversion Pipeline . . . 40
4.3 Related Work . . . 41
4.4 Experiments . . . 41
4.4.1 Log Mel Spectrogram Inversion Experiments . . . 43
4.4.2 Preference Testing . . . 43
4.5 Conclusion . . . 44

5 Prologue to the Article . . . 45

6 R-MelNet . . . 46
6.1 Introduction . . . 46
6.2 System Design . . . 47
6.2.1 A Dual-Purpose View of MelNet . . . 48
6.2.2 Architecture Details . . . 49
6.2.3 Reduced Resource Requirements . . . 49
6.2.4 Attention . . . 50

6.2.5 The Importance of Inference Stochasticity . . . 52
6.2.6 Framing the Context . . . 53
6.2.7 Converting Low-Resolution Mel-Spectrograms into Audio . . . 54
6.3 Experiments . . . 55
6.4 Conclusions . . . 57
6.5 Acknowledgements . . . 57

7 Prologue to the Article . . . 58

8 SUNMASK . . . 60
8.1 Introduction . . . 60
8.1.1 Autoregressive Models . . . 60
8.1.2 Non-Autoregressive Models . . . 61
8.1.3 Trade-offs Between Autoregressive and Non-autoregressive Approaches . . . 62
8.1.4 SUNMASK, a non-autoregressive sequence model . . . 63
8.2 Method . . . 64
8.2.1 Model Training . . . 66
8.2.2 Convolutional SUNMASK . . . 67
8.2.3 Transformer SUNMASK . . . 68
8.2.4 Inference Specific Settings . . . 69
8.3 Related Work . . . 70
8.4 Experiments . . . 71
8.4.1 Musical Evaluation . . . 71
8.4.2 Text Datasets . . . 72
8.4.3 Music Control . . . 73
8.4.4 Text Control . . . 74
8.5 Conclusion . . . 74
8.6 Appendix . . . 75
8.6.1 Musical Co-Creation and Possible Ethical Concerns . . . 75
8.6.2 Text . . . 76
8.7 Convolutional SUNMASK Model Hyperparameters and Training Information . . . 77
8.7.1 Architecture Design . . . 77
8.7.2 Sampling Details . . . 78
8.8 Transformer SUNMASK Model Hyperparameters and Training Information . . . 79
8.9 Sampling Runtime . . . 80
8.10 Code repository and samples player . . . 81
8.11 Creating a "Greatest Hits" . . . 81
8.12 Full Masking Comparison Figure . . . 82

8.13 Full Quantitative Analysis of Bach data . . . 84

9 Prologue to the Article . . . 85

10 Understanding Shared Speech-Text Representations . . . 87
10.1 Introduction . . . 87
10.2 Related Work . . . 88
10.3 Architecture and Training . . . 89
10.3.1 Maestro Architecture . . . 89
10.3.2 SLAM Architecture . . . 90
10.4 Text-Only Domain Adaptation . . . 91
10.4.1 Data . . . 91
10.4.2 Experiments and Results . . . 91
10.4.3 Text Encoder Data Quality Ablation . . . 92
10.4.4 Text Encoder Component Ablation . . . 93
10.5 Representation Space Analysis . . . 94
10.6 Conclusion . . . 98

11 Conclusion . . . 99
11.0.1 Overview . . . 99
11.0.2 Retrospective . . . 100

Bibliography . . . 102

List of Figures

1.1 Example four quarter note sequence D A D D at 70 beats per minute, shown in sheet music notation 2

2.1 Figure from van den Oord et al. (2016) showing autoregressive masking over space and channels as layer depth increases. 19
2.2 Figure from van den Oord et al. (2016) showing ideal captured context, using combined masks and additional architecture changes. . . 19
2.3 Detailed drawing of Gated PixelCNN layer from van den Oord et al. (2016) 19
2.4 Example cyclic generator for dataset "A, B, C, A, B, C, A, ..." . . . 24
2.5 Quarter note sequence D A D D at 70 beats per minute, shown in sheet music notation, as first seen in Section 1.1.1 31

4.1 Visualization of embedding computation 36
4.2 Encoding, attention, and one step of mel decoder 38
4.3 Comparison of standard minibatching versus truncation. 39

6.1 Log-mel spectrogram (T=448, F=256). 48
6.2 Downsampled log-mel spectrogram (T=112, F=32). 48
6.3 Attention alignment and sampled output from Mel-Net frontend along with log-mel spectrogram of final output waveform from WaveRNN backend, for example input 'their boat sank into the icy river', phonemized to 'dhehr bowt saengk ihntuw dhiy aysiy rihver'. 55
6.4 Example of alternate successful attention alignment, and two failed alignments for the same sentence as Figure 6.3, starting with different audio and text priming. 55

8.1 Step-unrolled denoising training for SUNMASK on polyphonic music, unrolled step length 2. Training data (left) consists of four voices corrupted by sampling a random mask per voice and replacing the masked data (red) with random pitches (green). SUNMASK takes both mask and corrupted training data as input, predicting denoised original data as output. In the second step, the model takes a sampled version of the model step predictions and the same mask as input, outputting another prediction of the original data. 63

8.2 Negative BLEU/Self-BLEU scores on EMNLP2017 News. Left (x-axis) is better, lower (y-axis) is better. Quality/variation is controlled by changing the temperature (t), and varying diffusion schedule (s). For SUNMASK, *typical* sampling results (Meister et al., 2022) are shown. 73
8.3 SUNMASK harmonization (bass, tenor, alto) of an existing melody (left), with a mask which highlights the left half (0 to 64) soprano voice (middle), or a left half mask but replacing right half melody as well (right) 73
8.4 SUNMASK harmonization (bass, tenor, alto) of existing melody (soprano) based on mask. 82
8.5 A 83
8.6 B 83
8.7 C 83
8.8 D 83
8.9 E 83
8.10 F 83

10.1 Maestro architecture. 90
10.2 T-SNE of LibriSpeech Maestro text (crosses) and speech (dots) encoder outputs (left), and shared encoder output (right) 95
10.3 T-SNE of SLAM text (crosses) and speech (dots) shared encoder outputs 95
10.4 T-SNE of LS Maestro text encoder outputs (left), and shared encoder text output (right), color coded by duration 96

List of Tables

4.1 Comparison of various log mel spectrogram to waveform conversion techniques. STRTF stands for "slower than real time factor" (calculated assuming output sample rate of 22.05 kHz), where 1.0 would be real-time generation for a single example. 40
4.2 Architecture hyperparameter settings 42
4.3 A / B user preference study results. RM uses approach in parenthesis at inference time. Columns "*C. A*" and "*% A*" indicate the number and percentage of users which preferred *Model A*. 44

6.1 MOS-P listening study results, comparing Fastspeech 2, Portaspeech, and R-MelNet. 56

8.1 Comparing SUNMASK, Coconet, and SUNDAE 66
8.2 Quantitative results from the Bach Mock grading function (Fang et al., 2020). Top rows compare to existing literature, bottom rows show ablation study of SUNMASK style models. Lower values represent better chorales. 72

10.1 LibriSpeech Maestro adaptation results. 89
10.2 Ablation of text encoder trained on different corpora with Switchboard (SWBD) Text Adaptation, using text encoders trained on different corpora. 93
10.3 Ablation of importance for in-domain text encoder components with AMI Text adaptation. 94
10.4 Shared space cosine retrieval probe accuracy (%) 97
10.5 Text and speech encoder retrieval probe accuracy (%) 98

List of Abbreviations

TTS	Text to Speech
ASR	Automatic Speech Recognition
VAE	Variational Auto-Encoder
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
KLD	Kullback-Liebler Divergence
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLL	Negative Log-Likelihood

Acknowledgments

To Johanna, thank you for everything - without your love and support none of this would be possible. I look forward to our future together.

Thank you to my family, whose continual support has been critical throughout my graduate career. I would particularly like to thank my mother, Sandra; father, Victor; and sister, Courtney.

Thank you to my PhD advisor, Aaron Courville, whose advice and guidance over many projects has shaped the future direction of my research, and how to effectively tackle research problems in general. I would also like to thank the members of the pre-doctoral committee, and the PhD defense committee for feedback and improvements during the PhD process.

This document and the research underlying it would not have been possible without the support of a staggering number of family members, friends, colleagues, mentors, and advisors. If our paths crossed over the past years then your contributions (large or small) have been a critical part of the research and life that lead to this dissertation.

Thank you to Laurent Dinh for translating the summary portion of this thesis. Thank you to Mohammad Pezeshski for showing me how to use this dissertation template by example and to Nicolas Chapados for providing the template. Thank you also to Céline Begin for patience during the submission process and for administrative help all throughout my studies.

1 Introduction

1.1 Creative Decisions

Effective decision making in uncertain conditions is a fundamental skill associated with intelligence, and a hallmark of human capability. Autonomous systems which attempt to solve human-level, real-world problems must deal with risk, incorporating available information with prior knowledge to take consistent and logical actions which satisfy given constraints. A broad spectrum of data driven techniques are available to solve these decision problems, and recent methods focus particularly on sequences of decisions where the "correct" decision may be ambiguous, ill-defined, or otherwise vague. Autonomous decision systems must nevertheless follow certain explicit and implicit rules, which may be defined for some domain through datasets or prior knowledge.

Consider the act of composing a piece of music. Beginning the composition, several key questions must be asked: What type of music will this be? Which musical tools and instruments can be used to create the mood or genre desired? Are there constraints related to the final presentation of this work?

Given answers to these high level questions, a myriad of follow-on choices must be made. What tempo and rhythm suit the chosen mood of the piece? What notes fit the overall theme and the intended presentation of the work? How long should this piece of music last? What level of complexity, both rhythmically and harmonically, will satisfy the intended audience?

Each of these choices are critical to the final outcome, and should be make in a holistic fashion while not limiting the resulting composition task too much. Composing modern pop music requires radically different choices than baroque era music, requiring different choices to satisfy the stylistic constraints from past practice. However, creative music should also *violate* certain expectations from past practice, in order to create a more satisfying musical journey. Knowing how

and when to "break the rules" is a key step for moving from rote musical exercises, toward expressive and interesting music.

During the composition journey, low-level decisions about notes, instruments choices, and discovery of interesting musical sequences (motifs) could all lead the composer back to re-evaluating the initial high level goals of the composition. This kind of complex multi-scale decision making under a variety of hard and soft constraints is exactly the problem setting where a competent learning algorithm would be useful.

The close relationship between musical composition and other types of creative work such as poetry, painting, dancing, and creative writing hint that a system which understands the complex dynamics of musical composition could be useful in other creative areas. In fact a broad enough approach to this setting should be applicable to a huge swath of computational tasks. This promise has been partially shown by the applicability of automated decision systems over the years including approaches from the field of *machine learning*, showing that general purpose algorithms which *learn* from data are broadly useful for understanding, automation, and human-in-the-loop creativity.

1.1.1 Formulation as a Sequence Learning Problem

Reducing music composition to a form of general sequence learning is a key step to applying modern machine learning tools for sequences. Consider a sequence of musical notes such as Figure 1.1. Denoting the piece x, made of four individual values $x_{1:4}$ gives a shorthand for referring to this particular musical piece.

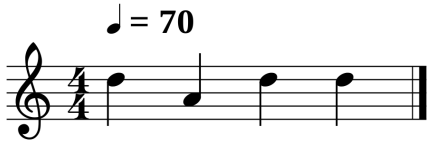

Figure 1.1 – Example four quarter note sequence D A D D at 70 beats per minute, shown in sheet music notation

Many different pieces of music x can be further collected into a dataset X, allowing for a variety of analyses. One possible analysis is to better understand how *likely* things in a sequence (here, notes) are to exist together, also known as the sequence likelihood. One approach is to build a model for the relationship between

some elements of a sequence using a function f which has parameters θ, f_θ. The parameters of this function f_θ are tuned on a given dataset, in order to estimate how expected a combination of elements are. One procedure for tuning the function f_θ from a set of examples X is maximum likelihood estimation (MLE) (Bishop, 1995), closely related to empirical risk minimization (ERM) (Vapnik, 1991). The function f can be constrained to model probabilities (or log probabilities), resulting in a function denoted as p_θ. This process is also referenced as maximum likelihood learning.

There is a vast literature on methods and procedures for maximum likelihood estimation throughout the broader machine learning literature (Murphy, 2012). This document focuses on a particular class of functions known as *neural networks* for learning probability functions from data (Goodfellow et al., 2016). These learned neural networks form the backbone of our approach to sequential decision modeling in uncertain conditions, by broadly treating the ability to model data likelihood as a particular form of decision system especially for cases where the space of "actions" and data are mixed, such as musical scores and written text. This setting is explored in detail in Chapter 8.

1.1.2 Motivation

In this document, I demonstrate methods to improve sequential decision models, where the term *sequential decision models* (Alagoz et al., 2010) is used to broadly encompass generative modeling, structured prediction, and planning.

The primary tie uniting these diverse areas is something I refer to as *configuration uncertainty*, where there is generally not a single correct answer but rather a set of possible correct configurations for the output. The full properties of correct configurations are generally difficult to specify, and part of the output relations must be discovered from weak labels or fully unsupervised losses. The models demonstrated for these areas incorporate several flavors of hierarchical structure driven by domain knowledge, available external conditioning, desired latent variables, and known output constraints.

In terms of techniques, recurrent neural networks (RNN) are used throughout the experiments shown in chapters 4, 6 10. RNNs are also reviewed at a high level in section 2.3. Masked modeling for model invariance and for imposing variable

ordering is discussed in chapters 4, 6, 8 with a review of masking for ordering shown in section 2.4. Attention and self-attention modeling are used in chapters 4, 6, 8, and 10, for incorporating conditional information as well as core modeling. The ideas behind attention are briefly reviewed in section 2.3.4.

This work includes results for several tasks such as harmonic composition of symbolic music, generating speech from text, and probing multi-modal representation learning to better understand speech-text modeling.

Chapters 4 and 6 focus on applications to speech generation and indirectly text modeling, while chapter 10 analyzes speech recognition. Chapter 8 focuses heavily on symbolic music generation with some text generation. Chapters 8 and 10 additionally show analysis of text modeling with chapter 10 heavily focused on the analysis of learned representations between text and speech inputs in a multi-modal speech-text modeling framework.

1.1.3 Document Organization

This dissertation begins by providing a background review for understanding this work in Chapter 2, including beginning with autoencoders, sequence abstractions, masking for variable ordering, and augmentations to probabilistic models. This is followed by chapters on four publications, and a conclusion discussing future research directions.

2 Background

2.1 Background Setup and Notation

2.2 Autoencoders and Latent Variables

Autoencoders are models that are trained to reconstruct their own input (Hinton and Zemel, 1994; Schmidhuber, 2008), using a model typically of the form $f(x) \Rightarrow$, $g(z) \Rightarrow \hat{x}$, $\arg\min_\theta d(x, \hat{x})$ for some loss function d, with $f(x)$ commonly referred to as the *encoder* and $g(z)$ commonly referenced as the *decoder*. To prevent learning trivial solutions (such as the identity function) the target of the mapping $f(x) \Rightarrow z$ is often constrained in some way. One common constraint is to make the dimensions of z smaller than the input dimensions of x, thus forcing compression (Bengio et al., 2009). Another approach is to require the model to solve the denoising problem by adding noise to x and requiring $\hat{x}$ to be the denoised version of the input x (Vincent et al., 2008). Others employ different regularization techniques, whether by contractive penalty (Rifai et al., 2011), to enforce sparse solutions (Olshausen and Field, 1997; Makhzani and Frey, 2013), or by the introduction of a prior and variational regularization (Kingma and Ba, 2014; Rezende et al., 2014). See Goodfellow et al. (2016) for a detailed treatment of autoencoder variants.

2.2.1 VAE

Variational autoencoders (Kingma and Ba, 2014; Rezende et al., 2014) are a specific autoencoder architecture that allows for explicit interpretation of the learned latent variable z. The mechanism by which this interpretation is enforced is a combination of sampled noise in the latent variable and an explicit regularization term in the training loss function. To ensure the ability to backpropagate into the encoder even when sampling (which normally breaks gradient flow), VAE uses the reparameterization trick (Evans et al., 1993) to recast the sampling for a normally

distributed variable $\mathcal{N}(\mu, \sigma) = \mathcal{N}(0, 1) * \sigma + \mu$. The regularization term in the loss function takes the form of a KL divergence between the predicted mean μ and standard deviation σ (produced by the encoder), and a prior assumption about the distribution which is typically $\mathcal{N}(0, 1)$. This regularization term effectively forces the model to trade off reconstruction quality against satisfying the KL divergence penalty. It is possible to use different priors and modified encoder architectures to target other probability distributions, and the VAE framework forms the backbone of methods for stochastic computation graphs (Schulman et al., 2015) and flexible probabilistic computation while still allowing for optimization by gradient descent. For a detailed treatment of VAE methods see Goodfellow et al. (2016) and Kingma et al. (2019).

2.2.2 VQ-VAE

A modification to the standard variational autoencoder (Kingma and Ba, 2014; Rezende et al., 2014), dubbed *vector quantized VAE (VQ-VAE)* (van den Oord et al., 2017) has *discrete* values in the latent space z in contrast to the normal autoencoder which has real valued vectors in z. This change can be inserted into the autoencoder setting seen earlier by the addition of a discretization function $q(*)$; $f(x) \Rightarrow z$, $q(z) \Rightarrow z_q \Rightarrow \hat{x}$, $\arg\min_\theta d(x, \hat{x})$. Discretized latent values have some unique qualities in that they are efficient to use with standard discrete compression methods and have finite expressivity which can reduce the complexity of follow-on processing. However, discontinuities at the boundary between discrete values and flat regions pose practical issues in gradient-based optimization. This motivates the use of gradient approximation methods (Jang et al., 2017; Maddison et al., 2016) and approximate training schemes against a relaxation of the discretized problem.

Approximations for Discrete Activations

Approximate training schemes often take the form of optimizing a smoothed version of the discrete boundary problem, gradually annealing the smooth function to more closely resemble the hard boundary over training. An alternative to the full relaxation approach is to utilize an approximation of the gradient for the true problem (which may not be defined at many points), such as REINFORCE or the so-called "straight-through" estimator (Bengio et al., 2013; Chung et al., 2016;

Courbariaux and Bengio, 2016). In straight-through estimation, a continuous approximate gradient function is often annealed in a similar fashion as relaxation-based approaches, to closer match the discrete function as in Chung et al. (2016). Alternatively, the gradient from the discrete module may be copied directly to the input, bypassing the discretization entirely in the backward pass (Bengio et al., 2013). However, in both cases of straight-through, the forward function remains *exactly* discrete.

VQ-VAE takes a straight-through style approach to this problem, utilizing vector quantization for the function $q(z)$. The vector quantization matches an input activation to the nearest element of a basis (based on $l2$ distance in the original setting) given by a secondary array. In VQ-VAE, this secondary array is initialized randomly and then updated throughout training, similar style to the broader setting of dictionary learning methods (Mairal et al., 2009). In van den Oord et al. (2017) this dictionary update is accomplished via secondary losses similar to the gradient-based calculation of K-means.

The forward calculation of VQ-VAE has no issue, but the lack of a well-defined gradient through the quantization function is a critical problem for standard gradient-based training. VQ-VAE uses the identity formulation of "straight-through", as also shown in Bengio et al. (2013), and on first impression it is surprising that this poor estimate for the gradient of the quantization function works at all. However, a deeper inspection reveals this choice may not be so bad after all.

Because we know that both the activations to be quantized and the basis functions themselves are adapted based on the input data and goodness-of-fit for vector quantization, the problem closely resembles classic K-means. The gradient through the decoder network is directed as if the last encoder activation was one of the K cluster centers in the vector embedding dictionary, rather than its true value. We can imagine an extreme example, where the dictionary contains every activation possible from examples in the dataset. This case means there is *always* one element in the dictionary that has a distance 0 assignment, and the identity gradient approximation is exactly correct.

Next, we can consider the case where some activations are repeated in the dataset. In this case, the story is the same as before, we simply need a smaller dictionary since there are fewer unique activations. This case is further extended by considering activations perturbed by small amounts of Gaussian noise, around

a central activation. Similar to standard vector quantization, as long as these centralized clusters remain well-separated, the error introduced in the gradient for VQ-VAE is still small. It is only when the centroid of the cluster assignment is a poor approximation of the datapoint(s) assigned that the error of this simple estimator becomes apparent.

VQ-VAE Construction

An additional point of interest in VQ-VAE is that the function of the encoder and decoder networks is different than a standard autoencoder. While the standard view works at a high level, most demonstrated implementations of VQ-VAE utilize a convolutional / transposed convolutional structure. VQ-VAE latents are therefore spatially distributed, based on the stride factors present in the convolutional encoders. For images, this means the latent values z are arranged in a 2D grid, and for 1D inputs such as audio the latent space is also 1D. Reducing this spatial latent to a small number of values (or even a single value) is generally too extreme of a bottleneck for effective training, making the default setting of VQ-VAE similar to "all convolutional" CNNs, with a quantized discrete space in the center.

The decoder module of a VQ-VAE is not required to be composed of transposed convolutions, in fact, many of the demonstrated successes in van den Oord et al. (2017) utilize autoregressive decoder networks instead. The design tradeoff between these two approaches is generally a choice for more computation at reconstruction time for the autoregressive decoder but higher potential quality, or faster generation at the expense of some quality loss. In my work, both types of network architectures were tested, but the transposed convolutional decoder's computational efficiency was crucial for my application to spatio-temporal modeling.

VQ-VAE Latent Space

This also means that the latent space, while discrete, isn't compact or compressive in the usual sense (for example, standard VAE (Kingma and Welling, 2013)). The latent space is comprised of indicator values, indexing the vector embedding space. Based on the effective receptive field of the decoder network the individual integers can't be reasonably manipulated, swapped, or interchanged as the output of the decoder is dependent on all the input values which fit in the receptive field.

In other words, the same cluster vector or integer index has a different meaning depending on its position in the spatial latent z. Though there may be possibilities of "bit flipping" to the next nearest cluster, or performing analysis on cluster distances directly this has not been demonstrated to date.

The lack of "neighborhoods" in the latent space or a simple method for sampling valid configurations of the latent may seem to relegate VQ-VAE to a strictly compressive setting rather than the generative setting where standard VAE methods are used. However, combined with a powerful autoregressive generative model known as PixelCNN it is possible to train a "prior" over the latent space, generating valid z configurations from scratch which are then decoded into valid generative outputs.

VQ-VAE is an area of active research, and many papers have demonstrated alternative dictionary update schemes (Roy et al., 2018; Kaiser et al., 2018), improved gradient approximation schemes with soft assignment (Sønderby, Poole, and Mnih, Sønderby et al.), as well as improvements to the fundamental model to better capture extremely long-range dependencies (Dieleman, van den Oord, and Simonyan, Dieleman et al.) or better association in the latent space (Graves et al., 2018). Combined with the autoregressive techniques we will discuss later in this document, VQ-VAE is becoming a key method for compression and generative modeling which is computationally efficient, easy to train, and useful for a variety of data modalities.

2.3 Sequence Abstractions

Markov models are among the simplest of models for sequences. In their base form, they assume that an input datapoint x with feature dimensions n can be considered ordered in some way. This ordering can be chosen arbitrarily, but many types of sequences such as timeseries have a natural ordering which is commonly assumed to be along the "time" dimension. Markov models assume a fixed size context of past information k, resulting in a per-step model of the sequence $x_i = f(x_{i-1}, x_{i-2}...x_{i-k})$ with some arbitrary function $f(*)$. This factorization makes an ideal setting for modeling conditional probability distributions like

$p(x_i|x_{i-1}, x_{i-2}...x_{i-k})$ which is an identical form to a single layer convolutional neural network with a shifted (or causal) filter window (van den Oord et al., 2016). A generic description for $f(*)$ describes a host of classic signal processing techniques, such as finite impulse response (FIR) filters, moving average smoothers, and venerable neural network approaches to timeseries such as time-delay neural networks (TDNN) (Lang and Hinton, 1988; Waibel, 1989). Since $f(*)$ is general, this also covers compositions of functions (such as deep convolutional neural networks) for sequence modeling (Fan et al., 2018). However given that any given layer has no sense of "history" and only a finite context, stacks of these layers combined may have larger receptive fields than each member layer (Yu and Koltun, 2015) but will always have a finite input receptive field (given finite layer depth).

Recurrent neural networks are another approach to sequence modeling that takes inputs with ordering and computes a function taken over the sequence. However, in addition to the input context, recurrent neural networks also define an accumulator, often called the "hidden state", which continually aggregates information about the current frame. This gives the RNN a theoretically infinite context, though in practice the fact that the aggregate information must be stored in a fixed-size vector means the memory is lossy and compressive by nature. Breaking this computation into two generic functions $f(*)$ and $g(*)$, we now see $h_i = g(h_{i-1}, x_{i-1}); x_i = f(h_i) \forall i \in n$, with h_i representing the hidden state calculated at step i, which captures information about $x_0, x_1, x_2...x_{i-1}$. This makes RNNs ideal for modeling the full conditional probability per-step $p(x_i|x_{i-1}, x_{i-2}...x_0)$, indirectly via $p(x_i|h_i)$.

2.3.1 RNN Families

Describing RNNs at this high level means there are numerous ways to actually implement $f(*)$ and $g(*)$. The most basic form of an RNN is known by multiple names including the *Elman RNN* (after the first author in Elman and Zipser (1988)), *tanh RNN*, or *simple RNN*. The Elman RNN is calculated with a simple hyperbolic tangent (tanh) activation function applied to the summation of two feed-forward activations projecting to the same dimensionality which are then summed. This calculation for the hidden state for a given timestep can be summarized as $h_i = \tanh(h_{i-1}W_h + x_iW_f + b)$, with the initial hidden state to start the recursive

calculation being initialized to either all 0, or small random values from a random or normal distribution. Furthermore, the initial hidden state can potentially be treated as another set of parameters and optimized during training, but common usage is to preserve the default initialization without learning. Note that there is no need for two separate bias terms, as one learned bias vector is sufficient to capture any additive constant.

It is well known that learning recurrent connections by gradient descent is difficult (Bengio et al., 1994; Bengio et al., 2013; Sutskever et al., 2013), and numerous methods have been developed for improving long-term credit assignment in RNNs. Two of the most well-known methods for this are the long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997a; Hochreiter, 1998; Gers, 2001) and gated recurrent unit (GRU) activations (Cho et al., 2014; Chung et al., 2014). These compute alternate forms of the recurrent neural network in a way that is more amendable to gradient descent, reducing vanishing and exploding gradient issues which are crucial to learning neural networks well. As shown by Bengio et al. (2013), repeated matrix multiplication (taking a matrix to a power M^p) will, depending on the spectral norm of the hidden matrix, either lead to vanishing or exploding. This problem is made worse the larger the power of the matrix (given the same spectral norm), which means longer sequences will have problems with vanishing or exploding.

Equation 2.1 LSTM step equations

$$
\begin{aligned}
f_i &= \text{sigmoid}(W_f x_i + U_f h_{i-1} + V_f c_{i-1}) \\
g_i &= \text{sigmoid}(W_g x_i + U_g h_{i-1} + V_g c_{i-1}) \\
\hat{c}_i &= \tanh(W_c x_i + U_c h_{i-1}) \\
c_i &= f_i \odot c_{i-1} + g_i \odot \hat{c}_i \\
o_i &= \text{sigmoid}(W_o x_i + U_o h_{i-1} + V_o c_i) \\
h_i &= o_i \odot \tanh(c_i)
\end{aligned}
$$

These equations are quite detailed, but on close inspection some high-level themes emerge. There are a large number of sigm activations, most of which form "gates" for information into, or out of the RNN. This is crucial to controlling

Equation 2.2 GRU step equations

$$
\begin{aligned}
z_i &= \text{sigmoid}(W_z x_i + U_z h_{i-1}) \\
r_i &= \text{sigmoid}(W_r x_i + U_r h_{i-1}) \\
\hat{h}_i &= \tanh(W_h x_i + U_h(r_i \odot h_{i-1})) \\
&\text{or} \\
\hat{h}_i &= \tanh(W_h x_i + r_i \odot (U_h h_{i-1})) \\
h_i &= (1 - z_i) \odot h_{i-1} + z_i \odot \hat{h}_i
\end{aligned}
$$

information flow, and splitting information flow into several segments allows the gradient to essentially skip backwards steps. In the case of GRU, if the gate z_i is close to 1, the current proposal activation h_i is largely ignored. The LSTM features several gates, but we also observe that they all control the flow of information into, or out of, the cell state c_i allowing for similar improvements in gradient flow. The cell state is also computed through the summation of two separate inputs c_{i-1} and $\hat{c}_i$, which further improves the flow of gradients backwards. LSTM has two types of state information, in h and c which must be preserved to form the state of the LSTM at a particular timestep. The primary drawback of these methods is an increase in parameter count due to the additional linear projections for gating activations. This results in 3 times more parameters in GRU for the equivalent output, h_i size in simple RNN, and 4 times in the case of LSTM. However, both of these methods see widespread use in practice and are generally the default activations employed when implementing RNNs for various applications. There have also been numerous metastudies discovering alternate architectures for RNN activations (Greff et al., 2015; Jozefowicz et al., 2015), which have found interesting variants in gating architectures which have shown improvement in certain cases and demonstrated the importance of initialization for LSTM and GRU.

2.3.2 Conditional RNN

RNNs are also capable of modeling per-step conditional distributions of the form $p(x_i|x_{<i}, m)$ for some conditioning information m. The simplest form of this is concatenating the conditioning vector m to every input vector x_i, but this can also be a summation (of the correct dimension) from either the vector directly, or a projected version that matches the necessary size. These two methods are a common way to merge information paths in neural networks, but worth reiterating here since they form the core of the following approaches. If the conditioning information is also a sequence of the same length as x, it can be directly modeled by combining the correct timestep vector from m per step, forming a conditional $p(x_i|x_{<i}, m_{\leq i})$. This is because the hidden state h captures all past information about $x_{<i}$ *and* $m_{<i}$, since m is another input, just like x. With these techniques, RNNs are extremely flexible methods for forming directed, conditional generative models. Because the hidden state is a summary of past information, we can also consider using the last hidden state as a compressed representation of the vector that the RNN processed. This means it is also possible to use the last state directly as a representation to classify whole sequences, or alternately use a function that combines all hidden states (such as a sum or a mean) for the representation vector used in follow-on processing.

Autoencoders, as discussed earlier, are a common approach to summarizing, compressing, and understanding information. What would a sequence autoencoder look like? We know the input needs to handle sequences. The output should be a sequence and it should also compress to a fixed size latent. Combining two RNNs with a compressive section should form an autoencoder. Recall the previous discussion regarding the last hidden state compressing (or *encoding*) the modeled sequence. Taking the last hidden state should form a compressive summary of a sequence, which is the same size no matter the length of the input. This latent variable, $h_{e[k]}$, represents the encoder hidden state h_e using the last element index k. The variable, $h_{e[k]}$, is considered to be directly equivalent to the latent variable, z, discussed before. z can be specially calculated based on $z = f_z(h_{e[k]})$, allowing for natural extension to many of the various autoencoder families such as VAE. The decoder can be formed using a conditional RNN where the conditioning vector is the encoded sequence representation. This leads to an overall formulation of $p(x_i|x_{<i}, h_{e[k]})$ using the decoder's conditional RNN, where $h_{e[k]}$ is the last hidden

state of the encoding RNN. The repetition of $x_{<i}$ in the conditioning is redundant in theory (since $h_{e[k]}$ already contains a compressed form of the same information) but this statement is closer to the practical implementation. In its base form, this approach can alternately be called encode-decode RNNs or sequence to sequence (seq2seq) (Cho et al., 2014; Sutskever et al., 2014) and it has seen widespread application in sequence modeling. This approach can also be extended to VAE-style latents (Bowman et al., 2016; Ha and Eck, 2017; Jaques et al., 2018; Roberts et al., 2018).

This model setup has applications far beyond autoencoding. Noting that the encode RNN and decode RNN are not restricted to be the same length or even the same sequence between encoder and decoder, thanks to passing through the compressive bottleneck, the seq2seq model can be seen as a general approach to sequence transduction (Graves et al., 2013; Graves and Jaitly, 2014). This can be seen as modeling a sequence y, step-wise as $p(y_i|y_{<i}, h_{e[k]}) \equiv p(y_i|h_{d[i]}, h_{e[k]})$ where $h_{e[k]}$ contains all compressed information for a sequence x, and $h_{d[i]}$ is the decoder hidden state capturing the current dynamics of the output sequence (and its past interaction with $h_{e[k]}$). This general formulation can be used for machine translation (a sentence in one language as x, the translated version of that sentence in another language y), speech recognition (input audio x, desired recognition symbols y), text to speech (input text x, output audio y), handwriting generation (input sentence x, output handwriting trace y), and many other applications.

2.3.3 Attention

A key issue remains, as shown in Sutskever et al. (2014) performance begins to drop for sequences of length 40 or more using these standard tools. This makes sense, as we can easily believe that compressing long sequences into a fixed-size vector could be more difficult than compressing short ones (depending on the sequences in question). In addition, using only the last hidden state may bias the conditioning vector $h_{e[k]}$ toward information at the end of the input sequence. Reversing the input sequence (Sutskever et al., 2014) may help this, but ultimately we desire different ways to process the input, so that different elements of the input sequence are reweighted in an automatic way. This is the fundamental idea behind attention-based sequence to sequence modeling (Bahdanau et al., 2015; Graves,

2013a). By introducing different aggregation functions (which are generally dynamic at every step) over the input hidden states, $c_i = a(h_{d[i]}, h_e)$, per-step conditions can be extracted from the input hidden states. This allows for a new model which flexibly uses the full input context $p(y_i|y_{<i}, h_e) == p(y_i|h_{d[i]}, a(h_d[i], h_e))$. When the function, a, uses neural network weights defined by the decoder such as softmax distributions (Bahdanau et al., 2015) or Gaussian distributions (Graves, 2013a), the network is able to learn to perform dynamic weighting solely through minimizing the output loss. Introducing attention greatly improves performance in neural machine translation systems (Bahdanau et al., 2015) when compared to non-attentional systems, while also boosting performance on longer sequences. A huge variety of attention modules are possible, depending on the application domain and inductive bias for the problem at hand (Graves, 2013b; Bahdanau et al., 2014; Martins and Astudillo, 2016; Niculae and Blondel, 2017; Niculae et al., 2018; Deng et al., 2018; Tachibana et al., 2017; Chorowski et al., 2015; See et al., 2017; Peng et al., 2018; Mensch and Blondel, 2018; Vaswani et al., 2017).

2.3.4 Attention-centric Models

The introduction of soft attention and alignment mechanisms has opened the door to architectures specifically designed using these features for variable length input and output sequences. The core of these models can be based on recurrent networks, as described in the previous subsection. They can also be convolution based, using dynamic convolution kernels to align over a variable length sequence (Wu et al., 2019). Encoder and decoder networks need not share the same building blocks, and many ideas from autoencoding can be used to learn more concise summaries of input data, which are then attended over or used to condition autoregressive decoding (Bowman et al., 2016). Driving these design choices to their limit, models which aggregate context over depth (similar to convolutional networks), but with full-context layer-wise locality have come to the forefront of sequence modeling (Vaswani et al., 2017; Bradbury et al., 2017; Radford and Wu, 2019; Radford et al., 2018). The most prominent variant of this idea for "attention-only" modeling of sequences can be seen in the Transformer architecture (Vaswani et al., 2017). A transformer variant for generative modeling can be seen in Chapter 8.

2.3.5 Beam Search

A critical component to directed generative models over sequences, particularly sequences of symbols, is *beam search* (Reddy, 1977; Bahdanau et al., 2015; Cho et al., 2014; Auli and Gao, 2014; Sutskever et al., 2014). Similar to Viterbi search (Viterbi, 1967), beam search allows for more likely sequence proposals under the factorized model used during training. The primary difficulty is that during training, we have ground truth information with which to optimize the model and break step-wise dependencies. However, at evaluation time our model only produces step-wise probabilities, meaning for any step, there will V candidates to choose from. Computing the next set of V candidates requires input of the previous step value, which at evaluation is chosen from a candidate set as well. The optimal (but infeasible) method to compute the most likely sequence is to evaluate every possible permutation at every step S, leading to S^V candidate sequences which could then be ranked by likelihood under the probabilistic model. It is clear that this amount of computation is infeasible for even moderate sequence lengths and candidate sizes. Alternatively, it is simplest to take the most probable candidate at every step in a *greedy* fashion. This leads to a computational complexity that is approximately linear in the number of steps. However, we can see it may be possible that a greedy best candidate with bad follow on probabilities could be much worse in overall sequence likelihood than another candidate which leads to many higher probability follow on steps. This result is similar to Viterbi search, breadth first search, and depth first search. Unlike Markov models, RNN based models do not make local independence assumptions, so using Viterbi search will not be guaranteed to give the optimal sequence for the model factorization described by standard RNNs.

We can instead adopt procedure similar to Viterbi search called *beam search*, where a number of beams, b, are created using the top b most probable elements for step 0 of the sequence based on the probabilities assigned by the neural network. We then evaluate each of the b prefixes, getting next step probabilities from the neural network, resulting in $b \times V$ proposed prefixes for the next step. Once all proposed prefix sequence probabilities are evaluated, they are ordered by probability and the top b highest-probable proposed prefixes become the new base prefixes. This is repeated until one of the b prefixes is recognized as being finished (often through use of a special symbol for end of sequence, or EOS). The beam capacity is then reduced from b to $b-1$, and the finished beam is preserved while the others continue.

This proceeds until there is no capacity left at which time, all b beams are marked complete. Once all beams are complete, they are again ranked according to highest probability and the best beam is returned.

One common modification is to normalize these beam probabilities based on length as depending on implementation ending a beam early can have an artificially high probability compared to longer sequences, as multiplying two probabilities will always result in a smaller value leading to inaccuracy in the final reranking of the b beams (Wu et al., 2016). It is also well known that likelihood is not a direct correlate for quality (Theis et al., 2015), so even though beam search can give higher probability sequences under the model it is no guarantee that these sequences will actually be good or useful. This can incentivize returning all top b beams directly to the user, and allowing them to select the most appropriate result. There are a number of methods for incorporating knowledge of the decoding procedure back into RNN training (Wiseman and Rush, Wiseman and Rush; Chorowski and Jaitly, 2016; Gu et al., 2017). These include scoring to improve proposal diversity (Vijayakumar et al., 2016; Li et al., 2016), adding stochasticity to the decoding procedure by sampling the top proposal prefixes, using noise directly to approximate beam search using greedy decoding (Cho, 2016), and using heaps with additional heuristics for A* search style decoding. The beam search procedure has strong ties to many classic methods for discrete optimization, which can further improve result quality given known constraints or secondary scoring methods beyond RNN probabilities.

2.4 Masked Conditional Modeling

The addition of conditional information to a modeling task can radically change (and often, improve) performance of practical systems. In more exact terms, it can be easier to estimate $p(x|c)$ instead of $p(x)$, if c contains useful information for separating groups or reducing the space of solutions for x. For example considering x as a word in the English language, a word probability modeling task becomes simpler if changed to $p(x|c)$ where c is a qualifier such as "is the name of household product" or "a picture containing the word x", or any other form of information normally used as input to machine learning systems. This induces a requirement

that c be provided to the system in order to predict a probability for x, but in many applications a wide array of side information may be available to improve modeling capabilities.

Information can also be factorized to form conditional distributions, in addition to the external conditioning above. Using the chain rule of probability, it is always possible to rewrite a joint probability as a product of conditional probabilities $p(x_1, x_2, x_3...) = \Pi p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)....$ This means that any probabilistic model of some information $p(x)$, which can in turn be broken into sub-parts $x_1, x_2, x_3, ...$ can be rewritten as a product of conditional distributions. Often this factorization can make modeling problems easier, as instead of learning probabilities in one shot, a piece of the overall example such as x_3 can be modeled based on other constituent parts of x, such as x_1 and x_2. Realizations of this factorization could be modeling images pixel-by-pixel, audio content sample-by-sample, or sentences one character or word at a time.

Recurrent neural networks are built around this factorization, as well as other autoregressive models such as NADE, MADE, RIDE, and Wavenet among others (Larochelle and Murray, 2011; Germain et al., 2015; Theis and Bethge, 2015; van den Oord et al., 2016; Domke et al., 2008). The primary downside of this factorized approach is the introduction of a dependence chain that is linear in the number of factorized pieces. Without independence assumptions, this means that sampling from such a model is on the order $O(N)$, where N is the number of factorized conditional distributions modeled. Depending on the model structure, this may also induce computational overhead at training as is the case for RNNs compared to the other mentioned autoregressive models.

The *self-factorized* approach can also be combined with the *external conditioned* method described beforehand, which results in probability factorizations such as $p(x_1, x_2, x_3|c) = \Pi p(x_1|c)p(x_2|x_1, c)p(x_3|x_2, x_1, c)$. This allows for combining secondary knowledge as well as internal structure, which can greatly improve modeling capabilities for likelihood estimation as well as sampling. These factorizations also have direct impact on model architectures, and are closely related to how a particular modeling problem is formulated in the design stage.

2.4.1 PixelCNN

A particular line of work which extends NADE and MADE, using specially structured RNNs and CNNs is a pair of models known as PixelRNN and PixelCNN, from van den Oord et al. (2016) and van den Oord et al. (2016). These models combine the idea of using masks to define factorized distributions with RNNs and CNNs, resulting in a powerful model for autoregressive generative modeling. Of the two methods, PixelCNN and follow on work such as PixelCNN++ (Salimans et al., 2016) have proven excellent methods for conditional or unconditional generative modeling, especially for images or other datatypes with spatial structure (Huang et al., 2017a, 2018). Key details of the PixelCNN model can be seen in Figure 2.1, Figure 2.2, and Figure 2.3.

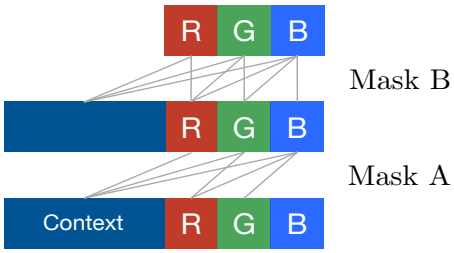

Figure 2.1 – Figure from van den Oord et al. (2016) showing autoregressive masking over space and channels as layer depth increases.

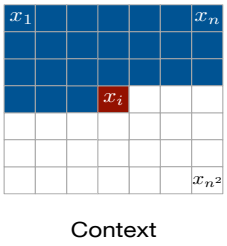

Figure 2.2 – Figure from van den Oord et al. (2016) showing ideal captured context, using combined masks and additional architecture changes.

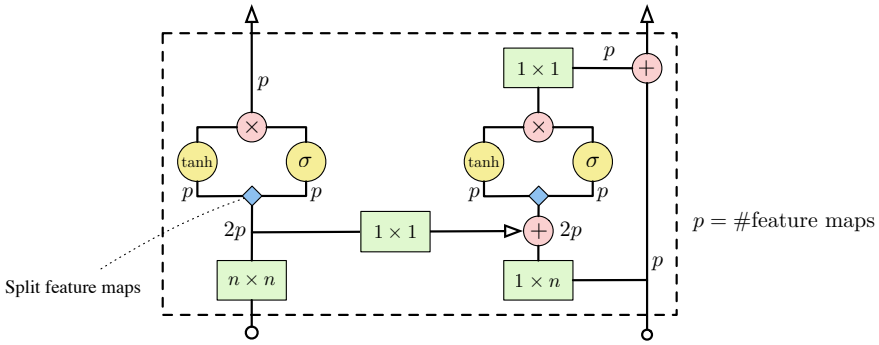

Figure 2.3 – Detailed drawing of Gated PixelCNN layer from van den Oord et al. (2016)

Viewing Figures 2.1 and 2.2 specifically, we see that the mask definitions define a conditional distribution wherein an image x with i columns of pixels, j rows,

and k channels is carefully masked (using a multiplication of the input with a same size *image* of 0s and 1s). Spatial and channel indexes into this image can be understood as $x_{i,j,k}$. The description of the masks over the channel axis $x_{:,:,k}$ is shown in Figure 2.1. The additional block marked *context* represents the generic ability to incorporate side information such as class identity, a spatial map (such as previous input image for video modeling), or both - this is an important feature, but doesn't particularly interact with the masking process due to the design of the architecture.

2.4.2 Mask Structure

Mask A defines a careful multiplication such that the center pixel of the input (since this is a form of convolutional neural network) only has access to information to *previous* channels in the order RGB, meaning a partial factorization of $p(x) = \Pi p(x_{:,:,r})p(x_{:,:,g}|x_{:,:,r})p(x_{:,:,b}|x_{:,:,gr})$. This is then combined with a secondary causal masking structure which only allows access to pixels left of and above the center pixel in the convolutional filter. Specifically for a center pixel $x_{i,j}$, this results in a factorization over spatial position and channels $p(x) = \Pi p(x_{<i,<j,r})p(x_{<i,<j,g}|x_{\leq i,\leq j,r})p(x_{<i,<j,b}|x_{\leq i,\leq j,gr})\forall i, j, k$. Mask B is then used in subsequent layers. Mask B defines a very similar factorization as Mask A, with the primary difference being a subtle change from $<$ to $\leq$ on certain terms $p(x) = \Pi p(x_{\leq i,\leq j,r})p(x_{\leq i,\leq j,g}|x_{\leq i,\leq j,r})p(x_{\leq i,\leq j,b}|x_{\leq i,\leq j,gr})\forall i, j, k$.

Mask A is applied in the first layer, and there is also a hard requirement *not* to use a residual connection (He et al., 2016a) from the input, as this would violate the causal factorization applied by the masking. After the input mask, every subsequent layer can use Mask B and optional residual connections, since the first layer preserves the causality of the masking factorization. Each convolutional layer performs local aggregation (typically 3×3, 5×5, or 7×7) but with a sufficiently deep network each output pixel should have a local receptive field (Sermanet et al., 2014) which can cover the entire input size, which combined with masking will exactly capture the necessary inputs for the factorization implemented. The specific form of the computation applied can be seen in Figure 2.3, and at a high level the model has 2 components per layer - a horizontal stack and a vertical stack. This is necessary to capture the idealized context as seen in Figure 2.2, for more detailed

discussion of the layer design and secondary context interaction, see (van den Oord et al., 2016).

A critical benefit of PixelCNN based methods centers on the use of *teacher forcing* to greatly improve speed during training, compared to unrolling of the probability factorization and tracking history using methods such as RNNs. In brief, the factorization $p(x_1, x_2, x_3...) = \Pi p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)...$ discussed earlier can be computed in two ways. The first, naive way is to calculate each partial action one by one, using the predicted outputs of each factor as input to the next. The second, if given training data, is to use the parts of the training data to make the evaluation of each part of the factorization *independent*, allowing the otherwise sequential factorization chain to be computed in parallel during training.

2.4.3 Teacher Forcing

Mathematically, this might look like $p(x_1, x_2, x_3...) = \Pi p(x_1)p(x_2|{\sim}x_1)p(x_3|{\sim}x_2, {\sim}x_1)...$, where ${\sim}x_i$ marks the fact that these elements (be they pixels, words, etc.) are the exact pieces from the training input, rather than a sequence of samples from the previous distributions in the chain as would be implied by the original formulation. This parallelization during training is not possible in RNNs, as the normal formulation is instead $p(x_1, x_2, x_3...) = \Pi p(x_1|h_0)p(x_2|{\sim}x_1, h_1)p(x_3|{\sim}x_2, h_2)...$, with $h_1 = f(h_0)$, $h_2 = f(h_1)$ and so on. This means that RNN factorization in a similar way as PixelCNN would still be approximately linear in the unrolled length, where the PixelCNN is effectively $O(1)$ during training due to parallelism from the *teacher forcing trick*.

Using training data to form independent sub-parts, which can then be evaluated in parallel during training for improved speed is an extremely powerful idea, which sees extensive use in a variety of settings. This issue is discussed further by (Goodfellow et al., 2016), in the section on *Structured Probabilistic Models for Deep Learning*. However, the PixelCNN model structure and "teacher forcing trick" do not change the dependency chain at sampling time, meaning autoregressive models are typically $O(N)$ during generation where each pass of N is a full forward evaluation through a typically very deep convolutional neural network - meaning the act of generating from such models is often prohibitive computationally, on a similar order as RNNs but generally requiring much larger models in terms of parameter

count.

There are extension methods (van den Oord et al., 2017) which use a discretized autoencoder structure to first compress the input, then use PixelCNN as a prior over the spatially reduced representation for sampling, rather than directly modeling input examples. This can greatly speed up sampling times for PixelCNN based methods, which are generally linear in the number of pixels $\times$ channels at generation time. Even small inputs (for example $32 \times 32 \times 3 = 3072$ neural network forward passes) form an extremely long sequential sampling chain as factorized by this model, making sampling speed a crucial aspect for fautoregressive generative modeling. We discuss the use of discretized and variational autoencoders in Section 2.2.2. There also exist computation methods related to memoization which can greatly improve the generation speed of PixelCNN type models (Ramachandran et al., 2017), which help independent of most modeling improvements and combine well.

2.4.4 Order

Putting aside computational issues, there is a secondary concern with "chain factorizations" commonly seen in autoregressive models. Imagine we train an autoregressive language model in the style of PixelCNN which takes an English sentence and factorizes it at the word level in left-to-right order,
$p(s) = \Pi p(w_1)p(w_2|w_1)p(w_3|w_2, w_1)....$ During training, we use the *teacher forcing trick* to parallelize training, and everything is training well. The generation procedure then *samples* words one by one, dependent on all previous words sampled that fall within the input receptive field for the current output to sample. However, we may randomly sample a particularly strange word near the end of the sentence, which is incongruous with the previous generations. It is clear that the choice of what order to factorize a given input is extremely important in practice, but how do we choose the best ordering for the factorization for a particular data format?

If we had an iterative or multi-pass sampling procedure, it will involve a more complicated, multi-order training but it would then be possible to resolve possible errors related to sampling two outputs which don't agree in the first pass. The resulting sampling would proceed in a fashion very similar to Gibbs sampling or coordinate descent (Bishop, 2006), repeatedly sampling some piece conditional on

all other existing knowledge, and proceeding in a random or round-robin fashion until convergence or boredom. This should allow for outputs which have stronger global agreement, as context is no longer causal but dependent on *all* known information. Many creative tasks such as drawing or music composition rarely involve an ordered one-pass approach to content creation, instead preferring a multi-step process of exploration, rough sketching, adjustment of component pieces, and final cleanup. This is a strong intuition that for the end goal of content generation, a one-pass ordered sampling procedure may not be the ideal method to obtain high quality results.

This intuition has driven many methods which leverage iterative inference in generative modeling (Gregor et al., 2015a; Huang et al., 2017a; Hadjeres et al., 2016). In particular, CoCoNet (Huang et al., 2017a) leverages a PixelCNN style architecture and additional data augmentation to train a model which is order invariant (as described by Orderless NADE (Larochelle and Murray, 2011)), resulting in a model which learns marginal and conditional distributions for *any* ordering of the data in expectation.

2.5 Augmenting Decision Making in Probabilistic Models

Though we train probabilistic models for many tasks, the end deployment of these systems is generally in decision making processes, where there *must* be a decision or action made. Machine learning systems often compute and optimize a relaxed *surrogate* for this decision process, for more details on the exact nature of this relaxation see the discussion of empirical risk minimization by Vapnik (1991).

Consider a dataset generated by a cyclic graph of the form seen in Figure 2.4. This dataset fundamentally always transitions from A to B, from B to C, and from C back to A. We can train on any offset or chunk size of this infinite sequence (e.g. [A, B, C], [C, A, B], [B, C, A]), and *never* see certain transitions in the training set. If we believe the test set and validation set are *random draws* from the same underlying distribution, a usual assumption in machine learning (Bishop, 2006), this means that transitions like A to C or C to B should have a 0 probability under

the model, or in other words, a 0% chance of occurring when sampling from the distribution defined by a model trained on this data.

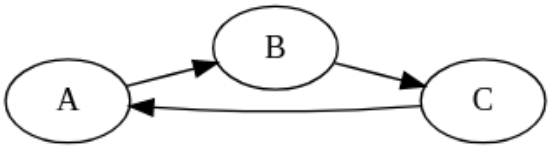

Figure 2.4 – Example cyclic generator for dataset "A, B, C, A, B, C, A, ..."

2.5.1 Learning and Assignment

Training a model on this dataset using standard maximum likelihood methods, such as a neural network which takes as input the previous symbol and is expected to output the probability of the next symbol, with a categorical cross-entropy loss over the set of potential symbols uncovers a fundamental difficulty. Maximum likelihood methods generally require some initial non-zero probability assignment for every category, but we can always decrease the probability assigned to a particular output and increase the probability of another output (ideally the correct one). However, we can *never* set the probability of some event to 0 (or within machine tolerance of 0, practically) due to numerical issues in the calculations made during modeling.

This can be seen specifically in the logarithm computation terms in most common losses such as binary or categorical cross-entropy, though this issue applies more broadly to maximum likelihood methods using gradient based optimization. More advanced modeling methods such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) cannot directly fix this issue, as it is a direct consequence of the loss structure chosen and the use of maximum likelihood training criteria (Goodfellow et al., 2016).

2.5.2 Markov Masks

Markov models or other counting based methods have no issue assigning 0 "probability" to transitions never seen in the dataset. However by assigning truly 0 probability to an event, we prevent ever utilizing that event in a given context, possibly preventing *generalization* to new situations if the train and test datasets

differ in some fundamental way (as they often do in practice). Is it possible to have the best of both, assigning some small probability everywhere and training with convenient relaxations, but still avoiding unreasonable decisions at evaluation time?

A simple correction for this particular task would be to change the symbols modeled, rather than having a symbol vocabulary of [A, B, C], to modeling transitions [A:B, B:C, C:A]. Though this solution is effective here, it can sometimes be difficult to model edges directly, due to increases in vocabulary size from the potential n^{n-1} pairwise connections in an n node graph (Belilovsky et al., 2016; Koller and Friedman, 2009).

A keen observer will also note that the previous "solution" still allows for impossible and disconnected predicted sequences such as [A:B, C:A, B:C]. There exists a simple solution to this problem, namely taking the highest probability output which is connected to the existing input, rather than naively taking the highest probability without any checking.

This change works well on the task as defined above, however things become more complex in cases with multiple valid transitions between nodes. We can generally consider taking the "top-K" outputs, rather than just the "top-1", taking into consideration the predictions made before and the current input. A form of this method, known informally as "beam search" (Reddy, 1977; Cho et al., 2014; Bahdanau et al., 2015) and discussed in Section 2.3.5, has widespread use in deep learning and in the broader structured prediction community as well as ties to Viterbi decoding and dynamic programming (Viterbi, 1967; Bellman, 1957). Recent work in the research community has begun exploring the combination of deep learning and methods which include search as a part of the decoding procedure (Vinyals et al., 2014; Freitag and Al-Onaizan, 2017), or even methods which are focused directly on search driven by modeled heuristics using neural networks (Pachet et al., 2011; Silver et al., 2016, 2017).

We can combine the benefits of Markov methods (via masking) with standard training procedures, using methods similar to (Germain et al., 2015; van den Oord et al., 2016; van den Oord et al., 2016; van den Oord et al., 2016). The high level procedure is masking per-step predictions in a sequential model by a 0-1 mask, with 1 where a transition had a count of at least 1 in the original data. The chosen mask can additionally be given to the model as an auxiliary input, allowing

the model to have a better conditional context with which to predict probabilities for each output. This masking procedure introduces some of the same issues as Markov models regarding generalization, but does allow for flexible models which can choose from within the available *good* options, which are *guaranteed* to have existed in the original data by design of the masks themselves.

2.5.3 Limitations of Constraints

There exists a tension between the design of models which enforce structure and models that generalize well to new data, especially if there is a distribution shift between training and evaluation settings. This struggle is somewhat consistent with previous choices regarding inductive bias in model design, though in the case of constraints a particularly bad choice of enforced constraint can result in models which *never* capture the intended solution. These tradeoffs should not be taken lightly, but given the ability to flexibly enforce constraints with a wide variety of methods, designers should have adequate leeway to craft a model which encompasses the solution set for the task at hand.

If we can more readily incorporate information about decision making processes directly in the training loop or as post-hoc correction, it may enable better performance, introspection, and safety guarantees in end deployment. Integrating rules (in a meaningful space for their definition by humans) can leverage years of past domain expertise in the design of expert systems, and combining these directly is key for applying deep learning methods to real world problems which require sequential decision making, planning, and reinforcement learning.

2.6 Speech Synthesis

The works in this dissertation develop tools and techniques related to text-to-speech synthesis (TTS) which involve a combination of expert signal processing techniques and learned hierarchical models. Though a full background of TTS methods are outside the scope of this document segment, there are several high level lines of research which are applicable to this discussion. Fundamentally, the problem of text-to-speech involves mapping an underspecified sequence of symbols

to a detailed representation of audio which can either be converted directly into audio or a representation of the raw audio. For example in English TTS, the symbol input might be *"Don't desert me here in the desert!"*, and the corresponding audio sequence a ten second long audio recording of the sentence. Pronunciation, timing, and speaker style are all unknowns when looking at the text alone, meaning the learned model must inject these factors when generating, and preserve their qualities (but vary them realistically) over the length of the sequence or sequences to generate.

This high level information can optionally be combined with side information about speaker type, per-timestep pitch, or many other performance level annotations. These annotations generally reduce the variability of the input to output mapping, but require careful labeling for accuracy. These labels must be captured from expert hand-labelers or generated from approximate algorithms that were developed by professionals in fields like digital signal processing (DSP) or natural language processing (NLP).

Historically there are a wide variety of approaches to TTS (many of which are language dependent), but there are two broad categories of solutions, *concatenative* TTS and *paramteric* TTS. *Concatenative* TTS is focused on creating speech by combining atomic units into an overall audio sequence, generally through a combination of heuristic search and audio-specific tools for handling boundary issues, fine timing, and overall alignment. *Parametric* TTS focuses on control of a parametric output unit. This allows for fine-grained changes and (theoretically) generalization outside of the atoms found in the training corpus. Despite the promise of generalization, parametric TTS tends to be much more sensitive to the dynamics of the control unit when compared to concatenative TTS approaches. These two settings are roughly equivalent to word level language modeling tasks versus character level.

2.6.1 TTS Components

Beyond these differences, another important component is the factorization and granularity of the TTS system. Breaking the task into three high level components (separate from the parametric/concatenative distinction), core components are revealed: text processing, audio processing, and text to audio agreement. Study of these parts can be a field unto its own (text: NLP and audio: DSP), but using

a few choice tools from their respective fields can greatly simplify the process of learning text to audio agreement, which is one of the key parts for data driven TTS.

Text processing for TTS can be extremely involved, with optional features specific to languages, speaker timing, and dialect. Advanced features here can reduce or eliminate variability in the output audio, and careful text processing is crucial to high quality synthesis. For neural TTS specifically the input generally has minimal feature engineering, instead pushing much of the complexity of timing estimation to the text to audio agreement layers which come deeper in the network. Text input sequences are presented as characters, bytes, byte pairs, or phonemes. Phonemes are a sound aware alphabet, commonly used in language understanding. Normal choices for this include the International Phonetic Alphabet (IPA) or ARPABet, but extracting these elements requires performing automatic speech recognition (ASR) over the input data. This introduces an implicit second step between characters (more generally called graphemes) and phonemes, or g2p. Pronunciations from g2p can be predefined, or learned by a separate machine learning model such as a seq2seq model. Phonemes are a particularly large improvement for English TTS, as pronounciation is particularly tricky (for example, *"Don't desert me here in the desert!"*), but other languages such as Spanish, or Romanian may be more pronounceable directly from characters, or have romanized versions such as *pinyin*. Alternatively characters, bytes or byte pairs can be input directly, forcing the model to implicitly learn g2p as part of the overall g2a (grapheme to audio) transformation.

Audio processing for TTS can also be complex with a huge range of transforms both lossy and perfectly invertible available for use. These approaches were primarily developed by the DSP and speech research communities using application specific methods, and many of these specific methods have been generalized and abstracted as machine learning models. There are two primary types of transforms used in neural TTS, speech agnostic time-frequency transforms, and speech specific vocoder transforms. Speech agnostic methods are built of time-frequency transforms which are primarily agnostic to the type of input waveform transformed. Some methods use perceptual transforms to compress the information, but this transform is fundamentally useful for a broad range input types. Examples of this approach include short time Fourier transforms (STFT), log-mel spectrograms, and

non-stationary constant Q transforms (NS-CQT) (Smith, 2019). These transforms can be lossy and require approximate inversion (as is the case for log-mel spectrograms) or exactly invertible (as in STFT). Vocoder transforms are collections of transforms, filters, and assumptions specific to human speech with well-defined inversion procedures. Examples of these include STRAIGHT, WORLD, Vocaine, and YANG (Kawahara et al., 2008; Morise et al., 2016; Agiomyrgiannakis, 2015; Kawahara et al., 2016). There are also *neural vocoders* (Wang et al., 2017; Shen et al., 2017; Mehri et al., 2016; Sotelo et al., 2017; Ping et al., 2017; Ping et al., 2018) which learn a neural network that can invert a time-frequency or vocoder transform, but these methods generally still need the high-level audio features to form intermediate losses during training. The exception to this is van den Oord et al. (2016), which trained on high level aligned text, voicing, and timing features but did not incorporate any transformed audio data. However, the subsequent combined system of Wavenet and Tacotron dubbed Tacotron 2 (Shen et al., 2017) used a neural vocoder-style Wavenet trained on log-mel spectrograms instead of aligned features directly.

Text to audio agreement is the last piece of the puzzle. Dynamic time warping (DTW) and hidden markov models (HMM) have been the go-to methods for sequence alignment (Rabiner, 1989; Fang, Fang), but the recent advent of seq2seq models with attention (as previously discussed in Section 2.3) has provided a convenient alternative for learning sequence alignment from data. Many different attention formulations are possible, the broad categories involve attention models which exploit domain knowledge such as monotonicty (sequences both proceed forward in time, but at variable rates), and those that don't exploit domain knowledge. In the former category common choices are Gaussian attention constrained to move in one direction (Graves, 2013b; Sotelo et al., 2017), guided attention (Tachibana et al., 2017), or softmax attention with locality constraints (Chorowski et al., 2015; Shen et al., 2017), while the latter case is generally a softmax attention (Bahdanau et al., 2014; Wang et al., 2017) or variants thereof.

2.6.2 Related TTS Work

All these components can be combined in different ways. There are those which operate on the *classic* feature engineered inputs, performing text-to-audio feature

alignment separately using an HMM. These models usually replace linear models or other simple estimators with neural networks (Zen et al., 2013; Fan et al., 2015; Wu et al., 2016). Some methods focus on the symbol-to-audio feature alignment process (Graves, 2013b; Wang et al., 2017; Taigman et al., 2017; Nachmani et al., 2018) (see Graves unpublished work extending the handwriting generation model to speech [1]), leaving the conversion from high-level audio features to realized audio up to a predefined DSP method such the inverse process of the vocoder, Griffin-Lim phase estimation (Griffin and Lim, 1984), or STFT inversion via overlap-add. Others focus mostly on the *neural vocoder* (van den Oord et al., 2016; Arık et al., 2017; Arik et al., 2017), extracting text, timing, and voicing features via other means and performing alignment as a separate input processing step. Many systems combine the alignment and neural vocoding into a single model (Sotelo et al., 2017; Shen et al., 2017; Ping et al., 2017; Ping et al., 2018), with pre-training or intermediate losses to improve convergence.

2.7 Symbolic Music Modeling

2.7.1 On the Relationship between Speech Synthesis and Music Modeling

Some music modeling approaches bear strong similarity to TTS, taking high level input such as text or stylistic cues to drive a low-level (time-frequency or raw waveform) generator to create new and interesting soundscapes, based on relationships defined in training data (Agostinelli et al., 2023; Copet et al., 2023; Kreuk et al., 2022). Publications focusing on the nascent field of "text-to-music" use many of the same techniques as TTS, with some key considerations. The primary consideration being that, unlike TTS, there is no explicit alignment between the text description of the desired audio, and the resulting signal produced by the model. This more closely resembles "global style" conditioning for sequence modeling, as seen in both TTS and handwriting synthesis systems.

Music models which target waveform or spectrogram outputs have the benefit of a flexible, general output representation and a large volume of data available

1. https://www.youtube.com/watch?v=-yX1SYeDHbg

for training. Due to these models making few assumptions about the underlying compositional theories and frameworks for genre, when compared to models which deal with specific attributes of *particular* genres, styles, or tertiary information, they often lack the detailed stylistic traits and interactivity desired by musicians in compositional tools. Instead text-to-music models act as high level inspiration or broad compositional ideations, quickly turning over new ideas and exploring new soundscapes and themes.

Music is woven into the fabric of many civilizations throughout history. Though there are many commonalities between musical systems (in both the written and oral tradition) from different cultures, each system of communicating music performance depends heavily on the underlying instruments and performance practices of the time and place in which the system was developed. The work in this thesis builds upon the study of written forms developed and used in the Western musical canon, commonly known as sheet music notation, and with analysis rooted in western music theory and classical performance practice in western Europe dating from approximately 1300 to 1920 CE (Strayer, 2013; Schoenberg and Stein, 1969).

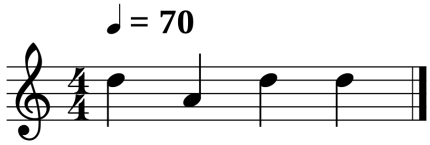

Figure 2.5 – Quarter note sequence D A D D at 70 beats per minute, shown in sheet music notation, as first seen in Section 1.1.1

2.7.2 Time-frequency Relations

Approaching music modeling from a similar perspective as TTS, we see that just as there are forms for conversion of written text to phonetic and sound-focused representations (such as International Phonetic Alphabet) which describe key details of the sentence ”performance” without finegrained details such as speaker identity, local rate variability, pitch inflection and so on. In a similar way sheet music notation is a useful description language for musical performances based on the foundations of the Western musical canon, which communicates high level, critical details about the important performance aspects of a piece (from composer to

performer) without fully detailed information about the final realization.

Just as time-frequency representations are commonly used for TTS systems, sheet music notation forms a type of time-frequency representation as well. These music specific time-frequency representations have different scaling for both pitch/frequency, and time/tempo than speech-specific formats, but fundamentally capture similar types of information. As such we can apply similar modeling techniques from other deep learning for sequences to music specific tasks. Models can directly use the multi-dimensional structure of the music representation, or flattened as a sequential modeling problem (Sturm et al., 2015; Dong et al., 2022; Liang, 2016; Min et al., 2023; Huang et al., 2017b). Many simplified musical forms (lead sheet format for jazz, tracker format for video game music, ABC format as a simplified MIDI-like representation) for specific genres exist, although transcribed data for uncommon genres is hard to find. In fact many folk music formats exist in the aural tradition entirely, with no standard written form and relatively few audio recordings. Systems modeling these types of music must "scale down" to the tiny available collections, as well as "scaling up" to larger settings with more available music pieces, and potentially larger models (Sturm et al., 2019). Applications of deep models for music composition will be discussed at length in chapter 8.

3 Prologue to the Article

Representation Mixing for TTS Synthesis. Kyle Kastner, Joao Felipe Santos, Yoshua Bengio, Aaron Courville. Published in ICASSP 2019 IEEE International Conference on Acoustics, Speech and Signal Processing.

Personal Contribution.

The idea for this model structure, combining phonetic and character representations within a single encoder was conceived by myself to solve issues with pronunciation in parametric TTS, as part of an ongoing speech research collaboration with Joao, Yoshua, and Aaron. All model training, dataset selection, and overall research project design were done by me. Joao helped choose the evaluation website, set up the server for the user study, provided helpful discussions during model development, and provided general guidance on user study evaluations. Previous iterations of TTS models inside Mila were developed as part of a focused effort spearheaded by Yoshua, and elements of those previous TTS designs were used in the final model for this paper as well. Aaron advised throughout model development and publication of this work, as well as being a core advisor for the larger speech research group throughout the publication process. I presented the results in an oral presentation, and this article was published in the proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), in 2019.

Affiliations

Kyle Kastner, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Joao Felipe Santos, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Yoshua Bengio, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CIFAR Senior Fellow.

Aaron Courville, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CIFAR Fellow.

Funding

We acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

4 Representation Mixing for TTS Synthesis

4.1 Introduction

TTS synthesis (Hunt and Black, 1996) focuses on building systems which can generate speech (often in the form of an audio feature sequence a), given a set of linguistic sequences, l. These sequences, l and a, are of different length and dimensionality thus it is necessary to find the alignment between the linguistic information and the desired audio. We approach the alignment problem by jointly *learning* to align these two types of information (Bahdanau et al., 2014), effectively translating the information in the linguistic sequence(s) into audio through learned transformations for effective TTS synthesis (Sotelo et al., 2017; Wang et al., 2017; Shen et al., 2017; Tachibana et al., 2018; Ping et al., 2018).

4.1.1 Data Representation

We employ *log mel spectrograms* as the audio feature representation, a well-studied time-frequency representation (Smith, 2019) for audio sequence a. Various settings used for this transformation can be seen in Table 4.2.

Linguistic information, l, can be given at the abstract level as graphemes (also known as characters when using English) or at a more detailed level which may include pronunciation information, such as phonemes. Practically, character-level information is widely available in open data sets though this representation allows ambiguity in pronunciation.

4.1.2 Motivating Representation Mixing

In some cases, it may be impossible to fully realize the desired audio without being given direct knowledge of pronunciation. Take as a particular example the sentence *"All travelers were exhausted by the wind down the river"*. The word

"wind" in this sentence can be either noun form such as *"The wind blew swiftly on the plains"*, or verb form for traveling such as *"... and as we wind on down the road"*, both of which have different pronunciation.

Without external knowledge (such as additional context, or an accompanying video) of which pronunciation to use for the word, a TTS system which operates on character input will always have ambiguity in this situation. Alternate approaches such as grapheme to phoneme methods (Rao et al., 2015) will also be unable to resolve this problem. Such cases are well-known in both TTS and linguistics (Black et al., 1998; Eddington and Tokowicz, 2015), motivating our desire to flexibly combine grapheme and phoneme inputs in a single encoder. Our method allows per example control of pronunciation during inference *without requiring* this information for all desired inputs, and similar methods have also been important for other systems (Ping et al., 2018).

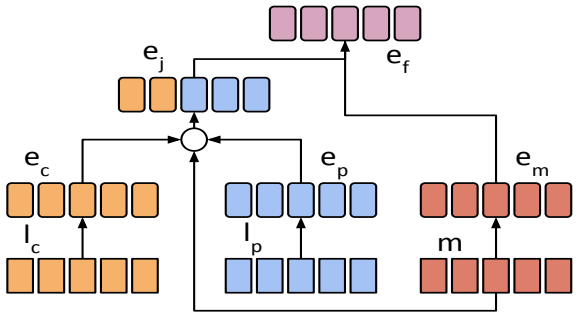

Figure 4.1 – Visualization of embedding computation

4.2 Representation Mixing Description

The input to the system consists of one data sequence, l_j, and one mask sequence, m. The data sequence, l_j, consists of a mixture between a character sequence, l_c, and a phonetic sequence, l_p. The mask sequence describes which respective sequence the symbol came from with an integer (0 or 1). Each time a training sequence is encountered, it is randomly mixed at the word level and we assign all spaces and punctuation as characters.

For example, a pair "*the cat*", "@d@ah @k@ah@t" (where @ is not a used symbol and serves only as an identifier to mark phoneme boundaries) can be resampled

as "*the* @*k*@*ah*@*t*", with a corresponding mask of $[0, 0, 0, 0, 1, 1, 1]$. On the next occurrence, it may be resampled as "@*d*@*ah cat*", with mask $[1, 1, 0, 0, 0, 0]$. This resampling procedure can be seen as data augmentation, as well as training the model to smoothly process both character and phoneme information without over-reliance on either representation. We demonstrate the importance of each aspect in Section 4.4.

4.2.1 Combining Embeddings

The full mixed sequence, l_j, separately passes through two embedding matrices, e_c and e_p, and is then combined using the mask sequence to form a joint mixed embedding, e_j. For convenience, e_c and e_p, are set to a vocabulary size that is the *max* of the character and phoneme vocabulary sizes. This mixed embedding is further combined with an embedding of the mask sequence itself, e_m, for a final combined embedding, e_f. This embedding, e_f, is treated as the standard input for the rest of the network. A diagram describing this process is shown in Figure 4.1.

$$e_j = (1 - m) * e_c + m * e_p \tag{4.1}$$

$$e_f = e_m + e_j \tag{4.2}$$

4.2.2 Stacked Multi-scale Residual Convolution

In the next sections, we describe the network architecture for transforming the mixed representation into a spectrogram and then an audio waveform. A full system diagram of our neural network architecture can be seen in Figure 4.2.

The final embedding, e_f, from the previous section is used as input to a stacked multi-scale residual convolutional subnetwork (SMRC). The SMRC consists of several layers of multi-scale convolutions, where each multi-scale layer is in turn a concatenation of 1×1, 3×3, and 5×5 layers concatenated across the channel dimension. The layers are connected using residual bypass connections (He et al., 2016a) and batch normalization (Ioffe and Szegedy, 2015) is used throughout. After the convolutional stage, the resulting activations are input to a bidirectional LSTM layer (Schuster and Paliwal, 1997; Hochreiter and Schmidhuber, 1997b). This ends the *encoding* part of the network.

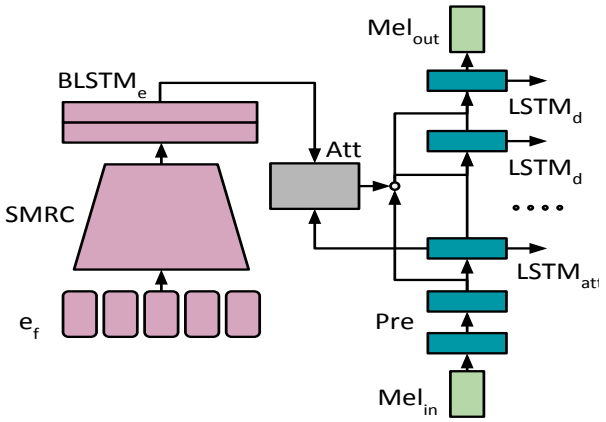

Figure 4.2 – Encoding, attention, and one step of mel decoder

4.2.3 The Importance of Noisy Teacher Forcing

All audio information in the network passes through a multilayer pre-net with dropout (Srivastava et al., 2014; Wang et al., 2017; Shen et al., 2017), in both training and evaluation. We found that using pre-net corrupted audio in every layer (including the attention layer) greatly improves the robustness of the model at evaluation, while corruption of linguistic information made the generated audio sound less natural.

4.2.4 Attention-based RNN Decoder

The encoding activations are attended using a Gaussian mixture (GM) attention (Graves, 2013) driven by an LSTM network (conditioned on both the text and pre-net activations), with a softplus activation for the step of the Gaussian mean as opposed to the more typical exponential activation. We find that a softplus step substantially reduces instability during training. This is likely due to the relative length differences between linguistic input sequences and audio output.

Subsequent LSTM decode layers are conditioned on pre-net activations, attention activations, and the hidden state of the layer before, forming a series of skip connections (Graves, 2013; Zhang et al., 2016). LSTM decode layers utilize cell dropout regularization (Semeniuta et al., 2016). The final hidden state is projected to match the dimensionality of the audio frames and a mean squared error loss is calculated between the predicted and true next frames.

4.2.5 Truncated Backpropagation Through Time (TBPTT)

The network uses truncated backpropagation through time (TBPTT) (Hochreiter and Schmidhuber, 1997b) in the decoder, only processing a subsection of the relevant audio sequence while reusing the linguistic sequence until the end of the associated audio sequence. TBPTT training allows for a particular iterator packing scheme not available with full sequence training, which continuously packs subsequences resetting only the particular elements of the sequence minibatches (and the associated RNN state) when reaching the end of an audio sequence. A simplified diagram of this approach can be seen in Figure 4.3.

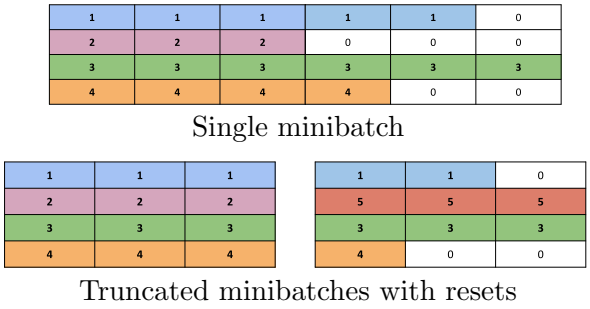

Figure 4.3 – Comparison of standard minibatching versus truncation.

4.2.6 Converting Features Into Waveforms

After predicting a log mel spectrogram sequence from linguistic input, further transformation must be applied to get an audible waveform. A variety of methods have been proposed for this inversion such as the Griffin-Lim transform (Griffin and Lim, 1984), and modified variants (Slaney et al., 1994). Neural decoders such as the conditional WaveNet decoder (van den Oord et al., 2016; Shen et al., 2017) or conditional SampleRNN (Mehri et al., 2016) can also act effectively as an inversion procedure to transform log mel spectrograms into audible waveforms.

Optimization methods also work for inversion and we utilize an L-BFGS based routine. This process optimizes randomly initialized parameters (representing the desired waveform) passed through a log mel spectrogram transform, to make the transform of these parameters match a given log mel spectrogram (Thomé, 2018).

This results in a set of parameters (which we now treat as a fixed waveform) with a lossy transform that closely matches the given log mel spectrogram. The

overall procedure closely resembles effective techniques in other generative modeling settings (Gatys et al., 2016).

Inversion Method	*Seconds per Sample*	*STRTF*
100 L-BFGS	1.49E-4	3.285
1000 L-BFGS	1.32E-3	29.08
Modified Griffin-Lim	2.81E-4	6.206
100 L + GL	4.11E-4	9.063
1000 L + GL	1.6E-3	35.28
WaveNet	7.9E-3	174.7
100 L + GL + WN	8.3E-3	183.7
1000 L + GL + WN	9.5E-3	210.0

Table 4.1 – Comparison of various log mel spectrogram to waveform conversion techniques. STRTF stands for "slower than real time factor" (calculated assuming output sample rate of 22.05 kHz), where 1.0 would be real-time generation for a single example.

4.2.7 Inversion Pipeline

This work uses a combined pipeline of L-BFGS based inversion, followed by modified Griffin-Lim for waveform estimation (Slaney et al., 1994). The resulting waveform is of moderate quality, and usable for certain applications including speech recognition. Using either L-BFGS or Griffin-Lim in isolation did not yield usable audio in our experiments. We also found that this ordering (L-BFGS then Griffin-Lim) in our two-stage pipeline works much better than the reverse setting.

To achieve the highest quality output, we then use the moderate quality waveform output from the two stage pipeline, converted back to log mel spectrograms, for conditional WaveNet synthesis. Though other work (Shen et al., 2017) clearly shows that log mel spectrogram conditioned WaveNet can be used directly, we find the two stage pipeline allows for quicker quality checking during development, as well as slightly higher quality in the resulting audio after WaveNet sampling. We suspect this is because the pre-trained WaveNet (Yamamoto et al., 2018) that we use is trained on ground-truth log mel spectrograms rather than predicted outputs. The synthesis speed of each setting is shown in Table 4.1.

4.3 Related Work

Representation mixing is closely related to the "mixed-character-and-phoneme" setting described in Deep Voice 3 (Ping et al., 2018), with the primary difference being our addition of the mask embedding e_m. We found utilizing a mask embedding alongside the character and phoneme embeddings further improved quality, and was an important piece in the text portion of the network. The focused user study in this paper also highlights the advantages of this kind of mixing independent of high-level architecture choices, since our larger system is markedly different from that used in Deep Voice 3.

The SMRC subnetwork is closely related to the CBHG encoder (Wang et al., 2017), differing in the number and size of convolutional scales used, no max-pooling, use of residual connections, and multiple stacks of multi-scale layers. We use a Gaussian mixture attention, first described by Graves (Graves, 2013) and used in several speech related papers (Sotelo et al., 2017; Skerry-Ryan et al., 2018; Taigman et al., 2017), though we utilize *softplus* activation for the mean-step parameter finding that it improved stability in early training.

Our decoder LSTMs utilize cell dropout (Semeniuta et al., 2016) as seen in prior work (Ha et al., 2016; Ha and Eck, 2017). Unlike Tacotron (Wang et al., 2017), we do not use multi-step prediction, convolutional layers in the mel-decoding stage, or Griffin-Lim inside the learning pathway. Audio processing closely resembles Tacotron 2 (Shen et al., 2017) overall, though we precede conditional WaveNet with a computationally efficient two stage L-BFGS and Griffin-Lim inference pipeline for quality control, and as an alternative to neural inversion.

4.4 Experiments

The model is trained on LJSpeech (Ito, 2017), a curated subset of the LibriVox corpus. LJSpeech consists of 13,100 audio files (comprising a total time of approximately 24 hours) of read English speech, spoken by Linda Johnson. The content of these recordings is drawn from scientific, instructional, and political texts published between 1893 and 1964. The recordings are stored as 22.05 kHz, 16 bit WAV files, though these are conversions from the original 128 kbps MP3 format.

Character level information is extracted from audio transcriptions after a normalization and cleaning pipeline. This step includes converting all text to lowercase along with expanding acronyms and numerical values to their spelled-out equivalents. Phonemes are then extracted from the paired audio and character samples, using a forced alignment tool such as Gentle (Ochshorn and Hawkins, 2017). This results in character and phoneme information aligned along word level boundaries, so that randomly *mixed* linguistic sequences can be sampled repeatedly throughout training. When using representation mixing for training, we choose between characters and phonemes with probability .5 for each word in all experiments.

Audio Processing	22.05 kHz 16 bit input, scale between $(-1, 1)$, log mel spectrogram 80 mel bands, window size 512, step 128, lower edge 125 Hz, upper edge 7.8 kHz, mean and std dev normalized per feature dimension after log mel calculation.
Embeddings	vocabulary size (v) 49, 49, 2, embedding dim 15, truncated normal $\frac{1}{\sqrt{v}}$
SMRC	3 stacks, stack scales 1×1, 3×3, 5×5, 128 channels each, batch norm, residual connections, $ReLU$ activations, orthonormal init
Enc Bidir LSTM	hidden dim 1024 ($128 \times 4 \times 2$), truncated normal init scale 0.075
Pre-net	2 layers, hidden dim 128, dropout keep prob 0.5, linear activations, orthonormal init
Attention LSTM	num mixes 10, *softplus* step activation, hidden dim 2048 (512×4), trunc norm init scale 0.075
Decoder LSTMs	2 layers, cell dropout keep prob 0.925, truncated normal init scale 0.075
Optimizer	mse, no output masking, Adam optimizer, learning rate $1E-4$, global norm clip scale 10
Other	TBPTT length 256, batch size 64, training steps 500k, 1 TitanX Pascal GPU, training time 7 days

Table 4.2 – Architecture hyperparameter settings

4.4.1 Log Mel Spectrogram Inversion Experiments

We use a high quality implementation of WaveNet, including a pre-trained model directly from Yamamoto et. al. (Yamamoto et al., 2018). This model was also trained on LJSpeech, allowing us to directly use it as a neural inverse to the log mel spectrograms predicted by the attention-based RNN model, or as an inverse to log mel spectrograms extracted from the network predictions after further processing. Ultimately combining the two stage L-BFGS and modified Griffin-Lim pipeline with a final conditional WaveNet sampling pass demonstrated the best quality, and is what we used for the user study shown in Table 4.3.

4.4.2 Preference Testing

Given the overall system architecture, we pose three primary questions (referenced in column Q in Table 4.3) as a user study:

1) Does representation mixing (RM) improve character-based inference over a baseline which was trained only on characters?

2) Does RM improve phoneme with character backoff for unknown word (PWCB) inference over a baseline trained on fixed PWCB?

3) Does PWCB inference improve over character-based inference in a model trained with representation mixing?

To answer these questions, we pose each as a preference test by presenting study participants with 20 paired choices. Each user was instructed to choose the sample they preferred [1] . The 20 tests were chosen randomly in a pool of 123 possible tests covering all three question types, from 41 possible sentences. Presentation order was randomized for each question and every user. The overall study consisted of 22 users and 429 responses across all categories (some users didn't select any preference for some questions).

We see that users clearly prefer models trained with representation mixing, even when using identical information for inference. This highlights the data augmentation aspects of representation mixing, as regardless of information type representation mixing (RM) gives clearly preferable results compared to static representations (Char, PWCB). The preference of representation mixing over static PWCB also

1. https://s3.amazonaws.com/representation-mixing-site/index.html

Q	***Model A***	*Model B*	*C. A*	*Total*	*% A*
1	RM (char)	Char	101	137	73.7%
2	RM (PWCB)	PWCB	106	145	73.1%
3	RM (PWCB)	RM (char)	86	147	58.5%

Table 4.3 – A / B user preference study results. RM uses approach in parenthesis at inference time. Columns "*C. A*" and "*% A*" indicate the number and percentage of users which preferred *Model A*.

means that introducing phoneme and character information without mixing is less beneficial than full representation mixing.

It is also clear that for representation mixing models, pronunciation information (PWCB) at inference gives preferable samples compared to character information. This is not surprising, but further reinforces the importance of using pronunciation information where possible. Representation mixing enables choice in input format, allowing the possibility to use many different inference representations for a given sentence with a single trained model.

4.5 Conclusion

This paper shows the benefit of *representation mixing*, a simple method for combining multiple types of linguistic information for TTS synthesis. Representation mixing enables inference conditioning to be controlled independently of training representation, and also results in improved quality over strong character and phoneme baselines trained on a publically available audiobook corpus.

5 Prologue to the Article

R-MelNet: Reduced Mel-Spectral Modeling for Neural TTS. Kyle Kastner, Aaron Courville. Published as arXiv preprint arXiv:2206.15276, 2022.

Personal Contribution.

The idea to pursue a detailed study in reproducing the MelNet architecture with the goal of reducing the amount of compute necessary for final inference, was driven by my interest in these problems and experience with variants of these methods. The holistic model development, dataset choice, and scope of the research project was set by me. Aaron provided feedback and advice during the model design process, as well as proposing ablation studies, methods for reducing the model size, and improvements to the stability of the attention mechanism based on early training failures.

Affiliations

Kyle Kastner, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Aaron Courville, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CIFAR Fellow.

Funding

We acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

6 R-MelNet

6.1 Introduction

Recent progress in text-to-speech (TTS) synthesis has been largely driven by the integration of neural networks which parameterize various factorizations of problems and subproblems in speech modeling (Black et al., 2007). This adoption has been driven by improved tooling (Paszke et al., 2019) as well as rapid improvement in stability and optimization for neural network training, allowing the development of ever larger and more complex TTS systems. Particularly, many recent neural TTS approaches feature a two-stage process, with one stage learning features from text-like inputs often in conjunction with a learned alignment procedure using attention modeling, or other language specific duration estimation models (Sotelo et al., 2017; Wang et al., 2017; Ping et al., 2018; Chen et al., 2021).

Following this first stage model (often denoted as the *frontend*), a secondary *backend* network which focuses on upsampling, refining, or otherwise estimating audio waveforms from structured, time-aligned information is used to predict the final necessary output (Kalchbrenner et al., 2018; Chen et al., 2020). Input information is generally a text-based representation, such as characters or phonemes, although recent models have also used self-supervised representations which combine audio and text information. Input modalities can be mixed at a per-datapoint level (Kastner et al., 2019; Ping et al., 2018; Fong et al., 2020), in order to provide data-based regularization during training as well as the ability to flexibly choose between character and phonetic representations during inference.

The intermediate audio representation used for these two-stage systems is often a well-studied time-frequency representation from digital signal and speech processing, such as vocoder features, linear spectrograms, or log-mel spectrograms. These features can be coupled with high level information about delivery style, speaker characteristics, pitch curves, or any number of other high level controls useful for

generative speech tasks.

The model types used for these various factorizations of the TTS problem come in a variety of forms: recurrent networks, convolutional networks, transformer models, or neural models combined with alternative probabilistic methods such as HMMs (Mehta et al., 2021). One common variety of recurrent architecture builds on an attention based system for handwriting generation and text-to-vocoder synthesis (Graves, 2013), using attention based recurrent networks to learn an alignment for turning character based inputs into audio outputs (Sotelo et al., 2017; Wang et al., 2017; Shen et al., 2018). The basis for our frontend model, MelNet (Vasquez and Lewis, 2019), builds upon this foundation as well as prior work on multi-dimensional recurrent models (Graves and Schmidhuber, 2008; Theis and Bethge, 2015; Kalchbrenner et al., 2015; Visin et al., 2015; Van Oord et al., 2016), in order to model time-frequency spectral features by jointly recurring over time and frequency.

6.2 System Design

The output target for the first tier of MelNet is a heavily downsampled log-mel spectrogram. In the case of a 5 tier melnet using a downsampling factor of 2 per tier (as used in R-MelNet), this results in data which is downsampled by 4 in time, and 8 in mel-frequency at the lowest level. As an example, we use a base log-mel spectrogram size (in time, frequency format) of (448, 256), corresponding to approximately 4 seconds of audio, as in Figure 6.1. This leads to an overall log-mel spectrogram (in time, frequency format) of (448, 256) reducing to a downsampled target of only (112, 32) for the first tier loss, as in Figure 6.2. This extremely low-resolution data is predicted autoregressively (over both time and frequency), conditioned by attending over input text features. Time-frequency autoregression is handled by a structured arrangement of recurrent neural networks (Vasquez and Lewis, 2019).

Subsequent tiers after the first in standard MelNet do not use an attention subnetwork, rather using the previously predicted, low-resolution log-mel spectrogram to condition the autoregressive generation (once again, over both time and frequency via recurrent networks) of a neighboring log-mel spectrogram - namely the

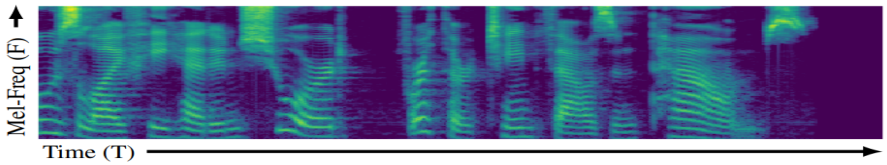

Figure 6.1 – Log-mel spectrogram (T=448, F=256).

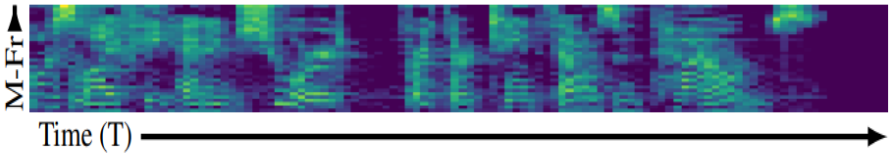

Figure 6.2 – Downsampled log-mel spectrogram (T=112, F=32).

log-mel spectrogram formed by downsampling in the same way but shifted by one column in time, or one row in frequency. Once predicted, the input conditioning and the predicted log-mel spectrogram are recombined by alternating either rows or columns into a map that is effectively twice as large in either time, or frequency. This means the overall tiers of the MelNet model a multi-scale sequence of log-mel spectrograms which gradually increase in resolution by 2 in one-dimension at every tier after the first.

6.2.1 A Dual-Purpose View of MelNet

The iterative checkerboard of upsampling tiers in MelNet is reminiscent of several other image generative models (Dinh et al., 2017; Menick and Kalchbrenner, 2018; Saharia et al., 2021), and we can crudely break the overall MelNet architecture into two segments - the first tier, which predicts a low-resolution target conditioned only on text information using an attention subnetwork, and all subsequent tiers, which upsample the initial low-resolution predictions (and all intermediate predictions from previous tiers) into a high-quality, holistic log-mel spectrogram. Given this view of the overall MelNet architecture, and given the high computational cost of autoregressive sampling over larger and larger multi-dimensional maps, a natural question arises: *Can we remove the upsampling tiers entirely, and opt instead to predict coarse, extremely low-resolution log-mel spectrograms which are paired with another inversion routine in order to obtain audio samples?*

Demonstrations of reduced tier samples from the original MelNet hint at the

feasibility of this approach. In this work, we show that indeed, it is possible to remove all tiers except the first and instead use a WaveRNN to convert low-resolution first tier log-mel spectrograms into waveforms. Our model, denoted R-MelNet produces quality TTS with fewer hardware resources and parameters compared to the original implementation.

6.2.2 Architecture Details

The R-MelNet frontend largely follows the recipe specified in MelNet, utilizing multi-dimensional recurrent neural networks, combined with an attention subnetwork over text features. Many of the details below are not specifically unique to R-MelNet, however we re-emphasize key aspects in order to clearly define our approach. The frontend network architecture uses 5 combined layers, consisting of time, frequency, and centralized stacks (Vasquez and Lewis, 2019). The attention network is inserted into the middle of the 5 combined layers, conditioning all subsequent outputs, and all layers are connected with residual connections (He et al., 2016b; Graves, 2013). We employ GRU units for all mel-spectrogram related recurrence, with hidden size for each GRU to 256. For the attention subnetwork, we use a combination of embeddings to train with representation mixing (Kastner et al., 2019; Ping et al., 2018; Fong et al., 2020) at a 0.5 per word swap probability between characters and phonemes, in order to enable either character-based or phoneme-based synthesis as well as providing data augmentation during training. The combined representation mixing embeddings are then fed directly into a bidirectional LSTM, forgoing the convolutional subnet which often links embeddings to the bidirectional LSTM in other work (Shen et al., 2018; Kastner et al., 2019). These resulting biLSTM hidden states form the memory which is attended over to condition the overall generation. All initial recurrent hidden states are fixed, starting from all zeros.

6.2.3 Reduced Resource Requirements

Neural networks are often limited by hardware resources, and GPU memory in particular is a frequent bottleneck for model training. We utilize several techniques to minimize memory requirements, keeping the overall memory necessary for training within 11 gigabytes. All neural network code for these experiments was written

using the PyTorch machine learning framework (Paszke et al., 2019), and trained on a single 2080Ti NVIDIA GPU. One common way to reduce memory requirements is changing the floating point representation, from full precision (32 bit) to half precision (16 bit). This can bring a host of training complications due to numerical inaccuracy, but generally gives a substantial reduction in memory usage. We use half precision (16 bit) representations for all data and model parameters, as well as an Adam optimizer (Kingma and Welling, 2015) adapted for 16 bit training (Brock et al., 2018). Despite these techniques, the memory required for moderately sized minibatches (on the order of 12 to 16) is still too large. One approach to further reduce memory use is to aggregate the gradient computation for a batch size N over K forward passes of sub-minibatches size M, where $M \times K = N$ (Hoffer et al., 2019), thus allowing these *virtual batches* to control memory usage independently of the effective batch size for optimization. We use a virtual batch size of 8, with an effective batch size of 16 in this work.

6.2.4 Attention

The attention subnetwork is a critical part of the overall architecture, as all text-based conditioning flows through this subnetwork, and the dynamics of the learned alignment define important aspects for TTS tasks. Attention calculations use many non-typical components and operations, and numerical precision is of critical importance. This is directly at odds with the half precision training strategy used in this work, and we made several modifications in order to ensure the stability of attention dynamics during training and sampling. R-MelNet uses a modified mixture of logistics attention (Vasquez and Lewis, 2019; Battenberg et al., 2020), and we note that many of these changes may be necessary due to framework level implementation details between the PyTorch used here and the Tensorflow used in MelNet. We parameterize the mixture of logistics attention with M mixture

components

$$\{\hat{\kappa}_i^m, \hat{\beta}_i^m, \hat{\alpha}_i^m\} = W_g h_i \tag{6.1}$$

$$\kappa_i^m = \kappa_{i-1}^m + s_+(\hat{\kappa}_i^m) + \epsilon_\kappa \tag{6.2}$$

$$\beta_i^m = s_+(\hat{\beta}_i^m) \tag{6.3}$$

$$\alpha_i^m = \frac{\exp(\hat{\alpha}_i^m)}{\sum_{m=1}^{M} \exp(\hat{\alpha}^{m_i})} \tag{6.4}$$

Throughout the following equations, γ_i is used as shorthand to represent the tuple of values $\kappa_i, \beta_i, \alpha_i$ parameterizing the location, scale, and weight for each mixture component. The term ϵ_κ is a small value to bound the minimum possible attention step, set to .001. Here s_+ represents a stable form of the softplus function

$$s_+(x) = \log(1 + \exp(-|x|)) + \max(0, x) \tag{6.5}$$

Crucially, the $\log(1+\exp(-|x|))$ function must be implemented in such a way that multiple numerical cases are handled in a branching fashion (Cook, 2012), and that each branch of the computation which can result in not a number (NaN) values does not propagate this value outside the region of numerical stability for that branch. In PyTorch, this is accomplished by creating an output array, creating masks matching each branch condition, and filling subsets of the output array with the results of each branch, rather than directly performing a *where* style conditional query.

This results in a final distribution function $F(u; \gamma_i)$ (Vasquez and Lewis, 2019)

$$F_i(u; \gamma_i) = \sum_{m=1}^{M} \alpha_i^m (1 + \exp(\frac{\kappa_i^m - u}{\beta_i^m}))^{-1} \tag{6.6}$$

$$\phi(u; \gamma_i) = F_i(\mu + 0.5; \gamma_i) - F_i(u - 0.5; \gamma_i) \tag{6.7}$$

Here we note that $\phi(u; \gamma_i)$ can be parameterized using sigmoid calculations by expanding the difference and combining exponential terms. We use the equivalence between sigmoid and tanh to formulate a numerically similar alternative, using a cheap approximation to tanh and the equivalence of β and $\frac{1}{\beta}$ when optimizing with the $s_+(x)$ function to avoid $\frac{1}{0}$ instability.

$$\tanh_{\sim} = \frac{x}{1 + |x|} \tag{6.8}$$

$$T(u; \gamma_i) = \frac{1}{2} + \frac{1}{2} \tanh_{\sim} \left(\frac{(u - \kappa_i^m)(\beta_i^m + \epsilon_\beta)}{2} \right) \tag{6.9}$$

$$\phi(u; \gamma_i) = \sum_{m=1}^{M} \alpha_i^m (T(u + 0.5; \gamma_i) - T(u - 0.5; \gamma_i)) \tag{6.10}$$

The term ϵ_β is used to bound the minimum value, and is fixed to .01. Finally, we use a straight-through style gradient reweighting in order to use the direct value of $\phi(u; \gamma_i)$ in the forward pass, but reweight the gradient according to the number of attention components $\frac{1}{\sqrt{M}}$. All intermediate attention values are calculated in full precision, and recast to half precision at the end of the attention weight calculation.

6.2.5 The Importance of Inference Stochasticity

The R-MelNet frontend uses mean squared error loss, as opposed to mixture density output and loss (Bishop, 1994; Graves, 2013; Vasquez and Lewis, 2019). Simple mean squared error, combined with numerical stability improvements, adaptive optimizers, and gradient clipping results in smooth, stable training. However when sampling, we note that the injection of stochasticity via noise is critical to achieve stable and high quality samples.

We form an approximate distribution using Q noise samples, by combining the average output mean predictions $\mu = \frac{1}{Q} \sum_{q=1}^{Q} \mu_q$ for a given model step with a particular standard deviation of noise, uniformly sampled over a predefined range R to get σ_r for this step. We next generate Q samples from a truncated Gaussian distribution defined by this (μ, σ_r) pair, and these inputs feed the next model step. This gives the next set of mean predictions $\hat{\mu}_{q=1}^{Q}$, which are then averaged to form a new estimated mean $\hat{\mu}$, a new $\hat{\sigma_r}$ is drawn from range R, and so on. This procedure iterates for the total number of necessary sample steps to give a complete output. R-MelNet uses noise range $0 \leq R \leq 0.75$, and $1 \leq Q \leq 100$ noise samples, with the best settings at .33 and 100 respectively. The result of sampling in this way can be seen in Figure 6.3.

This noise computation is easy to parallelize by permuting the differently noised inputs into the batch dimension, then averaging outputs along this dimension to

estimate the average μ. When utilizing this batch parallelism, the noisy sampling approach has relatively minor cost in terms of overall sampling speed. The level of noise is critically important, as without noise or with few numbers of noise samples Q the attention dynamics were highly unstable, and frequently led to issues with attention dynamics stalling as in Figure 6.4. Higher noise range R often resulted in more stable dynamics, but at the expense of final audio quality. Our results show that simple mean squared error losses can work when combined with noisy multi-sample inputs during sampling, though we recommend stochastic outputs (e.g. MDN) for future exploration. Naive sampling schemes which recompute hidden states are extraordinarily costly in MelNet style multi-dimensional recurrent models. Sampling practically requires a fast caching/memoization scheme which caches all previous values for time delayed and centralized stacks, only recomputing when absolutely necessary because dependent variables change. This means that time delayed and centralized stacks only need to be recomputed when the frequency stack finishes an entire row of frequency predictions, while the frequency stack uses cached versions of the time and centralized stacks for intermediate computation.

6.2.6 Framing the Context

One advantage of MelNet style architectures is the ability to condition on past log-mel spectrogram information, in order to prime the style of the resulting audio. This gives the ability to generate varied outputs for a single text prompt by varying past mel-spectrum information and associated text, a critical aspect of stylistic and expressive TTS (Graves, 2013; Chang et al., 2021; Stanton et al., 2021). This ability to prime the model with audio context, coupled with the ability to granularly control text information on a per-word basis using representation mixing, means it is possible to generate many possible candidate samples for a single desired phrase. Though this is advantageous for creating variable output, it also makes the selection of a single best model output from a collection of different priming audio and text sequences into a complex search problem.

For fully automated TTS, we use a simple heuristic search algorithm to refine and rerank proposed model samples, based on the smoothness, continuity, and completion of the attention weights. See Figure 6.4 for an example. After throwing out attentions which failed to reach the end of the text, or broke apart (detected

by contiguity of a mask based on the 95th percentile value), our ranking function looks at the nearness to the median length (across all seeds which succeeded), gap between the max and median attention trace value, gap between the max and min attention trace value, and the reciprocal of the min attention trace value. These four values are multiplied together, and the results ranked from lowest to highest, choosing the result with the lowest value. This means that attentions with a high constant value on the attention trace, that aren't unusually short or long compared to other seeds should get a low score, whereas uncertain or highly variable attentions, or ones with an especially fast or slow rate will get a higher score. This selection method could integrate automatic metrics as a direction for future work (Huang et al., 2022).

6.2.7 Converting Low-Resolution Mel-Spectrograms into Audio

As described above, predicting low-resolution log-mel spectrograms from R-MelNet is the first stage of the overall audio generation. Once the low-resolution output is predicted, another stage is necessary to turn the predicted output (see Figure 6.3) into listenable audio. We use a WaveRNN (Kalchbrenner et al., 2018) for this purpose, in order to preserve the autoregressive nature of audio decoding. Our setup follows the normal procedure for mel to audio WaveRNN training (McCarthy, 2018), with one crucial difference. We found that modifying the convolutional encoder of WaveRNN to upsample the time dimension of the $(112, 32)$ input by 4, back to the native time resolution performed poorly. Instead, it was beneficial to linearly interpolate (per frequency bin) the low resolution input up to $(448, 32)$ before input to the WaveRNN pipeline. Many models (Chen et al., 2020; Lam et al., 2022; Morrison et al., 2021) can be used for this mel-to-audio stage. Exploring which audio models best predict high-quality audio data given low-resolution input is an important direction for future work, as well as analyzing sources of pitch variation and pitch error (Morrison et al., 2021; Turian and Henry, 2020).

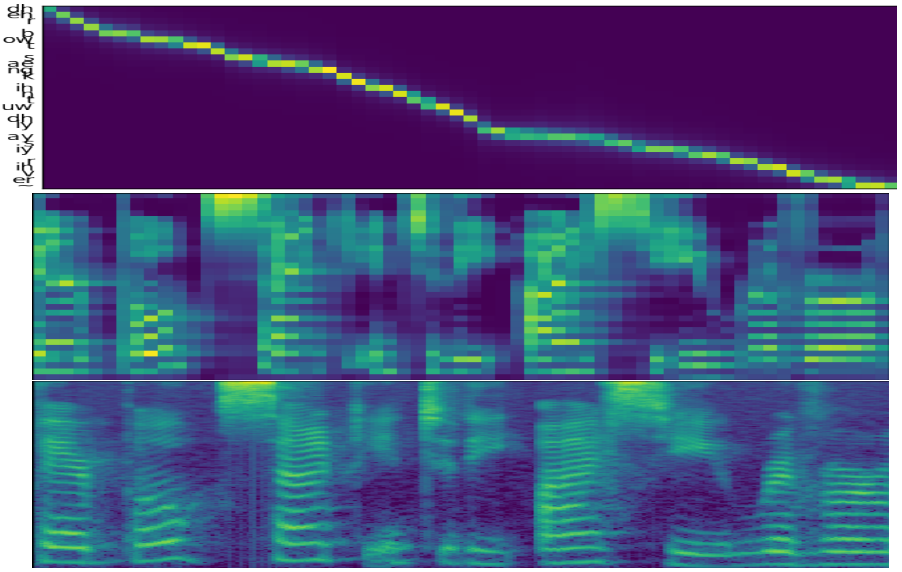

Figure 6.3 – Attention alignment and sampled output from Mel-Net frontend along with log-mel spectrogram of final output waveform from WaveRNN backend, for example input 'their boat sank into the icy river', phonemized to 'dhehr bowt saengk ihntuw dhiy aysiy rihver'.

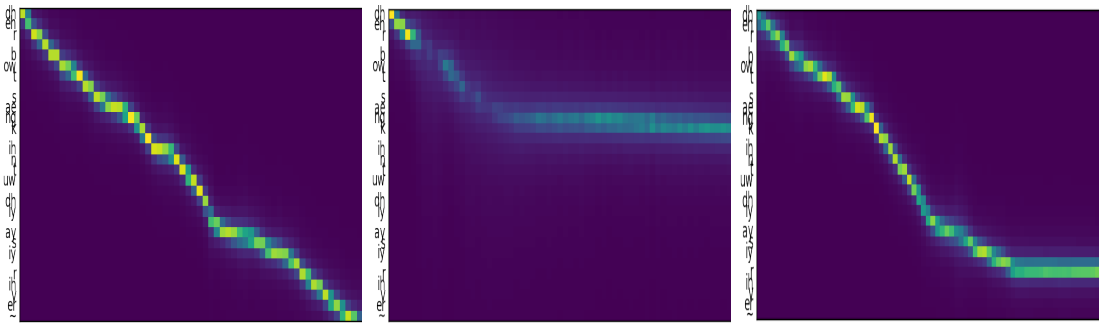

Figure 6.4 – Example of alternate successful attention alignment, and two failed alignments for the same sentence as Figure 6.3, starting with different audio and text priming.

6.3 Experiments

Given the massive number of components to this system, we encourage readers to refer to the experimental code [1] for hyperparameters and other key implementation details. All training for R-MelNet, as well as comparison baselines Fastspeech 2 and Portaspeech, used the LJSpeech dataset (Ito, 2017). We note that most audio preprocessing settings for R-MelNet follow precedent work (Yamamoto, 2018), with base audio at 22.050 kHz sample rate, using STFT size 6×200, STFT step

1. https://github.com/kastnerkyle/rmelnet

200, and base mels 256 before downsampling. We also found global mean and std normalization per frequency bin for the R-MelNet frontend was crucial for stable training.

Model	MOS-P (95% CI)	Parameters
R-MelNet	3.686 ± 0.144	43M
Fastspeech 2 (Wang et al., 2020)	3.6736 ± 0.151	27M
Portaspeech (Ren et al., 2021)	3.278 ± 0.183	21.8M

Table 6.1 – MOS-P listening study results, comparing Fastspeech 2, Portaspeech, and R-MelNet.

R-MelNet performs slightly better on average in a prosody-focused mean opinion score (MOS-P) test compared to Fastspeech 2 (as implemented by fairseq S^2, with HiFi-GAN vocoder (Kong et al., 2020)) and Portaspeech (also with HiFi-GAN vocoder), though confidence intervals for R-MelNet and Fastspeech 2 are highly overlapped. Test participants were asked to ignore small variations in audio quality, so although the resulting R-MelNet audio is slightly noisier than the equivalent outputs from Fastspeech 2 and Portaspeech, it did not overly affect ratings. In a quality based MOS test (MOS-Q) we would expect to see lower performance than these baselines, but pure audio quality was not the focus of our current work. We use 14 sentences from the Harvard Sentences (iee, 1969), along with 11 custom sentences for a total of 25 sentences, which were synthesized using all 3 models, and presented in random order to each user in the test. Given the ratings for this test (261 for each model, across 34 separate users), and overlap in confidence intervals, it is difficult to discern an absolute ranking of performance. We do not take these scores as a critique of the comparison methods, only as a validation that R-MelNet is of comparable quality to both Fastspeech 2 and Portaspeech, proving the validity of our approach for neural TTS. The spread of options for sampling and high variability in R-MelNet are potentially suited to interactive use, however in the current implementation sampling speed ($\sim$ 4x slower than real-time) and frame latency ($\sim$ 50 ms per mel frame) are not fast enough for human-in-the-loop usage. An ideal, pipelined version of R-MelNet could meet a 30 ms latency, sub real-time output threshold which would enable interactive use, but requires significant engineering effort. As such we leave this exploration to future work.

6.4 Conclusions

We describe R-MelNet, a two-stage architecture for neural text-to-speech synthesis. The R-MelNet frontend uses a one tier version of MelNet, which takes in a mixed sequence of character and phoneme features, along with an optional audio priming sequence, and outputs low-resolution mel-spectral features. These features are in turn upsampled, and input to a WaveRNN which generates an audio waveform. The audio from this generative pipeline is varied and expressive, giving many possible outputs for a single desired prompt. By incorporating practical methods for reducing computational complexity and memory usage such as half precision training with numerically stable implementations of core operations, our method enables training on a single commodity gpu. This resulting model compares favorably in terms of parameter count, and quality with recent non-autoregressive TTS methods when trained on an openly available single speaker dataset.

6.5 Acknowledgements

The authors would like to thank Tim Cooijmans, Cheng-Zhi Anna Huang, Yusong Wu, Laurent Dinh, and Johanna Hansen for helpful feedback on versions of this work. The authors would also like to thank all participants in our listening tests.

7 Prologue to the Article

SUNMASK: Mask Enhanced Control in Step Unrolled Denoising Autoencoders. Kyle Kastner, Tim Cooijmans, Yusong Wu, and Aaron Courville. Published in EvoMUSART 2023 International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar).

Personal Contribution.

The initial architecture combining SUNDAE and Coconet was constructed by me. After showing the initial model to the music and creativity group (which includes Tim, Yusong, and Aaron), subsequent refinement of the model processing and architecture was done by me, based on feedback from Tim, Yusong, and Aaron. The experiments were run by me, as well as the choices of dataset and expansion to various text experiments. Yusong created the key diagram showing the SUNMASK process, and Tim formulated the factorization of the model processing for the publication. Aaron provided advice and support throughout this cycle, as well as suggesting refinements to flesh out the experimental procedure and result presentation. I presented the work in an oral presentation when this paper appeared in the proceedings of EvoMUSART 2023, an EvoStar conference.

Affiliations

Kyle Kastner, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Tim Cooijmans, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Yusong Wu, Mila, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Aaron Courville, Mila, Département d'Informatique et de Recherche Opéra-

tionnelle, Université de Montréal, CIFAR Fellow.

Funding We acknowledge the support of the following organizations for research funding and computing support: NSERC,Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

8 SUNMASK

8.1 Introduction

Modern approaches to content generation frequently utilize probabilistic models, which can be parameterized and learned using artificial neural networks. Common types of neural probabilistic models used for generation can be broadly stratified to form two broad categories based on factorization: autoregressive models (AR), and non-autoregressive models (NAR). We introduce SUNMASK, a NAR generative model for structured sequences [1].

8.1.1 Autoregressive Models

AR modeling with deep neural networks has been a dominant approach for generative modeling and feature learning (Radford and Wu, 2019; Kalchbrenner et al., 2015; Theis and Bethge, 2015; Van Oord et al., 2016; Vasquez and Lewis, 2019) which has many crucial advantages in both training and inference. One key concern is the necessity of defining a "dependency chain" in the form of a (typically) directed acyclic graph (DAG). Sampling during inference can be accomplished in a straightforward manner using ancestral sampling - sampling from the first variable or variables in the DAG, using those to conditionally estimate a probability distribution for subsequent variables.

Many applications have straightforward orderings in which to define this chain of variables, based on domain knowledge. For example following the flow of time for timeseries modeling is often a logical choice, allowing models to make predictions into the future from the past. However in many other domains, for example images, language, or music, the process of defining a dependency chain over input variables (e.g. pixels, characters, words, or notes) is far from straightforward, as for any arbitrary ordering there can frequently be examples where this ordering *creates*

1. https://github.com/SUNMASK-web/sunmask. Last accessed 8 February 2023.

long-term dependencies, or otherwise makes satisfaction of dependencies during training and evaluation more difficult than another alternative ordering.

This divide becomes further compounded in many creative applications to these domains, as creators typically iterate repeatedly: forming a concept, sketching out the concept, and seeing where the creative flow (based on the sketch) may lead to alterations in the original concept, thus "rewriting" sketched steps. Though the resulting output may be perceived in a time-ordered fashion (for example, reading a book or listening to a song), the initial creation was performed globally and holistically. This global view is often critical to creating elements such as foreshadowing and tension which make the resulting output interesting or enjoyable. This iterative process is directly at odds with a strict AR factorization, and requires well trained AR models to cope with a high degree of uncertainty and multi-modality for long range dependencies, which can lead to logical mistakes or other errors.

8.1.2 Non-Autoregressive Models

An alternative methodology for generative modeling is non-autoregression (NAR), broadly covering a large number of different modeling approaches which attempt to remove assumptions about variable ordering, instead either hand-defining per-exemplar orderings, or modeling variables jointly without resorting to chain rule factorization (Hill et al., 2016; Gu et al., 2018). One way to define an ordering over variables is via masking of inputs or intermediate network representations (Germain et al., 2015; Uria et al., 2016, 2014; Yang et al., 2019; Van Oord et al., 2016; Papamakarios et al., 2017), and indeed modern AR approaches such as transformers (Vaswani et al., 2017) use an autoregressive mask internally to define the chain of variables order. These masks can either be constant over all training (as in standard AR transformers and PixelCNN (Van Oord et al., 2016)) or dynamic per example (as in MADE (Germain et al., 2015)). When masks are dynamic per example, we begin to see the relationship between enforcing AR via masking and NAR methods, as although some ordering is assumed this ordering is no longer constant, and it becomes possible to use the same trained model to evaluate the probability of a particular output variable under *multiple* possible orderings.

Closely linked to masking methods are so called *diffusion models*, which relax the variable ordering problem through noise prediction (Sohl-Dickstein et al., 2015;

Song and Ermon, 2019; Ho et al., 2020). Rather than predicting a new variable or variables given previous ones in an arbitrarily chosen DAG, diffusion models focus on predicting a less noisy version of many variables jointly, given a set of noisy input variables. Iteratively applying this learned denoising improvement operator should eventually result in predicting a fully clean output estimate, given either a noisy version of the target domain, or even starting from pure noise. Given this framing it is clear that diffusion models are closely linked to denoising methods in general, specifically denoising autoencoders, as well as modern density modeling approaches such as generative adversarial networks (GAN (Goodfellow et al., 2014)), variational autoencoders (VAE (Kingma and Welling, 2013)), flow-based models (NICE (Dinh et al., 2014), RealNVP (Dinh et al., 2017), Normalizing Flows (Rezende and Mohamed, 2015), IAF (Kingma et al., 2016), MAF (Papamakarios et al., 2017)), iterative canvas sampling (DRAW (Gregor et al., 2015b)), and noise contrastive estimation (NCE (Gutmann and Hyvärinen, 2010)). Particular applications of this denoising philosophy such as BERT (Devlin et al., 2019), WaveGrad (Chen et al., 2020), and GLIDE (Nichol et al., 2022), have resulted in large quality improvements for feature learning and data generation for text, images, and audio (Lee et al., 2018; Ramesh et al., 2022; He et al., 2022).

8.1.3 Trade-offs Between Autoregressive and Non-autoregressive Approaches

The choice between AR and NAR methods is not clear-cut. For many domains, high-quality models exist using both approaches but we can define some crucial parameters. Some NAR methods such as GAN or VAE are capable of generating output in only one inference step, however they are typically hard to train on certain data modalities (e.g. text data) comparing to AR counterparts. Other NAR methods such as diffusion models typically allow for choosing a diffusion length during inference, which is independent of that used at training. Choosing a low diffusion length can frequently lead to poor sample quality, and tuning this setting (among many others) is critical to high quality generation. However if the tuned diffusion length for a given sequence (of length T) is shorter than the length of those sequences, the NAR method has a computational advantage over the equivalent AR model (which would require T steps for a T length sequence). In addition,

the ability to tune this diffusion length can be useful in interactive applications, or when a variety of output is desirable. This setting can also be a curse, as even well-trained models perform poorly with improper diffusion settings. Several branches of current research are focused on improving guarantees and convergence speed for diffusion models (Huang et al., 2021; Song et al., 2021; Kingma et al., 2021; Lam et al., 2022).

8.1.4 SUNMASK, a non-autoregressive sequence model

We introduce SUNMASK, a NAR sequence model which uses masks over noised, discrete data to learn a self-improvement operator which transitions from categorical noise toward the data distribution in iterated steps. Given a target data representation, we train a model which can map from a noisy version of input data to a corrected form of the input. In this work, we use multinomial noise - namely entries are corrupted to 1 of P possible values (for a given set size P), with the number of noised entries in a sequence defining the relative noise level for the training example. This is similar to many diffusion approaches at a high level, and particularly shown to be an effective tool in SUNDAE (Savinov et al., 2022) and Coconet (Huang et al., 2017b). In addition to the use of multinomial noise, we also form a mask representing *where* the data was noised, feeding this mask alongside the input data to form a conditional probability distribution.

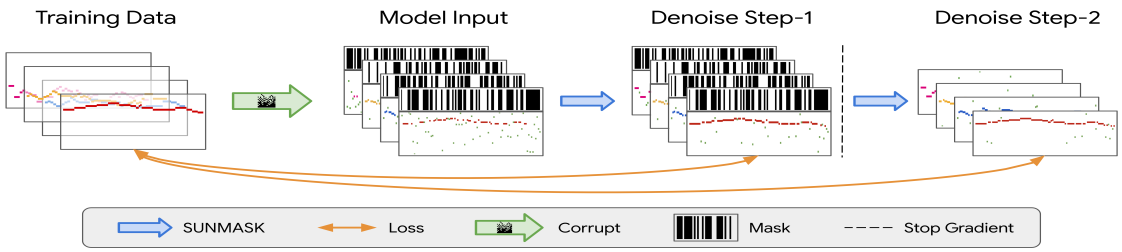

Figure 8.1 – Step-unrolled denoising training for SUNMASK on polyphonic music, unrolled step length 2. Training data (left) consists of four voices corrupted by sampling a random mask per voice and replacing the masked data (red) with random pitches (green). SUNMASK takes both mask and corrupted training data as input, predicting denoised original data as output. In the second step, the model takes a sampled version of the model step predictions and the same mask as input, outputting another prediction of the original data.

8.2 Method

The relationship between discrete diffusion and denoising autoencoders has been explored in previous work (Hoogeboom et al., 2022; Savinov et al., 2022; Austin et al., 2021; Hoogeboom et al., 2021). We build upon this foundation, combined with many insights from prior orderless modeling approaches, crucially Orderless NADE (Uria et al., 2014), Coconet (Huang et al., 2017b) (which is a more modern variant of Orderless NADE), and SUNDAE (Savinov et al., 2022).

SUNMASK is built around a process $x_t \sim f_\theta(\cdot|x_{t-1}; m)$ on a space $X = \{1, \ldots, v\}^N$ of arrays with categorical variables. This parametric transition function f_θ takes an additional argument $m \in 0, 1^N$. During training, m indicates variables that were not initially corrupted, and as a consequence we can use it during inference to tell f_θ which variables to trust.

Given a sequence of masks $m_0, \ldots, m_{T-1}$, the generating distribution of our model derives from a prior p_0 (typically uniform noise) and repeated application of f_θ:

$$p_T(x_T; m_0, \ldots, m_{T-1}) = \Big(\sum_{x_1, \ldots, x_{T-1} \in X} \prod_{t=1}^{T} f_\theta(x_t|x_{t-1}; m_{t-1}) \Big) p_0(x_0) \tag{8.1}$$

In practice, p_0 is typically elementwise iid uniform noise, and the masks $m_0, \ldots, m_{T-1}$ are drawn according to a schedule and may be held constant for several steps.

To train f_θ, we take a training example $x \sim p_{\text{data}}$ and draw a mask m. We apply the corruption procedure $x_0 \sim q(\cdot|x; m)$ to obtain x_0 which equals x where the mask m is true and uniform random values elsewhere. Then we iterate $x_t \sim f_\theta(\cdot|x_{t-1}; m)$ with the aim of reconstructing x.

As in SUNDAE, the transition f_θ models the variables as conditionally independent of one another. However SUNDAE has no direct concept of masking. SUNMASK thus combines past insights from the masked NAR models Orderless NADE and Coconet with existing concepts from SUNDAE, along with new model classes and inference schemes to form a powerful generative model. Similar to SUNDAE, our objective is to minimize $\frac{1}{2}(L^{(1)} + L^{(2)})$ where

$$L^{(t)}(\theta) = -\mathbb{E}_{m_0,\cdots,m_{t-1}}\mathbb{E}_{\substack{x \sim p_{\text{data}} \\ x_0 \sim q(\cdot|x,m_0) \\ x_1 \sim f_\theta(\cdot|x_0;m_0) \\ x_2 \sim f_\theta(\cdot|x_1;m_1) \\ \cdots \\ x_{t-1} \sim f_\theta(\cdot|x_{t-2};m_{t-2})}}\left[\frac{\sum_i (1-m_{t-1}^{(i)}) \log f_\theta^{(i)}(x^{(i)}|x_{t-1};m_{t-1})}{\sum_i 1-m_{t-1}^{(i)}}\right] \tag{8.2}$$

is the reconstruction loss for the elements of x that were corrupted. As in Orderless NADE (Uria et al., 2014) and Coconet (Huang et al., 2017b), we weigh each term according to the size of the mask, to ensure that the overall weight on each conditional $f_\theta^{(i)}$ is uniform across i. Unlike previous methods, we target *only masked variables* in the loss. In practice we choose $m_0 = \cdots = m_{t-1}$ during training and $t = 2$. Since we only go to $t = 2$, keeping the mask constant is a close enough approximation to the masking schedule used in inference. The choice of $t = 2$ is driven by the ablation study in SUNDAE, where $t = 2$ was found to account for nearly all performance gains in translation experiments, with higher unrollings showing no additional benefit. In addition higher values of t unrolling generally increase memory usage, making the training of high order unrollings complicated.

SUNMASK allows for direct control at inference using both proposal masks and noising of variables, combining elements of both SUNDAE and Coconet. We show a high level example of the unrolled training scheme, mask proposals, and input data processing in Figure 8.1.

The overall unrolled mask and iterative inference setting is largely independent of architecture choice, and as long as the internal architecture does not make any ordering assumption over the input data we can incorporate it into SUNMASK. We use two primary archetypes for the internal model in this paper: Attentional U-Net and Relative Transformer.

SUNMASK uses an unrolled training scheme, similar to that shown in SUNDAE, as well as a mask which is input to the model and defines manipulated variables as in Coconet. The loss is masked based on this manipulation mask, unlike Coconet or SUNDAE. The SUNMASK loss is further weighted by the total amount of masked variables. Comparisons of various high level modeling features between SUNMASK, Coconet, and SUNDAE are shown in Table 8.1.

Table 8.1 – Comparing SUNMASK, Coconet, and SUNDAE

Model	SUNMASK	Coconet	SUNDAE
Mask input to model	✓	✓	X
Masked loss	✓	X	X
Re-weighted loss	✓	✓	X
Unrolled loss	✓	X	✓
Inference mask schedule	✓	✓	X
Sampling rejection step	✓	X	✓
Mask control preserves data	✓	X	X

8.2.1 Model Training

During training, the internal architecture is combined with a *step unrolled* training procedure, as highlighted by SUNDAE (Savinov et al., 2022). Rather than directly randomizing positions, we re-write this as a masking scheme, first sampling a mask (with 0 randomize, 1 keep, which we denote as 0-active format) then performing randomization to one of P possibilities, for the masked subset of K variables. This random masking procedure is equivalent to the approach from SUNDAE, but using a mask allows us to further combine the mask information with the input data, in order to form a conditional probability estimate. In addition, this 0-active masking scheme makes direct comparison to masking schemes with absorbing states (such as OrderlessNADE (Uria et al., 2014), Coconet (Huang et al., 2017b), VQ-Diffusion (Gu et al., 2022) and OA-ARDM (Hoogeboom et al., 2022)) simpler, as the mask can be directly multiplied with the data in a 0-active format.

Each training batch is randomly sampled from the training dataset, and a corresponding noise value drawn from $rand(N)$ for N examples in the minibatch. This per-example noise value is then used to derive a per-step mask over T timesteps, by comparing noise $rand(N) < rand(N, T)$. During training, this means some examples have a high per-example noise value (e.g. .99), and thus many values masked and noised in the training, while other examples may have a low noise value (e.g. .01) drawn instead. Combined with a training loss which learns to denoise the input and focuses on imputing information about masked corrupted inputs, the overall model will learn a chain to go from more noisy data to less noisy step-wise, resulting in a learned improvement operator (Hoogeboom et al., 2021; Savinov et al., 2022).

This improvement operator can be applied to noisy data or pure noise, and iterate toward a predictive sample from the training distribution. See Multinomial

Diffusion (Hoogeboom et al., 2021) and SUNDAE (Savinov et al., 2022) for more detail on this proof, as well as fundamental work on denoising autoencoders (Alain and Bengio, 2014). In SUNMASK, we combine the mask used to noise the input with the input data itself, while modifying the loss to predict *only masked variables*. In addition, we downweight the loss by $\frac{1}{1+\sum 1-m_t}$ for each batch element, meaning that losses for heavily masked entries are downweighted compared to losses on examples with little masking, in a form of curriculum weighting based on expected estimation difficulty.

While a one step denoising scheme can be sufficient for learning the data manifold (Austin et al., 2021; Alain and Bengio, 2014), *unrolling* this denoising scheme into a multi-step process can have performance benefits. SUNMASK directly uses the unrolled loop scheme described in (Savinov et al., 2022), using a step value of 2. For a detailed description of the step unrolled training scheme, see the overview description from SUNDAE (Savinov et al., 2022). The masked and unrolled training can be seen as a container for any internal model which does not make ordering assumptions, and we utilize both convolutional U-Net (a variant of the GLIDE (Nichol et al., 2022) U-Net) and Relative Transformer (Dai et al., 2019; Huang et al., 2018; Payne, 2019) models for various experiments, shown in Section 8.4.

8.2.2 Convolutional SUNMASK

SUNMASK is most closely related to Coconet (Huang et al., 2017b) and SUNDAE (Savinov et al., 2022). Coconet (as an instance of OrderlessNADE using convolutional networks), trains by sampling a random mask per training example, using this mask to set part of the input (in one hot format) to zero. The mask is further concatenated to the zeroed data along the channel axis, and this combined batch is passed through a deep convolutional network with small 3×3 kernels. Convolutional SUNMASK uses a downweighted loss over only variables masked in the input. However, SUNMASK additionally uses the unrolled training scheme, as well as a different inference procedure due to preserving the values of masked out variables during training and sampling.

Our best performing convolutional SUNMASK architecture takes hints from recent image transformer and vector quantized generators, exchanging the small kernels used in Coconet for extremely large kernels of shape $4 \times P$ over the time and feature dimensions, somewhat analogous to input patches, removing the model's

translation invariance over the feature axis by setting kernel dimension equal to the total feature size. However this makes the number of parameters per convolutional layer extremely large. Convolutional SUNMASK adopts an attentional U-Net structure which reduces only across the time axis, modified from GLIDE (Nichol et al., 2022), rather than the deep residual convolution network used by Coconet. Combined with the addition of step unrolled training, we are only able to train convolutional SUNMASK with a batch size of 1 (expanded to effective batch size 2 due to step unrolling) on commodity GPU hardware with 16GB VRAM.

Due to the design choice of extremely large kernel sizes which depend on the size of the domain, we only use convolutional SUNMASK for polyphonic music experiments, see Section 8.4 for more details.

Attention is applied on the innermost U-Net block size as well as the middle block, with 1 attention head (Nichol et al., 2022). Convolutions are used in all resampling, and all resampling happens only on the time axis, making the Attentional U-Net effectively a 1-D architecture. However, rather than learning both instrument and pitch relations across channels, we isolate pitch relations and instrument relations into separate axes of the overall processing, the "width" and "channels" axes, respectively assuming $(N, C, H, W) == (N, I, T, P)$ axes. As is standard in many U-Net designs, we double the number of hidden values for layers every time the resolution is halved, with the reverse process being used when upsampling. Though the parameter count here is large, it is similar in spirit to other approaches to small datasets on text (Al-Rfou et al., 2019).

8.2.3 Transformer SUNMASK

Transformer SUNMASK relates closely to the transformer used in SUNDAE. The architecture uses a relative multi-head attention (Dai et al., 2019; Huang et al., 2018) and has no autoregressive masking. SUNMASK transformer also uses larger batch sizes, typically 20 or larger, though this is far smaller than the batch sizes seen in the experiments of SUNDAE. Sequence length and data iterator strategy were both a critical aspect for training transformer SUNMASK. We found short sequences (from 32 to 128) worked best, along with iteration strategies that were example based.

Transformer SUNMASK was trained on every dataset used in this paper, and we show performance in Section 8.4, as well as comparisons to convolutional SUN-

MASK on symbolic polyphonic music modeling. Both convolutional and transformer based SUNMASK use the Adam optimizer, with gradient clipping by value at 3. Inference hyperparameter types and general sampling strategies used are the same with both models, though specific hyperparameter values may change between datasets.

There is a large discrepancy in model parameter count between our best performing convolutional models for JSB, and our best transformers. Training larger transformers can work well for generation (Al-Rfou et al., 2019), but our large parameter transformers (on the order of 400M parameters) had poor generative performance on JSB.

Pitch size / vocabulary size, sequence length, and batch size changed for the transformers used in the text experiments, but the global architecture remained in the style of "decoder only" transformers (Radford and Wu, 2019), similar to SUNDAE. Notably, we use vocabulary size $5.7k$, sequence length 52, batch size 48 for EMNLP2017 News and vocabulary size 27, sequence length 64, batch size 20, and a slightly extended training step length of 150000 for text8.

8.2.4 Inference Specific Settings

Well-trained SUNMASK models should be applicable to full content generation, as well as a variety of partially conditional generative tasks such as infilling and human-in-the-loop creation. Basic sampling involves creating a set of variables, with all variables randomly set to 1 of P values in the domain (or partial randomization in the case of infilling) along with an accompanying mask, which is initially all 0 for full generation, or mixed 1s and 0s for partial generation tasks. Given this data and mask as input, the trained model then predicts a probability distribution over all possible P values, for all variables. Despite the use of masked losses in training, we sample these prediction distributions for *all* variables. These predictions are then accepted or rejected from the original set, resulting in a new variable set. We then sample a new mask (based on a predefined schedule) and combine it with the initial mask, then iterate this overall process, updating at least some of the variables at each step.

During inference we use several key techniques to improve generative quality. We use typicality sampling (Meister et al., 2022) on the output probability dis-

tribution and a variable number of diffusion steps, on the order of 100 to 2000. Masks are randomly sampled using the schedule defined in (Huang et al., 2017b) which linearly decreases the number of masked variables over time according to $\alpha_n = \max(\alpha_{\min}, \alpha_{\max} - \frac{n}{\eta N}(\alpha_{\max} - \alpha_{\min}))$ with $\alpha_{\min} = .001$, $\alpha_{\max} = .999$, and $\eta = 34$, along with an optional triangular linear ramp-up and ramp-down schedule for the probability of accepting predictions from the model into the current variable set at each step, as shown in (Savinov et al., 2022).

Tuning hyperparameters for inference is critical to success, as improper settings can drastically lower the performance of SUNMASK, see Section 8.4 for variance over various inference settings in different tasks. For human-in-the-loop applications, the existence of these controls can allow a number of fine-grained workflows to emerge, driven by expert users to create and curate interesting output (Esser et al., 2021; Crowson et al., 2022), demonstrated in Figure 8.3.

8.3 Related Work

We state here some key related approaches, as well as how our method differentiates from these previous settings. A number of recent publications on diffusion models and feature learning have incorporated masks as part of their overall training scheme (Hoogeboom et al., 2022; He et al., 2022), however these papers use masks for blanking, rather than as indicators over stochastic variables. Many infilling models (Donahue et al., 2020; Devlin et al., 2019), and masked image models (He et al., 2022) feature conditional modeling with a mask (blank) token, predicting the variables masked from the input for feature learning or generative modeling. XLNet (Yang et al., 2019) combines the infilling and autoregressive paradigms, learning arbitrary permuted orders over masked out variables, using blank-out masking and randomly generated autoregressive ordering similar to OrderlessNADE and Coconet. Conditional diffusion generators (Meng et al., 2021) and GAN generators (Fedus et al., 2018) have the combination of mask indicators as well as preserving stochasticity of the masked variables. However these methods do not use an unrolled training scheme, and generally target image related tasks, with the notable exception of maskGAN. Many models use a concept of a working canvas, and do repeated inference steps for generation or correction of data (Gregor

et al., 2015b; Bachman and Precup, 2015; Ganin et al., 2018), SUNMASK differs from these models due to architecture choices, training scheme, and loss weighting, as well as application domain (Mittal et al., 2021; Pati et al., 2019; Rombach et al., 2021; Nichol et al., 2022).

8.4 Experiments

We demonstrate the use of SUNMASK for polyphonic symbolic music modeling on the JSB dataset (Allan and Williams, 2004; Boulanger-lewandowski et al., 2012). The JSB dataset consists of 382 four-part chorales, originally written by Johann Sebastian Bach. These chorales are quantized at the 16th note interval, cut into non-overlapping chunks of length 128, skipping chunks which would cross the end of a piece. This processing results in a training dataset of 4956 examples, with each example being size $(4, 128)$. We train convolutional and transformer versions of both SUNMASK and SUNDAE for comparison, as well as the pretrained Coconet (Huang et al., 2017b). For polyphonic music, the quantized data was rasterized in soprano, alto, tenor, bass (SATB) order, as in Music Transformer (Huang et al., 2018) and BachBot (Liang, 2016), then chunked into non-overlapping training examples. Results are shown in Table 8.13. These results are evaluated on Bach ground truth data (Bach GT), BachMock Transformer (BachMock (Fang et al., 2020; Liu et al., 2020)) (closely related to the decoder from VQ-CPC (Hadjeres and Crestel, 2020)), Coconet, SUNDAE (SD), and SUNMASK convolutional (SMc) and SUNMASK transformer (SMt). Model sampling variants are indicated as Typical Sampling (T).

8.4.1 Musical Evaluation

The grading function used for evaluation, referred to as BachMock here, is designed specifically to correlate with expert analysis on Bach. In particular using this metric to choose correct examples in a paired comparison test, outperforms novice, intermediate, and expert listeners by varying margins (Fang et al., 2020). This indicates that scoring well on the aggregate metric should correlate to high sample quality. The metric has many sub-parts, ranking various musical attributes

Table 8.2 – Quantitative results from the Bach Mock grading function (Fang et al., 2020). Top rows compare to existing literature, bottom rows show ablation study of SUNMASK style models. Lower values represent better chorales.

Model	Note	Rhythm	Parallel Errors	Harmonic Quality	Interval	Repeated Sequence	**Overall**
Bach Data	0.24 ±0.15	0.23±0.14	0.0±0.69	0.41±0.2	0.55±0.4	1.29±0.88	*4.91±1.63*
BachMock	0.37±0.22	0.26±0.14	2.16±3.22	0.54±0.31	0.71±0.68	1.86±2.81	8.94±4.64
SMc-T BEST20	0.39±0.16	0.53±0.26	0.0±0.81	0.68±0.27	0.75±0.42	1.44±0.52	**7.16±0.97**
AugGen	-	-	-	-	-	-	8.02±2.92
Coconet	**0.44±0.23**	1.85±0.39	2.61±6.56	1.38±0.39	**0.86±0.73**	6.07±1.76	17.00±6.58
SD	0.59±1.82	0.93±0.84	6.42±4.11	0.98±0.67	1.99±5.68	2.45±2.39	23.25±21.45
SD-T	0.63±2.40	0.60±0.96	3.82±4.98	0.96±0.64	2.50±5.03	**1.52±3.43**	20.09±23.88
SMc	0.87±2.05	0.63±0.77	1.38±6.00	1.02±0.49	2.07±5.72	2.32±2.31	22.47±20.80
SMc-T	0.57±1.79	0.69±0.35	1.28±3.73	**0.93±0.49**	1.10±4.68	1.81±0.83	**13.43±19.27**
SMt	3.00±1.85	0.74±0.90	**0.00±1.95**	1.64±0.70	7.90±5.58	3.10±2.97	42.87
SMt-T	3.74±2.16	**0.58±0.56**	0.00±2.56	1.73±0.73	7.74±4.73	2.35±1.79	46.21±17.30

crucial to codifying the style of J.S. Bach. AugGen (Liu et al., 2020) incorporated this metric into an iterative training and sampling scheme which improved final generative capability for a fixed model, showing the effectiveness of BachMock in practice for ranking machine generated samples. For each grading function in the Bach Mock grading evaluation, we show the median value and ± standard deviation (showing the average of each interval SATB performance for brevity), as well as the overall grade. Lower values forall metrics are better, and we see the strongest results for convolutional SUNMASK with typicality sampling. Combined with final top-N ($N = 20$) selection out of a candidate set of 200 samples, the overall sample quality outperforms strong baselines. This high quality subset (SMc-T BEST20) rivals both the "BachMock" transformer and the dataset itself on this metric. We find SUNMASK generations are qualitatively good and listenable overall, even though some SUNMASK samples do fare poorly by the grading metrics.

8.4.2 Text Datasets

The EMNLP 2017 News dataset is a common benchmark for word-level language modeling (Caccia et al., 2020), containing a large number of news article sentences (Lu et al., 2018). Preprocessing steps collapse to sentences containing the most common 5700 words, resulting in a training set of $200k$ sentences with a test set of $10k$. The overall maximum sentence length is 51. Common processing for this dataset includes padding all sentences up to this maximum length, different than the standard long sequence chunking commonly used in other language modeling tasks.

We show the results of several SUNMASK models for generating sentences similar to EMNLP2017News, comparing to benchmarks using the standard Negative BLEU/Self-BLEU evaluation (Zhu et al., 2018; Caccia et al., 2020) over generated corpora of 1000 sentences in Figure 8.2. This set of scores, varied across temperature, is compared against baseline scores (Lin et al., 2017; Yu et al., 2017; Che et al., 2017; Guo et al., 2020; de Masson d'Autume et al., 2019; Vaswani et al., 2017), similar to the evaluation shown in SUNDAE (Savinov et al., 2022). These reference benchmarks used 10000 sentences to form performance estimates.

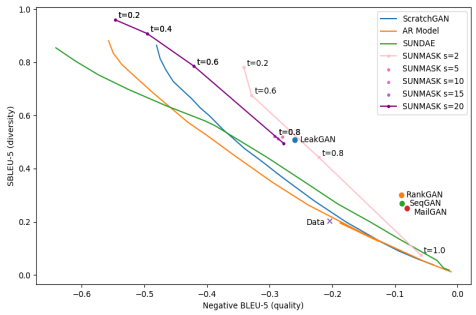

Figure 8.2 – Negative BLEU/Self-BLEU scores on EMNLP2017 News. Left (x-axis) is better, lower (y-axis) is better. Quality/variation is controlled by changing the temperature (t), and varying diffusion schedule (s). For SUNMASK, *typical* sampling results (Meister et al., 2022) are shown.

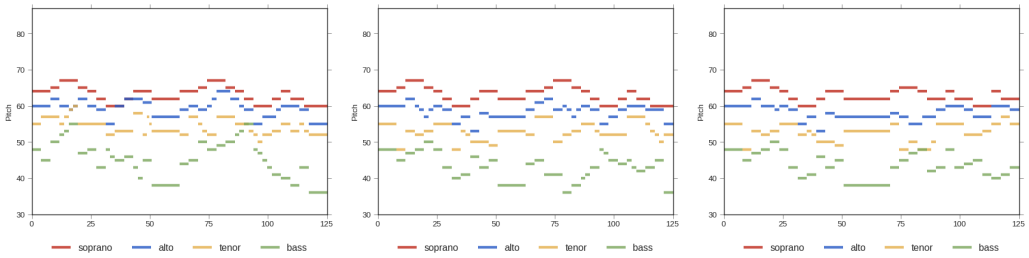

Figure 8.3 – SUNMASK harmonization (bass, tenor, alto) of an existing melody (left), with a mask which highlights the left half (0 to 64) soprano voice (middle), or a left half mask but replacing right half melody as well (right)

8.4.3 Music Control

Given the flexibility of masking at inference, we perform a number of qualitative queries to inspect how the model adapts based on noise and mask value. Figure 8.3 demonstrates the use of SUNMASK for musical inpainting, holding the top voice (soprano) either fully or partially fixed to the well-known melody "Ode to Joy", by Ludwig van Beethoven.

8.4.4 Text Control

Masking can also used to variably increase or decrease the weight on various pre-specified terms, held fixed throughout inference. The combination of these words, and their mask status can be seen to influence the overall tone of the selected text passages which showed the strongest effect in a particular inference batch. The following qualitative samples using masks for word influence are drawn from SUNMASK Transformer on EMNLP2017News dataset. Though the generation quality is flawed, we clearly see a relationship between the masked word and the emergent surrounding context, for example highlighting **disaster** draws forth injured, displaced, and pressure, while **success** instead references happy, nice, good, and playing.

Success unmasked, *disaster* masked

— I think I want to leave **success** at the end of the *disaster* , but because that ' s a nice to say it ' s not good to be the challenge and this is a very good thing <eos>

— That was the job I was **success** to have to pay my *disaster* but hopefully I have been able to pull playing in the first couple of the season , I ' ve been happy to go through this team , he said <eos>

Success masked, **disaster** unmasked

— Although more than 80 , 000 *success* have been displaced in the **disaster** since the last year , more than 700 , 000 lives have been injured in the country , and 70 of them were killed , according to the UN media <eos>

— I haven ' t had a *success* at the league , the **disaster** and picked running with the door ago we have Champions , and I was a couple of pressure . . . and it was a lot of times <eos>

8.5 Conclusion

We introduce SUNMASK, a method for masked unrolled denoising modeling of structured data. SUNMASK separates the role of masking and correction by conditioning predictions on the mask, allowing for fine-grained control at inference.

When applied to text as well as symbolic polyphonic music, SUNMASK is competitive with strong baselines, outperforming reference baselines on music modeling. Leveraging the separation of mask and noise allows for subtle control at inference, paving the way for a variety of domain specific applications and generative pipelines for human-in-the-loop creation.

8.6 Appendix

8.6.1 Musical Co-Creation and Possible Ethical Concerns

A chief concern in generative co-creation as demonstrated by our music sampling, is direct plagiarism through the training corpus or indirect plagiarism of outside work. Direct plagiarism has a number of mitigation strategies, either with exact matches, approximate dataset matches (Papadopoulos et al., 2014) (note sequences across voices, regardless of duration), or secondary tools such as automated copyright matchers. The latter category (automated copyright matchers) can also be used for the most difficult plagiarism - indirect plagiarism.

Due to most Western music sharing similar underlying rules and structure, it is possible to accidentally stumble upon a copyrighted work without a version of that work ever appearing in the training corpus. This goes especially for models trained on foundational scores, for example the underlying harmonic rules J.S. Bach followed and popularized underwrite a vast swath of the classical canon. Direct debut of any generative co-creation tool should have at least some consideration for tagging or labeling possible matches and conflicts, letting creators inspect the relevant matches to decide for themselves if there is an issue which warrants modification. Given the complexities of musical copyright, there is no clear cut automated solution but using recognition tools to provide information can help mitigate surprise issues for end users (Briot and Pachet, 2020).

The music dataset used in this work (JSB) is fully in the public domain, as such the ethical concerns listed above are minimized, and any secondary issues related to music copyright are unlikely to be a problem with the direct output of this model. The short, quantized MIDI-style output from SUNMASK is a not a suitable

format for general listening, needing substantial post-processing, combination, and interpretation in order to form a musical piece (Sturm et al., 2019).

8.6.2 Text

There are many complexities around generative modeling of language, and especially with generative co-creation of text. The particular datasets and schemes used in this work are relatively limited compared to more direct and large scale applications of language models, however it is always a key concern to think about the limitations and biases of the underlying datasets used to train these models.

Given the propensity for using SUNMASK and related methods to infill given context, certain applications should have strong investigations into the biases and private information present in the underlying data. Imputing missing information in order to strengthen downstream classification (not directly shown in this work, but certainly possible) can be problematic with respect to imputed features amplifying underlying biases in the training corpus, or violating user privacy. Many mitigation and detection strategies proposed by researchers in fairness, bias, and privacy in machine learning (Smith et al., 2022) should be directly applicable to SUNMASK if deemed necessary, given the commonality of the modeling and training schemes to other well studied methods such as AR transformers, and deep learning more generally.

The text datasets used in our experiments are a common benchmark, chosen to enable comparison to existing work. Any underlying biases or issues with models trained on these particular datasets will be shared across *many* generative language models, and any proposed corrections specific to these datasets should be directly applicable to SUNMASK.

8.7 Convolutional SUNMASK Model Hyperparameters and Training Information

Training	
Input channels	4
Pitch count	57
Sequence length	128
Training steps	50000
Batch size	1
Unrolled steps	2
Hidden size	64
Kernel size	(4, 57)
Block scales	(1, 1, 2, 4)
Residual layers per block	3
Optimizer	Adam
Learning rate	1E-4
Total parameter count	417M
Downweight multiplier	.75
Downweight learning rate steps	5000

8.7.1 Architecture Design

Attention is applied on the innermost U-Net block size as well as the middle block, with 1 attention head (Nichol et al., 2022). Convolutions are used in all resampling, and all resampling happens only on the time axis, making the Attentional U-Net effectively a 1-D architecture. However, rather than learning both instrument and pitch relations across channels, we isolate pitch relations and instrument relations into separate axes of the overall processing, the "width" and "channels" axes, respectively assuming $(N, C, H, W) == (N, I, T, P)$ axes. As is standard in many U-Net designs, we double the number of hidden values for layers every time the resolution is halved, with the reverse process being used when upsampling. Though the parameter count here is large, it is similar in spirit to other approaches to small datasets on text (Al-Rfou et al., 2019).

Sampling	
Temperature	.6
Steps	$2 \times I \times T = 2 \times 4 \times 128$
Mask dwell	1
Active balance	False
Final mask dwell	0
Keep prob	”triangular”
Sampling	typical
Top k	3
Top p	False
Intermediate noise	False
Mask max	.999
Mask min	.001

8.7.2 Sampling Details

All parameters and sampling designs were tuned based on generated sample quality, rather than direct tuning to the grading function used for final metric calculation. It is likely that these numbers could be greatly improved, but tuning directly against this metric may also result in less musical samples that exploit quirks in the metric calculation.

During sampling we have a number of additional parameters to set. Crucially, the use of typical sampling (Meister et al., 2022) and strong filtering (either small top-k, small top-p, or both) resulted in generally stronger samples than both the equivalent, or typical sampling with looser settings. We note that the importance of this setting is demonstrated in the paper introducing typical sampling, in the difference between settings for summarization and story generation. These settings also interact heavily with the temperature setting.

When typical sampling top-k values or top-p values are too large, we see samples end up with the same result for either typical or standard sampling, so setting these filters is critical to see the full impact of typical sampling.

Triangular keep probability is described in SUNDAE, and we utilize it here as well, linearly ramping accept probability from 0 to 1 at $\frac{steps}{2}$, ramping back down to 0 at $\frac{steps}{}$. The keep probability schedule excludes the optional ”final mask dwell”, which we only utilize alongside intermediate noise. We also found fixed keep probabilities (such as .33, .5, and even 1.0) also performed well.

Mask proposals follow the scheme proposed by Coconet, sampling bernoulli masks with probability p, ramping from the mask min to mask max, over the total range of *steps*. These masks are further combined with pre-specified masks, specifically we allow two types of secondary specification. Focus masks hold the input value fixed at every diffusion step, and always specify a mask value of 1 in the model input. Keep masks allow a mask value of 0 or 1 (depending on the bernoulli random sampled mask) in the model input, but the value will be reset to the specified input value at each step of diffusion. The allowance of these two different mask types is unique to SUNMASK, and seems to have a large impact on the sampled outcome based on our testing.

When using intermediate noise, it is beneficial to set a final mask dwell, which holds the last mask (which is set to all 1) constant and then samples repeatedly to form final corrections. During final mask dwell, we set accept probability to 1, as triangular sampling would by default set accept probability to near 0. Intermediate noise is effectively disabled everywhere the mask is 1, so this is similar in spirit to noise tapering in other applications using gaussian style noise, and allows the SUNMASK learned improvement operator to make small changes and fixes to the general "skeleton" of the proposed sample.

8.8 Transformer SUNMASK Model Hyperparameters and Training Information

There is a large discrepancy in model parameter count between our best performing convolutional models for JSB, and our best transformers. Training larger transformers can work well for generation (Al-Rfou et al., 2019), but our large parameter transformers (on the order of 400M parameters) had poor generative performance on JSB. Given the foundational work in SUNDAE, it is clear that it should be possible to train these larger transformer models well, and finding the correct recipe may drastically improve the quality of the transformer generated musical examples.

Pitch size / vocabulary size, sequence length, and batch size changed for the transformers used in the text experiments. Notably, we use vocabulary size $5.7k$,

Training	
Pitch count	57
Training steps	120000
Sequence length (JSB)	256
Batch size (JSB)	20
Unrolled steps	2
Transformer layers	16
Embedding dim	512
Transformer input dim	512
Transformer hidden dim	2048
Transformer attention heads	8
Transformer head dim	64
Optimizer	Adam
Learning rate max	5E-5
Learning rate max	5E-6
Ramp up steps (min to max)	5000
Ramp down steps (max to min)	100000
Gradient clip (value)	3
Total parameter count	50M

sequence length 52, batch size 48 for EMNLP2017 News and vocabulary size 27, sequence length 64, batch size 20, and a slightly extended training step length of 150000 for text8.

8.9 Sampling Runtime

One chief drawback of the currently implemented SUNMASK models, primarily the convolutional SUNMASK used in JSB, is the time to sample. In part due to the parameter count, as well as the non-standard details of the architecture (kernel size in particular), sampling runs at roughly 8× slower than real-time, taking approximately 4 minutes to generate a 38 second sample (4 voices, each with 128 steps, at 16 steps per measure quantization) on V100 GPUs. On A100 GPUs, this goes directly to 66 seconds to generate the same size sample, which gives some indication that simply updating hardware may radically improve the runtime of convolutional SUNMASK. Increasing batch sizes improves the effective amortized

sample speed, but at an increase of latency. In addition, the non-causal nature of diffusion sampling means that pipelined sampling to reduce the effective latency felt by end-users is not easily applicable, compared to standard AR methods.

However there are many direct optimizations available for these architectures from both the computer vision literature at large, and specifically for symbolic music modeling (Huang, Hawthorne, Roberts, Dinculescu, Wexler, Hong, and Howcroft, Huang et al.). More exploration of computational improvements, toward fully interactive use remain a key research direction.

8.10 Code repository and samples player

We attach several folders of samples (in midi format) from our model for music, as well as the evaluation sentences for BLEU / self-BLEU testing on EMNLP2017 News as part of the supplemental material.

Full reproduction code and sample listening page can be found at
https://github.com/SUNMASK-web/SUNMASK

8.11 Creating a "Greatest Hits"

Music demonstrations of the model labeled "BachMock" transformer can be heard at https://alisawuffles.github.io/post/grading-function/. We find SUNMASK generations are qualitatively on a similar level as these sample generations, though some SUNMASK samples do fare poorly by the grading metrics. However the best SUNMASK samples have remarkably good grades, on a similar level as the best "BachMock" samples shown in the linked post, and indeed to a similar level as the data itself.

Of particular note is the high variance in the grade of all SUNMASK models compared to either Coconet, or the baselines. Given the existence of the grader function, it is possible to prune generations from our SUNMASK diffusion models to improve the overall output. Generating 200 samples from the best SUNMASK method, and pruning to only the top 20 overall grades, we see that it is possible

to produce high quality subsets which rival the "BachMock" transformer and the dataset itself on this metric.

Model	Note	Rhythm	Parallel Errors	Harmonic Quality	S Intervals	A Intervals	T Intervals	B Intervals	Repeated Sequence	**Overall**
Bach GT	0.24 (0.15)	0.23 (0.14)	0.0 (0.69)	0.41 (0.2)	0.47 (0.28)	0.49 (0.22)	0.53 (0.24)	0.69 (0.4)	1.29 (0.88)	4.91 (1.63)
BachMock	0.37 (0.22)	0.26 (0.14)	2.16 (3.22)	0.54 (0.31)	0.53 (0.35)	0.71 (0.34)	0.73 (0.38)	0.89 (0.68)	1.86 (2.81)	8.94 (4.64)
SMc-T	0.57 (1.79)	0.69 (0.35)	1.28 (3.73)	0.93 (0.49)	0.80 (4.51)	0.99 (4.01)	1.20 (4.68)	1.40 (3.91)	1.81 (0.83)	13.43 (19.27)
SMc-T-BEST20-200	**0.39 (0.16)**	**0.53 (0.26)**	**0.0 (0.81)**	**0.68 (0.27)**	**0.59 (0.25)**	**0.88 (0.42)**	**0.80 (0.20)**	**0.71 (0.27)**	**1.44 (0.52)**	**7.16 (0.97)**

While selecting directly against this metric makes it a less useful ranking scheme for the various methods, listening to the resulting samples also reveals that this ranked subset are also qualitatively among the best of this cohort. We stress that this simple generate-and-test method could be used with all available models, and potentially as part of training itself, as in published work on augmented generative training (Liu et al., 2020).

Given that the best scoring example from Coconet has an overall grade of 12.26, the best SUNDAE example (SD-AT in main table) grade 7.78, best SUNMASK example from the small set (SMc-T in main table) grade 7.14, and the best SUNMASK example grade from the larger 200 set 4.93 it seems SUNMASK may be a better candidate for this kind of scheme due to higher generated sample variance. Generating more samples and then curating a top performing subset should yield better scores for all methods tested. Comparing this approach against a broader swath of high-performance, template based infilling methods (Hadjeres and Crestel, 2020; Bretan et al., 2016, 2017) remains an important future direction.

8.12 Full Masking Comparison Figure

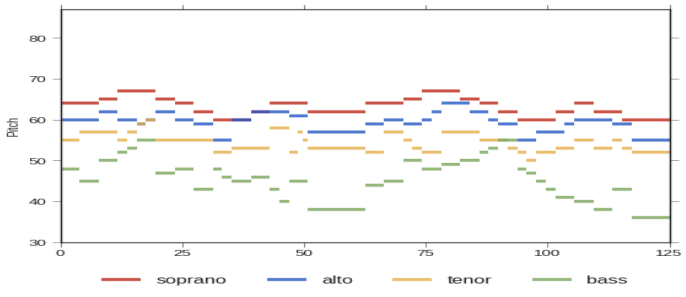

Figure 8.4 – SUNMASK harmonization (bass, tenor, alto) of existing melody (soprano) based on mask.

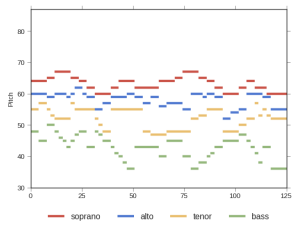

Figure 8.5 – A

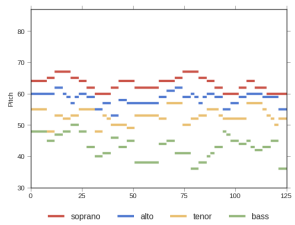

Figure 8.6 – B

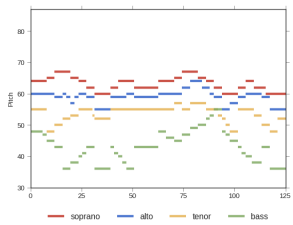

Figure 8.7 – C

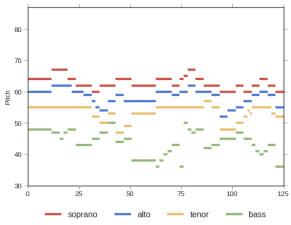

Figure 8.8 – D

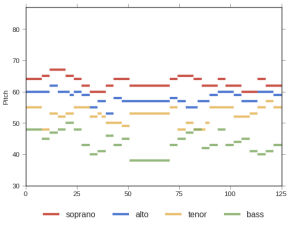

Figure 8.9 – E

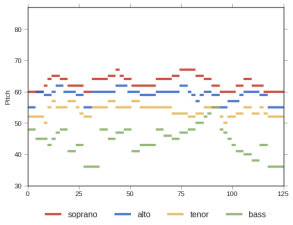

Figure 8.10 – F

A: SUNMASK harmonization (bass, tenor, alto) of an existing melody (soprano), based on pre-defined mask which only highlights the first beat of each measure (0 through 4 of every 16 on the x-axis), but preserves all notes of the melody by overriding predictions in that voice each step.
B: SUNMASK harmonization (bass, tenor, alto) of an existing melody (soprano), based on pre-defined mask which only highlights the first half of the melody (0 through 64 on the x-axis), but preserves all notes of the melody by overriding predictions in that voice at each diffusion step.
C: SUNMASK harmonization (bass, tenor, alto) of an existing melody (soprano), based on pre-defined mask which only highlights the last half of the melody (64 through 128 on the x-axis), but preserves all notes of the melody by overriding predictions in that voice at each diffusion step.
D: SUNMASK harmonization and infilling of an existing melody (soprano), based on pre-defined mask on the first beat of each measure (0 to 4 of every 16 on the x-axis).
E: SUNMASK harmonization and infilling of an existing melody (soprano), based on pre-defined mask which only preserves the first half of the melody (0 to 64 on the x-axis).
F: SUNMASK harmonization and infilling of an existing melody (soprano), based on pre-defined mask which only preserves the last half of the melody (64 through 128 on the x-axis).

8.13 Full Quantitative Analysis of Bach data

Model	Note	Rhythm	Parallel Errors	Harmonic Quality	S Intervals	A Intervals	T Intervals	B Intervals	Repeated Sequence	**Overall**
Bach GT	0.24 (0.15)	0.23 (0.14)	0.0 (0.69)	0.41 (0.2)	0.47 (0.28)	0.49 (0.22)	0.53 (0.24)	0.69 (0.4)	1.29 (0.88)	4.91 (1.63)
BachMock	0.37 (0.22)	0.26 (0.14)	2.16 (3.22)	0.54 (0.31)	0.53 (0.35)	0.71 (0.34)	0.73 (0.38)	0.89 (0.68)	1.86 (2.81)	8.94 (4.64)
Coconet	**0.44 (0.23)**	1.85 (0.39)	2.61 (6.56)	1.38 (0.39)	**0.70 (0.17)**	**0.86 (0.73)**	**0.86 (0.42)**	**1.02 (0.38)**	6.07 (1.76)	17.00 (6.58)
SD	0.59 (1.82)	0.93 (0.84)	6.42 (4.11)	0.98 (0.67)	1.17 (5.09)	2.65 (4.08)	1.57 (5.68)	2.57 (3.28)	2.45 (2.39)	23.25 (21.45)
SD-A	0.61 (1.80)	0.93 (0.84)	5.85 (4.02)	0.98 (0.66)	1.28 (5.15)	2.69 (4.20)	1.57 (5.61)	3.28 (3.44)	2.30 (2.38)	22.88 (21.87)
SD-T	0.63 (2.40)	0.60 (0.96)	3.82 (4.98)	0.96 (0.64)	1.21 (5.03)	3.40 (4.99)	3.02 (5.02)	2.36 (3.90)	1.52 (3.43)	20.09 (23.88)
SD-AT	0.52 (2.42)	0.60 (0.95)	3.18 (5.10)	0.96 (0.64)	1.24 (5.00)	3.93 (5.03)	2.22 (5.04)	2.00 (3.91)	1.80 (3.39)	18.90 (24.15)
SMc	0.87 (2.05)	0.63 (0.77)	1.38 (6.00)	1.02 (0.49)	1.41 (5.28)	2.02 (4.36)	1.94 (5.72)	2.91 (4.94)	2.32 (2.31)	22.47 (20.80)
SMc-A	1.02 (2.22)	**0.47 (0.77)**	3.92 (3.91)	**0.91 (0.55)**	2.32 (5.23)	3.54 (4.98)	2.74 (5.30)	5.96 (4.59)	2.23 (3.82)	27.82 (18.82)
SMc-T	0.57 (1.79)	0.69 (0.35)	1.28 (3.73)	0.93 (0.49)	0.80 (4.51)	0.99 (4.01)	1.20 (4.68)	1.40 (3.91)	1.81 (0.83)	**13.43 (19.27)**
SMc-AT	0.66 (1.90)	0.55 (0.29)	2.76 (3.63)	0.94 (0.47)	0.91 (4.11)	1.10 (4.00)	1.26 (4.26)	1.45 (4.56)	2.05 (0.96)	16.50 (17.96)
SMc-N	2.07 (2.29)	0.63 (1.33)	7.24 (6.79)	1.06 (0.70)	5.97 (4.77)	7.62 (4.81)	7.37 (4.15)	6.73 (4.92)	1.78 (2.72)	41.67 (21.71)
SMc-AN	2.24 (2.68)	0.69 (0.35)	7.85 (4.28)	1.29 (0.47)	5.08 (4.96)	8.84 (5.41)	8.98 (4.16)	6.96 (4.02)	1.99 (1.27)	47.73 (19.19)
SMc-TN	2.36 (2.94)	0.74 (1.02)	4.42 (6.22)	1.33 (0.70)	6.79 (4.33)	8.19 (4.73)	4.83 (5.06)	6.45 (3.81)	2.36 (2.20)	47.00 (17.32)
SMc-ATN	2.24 (2.36)	0.58 (0.49)	6.82 (4.81)	1.56 (0.54)	6.46 (4.14)	8.51 (4.43)	7.21 (4.28)	7.60 (3.11)	**1.47 (1.02)**	43.85 (18.41)
SMt	3.00 (1.85)	0.74 (0.90)	**0.00 (1.95)**	1.64 (0.70)	8.94 (4.66)	6.49 (4.99)	8.47 (5.58)	7.72 (4.41)	3.10 (2.97)	42.87
SMt-A	3.00 (1.85)	0.74 (0.90)	0.00 (1.95)	1.64 (0.70)	8.94 (4.66)	6.49 (4.99)	8.47 (5.58)	7.72 (4.41)	3.10 (2.97)	42.87
SMt-T	3.74 (2.16)	0.58 (0.56)	0.00 (2.56)	1.73 (0.73)	8.75 (4.62)	6.22 (3.99)	8.05 (4.73)	7.95 (4.49)	2.35 (1.79)	46.21 (17.30)
SMt-AT	3.74 (2.16)	0.58 (0.56)	0.00 (2.56)	1.73 (0.73)	8.75 (4.62)	6.22 (3.99)	8.05 (4.73)	7.95 (4.49)	2.35 (1.79)	46.21 (17.30)

9 Prologue to the Article

Understanding Shared Speech-Text Representations. Gary Wang⋆, Kyle Kastner⋆, Ankur Bapna, Zhehuai Chen, Andrew Rosenberg, Bhuvana Ramabhadran, Yu Zhang. Published in ICASSP 2023 IEEE International Conference on Acoustics, Speech and Signal Processing.

Personal Contribution.
The core drive to deeper understand the learned representations of speech-text models has been a factor for several prior papers, and follow-on work with many of these co-authors after this paper. The overall experimental plan for this particular set of studies was conceived and iterated on by all members of the team, with Gary spearheading the text-encoder studies, while I focused on a qualitative and quantitative understanding of the learned representation spaces present in trained MAESTRO models. The cross-modal retrieval experiments for quantitative probing of latent spaces, and relation to retrieval from unimodal spaces were my design and overall direction. The plot based analysis built on early visualizations of SLAM representation on LibriSpeech by Ankur, extending to various latent layer and dataset combinations to tease out what factors make the learned representations transferrable to new domains. The paper writing, rebuttals, poster, and video slide presentations were developed and edited by all members of the team. ⋆ denotes equal first-author contribution between Gary and myself for the core technical analyses present in the paper. I recorded the video presentation and presented the work as a poster, when this work appeared in the proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), in 2023.

Affiliations

Gary Wang, Google.
Kyle Kastner, Google.
Ankur Bapna, Google.
Zhehuai Chen, Google.
Andrew Rosenberg, Google.
Bhuvana Ramabhadran, Google.
Yu Zhang, Google.

Funding
We acknowledge the support of the following agencies for research funding and computing support: Google.

10 Understanding Shared Speech-Text Representations

10.1 Introduction

The availability of vast amounts of untranscribed speech and text resources has contributed to increased interest in semi-supervised and unsupervised learning approaches for Automated Speech Recognition (ASR). A range of techniques that incorporate text resources at various entry points in an end-to-end ASR model have been explored in the literature. Text injection approaches that either inject text into the encoder (Bapna et al., 2021, 2022; Chen et al., 2022; Sainath et al., 2022) or decoder (Meng et al., 2022) have shown success in low resource applications as well as domain and language transfer (Thomas et al., 2022; Chen et al., 2022; Sainath et al., 2022), while rescoring approaches (Udagawa et al., 2022) use first-pass recognition hypotheses to rescore with a large language model. Maestro (Chen et al., 2022) is an approach to train a speech model using a joint speech-text representation on transcribed speech, untranscribed speech and unspoken text. The joint speech-text representation is learned in two ways. First, using the paired data, activations from two modal encoders, the speech encoder and the text encoder, are aligned and optimized to be similar through a consistency loss term. Second, the speech encoder activations and text encoder activations are passed through a common shared encoder. The output of the shared encoder is the joint speech-text representation that serves as input to the ASR decoder (RNN-T (Graves, 2012; Graves et al., 2013)). These jointly learned representations have yielded state-of-the-art results in not only well-benchmarked, monolingual and multilingual ASR as well as speech translation but have further demonstrated the richness of these learned representations with the ability to build ASR systems for languages with no transcribed speech (Chen et al., 2022). While we know that joint speech-text representation learning improves ASR, our understanding of the value and structure of the learned representations is less well

developed.

In this work, we expand on this understanding in two directions. First, we evaluate the ability to transfer information from one domain to another through the joint representation (Section 10.4). We explore which components of the text encoder are robust across corpora, and which are sensitive. Second, we investigate the modal representations from the speech and text encoders (Section 10.5). We inspect the cross-modal consistency loss as a signal of robustness, and the ability for this loss term to generalize across corpora through T-SNE visualization of activations and a retrieval probe task.

In this analysis, we compare the representations learned by Maestro (Chen et al., 2022) and SLAM (Bapna et al., 2021). The contributions of this work are:

Speech-Free Domain Adaptation (Section 10.4)

- We show that for speech-free domain adaptation with speech-text representations, duration/alignment is the most important aspect to model. Corpus specific text-encoders are not substantially better than using a general purpose encoder with a corpus specific duration model.
- We demonstrate that Maestro enables speech-free domain adaptation with only text data using the corpora in SpeechStew (Chan et al., 2021).

Representation Space Analysis (Section 10.5)

- We show that the Maestro shared encoder learns a unified shared speech-text representation space. However, the modal encoders, even when trained with a consistency loss, learn distinct representations .
- Using a cross-modal retrieval task as a probe, we demonstrate that the shared encoder representations provide better retrieval performance.

10.2 Related Work

The shared speech-text representations we explore in this paper are based on an architecture in which data is passed through modal encoders, whose activations are then consumed by a shared encoder. This structure is used by SLAM (Bapna et al., 2021) and mSLAM (Bapna et al., 2022) where the speech and text encoder outputs are concatenated. Maestro extended this to align the modal activations

and present them independently to the shared encoder. Related approaches either eliminate this alignment step entirely, using a fixed upsampling of text embeddings (Thomas et al., 2022) or use a simpler, deterministic or independent Gaussian duration model (Sainath et al., 2022). In contrast to an independent text encoder, (Sato et al., 2022) performs aligned-text injection using residual adapters in the speech encoder. Alternately, SpeechBert (Chuang et al., 2019) and (Sunder et al., 2022) operate by unifying speech and language model embeddings.

Canonical Correlation Analysis (CCA) between different modalities is a classical technique (Hotelling, 1936) that enables a more direct approach to merging representations. Later CCA was extended to DeepCCA, drawing from the original formulations, but incorporating neural networks e.g (Sun et al., 2020). Acoustic word embeddings take similar inspiration from CCA, learning joint projection spaces e.g. (Hu et al., 2020; Kamper et al., 2016).

10.3 Architecture and Training

Table 10.1 – LibriSpeech Maestro adaptation results.

Method	**Pair Data**	**Text Data**	LibriSpeech		AMI		CV	SWBD	TED
			test-clean	test-other	ihm	sdm1	test	test	test
LS Maestro init	LS	LS LM	2.22	4.27	35.42	60.02	22.48	31.77	7.73
Text Adaptation									
AMI	LS	AMI Sup	1.57	3.09	**30.26**	**54.52**	22.46	29.90	7.10
CV	LS	CV Sup	1.57	3.06	34.21	57.13	21.16	31.19	7.11
SWBD	LS	SWBD Sup	1.51	3.07	31.19	55.40	**20.70**	**28.97**	6.66
TED	LS	TED Sup + LM	**1.45**	**3.05**	32.08	55.36	21.03	29.83	**6.05**

10.3.1 Maestro Architecture

The Maestro architecture (Chen et al., 2022) consists of a 600 million parameter full context Conformer transducer that splits a standard ASR encoder into two sub-encoders, a speech encoder, and a shared encoder and introduces a uni-modal text encoder (See Figure 10.1). The speech encoder ingests log-mel features as input and the shared encoder takes the output of the speech or text modal

encoders. The speech encoder consists of the first 6 conformer layers, and the shared encoder consists of the last 18 conformer layers of the base ASR encoder. We use the text encoder architecture described in (Chen et al., 2022). The text encoder is trained on paired data, since ground-truth text/wordpiece(wpm) and feature mappings are available. It includes a text/phoneme encoder, a trained duration model and re-sampling layer, and a refiner. During inference, the duration model is responsible for predicting the duration of each input text/phoneme token, and the re-sampling layer upsamples the text representations according to their respective durations. The refiner takes the upsampled representations and refines them further for modality matching. Maestro adds an additional RNN-T phoneme decoder of the same parameter size to train the duration model. In contrast to (Chen et al., 2022), unless explicitly mentioned our experiments do not include a second pass fine-tuning.

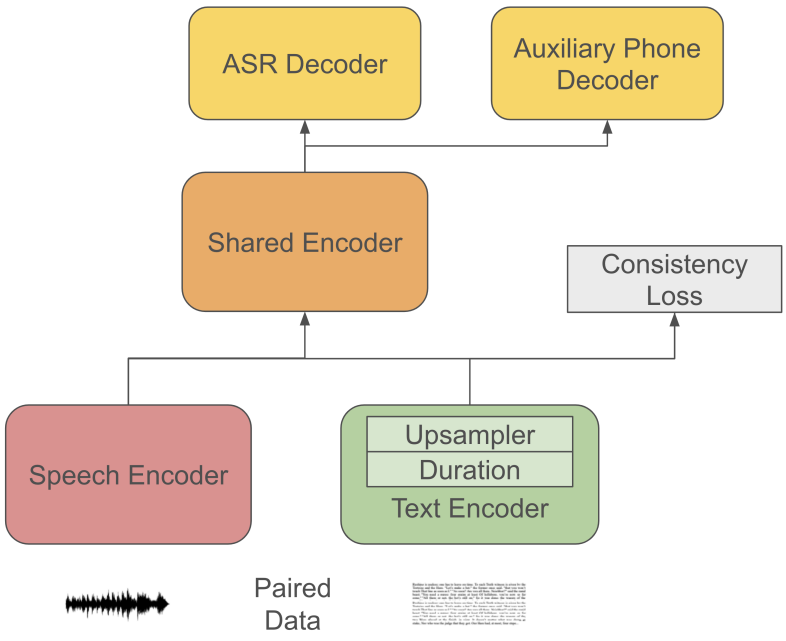

Figure 10.1 – Maestro architecture.

10.3.2 SLAM Architecture

SLAM (Bapna et al., 2021, 2022) uses a similar structure to Maestro, decomposing its encoder into a speech encoder, text encoder and shared encoder. The major distinctions are 1) the structure of the text encoder, and 2) training of

the model. First, the SLAM text encoder consists of a text embedding layer, and sinusoidal position embedding. Second, when training on transcribed data, the outputs of the speech and text encoders are concatenated and used as input to the shared encoder (as opposed to Maestro which consumes *either* the speech or text activations). To align representations and encourage consistency between speech and text, SLAM uses a contrastive Speech-Text Matching loss, and a Translation Language Model loss similar to (Zheng et al., 2021). mSLAM (Bapna et al., 2022) replaces the Speech-Text matching loss with a CTC loss on the speech representations of the transcribed data.

10.4 Text-Only Domain Adaptation

10.4.1 Data

We use subsets of SpeechStew (Chan et al., 2021) comprising of LibriSpeech, AMI, Common Voice (CV), Switchboard (SWBD), and TED corpora for text adaptation experiments. For all experiments, we initialize from a converged LibriSpeech (LS) Maestro model trained on 3 types of data: LibrilLight (Kahn et al., 2020) as untranscribed speech, LibriSpeech (960 hours) as paired data, and LibriSpeech LM text as unspoken text. For text adaptation, we continue to train the LS Maestro model with text from different SpeechStew corpora. During adaptation, we construct a minibatch with 256 text utterances from the target domain, and 256 transcribed speech utterances from LibriSpeech. The use of supervised data during text-only adaptation stabilizes training and minimizes catastrophic forgetting.

10.4.2 Experiments and Results

We present results of adapting with text only data from various corpora. The text encoder is always trained on LibriSpeech data. Table 10.1 shows the results of text-only adaptation across 4 datasets, compared to the LS Maestro baseline. We can observe that, for all datasets, using **in-domain text** successfully enables adaptation to the target domain despite using durations predicted on Librispeech, delivering a relative WER improvement as high as 20% on TED. We also observe

that this speech-free text adaptation benefits other testsets, even though their in-domain text is never observed during training. This observation is the strongest in Switchboard, where Switchboard text adaptation improves performance across AMI, Common Voice and TED testsets with over 7+% relative WER gains.

10.4.3 Text Encoder Data Quality Ablation

Next, we study the impact of domain on text injection performance. The text encoders can be trained on any paired data and the duration model used for upsampling during text injection can be domain specific. As an ablation study on text adaptation, we use the LS Maestro model and corpus specific text encoders trained on the remaining SpeechStew corpora. We study the impact of duration models trained on different domains while using text from the Switchboard (SWBD) corpus alone for adaptation. We initialize a Maestro model from w2v-BERT (Chung et al., 2021) pretrained speech encoder and randomly initialize the text encoder. We subsequently train the text encoder with corpus-specfic paired data for 80k steps to convergence. We use the training recipe outlined in section 10.4.2, where we load and freeze the text encoder during text injection. Table 10.2 presents the performance measured as WER and MSE consistency loss (text encoder loss) of the different text encoders for text adaptation on SWB. We hypothesize that the MSE values provide an indication of quality of the text encoders. Supporting this hypothesis, we observe a trend between the Text Encoder Loss and the SWBD text adaptation performance. It is interesting to note that durations trained on the TED corpus yield the best performance on SWB. This may suggest that TED duration model has higher quality across domains (as indicated by the Text Encoder Loss), or if there is a similarity between SWBD and TED speaking styles.

Table 10.2 – Ablation of text encoder trained on different corpora with Switchboard (SWBD) Text Adaptation, using text encoders trained on different corpora.

Method	**Text Encoder Loss**	SWBD
SWBD Supervised FT (Lower Bound)	-	23.24
LS Maestro init	-	31.77
AMI Duration Training	6.5	29.26
CV Duration Training	5.3	29.18
TED Duration Training	4.1	**28.45**

10.4.4 Text Encoder Component Ablation

We next evaluate the impact of different components of the text-encoder using the AMI corpus for text adaptation. AMI comprises spontaneous elicited meeting speech, while LibriSpeech is read books. We examine the two main components of the text encoder, the duration model, and the refiner. The duration model includes a text/phoneme projection, duration prediction and up-sampling layers. The refiner is responsible for taking the upsampled encoding, and refining it to match the speech encoder output. We start with an entirely in-domain AMI text encoder, and then measure the effect of using the AMI-trained duration model, but keeping the LibriSpeech refiner. Note that since the paired training data is always LibriSpeech, using the LibriSpeech refiner allows us to investigate the value of the duration model without introducing the additional complexity of mismatched data used in training the speech encoder and text encoder refiner. Table 10.3 includes the results.

Table 10.3 – Ablation of importance for in-domain text encoder components with AMI Text adaptation.

Method	AMI	
	ihm	sdm1
AMI Supervised FT (Lower Bound)	11.60	28.03
LS Maestro init	35.42	60.02
LS Duration & Refiner	30.26	54.52
AMI Duration & Refiner	28.78	53.82
AMI Duration, LS Refiner	**27.20**	**52.13**

As we switch to an AMI text encoder entirely, we see an improvement in adaptation performance due to the in-domain text encoder. However, if we use and freeze only the AMI *duration* model, and utilize a LibriSpeech refiner, the performance improves even further. This suggests that the primary value of a domain-specific text-encoder for text-only adaptation comes from having a high quality duration model for the new domain.
It remains to be seen to what degree duration models can be transferred between related corpora. For example, AMI is spontaneous, elicited speech, while LibriSpeech is read. The impact of the duration model may be less pronounced when adapting from LibriSpeech to another read corpus.

10.5 Representation Space Analysis

To analyze representation spaces learned by Maestro and SLAM, we use a combination of visual inspection (via T-SNE (Van der Maaten and Hinton, 2008)) and cosine similarity retrieval probes on mean-pooled speech and text embedding pairs. Although T-SNE is a low-dimension approximation of the overall high-dimension space, we see that the text and speech embeddings clearly appear to capture disparate regions, and different datasets reside in different areas of these regions (Figure 10.2). We see a unification of both modalities after projection through the shared encoder, and the sub-regions occupied by each dataset in those modalities are combined into one shared space. Compared to a

baseline SLAM (Bapna et al., 2021) text and speech encoding (Figure 10.3), the Maestro shared encoding indicates a larger unification of the two disparate information sources. While the included images are drawn from the LibriSpeech (LS) Maestro model, these effects are consistent across models for other domains.

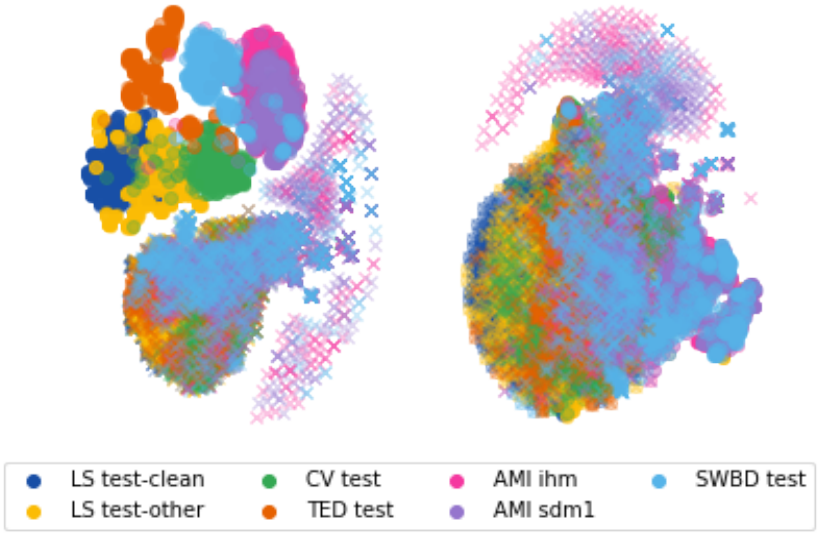

Figure 10.2 – T-SNE of LibriSpeech Maestro text (crosses) and speech (dots) encoder outputs (left), and shared encoder output (right)

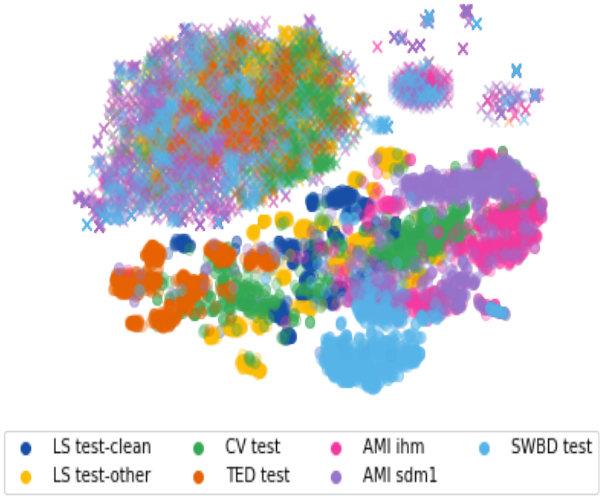

Figure 10.3 – T-SNE of SLAM text (crosses) and speech (dots) shared encoder outputs

The combined joint space after Maestro's shared encoder is an ideal candidate for retrieval tests between speech and text examples to examine the learned representations, as is common in many other settings (Conneau et al., 2022; Baevski et al., 2021; Bapna et al., 2021, 2022; Radford et al., 2021; Hinton and

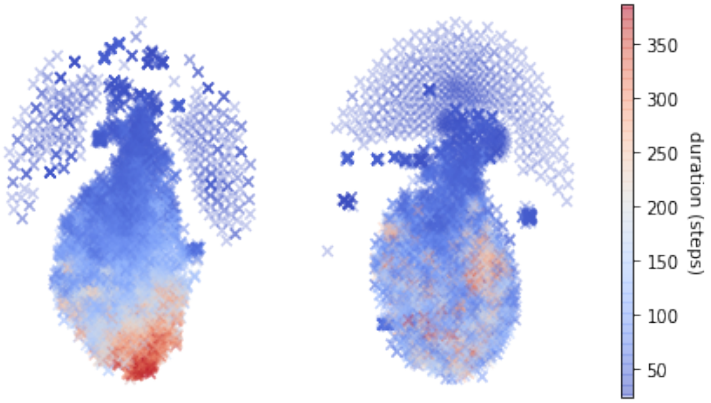

Figure 10.4 – T-SNE of LS Maestro text encoder outputs (left), and shared encoder text output (right), color coded by duration

Salakhutdinov, 2011). We utilize candidate sets of 1,000 paired speech and text examples from each dataset, testing for successful retrieval of the text which matches the speech example. These retrieval results are detailed in Table 10.4, and are used as a secondary inspection mechanism to understand the representation space. We leave larger considerations of retrieval as an explicit speech task for future work, focusing instead on simple exploratory retrieval probes to better understand the shared speech-text representations learned by Maestro and SLAM.

Stronger retrieval performance indicates the effectiveness of the shared encoder to remove unwanted information and unify representations before passing to downstream speech tasks like ASR or Speech Translation. For example, we observe that duration information is strongly represented in the text encoder outputs (Figure 10.4), but following the shared encoder, this influence is more diffuse. We note that retrieval is generally a challenging inspection mechanism; non-unified spaces often have extremely degraded retrieval performance, rendering direct retrieval probes ineffective. For example, cosine retrieval using the baseline SLAM model from Figure 10.3 performs at or below 1% in the same settings as Table 10.4. This makes the Maestro retrieval performance all the more remarkable.

The effectiveness of this approach, along with related findings on multi-language retrieval (Conneau et al., 2022), opens a number of research questions about retrieval using modern multi-modal models, for example using shared spaces to directly sub-select vocabularies for ASR based on top-k retrieval results, or expanding the potential and limits of data augmentation in a shared space. As an

alternative to direct retrieval, ASIF (Norelli et al., 2022) uses the concept of cross-modal anchoring in order to effectively retrieve between unimodal spaces. Using anchored retrieval allows probing the effectiveness of unimodal or non-unified spaces using paired multi-modal data, without further training. Higher ASIF retrieval indicates that two modal views of a data point are substantially related (relative to distractors) even if the respective feature spaces are not strictly unified.

This allows us to test the output of the speech and text encoders directly. 90% of the target set act as anchors, while the remaining portion of the target set are used to evaluate. We repeat this procedure for 100 trials of 100 examples each ($10,000$ evaluations total), randomizing test points and anchors within the target set each trial and reporting the mean accuracy of the final result. Table 10.5 shows the results using 900 anchors and 100 retrievals per trial, with ASIF hyperparameters set to top k 90 (10%) and power 8. We also report the accuracy of direct cosine retrieval under this setting, to highlight where anchoring is useful for non-unified spaces.

Table 10.4 – Shared space cosine retrieval probe accuracy (%)

Method	LibriSpeech		AMI		CV	SWBD	TED
	test-clean	test-other	ihm	sdm1	test	test	test
LS Maestro	83.5	68.8	23.2	12.9	28.8	35.8	70.3
AMI Maestro	89.3	76.7	**47.1**	**31.6**	30.6	53.3	75.1
CV Maestro	**97.5**	**96.5**	32.1	15.8	72.4	48.6	**94.9**
SWBD Maestro	96.1	90.0	31.4	8.3	42.8	**56.8**	79.7
TED Maestro	95.8	94.5	38.2	19.3	**76.0**	53.2	91.7
LS+C4 mSLAM	0.30	0.20	0.00	0.30	0.10	0.00	0.30

Here we find that relative representation retrieval (as in ASIF (Norelli et al., 2022)) reveals the effectiveness of anchoring non-unified spaces particularly seeing improved performance when direct cosine retrieval is degraded (for example SWBD Maestro on TED in Table 10.5). This also reinforces the visualization findings, indicating that the Maestro shared representation yields a substantially more unified representation space. The text and speech encoder representations, while related, remain distinct in many settings. While numbers across Table 10.4 and Table 10.5 cannot be directly compared due to experimental differences, within table variations highlight the differences between direct cosine and ASIF retrieval approaches.

Table 10.5 – Text and speech encoder retrieval probe accuracy (%)

Method	LibriSpeech		AMI		CV	SWBD	TED
	test-clean	test-other	ihm	sdm1	test	test	test
LS Maestro (Direct)	20.5	19.3	7.65	6.16	7.43	13.88	11.89
LS Maestro (ASIF)	45.7	31.2	7.47	5.61	10.2	10.76	16.64
AMI Maestro (Direct)	67.2	48.9	45.2	32.6	19.0	44.7	43.9
AMI Maestro (ASIF)	33.6	17.7	14.9	10.5	7.88	16.7	21.5
CV Maestro (Direct)	76.3	61.7	19.1	10.0	40.0	29.3	44.8
CV Maestro (ASIF)	50.4	34.8	14.3	7.61	20.1	19.4	28.9
SWBD Maestro (Direct)	20.3	14.1	15.9	8.61	10.3	25.3	13.8
SWBD Maestro (ASIF)	49.0	23.0	13.8	7.32	8.94	19.7	29.0
TED Maestro (Direct)	80.6	64.3	24.5	13.3	29.0	40.6	77.9
TED Maestro (ASIF)	43.6	26.9	13.3	7.96	11.8	16.8	25.8
LS+C4 mSLAM (Direct)	1.96	2.0	1.54	1.10	1.5	1.63	1.52
LS+C4 mSLAM (ASIF)	8.63	10.5	3.99	3.06	1.79	6.03	5.78

10.6 Conclusion

Through text-only domain adaptation and inspection of the learned speech and text representations, we draw the following conclusions and hypotheses. Maestro enables effective speech-free domain adaptation across diverse corpora. Our ablation studies demonstrate that duration/alignment are critical elements for successful adaptation. General purpose text encoders with corpus-specific duration models enable adaptation with minimal targeted customization. We hypothesize that the consistency MSE measure, text encoder loss, serves as an effective proxy for the effectiveness of a text-encoder for domain adaptation. Furthermore, we inspect the representations learned by Maestro and SLAM across a variety of standard corpora, finding that modal encoders retain distinct information for each modality: speech and text. However, after shared encoding the Maestro modal representations are unified, and the overall shared space is a powerful, generic representation of speech or text, based on both visual inspection and retrieval probe tasks. We hypothesize that the coherence of this output space in joint speech-text representation learning in Maestro enables its power for text-only adaptation and state-of-the-art performance in both automatic speech recognition (ASR) and speech translation (ST).

11 Conclusion

These systems when trained allow for sampling a new data sequence which is similar to that on which it was trained, representing a particular form of "decision modeling" for generative applications where each sampled datapoint must handle the fundamental uncertainty of other possible data configurations (alternate decisions) in order to create a holistic generation. With addition of conditional variables such as fixed context, or phonetic controls, this also forces the sequential decision model to flexibly account for plausible controls that may be unseen at training. This opens the door to inference time adaptation, human in-the-loop learning for follow-on tuning, and a broad spectrum of real-world applications *if* the core sequential decision system is robust and general.
Throughout this document, a wide variety of tools and techniques have been covered. Particularly, this work has focused on methods for weakly aligned semi-supervised and unsupervised sequence modeling, in order to build strong representations of data with minimal external information. In return many of these methods must make assumptions in data representation, model architecture, or loss construction, in order to effectively incorporate prior knowledge about the data domain into the overall model for effective training.

11.0.1 Overview

In Chapter 4, domain knowledge came in the form of structured swapping in the text input representation. By using the knowledge of word boundaries given for English speech and text, model training, control, and generalization are improved compared to baseline methods. Building on these concepts, as well as ongoing development in text-to-speech synthesis, Chapter 6 further factorized the overall TTS problem using a low resolution time-frequency representation of speech audio. This low-resolution factorization allowed the initial R-MelNet model to characterize important high-level information in the speech signal, leaving fine

detail to the second stage "upsampler" model using WaveRNN. Similar concepts run throughout the preceding works to Chapter 10, and are critical concepts in the original design of Maestro. In particular, the text encoder design hinges on concepts from TTS, using learned durations to greatly improve and align the learned text representations to those learned in the speech encoder. These insights motivated the studies in Chapter 10 probing how the Maestro model works, which components drive overall ASR task performance, and analyzing the learned subspaces of this model for applications such as multi-modal retrieval. Chapter 8 incorporates a view of modeling as iterative prediction, using masked representations to learn piecewise conditional information for a generative system. Leaning further into iterative prediction, SUNMASK features multi-step processing and losses, which require the model to learn to fix its own mistakes for successful training. Factorizing the model in order to enhance iterative prediction builds on prior knowledge and insight, to build a system specifically designed for the inherent biases of iterative prediction, improving model training and sampling. These modeling approaches reflect the times in which they were developed, and though many of the overall systems are no longer cutting edge, the techniques and insights used in their development remain of interest even as underlying modeling techniques change. Recent trends toward non-autoregressive modeling and particularly discrete sequence modeling, highlight the extended relevance of the most recent works in this dissertation. Discrete sequence modeling using non-autoregressive generative models will be an area of active interest and development in the next few years.

11.0.2 Retrospective

Over the course of my graduate research, a staggering amount of growth and change has occurred in the field of deep learning, leading to sea changes in a wide variety of application fields. Deep architectures have become the defacto modeling technique for nearly all learning based systems, and with this adoption has come a huge variety of proposed approaches both specialized and general. Many of the methods described by work mentioned in this dissertation have been muted by the passage of time and steady progress from collective effort on the research front. However the foundational ideas of these methods: structured factorization

of problems, inclusion of secondary information to guide modeling and inference, architectures capturing structure and invariance from domain knowledge, and custom output loss spaces capturing problem information to improve model training, will continue to be evergreen ingredients to building real world systems. Modeling sequences under uncertain conditions has been, and will continue to be, a hallmark problem in machine learning. Recent advances in large language modeling and generative AI build on a rich foundation of past work, including the methods described in 1, while adding new modeling techniques and information sources to build powerful tools for modeling complex dynamics from data. Semi-supervised and generative modeling approaches will continue to find new footholds, and new impact in the years to come.

Bibliography

(1969). Ieee recommended practice for speech quality measurements. *IEEE Transactions on Audio and Electroacoustics 17*(3), 225–246.

Agiomyrgiannakis, Y. (2015). Vocaine the vocoder and applications in speech synthesis. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 4230–4234. IEEE.

Agostinelli, A., T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, et al. (2023). Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*.

Al-Rfou, R., D. Choe, N. Constant, M. Guo, and L. Jones (2019). Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI conference on artificial intelligence*, Volume 33, pp. 3159–3166.

Alagoz, O., H. Hsu, A. J. Schaefer, and M. S. Roberts (2010). Markov decision processes: a tool for sequential decision making under uncertainty. *Medical Decision Making 30*(4), 474–483.

Alain, G. and Y. Bengio (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research 15*(1), 3563–3593.

Allan, M. and C. Williams (2004). Harmonising chorales by probabilistic inference. *Advances in neural information processing systems 17*.

Arik, S., G. Diamos, A. Gibiansky, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou (2017). Deep voice 2: Multi-speaker neural text-to-speech. *arXiv preprint arXiv:1705.08947*.

Arık, S. Ö., M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, et al. (2017). Deep voice: Real-time neural text-to-speech. In *International Conference on Machine Learning*, pp. 195–204.

Auli, M. and J. Gao (2014). Decoder integration and expected bleu training for recurrent neural network language models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14)*, pp. 136–142.

Austin, J., D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg (2021). Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems 34*, 17981–17993.

Bachman, P. and D. Precup (2015). Data generation as sequential decision making. *Advances in Neural Information Processing Systems 28*.

Baevski, A., W.-N. Hsu, A. Conneau, and M. Auli (2021). Unsupervised speech recognition. *Advances in Neural Information Processing Systems 34*, 27826–27839.

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

Bahdanau, D., J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio (2015, August). End-to-end attention-based large vocabulary speech recognition. *ArXiv e-prints abs/1508.04395*.

Bapna, A., C. Cherry, Y. Zhang, Y. Jia, M. Johnson, Y. Cheng, S. Khanuja, J. Riesa, and A. Conneau (2022). mSLAM: Massively multilingual joint pre-training for speech and text. *arXiv preprint arXiv:2202.01374*.

Bapna, A., Y.-a. Chung, N. Wu, A. Gulati, Y. Jia, J. H. Clark, M. Johnson, J. Riesa, A. Conneau, and Y. Zhang (2021). SLAM: A unified encoder for speech and language modeling via speech-text joint pre-training. *arXiv preprint arXiv:2110.10329*.

Battenberg, E., R. Skerry-Ryan, S. Mariooryad, D. Stanton, D. Kao, M. Shannon, and T. Bagby (2020). Location-relative attention mechanisms for robust

long-form speech synthesis. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6194–6198. IEEE.

Belilovsky, E., K. Kastner, G. Varoquaux, and M. Blaschko (2016). Learning to discover probabilistic graphical model structures.

Bellman, R. E. (1957). *Dynamic Programming*. NJ: Princeton University Press.

Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu (2013, May). Advances in optimizing recurrent networks. In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*.

Bengio, Y., N. Léonard, and A. Courville (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009). Curriculum learning. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM.

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

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. London, UK: Oxford University Press.

Bishop, C. M. (1994). Mixture density networks.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Black, A. W., K. Lenzo, and V. Pagel (1998). Issues in building general letter to sound rules.

Black, A. W., H. Zen, and K. Tokuda (2007). Statistical parametric speech synthesis. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, Volume 4, pp. IV–1229. IEEE.

Boulanger-lewandowski, N., Y. Bengio, and P. Vincent (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1159–1166.

Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21.

Bradbury, J., S. Merity, C. Xiong, and R. Socher (2017). Quasi-recurrent neural networks. In *International Conference on Learning Representations*.

Bretan, M., S. Oore, D. Eck, and L. Heck (2017). Learning and evaluating musical features with deep autoencoders. *arXiv preprint arXiv:1706.04486*.

Bretan, M., G. Weinberg, and L. Heck (2016). A unit selection methodology for music generation using deep neural networks. *arXiv preprint arXiv:1612.03789*.

Briot, J.-P. and F. Pachet (2020). Deep learning for music generation: challenges and directions. *Neural Computing and Applications 32*(4), 981–993.

Brock, A., J. Donahue, and K. Simonyan (2018). Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

Caccia, M., L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin (2020). Language gans falling short. In *International Conference on Learning Representations*.

Chan, W. et al. (2021). SpeechStew: Simply mix all available speech recognition data to train one large neural network. *arXiv preprint arXiv:2104.02133*.

Chang, J.-H. R., A. Shrivastava, H. S. Koppula, X. Zhang, and O. Tuzel (2021). Style equalization: Unsupervised learning of controllable generative sequence models. *arXiv preprint arXiv:2110.02891*.

Che, T., Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio (2017). Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

Chen, N., Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi, and W. Chan (2020). Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*.

Chen, N., Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi, N. Dehak, and W. Chan (2021). Wavegrad 2: Iterative refinement for text-to-speech synthesis. *arXiv preprint arXiv:2106.09660*.

Chen, Z., A. Bapna, A. Rosenberg, Y. Zhang, B. Ramabhadran, P. Moreno, and N. Chen (2022). Maestro-u: Leveraging joint speech-text representation learning for zero supervised speech asr. In *IEEE SLT*.

Chen, Z., Y. Zhang, A. Rosenberg, B. Ramabhadran, P. Moreno, A. Bapna, and H. Zen (2022). Maestro: Matched speech text representations through modality matching. *arXiv preprint arXiv:2204.03409*.

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

Cho, K., B. van Merriënboer, D. Bahdanau, and Y. Bengio (2014, October). On the properties of neural machine translation: Encoder–Decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (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 (EMNLP)*, pp. 1724–1734.

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

Chorowski, J., D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio (2015, June). Attention-based models for speech recognition. *ArXiv e-prints abs/1506.07503*.

Chorowski, J. and N. Jaitly (2016). Towards better decoding and language model integration in sequence to sequence models. *arXiv preprint arXiv:1612.02695*.

Chuang, Y.-S., C.-L. Liu, H.-Y. Lee, and L.-s. Lee (2019). Speechbert: An audio-and-text jointly learned language model for end-to-end spoken question answering. *arXiv preprint arXiv:1910.11559*.

Chung, J., S. Ahn, and Y. Bengio (2016). Hierarchical multiscale recurrent neural networks. *CoRR abs/1609.01704*.

Chung, J., Ç. Gülçehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. Technical Report Arxiv report 1412.3555, Université de Montréal. Presented at the Deep Learning workshop at NIPS2014.

Chung, Y.-A. et al. (2021). W2v-BERT: Combining contrastive learning and masked language modeling for self-supervised speech pre-training. *arXiv preprint arXiv:2108.06209*.

Conneau, A., M. Ma, S. Khanuja, Y. Zhang, V. Axelrod, S. Dalmia, J. Riesa, C. Rivera, and A. Bapna (2022). Fleurs: Few-shot learning evaluation of universal representations of speech. *arXiv preprint arXiv:2205.12446*.

Cook, J. D. (2012). Trick for computing log(1+x). https://www.johndcook.com/blog/2012/07/25/trick-for-computing-log1x/. Accessed: 2022-03-20.

Copet, J., F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. Défossez (2023). Simple and controllable music generation. *arXiv preprint arXiv:2306.05284*.

Courbariaux, M. and Y. Bengio (2016). Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR abs/1602.02830*.

Crowson, K., S. Biderman, D. Kornis, D. Stander, E. Hallahan, L. Castricato, and E. Raff (2022). Vqgan-clip: Open domain image generation and editing with natural language guidance. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVII*, pp. 88–105. Springer.

Dai, Z., Z. Yang, Y. Yang, J. G. Carbonell, Q. Le, and R. Salakhutdinov (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 2978–2988.

de Masson d'Autume, C., S. Mohamed, M. Rosca, and J. Rae (2019). Training language gans from scratch. *Advances in Neural Information Processing Systems 32*.

Deng, Y., Y. Kim, J. Chiu, D. Guo, and A. M. Rush (2018). Latent alignment and variational attention. *arXiv preprint arXiv:1807.03756*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186.

Dieleman, S., A. van den Oord, and K. Simonyan. The challenge of realistic music generation: modelling raw audio at scale.

Dinh, L., D. Krueger, and Y. Bengio (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., J. Sohl-Dickstein, and S. Bengio (2017). Density estimation using real nvp. In *International Conference on Learning Representations*.

Domke, J., A. Karapurkar, and Y. Aloimonos (2008). Who killed the directed model? In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE.

Donahue, C., M. Lee, and P. Liang (2020). Enabling language models to fill in the blanks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2492–2501.

Dong, H.-W., K. Chen, S. Dubnov, J. McAuley, and T. Berg-Kirkpatrick (2022). Multitrack music transformer: Learning long-term dependencies in music with diverse instruments. *arXiv preprint arXiv:2207.06983*.

Eddington, C. M. and N. Tokowicz (2015). How meaning similarity influences ambiguous word processing: The current state of the literature. *Psychonomic bulletin & review 22*(1), 13–37.

Elman, J. L. and D. Zipser (1988). Learning the hidden structure of speech. *Journal of the Acoustical Society of America 83*.

Esser, P., R. Rombach, and B. Ommer (2021, June). Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12873–12883.

Evans, W., S. Rajagopalan, and U. Vazirani (1993, July). Choosing a reliable hypothesis. In *Proceedings of the 6th Annual Conference on Computational Learning Theory*, Santa Cruz, CA, USA, pp. 269–276. ACM Press.

Fan, A., M. Lewis, and Y. Dauphin (2018). Hierarchical neural story generation. *abs/1805.04833*.

Fan, B., S. W. Lee, X. Tian, L. Xie, and M. Dong (2015). A waveform representation framework for high-quality statistical parametric speech synthesis. In *Proceedings of APSIPA Annual Summit and Conference*, Volume 16.

Fang, A., A. Liu, P. Seetharaman, and B. Pardo (2020). Bach or mock? a grading function for chorales in the style of js bach. *arXiv preprint arXiv:2006.13329*.

Fang, C. From dynamic time warping (dtw) to hidden markov model (hmm).

Fedus, W., I. Goodfellow, and A. M. Dai (2018). Maskgan: Better text generation via filling in the _. In *International Conference on Learning Representations*.

Fong, J., J. Taylor, and S. King (2020). Testing the limits of representation mixing for pronunciation correction in end-to-end speech synthesis. In *INTERSPEECH*, pp. 4019–4023.

Freitag, M. and Y. Al-Onaizan (2017). Beam search strategies for neural machine translation. *CoRR abs/1702.01806*.

Ganin, Y., T. Kulkarni, I. Babuschkin, S. A. Eslami, and O. Vinyals (2018). Synthesizing programs for images using reinforced adversarial learning. In *International Conference on Machine Learning*, pp. 1666–1675. PMLR.

Gatys, L. A., A. S. Ecker, and M. Bethge (2016). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423.

Germain, M., K. Gregor, I. Murray, and H. Larochelle (2015). Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*.

Gers, F. (2001). Long short-term memory in recurrent neural networks.

Goodfellow, I., Y. Bengio, and A. Courville (2016). Deep learning. Book in preparation for MIT Press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

Graves, A. (2013a). Generating sequences with recurrent neural networks. Technical report, arXiv:1308.0850.

Graves, A. (2013b). Generating sequences with recurrent neural networks. *CoRR abs/1308.0850*.

Graves, A. (2013, August). Generating sequences with recurrent neural networks. *arXiv:*`1308.0850 [cs.NE]`.

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

Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recurrent neural networks. In *ICML'2014*.

Graves, A., J. Menick, and A. van den Oord (2018). Associative compression networks for representation learning. *CoRR abs/1804.02476*.

Graves, A., A.-R. Mohamed, and G. Hinton (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649. IEEE.

Graves, A. and J. Schmidhuber (2008). Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in neural information processing systems 21*.

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber (2015). LSTM: a search space odyssey. *arXiv preprint arXiv:1503.04069*.

Gregor, K., I. Danihelka, A. Graves, and D. Wierstra (2015a). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.

Gregor, K., I. Danihelka, A. Graves, and D. Wierstra (2015b). Draw: A recurrent neural network for image generation. In *Proceedings of The 32nd International Conference on Machine Learning (ICML)*.

Griffin, D. and J. Lim (1984). Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing 32*(2), 236–243.

Gu, J., J. Bradbury, C. Xiong, V. O. Li, and R. Socher (2018). Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

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

Gu, S., D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo (2022). Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10696–10706.

Guo, J., L. Xu, and E. Chen (2020). Jointly masked sequence-to-sequence model for non-autoregressive neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 376–385.

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

Ha, D., A. Dai, and Q. V. Le (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Ha, D. and D. Eck (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

Hadjeres, G. and L. Crestel (2020). Vector quantized contrastive predictive coding for template-based music generation. *arXiv preprint arXiv:2004.10120*.

Hadjeres, G., F. Pachet, and F. Nielsen (2016). Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*.

He, K., X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009.

He, K., X. Zhang, S. Ren, and J. Sun (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*.

He, K., X. Zhang, S. Ren, and J. Sun (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1367–1377.

Hinton, G. and R. Salakhutdinov (2011). Discovering binary codes for documents by learning deep generative models. *Topics in Cognitive Science 3*(1), 74–91.

Hinton, G. E. and R. S. Zemel (1994). Autoencoders, minimum description length, and Helmholtz free energy. In D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6 (NIPS'93)*, pp. 3–10. Morgan Kaufmann Publishers, Inc.

Ho, J., A. Jain, and P. Abbeel (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems 33*, 6840–6851.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*(02), 107–116.

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

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

Hoffer, E., T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry (2019). Augment your batch: better training with larger batches. *arXiv preprint arXiv:1901.09335*.

Hoogeboom, E., A. A. Gritsenko, J. Bastings, B. Poole, R. van den Berg, and T. Salimans (2022). Autoregressive diffusion models. In *International Conference on Learning Representations*.

Hoogeboom, E., D. Nielsen, P. Jaini, P. Forré, and M. Welling (2021). Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems 34*.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika 28(3/4)*, 321–377.

Hu, Y., S. Settle, and K. Livescu (2020). Multilingual jointly trained acoustic and written word embeddings. In *Interspeech*.

Huang, A., S. Chen, M. Nelson, and D. Eck (2018). Towards mixed-initiative generation of multi-channel sequential structure.

Huang, C.-W., J. H. Lim, and A. C. Courville (2021). A variational perspective on diffusion-based generative models and score matching. *Advances in Neural Information Processing Systems 34*.

Huang, C.-Z. A., T. Cooijmans, A. Roberts, A. Courville, and D. Eck (2017a). Counterpoint by convolution. In *Proceedings of ISMIR 2017*.

Huang, C.-Z. A., T. Cooijmans, A. Roberts, A. C. Courville, and D. Eck (2017b, October). Counterpoint by convolution. In *Proceedings of the 18th International Society for Music Information Retrieval Conference*, Suzhou, China, pp. 211–218. ISMIR.

Huang, C.-Z. A., C. Hawthorne, A. Roberts, M. Dinculescu, J. Wexler, L. Hong, and J. Howcroft. The bach doodle: Approachable music composition with machine learning at scale.

Huang, C.-Z. A., A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck (2018). Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations*.

Huang, W.-C., E. Cooper, Y. Tsao, H.-M. Wang, T. Toda, and J. Yamagishi (2022). The voicemos challenge 2022. *arXiv preprint arXiv:2203.11389*.

Hunt, A. J. and A. W. Black (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, Volume 1, pp. 373–376. IEEE.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Ito, K. (2017). The lj speech dataset. https://keithito.com/LJ-Speech-Dataset/.

Jang, E., S. Gu, and B. Poole (2017). Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations 2017*. OpenReviews. net.

Jaques, N., J. Engel, D. Ha, F. Bertsch, R. Picard, and D. Eck (2018). Learning via social awareness: improving sketch representations with facial feedback. *arXiv preprint arXiv:1802.04877*.

Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical evaluation of recurrent network architectures. In *ICML'2015*.

Kahn, J., M. Rivière, W. Zheng, et al. (2020). Libri-Light: A benchmark for asr with limited or no supervision. In *Proc. ICASSP*, pp. 7669–7673.

Kaiser, L., A. Roy, A. Vaswani, N. Parmar, S. Bengio, J. Uszkoreit, and N. Shazeer (2018). Fast decoding in sequence models using discrete latent variables. *CoRR abs/1803.03382*.

Kalchbrenner, N., I. Danihelka, and A. Graves (2015). Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Kalchbrenner, N., E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. v. d. Oord, S. Dieleman, and K. Kavukcuoglu (2018). Efficient neural audio synthesis. *arXiv preprint arXiv:1802.08435*.

Kamper, H., W. Wang, and K. Livescu (2016). Deep convolutional acoustic word embeddings using word-pair side information. In *ICASSP*.

Kastner, K., J. F. Santos, Y. Bengio, and A. Courville (2019). Representation mixing for tts synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5906–5910. IEEE.

Kawahara, H., Y. Agiomyrgiannakis, and H. Zen (2016). Yet another geralized vocoder.

Kawahara, H., M. Morise, T. Takahashi, R. Nisimura, T. Irino, and H. Banno (2008, March). Tandem-straight: A temporally stable power spectral representation for periodic signals and applications to interference-free spectrum, f0, and aperiodicity estimation. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3933–3936.

Kingma, D. P. and J. Ba (2014, December). Adam: A Method for Stochastic Optimization. *arXiv:*`1412.6980 [cs.LG]`.

Kingma, D. P., T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems 29*.

Kingma, D. P., T. Salimans, B. Poole, and J. Ho (2021). Variational diffusion models. In *Advances in Neural Information Processing Systems*.

Kingma, D. P. and M. Welling (2013, December). Auto-Encoding Variational Bayes. *ArXiv e-prints*.

Kingma, D. P. and M. Welling (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. and M. Welling (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kingma, D. P., M. Welling, et al. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning 12*(4), 307–392.

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kong, J., J. Kim, and J. Bae (2020). Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems 33*, 17022–17033.

Kreuk, F., G. Synnaeve, A. Polyak, U. Singer, A. Défossez, J. Copet, D. Parikh, Y. Taigman, and Y. Adi (2022). Audiogen: Textually guided audio generation. In *The Eleventh International Conference on Learning Representations*.

Lam, M. W., J. Wang, D. Su, and D. Yu (2022). Bddm: Bilateral denoising diffusion models for fast and high-quality speech synthesis. In *International Conference on Learning Representations*.

Lang, K. J. and G. E. Hinton (1988). The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University.

Larochelle, H. and I. Murray (2011). The Neural Autoregressive Distribution Estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'2011)*, Volume 15 of JMLR: W&CP.

Lee, J., E. Mansimov, and K. Cho (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *EMNLP*.

Li, J., W. Monroe, and D. Jurafsky (2016). A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*.

Liang, F. (2016). Bachbot: Automatic composition in the style of bach chorales. *University of Cambridge 8*, 19–48.

Lin, K., D. Li, X. He, Z. Zhang, and M.-T. Sun (2017). Adversarial ranking for language generation. *Advances in neural information processing systems 30*.

Liu, A., A. Fang, G. Hadjeres, P. Seetharaman, and B. Pardo (2020). Incorporating music knowledge in continual dataset augmentation for music generation. *arXiv preprint arXiv:2006.13331*.

Lu, S., Y. Zhu, W. Zhang, J. Wang, and Y. Yu (2018). Neural text generation: Past, present and beyond. *arXiv preprint arXiv:1803.07133*.

Maddison, C. J., A. Mnih, and Y. W. Teh (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Mairal, J., F. Bach, J. Ponce, G. Sapiro, and A. Zisserman (2009). Supervised dictionary learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 1033–1040. NIPS Foundation.

Makhzani, A. and B. Frey (2013). K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

Martins, A. and R. Astudillo (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pp. 1614–1623.

McCarthy, O. (2018). WaveRNN in PyTorch. https://github.com/fatchord/WaveRNN. Accessed: 2022-03-20.

Mehri, S., K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio (2016). Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*.

Mehta, S., É. Székely, J. Beskow, and G. E. Henter (2021). Neural hmms are all you need (for high-quality attention-free tts). *arXiv preprint arXiv:2108.13320*.

Meister, C., T. Pimentel, G. Wiher, and R. Cotterell (2022). Typical decoding for natural language generation. *arXiv preprint arXiv:2202.00666*.

Meng, C., Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon (2021). Sdedit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*.

Meng, Z., T. Chen, R. Prabhavalkar, Y. Zhang, G. Wang, K. Audhkhasi, J. Emond, T. Strohman, B. Ramabhadran, W. R. Huang, E. Variani, Y. Huang, and P. J. Moreno (2022). Modular hybrid autoregressive transducer. In *to appear in IEEE SLT*.

Menick, J. and N. Kalchbrenner (2018). Generating high fidelity images with subscale pixel networks and multidimensional upscaling. In *International Conference on Learning Representations*.

Mensch, A. and M. Blondel (2018). Differentiable dynamic programming for structured prediction and attention. In *35th International Conference on Machine Learning*.

Min, L., J. Jiang, G. Xia, and J. Zhao (2023). Polyffusion: A diffusion model for polyphonic score generation with internal and external controls. *arXiv preprint arXiv:2307.10304*.

Mittal, G., J. Engel, C. Hawthorne, and I. Simon (2021). Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*.

Morise, M., F. Yokomori, and K. Ozawa (2016). World: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems 99*(7), 1877–1884.

Morrison, M., R. Kumar, K. Kumar, P. Seetharaman, A. Courville, and Y. Bengio (2021). Chunked autoregressive gan for conditional waveform synthesis. *arXiv preprint arXiv:2110.10139*.

Murphy, K. P. (2012). *Machine Learning: a Probabilistic Perspective*. Cambridge, MA, USA: MIT Press.

Nachmani, E., A. Polyak, Y. Taigman, and L. Wolf (2018). Fitting new speakers based on a short untranscribed sample. *arXiv preprint arXiv:1802.06984*.

Nichol, A. Q., P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen (2022). Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *International Conference on Machine Learning*, pp. 16784–16804. PMLR.

Niculae, V. and M. Blondel (2017). A regularized framework for sparse and structured neural attention. In *Advances in Neural Information Processing Systems*, pp. 3338–3348.

Niculae, V., A. F. Martins, M. Blondel, and C. Cardie (2018). Sparsemap: Differentiable sparse structured inference. *arXiv preprint arXiv:1802.04223*.

Norelli, A., M. Fumero, V. Maiorca, L. Moschella, E. Rodolà, and F. Locatello (2022). Asif: Coupled data turns unimodal models to multimodal without training. *arXiv preprint arXiv:2210.01738*.

Ochshorn, R. M. and M. Hawkins (2017). Gentle forced aligner [computer software]. https://github.com/lowerquality/gentle.

Olshausen, B. A. and D. J. Field (1997, December). Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research 37*, 3311–3325.

Pachet, F., P. Roy, and G. Barbieri (2011, July). Finite-length markov processes with constraints. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, Barcelona, Spain, pp. 635–642.

Papadopoulos, A., P. Roy, and F. Pachet (2014). Avoiding plagiarism in markov sequence generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 28.

Papamakarios, G., T. Pavlakou, and I. Murray (2017). Masked autoregressive flow for density estimation. *Advances in neural information processing systems 30*.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32*.

Pati, A., A. Lerch, and G. Hadjeres (2019). Learning to traverse latent spaces for musical score inpainting. In A. Flexer, G. Peeters, J. Urbano, and A. Volk (Eds.), *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*, pp. 343–351.

Payne, C. (2019). Musenet. *openai.com/blog/musenet*.

Peng, H., S. Thomson, and N. A. Smith (2018). Backpropagating through structured argmax using a spigot. *arXiv preprint arXiv:1805.04658*.

Ping, W., K. Peng, and J. Chen (2018, July). ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech. *ArXiv e-prints*.

Ping, W., K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller (2017). Deep voice 3: 2000-speaker neural text-to-speech. *arXiv preprint arXiv:1710.07654*.

Ping, W., K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller (2018). Deep voice 3: Scaling text-to-speech with convolutional sequence learning. In *International Conference on Learning Representations*.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*(2), 257–286.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR.

Radford, A., K. Narasimhan, T. Salimans, and I. Sutskever (2018). Improving language understanding by generative pre-training.

Radford, A. and J. Wu (2019). Rewon child, david luan, dario amodei, and ilya sutskever. 2019. *Language models are unsupervised multitask learners. OpenAI Blog 1*(8), 9.

Ramachandran, P., T. L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M. A. Hasegawa-Johnson, R. H. Campbell, and T. S. Huang (2017). Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*.

Ramesh, A., P. Dhariwal, A. Nichol, C. Chu, and M. Chen (2022). Hierarchical text-conditional image generation with clip latents.

Rao, K., F. Peng, H. Sak, and F. Beaufays (2015). Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *ICASSP*.

Reddy, R. D. (1977). Speech understanding systems: A summary of results of the five-year research effort.

Ren, Y., J. Liu, and Z. Zhao (2021). Portaspeech: Portable and high-quality generative text-to-speech. *Advances in Neural Information Processing Systems 34*.

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

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

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011, June). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of theTwenty-eight International Conference on Machine Learning (ICML'11)*.

Roberts, A., J. Engel, C. Raffel, C. Hawthorne, and D. Eck (2018). A hierarchical latent vector model for learning long-term structure in music. *CoRR abs/1803.05428*.

Rombach, R., A. Blattmann, D. Lorenz, P. Esser, and B. Ommer (2021). High-resolution image synthesis with latent diffusion models.

Roy, A., A. Vaswani, A. Neelakantan, and N. Parmar (2018). Theory and experiments on vector quantized autoencoders. *CoRR abs/1805.11063*.

Saharia, C., J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi (2021). Image super-resolution via iterative refinement. *arXiv preprint arXiv:2104.07636*.

Sainath, T. N., R. Prabhavalkar, A. Bapna, Y. Zhang, Z. Huo, Z. Chen, B. Li, W. Wang, and T. Strohman (2022). Joist: A joint speech and text streaming model for asr. In *IEEE SLT*.

Salimans, T., A. Karpathy, X. Chen, and D. P. Kingma (2016). Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications.

Sato, H., T. Komori, T. Mishima, Y. Kawai, T. Mochizuki, S. Sato, and T. Ogawa (2022). Text-only domain adaptation based on intermediate ctc. In *Interspeech*.

Savinov, N., J. Chung, M. Binkowski, E. Elsen, and A. van den Oord (2022). Step-unrolled denoising autoencoders for text generation. In *International Conference on Learning Representations*.

Schmidhuber, J. (2008). Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on Anticipatory Behavior in Adaptive Learning Systems*, pp. 48–76. Springer.

Schoenberg, A. and L. Stein (1969). *Structural functions of harmony*. Number 478. WW Norton & Company.

Schulman, J., N. Heess, T. Weber, and P. Abbeel (2015). Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536.

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

See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Volume 1, pp. 1073–1083.

Semeniuta, S., A. Severyn, and E. Barth (2016). Recurrent dropout without memory loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 1757–1766.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations*.

Shen, J., R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, et al. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4779–4783. IEEE.

Shen, J., R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, et al. (2017). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *arXiv preprint arXiv:1712.05884*.

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

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis (2017, October). Mastering the game of go without human knowledge. *Nature 550*, 354–.

Skerry-Ryan, R., E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. Weiss, R. Clark, and R. A. Saurous (2018). Towards end-to-end prosody

transfer for expressive speech synthesis with tacotron. In *international conference on machine learning*, pp. 4693–4702. PMLR.

Slaney, M., D. Naar, and R. Lyon (1994). Auditory model inversion for sound separation. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, Volume 2, pp. II–77. IEEE.

Smith, E. M., M. H. M. Kambadur, E. Presani, and A. Williams (2022). " i'm sorry to hear that": finding bias in language models with a holistic descriptor dataset. *arXiv preprint arXiv:2205.09209*.

Smith, J. O. (accessed 2019). *Spectral Audio Signal Processing*. http://ccrma.stanford.edu/~jos/sasp/. online book, 2011 edition.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR.

Sønderby, C. K., B. Poole, and A. Mnih. Continuous relaxation training of discrete latent variable image models.

Song, Y., C. Durkan, I. Murray, and S. Ermon (2021). Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems 34*.

Song, Y. and S. Ermon (2019). Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems 32*.

Sotelo, J., S. Mehri, K. Kumar, J. F. Santos, K. Kastner, A. Courville, and Y. Bengio (2017). Char2wav: End-to-end speech synthesis.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*(1), 1929–1958.

Stanton, D., M. Shannon, S. Mariooryad, R. Skerry-Ryan, E. Battenberg, T. Bagby, and D. Kao (2021). Speaker generation. *arXiv preprint arXiv:2111.05095*.

Strayer, H. R. (2013). From neumes to notes: The evolution of music notation. *Musical Offerings 4*(1), 1.

Sturm, B., J. F. Santos, and I. Korshunova (2015). Folk music style modelling by recurrent neural networks with long short term memory units. In *16th International Society for Music Information Retrieval Conference, late-breaking demo session*, pp. 2.

Sturm, B. L., O. Ben-Tal, Ú. Monaghan, N. Collins, D. Herremans, E. Chew, G. Hadjeres, E. Deruty, and F. Pachet (2019). Machine learning research that matters for music creation: A case study. *Journal of New Music Research 48*(1), 36–55.

Sun, Z., P. K. Sarma, W. A. Sethares, and Y. Liang (2020). Learning relationships between text, audio, and video via deep canonical correlation for multimodal language analysis. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8992–8999. AAAI Press.

Sunder, V., E. Fosler-Lussier, S. Thomas, H.-K. J. Kuo, and B. Kingsbury (2022). Tokenwise contrastive pretraining for finer speech-to-bert alignment in end-to-end speech-to-intent systems. In *Interspeech*.

Sutskever, I., J. Martens, G. E. Dahl, and G. E. Hinton (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1139–1147.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112.

Tachibana, H., K. Uenoyama, and S. Aihara (2017). Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. *arXiv preprint arXiv:1710.08969*.

Tachibana, H., K. Uenoyama, and S. Aihara (2018). Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. In *ICASSP*.

Taigman, Y., L. Wolf, A. Polyak, and E. Nachmani (2017). Voice synthesis for in-the-wild speakers via a phonological loop. *arXiv preprint arXiv:1707.06588*.

Theis, L. and M. Bethge (2015). Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pp. 1927–1935.

Theis, L., A. van den Oord, and M. Bethge (2015, November). A note on the evaluation of generative models. *ArXiv e-prints*.

Thomas, S., B. Kingsbury, G. Saon, and H.-K. J. Kuo (2022). Integrating text inputs for training and adapting rnn transducer asr models. In *IEEE ICASSP*.

Thomé, C. (2018). Personal communication.

Turian, J. and M. Henry (2020). I'm sorry for your loss: Spectrally-based audio distances are bad at pitch. *arXiv preprint arXiv:2012.04572*.

Udagawa, T., M. Suzuki, G. Kurata, N. Itoh, and G. Saon (2022). Effect and analysis of large-scale language model rescoring on competitive asr systems. In *Interspeech*.

Uria, B., M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research 17*(1), 7184–7220.

Uria, B., I. Murray, and H. Larochelle (2014). A deep and tractable density estimator. In *International Conference on Machine Learning*, pp. 467–475. PMLR.

van den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016, September). WaveNet: A Generative Model for Raw Audio. *ArXiv e-prints*.

van den Oord, A., N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves (2016). Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems 29*.

van den Oord, A., N. Kalchbrenner, and K. Kavukcuoglu (2016, January). Pixel Recurrent Neural Networks. *ArXiv e-prints*.

van den Oord, A., O. Vinyals, and k. kavukcuoglu (2017). Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30*, pp. 6306–6315.

Van der Maaten, L. and G. Hinton (2008). Visualizing data using t-sne. *Journal of machine learning research 9*(11).

Van Oord, A., N. Kalchbrenner, and K. Kavukcuoglu (2016). Pixel recurrent neural networks. In *International conference on machine learning*, pp. 1747–1756. PMLR.

Vapnik, V. (1991). Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pp. 831–838.

Vasquez, S. and M. Lewis (2019). Melnet: A generative model for audio in the frequency domain. *arXiv preprint arXiv:1906.01083*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Vijayakumar, A. K., M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra (2016). Diverse beam search: Decoding diverse solutions from neural sequence models.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, New York, NY, USA, pp. 1096–1103. ACM.

Vinyals, O., L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton (2014). Grammar as a foreign language. Technical report, arXiv:1412.7449.

Visin, F., K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio (2015). Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 260–269.

Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition. *Neural Computation 1*, 39–46.

Wang, C., Y. Tang, X. Ma, A. Wu, D. Okhonko, and J. Pino (2020). fairseq s2t: Fast speech-to-text modeling with fairseq. *arXiv preprint arXiv:2010.05171*.

Wang, Y., R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, et al. (2017). Tacotron: A fully end-to-end text-to-speech synthesis model. *arXiv preprint*.

Wiseman, S. and A. M. Rush. Sequence-to-sequence learning as beam-search optimization.

Wu, F., A. Fan, A. Baevski, Y. Dauphin, and M. Auli (2019). Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Wu, Y. et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Wu, Z., O. Watts, and S. King (2016). Merlin: An open source neural network speech synthesis system. *Proc. SSW, Sunnyvale, USA*.

Yamamoto, R. (2018). DeepVoice 3 in PyTorch. https://github.com/r9y9/deepvoice3{_}pytorch. Accessed: 2022-03-20.

Yamamoto, R., M. Andrews, M. Petrochuk, W. Hy, cbrom, O. Vishnepolski, M. Cooper, K. Chen, and A. Pielikis (2018, October). r9y9/wavenet_vocoder: v0.1.1 release. https://github.com/r9y9/wavenet_vocoder.

Yang, Z., Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems 32*.

Yu, F. and V. Koltun (2015). Multi-scale context aggregation by dilated convolutions. *CoRR abs/1511.07122*.

Yu, L., W. Zhang, J. Wang, and Y. Yu (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, Volume 31.

Zen, H., A. Senior, and M. Schuster (2013, May). Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7962–7966.

Zhang, S., Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio (2016). Architectural complexity measures of recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1822–1830.

Zheng, R. et al. (2021). Fused acoustic and text encoding for multimodal bilingual pretraining and speech translation. *arXiv preprint arXiv:2102.05766*.

Zhu, Y., S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu (2018). Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 1097–1100.